1025: MAGIC 2010

Multi Autonomous Ground International Challenge

Volume I: Final Report
22nd October 2010

Supervisors
Associate Professor Ben Cazzolato
Dr Steven Grainger
Dr Chris Madden

Students
Mark Baulis (a1147334)
Adam Cundy (a1162416)
Nigel Gaskin (a1160814)
Peter Hardy (a1149159)
Phuong Huynh (a1160609)
Sundar Komandurelaiyavalli (a1154422)
Konrad Pilch (a1160221)
Anton Skeletlee (a1148726)
Benjamin Quast (a1162326)
Stella Wong (a1159937)
EXECUTIVE SUMMARY

The Multi-Autonomous Ground-robotic International Challenge (MAGIC) was a competition that required a team of autonomous mobile robots to collaboratively perform a reconnaissance mission in a dynamic urban environment, generating a complete map of the previously unknown area and identifying objects of interest (such as potential threats) within the area. The challenge consisted of three stages;

Stage 1: The submission of a technical proposal or tender, to bid for a spot in the challenge through a proposed design (October 2009)

Stage 2: A practical demonstration to competition judges of the progress of the proposed design, along with a technical submission of work done (June 2010)

Stage 3: Grand Final with USD 750,000 first prize winner (November 2010)

Due to the significant amount of the prize money for first, second and third places and the fact that it was an international challenge, competition came from around the globe with well-funded universities and robotic/automation companies. A consortium of partners led by Strategic Engineering, a consultancy firm specialising in robotics and automation, in partnership with University of Adelaide was shortlisted after the first stage with nine other international teams and two self-funded teams. Students from the University of Adelaide were given the opportunity to work with Strategic Engineering (starting from March 2010) to design and manufacture the proposed autonomous robots.

The MAGIC competition provided quite explicit requirements for successful completion of the target challenge and hence dictated heavily the hardware and software requirements of the system. The system was broken down into several components. These were:

(i) The ability to perform accurate localisation both indoors and outside
(ii) Generation of mapped representations of the robots environment
(iii) Autonomous single and multi-vehicle control
(iv) The detection, identification and locating of specified objects of interest.

**Localisation:** The ability to map the robots environment accurately was an underlying requirement for the MAGIC competition as it forms the basis of the mapping tasks. To produce the maps accurately and to effectively navigate, the robots needed to be able to determine their position as precisely as practical. Subsequently, it was decided that an accuracy of within 200mm in comparing the virtual maps and the real world environment was desirable and would enable successful completion of all MAGIC requirements. Different algorithms and methods were required to achieve both indoor and outdoor localisation. The significance of considering both scenarios is the lack of GPS signal indoors, while GPS systems can be utilised in most outdoor environments. Hence a Kalman filter was utilised to combine differential Global Positioning System (dGPS) data and an Inertial Measurement Units (IMU) data for outdoor environments, while simultaneous location and mapping (SLAM) was used for indoor environments, which utilised the IMU and a Light Detection and Ranging (LiDAR) unit.

**Mapping:** Mapping was placed into two categories. Physical mapping provides a physical representation of the UGV surroundings in the form of a three-dimensional occupancy grid. The second
category was that of conceptual maps. These were designed to stem from the physical maps and use the information gathered to interpret the surroundings and enable the robots to have some form of world perception. These primarily came in the form of features discovered in the competition area, heights of objects detected and an estimation of areas with the greatest outlook or vantage within the local vehicle surroundings. All of this information is fed both into a Human Machine Interface (HMI), where human operators are able to monitor and direct UGV movements if necessary, and into an autonomous control system, providing crucial information about the surrounding environment for automatic UGV navigation decisions to be made.

Control: The robots were required to be controlled autonomously both on an individual basis and in a team. The development of this area of the project initially focused on tele-operation of the UGVs and then progressed to a fuzzy logic semi-autonomous path tracking algorithm as well as significant research into waypoint generation for fully autonomous control. Due to the Adelaide University being in possession of only one fully working UGV for the majority of the year, team automation was not prominent in the development focus.

Camera Vision: A further design requirement included the detection, identification and locating of three competition-defined objects of interest that were either mobile or stationary. The camera vision focused primarily on colour to detect and identify objects of interest as the MAGIC competition organisers provided objects of interest that were distinguished by bright red or navy blue colours.

Unmanned Ground Vehicle (UGV): A significant number of highly specialised goals were required by the MAGIC competition and hence it was decided that a customised platform solution was required to successfully apply all of these goals, hence a significant amount of time and money was spent on this aspect. A skid steer, six wheeled vehicle was designed with the goal of traversing a variety of terrain, exhibit good reliability and be highly manoeuvrable. The UGV developed is capable of traveling up to 10km/h over terrains such as grass, gravel, concrete and indoor surfaces. It is capable of running for more than three hours continuously and can skid steer 180 degrees in a 1-2m turning circle even on high friction surfaces.

Ground Control Station (GCS): The ability to remotely monitor and take over control if necessary was highly desirable and hence a remote base station was designed to display relevant data and provide a human machine interface (HMI).

This report focuses mainly on the work done on this project by the student team at the University of Adelaide, however some overlap occurs between work conducted by Strategic Engineering. The report includes hardware selection, mechanical design and the creation of software required to interact with the sensors for each subsystem. Most of the systems have been extensively developed and tested with varying levels of success. All of the systems have been developed from the ground up and have been discussed in the report. However, the opportunity for further development is ample and has been allowed for in documenting the progress to date.

A practical demonstration to the competition judges was conducted at Strategic Engineering (in Sydney where they are based) in June 2010, and unfortunately the team was unsuccessful. Following this demonstration, the project was independently continued by the student team at the University of Adelaide as their honours project. Since then this project has been successful in creating a system that would be capable of performing in the MAGIC competition but also opens up the possibilities for further robotics research at the University of Adelaide.
DISCLAIMER

The content of this report is entirely the work of the following students from the University of Adelaide with the support of Strategic Engineering. Any content obtained from other sources has been referenced accordingly.

Mark Baulis  Adam Cundy
Nigel Gaskin  Phuong Huynh
Sundar Komandurelaiyavalli  Konrad Pilch
Anton Skeketee  Benjamin Quast
Stella Wong

October 2010
There were numerous people instrumental to the project whom we would like to thank:

- Supervisors Ben Cazzolato, Steven Grainger and Chris Madden for their support and guidance in this project
- Brad Alexander and Wynand Marias for help with software development
- The whole team at Strategic Engineering and particularly Richard Aplin for his patience with the students
- Paul Arthur for enlisting sponsors for the project

We would like to add a special thanks for financial support to:

- Professor Peter Dowd, the Executive Dean of the Faculty of Engineering, Computing and Mathematical Sciences (ECMS)
- James McWha, Vice Chancellor
- Fred McDougall, Deputy Vice Chancellor

We would also like to mention a warm acknowledgment to all our sponsors who offered extensive assistance in addition to the hardware/software that was provided. Brett Motum from Maxon Motors went beyond the duty of call, providing selection suggestions and troubleshooting. Tony Barnett from Leuze Australia was particularly helpful with arranging a discounted price and guiding the team in the operation of Leuze hardware. Novatel have been generous in offering their high performance GPS units at a discounted rate. Thanks to Graeme Hooper from GPSatSystems for his support in setting up the Novatel units to suit our requirements. On the software side, Andy Rhind from Symmetry Innovations for the generous training, support and software licenses he provided.

A number of other sponsors were kind enough to provide equipment and gear at significantly reduced prices, or even cost prices. These include Emtrion, Siomar Battery Industries and Atlantek Vision.

A very big thank you must also be extended to all the staff in both the Mechanical and Electrical workshops in the School of Mechanical Engineering who have provided invaluable support to the project. Thanks also to Mr. Zebb Prime, Mr. William Robertson and Mr. William Constantine for their help with some technical issues experienced.
CONTENTS

VOLUME I
Executive Summary iii
Disclaimer v
Acknowledgements vii
Acronyms xv
1 introduction 1
  1.1 Background 1
  1.2 Significance 1
  1.3 Focus 2
    1.3.1 Initial Focus 2
    1.3.2 New Focus 2
    1.3.3 Design Goals 3
  1.4 Team Structure and Allocation 4
    1.4.1 Time and Expenditure 5
  1.5 Systems Overview 7
  1.6 Report Structure 7
2 literature review 11
  2.1 Autonomous Vehicles and Events 11
    2.1.1 Competitions 11
    2.1.2 Vehicles 13
  2.2 Locomotion 15
    2.2.1 Legged Systems 15
    2.2.2 Hover Systems 16
    2.2.3 Track Systems 16
    2.2.4 Omnidirectional Wheels 17
    2.2.5 Conventional Wheels 17
  2.3 Localisation 18
    2.3.1 Differential GPS 19
    2.3.2 Outdoor Localisation 22
    2.3.3 Indoor Localisation 25
  2.4 Mapping 30
    2.4.1 Physical Mapping Background 31
    2.4.2 Conceptual Mapping Background 34
  2.5 Autonomous Control 34
    2.5.1 Single Vehicle Autonomous Control 35
    2.5.2 Multiple Vehicles Autonomous Control 44
  2.6 Computer Vision 46
    2.6.1 The Difficulties of Computer Vision 46
    2.6.2 Colour Spaces 48
    2.6.3 Current Examples of Computer Vision 49
3 hardware 53
3 ugv mechanical and electrical design 55
3.1 Localisation and Mapping ........................................ 55
  3.1.1 Accurate Positioning ........................................ 56
  3.1.2 Environmental Awareness ................................. 60
  3.1.3 Distance to Object Measurement ......................... 60
  3.1.4 Component orientation control .......................... 63
3.2 Communication and Computation .............................. 71
  3.2.1 Communication ............................................ 71
  3.2.2 On-board Computation .................................. 74
3.3 Mobility and Electrical ....................................... 75
  3.3.1 Drive System ............................................ 75
  3.3.2 Power Supply ........................................... 83
  3.3.3 UGV Chassis ........................................... 85
  3.3.4 Hardware Layout ....................................... 88
  3.3.5 Future Work for Chassis ................................ 91
  3.3.6 Power Management and Data Connections ............... 91
3.4 Final UGV Design ............................................ 95
  3.4.1 Future work ............................................. 97
4 GCS DESIGN .................................................... 99
  4.1 GCS Hardware ............................................... 99
    4.1.1 Visual Displays ....................................... 101
    4.1.2 GCS Computer ....................................... 102
    4.1.3 User input Peripherals ............................... 104
    4.1.4 Wireless Communication Hardware ................. 104
    4.1.5 Differential Global Positioning Hardware ........ 104
    4.1.6 Hardware Connections ................................ 104
  4.2 GCS Mechanical Case ...................................... 105
    4.2.1 Requirements .......................................... 105
    4.2.2 Final Design .......................................... 106
  4.3 Future Work ................................................ 109
II SOFTWARE .................................................. 111
5 SOFTWARE DEVELOPMENT ...................................... 113
  5.1 System Specifications ....................................... 114
  5.2 Operating System ......................................... 115
    5.2.1 Inter Process Communication ......................... 116
  5.3 UGV Software ............................................... 117
    5.3.1 Motor Control ......................................... 117
    5.3.2 Ultrasonic Driver .................................... 120
    5.3.3 LiDAR Driver .......................................... 121
    5.3.4 IMU Driver ........................................... 123
    5.3.5 dGPS Driver .......................................... 124
    5.3.6 Camera Driver ........................................ 130
  5.4 GCS Software ............................................... 132
    5.4.1 Human Machine Interface ............................. 132
6 OUTDOOR LOCALISATION .................................... 137
  6.1 Functional Requirements ................................... 137
  6.2 System Model .............................................. 139
  6.3 Types of Kalman Filters ................................... 139
  6.4 State Space Model ......................................... 140
  6.5 Extended Kalman Filter ................................... 144
| 6.6  | Implementation                                                   | 146 |
| 6.7  | Tuning                                                          | 147 |
| 6.7.1| Dynamic Model                                                   | 147 |
| 6.7.2| Measurement Noise                                               | 150 |
| 6.7.3| Process Noise                                                   | 151 |
| 6.7.4| Final Tuning Values                                            | 152 |
| 6.8  | Results                                                         | 152 |
| 6.8.1| Full Differential GPS Reception                                 | 152 |
| 6.8.2| Differential GPS with Simulated GPS Dropout                     | 153 |
| 6.8.3| Full Standard GPS Reception                                     | 153 |
| 6.9  | Future Work                                                     | 156 |
| 7    | Indoor Localisation                                            | 157 |
| 7.1  | Background                                                      | 157 |
| 7.2  | Functional Requirements                                         | 158 |
| 7.3  | Integration with Project                                        | 159 |
| 7.4  | SLAM Process                                                    | 160 |
| 7.5  | Filter                                                          | 162 |
| 7.5.1| State Space Model                                              | 162 |
| 7.5.2| Extended Kalman Filter                                          | 167 |
| 7.5.3| Data association                                                | 172 |
| 7.5.4| Landmark Extraction                                            | 172 |
| 7.6  | Simulation Results                                              | 177 |
| 7.6.1| Testing landmark extraction                                     | 177 |
| 7.6.2| Testing data association                                        | 179 |
| 7.6.3| Testing entire SLAM process                                     | 179 |
| 7.7  | Future Work                                                     | 179 |
| 8    | Mapping                                                         | 181 |
| 8.1  | Physical Mapping                                                | 181 |
| 8.1.1| Occupancy Grid                                                  | 182 |
| 8.1.2| Occupancy Grid Testing                                          | 187 |
| 8.1.3| Visibility Map                                                  | 194 |
| 8.1.4| Elevation Map                                                   | 194 |
| 8.2  | Conceptual Mapping                                              | 195 |
| 8.2.1| Map Testing                                                     | 195 |
| 8.2.2| Exploration Map                                                 | 196 |
| 8.2.3| Feature Map                                                     | 197 |
| 8.2.4| Vantage Map                                                     | 199 |
| 8.3  | Future Work                                                     | 200 |
| 9    | Control                                                         | 203 |
| 9.1  | Four components of Control                                      | 203 |
| 9.1.1| World Perception                                                | 203 |
| 9.1.2| Waypoint Generation                                             | 204 |
| 9.1.3| Path Generation                                                 | 205 |
| 9.1.4| Path Tracking                                                   | 211 |
| 9.2  | Control Implementation                                          | 220 |
| 9.2.1| Manual Control                                                  | 220 |
| 9.2.2| Semi-Autonomous Control                                         | 221 |
| 9.2.3| Autonomous Control                                              | 222 |
| 9.3  | Implementation                                                  | 222 |


9.3.1 Manual Control ..................................................... 223
9.3.2 Semi-Autonomous Control .................................. 225
9.3.3 Autonomous Control ............................................. 225
9.4 Future Work .......................................................... 225

10 VISION ...........................................................................

10.1 Objects of Interest for Computer Vision ......................... 229
10.2 Object Detection ....................................................... 230
  10.2.1 Colour Thresholding ........................................ 230
  10.2.2 Image Morphology and Contours ........................ 232
  10.2.3 Other Object Detection Techniques Considered .. 233
10.3 Object Identification ................................................ 234
  10.3.1 Identification by dimensions ............................... 235
  10.3.2 Identification confirmation by colour .................. 235
  10.3.3 Identification confirmation by continuity .............. 236
10.4 Object Location ......................................................... 237
  10.4.1 Camera Calibration ........................................ 237
  10.4.2 Final Testing and Results ................................ 238
10.5 Future Work ............................................................ 240

III CONCLUSION .......................................................... 243

11 CONCLUSION .......................................................... 245

REFERENCES .................................................................... 247

Volume II ........................................................................ 1

IV APPENDIX ....................................................................

A APPENDIX A : ADMINISTRATION .................................. 3
  A.1 MAGIC 2010 Information Pack ..................................... 3
  A.2 MAGIC 2010 Project Contract ..................................... 43
  A.3 Project Expenditure and Student Hours ..................... 54
  A.4 Risk Assessment ....................................................... 54
  A.5 Safe Operating Procedure (SOP) ............................... 75
  A.6 Failure Modes and Effects Analysis (FMEA) ............. 79
B APPENDIX B : ADDITIONAL UGV AND GCS DESIGN INFORMATION 97
  B.1 Additional Hardware Selection Information .............. 97
    B.1.1 General Decision Factors ...................................... 97
    B.1.2 On-board Computer System ................................. 97
    B.1.3 Ethernet Switch .................................................. 98
    B.1.4 Wireless Communication ...................................... 99
    B.1.5 Light Detection and Ranging Sensor (LiDAR) ....... 100
    B.1.6 Camera ............................................................. 103
    B.1.7 Differential Global Positioning System (dGPS) ...... 105
    B.1.8 Inertial Measurement Unit (IMU) ......................... 106
    B.1.9 Material Types and Selection Criteria .................. 107
  B.2 Obsolete Hardware Selection .................................. 110
    B.2.1 Laser Pointer ..................................................... 110
    B.2.2 Input/Output Board ............................................ 110
  B.3 PTU and GCS Obsolete Designs ............................... 112
    B.3.1 PTU Operational Design Not Used: Stationary Positions ..... 112
# Appendix C: Drawings

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>Chassis Drawings</td>
<td>123</td>
</tr>
<tr>
<td>C.2</td>
<td>PTU Drawings</td>
<td>123</td>
</tr>
<tr>
<td>C.3</td>
<td>Coupling Drawings</td>
<td>139</td>
</tr>
<tr>
<td>C.4</td>
<td>Electrical Drawings</td>
<td>147</td>
</tr>
<tr>
<td>C.5</td>
<td>UGV Assembly Guide</td>
<td>161</td>
</tr>
<tr>
<td>C.5.1</td>
<td>On/Off Switch subassembly</td>
<td>161</td>
</tr>
<tr>
<td>C.5.2</td>
<td>Wireless Unit subassembly</td>
<td>164</td>
</tr>
<tr>
<td>C.5.3</td>
<td>Drive assembly</td>
<td>166</td>
</tr>
<tr>
<td>C.5.4</td>
<td>E-stop subassembly</td>
<td>168</td>
</tr>
<tr>
<td>C.5.5</td>
<td>Lex box subassembly</td>
<td>170</td>
</tr>
<tr>
<td>C.5.6</td>
<td>IMU subassembly</td>
<td>170</td>
</tr>
<tr>
<td>C.5.7</td>
<td>DC converter subassembly</td>
<td>171</td>
</tr>
<tr>
<td>C.5.8</td>
<td>DIN rail subassembly</td>
<td>172</td>
</tr>
<tr>
<td>C.5.9</td>
<td>GPS antenna subassembly</td>
<td>173</td>
</tr>
<tr>
<td>C.5.10</td>
<td>GPS unit subassembly</td>
<td>174</td>
</tr>
<tr>
<td>C.5.11</td>
<td>Ultrasonic sensor subassembly</td>
<td>175</td>
</tr>
<tr>
<td>C.5.12</td>
<td>EPOS units subassembly</td>
<td>175</td>
</tr>
<tr>
<td>C.5.13</td>
<td>Pan Tilt Unit subassembly</td>
<td>177</td>
</tr>
<tr>
<td>C.5.14</td>
<td>Overall Assembly Procedure</td>
<td>182</td>
</tr>
</tbody>
</table>

# Appendix D: Datasheets

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.1</td>
<td>UGV Drive System</td>
<td>185</td>
</tr>
<tr>
<td>D.1.1</td>
<td>UGV Drive Motor</td>
<td>185</td>
</tr>
<tr>
<td>D.1.2</td>
<td>UGV Drive Motor Encoder</td>
<td>185</td>
</tr>
<tr>
<td>D.1.3</td>
<td>UGV Drive Motor Gearbox</td>
<td>185</td>
</tr>
<tr>
<td>D.2</td>
<td>UGV PTU Tilt System</td>
<td>189</td>
</tr>
<tr>
<td>D.2.1</td>
<td>Tilt Motor</td>
<td>189</td>
</tr>
<tr>
<td>D.2.2</td>
<td>Tilt Motor Planetary Gearhead</td>
<td>189</td>
</tr>
<tr>
<td>D.2.3</td>
<td>Tilt Motor Worm Gearhead</td>
<td>189</td>
</tr>
<tr>
<td>D.2.4</td>
<td>Tilt Motor Encoder</td>
<td>189</td>
</tr>
<tr>
<td>D.3</td>
<td>UGV PTU Pan System</td>
<td>194</td>
</tr>
<tr>
<td>D.3.1</td>
<td>Pan Motor</td>
<td>194</td>
</tr>
<tr>
<td>D.3.2</td>
<td>Pan Motor Planetary Gearhead</td>
<td>194</td>
</tr>
<tr>
<td>D.3.3</td>
<td>Pan Motor Encoder</td>
<td>194</td>
</tr>
<tr>
<td>D.4</td>
<td>GPS System</td>
<td>198</td>
</tr>
<tr>
<td>D.4.1</td>
<td>GPS Receiver</td>
<td>198</td>
</tr>
<tr>
<td>D.4.2</td>
<td>GPS Receiver Enclosure</td>
<td>198</td>
</tr>
<tr>
<td>D.4.3</td>
<td>GPS Antenna</td>
<td>198</td>
</tr>
<tr>
<td>D.5</td>
<td>Pan-Tilt Unit</td>
<td>205</td>
</tr>
<tr>
<td>D.5.1</td>
<td>Camera</td>
<td>205</td>
</tr>
<tr>
<td>D.5.2</td>
<td>Camera Lens</td>
<td>205</td>
</tr>
<tr>
<td>D.5.3</td>
<td>LiDAR</td>
<td>205</td>
</tr>
<tr>
<td>D.6</td>
<td>Other UGV Sensors</td>
<td>210</td>
</tr>
<tr>
<td>D.6.1</td>
<td>IMU</td>
<td>210</td>
</tr>
<tr>
<td>D.6.2</td>
<td>Ultrasonics</td>
<td>210</td>
</tr>
<tr>
<td>D.7</td>
<td>Communication and Computer</td>
<td>216</td>
</tr>
<tr>
<td>D.7.1</td>
<td>UGV Onboard Computer</td>
<td>216</td>
</tr>
</tbody>
</table>

xiii
D.7.2 Wireless Router ........................................ 216
D.7.3 UGV Four-port Router ................................. 216
D.8 Power Source and Distribution ......................... 221
D.8.1 24V DC-DC Converter ................................. 221
D.8.2 5V and 12V DC Converters ........................... 221
D.8.3 Batteries ................................................. 221
E APPENDIX E : HAND CALCULATIONS .................. 227
E.1 Drive Assembly Motors .................................... 227
E.1.1 Speed requirements for wheel motors ............... 227
E.1.2 Bearing Load Calculation .............................. 227
E.2 GPS ......................................................... 231
E.3 Inertial Measurement Unit ............................... 231
E.4 Pan/Tilt Unit ............................................. 233
F APPENDIX F : TEST PROCEDURES ....................... 243
F.1 Test Procedures .......................................... 243
F.1.1 Initial Drive Test ........................................ 243
F.1.2 Tilt Motor Test ......................................... 243
F.1.3 UGV Weight Test ....................................... 244
F.1.4 UGV Turning Force Test .............................. 245
F.1.5 Drive test (with design changes) ..................... 245
F.1.6 Battery Runtime Test ................................... 246
G APPENDIX G : SOFTWARE DEVELOPMENT ................. 247
G.1 QNX Operating System .................................... 247
G.1.1 Real-Time Operating System .......................... 247
G.1.2 QNX ...................................................... 248
G.1.3 Message Passing ........................................ 248
G.2 Ubuntu ..................................................... 249
G.2.1 Ease of Use .............................................. 249
G.2.2 Graphical Applications ............................... 249
G.2.3 Portability .............................................. 249
G.2.4 Cost ..................................................... 250
G.3 OpenCV ...................................................... 250
G.4 Camera Data Format ...................................... 252
G.5 Image Analysis using MATLAB ......................... 252
G.6 Kalman Filter Comparison .............................. 269
G.7 Mapping .................................................... 273
G.8 Control ..................................................... 273
H APPENDIX H : SOFTWARE CODE ......................... 277
H.1 UGV Code .................................................. 277
H.1.1 Drivers .................................................. 277
H.1.2 Localisation ............................................. 336
H.2 GCS Code .................................................. 393
H.2.1 Mapping ............................................... 393
H.2.2 Vision .................................................. 427
H.2.3 Control .................................................. 445
I APPENDIX I : FIRMWARE SETTINGS ....................... 457
ACRONYMS

AWG  American Wire Gauge
CAD  Computer Aided Design
CAN  Controller Area Network
CFD  Computational Fluid Dynamics
CMY  Cyan Magneta Yellow
CPU  Central Processing Unit
DARPA  Defence Advanced Research Projects Agency
DC  Direct Current
dGPS  Differential Global Positioning System
EC  Electrically Commutated
EKF  Extended Kalman Filter
EPOS  Electronic Positioning
E-Stop  Emergency Stop
FEA  Finite Element Analysis
GCS  Ground Control Station
GPL  General Public License
GPS  Global Positioning System
GUI  Graphical User Interface
HD  Hard Drive
HMI  Human Machine Interface
HSV  Hue Saturation Value
IMU  Inertial Measurement Unit
INS  Inertial Navigation System
IP  International Protection Rating
LCD  Liquid Crystal Display
LiDAR  Light Detecting and Ranging sensor
MAGIC  Multi-Autonomous Ground International Challenge
MoD  Ministry of Defence
OS  Operating System
OOI  Object of Interest
OpenCV  Open Source Computer Vision
PCB    Printed Circuit Board
PCI    Peripheral Component Interconnect
POE    Power Over Ethernet
POSE   Position and Orientation
PTU    Pan Tilt Unit
RAM    Random Access Memory
RANSAC Random Sampling Consensus
RFID   Radio Frequency Identification
RGB    Red Green Blue
RPM    Revolutions Per Minute
RTOS   Real Time Operating System
SLAM   Simultaneous Localisation and Mapping
TCP/IP Transmission Control Protocol / Internet Protocol
TNC    Threaded Neill-Concelman
UAV    Unmanned Aerial Vehicle
UGV    Unmanned Ground Vehicle
UKF    Unscented Kalman Filter
USB    Universal Serial Bus
INTRODUCTION

1.1 BACKGROUND

The Multi-Autonomous Ground-robotic International Challenge (MAGIC) was instigated by the Australian and American departments of defence, the Defence Science and Technology Organisation (DSTO) of Australia and the USA equivalent, Research Development and Engineering Command (RDECOM) with the purpose of developing technology related to coordinated teams of autonomous robots. The challenge requires competitors to engineer a team of robotic vehicles, which are able to complete a reconnaissance mission in a mock combat environment, featuring both static and mobile threats. The competition comes with a USD 750,000 first prize, and hence, attracted significant interest from large universities and companies around the world. Eligibility into the challenge was based on the submission of a technical proposal, which led to the selection of twelve teams, ten of which were provided USD 50,000 to develop their proposal. One of these ten teams was a consortium of interested parties mainly consisting of a partnership between Strategic Engineering, a Sydney based consultancy firm, and an undergraduate team from the University of Adelaide. In addition to this monetary bonus, the University of Adelaide provided a further AUD 100,000 towards the development of the technical proposal, as a team of final year engineering students at the university took on the challenge as their design and build honours project.

1.2 SIGNIFICANCE

Completely autonomous systems are an area of significant research and development. These systems are also of interest to the defence industry, the robotics industry as well as in the academic world. Autonomous systems are made up of many subsystems of which are considered mature in their development; however, the complete integration of these systems is still in its infancy. This requires the integration of hardware with software, as well as the integration of software modules with the underlying software architecture. Autonomous robotics is still rapidly developing and significant research and technological developments are ongoing. Thus, the MAGIC competition is significant due to the fact that it is generating more interest to the area of autonomy as well as potentially accelerating its development among the robotics community.

Moreover, this project is notably different from many other common autonomous applications, as it is focused on multiple autonomous control and co-ordination. Most current applications are focused on single autonomous vehicle control. However, this venture requires control on both a macro level, such that the team of robots are viewed as one large autonomous unit, and also on a micro level, which includes the autonomy of individual robots, but also focuses right down to the individual systems on-board each of the units.

In this particular application, the vehicles being designed were required to traverse both outdoor and indoor environments effectively. These two aspects of fully autonomous control required numerous pieces of hardware and associated software to be selected and developed to achieve both accurate indoor and outdoor localisation, as well as precise mapping for robust navigation of a system of autonomous robots.

The applications of this type of system are varied. The ideas explored and implemented in this project can be developed into a defence-related system including but not restricted to reconnaissance,
exploration, search and rescue and other battle-zone applications. It can also have significant importance to the robotics industry for practical applications involving semi-intelligent autonomous robotics. In the future this sort of technology may also be adapted for domestic situations such as search and rescue, hostage scenarios and other practical industrial applications.

1.3 Focus

Throughout the course of the students involvement in the project, the focus shifted dramatically. The focus was initially to satisfy and compete in the MAGIC competition. This was required to be reviewed when the teams entry into the final stage of the competition was unsuccessful. Hence the focus shifted to account for the reduced support of project sponsors, loss of extra non-student members involved and to broaden the scope of possible future uses of the vehicle. Although the focus was altered, much of the original intent was maintained, so the initial focus will be discussed and then the alterations will be noted.

1.3.1 Initial Focus

The initial focus for the project was based on the MAGIC 2010 competition, and hence, the majority of the decision factors involved in the development of the Unmanned Ground Vehicles (UGVs) were based on requirements related to this competition.

The initial MAGIC performance criteria required that there be multiple Unmanned Ground Vehicles (UGVs) that are able to communicate wirelessly to a central command center, referred to as the Ground Control Station (GCS). The MAGIC competition required that there be two types of UGVs, one being capable of mapping and localisation (called the sensor UGV), the other capable of neutralising threats found (called the disruptor UGV). While developing the UGVs it was decided that the requirements for a sensor and disruptor UGV were essentially identical and hence they were combined into one design as this gave the disruptor UGV the same level of environmental awareness as the sensor UGVs.

The scope of the project up till June of 2010 at the University of Adelaide, was focused on development in a few key areas required for the competition. These key areas included chassis design and pan/tilt unit and many of the software systems, including developing indoor and outdoor localisation, mapping, computer vision and some hardware drivers.

1.3.2 New Focus

In July of 2010 when the team was unsuccessful in progressing to the final stage of the competition, the focus shifted to allow for the reduced support in developing the project and to better suit the desires of the new client, the University of Adelaide. In particular, the University desired a capable and robust platform that could be used for the further development in the areas of localisation, mapping, camera vision and autonomous control. The goals of the project were simplified somewhat to reflect the reduction of resources available and the commitment required by the student in areas initially covered by other parties.

Localisation: The ability to map the robots environment accurately was an underlying requirement for the MAGIC competition as it forms the basis of the mapping tasks. To produce the maps accurately and to effectively navigate, the robots needed to be able to determine their position as precisely as practical. Subsequently, it was decided that an accuracy of within 200mm in comparing
the virtual maps and the real world environment was desirable and would enable successful completion of all MAGIC requirements. Different algorithms and methods were required to achieve both indoor and outdoor localisation. The significance of considering both scenarios is the lack of GPS signal indoors, while GPS systems can be utilised in most outdoor environments. Hence a Kalman filter was utilised to combine differential Global Positioning System (dGPS) data and an Inertial Measurement Units (IMU) data for outdoor environments, while simultaneous location and mapping (SLAM) was used for indoor environments, which utilised the IMU and a Light Detection and Ranging (LiDAR) unit.

**Mapping:** Mapping was placed into two categories. Physical mapping provides a physical representation of the UGV surroundings in the form of a three-dimensional occupancy grid. The second category was that of conceptual maps. These were designed to stem from the physical maps and use the information gathered to interpret the surroundings and enable the robots to have some form of world perception. These primarily came in the form of features discovered in the competition area, heights of objects detected and an estimation of areas with the greatest outlook or vantage within the local vehicle surroundings. All of this information is fed both into a Human Machine Interface (HMI), where human operators are able to monitor and direct UGV movements if necessary, and into an autonomous control system, providing crucial information about the surrounding environment for automatic UGV navigation decisions to be made.

**Control:** The robots were required to be controlled autonomously both on an individual basis and in a team. The development of this area of the project initially focused on tele-operation of the UGVs and then progressed to a fuzzy logic semi-autonomous path tracking algorithm as well as significant research into waypoint generation for fully autonomous control. Due to the Adelaide University being in possession of only one fully working UGV for the majority of the year, team automation was not prominent in the development focus.

**Camera Vision:** A further design requirement included the detection, identification and locating of three competition-defined objects of interest that were either mobile or stationary. The camera vision focused primarily on colour to detect and identify objects of interest as the MAGIC competition organisers provided objects of interest that were distinguished by bright red or navy blue colours.

**Unmanned Ground Vehicle (UGV):** A significant number of highly specialised goals were required by the MAGIC competition and hence it was decided that a customised platform solution was required to successfully apply all of these goals, hence a significant amount of time and money was spent on this aspect.

**Ground Control Station (GCS):** The ability to remotely monitor and take over control if necessary was highly desirable and hence a remote base station was designed to display relevant data and provide a human machine interface (HMI).

Section 1.3.3 discusses the particular requirements agreed upon in the contract made with the University.

### 1.3.3 Design Goals

The primary goals for the project was a compilation of requirements from the original MAGIC rules document (see Appendix A) and were adapted to a set of specific goals set down in the student project contract. This. The eventual non-participation in the MAGIC competition presented an opportunity to re-evaluate the desired specifications for the UGV. A complete set of desired specifications are included in the project contract (Appendix A).
1.4 TEAM STRUCTURE AND ALLOCATION

Due to the MAGIC 2010 project being of significant size and complexity, with a large number of people working on it simultaneously the need to effectively allocate roles and manage these roles was crucial in achieving a successful result. In particular, with the sharing of resources between Strategic Engineering in Sydney and the University of Adelaide, concise and effective communication was critical. The project management, though largely undertaken by Richard Aplin from Strategic Engineering, required some sharing of managerial roles especially in-house at the University of Adelaide. Communication between the Adelaide team and Richard Aplin was largely undertaken by Associate Professor Benjamin Cazzolato, Dr Christopher Madden and Nigel Gaskin, the elected student team leader. The student leader’s role, in particular, was similar to that of a systems engineer and required coordination of all the sections, which unite the entire project.

When the project was still working towards the goal of successful participation in the MAGIC 2010 competition, the project team was quite considerable and consisted of primarily the following, however several consultants were also involved on occasions:

University of Adelaide:
- Ten undergraduate honours students
- Four specialist researchers supervising the students
- A part time programmer

Strategic Engineering:
- Numerous undergraduate students working full time (for their work experience); approximately ten at any one point in time, this number fluctuated as new students started their placement and others finished
- Four specialist engineers
- A full time programmer

Richard Aplin, the owner of Strategic Engineering had the role of overseeing the whole project and communicating between all vested parties. Strategic Engineering were responsible for initial system design and conception, as well as initial UGV design and driver development as this occurred before the inclusion of the University of Adelaide student team.

The University of Adelaide team was tasked with the development and refinement of mechanical system designs including the designs of the UGV chassis and pan-tilt unit. Additionally, the team had input in the development of the drivers for many of the hardware devices being used. The team from the University of Adelaide was also responsible for the development of other software, such as localisation programs, mapping software, as well as computer vision and autonomous control. Dr Chris Madden and Mr Wynand Marias were employed by the university for some of this period to lead the software development primarily focusing on the mapping and vision systems. Dr Brad Alexander was also conducting research into high level mapping and control algorithms, suited to this application.

The MAGIC 2010 student project team allocated tasks on the basis of member’s strengths. In addition to the particular technical section allocated to each individual, specific roles assisted in defining tasks and responsibilities in administrative areas. The administrative roles seen in Table 1.1 were chosen by the group for the purpose of organisation and assuring that all aspects of the required project could be covered effectively. The technical tasks seen in Table 1.2 have developed significantly as the project has progressed and the completion of the project has required many students to diversify into several of the technical areas.
Table 1.1: Administrative Roles

<table>
<thead>
<tr>
<th>Administrative Roles</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team Leader</td>
<td>Nigel Gaskin</td>
</tr>
<tr>
<td>Secretary (Entire team)</td>
<td>Mark Baulis</td>
</tr>
<tr>
<td>Timesheet Keeper</td>
<td>Konrad Pilch</td>
</tr>
<tr>
<td>Treasurer</td>
<td>Benjamin Quast</td>
</tr>
<tr>
<td>Secretary (Mechanical Team)</td>
<td>Phuong Huynh</td>
</tr>
<tr>
<td>Report Coordinator, T-Shirt Design, Posters, Exhibition Video</td>
<td>Stella Wong</td>
</tr>
</tbody>
</table>

Table 1.2: Technical Task Allocation

<table>
<thead>
<tr>
<th>Technical Tasks</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdoor Localisation, Driver and Software Development</td>
<td>Anton Skeketee</td>
</tr>
<tr>
<td>Indoor Localisation</td>
<td>Konrad Pilch</td>
</tr>
<tr>
<td>Ground Control Station Design</td>
<td>Stella Wong</td>
</tr>
<tr>
<td>Conceptual Mapping and Control</td>
<td>Sundar Komandurelaiyavalli</td>
</tr>
<tr>
<td>Pan/Tilt Unit and Electrical Wiring</td>
<td>Adam Cundy</td>
</tr>
<tr>
<td>UGV Design and Alterations</td>
<td>Benjamin Quast and Phuong Huynh</td>
</tr>
<tr>
<td>Computer Vision, Driver and Software Development</td>
<td>Mark Baulis</td>
</tr>
<tr>
<td>Systems Co-ordination, Mechanical Alterations, Control Implementation and Driver Development</td>
<td>Nigel Gaskin</td>
</tr>
</tbody>
</table>

Following the unsuccessful June down selection process, the project continued in a purely research and development manner as an honours project at the University of Adelaide. This honours project was no longer in partnership with Strategic Engineering, as they were only a competition partner. Furthermore, due to limited funds, Dr Chris Madden and Mr Wynand Marias had limited support afterwards as well, phasing out their involvement. One final change included one of the student members, Mr Peter Hardy, who withdrew from the project for health reasons.

1.4.1 TIME AND EXPENDITURE

The total project budget available for MAGIC 2010 was $100,000. All of this funding was generously provided by the University to allow the project the utmost chance in succeeding. The total project expenditure to date is $92,103. The breakdown of the expenditure can be seen in Figure 1.1.
The expenditure is divided into personnel, UGV and GCS hardware and manufacturing and miscellaneous costs. Personnel represents the salary of academic personnel that were hired to help with the project. This help allowed students to work with experts in the field of programming and computer vision, allowing development in these areas of the project that would otherwise been insurmountable. UGV hardware and manufacturing represents the costs of all components purchased by Adelaide University relating to the UGV, including any manufacturing processes related to the UGV that were outsourced. GCS Hardware costs represent the costs of all components purchased by Adelaide University relating to the GCS Miscellaneous covers various expenditures related to the project that are unable to be included under the previously mentioned expenses. Miscellaneous expenses included costs such as student and staff training and traveling expenses that were required due to working with a company in Sydney.

The final UGV value in terms of on-board hardware for one complete UGV has been evaluated to be $24,617.50. The value of the UGV hardware differs to total expenditure on UGV hardware due to sponsorship deals with various suppliers and that the purchasing of hardware was divided between Adelaide University and project partner Strategic Engineering. The project was able to acquire the hardware to produce two complete UGVs and the sponsorship deals and partnership with Strategic Engineering allowed $49,235 worth of hardware to be purchased at $35,429. This was an effective saving of $13,806. A breakdown of the individual value of all hardware required to produce the final UGV design is detailed in Appendix A.

Besides hardware, the UGVs required a significant amount of time to be invested, both by the students and the mechanical and electrical workshop staff. The student time invested totaled to a combined 6491.13 hours as of the 17th of October. The total student labour costs were calculated at the assumption that the students would be receiving a $50,000 salary. Assuming a 40 hour working week for 52 weeks of the year, this equates to $24.04 per hour. Also direct and indirect costs associated with each student were calculated where where direct costs were assumed to be 30% of the received salary and indirect costs were to be 130% of the received salary. The total cost of the students based on these assumptions if they were employed by the University for the duration of the project are detailed in Table A.4.

Each project group was allotted a certain amount of mechanical and electrical workshop time for the year. This workshop time was provide free of charge, however these cost must still be calculated to determine value of the project. Workshop time was defined as both manufacturing time and use
of staff expertise. The combined workshop time used by project 1025 amounted to 118.5 hours as of the 17th of October. Assuming the workshop time was valued at $50 per hour, the resulting cost of the workshop use was $5,925. A breakdown of the monthly workshop hours used per workshop is detailed in Appendix A.

### 1.5 SYSTEMS OVERVIEW

When the student group began working on MAGIC 2010 there had already been a significant amount of work completed, largely by Strategic Engineering. Although positive in terms of progress, this presented some problems early on since there was a steeper learning curve required for the students and restrictions as to the direction of progress, due to the decisions already made before embarking on the project. Initially, the students were allocated specific areas of responsibility that they set out to complete diligently. However, there were many links missing between sections, thus, when it came to integrate systems, there would be considerable lost time and productivity. Hence, in order to clearly define the project in a simple and easily understood manner a series of carefully designed flow charts were devised to relate each system together and also to visually see how these systems developed from the highest level to the inner workings and complexities that made up the whole of MAGIC 2010. Figure 1.2 shows the overall systems flowchart and demonstrates how each of the systems relate from both the hardware and software perspectives. Developing these flow charts proved to be invaluable in summarising the project and enabled simple allocation of sections that had yet to be covered. In developing these flow charts great care was taken with the intention that a non-project related individual with a medium level of technical understanding could be able to quickly understand the project. The flow charts referred to in the systems flow chart can be found throughout the report in the context referred to.

Several other system related tools were used in developing this project such as a risk assessment, safe operating procedure and failure modes and effects analysis (FMEA). These were all important for analysing areas of possible risk and can be viewed in Appendix A.

### 1.6 REPORT STRUCTURE

The project has progressed significantly closer towards completing all design specification goals with visible progress in areas such as research, planning, design, build and software development. A project of this magnitude and breadth requires a significant amount of research in the areas of locomotion, particularly about wheeled, legged, hover, and conventional locomotion, localisation,
including GPS/IMU integration, Kalman filters, and Simultaneous Localisation and Mapping (SLAM), mapping techniques, including physical mapping and conceptual mapping, vision analysis, concentrating on object detection, and also autonomous control, which incorporates the design and implementation of world perception, waypoint generation, path generation and path tracking algorithms. A substantial amount of background work has been carried out in these fields and this has been presented in the literature review in Chapter 2.

From here the report is divided into two volumes, with this being the first volume and further broken up into three sections. The first concentrates on hardware design and development, the second part concentrates on software design and development and the third part contains a wrap up of the project overall. Volume two of this report contains all the appendices holding extra information required for this project including, significant amounts of work completed in both hardware and software that became either unnecessary or required a new direction for success. The appendices also have detailed assembly instructions, drawings, datasheets and test procedures. They also contain all relevant calculations, firmware settings and copies of code.

Part 1 is comprised of Chapters 3 and 4, which detail the design, development and testing of the UGV and GCS design respectively. These chapters discuss the hardware selected to achieve the goals of localisation, mapping, communication, computation, visual identification of objects of interest and control, as well as the function of the selected hardware. Discussion of chassis development and drive system method and equipment is also discussed and final overall UGV design is presented.

Part 2 concentrates on the software design and development. The UGVs require a number of software modules in order to convert raw data from a number of different sensors into meaningful form for such uses as localisation, mapping, control and navigation. Chapter 5 describes the software development process that has occurred as part of this project. Chapter 6 and 7 detail the methods used to achieve outdoor localisation and indoor localisation respectively as well as detail explaining why these methods were selected. Chapter 8 describes the different mapping techniques used, Chapter 9 goes into further detail about the control system used and Chapter 10 explains how computer vision is utilised in this project.

Part 3 details to progress that has been made by the team at the University of Adelaide and the possibilities for future work in this area.
Over the past decade, autonomous machines have become increasingly popular, with many events and competitions encouraging research in this area and most recently in the area of multi-autonomous robotic systems. Advances in technology have offered the ability to build autonomous robots, and more recently, systems of autonomous robots. The main difficulty in building multi-autonomous robotic systems lies in the advanced architectural structure and intelligence required for a coordinated system of autonomous robots to execute tasks in real time.

Unmanned ground vehicles (UGVs) are effective in carrying out tasks that would otherwise be difficult or risky for a conventional manned vehicle. Hence, there is an extensive range of applications in which multiple UGVs may be beneficial, such as urban hostage situations, disaster relief and military conflicts. In military applications (see Figure 2.1), robots are used to save ‘friendly’ lives and reduce direct risks to human life by exposing robots to the unknown and volatile situations faced in the field instead. Many have already been deployed in the Iraq war, ranging from combatant robots to scouting robots. These ground robots range from 15 kg to 43 tonnes and have a wide range of capabilities. This project was aimed at building a robot to compete in the MAGIC 2010 competition; however the robot may be developed to accomplish other similar tasks in the future.

Many defence departments are aware of the significance that autonomous robot teams will have in future warfare. This is verified by a number of robot competitions with cash prizes as incentives, instigated to further develop the technology within this area. Commonly these competitions have involved mock urban environments with a number of threats or stimulants that require advanced robot intelligence to interact appropriately.

### Competitions

Robot competitions are gaining popularity, generally initiated as a means of increasing technological awareness of the robotics industry, bridging the gap between industry and educational
institutions and to promote development of technologies that could be used in various military uses. Participants include hobbyists, private firms, educational institutions and research institutions. Most often the competitions require custom-made robots to fulfil a number of tasks while satisfying design constraints in the form of size and weight restrictions. Many of the tasks involved in such competitions are set in urban environments and include navigational exercises, object or threat identification and intelligent response to various stimuli. To encourage proper design processes, the competitions are separated into stages that normally require the submission of a technical proposal before continuing into the next stage. These may also include presentations detailing how the robot was designed and built as part of the competition assessment. Many past national robotics competitions of a large scale have been organised by departments of defence where cash prizes are offered as incentives. The MAGIC 2010 competition is similar to a number of other competitions that have run over the past few years, hence, the following competitions were investigated to assist with ideas for the types of technology required in this project.

The Defence Advanced Research Projects Agency (DARPA) competition was held in 2004, 2005 and 2007 with the aim of developing and demonstrating abilities in autonomous unmanned ground vehicles. In 2004 and 2005, the competition required a fully autonomous vehicle to navigate set routes that traversed narrow tunnels and several sharp turns. In 2007, the task included dynamic traffic set in an urban environment that required the vehicle to abide by all road rules and merge into traffic, all while negotiating other traffic and obstacles.

In 2008, Singapore’s first government sponsored robotics competition, the TechX challenge required robots to navigate through outdoor and indoor urban settings, climb stairs, operate a lift, search and engage designated static targets before proceeding back to the starting point as seen in Figure 2.2. The competition was modelled after the DARPA competition in that it aimed to encourage adapting existing technology for various military uses. None of the six finalists completed the mission with most teams encountering difficulties at the stage of operating the elevator. This was primarily due to the case of the reflective nature of the elevator panels disrupting many sensors.

Figure 2.2: Sequence of tasks required for the TechX competition (Defence Science and Technology Agency 2007)
The United Kingdom Ministry of Defence (MoD) initiated a national robotics competition in which the robots were required to identify all threats including snipers, explosive devices, enemy vehicles and troops in a cluttered urban environment before reporting the information back to ground troops. Unlike the previous competitions described, unmanned aerial vehicles (UAV) were permitted. To successfully complete the challenge within the one hour time limit, teams of robots were used to systematically survey the area for threats. This required highly complex system architecture for robot coordination.

Similar in many respects to each of these competitions, the Multi-Autonomous Ground-robotic International Challenge (MAGIC) focused on the coordination of a team of robots to achieve autonomy. MAGIC was established jointly by the U.S. and Australian Departments of Defence to inspire innovative designs that may be used for military applications. The competition tested a broad range of skills from navigational abilities as needed in DARPA, intelligent response as required in TechX and highly complex system architecture as seen in the MoD challenge.

### 2.1.2 VEHICLES

Many military robots and winning entrants from robotic competitions demonstrate impressive designs and features. As many of these competition goals are similar in line with the tasks proposed in the MAGIC challenge, some of these designs and features may be utilised for this project and are examined here.

The 2007 winner of the DARPA challenge was Boss from Tartan Racing as seen in Figure 2.3. The vehicle was mounted with more than a dozen lasers and cameras to autonomously navigate the road with live traffic. High level route planning was used in determining the best pathway through a busy road network. Motion planning required consideration of static and dynamic objects as well as road boundaries, stop lines, speed limits and parking lot boundaries. Although the scenario differs from MAGIC, Tartan Racing’s use of a lasers and cameras was utilised in the MAGIC project.

The winner of the UK Ministry of Defence Grand Challenge, the Stellar team produced a fully autonomous system of robots with two types of UAVs (a high level and medium level), one UGV (See Figure 2.5a) and a Ground Control Station (GCS) as seen in Figure 2.4. The GCS received transmitted sensor feeds (optical and infra-red imagery from the UAVs, and optical, radar and infra-red imagery from the UGV). Hence, the UGV was equipped with a thermal imaging camera to obtain infra-red imagery. The software architecture of the system was intensely packed with threat detection software, a logging database and decision making software that guided the robots appropriately. Many aspects of this were found to be useful for MAGIC, including the need for
feedback of sensor data to a GCS. Although the use of infra-red imagery was not utilised for this particular project in favour of more reliance on camera vision, it is still an option for further development.

Figure 2.4: Stellar’s high-level architecture (Wahren 2010)

Another contestant of the MoD Challenge, team MIRA competed in the competition with a UGV, UAV and a blimp. The autonomous UGV, named MIRA Autonomous Control Engineering (MACE), featured an advanced marksmen system that allowed quick detection of observers using lenses from binoculars, TV cameras as well as sniper scopes (see Figure 2.5b). It senses the retro-reflection from a prepared lens and sends the information to the GCS. The use of UAV data is highly desirable for further development of the MAGIC project and reduces the reliance on onboard sensors; however, it was decided not to undertake this added level of complexity for this project.

Wayfarer robots developed by the US military defence excel in urban reconnaissance and surveillance by following walls and streets while mapping (see Figure 2.6). The Wayfarer Packbot uses LiDAR and stereo vision for obstacle avoidance and mapping, as well as a Global Positioning Sensor (GPS) and odometry for localisation. This use of LiDAR and vision systems for obstacle avoidance and mapping was found to be a significant requirement for the MAGIC project, and hence, research in this area was investigated further in Section 2.4.
2.2 LOCOMOTION

There are several different forms of locomotion for robots and choice is largely dependent on requirements and terrain type that the robot will be traversing. The first robots were based on platforms fitted with conventional wheels attached to portable motors. As technology advanced, several other forms of locomotion evolved from legged systems attempting to emulate human movement, hover systems designed to remove difficulties associated with the terrain, track systems widely used in military vehicles and omnidirectional wheels, a recent development from the conventional wheels. Each of these is investigated as a potential means of locomotion for the MAGIC robots.

2.2.1 LEGGED SYSTEMS

Inspired by biological systems, legged systems offer advantages in manoeuvrability and are very successful in moving through a wide variety of harsh environments. However, such a system imposes higher design complexity and requires extensive resources to construct and control.

An example of a legged robot used in military applications is the Big Dog robot designed by Boston Dynamics (see Figure 2.7a). Balanced on four legs closely emulating dog limbs, the robot is capable of transporting up to 150kg across rough terrain including inclines of up to 35° (Boston Dynamics 2008).

Bipedal robots are classified under two legged robots and are the most challenging of all legged systems as stability is often difficult to achieve. It is due to this difficulty that sophisticated control is required and can incur large expenses. This is demonstrated by Honda’s ASIMO (see Figure 2.7b), a humanoid that boasts impressive human-like movement but at extremely high costs.

Six legged robots provide higher stability resulting in easier control but entails harder coordination of the limbs. Full simulation of insect-like locomotion still remains a challenge as insects combine a small number of active degrees of freedom with passive structures, such as microscopic barbs and textured pads which have yet to be efficiently achieved. Currently, one of the most advanced six legged robots is the Lauron IV, developed by the FZI Karlsruhe as seen in Figure 2.7c.

Research has also demonstrated that power consumption in legged systems is nearly two orders of magnitude higher than for wheeled locomotion on hard, flat surfaces (Roland 2004). For the MAGIC robot, power consumption is a major concern because it directly affects the operating...
duration of the robot, and the terrain that the UGVs are required to traverse is relatively flat. Hence, this form of locomotion might only be considered as a future work prospective.

2.2.2 Hover Systems

Hovercrafts use pressurised air propelled through cushions at the base of the vehicle in order to hover and travel. As the vehicle is supported by air, it is able to travel over a wide range of terrain including water, land and ice. As a result, many commercial hovercrafts are built on large scales with the capacity to carry passengers.

Unlike wheeled vehicles that use encoders to measure the distance travelled that can be used to estimate position, localisation techniques with hovercrafts are difficult. Hovercrafts have a very high slide factor at turns, varying velocity based on terrain conditions, and hence, are unstable in control. Likewise, complete autonomy is also difficult to achieve and thus, are not popular in unmanned robotics. For the MAGIC robots, localisation is a major concern, because it directly affects the accuracy of the mapping, and subsequent robot obstacle negotiation and object placement. Thus, the use of hovercraft technology for the MAGIC robots would not be ideal, but has some promise as demonstrated by the 2009 Final Year Project “Project HOVER”, in which an autonomous hovercraft was built for mine detection.

2.2.3 Track Systems

Track systems are especially efficient over rough terrain due to their large contact areas between the tyre and ground. As a result, UGVs constructed with tracks generally perform better than conventional wheels over loose surfaces. Thus, it is popular in military use where rough environmental conditions are commonly encountered. Further development of tracks have resulted in the addition of flippers that increase ease in traversing uneven terrain (see Figure 2.8).

Due to the nature of a tracked system, only skid steering is possible for directional manoeuvring. Skid steering controls the orientation of the robot on the surface by reversing the velocities of each side so that rotational movement on the spot is possible. This is effective on low friction surfaces however the use of tracks means that localisation via the use of encoder counts or velocity control is generally inaccurate as the tracks do not necessarily correspond to the rotations or velocity of the motors. For the MAGIC project, this form of localisation was deemed suitable; however, it was eventually decided that a more optimal solution was desired, since the need for cross-terrain performance was not a high requirement, whereas the need for accurate localisation was critical.
2.2.4 OMNIDIRECTIONAL WHEELS

Omnidirectional wheels offer movement in any direction despite the orientation of the robot. They generally consist of several small angled rollers embedded along the circumference of the primary wheel as seen in Figure 2.9.

These wheels offer significant advantages in congested environments due to their high manoeuvrability. Hence, they are popularly utilised in service vehicles in airports and factory equipment. However, in robotic applications, localisation is complex with these wheels, and they are generally used in flat environments, which poses a potentially serious problem should the MAGIC robots face any off-road travel.

2.2.5 CONVENTIONAL WHEELS

Conventional wheels have been the most popular choice of locomotion in both mobile robotics and vehicles in general. They achieve very high efficiencies with relatively simple mechanical implementation. Moreover, conventional wheels have greater ease with localisation and mapping methods at low construction costs. Currently, there are two main steering systems that are used by many ground vehicles; Ackermann steering as used in four wheeled automobiles and skid steering which is based on controlling the velocities of the motors on each side of the vehicle.

2.2.5.1 ACKERMANN STEERING

Ackermann steering requires an input which typically changes the direction of the front wheels, allowing the vehicle to turn (see Figure 2.10a). For robot applications, this method of steering...
only requires a simple controller that controls the steering motor. However, it involves fairly high geometrical accuracy of the steering motor which imposes some difficulty. A disadvantage of Ackermann steering used in four wheeled automobiles is that the system may have difficulty traversing rough, slippery or boggy terrain. This has been overcome in the past by using a combination of Ackermann steering and caterpillar tracks in some military vehicles known as “halftracks”, which was a possible solution for the MAGIC robots.

2.2.5.2 SKID STEERING

In skid steering systems, the vehicle has independent motors on each side and achieves steering through powering the motors at different speeds. This is shown in Figure 2.10b where \( v_L, v_R, v \) and \( \theta \) represent left wheel velocity, right wheel velocity, body velocity and angle respectively. With this type of steer system, the vehicle has the ability to rotate on the spot by powering one side forward and the other in reverse. Hence, it allows higher manoeuvrability compared with the Ackermann steering system previously mentioned. The manoeuvrability is compromised somewhat by decreased accuracy with localisation due to encoder counts no longer representing just forward movement, but a skid steer system with wheels is a good option for the MAGIC robots because it combines the speed, low cost and construction simplicity of wheels with the ability to traverse rough terrain.

![Figure 2.10: Different types of steering systems](image)

2.3 LOCALISATION

All autonomous vehicles require the ability to determine their location, and it is fundamental to many of the other systems required. Localisation is the ability of an autonomous vehicle to accurately determine the pose of the vehicle. The pose is defined as the position and heading of the vehicle in three dimensions in a fixed reference frame. Localisation requires sensors that measure the pose of the vehicle. It is possible to use only a GPS (Global Positioning System) receiver for this purpose but this is only reliable outdoors and is also affected by areas where the view of the sky is limited by buildings or natural features. Many sensors, like a GPS, are restricted in their operational range or function, so a sufficient range of sensors must be chosen so that the pose can be determined in all environments. The most common approach is to fuse a GPS and an IMU (Inertial Measurement Unit) but many other sensors are possible. When the GPS position is not available (i.e. the GPS signal is too weak), pose is determined from the IMU using dead
reckoning. Dead reckoning is the estimation of a current position based on the previously estimated position and measurements from inertial sensors. Thus, dead reckoning is prone to drift because the accuracy of one position estimate relies on the accuracy of the previous estimate, so any errors accumulate over time. Visual techniques, relying on camera images or LiDAR (Light Detection And Ranging) measurements, have been developed to get around this limitation. These techniques give an absolute measurement of position and are often referred to as Simultaneous Localisation and Mapping (SLAM) because a map of the environment is typically generated concurrently with localisation of the vehicle, using the same sensors. The fusion of the data from multiple sources is typically performed through a Kalman filter because this will filter noise from the measurements and combine multiple measurements of the same element of pose, depending on their accuracy. The Kalman filter is not the only possible choice to combine the sensor data but is the most common choice.

2.3.1 DIFFERENTIAL GPS

Global Positioning System based localisation is very common for UGVs operating in outdoor environments of which they have no prior knowledge. GPS receivers are used to determine position to a good level of accuracy, in an earth fixed co-ordinate system. The system relies on a set of satellites orbiting the earth and the position is determined via triangulation from these satellites. This means that the position can be determined on an absolute scale and not relying on a previous estimate of position. Hence errors do not accumulate over time and any errors present are relatively fixed in space and time. The distance from any one satellite is determined by the length of time it takes a signal to travel at known velocity from the satellite to the receiver. This requires very accurate timing of both the satellite and receiver and so their clocks are calibrated by the GPS network. The clock on a receiver is updated from the satellites and the clock on each satellite is regularly calibrated with a very accurate clock on earth. From the signal received from each satellite, the receiver can determine a sphere of possible solutions. When the distance from multiple satellites is determined, the position of the receiver is then at the intersection of the spheres of solutions from each individual satellite (Kaplan and Hegarty 2006, pp. 25-27). Figure 2.11 is included to demonstrate the plane of solutions that are possible when the distance from two satellites is known. When signals from three satellites are received, a position can be determined to a point in space (Figure 2.12).

While not the principle purpose of the localisation system on a UGV, some knowledge of the vehicle's velocity is useful for control and autonomous applications. GPS receivers are capable of measuring velocity, but typically with less accuracy than their calculations of position. The simplest method to use is an approximate derivative of the positioning solution. This is not useful for a UGV, as it does not add any new measurements to the system, only a different representation of the same measurements. Another method to calculate velocity is to look at the Doppler shift of a GPS signal. The Doppler shift is the change in the frequency of the signal from the satellite caused by the relative velocity of the satellite and receiver. As the nominal frequency of the signal is known and the position and velocity of the satellite can be estimated, the velocity of the receiver can also be estimated (Kaplan and Hegarty 2006, p. 58). This technique is common in high end GPS receivers but rare in low end receivers due to the extra complexity in implementing the system.

There are several possible sources of error that reduce the accuracy of a position determined by a GPS receiver. The timing of signals used for determining distance travelled is critical but is also very carefully managed by the GPS system, meaning this error is relatively small. The three common sources of error are radio frequency interference, multipath errors and ionisation errors. Fortunately the primary (L1) frequency band used for the GPS system is reserved for satellite navigations
signals, meaning accidental interference is relatively rare. The secondary (L2 and L5) frequency bands used in dual band GPS receivers suffer from interference much more but these are only required to augment a L1 signal. The 1200-1500 MHz frequencies used for L2 and L5 are also shared with mobile phones, radio systems, radar systems and other aeronautical navigation systems making interference a very common problem. Deliberate interference (jamming or spoofing) on the L1 frequency is possible but rare, as there are encrypted GPS signals available for high security applications (Kaplan and Hegarty 2006, p. 244). Radio interference is not a common problem for GPS receivers on UGVs. Multipath errors are much more problematic. They occur when a signal reaches a GPS receiver by a reflection, from an urban or natural feature, the position calculated is incorrect as the length of the path between the satellite and receiver has been altered (Figure 2.13). This problem is very likely to occur in a mixed outdoor and indoor environment that many UGVs operate within. Ionospheric scintillation is another problem that affects GPS signals. The ionosphere is the region of the atmosphere that exists approximately 50km above the Earth’s surface. Typically
it causes a small delay in the signals but fluctuations in the number and position of free electrons within the ionosphere can affect the length of the delay and occasionally prevent GPS signals reaching the ground (Kaplan and Hegarty 2006, p. 295). These numerous errors mean that the GPS position can only be guaranteed to within 10m 95% of the time (Kaplan and Hegarty 2006, p. 379).

Figure 2.13: GPS multipath leads to large errors in a GPS solution. The straight line signal from satellite 2 is blocked while a reflection reaches the GPS receiver. (Cayzac 2006)

As the range of uses of GPS technology has developed over time, a requirement for increases in the accuracy of the network has developed. This allows the technology to be used for tasks where position information must be accurate to within 10 metres. An example of this is in surveying but as only stationary positions generally need to be found, more accuracy is achieved by averaging the measurements at one position over time (Novatel n.d., p. 73). This technique is only useful for slow moving or stationary situations and in the case of a UGV this technique is not viable. Instead differential GPS was developed.

Differential GPS is a system capable of accurately determining a receiver’s position by applying a correction to the position found using triangulation. Applying corrections in this way is only viable when errors are fairly constant within a large area (>10km square) (Kaplan and Hegarty 2006, p. 380). GPS errors are relatively fixed because many errors are caused by atmospheric disturbance and radio interference, which will not vary greatly over a large area (Kaplan and Hegarty 2006, pp 306, 381). There are several methods used to achieve this correction. The simplest method uses post processing. All data is recorded using a normal GPS unit. Data is then sent to a central system that applies a correction to that location using the time the measurement was taken. This is not suited to real time systems as there is a significant delay in determining position. The accuracy of the reading
still depends on the length of time over which the measurement was taken. Another reasonably straightforward method involves corrections that are readily available over a wide area and are referred to as Wide Area Augmentation Services (WAAS). These services can use land based base stations, whose positions are known very accurately, to calculate the corrections in real time and then send them over a publicly available radio frequency. Typically, a single frequency on the FM radio network is used but it depends on the implementation. These are often installed by shipping ports and airports, to aid in transport navigation when entering and exiting these facilities. WAAS is also available from a satellite based system but no such system has been developed in Australia. Examples include CDGPS in Canada and EGNOS in Europe. The most accurate and adaptable dGPS network is for the end user to set up base stations and transmit corrections. This works just like a WAAS but requires the set up of a base station close to the area that is being used. Typically, corrections are sent using a readily available RF modem (Novatel 2010, p. 55). The last option is the only one viable for most UGVs.

2.3.2 OUTDOOR LOCALISATION

The most common method for high accuracy vehicle localisation in an outdoor setting is the combined use of an IMU and GPS. This method is used because it provides both heading and position information at a rapid rate (Shin 2001, p. 1) and so is used extensively for localisation. It is suited to outdoor use, where the GPS can communicate with a sufficient number of satellites to be effective. Even though the GPS measurements are more accurate than the position calculated from the IMU measurements, an IMU is needed because it is much faster and operates independently of the external inputs (satellites) required by the GPS. The capabilities of the IMU allow the pose data to be updated faster and during times of short GPS outage that are especially likely to occur in an urban environments. The GPS signal will be affected by buildings blocking the signal and multiple signal paths reducing the overall accuracy. Multiple signal paths are created by reflections from buildings and other objects, so that the GPS unit will place satellites incorrectly and can receive multiple signals from the same satellite. This creates errors when the GPS is determining pose. Hence, the IMU helps to eliminate the errors from the GPS unit and together form a relatively robust system for outdoor localisation.

IMU/GPS integration was the most common approach for localisation in the DARPA grand challenge, where a proven technology could be used for the requirements of the challenge. One example is Team CIMAR's NaviGATOR which used an off the shelf unit manufactured by Smiths Aerospace (Crane et al. 2006, p. 607). The Smiths 'North-finding Module' used by this team consisted of a GPS, an IMU and an external odometer. This unit processed the sensor data using a Kalman filter to give a best estimate result. This unit was deemed to be sufficiently accurate for outdoor use and in the event of a loss of a GPS signal, could operate with sufficient accuracy for a distance of a few hundred metres. The vehicle was stopped as a precaution after this distance as the accuracy of its pose was found to be poor by that point. The Smiths module is 'Tactical Grade' and so is too costly for use in many scenarios. It is also limited in that the Kalman filter is not controlled by the end user, so it cannot be used with sensors other than the included GPS and IMU. There has been much work in recent times to improve systems (Shin 2001, Kim et al. 2004) using a low cost IMU and a GPS that can be implemented by the end user, on hardware that will also be used for other functions on an autonomous vehicle. This reduces the cost greatly and makes the result much more adaptable to the needs of the end user. The combination of a GPS and IMU through the use of a Kalman filter on board an autonomous vehicle is a proven and reliable technique that is used on the MAGIC UGVs, enabling reliable localisation of the UGVs, including short distances travelled when GPS data is not available.
2.3 LOCALISATION

2.3.2.1 KALMAN FILTER

The Kalman filter is a method for filtering data to give an estimate of the actual value of a state that is being measured and minimises the variance of the estimate. Its existence was first published in 1960, in a paper by R. E. Kalman (Welch and Bishop 2006, p. 1) but has become very widely used in tracking and estimation due to its simplicity, optimality and robustness (Julier and Uhlmann n.d., p. 1). The premise of the filter is that measurement noise is Gaussian and that the variance of the measurement noise is known. Sensors do not always have Gaussian noise but the filter has been proven to be reasonably successful despite this. The filter uses an estimate of the variance of a measurement to give a weighting to each measurement. This means that if there are two measurements of the same state, then the measurement with the higher weighting will be used more than the measurement with the lower weighting. Also measurements that have a very high variance will be used less than those with a low variance. Measurements that have a lower weighting will be slower to update to actual changes, but the effect of the noise in the measurement should be minimised. The measurements with a small variance will have much less noise in them, so the estimate will update faster to actual changes. This is represented graphically in Figure 2.14 where -0.38 V is the actual value, the measurements are marked by crosses and the estimate is the continuous line. Note how varying the estimate of variance changes the susceptibility to noise of the estimate.

Numerous types of Kalman filters have been developed to deal with specific scenarios and situations that exist. The most basic Kalman filter is quite simple to understand but works only in the continuous time domain. Given that Kalman filters are normally implemented using sensors that measure at a finite rate, a Kalman filter is needed that works in the discrete time domain. The discrete Kalman filter is a simple extension of the basic Kalman filter and allows it to be implemented on a computer. The discrete Kalman filter relies on the dynamics of the system and the sensors performance to be described linearly, making this somewhat restricted in possible real world implementations (Welch and Bishop 2006, p. 7). To get around this problem, linearisation using a Jacobian is used for the Extended Kalman filter and offers significantly better performance than the discrete Kalman filter for most real world systems. Recently, the Unscented Kalman filter has also been developed, which does not rely on a linearisation of the system dynamics and sensor performance and so achieves slightly better results than the Extended Kalman Filter (Julier & Uhlmann n.d., p. 11). Both types of filters rely on knowledge of the variances of the measurements and states used in the system. However, the tuning of these parameters to achieve best performance is typically a significant complication.

The complexity of each type of filter is related to the difficulty of successfully implementing the filter. The computational requirements of an Unscented Kalman Filter are less than for an Extended Kalman Filter and the accuracy greater in many cases (Orderud n.d., p. 4) but their implementation is also more complicated. The Unscented Kalman filter relies on an estimation of the distribution of the error characteristics of each state but this can be very hard to determine and incorrect estimation of the distribution leads to instability. The Extended Kalman filter is somewhat more robust than an Unscented Kalman filter but is limited by the assumptions used while linearising the system model. The linearisation is based on local linearity, meaning that changes over a very short time period are assumed linear. If this condition is not true, the Extended Kalman filter becomes unstable (Julier & Uhlmann n.d., p. 2). As a very simple system model is used for the localisation problem, the linearisation should always hold true for filters used for localisation.

Kalman filters are not the only method for fusing and reducing noise in a measurement. Other options include the particle filter. Thrun et al (2000) propose the use of a particle filter for localisation of mobile robots. The particle filter is more capable of localising the robot in a global reference frame and is utilised specifically for situations where the robot travels between locations that do
not depend on each other. The particle filter also has much greater computational requirements than a Kalman filter, meaning that they may not be realisable for real time applications (Orderud n.d., p. 6). Particle filters typically use Markov localisation, a technique that is specifically suited to areas where the map is known prior to exploration. Markov techniques use measurement of features that can be identified on the map and determines the vehicles position relative to these.
It is still a probabilistic technique because it uses a weighting depending on the reliability of a measurement. Techniques other than the Kalman filter are available for fusing data used in robot localisation but they typically relate to some special requirements and as such the Kalman filter is commonly considered the best option available for general purpose and subsequently will be used in the MAGIC project to fuse data for the purpose of localisation.

2.3.3 Indoor Localisation

The method of utilising a filtered system combining GPS and IMU data (described in Section 2.3.2), although relatively robust outdoors is very inaccurate indoors, because GPS signals do not penetrate buildings and other similar structures. Thus, this system would be reduced to a system dependent on dead reckoning calculations from the IMU. It is further explained in Section 2.3.2 that this is prone to significant drift and over time causes the position estimate to be very inaccurate. This would be catastrophic for the overall system. A different system is required that will work as a substitute for the GPS measurement. A very popular method is called Simultaneous Localisation and Mapping (SLAM) and uses features of the environment as landmarks that can be reobserved and corrected against.

SLAM is a popular technique used for autonomous control of a robot or group of robots. It is named after the identically titled SLAM problem that is well recognised and agreed upon by researchers in the field of robotics. Montemerlo and Thrun (2007) define SLAM as the “problem of a mobile robot moving through an environment of which no map is available a priori. The robot makes relative observations of its ego-motion and of features in its environment, both corrupted by noise. The goal of SLAM is considered by many to be a key prerequisite to truly autonomous robots.” Thus the function of SLAM is to localize the position of the robot, as well as creating a virtual map of the surroundings. This map is then used by the robot to plan and traverse an appropriate route based on predefined route strategy parameters. SLAM was coined by Durrant-Whyte and Leonard in their seminal work based on ideas first posited by Smith, Self and Cheeseman (Riisgaard & Blas 2005). A pictorial representation of SLAM is given in Figure 2.15.

Other conventional methods of localisation, such as the use of a Global Positioning System (GPS), differential GPS (dGPS), wheel encoders, or motor input control give results and data as absolute values, such as the coordinates from a GPS, or alternatively relative to the current position of the robot. SLAM differs from these conventional methods as it is primarily a correction technique that uses an estimate for the position and re-evaluates this estimate based on the new position of previously encountered local landmarks. For this reason, SLAM is used primarily for localisation indoors, where there is an abundance of landmarks and also very importantly, when powerful systems such as GPS are ineffectual or indeed not functional, due to interference or objects such as the roof obstructing communication (Montemerlo & Thrun 2007). However, for this same reason, SLAM, is much less effective in areas with few local landmarks, such as in barren landscapes.

The mapping aspect of SLAM is undertaken through the use of software that reads data from sensors such as lasers, sonar, or vision sensors. Many different maps can be drawn ranging in complexity, however; each is based on features, such as objects and walls, identified for the SLAM process. More complex maps can include layered two-dimensional grids, so-called 2.5-dimensional grids or in more complex systems three-dimensional maps. Figure 2.16a shows how SLAM is used to correct maps that would be otherwise drawn by the robot. In Figure 2.16a, uncertainty in the robots motion has skewed its estimate of its current position and it sees the wall oriented differently from different positions. Figure 2.16b shows the same scenario, when SLAM is involved. Having seen the wall again, the robot undertakes landmark association, effectively realising that it is seeing the wall repeatedly. Hence, the robot can correct its estimate for its position, meaning that it can
map the wall more accurately. Without correction, the robot is unaware that several boundaries identified are in fact the same feature.

Figure 2.16: SLAM mapping (Montemerlo & Thrun 2007, p. 3)

The mapping has many uses based on the circumstances and application of the SLAM. First, the mapping gives an indication of the topology and geography of the area that may be as yet unexplored. This allows the robot to function in unknown areas. Additionally, the mapping is an essential part of any route planning to be calculated remotely or autonomously by the robot. Autonomous navigation is a natural extension of the two functions of self-localisation and local area mapping.

Since SLAM is a recent idea, progress has occurred in a few different directions, and it is constantly advancing due to improvements in the accuracy and reliability of sensors. There have been many popular versions of SLAM and each of these has also had minor variations based on the research of different people. SLAM is usually achieved with the use of lasers, sonar or visual sensors, such as cameras. Each of these has its own advantages and weaknesses, and more importantly many of the processes are different for the various types of sensor.
The term SLAM has over time developed into a word meaning not just the problem, but also of processes that attempt to solve this problem. In particular, there are a lot of variations of SLAM that have been published and researched. Many have subsequently been improved upon or edited. Therefore, the earlier types of SLAM have become quite famous, many carrying names that are widely recognised such as EKF-SLAM (Riisgaard & Blas 2005), UKF-SLAM (Andrade-Cetto & Sanfeliu 2006), FastSLAM (Montemerlo & Thrun 2007), 3D-SLAM (Nüchter 2009) and Rat-SLAM (Milford 2007).

Literature available to robotics and SLAM researchers predominantly focuses on improvements in the various stages of SLAM. These stages are landmark extraction, data association, filtering and navigation. The following sections will review the literature within the context of these four stages primarily considering SLAM as a means of localisation.

2.3.3.1 LANDMARK EXTRACTION

Although SLAM relies very heavily on the ability to find and identify useful landmarks, most recent publications do not concentrate on the actual landmark extraction, but more on the processes after landmarks have been established. Riisgaard and Blas (2005), from the Massachusetts Institute of Technology, discussed three types of landmark extraction techniques and gave examples and further discussion on two of them. In particular, they found that looking for extrema in the laser data (Spike Landmark Extraction) or straight lines (RANSAC) is sufficient for most purposes that do not require a comprehensive map of the surroundings and are more interested in the process of localisation. Spike landmark extraction searches for points that are in the foreground of a scan. As an example, Figure 2.17 shows a distance-sensing laser finder scanning an area containing a table. The points towards the bottom of the picture are points on the legs of the table that are in the foreground of the scan.

Riisgaard and Blas (2005) also mentioned scan-matching as a more complicated and computationally expensive process that takes into account all of the data. Scan matching involves comparing subsequent snapshot photos of the surroundings and using high-level feature recognition to associate certain objects in both photos, thus allowing the system to calculate the movement of the robot. Montemerlo and Thrun (2007), stated that most SLAM maps are "typically represented as point features, or landmarks", although "higher order geometric features, such as lines have also been used".

Figure 2.17: Spike Landmarks. The red dots are table legs extracted as landmarks (Riisgaard & Blas 2005, p. 20)
Nüchter (2009), while attempting to utilise multiple lasers simultaneously for the one robot, states that his algorithm utilises a scan-matching technique where all of the data from a scan is used as a single picture and compared to other readings. This he uses successfully to implement a 3-dimensional version of SLAM in the RoboCup real rescue league in Lisbon in 2004 (Nüchter 2009).

Riisgaard and Blas (2005) indicate that for an indoor area with lots of straight lines such as walls, utilizing a stochastic method such as Random Sample Consensus (RANSAC) in order to identify straight lines is an effective approach. Figure 2.18 shows how this can be utilised. Zuliani (2009) considers RANSAC from a theoretical angle and finds that RANSAC is a very robust technique against outliers (and hence excessive noise) in a set of data. Further, Zuliani (2009) shows that it is robust in data sets where outliers constitute up to half of all data points.

Figure 2.18: The RANSAC algorithm finds the main lines in a laser scan. The green lines are the best fit lines representing landmarks. Red points are point representations of the landmarks. In the centre of the picture represented by the small triangle is a person and it has not been detected as RANSAC is robust against people.

2.3.3.2 DATA ASSOCIATION

Perhaps more important than the concept of what landmarks to use, is how to correctly identify a previously seen landmark or to realize a new landmark has been found. Not associating a landmark to another landmark representing the same object or alternatively associating the landmarks representing different objects can lead to the propagation of major uncertainties in the estimation of pose technique, rendering the SLAM algorithm useless (Andrade-Cetto & Sanfeliu 2009).

Montemerlo and Thrun (2007) explain that a maximum-likelihood rule is usually used for data association, where each known landmark is associated a likelihood of generating a newly found landmark. If the likelihood conferred to a landmark is below a certain threshold it is considered different and thus a landmark is added to the database if all the likelihoods are lower than this threshold. The easiest ways to assign likelihoods to landmarks is to consider the distances between landmarks (Risgaard & Blas 2005). In particular, the Euclidean distance between landmarks can be used. Alternatively, Risgaard and Blas (2005) introduce the concept of the Mahalanobis distance, which uses innovation and innovation covariance defined in an Extended Kalman Filter application of SLAM. Other methods that are used are Bayesian Inference (Andrade-Cetto & Sanfeliu 2009) as
well as Local Map Sequencing, Joint Compatibility Branch and Bound, Combined Constraint Data Association, Iterative Closest Point and Multiple Hypothesis Tracking (Montemerlo & Thrun 2007). These latter methods are complex multi-association techniques, which are unlikely to be necessary for the scale and constraints of most projects (Riisgaard & Blas 2005).

2.3.3 FILTERING

An integral part of a SLAM filter is the filtering of raw data as well as of the landmark results. Filtering is important in SLAM in order to minimise the impact that noise has on readings. The most common types are extended Kalman filters, unscented Kalman filters or particle filters.

Smith, Self and Cheeseman (1986) when first introducing the problem of simultaneous localisation and mapping (not yet called SLAM then) introduced a simple Kalman filter approach. This work is often quoted as the first work in the area of SLAM (Riisgaard & Blas 2005). Since then this approach has been further developed, with the defacto standard being EKF-SLAM (Montemerlo & Thrun 2007), which utilizes an extended Kalman filter in place of the simple Kalman Filter giving it more robustness and relaxing a few linearity assumptions (Nüchter 2009). An example EKF SLAM structure is given in Figure 2.19.

EKF-SLAM was first explained and implemented by Durrant-Whyte and Bailey (2002). Figure 2.20 shows an example simulation of an EKF-SLAM approach. In the same paper, they considered an extension of this technique which was to utilize a Rao-Blackwell particle filter. This led to research in this area sparking the creation of FastSLAM 1.0 and FastSLAM 2.0 a very popular algorithm that uses a Rao-Blackwell particle filter for the tracking of multiple hypotheses, creating a sort of chain of tiny filters (Montemerlo & Thrun 2007). Figure 2.21 from Durrant-Whyte and Bailey (2006, pg. 6) exhibits a “single realisation of robot trajectory in the FastSLAM algorithm. The ellipsoids show the proposal distribution for each update stage, from which a robot pose is sampled and, assuming this pose is perfect, the observed landmarks are updated. Thus, the map for a single particle is governed by the accuracy of the trajectory. Many such trajectories provide a probabilistic model of robot location”. Wedema (2009) compared EKF-SLAM to FastSLAM and found that FastSLAM is
much more robust against incorrect data associations, whereas EKF-SLAM was prone to associating data points incorrectly; however, it is a more consistent algorithm.

As previously mentioned, many variations have been employed, researched and tested over the years. In particular, many researchers have opted to utilize the more recent unscented transformation of the unscented Kalman filter (Andrade-Cetto & Sanfeliu 2006). This has resulted in the creation of the UKF-SLAM. Although it is not noted to be remarkably more accurate for localisation. For the purpose of this project, the SLAM process is being utilised only for localisation and not navigation. Navigation for MAGIC 2010 has been undertaken through the use of a separate path planning and path navigation algorithm. The literature behind this is reviewed in Section 2.5. However, in its purest form SLAM incorporates the process of navigation using the maps or landmark databases created. Of particular note is RatSLAM. Milford (2007) uses a model based on the rat hippocampus to come up with effective navigation strategies, which are utilised for the navigation in RatSLAM. Rather than concentrate on reactive behaviour that is a common trait amongst modern robots, this hippocampus model technique is used to predict future outcomes of behaviour and attempts to use pre-emptive navigation. This project, however, treats navigation as a separate problem.

2.3.3.4 NAVIGATION

For the purpose of this project, the SLAM process is being utilised only for localisation and not navigation. Navigation for MAGIC 2010 has been undertaken through the use of a separate path planning and path navigation algorithm. The literature behind this is reviewed in Section 2.5. However, in its purest form SLAM incorporates the process of navigation using the maps or landmark databases created. Of particular note is RatSLAM. Milford (2007) uses a model based on the rat hippocampus to come up with effective navigation strategies, which are utilised for the navigation in RatSLAM. Rather than concentrate on reactive behaviour that is a common trait amongst modern robots, this hippocampus model technique is used to predict future outcomes of behaviour and attempts to use pre-emptive navigation. This project, however, treats navigation as a separate problem.

2.4 MAPPING

The purpose of map creation is to use information extracted from sensors to create a digital representation of the surrounding environment which can be used for obstacle avoidance, navigation, path planning and strategy creation to allow a robot to achieve certain objectives. To complete the objectives of the MAGIC 2010 competition, a cooperative autonomous unmanned ground vehicle (UGV) team is required to create a complete map of the MAGIC environment, as well as identify and neutralise static and mobile objects of interest (OOI). Hence, to satisfy these objectives both
physical and conceptual maps are necessary. Physical maps primarily contain information about
the physical environment extracted from sensor inputs, such as elevation and UGV visibility, and
are generally used for path planning and navigation within the physical environment. In contrast,
conceptual maps derive environmental information from the physical maps and merge this with
additional information such as the position of the OOI, safe and unsafe regions, explored and
unexplored regions, and other important features to allow for navigation and strategy creation.

2.4.1 PHYSICAL MAPPING BACKGROUND

Physical maps are created using data from robot sensors and provide the foundation for robot
navigation and localisation. Early approaches to robot mapping and navigation used stereo vision
cameras to identify lines and vertices of objects for obstacle recognition, and utilised basic control
algorithms to cause the robot to avoid these obstacles. The problem with this method was that snap
decisions about the existence of obstacles were made based on the immediate incoming data. Hence
even with filtering, noisy data usually caused the robot to behave erratically and fail on a regular
basis (Martin and Moravec, 1996, p. 7). Robot failures induced by noise-affected sensor data led to
probabilistic mapping methods where the existence of obstacles was not determined immediately,
but by repeated event observations. By using repeated event observations to determine obstacle
existence the impact that noise-affected data had on the system was minimised.

The first inception of probabilistic mapping was the occupancy, or evidence, grid created at Carnegie
Mellon University, Pennsylvania in 1983 to combat the problem of dealing with ambiguous data
from inexpensive ultrasonic sensors (Martin and Moravec, 1996, p. 7). The occupancy grid is basically
a two or three dimensional grid which stores data regarding the probability of the existence of
obstacles in the environment along with their position relative to the sensor. The performance of
the occupancy grid mapping method compared to the prior methods using boundary identification,
referred to in the following extraction as the Cart program, can be summarised as follows: ‘After
ten years of development the Cart program was still unreliable at crossing a room, but the very first
grid program succeeded every time’ (Elfes 1987, cited in Martin and Morevac, 1996, p. 7). Table 2.1
details the characteristics of various probabilistic based mapping methods. Of particular interest is the occupancy grid. As strong convergence allows it to deal with sensor noise of any probabilistic distribution, it can handle raw data, and a limited representation of a dynamic environment means the generating algorithm can cope with the existence of mobile OOI in the MAGIC environment. The Dogma, or dynamic occupancy grid mapping algorithm is also an appealing solution to the MAGIC mapping problem, however Dogma’s utilisation of the expectation maximisation (EM) algorithm to establish correspondence between the occupancy grid framework and the dynamic environment leads to a weak convergence, meaning that sensor noise will affect the produced map.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Kalman</th>
<th>Occupancy Grids</th>
<th>Dogma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertainty</td>
<td>posterior poses and map</td>
<td>posterior map</td>
<td>posterior map</td>
</tr>
<tr>
<td>Convergence</td>
<td>strong</td>
<td>strong</td>
<td>weak</td>
</tr>
<tr>
<td>Local</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Incremental</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Requires Poses</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Sensor Noise</td>
<td>Gaussian</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>Can map cycles</td>
<td>yes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Map dimensionality</td>
<td>$\sim 10^3$</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>Correspondence</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Handles raw data</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Dynamic Environments</td>
<td>limited</td>
<td>limited</td>
<td>yes</td>
</tr>
</tbody>
</table>

The occupancy grid works by utilising distance information regarding the position of obstacles in the environment extracted from a sensor and a probability formula to filter the data and update the grid. The distance information obtained from the sensors combined with a known sensor orientation allows the position of obstacles in the environment to be determined. However in addition to the determination of obstacle position, the existence of vacant space can be determined using ray tracing algorithms to determine the squares in the occupancy grid between the sensor and the obstacle which are empty. Hence a complete representation of the environment can be created using simple distance sensors. An example of an occupancy grid and the corresponding real world environment is displayed in Figure 2.2 which compares an architectural drawing of a building and the corresponding occupancy grid generated map created using a laser rangefinder. It should be noted that the occupancy grid generated map is actually more accurate than the architectural blueprint in several locations.

2.4.1.1 Occupancy Grid Mapping

The occupancy grid works by utilising distance information regarding the position of obstacles in the environment extracted from a sensor and a probability formula to filter the data and update the grid. The distance information obtained from the sensors combined with a known sensor orientation allows the position of obstacles in the environment to be determined. However in addition to the determination of obstacle position, the existence of vacant space can be determined using ray tracing algorithms to determine the squares in the occupancy grid between the sensor and the obstacle which are empty. Hence a complete representation of the environment can be created using simple distance sensors. An example of an occupancy grid and the corresponding real world environment is displayed in Figure 2.2 which compares an architectural drawing of a building and the corresponding occupancy grid generated map created using a laser rangefinder. It should be noted that the occupancy grid generated map is actually more accurate than the architectural blueprint in several locations.
2.4.1.2 BAYESIAN INFERENCE IN PROBABILISTIC MAPPING

Probability algorithms are the most popular method of map updating currently used as their ability to efficiently filter sensor noise allows for accurate and robust map production (Thrun 2002). The most commonly used method of information processing within mapping algorithms is Bayes Theorem, commonly referred to as Bayesian Inference, where the probability of the existence of an object at a position is based on sensor information and the total events that have been previously observed at that position. Bayes Theorem is displayed in Equation 2.1 below where $P(p_i)$ is the probability of an object existing at position $i$, $P(\neg p_i)$ is the probability of an object not existing at position $i$, and $P(s)$ is the probability of a sensor reading indicating that an object exists.

$$P(p_i | s) = \frac{P(s | p_i) \times P(p_i)}{P(s | p_i) \times P(p_i) + P(s | \neg p_i) \times P(\neg p_i)} \quad (2.1)$$

As the value $P(s | \neg p_i)$ is usually unknown it is common practice to replace the denominator with the value $1/\alpha$ (Hackbarth, n.d, p. 580) where $\alpha$ is chosen between 0 and 1 depending on the willingness to assume that event $i$ has occurred before any information regarding it can be obtained. In terms of an occupancy grid framework Bayes Theorem can be rewritten as

$$P(C_{i+1}) = \frac{\alpha P(C_i)}{\alpha P(C_i) + (1-\alpha)(1-P(C_i))} \quad (2.2)$$

for cells where objects are detected, and conversely

$$P(C_{i+1}) = \frac{(1-\alpha)P(C_i)}{(1-\alpha)P(C_i) + (\alpha)(1-P(C_i))} \quad (2.3)$$

for cells which are determined to be vacant.
Early probabilistic-based occupancy grids contained two values in each cell, the probability that the cell was occupied and the probability that the cell was empty. The probabilities of cell occupancy and vacancy were each assigned a different \( \alpha \) value to account for cell overlap in sensor readings. However it has since been demonstrated that one probability value per cell is sufficient (Martin & Morevac 1996).

### 2.4.2 Conceptual Mapping Background

Conceptual maps are a part of mapping that is used to depict what the robot sees in the surroundings. Conceptual maps not only provide valuable data to the UGVs, but these are also designed in such a way that they can easily be understood by humans when displayed on a Human Machine Interface (HMI).

Navigational ability is a pre-requisite for the successful functioning of any autonomous robot. Navigation mainly comprises of mapping, localisation and path planning (Stachniss et al. 2009 p. 3). Mapping involves the robot representing what it sees via its sensors, usually in the form of arrays of data. Localisation is concerned with determining the pose of the robot, usually with respect to a fixed reference point. Finally, path planning deals with calculating and plotting trajectories that a robot should follow in order to get to a destination. These three elements cannot be developed independently, but rather in a coupled fashion due to their interdependence.

Figure 2.23 illustrates the overlapping nature of all three elements to navigation. This interdependent nature of the three elements is somewhat apparent. In order to accurately generate a map about its surroundings, a robot needs to know its pose with accuracy; and in order to determine its pose the robot requires a certain level of information on the map of its surroundings. Similarly, in order to calculate and follow a trajectory, the robot requires information on both its pose and the map of the surroundings. Hence, it is obvious that the three elements of navigation are coupled.

#### 2.4.2.1 Mapping Methods

The main mapping method is interpreting data from the sensors of a robot and storing that data in the form of arrays. This usually involves using Bayesian inference or adhering to specific designer created criteria. For instance, a conceptual map depicting regions that have been explored may be done by reading data from a sensor such as a LiDAR and interpreted using a specific criterion. This may be reading probability values and checking against a threshold to determine occupation probability. If occupation probability is high or low, it can be said with confidence that the area under consideration has been explored. Similar constraints and criteria are usually used in interpreting either raw data from sensors or from other maps in order to generate conceptual maps. The conceptual maps in this project are constructed in this way.

### 2.5 Autonomous Control

Autonomous control is a growing field of robotics and is rapidly advancing, such that many autonomous robots are finding their way into everyday living (Beetz 2002) including vacuum cleaner robots (iRobot 2010) and automatic parking vehicles (Hanlon 2006). None of these robots would be very successful if the entirety of their function was to do a series of finely tuned routine actions repeatedly before deactivating. The Roomba and automatic parking vehicles are programmed to cope with varying environments, particularly due to human involvement, such as a chair located in a different position impeding the vacuum cleaner robot or two cars leaving less clearance
than normal for an autonomous parking vehicle. Thus, autonomous robots require capabilities that allow them to adapt to the situation and not require manual configuration. The MAGIC Unmanned Ground Vehicles (UGVs) will require some autonomous capabilities. More formally, Fahimi (2009) asserts that an autonomous robot is one equipped with methods to automatically overcome potentially varying environmental constraints while it is performing its desired task. This project involves UGVs, so the discussion of autonomous control will be limited to autonomous control of vehicles. Both single vehicle autonomous control and multiple vehicle autonomous control will be discussed.

### 2.5.1 Single Vehicle Autonomous Control

The purpose of single vehicle autonomous control is the successful travel of the vehicle from a specific origin to a specific destination. This must be done regardless of whether or not the environment has been explored and without any manual interference. In this discussion, we shall assume the vehicle is able to sense and identify objects as well as accurately localise itself. Thus, the vehicle requires a strategy that will allow it to traverse from its origin to its destination. For practicality purposes, such a strategy must minimise the potential for damage due to obstacles or moving objects. Other desirable but, not always feasible characteristics of such a strategy include the minimisation of computational effort, power consumption, path length as well as path duration.

The control system of an autonomous robot is the ‘brain’ of the robot, and dictates all the actions the robot must perform. In simple terms, the responsibilities of the control system include interpreting required information from the maps and other sensors, determining the necessary course of action based on pre-programmed algorithms for a variety of scenarios, telling the robot where to go and how to get there, and coordination of the various subsystems in the robot. An important component of the control system of any autonomous robot is vehicle navigation. A well established methodology for solving the vehicle navigation problem is by viewing it as a combination of four subsystems. These subsystems are world perception, path planning, path generation and path tracking (Shin & Singh 1990), as shown in Figure 2.24.
2.5.1.1 WORLD PERCEPTION

World perception involves gathering information about the environment in which the robot is in, and typically storing the data in the form of a map. This has been achieved in a variety of ways. In order to create maps the three most common sensory input devices used are laser range finders, cameras and radar. With these, the maps created are occupancy grids in either two or three dimensions, feature maps or topology maps. Each of these maps has their own advantages. In particular, the occupancy grids are desirable for presentation and viewing purposes. Feature maps are useful for strategies such as Simultaneous Localisation and Slam (SLAM), but also for object recognition. On the other hand, topological maps are sufficient for navigation and object avoidance techniques (Milford & Wyeth 2010).

2.5.1.2 PATH PLANNING

Path planning typically generates a set of waypoints or destinations that the robot must reach, based on information from world perception (Shin and Singh 1990). There are numerous path planning methods. Common forms of path planning include the road-map method, cell decomposition method and the potential field method.

The road-map method has many varieties, the most recent of which is the probabilistic road-map planner (PRM). It computes the path for a robot in an environment of stationary objects without collisions. This method also applies to robots with a large number of degrees of freedom (Geraerts & Overmars 2002). The PRM is made up of two phases: a learning phase and a query phase (Zhang, Ferrari & Qian 2009).

During the learning phase, a graph (the road-map) is constructed. The graph is made up of nodes representing collision-free configurations and edges representing collision-free paths. These are constructed according to the following two iterated steps:
(i) Select a random robot configuration. Test for collisions. Repeat until the configuration selected has no collisions.

(ii) Using a fast local planner, attempt to connect the former configuration to the road-map. (Geraerts & Overmars 2002)

The query phase is used to find a path from the original configuration to the final configuration, by connecting these configurations to the road-map and searching the road-map for a sequence of local paths that links these nodes.

Kavraki (2010) adds that these two phases do not have to occur separately and that they may be interwoven. This allows the size of the road map to adapt to difficulties encountered by the query phase. Because, the PRM has generally very small query times, this method is suitable for many-degree-of-freedom robots.

The cell decomposition method also relies on rules that are derived using the geometry of the obstacle field. Many problems such as motion planning for a number of circular or rectangular objects bounded by walls and motion planning for multiple robots have been solved using the geometrical methods. These methods have also been extended to the case of moving obstacles (Fahimi 2009).

In the cell decomposition method, the workspace $W$ is modelled in two dimensions utilising a quad tree data structure with all obstacles $B_i$ added and represented as polygons. A quad tree is a special version of a more general (mathematical) tree type, the $2^n$ tree where $n = 2$. This type of tree is a very recent development for collision avoidance and motion planning. A $2^n$ tree is a tree, where each non-leaf node has exactly $2^n$ children. Each node is used to represent cells of the workspace and contains a list of obstacles found in the cell. Thus, each cell can then be classified as one of empty, full or mixed. It is empty if it contains no obstacles, mixed if it contains at least one obstacle smaller than the cell and full in all other cases. The method then finds a connected path among the empty cells to the destination (Katevas, Tzafestas & Pnevmatikatos 1998).

In the potential field method, an artificial potential field is assigned to the area where a robot works. The obstacles in the area are assigned a repulsive potential while the goal position of the robot is described by an attractive potential. Then, the path of the robots is calculated by using the gradient of the total artificial potential. Different mathematical definitions have been used for defining the artificial potential fields and different strategies have been introduced for using the gradient of the total potential to find a path for the robot (Fahimi 2009). Fahimi (2009) provides the following algorithm for the potential field method.

(i) The uniform flow strength, the goal strength, and the safety ratio are selected. It is recommended that the goal strength should be much larger than the uniform flow strength for a more effective potential field.

(ii) The start position $x_s = (x_{s1}, x_{s2})$ and the goal position $x_g = (x_{g1}, x_{g2})$ are defined.

(iii) The obstacle line segment’s parameters for line obstacle $i$ are determined. Extra care must be taken in defining the reference point $(x_{0i1}, x_{0i1})$ and $\theta_i$ such that the normal unit vector $n_i$ points to the outside of the obstacle. If this condition is not met, the generated paths may interfere with the obstacles.

(iv) The direction of the uniform flow is obtained from

$$\alpha = \arctan \frac{x_{g2} - x_{s2}}{x_{g1} - x_{s1}}$$

(2.4)
(v) The elements $P_{ij}$ are calculated using

$$P_{ij} = \begin{cases} 
1/2, & \text{if } i = j \\
i_{ij}/2\pi, & \text{if } i \neq j 
\end{cases}$$

(2.5)

and $W_i$ from

$$W_i = U_i \frac{\partial}{\partial n_i} (x_{ci1} \cos \alpha + x_{ci2} \sin \alpha) - \frac{\lambda g}{2\pi \partial n_i} \ln R_{gi}, \quad i = 1, \ldots, m$$

(2.6)

(vi) The parameter $a_{\text{max}}$ is evaluated using

$$a_{\text{max}} = \frac{\lambda g + \sum_{i=1}^{m} L_i \sum_{j=1}^{m} J_{ij} W_j}{\sum_{i=1}^{m} L_i \sum_{j=1}^{m} J_{ij} L_j}$$

(2.7)

and $a$ is found from

$$r_a = \frac{a}{a_{\text{max}}}$$

(2.8)

(vii) The strength per unit length for the $m$ panels are calculated using

$$\lambda_i = \sum_{j=1}^{m} J_{ij} (-aL_j + W_j) \quad i = 1, \ldots, m$$

(2.9)

(viii) One can then find the velocity components to find the direction of the local tangent to the path. The path is generated numerically. The path starts at $x_1 = x_s$. At any step $k$, the direction of the instantaneous velocity (the direction of local tangent to the path) and the new position are calculated as follows.

$$u_k = u(x_k)$$

(2.10)

$$\hat{u}_k = \frac{u_k}{|u_k|}$$

(2.11)

$$x_{k+1} = x_k + \Delta s \hat{u}_k$$

(2.12)

where $\Delta s$ is an arbitrary small distance. The iteration in $k$ continues until the point found for the path is closer than the defined small distance $\Delta s$.

$$\|x_{k+1} - x_g\| \leq \Delta s$$

(2.13)

(ix) The array of points $x_k$ is the planned path.

### 2.5.1.3 Path Generation

Path generation creates a path that the robot should follow in order to reach the various waypoints. There are plenty of well established methodologies through which path generation may
be done. Some of the methods include generating straight lines with simple arcs, using special curves called clothoids and generating cubic polynomials. Each method has its own advantages and disadvantages. The underlying aim in any path generation algorithm is to make the path to be followed by the robot as simple as possible. Making a path simple includes rounding off sharp corners, avoiding complex turns and making the path smooth and continuous. The reason for making the path simple is that an easily followable path reduces errors significantly, and greatly assists the path tracking algorithm. In the following discussion, two well established path generation algorithms will be described.

(i) Clothoid curves

Clothoid curves are a family of curves that are posture continuous. They have the unique property that their curvature varies linearly with the length of the curve. In mathematical terms, this may be represented by the equation

\[ c(s) = ks + c_i \]  

(2.14)

where \( c \) is the curvature, \( s \) is the length of the curve, and \( k \) is the sharpness (rate of change of curvature) of the curve (Shin & Singh 1990).

Prior to presenting the mathematics involved in clothoids, the term posture will be defined. Posture is defined as the quadruple of \((x, y, \theta, c)\) where \( x \) and \( y \) represent position of the robot, \( \theta \) represents tangent direction and \( c \) represents curvature. The reason for defining posture as a quadruple of these parameters is because posture may be thought of as the state of the robot. If the path generation algorithm ensures that posture is continuous, then path tracking of the robot is significantly simplified (Shin & Singh 1990).

Shin and Singh (1990) have determined the mathematical equations required for generating clothoids. While the details of the algorithm used in generating the path is too detailed to be presented here, the basic conditions are given in the following discussion. Given an initial posture, sharpness of the clothoid segment and distance along that segment, one may determine position, orientation and curvature at any point along the curve as follows:

\[ \theta(s) = \theta_i + \int_0^s c(\xi) d\xi = \frac{k}{2} s^2 + c_i s + \theta_i \]  

(2.15)
Shin and Singh (1990) have also provided a comparison of robot response to a path constructed of simple lines and arcs, and a path constructed using clothoids for two different open loop conditions.

(i) Cubic Polynomial

The cubic polynomial approach to path generation has been developed by Nagy and Kelly (2001), and the algorithm used is presented in this section. The algorithm is developed for an Ackermann steered vehicle, but can be adapted for the skid-steered UGV used in this project. The algorithm takes into account the initial and final postures of the UGV and develops a cubic polynomial that has the two postures as its solution. The cubic polynomial is a simple and effective algorithm that may be implemented for path generation.

The state equations for the robot are given by:

\[ \dot{x} = V(t) \cos \theta(t) \]  \hspace{1cm} (2.18) 

\[ \dot{y} = V(t) \sin \theta(t) \]  \hspace{1cm} (2.19)
\[ \dot{\theta} = \kappa(t)V(t) \] \hspace{1cm} (2.20)

\[ \dot{\kappa} = \dot{\alpha}(t)/L \] \hspace{1cm} (2.21)

where \( x \) and \( y \) represent position, \( V \) represents linear velocity, \( \theta \) represents orientation, \( \kappa \) represents curvature, \( \dot{\alpha} \) represents steer angle velocity and \( L \) is the wheelbase.

*It should be noted that a dot above any quantity represents time derivative.* The definition for posture is the same as given in Section (i).

The four equations given above may be interpreted as a set of nonlinear equations. Nagy and Kelly (2001) have determined these nonlinear equations as follows:

\[ x = f_1(a, b, c, s) \] \hspace{1cm} (2.22)

\[ y = f_2(a, b, c, s) \] \hspace{1cm} (2.23)

\[ \theta = f_3(a, b, c, s) \] \hspace{1cm} (2.24)

\[ \kappa = f_4(a, b, c, s) \] \hspace{1cm} (2.25)

where \( a, b \) and \( c \) are constants; and \( s \) is the arc length.

Giving the initial posture a subscript of \( o \) and the final (or desired) posture a subscript of \( f \), Nagy and Kelly (2001) formulated the equations that will generate a valid cubic path between the two postures. These equations are presented below. *All coordinates are expressed in terms of the initial posture, so that initial position and heading coordinates are zero.* The constraints that must be satisfied are:

\[ x(s_f) = x_f \] \hspace{1cm} (2.26)

\[ y(s_f) = y_f \] \hspace{1cm} (2.27)

\[ \kappa(0) = \kappa_o \] \hspace{1cm} (2.28)
\[ \theta(s_f) = \theta_f \]  \hspace{1cm} (2.29)

\[ \kappa(s_f) = \kappa_f \]  \hspace{1cm} (2.30)

The implementation of this method is done in an iterative fashion. The first iteration is performed by assuming the \( c \) parameter to be zero, and using an approximation of the observed average relationship between \( s \) and the total change in heading between the start and end postures over a large sample set to determine a value for \( s \).

The equations are as follows:

\[ d = \sqrt{x_f^2 + y_f^2} \]  \hspace{1cm} (2.31)

\[ \Delta \theta = |\theta_f| \]  \hspace{1cm} (2.32)

\[ s = d \left( \frac{\Delta \theta^2}{s^2} + 1 \right) + \frac{2}{5} \Delta \theta \]  \hspace{1cm} (2.33)

\[ c = 0 \]  \hspace{1cm} (2.34)

\[ a = \frac{6\theta_f}{s^2} - \frac{2\kappa_o}{s} + \frac{4\kappa_f}{s} \]  \hspace{1cm} (2.35)

\[ b = \frac{3}{s^2} (\kappa_o + \kappa_f) + \frac{6\theta_f}{s^2} \]  \hspace{1cm} (2.36)

This algorithm is then repeated iteratively until termination. The termination conditions for the algorithm are based on the accuracy in the destination position that is needed (Nagy & Kelly 2001). Nagy and Kelly found the algorithm outlined in Section (i) to be practical for an Ackermann steered vehicle over the following relative end posture envelope (Nagy & Kelly 2001):

\[ -1.0 \text{m} < x_f < 1.0 \text{m} \]  \hspace{1cm} (2.37)

\[ 1.0 \text{m} < y_f < 5.0 \text{m} \]  \hspace{1cm} (2.38)
\[ -\frac{4\pi}{5} < \theta_r < \frac{4\pi}{5} \]  
\[ -0.1 \frac{1}{m} < \kappa_r < 0.1 \frac{1}{m} \]  

Figure 2.27 is an example of the results that Nagy and Kelly obtained.

Path tracking checks the progress of the robot with the original path and corrects for any errors that may be present. Various tracking systems exist. They tend to be based on the kinematic model of the vehicle with a feedback loop to correct for the path being traversed (Fahimi 2008).

2.5.1.4 PATH TRACKING

2.5.1.5 PLAN-BASED AND REACTIVE SYSTEMS

As an extension of the approaches described above, the system can either be reactive, whereby the route is recalculated once there is new information, or it can be plan-based. “Plan-based systems are systems where the robot generates the control signals by maintaining and executing a plan that is effective and has a high expected utility with respect to the robot’s dynamically changing belief state. In this control paradigm, plan management operations and plan execution are performed in concurrent threads of control. The plan management operations pre-compute control decisions and install them into the plan. The plans generate the control signals for the subsequent feedback cycles” (Beetz 2002). Figure 2.28 gives an example of plan based features:

\[ \text{With plan adaptor Whenever the robot receives new requests} \]
\[ \text{if its set of jobs might interfere} \]
\[ \text{then it changes the course of action} \]
\[ \text{to avoid these interferences} \]

\[ \text{Concurrent reactive plan} \]

\[ \text{With plan adaptor Whenever the robot detects an open door} \]
\[ \text{that was assumed to be closed} \]
\[ \text{if this situation is an opportunity} \]
\[ \text{then it changes its course of action} \]
\[ \text{to make use of the opportunity} \]

\[ \text{Concurrent reactive plan} \]

Figure 2.28: Plan-based policies (adapted from Veres 2005)
These plan-based system architectures have their roots in attempting to understand and model the decision making process of humans to further their goals. Thus, the system combines three sets: beliefs, desires and intentions. Subsequently, system functions, such as an option function, can be defined that map the set of beliefs and intentions into the set of desires. The system would then execute based on the appropriate intentions value (Veres 2005). A system architecture is shown below in Figure 2.29.

![Simple Plan-Based Architecture](image)

**Figure 2.29:** A plan-based architecture (adapted from Veres 2005)

### 2.5.2 MULTIPLE VEHICLES AUTONOMOUS CONTROL

The difference between multiple vehicle autonomous control and single vehicle autonomous control is that using multiple vehicles allows for different strategies. Of course, the vehicles may just take the same path and stay as a group, but that is not utilising the potential of having multiple vehicles with the same ability of choosing way points to reach the target. With numerous vehicles, the role of each vehicle may differ and while one may take the main path towards the destination others may take a different route to maximise exploration and the hazards of having too many vehicles in close proximity.

#### 2.5.2.1 THREE LAYERED SYSTEM

Different path planning approaches have been integrated in a hierarchical manner to address obstacle avoidance for mobile robots in cluttered environments. For example, a three-layer hierarchical path planning system has been presented that consists of global planning, local navigator, and collision avoidance algorithms (Fahimi 2009). Thus, the global planning system sets destinations for each vehicle. The actual route taken is determined by the local navigator, while constantly running the collision avoidance algorithm.
2.5.2.2 Two Layered System

There are other systems in use such as the following two-tiered approaches, where the first layer system is a global planning system as before, but the second layer performs the same functions as the second and third layer of the systems described above. A relatively simple example of such a system uses a spring-damper model system to model the interactions between all objects including vehicles and obstacles (Fahimi 2009). Fahimi (2009) details an extension of the potential field approach discussed in Section 2.5.1.2 for the situation involving multiple robots. It is important in this scenario to consider the timing of the robots’ motions for collision avoidance. This can be done by introducing artificial potential fields in the robots’ extended configuration space-time. Another method for addressing multiple mobile robots using the potential field approach is using potentials that are functions of the relative speeds of the robots as well as their distances. The potential method can have problems such as the existence of local minima in the artificial potential field, which can inadvertently trap the robot, preventing it from reaching its destination. Methods, such as search can counteract this but at high computational expense. Another method for avoiding the generation of local minima is the addition of multiple attraction potentials, whose positions are determined by a genetic algorithm. Also, there have been methods developed that avoid minima for certain scenarios (Fahimi 2009). It has been shown that harmonic potential functions do not suffer from local minima and lead to unique solutions.

Figure 2.30: Obstacle avoidance with multiple mobile robots (Fahimi 2009)

An example of how this works in real time is shown in Figure 2.30. The figure shows six snapshots of a robot group in motion. The start (S) and goal (G) points are marked on the figure. At the centre of each frame we assume there are obstacles that are leaving a narrow passage for the robots to
pass through. The robots are represented by rectangles with arrows designating their direction of movement. As can be seen, the robots cannot pass the opening at the same time and are likely to collide. The only way is to go through the opening one at a time. Also, note that robots 1 and 2 are symmetrical. A symmetry detecting algorithm slows down robot 1 to avoid collision. As seen in Figure 2.30, robots 1 and 2 slow down while robot 3 is passing the opening, after which robot 2 speeds up and robot 1 follows. The robots manage to arrive at their corresponding goal points without colliding with each other and the obstacles. Robots 1, 2, and 3 are at their goals after 58, 50, and 27 s, respectively (Fahimi 2009).

2.6 COMPUTER VISION

Computer vision is the science and mathematics of computer technology that sees. Alternatively, it is the extraction of useful information from pictures or sequential pictures to make a decision or construct a representation of the environment (Bradski & Kaehler 2008). Thus, for the MAGIC competition, computer vision has the potential to be an extremely effective tool in the detection and identification of the competition-specified objects, shown in Figure 2.31. The use of a camera for this work is by no means a unique science. Most people associate computer vision with surveillance cameras and web-cameras, yet it is becoming increasingly pervasive in many areas of society, including product inspection in manufacturing lines, detection of events in security monitoring, modelling three dimensional objects in medical and mapping applications and in the interaction between humans and computer technology (Bradski & Kaehler 2008). According to Forsyth and Ponce (2003), the main functions used in the hundreds of applications of computer vision, are:

- image enhancement – accentuating important regions of interest,
- inspection – ensuring objects are within specification,
- interpretation – assisting quantitative assessments of objects and
- organisation – structuring image libraries.

Thus, computer vision is ideal for the MAGIC competition because of a number of useful properties when considered as a sensory tool. First, it is non-destructive and usually discreet. The vision equipment can be quite small and can gather pictures without impacting on the surrounding environment. Second, computer vision is relatively cheap compared to the cost of other robotics sensors (Forsyth & Ponce 2003). When it is considered that computer vision tools can extract colour, texture, position, dimensions, orientation, edges, paths and predicted paths of multiple objects concurrently, all using the one sensor, it makes sense that vision systems are being increasingly used in many industries. The cost of computer vision systems has dropped dramatically in recent years, making computer vision much more practical for many applications (Forsyth & Ponce 2003).

2.6.1 THE DIFFICULTIES OF COMPUTER VISION

Initially, it is difficult for humans to understand the complexities that computer vision systems face. Humans have been taught to identify objects with their appropriate names since an early age and it is this accumulated knowledge that computer vision systems lack. In fact, the brain contains an extremely complex visual feedback system of which little is understood. The human senses, as well as muscle senses, are gathered as cross-associations from a person’s visual memory, in identifying objects. The human eye also is a highly sophisticated piece of hardware, not only sending large amounts of data to the brain but adjusting focus using the iris and tuning reception of light information using the retina (Forsyth & Ponce 2003). All of this contrasts heavily with the
current, relatively simple technology of computer vision, where the image received by a computer is seen as a grid of numbers, depicted in Figure 2.32. Therefore to identify any desired object, computer vision largely involves various kinds of mathematics and programming skills to extract information from the number arrays. For the MAGIC project, this involves scanning the images taken by a camera on the UGV for the colours of the competition-specified objects of interest, and examining the detected regions of colour for properties that match those of the desired objects.

Another obstacle facing computer vision is depth perception. For a human, the brain cortex uses the two slightly different perspectives from each eye to construct an image with depth of the observed scene. This phenomenon is known as parallax, since both eyes are observing the same object, but from slightly different positions and orientations (Stereo Vision 2010). This effect is demonstrated in Figure 2.33, where the view from each eye is shown at the left of the picture and the combined image created by the brain is seen in the lower right hand corner of the picture.
In computer vision, the approach of using two cameras to be able to perceive depth is known as stereo vision. The advantage of stereo vision is that objects seen may be given three-dimensional locations, which is useful for understanding the surrounding environment to a greater extent than with the two-dimensional locations, that can be extracted from images from a single camera. However, stereo vision means the cost of two cameras has to be borne. For a single camera and simple scenarios, a technique known as homography can assist in the estimation of some depth information from two-dimensional images. In computer vision, planar homography maps the coordinates on one plane to another (Bradski & Kaehler 2008, p. 384). For distance estimation, the assumption is made that the ground plane, on which the object is residing, has a known coordinate translation to the plane of the camera image sensor (Madden C 2010, pers. comm. 2 April 2010). The distance calculation from the camera to the object is a simple trigonometric calculation, as shown in 2.34. In urban environments, this is usually an acceptable approximation. However, the assumption can lead to large errors if the ground is not flat or the base of the object is obstructed from the camera view, meaning the coordinate transformation between the camera image sensor plane and the object ground plane becomes very approximate. As a result of these errors, cameras are often used in conjunction with a distance sensor (such as a laser device), that points parallel to the camera onto the object when the object is in the centre of the camera view. A combination of a camera and a distance sensor mounted on a rotating and tilting unit on top of each UGV was used in this project to give the identified objects real world coordinates.

2.6.2 COLOUR SPACES

To understand how computer vision calculations are made to identify colours, a number of colour models need to be investigated. The RGB (Red Green Blue) colour model is used by hardware in the conversion of light to digital values and vice-versa in both the input and output of colours to devices such as scanners, cameras and computer monitors. By mixing full intensity red, green and blue together in the RGB format, the result is white light. Thus, RGB is called an additive model (Masterson & Frank, 2007). As shown in Figure 2.35, this is in contrast to the more commonly known CMY (Cyan Magenta Yellow) colour model, used in printer inks and paints. The CMY colour model is used when light falls on a colour, such as red, and all light except red is absorbed. Thus, the full intensity of each CMY colour produces black, subsequently being called a subtractive model (Masterson & Frank, 2007). Other than RGB, the most common colour space used in computer vision is HSV (Hue Saturation Value). HSV is a colour space where colours are recognised as being the same colour under varying light intensity changes, which is closer to human colour
2.6.3 Current Examples of Computer Vision

Two recent examples of mobile robots in urban environments that used computer vision are the Madrid Polytechnic University UAV and the DARPA Grand Challenge. Both examples were helpful in demonstrating the capabilities of computer vision systems coupled with other sensors as in the MAGIC project.
The Madrid Polytechnic University has three unmanned aerial vehicles (UAVs), including two gas powered Bergen Industrial Twin 52 c.c. helicopters (see Figure 2.37) and a Rotomotion SR20 electric UAV. Fitting computer vision systems to these UAVs, the research group was able to test a number of computer vision techniques. The future of these UAVs lies in dangerous or mundane tasks in inspecting infrastructure such as power, oil, gas lines and roads. A UAV would be an economically viable and feasible alternative to the current human-operated inspection vehicles.

One of the major benefits of a computer vision system for an autonomous vehicle is the ability to manoeuvre relative to an identified object, especially when the GPS signal experiences a dropout or low strength (such as in high urban canyons). The Madrid Polytechnic University group was especially interested in using a form of visual SLAM to locate the UAVs and the objects which the attached cameras were observing in the event of a lack of a GPS signal. This feature could also be used on the MAGIC UGVs, but was too advanced to implement but may be included in future years. According to Campoy et. al. (2008, p. 111), this is achieved through a number of image processing techniques, including:

- a Harris Corner detector - to detect corners of objects,
- a Scale Invariant Feature Detector (SIFT) - to detect points of interest, or keypoints,
- the Hough transform - to find straight lines, and
- the Canny edge detector - to identify curves.
Feature tracking describes a technique used to follow objects, by identifying the features of the object, such as curves, edges, shapes, corners and other characteristic geometry and then detecting these same features in subsequent images in the visual feed. The Madrid Polytechnic group used two main feature tracking methods. The first technique tracks corner features and points of interest in sequential images and is called the Lukas-Kanade algorithm. It identifies a particular feature, $p_i = (x, y)$ in the image $I_k$ and then looks for the feature with a similar intensity and small position deviation in the consecutive image, $I_{k+1}$. The displacement from the original feature position is converted to a velocity and labelled as the flow vector $t = (t_x, t_y)$. Thus, the position of the feature in the second image is the original position plus the flow vector, $p'_i = (x, y) + t$. This process is prone to noise and its reliability is increased by the addition of a Kalman Filter. This technique is also quite useful in giving an estimate of the velocity of the UAV itself (Campoy et. al. 2008, p. 112).

The second technique used by the Madrid Polytechnic group to track objects makes use of the SIFT descriptor. A template of features was constructed from an image of the desired object to be tracked. These features were then searched for in the visual feed. The strength of the SIFT system is its robustness, such that the identification of the object does not depend on the relative positions of the template and the tracked object. Once it has been identified, a series of transformations are made to determine the perspective the UAV had of the object, including rotation, scaling and translation to account for any rotation and movement of the object relative to the UAV in between frames (Campoy et. al. 2008, p. 112).

Appearance based tracking

Instead of searching for features to identify an object, appearance based tracking uses a section of pixels that have been identified as the desired object to be tracked. Due to the relative movement of both the UAV and the object (if it is moving), a complex warping model is used to match the final patch of pixels with the original template as shown in Figure 2.38. Both appearance based and feature tracking would be helpful for increased accuracy in identification of the competition-specified objects, but were too advanced for implementation in the MAGIC project this year.
The Defence Advanced Research Agency (DARPA) Grand Challenges in 2004, 2005 and 2007 provided definitive proof that the use of computer vision to guide movements is indispensable for autonomous vehicles traversing unknown areas at a reasonable speed. At least one camera was mounted on the majority of vehicles in the DARPA challenge, since the competition focussed on safety as much as speed. In fact, the Stanford Racing Team’s car crossed the line first but the team was awarded second place, since the Tartan Racing Team’s vehicle fulfilled the challenge guidelines most closely. A high situational awareness accomplished by the fusion of data from multiple cameras and sensors was required to avoid other cars on the populated roads as well as static obstacles near the roadways. For “Junior”, the Stanford Racing Team’s vehicle, computer vision was particularly required for travelling along bituminised roads at as high a speed as possible. The processed images were analysed for lane markings, which were then distinguished by colour and pattern. Curbs on road edges were identified by a laser scanner. These physical characteristics of the vehicle route were then correlated with supplied high definition maps, correcting possible errors in the GPS positioning. Warning signs and cluttered laneway markings were also able to be filtered out of the images using computer vision algorithms. Thus, through the fusion of vision and other sensor data, the pose of “Junior” was accurate to within 5 centimetres laterally and half a metre in the direction “Junior” was travelling. The fusion of data from multiple sensors to increase reliability of measurements is implemented in the MAGIC UGVs. However, the vision system was only used to identify objects of interest and their location, and did not assist with any other localisation via environment feature recognition.
Part I

HARDWARE
A large amount of hardware was required to be bought or designed to achieve the goals of this project. Figure 3.1 displays the three major categories that were defined based on the project goals. The first category, localisation and mapping was a critical aspect of the project. Localisation is essentially the ability to determine the position of the Unmanned Ground Vehicles (UGVs) at any time and in any environment. Mapping is the ability to determine what is present in the surroundings and record them in a form that both a computer can use and a human can visualise. The second important goal that required hardware on the UGVs was its requirement to be able to communicate both externally with each other and the Ground Control Station (GCS) while also communicating internally between hardware. The third critical goal was the ability to be able to mobile and control the UGV in a range of indoor and outdoor environments.

The level one hardware flow chart seen in Figure 3.2 refers to the overall hardware architecture and demonstrates at a high level how the UGVs and the GCS communicate with each other with regards to hardware. Figure 3.3 drops down a level to provide more detail about the hardware structure of an actual UGV. This is a level two hardware flow chart and it indicates what hardware will be mounted inside and outside the UGV. This flow chart also indicates hardware that has been selected and purchased, as opposed to hardware that has been designed and built specifically for this purpose. It also is complete with references to sections where more information regarding how these pieces of hardware were either selected or designed.

**Figure 3.1:** Flow chart showing the main sections of the Hardware Architecture

### 3.1 LOCALISATION AND MAPPING

The localisation and mapping was grouped into three separate sections each with a unique set of hardware requirements. A flow chart detailing these three sections is shown in Figure 3.4. The hardware utilised within the UGV had to be capable of self positioning or localisation to an accuracy of at least 20 cm in both outdoor and indoor environments. Further, hardware was needed to determine the distance from the UGVs to objects situated within the immediate environment, which along with the accurate positioning hardware was able to contribute to producing 3D map representations of these environments and be capable of avoiding potentially damaging collisions. Hardware was also required to enable further environmental awareness by being able to locate, identify and neutralise objects of interest.
3.1.1 ACCURATE POSITIONING

Accurate positioning is required to ensure that the UGVs can navigate through doorways and that the position of each UGV can be determined with a desired accuracy of at least 20 cm. The basis of having accurate positioning in hardware capabilities was critical in enabling software development to achieve the accuracy desired and stems to other crucial aspects such as mapping and navigation (control). To achieve accurate positioning outdoors a differential Global Positioning System (dGPS) unit was selected. Indoor positioning was enabled with the use of an Inertial Measurement Unit (IMU).

3.1.1.1 DIFFERENTIAL GLOBAL POSITIONING SYSTEM

A Global Positioning System (GPS) unit is critical to the localisation of most UGVs designed for outdoor operation. GPS units offer simple, reliable and accurate position information and are widely commercially available. The features of a GPS system are discussed in Section 2.3.1. One of the unique aspects of the GPS system, when compared to other methods of localisation, is that position is determined from a global perspective and does not drift or accumulate errors relative to the changing position of the desired target being located. The option of then obtaining a differential GPS (dGPS) system which enables further accuracy was a valuable improvement for the desired localisation, accuracy requirements. A dGPS requires the use of a base station of known exact location that can differentiate the relative position of the UGV to acquire improved accuracy.

A GPS supplier Novatel offered upgraded firmware on their product for the duration of the MAGIC competition enabling a claimed 2 cm accuracy on their dGPS OEMV-2 model mounted in Novatel’s
Figure 3.3: Level two - UGV hardware flowchart
standard FlexPak G2 case and matched with a Novatel GPS 701 GG Antenna. Hence, this supplier and particular hardware was chosen. The specifications of the Novatel OEMV-2 model chosen are listed in Table 3.1. An image of the selected GPS is shown in Figure 3.5 and additional information on the selection criteria for the GPS unit can be found in Appendix B. The datasheet for this component can be found in Appendix D.

Table 3.1: Specifications of the Novatel OEMV-2 unit

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Actual Specifications (Novatel 2010)</th>
<th>Acceptable Specifications (Appendix B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>20 Hz</td>
<td>20 Hz</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.01 m to 1.8 m</td>
<td>0.2 m</td>
</tr>
<tr>
<td>Serial Communications</td>
<td>RS232, RS422, USB, Digital</td>
<td>RS232</td>
</tr>
<tr>
<td>Size</td>
<td>147 x 113 x 45 mm</td>
<td>&lt;200 mm in any dimension</td>
</tr>
<tr>
<td>Weight</td>
<td>338 g</td>
<td>&lt;500 g</td>
</tr>
<tr>
<td>Power requirements</td>
<td>1.4 W at 12 V</td>
<td>&lt;5 W, &lt;24 V</td>
</tr>
<tr>
<td>Antenna connection</td>
<td>TNC (Coaxial)</td>
<td>No special requirements</td>
</tr>
<tr>
<td>Antenna size</td>
<td>185 mm (diameter) 69 mm (height)</td>
<td>&lt;200 mm in any dimension</td>
</tr>
<tr>
<td>Antenna weight</td>
<td>0.5 kg</td>
<td>&lt;1 kg</td>
</tr>
<tr>
<td>Weatherproof</td>
<td>IEC 60529 IPX7 submersible up to 1 m</td>
<td>Antenna needs to be showerproof. Receiver stored inside UGV.</td>
</tr>
<tr>
<td>Cost</td>
<td>$2940</td>
<td>minimise</td>
</tr>
</tbody>
</table>

3.1.1.2 INERTIAL MEASUREMENT UNIT

The Inertial Measurement Unit (IMU) is required to work in conjunction with other sensors for localisation. The dGPS is capable of providing a very accurate position of the UGV, but is limited to outdoor areas with an open environment for signal strength. Hence, this system is unavailable
indoors and unreliable in urban areas with dense population of buildings. The GPS is also slow to update, with a typical speed of around 5Hz. This can be a problem when the UGV is moving quickly and so another sensor is needed that is faster and can work during GPS signal dropouts. An IMU is a package of inertial sensors and typically consists of 3 orthogonally mounted gyroscopes, 3 accelerometers, a magnetometer and a thermometer. By combining the raw data, it is possible to obtain an estimate of heading and acceleration, which is sufficient for a dead reckoning navigation model (See Section 2.3). However this system is unreliable for any prolonged periods of time due to the integrating of acceleration required and the introduction of drift. This dead reckoning model should be sufficient to navigate over short distances, such as those required for indoor navigation and will be used when accurate GPS position information is not available. The datasheet for this component can be found in Appendix D.

The IMU model chosen was the Microstrain 3DM-GX3-25, displayed in Figure 3.6. This model was deemed comparable to other IMUs on the market and a relationship with the supplier enabled the IMUs to be purchased at a discounted price. The University of Adelaide also has previous experience with Microstrain sensors and this has proven to be useful in developing robust code to read the data from the IMU. The specifications of the IMU can be seen in Table 3.2 along with the
minimum acceptable requirements. Additional information on the selection process of the IMU can be found in Appendix B.

Table 3.2: Specifications of the Microstrain 3DM-GX3-25 chosen.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Actual Specifications (Microstrain n.d.)</th>
<th>Acceptable Specifications (Appendix B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (heading)</td>
<td>±2°</td>
<td>±2°</td>
</tr>
<tr>
<td>Gyro Range</td>
<td>±300 deg/sec</td>
<td>±50 deg/sec</td>
</tr>
<tr>
<td>Speed</td>
<td>Up to 250 Hz, Sensors sampled at 30kHz and update estimate onboard</td>
<td>25 Hz, Sensors sampled at 250Hz</td>
</tr>
<tr>
<td>Communications</td>
<td>USB2.0 / RS232</td>
<td>RS232</td>
</tr>
<tr>
<td>Connector</td>
<td>mini DB9</td>
<td>No special requirements</td>
</tr>
<tr>
<td>Size</td>
<td>45 x 25 x 11mm</td>
<td>&lt;100 mm in any direction</td>
</tr>
<tr>
<td>Weight</td>
<td>18 g</td>
<td>&lt;200 g</td>
</tr>
<tr>
<td>Power requirements</td>
<td>5-12V, ~0.5W</td>
<td>&lt;24V, &lt;5W</td>
</tr>
<tr>
<td>Cost</td>
<td>$1421.25</td>
<td></td>
</tr>
</tbody>
</table>

3.1.2 ENVIRONMENTAL AWARENESS

Environmental awareness was required for the location, identification and neutralisation of objects of interest (OOI) which was achieved with the combined use of a camera and laser pointer. The laser pointer is no longer required for the project but was necessary for the MAGIC 2010 competition. For more information on the laser’s selection process see Appendix B.

3.1.2.1 CAMERA

The camera model purchased was a Baumer TXG20c-P digital colour camera. An additional wide-viewing lens for greater viewing range, a Kowa LM3NCM lens, was also purchased to screw on to the end of the Baumer camera. Both the camera and the lens are displayed in Figure 3.7. A discount on the lenses and cameras was obtained from Atlantek Vision because of the bulk purchase order made. The minimum camera specifications are compared to the Baumer TXG20c-P specifications in Table 3.3. Additional information on the selection process for the camera can be found in Appendix B. The datasheet for this component can be found in Appendix D.

3.1.3 DISTANCE TO OBJECT MEASUREMENT

Both long range and short range distance to object measurement was required. The long range solution was a Light Detection and Ranging Sensor (LiDAR) and the short range solution were ultrasonic sensors.
Table 3.3: Specifications of the Baumer TXG20c-P digital colour camera with Kowa LM3N3CM Lens attached

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>1624 x 1232 pixels</td>
<td>min 800 x 600 pixels</td>
</tr>
<tr>
<td>Comms. Ports</td>
<td>Gigabit Ethernet (transfer rate 1000 Mbits/sec)</td>
<td>max Gigabit Ethernet</td>
</tr>
<tr>
<td>Datarate</td>
<td>max possible: 16 Mbits/sec</td>
<td>max 50 Mbits/sec</td>
</tr>
<tr>
<td>Size</td>
<td>total 42 x 42 x 96.2 mm</td>
<td>max 100 x 100 x 120 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>total 185 g</td>
<td>max 300 g</td>
</tr>
<tr>
<td>Power</td>
<td>4.3 W, PoE</td>
<td>max 10 W, Power over Ethernet (PoE)</td>
</tr>
<tr>
<td>Output</td>
<td>8P8C Modular Jack RJ45 - screw lock type, 12 bit A/D</td>
<td>Bonus feature only. May be used for powering and controlling laser.</td>
</tr>
<tr>
<td>Data Format</td>
<td>RGB Bayer Mosaic (3 variations), YUV Packed (3 variations)</td>
<td>Min RGB</td>
</tr>
<tr>
<td>Cost</td>
<td>$2248.66</td>
<td>minimise</td>
</tr>
</tbody>
</table>

Figure 3.7: Baumer TXG20c-P digital colour camera (left) and Kowa LM3N3CM lens (right) (Baumer n.d.)

3.1.3.1 LIGHT DETECTION AND RANGING SENSOR

The LiDAR uses infrared beams with a 190° scan range and 0.36° angular spacing to measure distances of up to 65m, this angular spacing results in a 1m spread at 65m. The large scan angle means that the pan/tilt unit (PTU) will not need to pan as much and this reduces the power consumption of the pan motor.

The result from the decision matrix which can be found in Appendix B was that the 2D LiDAR was the most appropriate sensor to provide range information to the UGV. The 2D LiDAR chosen was the Leuze Rod4-08 plus, displayed in Figure 3.8. This LiDAR was chosen since it satisfied the UGVs range, accuracy and data transfer requirements. The Rod4-08 plus is also equipped with a 100Mbit/s Ethernet connection and a mount for easy bolt on installation to a pan/tilt unit. Also a business relationship with the suppliers allowed the sensors to be purchased at cost price, which is a significant cost reduction considering that up to ten sensors were initially planned to be purchased.
The minimum acceptable specifications of the LiDAR are tabulated in Table 3.4 along with the actual specifications of the Rod4-08 plus LiDAR. The specifications required for the LiDAR were created by considering the need of the sensor to observe distant objects accurately; the requirement to provide associated range data at a relatively high rate, and also the need for the sensor to be easily integrated into the UGV’s electrical, mechanical, and computer systems. Additional information on the selection process for the LiDAR can be found in Appendix B. The datasheet for this component can be found in Appendix D.

Figure 3.8: Leuze Rod4-08 plus 2D LiDAR

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>65 m</td>
<td>Min 50 m</td>
</tr>
<tr>
<td>Accuracy</td>
<td>5 mm</td>
<td>Min 10 mm</td>
</tr>
<tr>
<td>Angular Resolution</td>
<td>0.36 degrees</td>
<td>Min 1 degrees</td>
</tr>
<tr>
<td>Scanning rate</td>
<td>25 Hz</td>
<td>Min 20 Hz</td>
</tr>
<tr>
<td>Angular range</td>
<td>190 degrees</td>
<td>180 degrees</td>
</tr>
<tr>
<td>Environmental Protection class</td>
<td>IP65</td>
<td>Min IP65</td>
</tr>
<tr>
<td>Power consumption</td>
<td>24 W</td>
<td>Max 50 W</td>
</tr>
<tr>
<td>Voltage supply</td>
<td>24 V</td>
<td>5 V, 12 V or 24 V</td>
</tr>
<tr>
<td>Communication connections</td>
<td>Ethernet/RS232 Serial</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Weight</td>
<td>2.3 kg</td>
<td>5 kg</td>
</tr>
<tr>
<td>Laser class</td>
<td>Class 1</td>
<td>Max Class 2</td>
</tr>
<tr>
<td>Cost</td>
<td>$3728.5</td>
<td>minimise</td>
</tr>
</tbody>
</table>
3.1.3.2 ULTRASONIC SENSOR

The short range distance to object measurement requirements were met with the use of ultrasonic sensors. The main reason for using ultrasonic sensors in the UGV is to detect objects at close range (2-3m) to prevent the UGV from colliding into obstacles, should the navigation algorithms fail or an obstacle not be detected by the LiDAR. The low cost of purchasing ultrasonic sensors easily offsets the potential cost of damage to equipment should the UGV hit an obstacle at high speed. The ultrasonic sensors to be used on each UGV are two XL-MaxSonar-WR1 sensors, as shown in Figure 3.9. These were selected as a relationship with the supplier enabled sensors and mounting accessories to be purchased with a 40% discount. Furthermore, an IP-67 environmental protection rating (MaxBotix, n.d) means that the ultrasonic sensors are protected from dust infiltration and can be immersed in low pressure water, which will provide protection against rain, dew or splash. Communication with the sensors is being achieved by wiring to a microcontroller due to multiple serial ports being unavailable. The mounting of the sensors is simple, requiring only drilled holes on the front of the UGV chassis, a rubber O-ring and a lock nut. The datasheet for this component can be found in Appendix D.

![Figure 3.9: XL-MaxSonar-WR1 ultrasonic sensor](image)

3.1.4 COMPONENT ORIENTATION CONTROL

The LiDAR sensor, camera and laser pointer require additional movement that the UGV cannot provide to map the surrounding area and track and classify objects. This was solved by the use of a pan/tilt unit (PTU). The PTU is a freely rotating mount for the ROD4 LiDAR sensor, Baumer camera and now obsolete JPL laser pointer consisting of two motors and a structure which can rotate the hardware to providing the required viewing range. The tilt motor spans the vertical plane and works cooperatively with the pan motor that controls the horizontal span. The PTU was mounted on top of the chassis towards the front for maximum viewing capabilities.

Early in the design phase of the PTU, supervisors and the mechanical workshop requested that the manufacturing difficulty of the design be reduced significantly or outsource the manufacture. This request was made due to the limited hours available at the Adelaide University workshop. As the means of manufacturing the PTU was uncertain in the early design stages, two designs were produced in parallel; one by the University of Adelaide students and the other by project partners, Strategic Engineering. The Strategic Engineering design was made such that it was difficult to manufacture however, it was simple to assemble and had robust functionality. The Adelaide University design was made significantly easier to manufacture however assembly was more complex and the overall function was less durable. Later in the project, it was decided that project partners, Strategic Engineering would fund the outsourced manufacture of the PTU and hence, their design was used for the project, which will now be referred to as the primary design. This section will focus on the primary design. For more information on the secondary design, refer to Appendix B. The two final designs can be seen in Figure 3.10a and 3.10b.
3.1.4.1 REQUIREMENTS

Range - The PTU was required to provide the LiDAR, camera and laser pointer with 360° range of horizontal vision as well as 90° range of vertical vision for the LiDAR and camera, comprising of 45° above and below the horizontal.

Robust - The PTU was required to be very robust as it was subjected to vibration while the UGV was manoeuvring and given its position may collide with objects during testing. Any vibration will affect the accuracy of the LiDAR and camera.

Viewing Range Interference - The PTU structure cannot interfere with the vision of the LiDAR sensor, camera or laser pointer. This is particularly difficult with the LiDAR as its viewing window spans half of the unit as can be seen in Figure 3.10a.

Weight - The overall weight of the UGV is required to be less than 40kgs. Additionally, the required torque of the motors is dependent on the weight of the components it is turning and hence, the PTU weight must be kept to a minimum.

Mounting - The PTU will be mounted to the chassis near a number of other components and in order for it to rotate and tilt free of restrictions, there must be no obstructions. This will result in a large amount of space being occupied for any unnecessary additions to the horizontal envelope of the PTU. Hence the PTU structure is required to encapsulate the LiDAR with minimal additional volume (the LiDAR being largest component and that which the PTU will be built around).

Materials - Aluminium was chosen as the material best suited to make the structure from for the reasons discussed in section 3.3.3.2

There are also a number of requirements for the motor assemblies used to pan and tilt, these will be discussed in the relevant section.
3.1.4.2 STRUCTURAL DESIGN

As aluminium was being used for the PTU structure, the strength of the design was not of high concern. The weaker points of the design such as motors and bearings would fail first. The motors and bearings were chosen with high safety factors for this reason.

The geometry of the hardware and the requirements limit the structural positioning and hardware mounting options. However in order to reduce the required torque to rotate the system the horizontal and vertical axises were positioned at close proximity to the centre of gravity of their relative rotational sections. To reduce the difficulty of this requirement the LiDAR was mounted with inverted orientation. The centre of gravity of the LiDAR (the largest and heaviest component being mounted) is marginally above the viewing window. Hence by mounting the LiDAR sensor in an inverted fashion the horizontal axis can be positioned directly below the viewing window. Additionally the LiDAR contains a flat base which facilitates mounting the camera and laser pointer and provides a more direct path for the component cables.

The component harness of the PTU was built predominatly around the LiDAR due to its relatively large size and difficulty to mount. As shown in Figure 3.11 there are only 4 mounting holes along the back of the LiDAR which can be used for mounting purposes. Hence the main harness of PTU was built along the conditions of the LiDAR (around its viewing window) and is bolted to the LiDAR using the four bolt holes. This design is very compact and uses minimal material, which in turn minimises weight and cost.

![Rear View of LiDAR sensor](image)

A flat plate was mounted between the two harness pieces along the base of the LiDAR sensor (top of the PTU due to inverted orientation) to mount the camera and laser pointer. This plate is used to mount the camera and laser pointer directly above the LiDAR for maximum vision and to minimise the movement of their centre of gravity while panning and tilting. The camera is mounted to the plate via two countersunk bolts which come up through the bottom of the plate and into two threaded holes on the camera. As shown in Figure 3.10a the laser pointer is mounted directly above the camera via an adjustable fitting. The harness passes around the camera to allow additional restriction to the cameras movement via two more bolts. This design produces no additional width to the design and is very resistant to vibration and collision.

The LiDAR harness is supported by two axes on either side. One axis is the output shaft of the tilt motor and the other side is a custom made shaft. This is held in positions by a U-shaped section which connects to the lid of the base of the PTU as shown in Figure 3.12. The U-shaped section is also used to mount the motor used to tilt the LiDAR harness. Rotational limitations
were implemented in the form of a rubber stopper on the LiDAR harness. This restricts the tilting capabilities of the PTU to +/- 45°, this design can be seen in Figure 3.13. The tilt motor is encased within an aluminium box to protect the motor and cables from damage.

![PTU Diagram](image)

**Figure 3.12**: Front view of PTU showing the position of the tilt axis and u-shaped section

The base of the PTU which houses and mounts the motor used to pan is fixed to the top of the UGV. The pan motor encased within the base rotates a lid on top of the base. This allows the PTU to pan in a very robust manner as well as raise the PTU up an additional 83mm, which allows the LiDAR and camera to view the surrounding area from a more advantageous angle.

### 3.1.4.3 PAN AND TILT MOTORS

Both the pan and tilt mechanisms use a motor assembly consisting of a motor, gearbox(s), and an encoder which drive and measure the position of the PTU. A decision matrix has not been included in this selection, since the motors were selected by project partners Strategic Engineering. However, it must still be verified that the supplied motors meet the minimum requirements.

### MOTORS AND GEARBOXES

The minimum requirements of the motor assemblies for the PTU were obtained by considering the weight, size, backlash, counts per turn, torque and power requirements. Cost was not considered as a constraint since Maxon Motors were sponsoring the MAGIC project, and suitable motors could be obtained from Maxon Motors at a discounted price.

It was desirable for the pan and tilt motors to run on 24VDC so they can be powered by the same source as the drive motors. The torque requirements of the PTU motors are determined by the mass moment of interia, required angular acceleration, amount of vibration subject and friction in the PTU. The PTU was designed to cover a 90° vertical span in two seconds by project partners Strategic Engineering. This required that the minimum speed of the gearbox output shaft was 7.5 RPM, which would in turn require a minimum torque of 0.024Nm, as determined by the hand
calculations in Appendix D. Furthermore, the PTU horizontal turn was designed to rotate 180° in four seconds by project partners Strategic Engineering; the minimum speed of the gearbox output shaft was 15 RPM, which in turn required a minimum torque of 0.08Nm as shown by the hand calculations in Appendix E. Another issue which required a decision during the selection of the motors was that of backlash. Backlash in this system refers to the positional error of the PTU due to component clearance and flexure. The result of backlash was to introduce positional error into the encoder measurements. Backlash is undesirable in most situations and particularly in scenarios which require precision such as in the PTU design for this project. Hence in the decision process a motor/gearbox combination was chosen which had the required torque and RPM with minimal backlash. Among the possible motor selection, the minimum backlash available was 1°.

As shown in Table 3.5 the supplied motor and gearbox combinations meet the necessary torque, speed, nominal voltage, starting current and backlash error requirements. The torque of each motor is much higher than the required due to vibration and fiction in the system which will make it harder to turn. Hence, the supplied motor and gearbox combinations are acceptable for both the pan and tilt systems. For the tilt section, the MaxonMotors EC32 15W 24V Flat motor (Model number 267121), GP32 18:1 Planetary Gearhead (Model number 166159) and GSR25 10:1 Worm Gearhead were used. Originally for the pan section, the MaxonMotors EC32 15W 24V Flat motor (Model number 226006) and GP32 86:1 Planetary Gearhead (Model number 166167) were supplied. However due to problems with the motor in this combination and changes to the PTU design, it was no longer an option. Strategic Engineering supplied a second set of assembled pan motors consisting of the Maxon Motors Amax26 11W 24V (Model number 110963) and GP32A 28:1 Planetary Gearhead (Model number 166192). This motor and gearbox combination meets the
acceptable torque and speed requirements, while the nominal voltage, starting current and backlash error are all acceptable. This is shown in Table 3.5.

**ENCODERS**

The counts per turn of the PTU is to be maximised in order to increase the encoder accuracy, though most encoders have significantly more counts per turn than required. It is also an important requirement that the encoder be compatible with the respective motor and gearbox combination. The motors supplied by Strategic Engineering came with the MR type L Encoder (Model number 228452) for the tilt motor assembly and the ENC 22 Encoder (Model number 110521) for pan motor assembly. Both encoders are compatible with the corresponding motor and gearbox combination while having a high number of counts per turn.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Pan Assembly</th>
<th>Tilt Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Specifications</td>
<td>Acceptable Specifications</td>
<td>Actual Specifications</td>
</tr>
<tr>
<td>Assembly Weight</td>
<td>825g</td>
<td>281g</td>
</tr>
<tr>
<td>Assembly Length</td>
<td>104.3mm</td>
<td>81.2mm</td>
</tr>
<tr>
<td>Assembly Diameter</td>
<td>32/50mm</td>
<td>26mm</td>
</tr>
<tr>
<td>Torque</td>
<td>2.359Nm</td>
<td>0.3612Nm</td>
</tr>
<tr>
<td>Rotational Speed</td>
<td>15.56RPM</td>
<td>280.42RPM</td>
</tr>
<tr>
<td>Backlash Error</td>
<td>0.8°</td>
<td>0.8°</td>
</tr>
<tr>
<td>Nominal Voltage</td>
<td>24V</td>
<td>24V</td>
</tr>
<tr>
<td>Encoder Counts</td>
<td>500 counts per turn</td>
<td>100 counts per turn</td>
</tr>
<tr>
<td>Cost</td>
<td>$506.71</td>
<td>$237.18</td>
</tr>
</tbody>
</table>

Table 3.5: Summary table for the PTU tilt and pan motors assemblies (Maxon Program 09/10)

**3.1.4.4 PAN AND TILT MECHANISMS**

The pan and tilt mechanisms made use of bearings and motors mounted to relatively stationary parts to allow low friction rotation. The pan motor is bolted to a mount using the four available threaded holes in the end of the motor gearbox. This mount is bolted to the top of the PTU base using a series of countersunk bolts. The lid is mounted to the output shaft of the pan motor via custom coupling which is fitted over the output shaft and held there using two grub screws. This assembly can is displayed in Figure 3.14. A large tapered roller bearing was fitted over the coupling which will bear the load of the base lid and everything above. There is also a circular groove cut in the PTU base top for a ball thrust bearing between the lid and the top of the base to allow smoother rotation. This bearing supports minimal load.

The tilt motor is bolted to the u-shaped section using three countersunk bolts. The output shaft of the tilt motor serves as the horizontal shaft. There is a key way in the output shaft of the tilt motor and the LiDAR harness to transfer torque via a key. A grub screw passes through a hole in the
harness and rests against the motor output shaft to hold the shaft in place. On the other side of LiDAR harness a custom built shaft is fixed to the LiDAR harness using a grub screw in the same fashion as the other side. This shaft rests on a deepgrove bearing to allow low friction rotation.

The gearboxes in the tilt section were chosen with less than 1° of backlash however two gearboxes were used in this section along with a key to transfer torque. The resulting backlash in this section could not be known before testing. After the PTU was made and assembled this backlash was tested. A tortional spring was designed to reduce the effects of the backlash to within 1°. This spring was mounted between the LiDAR harness rubber stopper and the U-shaped section, this can be seen in Figure 3.15. This spring increased the amount of torque required to tilt, which in turn reduced the effects of backlash. The backlash angle still takes effect when tilting upwards however this angle is measurable and can be corrected in coding.

The Hall-effect sensor, power and encode cables for the pan motor can simply be fed straight down and out the bottom of the PTU base into the chassis with no exposure to the outside environment. This cannot be done with the tilt motor cables as the tilt motor is enclosed within a protective aluminium case. The unusual ribbon cable used for the Hall-effect sensor and power on the tilt motor necessitated the use of a ribbon to cable converter shown in Figure 3.16, which was mounted to the U-shaped bracket within the confines of the case. The resulting cable and the encoder cable were fed outside the box by use of a plugs mounted in the aluminium casing wall. From this external, waterproof cables are feed through holes in the UGV top to the motor controllers. This can also be seen in Figure 3.16.

**BEARINGS**

A tapered roller bearing was chosen for this design as it supports a large portion of the PTU in the axial direction. However, it is still required to take a similar load in the radial direction when
the PTU is being installed and moved. The bearing was required to support a load of 4.536kg, which can increase by approximately ten times due to vibration Ben Cazzolato (2010, pers. comm. February). This results in a dynamic force of 444.5N on the bearing. The bearing chosen was a 4T32005X produced by NTN bearings, it can be seen in Table 3.6 that this bearings specifications are well above the required. Maximum speed was not considered due to the extremely slow rotation speed. The calculations for this can be found in appendix E.

The bearing chosen for the tilt section was a simple deep groove bearing as it will be subjected to mainly axial forces with some radial forces. The bearing will be required to support 2.927kg,
which also can increase by ten times due to vibration, resulting in a 287.27 N dynamic load on the bearing. The bearing chosen was a 6001LLU deep grove bearing produced by NTN. It can be seen in Table 3.6 that this bearings specifications are also well above the required. Maximum speed was not considered for this bearing either, due the to slow rate of turn. The calculations for this can be found in appendix E.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Pan Bearing Requirements</th>
<th>Acceptable Specifications</th>
<th>Tilt Bearing Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static radial load rating</td>
<td>33.5kN</td>
<td>44.45N</td>
<td>2.39kN</td>
<td>28.72N</td>
</tr>
<tr>
<td>Dynamic radial load rating</td>
<td>27.8kN</td>
<td>444.5N</td>
<td>5.1kN</td>
<td>287.27N</td>
</tr>
</tbody>
</table>

### 3.1.4.5 FUTURE WORK

The main focus of any future work with the PTU should be on weight reduction. In future, the movement of the UGV will be better tuned and the PTU will be at less risk allowing the design to be made lighter. This could be done by reducing the thickness of parts and possibly selecting a lighter material for the structure.

### 3.2 COMMUNICATION AND COMPUTATION

Wireless communication is required for the UGVs to transfer information with each other and with the GCS to facilitate UGV cooperation and data amalgamation.

#### 3.2.1 COMMUNICATION

The requirements of the wireless communication system were a high bandwidth to allow camera images to be sent from UGVs to the GCS and a long communication range as the UGVs could be up to 500 metres away from the GCS in the MAGIC competition. The bandwidth is a measure of the rate of data transfer that is possible with the system. The frame rate and size of the images can be scaled to suit the capabilities of the wireless communications system but the higher the bandwidth, the lower the latency in tracking or identifying objects using the camera.

The wireless modem was selected by project partners, Strategic Engineering, who chose a multi antenna system capable of operating in either of the common frequencies ranges. Initially the unit chosen was the N-Tron 702-W wireless modem, as its 2.4GHZ and 5.8GHZ operating frequencies provide high bandwidth data transmission at a claimed range of up to 60km and a power consumption of 4.8W at 24VDC (N-Tron, n.d) means that the modem is low power and the 24V requirement is compatible with the UGV’s power board. The N-Tron 702-W wireless modem is also reliable, with a mean time between failures of above 1,000,000 hours (N-Tron, n.d), and it comes with a rugged enclosure that is specifically designed to operate in industrial environments (N-Tron, n.d).
A relationship with the supplier enabled the modems to be purchased at a discounted price. The specifications of the N-Tron RF modem are displayed in Table 3.7.

Table 3.7: Specifications of the wireless modems (N-Tron, n.d, Ubiquiti 2010a, Ubiquiti 2010b)

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Actual Specifications (N-Tron 702-W)</th>
<th>Actual Specifications (Ubiquiti RouterStation Pro with SR-71A)</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission range</td>
<td>Above 60km</td>
<td>Above 50km</td>
<td>500m</td>
</tr>
<tr>
<td>Data transfer rate</td>
<td>300Mb/s</td>
<td>300Mb/s</td>
<td>Min 100Mb/s</td>
</tr>
<tr>
<td>PoE requirements</td>
<td>48V</td>
<td>48V</td>
<td>Not required</td>
</tr>
<tr>
<td>Power consumption</td>
<td>4.8W</td>
<td>~6W</td>
<td>Max 20W</td>
</tr>
<tr>
<td>Voltage supply</td>
<td>20-49V</td>
<td>20-49V</td>
<td>≤24V</td>
</tr>
<tr>
<td>Weight</td>
<td>0.86kg</td>
<td>0.21kg</td>
<td>Max 1kg</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>2.4GHz, 5.8GHz</td>
<td>2.4 GHz, 5.8GHz</td>
<td>2.4GHz for IEEE 802.11 compatibility</td>
</tr>
<tr>
<td>Size</td>
<td>132 x 188 x 38 (H x W x D) mm</td>
<td>134x152x20 (H x W x D) mm</td>
<td>200 x 200 x 100 (H x W x D) mm</td>
</tr>
</tbody>
</table>

Testing of the N-Tron unit by Strategic Engineering after purchase showed that data rates achievable were much less than those claimed and that the provided firmware was not very reliable. Strategic Engineering could not achieve real world transfers at greater than 20Mb/s. In the hope of achieving better performance, these were returned to the supplier and replaced with the Ubiquiti RouterStation Pros with Ubiquiti SR-71A radio modems mounted in the on board mini-PCI slot (See Figure 3.17). The antennas chosen were the RF Shop 9dBi rubber ducky (DA-2450-09-01 - Figure 3.18) for the UGVs and the 7dBi magnetic mount desktop antenna (M22A) for the base station. The specifications of these new units are very similar to the N-Tron but data rates of up to 70Mb/s were achieved. This is still less than the 100Mb/s aimed for but is sufficient for the honours project at the University of Adelaide as only two UGVs have been built.

Apart from communication between UGVs and the GCS, on-board hardware also required communication. Ethernet was one of the chosen methods for data communication between some of the hardware and the Lex on-board computer. Since the computer chosen only had one ethernet port, a switch was required to provide more ethernet ports to communicate between the sensors and the computer. The devices that require an ethernet port are the Leuze Rod4, the Baumer camera and the Ubiquiti wireless modem. An ethernet switch does not perform any network management and is only required to provide connections. This is slightly different from the ethernet router, which is a device that manages the network traffic as well as providing switching capabilities. It is possible to include power in an ethernet cable, which is referred to as Power over Ethernet (PoE), reducing the number of external cables required. The camera will be mounted on top of the PTU and will therefore be in an exposed position. It was decided that reducing the number of cables to this


Figure 3.17: Ubiquiti RouterStation Pro (i4-wifi n.d.)

Figure 3.18: 9dBi Rubber Ducky Antenna used for wireless communication

device was very important. PoE systems transmit power as well as data over the same ethernet cable.

The unit chosen was a Baumer 4 Port PoE Gigabit Ethernet Switch, as shown in Figure 3.19, which is an industrial switch, with a low weight, size and power consumption (Baumer Optronic GmbH 2010). The Baumer ethernet switch was chosen specifically because it could be purchased together with the Baumer camera so that discounts were more easily negotiated. The specifications of the Baumer ethernet switch are displayed in Table 3.8. Additional information on the selection process of the ethernet switch can be found in Appendix B. The datasheet for this component can be found in Appendix D.
Table 3.8: Specifications of the Baumer 4 port switch (Baumer Optronic GmbH 2010)

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>1 GB/s</td>
<td>1 GB/s</td>
</tr>
<tr>
<td>Number of Ports</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Size</td>
<td>22x99x113.5 mm</td>
<td>&lt;200mm in any dimension</td>
</tr>
<tr>
<td>Weight</td>
<td>150g</td>
<td>&lt;400g</td>
</tr>
<tr>
<td>Power Requirements</td>
<td>24-48V, 6W</td>
<td>≤24V, ≤10W</td>
</tr>
</tbody>
</table>

The task of removing the UGV lid to access the Lex on-board computer was time consuming and was performed often. For this reason an external USB and ethernet port for the on-board computer (discussed in the net section) was added on the rear panel of the UGV. The selected USB port has model number PS-PO73-USB Type A and the ethernet port has model number 1457-RJ45 Jack. These ports are shown in Figure 3.20.

3.2.2 ON-BOARD COMPUTATION

Data processing was required both on-board the UGV and externally at the GCS. The main constraints for the processor on the UGV was that it needed to be small (laptop or smaller), and have a significant number of input and output ports. In addition, the processor needed to allow for more complex programming abilities than a micro-controller. Numerous factors were important in selecting a suitable form of processing data and controlling the UGVs on board, however the selection was significantly simplified by the generous donation of six embedded, integrated and fanless computer systems called a Lex box. Although the donation of these pieces of hardware was the governing factor in deciding on this particular on-board computer system, it was also a highly suitable selection for a number of reasons.

The Lex Box (as shown in Figure 3.21) was a compact and ideal solution to the required on-board computation and was made even more so by the fact that it was supplied for free. The actual and
required specifications for the Lex Box are displayed in Table 3.9. Additional information on the selection criteria of the Lex Box can be found in Appendix B. The datasheet for this component can be found in Appendix D.

3.3 Mobility and Electrical

In order for the UGVs to complete the desired goals successfully, it was necessary that all hardware required by the UGV was mobile and able to move as a single unit. This requirement drove the development of a chassis and drive system which would be able to mount and house all necessary hardware in a useable manner and be able to move this mounted hardware over a variety of commonly encountered terrain.

3.3.1 Drive System

The drive system used within each UGV is a six wheel drive configuration powered by electric motor assemblies. Motor controllers were employed to manipulate the motors and manoeuvre the UGV as required. Steering of the UGV was achieved via the skid steer method.
Table 3.9: Specifications of the Lex Box (Intel 2009, p. 6)

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Actual Specifications</th>
<th>Acceptable (min) Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel Atom N270 1.6GHz processor</td>
<td>Min Pentium 3 processor</td>
</tr>
<tr>
<td>Ports</td>
<td>1 LAN, 1 VGA, 6 COM, 2 USB</td>
<td>Min 1 LAN, 4 COM, 2 USB</td>
</tr>
<tr>
<td>Memory</td>
<td>On board DDRII module 1GB RAM, On-board SSD 2 GBbytes</td>
<td>Min 512 MB RAM, Min 512 MB HDD</td>
</tr>
<tr>
<td>Weight (kg), Size (mm)</td>
<td>1.8, 80x187x130</td>
<td>Max 3, Max Approx. 200 cube</td>
</tr>
<tr>
<td>Power (V)</td>
<td>On board DC +12±5% convert to +3/+5 for system</td>
<td>≤24</td>
</tr>
</tbody>
</table>

3.3.1.1 DRIVE METHOD AND STEERING

A six wheel drive design was utilized on each UGV. This design incorporated six separate motors driving each of the individual six wheels, as displayed in Figure 3.22a. Also considered was a two motor configuration that accommodated six wheels. This would consist of two motors, one on either side of the UGV, each driving three wheels using a timing belt (Figure 3.22b).

![Render CAD image of the six motor configuration](image1)

![Two motor configuration with timing belt](image2)

Figure 3.22: Possible UGV drive methods (Strategic Engineering 2009)

The drive system configuration that was chosen and used was the six motor configuration, which was selected by project partners, Strategic Engineering. The space occupied within the UGV is significantly less with a six motor configuration, as shown in Figure 3.22a. The advantage of space reduction was that the UGV chassis became more compact, helping to reduce weight.
3.3.1.2 DRIVE ASSEMBLY

The chosen method of driving meant that there were six drive assemblies per UGV. The drive assembly is a group of different components and parts that are responsible for the propulsion of the UGV. The drive assembly is comprised of the motor assembly, wheel, bearings and coupling. The motor assembly and wheels are the fundamental requirements for the drive assembly, providing propulsion through rotational motion of the wheels by the motors. The coupling acts as an attachment between the drive motor output shaft and the wheel as well as a means to attach the motor to the chassis.

WHEELS

Wheel selection was important in determining the torque and speed capacity required by the motor assembly. Knowing the geometry of the wheel is important to determine torque output and linear speed at the wheel. It was also important to determine the frictional force created by the tyres when being dragged over various surfaces. Knowing this frictional force value allows an estimate of the amount of torque needed to perform skid steering to be calculated. The UGV makes use of two types of tyres, these were the MBS T3 (Figure 3.23a) and the MBS Roadie tyres (Figure 3.23b).

![Figure 3.23: Types of tyres used on the UGV](image)

(a) MBS T3 black mountain board wheels  (b) MBS roadie mountain board wheels

Both these tyres had a 200mm outer diameter and 50mm thickness, however the MBS T3 tyres had a deep tread pattern whereas the MBS Roadie tyres were slicks. These wheels allow for sufficient grip on terrain and provides enough clearance between the UGV chassis and the ground. Physical testing was performed by dragging the tyres across various surfaces whilst carrying a known payload, and a spring gauge was used to measure the amount of force required to drag the payload. This testing was important to determine the frictional force created by the tyres and would allow the approximate torque required by the motor assembly to perform skid steering to be calculated. This test also determined the optimised tyre setup that required the least amount of motor torque to turn. Reducing the frictional force was desired as this would mean a smaller capacity motor assembly could be utilised, which would reduce the cost of the motor assembly as well as the power consumption. The tests were carried out with six MBS T3 tyres carrying a 40 kg payload. Another set of tests with the 40 kg payload were also performed with MBS Roadie tyres replacing the 4 outer MBS T3 tyres. The results of the testing are displayed in Table 3.10.
Table 3.10: Frictional force values of tyres carrying a 40kg payload (Maxon Motor 2009)

<table>
<thead>
<tr>
<th>Terrain</th>
<th>6 x MBS T3 tyres</th>
<th>4 x MBS Roadie and 2 x MBS T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carpet friction</td>
<td>245 N</td>
<td>181 N</td>
</tr>
<tr>
<td>Concrete friction</td>
<td>196 N</td>
<td>137 N</td>
</tr>
<tr>
<td>Linoleum friction</td>
<td>117 N</td>
<td>98 N</td>
</tr>
</tbody>
</table>

From the testing it was concluded that using slick tyres on the outer 4 wheels reduced the amount of frictional force created with the terrain by up to 30% in some cases. Hence the 2x MBS T3 and 4xMBS tyre setup was utilised on the UGVs.

**Motor Assembly**

The main constraint on motor assembly selection was the design specification to drive and manoeuvre the robot with a maximum load of 40kg using the skid steer method. Specific goals were required to be met in that the UGV had to be able to turn on a variety of surfaces including sealed and unsealed roads and paths, grassed and sandy areas; and achieve translational speeds of up to 10 kilometres per hour (kph). Based on the wheel geometry and friction force values, the torque of the drive assembly at the output shaft had to be greater than 1.89 Nm and the motor assembly output shaft had to be capable of a speed of 266 revolutions per minute (RPM). Detailed calculations can be seen in Appendix E. Torque and speed output of the motor assembly was the primary decision factor dictating motor selection in order to meet the set goals of turning and translational speed. In addition to these goals, the size, weight and cost of the motor assemblies was also minimised where possible.

The motor assembly was selected by project partners, Strategic Engineering, and the motor selected was the Maxon Motor EC 45 Flat brushless motor and the gearbox selected was Maxon Motor GS45A gearbox with a gear ratio of 18:1. The selected motor assembly is displayed in Figure 3.24 and the motor assembly properties are shown in Table 3.11.
Table 3.11: Specifications of the chosen motor assembly

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max output speed</td>
<td>292 RPM</td>
<td>266 RPM</td>
</tr>
<tr>
<td>Max output torque (Nm)</td>
<td>1.98 Nm</td>
<td>1.89 Nm</td>
</tr>
<tr>
<td>Power (Continuous)</td>
<td>18.0 V 3.53 A</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>334 g</td>
<td>Minimise</td>
</tr>
<tr>
<td>Cost</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As displayed in Table 3.11 the motor assembly meets the torque and speed requirements and hence was an acceptable choice. In addition to meeting the requirements, these motor assemblies were also able to be purchased at a significant discount due to a sponsorship agreement with Maxon Motor.

During testing, it was found that there were numerous problems with the wheel motors which include,

- gears coming loose resulting in no torque being transferred to the gearbox
- motor and gearbox assemblies coming loose
- bearings for the gearbox shaft being damaged
- faulty encoders.

To resolve the issue of the gear coming loose, ball peening was used to deform the hole in the gear to create an interference fit which holds the gear on its shaft. Some motors have become unusable and new motors have been ordered. It is unknown what caused these problems, however it is important to note that due to these problems that delayed the testing phase of the project, the desired amount of testing could not be done.

**WHEEL COUPLINGS**

The wheel couplings were designed to secure the wheel to the drive motor shaft, house the bearings and as a means of fastening the drive motor assembly to the chassis. Since there are six wheels, each UGV will require six coupling assemblies. The coupling assembly consists of a sleeve shaft, bearing hub and a gearbox mount.

The sleeve shaft is attached to the motor assembly output shaft with a grub screw and attached the wheel with a standard M6 screw. The coupling transfers torque to the wheel with the use of key. A labelled diagram of the drive assembly is shown in Figure 3.25.

Initially the sleeve shaft was fixed to the motor assembly output driveshaft with the use of a 3mm grub screw. However it was found during physical testing of the UGV that the grub screws were failing. Figure 3.26 displays the fracture surface of one of the grubscrews that failed during testing.

Inspection of the fracture surface by an expert concluded that grub screws were failing due to excess stress and not from fatigue. This was concluded due to the shiny fracture surface and lack of striations. To reduce the shear stress, the diameter of the grub screw was increased to 4 mm. This would increase the area over which the torque force would act and hence reduce stress. The
change in diameter results in a calculated 57% reduction in shear stress within the grub screw when transmitting torque from the motor.

The bearing hub was used to house the bearings and forms one half of a clamp which is used to fix the drive assembly to the chassis. The gearbox mount is attached to the motor assembly with four M3 screws and also acts as the other side of the clamp. The bearing hub and gearbox mount are fastened to each other and the chassis with eight M4 screws. An exploded view of the described coupling is displayed in Figure 3.27.
Bearings are required for smooth operation of the sleeve shaft connected to the motor and to minimize wear between the sleeve shaft and the chassis. Roller bearings were selected mainly because they are compatible with the bearing hub and motor shaft. The bearings are subjected to a radial load due to the contact forces between the sleeve shaft and the bearings, which is distributed across all 12 bearings. The load requirement for the bearings are that each bearing must be able to sustain a radial load of 327N, the calculation of which is included in Appendix D. The bearings that have been selected are 6001LLU deep groove bearings manufactured by NTN. The desired and actual specifications of the 6001LLU deep groove bearings are displayed in Table 3.12 where it can be seen that the selected bearings surpass requirements. The static load requirements were calculated by considering the 40kg weight limit of the UGV as specified by the MAGIC competition requirements (DSTO, 2009, p. 11), and a personal communication with Ben Cazzolato (2010, pers. comm. February) revealed that dynamic loading is typically ten times that of a static load. A CAD model of the 6001LLU roller bearings is shown in Figure 3.28.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum speed</td>
<td>18000 RPM</td>
<td>6000 RPM</td>
</tr>
<tr>
<td>Static radial load rating</td>
<td>2.39kN</td>
<td>32.7N</td>
</tr>
<tr>
<td>Dynamic radial load rating</td>
<td>5.1kN</td>
<td>0.327kN</td>
</tr>
</tbody>
</table>
Encoders are measuring devices which are attached to motors for the purpose of obtaining odometry information. The encoders were mainly selected with regard to compatibility with the drive motors (hence from Maxon Motors range) and the angular resolution that they could provide, where a high resolution is desirable for localisation of the UGV. During the design process, it was determined by project partners, Strategic Engineering, that at least 250 counts per turn would be suitable. The encoders selected have a resolution of 500 counts per turn, which satisfies the UGVs requirements. The encoders selected were Maxon Motors L type encoder with three channels, as shown in Figure 3.29.

In order to control the motors, an appropriate motor controller is required. The motor supplier, Maxon Motor, provided their own controller for the motors, known as an Electronic Positioning (EPOS) unit. An EPOS motor controller was chosen since it is best suited to the Maxon Motor equipment already selected for use on the UGVs. An EPOS2 24/5 motor controller was selected, which is shown in Figure 3.30, where one motor controller is required for each motor. There were two models of EPOS motor controllers to choose from, which includes the old model EPOS and the new model referred to as EPOS2. The EPOS2 model was chosen over the old model because the EPOS2 model offers high efficiency, gateway function USB - to - CAN, 12 bit resolution instead of 10 bit while still maintaining the same dimensions as the old model (Maxon Motor 2007). The minimum and actual specifications of the EPOS2 24/5 motor controller is shown in Table 3.13 and an image of the EPOS2 motor controller is displayed in Figure 3.30.
All of the sensory and drive equipment required a supply of electrical power. This section discusses requirements and subsequent selection of the power supply implemented on the UGV.

3.3.2 POWER SUPPLY

Since the UGV must be mobile, all electrical power needs had to be supplied by a portable source. The diverse range of components that required electrical power also meant that the power supply had to provide electrical power at a number of different voltages. The selected power source also had to have the capacity to run all equipment simultaneously for a minimum of 45 minutes. Safety measures had to be implemented within the circuitry to allow quick termination of electrical power at the user’s discretion and also to prevent damage to hardware in the event of a fault.

3.3.2.2 POWER SOURCE

The UGVs required a portable power source and hence, lithium-ion batteries were selected due to high energy density and relative light weight properties. This was beneficial since there was a weight budget of 40kg. Lead acid batteries were considered during the selection process for their robustness and affordability, however due to the weight restriction being 40kg this type of battery was deemed unsuitable since they were much heavier, and would likely cause the UGVs to exceed the weight restriction. The battery cell type selected was the Sanyo UR18650W cell with a built in safety circuit to prevent excessive discharge of the battery. Six lots of 7x3 cell packs are connected together in series to achieve the voltage required. The actual and required specifications of the batteries are listed in Table 3.14. Some of the key requirements of the batteries is that
they must be capable of a maximum voltage of 24V and a maximum current draw of 30A. The minimum acceptable current draw was calculated by considering the nominal current draw of all UGV components, and the minimum acceptable capacity was calculated assuming an approximate current requirement of 30A at 24V for 45 minutes, which is the minimum run time desired. The selected batteries has a maximum voltage of 24V and a maximum current draw of 60A which can be seen in Table 3.14. The minimum and actual specifications of the power source required for the UGVs are shown in Table 3.14, and an image of the battery cell is shown in Figure 3.31.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage (V)</td>
<td>24</td>
<td>24 for EPOS compatibility</td>
</tr>
<tr>
<td>Maximum Current (A)</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>Capacity (A-h)</td>
<td>45</td>
<td>28</td>
</tr>
</tbody>
</table>

Figure 3.31: Sanyo UR18650W Battery Pack (Siomar Battery Industries, n.d)

**Power Management Hardware**

Due to the variety of hardware that was required to be powered by the onboard power source, a number of supply voltages had to be made available.

The variation in power requirements of all the hardware meant that three supply voltages had to be made available. These voltages were 24V, 12V and 5V. To achieve each of these voltages, DC/DC converters were used to step down the voltages as required. A DC/DC converter was used for each of the three voltages an example of a DC/DC converter can be seen in Figure 3.32. Although the power source is 24V by design, a 24V DC/DC converter was used to smooth the variance in voltage that would be experienced from direct connection to the battery. All 24V hardware used the 24V DC/DC power supply excluding the motor controllers which had a direct connection to the batteries. The implementation of these DC/DC converters into electrical circuit is further discussed in Section 3.3.6. The datasheet for these components can be found in Appendix D.

**Battery Charger**

The battery charger was required to be specific to Lithium-ion batteries, since the charging requirements of Lithium-ion batteries are very different to charging requirements for other common
battery types. Battery damage or even explosions can occur if the batteries are not correctly charged. Careful consideration of charging the batteries occurred before the UGV was finalised and ready for testing. The battery charger selected was the Innovative Energies SR100LI SmartCharger, since it was compatible with the selected batteries. The SR100LI SmartCharger is shown in Figure 3.33.

The chassis provides a means of attaching and housing all of the required hardware in a robust way whilst providing sufficient protection to vulnerable equipment and ease of access to frequently used items.

### SAFETY MEASURES

Several safety measures were implemented on the UGV. These safety measures allow manipulation of the electrical circuitry to shut off hardware. The safety measures implemented in the UGV were in the form of an E-stop and circuit breakers.

The E-stop allows the UGV motor assemblies to be disabled at the operators discretion. The e-stop activates the quick stop function in the motor controller firmware rendering the motor assemblies inoperable. The e-stop is actuated via a button mounted to the top of the UGV, and is displayed in Figure 3.34.

The electrical circuitry also contains circuit breakers as a safety measure to protect the hardware in the event of a battery fault or short circuit. The implementation of this protection will be discussed further in Section 3.3.6

### 3.3.3 UGV CHASSIS

The chassis provides a means of attaching and housing all of the required hardware in a robust way whilst providing sufficient protection to vulnerable equipment and ease of access to frequently used items.
The chassis is required to have sufficient room to mount all hardware. Some of the hardware that needed to be housed was vulnerable to water, and as the UGV is required to operate in moist conditions such as light rain, the chassis must be able to provide protection for these components. The chassis also needs to shield fragile equipment from impacts, as there is a possibility that the UGV could collide with other objects as it explores unknown territory. The chassis is also required to be as light as possible due to the weight restriction on each UGV. The chassis is estimated to be one of the heaviest parts of the UGV and hence, the panel sizes and thicknesses must be optimized in order to save as much weight as possible. Also the design had to be simple to manufacture in order to keep manufacturing times and costs to a minimum. Thus the design was to be as functional as possible at low costs. Since the UGV would be operating in an urban environment, it was important that the chassis dimensions were such that the UGV could still fit through a standard doorway of approximately 900mm. The chassis also had to be designed such that mounted hardware did not overheat and malfunction.

3.3.3.2 CHASSIS MATERIAL
The material selected to make the chassis was aluminium as it proved to be the most suitable material for this application. A in-depth discussion about why aluminium was chosen can be found in Appendix C.

3.3.3.3 CHASSIS GEOMETRY
The design of the chassis geometry was of high importance since it not only affects the rigidity of the chassis structure itself, but impacts on the level of protection of the interior and exterior components, also affects vehicle dynamics and dictates hardware layout.

The main focus of the chassis design was simplicity. Utilizing a simple design would allow the manufacturing and material costs to be reduced. The primary function of the chassis was hardware storage and consequently much of the design was dictated by this factor. Due to the strict scope of the chassis design and the limits on manufacturing resources, a basic and functional platform design was conceived. The final chassis design, as seen in Figure 3.35

The final chassis geometry is a rectangular prism consisting of 7 panels. This design incorporated a completely sealable internal volume for the housing of hardware which was to be protected from the elements, whilst also providing a large surface area on the top for the mounting the PTU and
various antennae. The flat walled design allowed all hardware parts to be mounted easily so no additional mounting had to be manufactured to accommodate any complex curved surfaces in the chassis. The dimensions of the chassis are 490 x 320 x 200mm. These dimensions were driven by the size of the hardware needing to be accommodated within the internal volume and by a desire to minimise interference with obstacles encountered in the environment. The values of the length and width of the chassis were chosen to allow the UGV to easily manoeuvre through a standard door way. The side and front chassis panels as well as the portion of the lid bearing the load of the PTU were 6.35mm (1/4”) wall thickness. The other portion of the lid and the bottom panel were a wall thickness of 3.18mm (1/8”). This thickness was a compromise between weight, panel deflection and chassis rigidity. The panels was assembled together using M3 screws which were fastened into the thickness of the panels.

3.3.3.4 ALTE RATIONS

Some mechanical modifications were made to the chassis panels with an aim to reduce the total weight of the chassis. Reducing the total mass of the UGV would have numerous benefits including decreasing motor capacity and power consumption which allows for increased run time for the selected power source. These modifications were made in the form of reducing the wall thickness of certain sections of the chassis panels. These sections of the 6.35mm panels were milled to a depth of 4mm, as shown in Figure 3.34.

A 15mm buffer of unmilled material was left around the edge of the panels and all mounting holes. These buffers were left to act as stiffeners to help prevent deflection of the panels under loading by leaving regions of increased stiffness throughout the panel. Table 3.15 displays the volumes of each of the chassis panels before and after the milling process and subsequent total weights assuming an aluminium density of 2700 kg/m$^3$. The volume of the panel also took into account any significant holes that had been drilled through the chassis panels. From Table 3.15 it can be seen that milling the panel resulting in a total weight reduction of 1.34 kg.
3.34 HARDWARE LAYOUT

The selected hardware had to be arranged in the UGV in a manner in which all hardware was able to operate most effectively. Accessibility of hardware by the operator and reducing cable lengths also had an influence on hardware placement.
3.3.4.1 **EXTERNAL HARDWARE**

This section discusses each piece of hardware that was mounted externally. Figure 3.37 shows the hardware locations graphically.

![Figure 3.37: Rendered image of external hardware layout](image)

The PTU had to be mounted externally due to the nature of the devices that The PTU holds. The PTU was mounted towards the front of the UGV upon the fixed section of the top panel to avoid movement when the lid was being opened. The PTU was mounted sufficiently far enough away from other obstructions to allow for the full range of movement of the PTU. A number of items were attached to the PTU to utilise the movement capabilities, these items included the LiDAR, camera and laser. The LiDAR was mounted on the PTU to make use of the horizontal rotation capacities allowing the LiDAR to achieve a full 360 degree range view. Mounting the LiDAR to the PTU also raised the LiDAR above any potential obstructions in the form of other hardware mounted to the lid of the UGV. Both the camera and laser were mounted to the PTU to allow target tracking and subsequent neutralisation of detected OOI. Both the dGPS and wireless modem antenna were mounted externally on the UGV to transmit and receive data more effectively. Due to the nature of the emergency stop (E-stop) button it was required to be in a position where it could be easily actuated in case of an emergency. The position deemed to be the most easily accessible was on top of the UGV. The ultrasonic sensors were to be attached to the front of the UGV to allow detection of close proximity obstacles out of range of the LiDAR.
3.3.4.2 **INTERNAL HARDWARE**

This section discusses each piece of hardware that was mounted internally. Figure 3.38 shows the hardware locations graphically.

![Internal Hardware Layout](image)

Figure 3.38: Rendered CAD image of internal hardware layout

Each of the 6 drive motor assemblies were internally mounted on to the side panels of the UGV to minimise shaft length and volume occupied. Components that were anticipated to need frequent access were mounted towards the rear of the UGV. Components identified as to needing frequent access were the onboard computer, wireless unit and GPS unit. Mounting components towards the rear meant that these components would be more easily accessible through the lighter weight rear portion of the lid. The power source was one of the heaviest items to be carried by the UGV, weighing approximately 6kg in total. Ideally the overall centre of mass of the UGV was to coincide with the geometric centre of the UGV to reduce the yaw moment of inertia, thereby reducing the turning torque required and hence reducing power consumption. So to centralise the overall weight of the UGV as much as possible the batteries were mounted towards the rear of the internal space of the UGV, to offset the weight of the PTU mounted towards the front of the UGV. This battery positioning also allows easy access through the rear portion of the lid, which allows for quick changes if the batteries need to be replaced. Some of the hardware had the ability to be mounted to a standardised DIN rail and so this capability was utilised. Two 250mm lengths of DIN rail were bolted to either side wall in the front portion of the UGV and the Ethernet switch, terminal blocks and circuit breakers were attached. In order to minimise cable length from the motor controllers to...
the drive and PTU motors, the motor controllers were mounted towards the front of the UGV. To avoid magnetometer interference from the magnetic fields generated by the motors and electronics within the chassis the IMU was mounted the furtherest distance possible from motors. Ideally the IMU would be mounted externally to be far away from the motors, however since the IMU was not water proof it could not be mounted externally hence it was mounted underneath the PTU panel. The IMU was mounted in the geometric centre of the UGV for ease of acceleration calculations. There were three DC/DC converters of different voltages mounted within the UGV, specifically a 24V, 12V and 5V. Access to the converters was not required after installation and hence each of the converters was mounted low down on the side panels of the UGV. The positioning of each converter within the UGV was dictated by the circuit in which the converter was a part of.

### 3.3.5 Future Work for Chassis

The chassis implemented on the UGV achieves all desired goals set. However further improvements could be made to the functional aspects of the chassis, specifically to improve the chassis accessibility. Currently if the UGV is completely sealed there is significant turnaround time to unseal the UGV chassis and then reassemble it. This could be improved by the addition of a quick release lid, an example of what could be designed is shown Figure 3.38.

![Figure 3.38: Rendered image of possible quick release lid design](image)

### 3.3.6 Power Management and Data Connections

This section will discuss the design used to transmit power from the batteries to all the hardware chosen and data to and from all the components. As previously mentioned in Section 3.3.2.2, not
all the hardware runs on 24V power which necessitates the use of DC/DC converters to step down the power, before supplying it to certain components. This was combined with the use of terminal blocks to separate the main circuit into different voltage requirements of the hardware. This section will also discuss the layout and type of data connections used between the different components. There are several requirements of the electrical design of the UGV for ease of use and component and human safety.

As per the contract and for ease of use a very accessible on/off switch was required to be mounted to the outside of the UGV controlling power to all hardware. As previously mentioned in Section 3.3.2.2, circuit breakers were also required to protect the hardware. An easy way to charge the batteries using the battery charger will they are mounted in the UGV chassis was also necessary to simplify the charging process. Additionally the batteries were required to have some way of disconnecting them from the electrical circuit to allow the system to be disassembled/reassembled without remaking the circuit. The wires used must have a current rating over that which they will be required to supply. Also, the wiring and cabling must be minimised and positioned such that it does not interfere with the accessibility of the components mounted within the UGV.

### 3.3.6.1 ELECTRICAL CONNECTIONS

The basic electrical design runs from the batteries, through the main on/off switch, to a central terminal block. From this terminal block the circuit is split into four separate parallel circuits. Three of which run through the DC/DC converters making two 24V circuits, one 12V and one 5V. As previously discussed in Section 3.3.2.2, no DC/DC converter was used when powering the motor controllers and motors. Additionally to protect the hardware from damage circuit breakers and fuses were used. A 65A circuit breaker was placed in the 24V circuit without a DC/DC converter, and a 25A circuit breaker was placed where the other three circuits split. A fuse was fitted on the LiDAR as it was regularly powered off other power supplies during testing. Each of the four circuits in parallel power a separate terminal block. From this all hardware can be connected to the terminal block with its relative voltage. All electrical wiring was performed with two gauges of wire with American Wiring Gauges (AWG) of 12 and 19. The maximum current loading of 12 and 19 AWG wire is 9.3A and 1.8A respectively. All wiring before the separate voltage circuits, due to the high current in these sections, was done using 12AWG wire. The wiring of the four separate circuits and power supply, due to the reduction in current in this section, was done using 19AWG wire. Quick release connections were fitted to the battery leads, so the batteries could be quickly disconnected from the main UGV circuit when needed. The voltage and current requirements of each piece of hardware along with the positioning of the fuses and circuit breakers can be seen in Figure 3.40. The electrical drawings can be found in Appendix C.

### 3.3.6.2 DATA CONNECTIONS

The basic design of the data cabling layout involves all sensors connecting to the Lex on-board computer, the camera and LiDAR doing so via the four port ethernet hub, to process information and give out new commands. Figure 3.41 shows this layout and the types of connections between hardware. Details of the hardware interfaces and connections to the Lex box can be found in Table 3.16. The data connection drawings can be found in Appendix C.
UGV Power Connections Flowchart

Batteries (Lithium Ion)
24V, 18A, 6 lots of 5x3 cells

65A Circuit Breaker

EPOS Motor Controllers

Motors
Tilt: 24V DC, 1A max
Pan: 24V DC, 2A max
Drive: 24V DC, 8A

DC/DC Converter
24V - 24V

LIDAR
24V DC, 1.5A

Fuse
24V DC, 1.5A

4 Port
24V DC, 1A, POE

Wireless Comm
24V DC, 200mA max

Lex Box
12V DC, 5A

DC/DC Converter
24V - 12V

Camera
24V, 180mA, POE

LIDAR
24V DC, 1.5A

Camera
24V, 180mA, POE

Lex Box
12V DC, 5A

DC/DC Converter
24V - 5V

dGPS
12V DC, 750 mA

IMU
5V DC, 5mA

Ultrasounds
5V DC, 3.4mA

Level 4 Flowchart

POE - Power over Ethernet
DC - Direct Current

Figure 3.40: Flowchart showing all electrical connections between the hardware in the UGV
Figure 3.4: Flowchart showing all data connections between the hardware in the UGV.
Table 3.16: Summary of hardware interfaces and connections

<table>
<thead>
<tr>
<th>Device</th>
<th>Required Interface</th>
<th>Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPOS Motor Controllers (for PTU)</td>
<td>CAN</td>
<td>miniPCI for EPOS Controller</td>
</tr>
<tr>
<td>EPOS Motor Controllers (Drive motors)</td>
<td>CAN</td>
<td>Dual Row Female Receptacle (4 poles)</td>
</tr>
<tr>
<td>Motor Encoders</td>
<td>EIA 422</td>
<td>DIN41651 (10 poles) (connected directly to EPOS unit)</td>
</tr>
<tr>
<td>Microstrain IMU</td>
<td>RS232</td>
<td>mini DB9, (Cable included)</td>
</tr>
<tr>
<td>Novatel dGPS</td>
<td>RS232</td>
<td>DB9 (Crossover cable, included)</td>
</tr>
<tr>
<td>Ultrasonics</td>
<td>RS232</td>
<td>Solder Pads</td>
</tr>
<tr>
<td>GPS Antenna</td>
<td>RG-58, 50Ohm Impedance</td>
<td>TNC (connected directly to GPS unit)</td>
</tr>
<tr>
<td>Baumer 4 Port Ethernet Hub</td>
<td>Ethernet</td>
<td>8P8C (Cat 6 Cable)</td>
</tr>
<tr>
<td>Baumer Camera</td>
<td>Ethernet</td>
<td>8P8C (Cat 6 Cable)</td>
</tr>
<tr>
<td>Leuze ROD4 LiDAR Sensor</td>
<td>Ethernet</td>
<td>M12 Socket, D coded, 4 Pin</td>
</tr>
<tr>
<td>Ubiquiti Wireless Modem</td>
<td>Ethernet</td>
<td>8P8C (Cat 6 Cable)</td>
</tr>
<tr>
<td>Ubiquiti Antennas</td>
<td>RG-58, 50Ohm Impedance</td>
<td>RP-SMA (connected directly to N-Tron Modem)</td>
</tr>
<tr>
<td>Arduino Duemilanove Microcontroller</td>
<td>USB</td>
<td>USB</td>
</tr>
</tbody>
</table>

3.4 FINAL UGV DESIGN

The final UGV design consists of a combination of the selected hardware arranged and mounted carefully within the custom made chassis and PTU. The successful integration of hardware enabled the UGV to fulfil the requirements of manoeuvrability, localisation, identification of OOI (Objects Of Interest) and communication to other UGVs and the Ground Control Station (GCS).

Driven by six individual motor assemblies, the UGV has the capability of attaining a maximum speed of 10km/h with high manoeuvrability directed through skid steering, enabling full rotation on the spot. Powered by six portable rechargeable batteries, the UGV has over 3 hours of normal operation time due to the high energy density of the lithium ion batteries used. The modular design of the aluminium chassis enables ease of encasing hardware in a compact and robust manner. Dimensions of the chassis optimised volume and considered ease of passing through standard doorways (See Figure 3.42).

During travel, the location of the UGV relative to the GCS is achieved through a dGPS with a maximum positional accuracy of up to 2cm. However, in cases of signal failure that can occur inside dense buildings, an IMU was utilised to meet the positioning requirements when in these situations. The PTU mounted on top of the UGV provides a full horizontal scanning range and a 70°
vertical tilt range for the camera, LiDAR and laser. The horizontal scanning range has been reduced to 70° from 90° to avoid damaging wires. The colour camera captures visual information of the surroundings, detecting environmental features and both stationary and moving OOIIs. The LiDAR calculates the distance of these objects from the UGV with a range of up to 65m. The GCS utilises this information to create a variety of different maps of the explored area. The laser simulates a weapon for neutralisation and is activated upon successful identification of a threat. Following identification of OOIIs with the use of the camera, the information of the object’s location is sent to the GCS through wireless antennas. This process is displayed in Figure 3.43.

Figure 3.43: As an object is detected by the UGV, the information of the object’s location is sent to the GCS wirelessly
This process is accelerated through the use of multiple UGVs that can communicate with each other through the same wireless antennas used for communication with the GCS. Multiple UGVs coordinate across the field so that exploration of the area is achieved more efficiently. It allows an area beyond the wireless range between a UGV and GCS to be explored by using a second UGV as a medium to transfer information. This is depicted in Figure 3.4.

![Figure 3.4: A second UGV is used as a medium to transfer information between the GCS and a UGV beyond the wireless range](image)

An on-board computer in the UGV enables it to perform these intelligent tasks which would not be possible through a microcontroller. Additionally, as the on-board computer is capable of handling advanced programming, the UGV may be customised to perform different tasks for various applications. A summary of the UGV’s key specifications are listed in Table 3.1.

### 3.4.1 Future Work

The UGV design has been completed successfully, however there are areas in which the design may be further improved. Currently, the UGV has a variety of fasteners of different lengths and sizes. This can be simplified to a smaller set of fasteners of a common size and length. Since the design was intended to be for Defence purposes it would be likely that the vehicles will end up operating in harsh terrain. Hence, the chassis can be improve with the use of additional rubber sealing to prevent dust and water from entering and potentially damage expensive hardware. Currently the design is splash proof and the level of improvement required will depend on the terrain that it is expected to operate in. Another issue to consider is weight reduction of the UGV chassis and PTU frame to reduce the power consumption of the motors and allow for longer operation times. The wheel coupling should also be investigated to improve reliability. During testing it was discovered that the interior of the UGV became quite warm and hence, the addition of temperature sensors inside the UGV chassis to monitor the interior temperature is advised. Vibration is another problem that was found during testing over uneven terrain. This may be detrimental to the sensors over long periods of time and will affect accuracy of measurements.
Table 3.17: Summary of UGV Specifications

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions (mm)</td>
<td>$600 \times 470 \times 630$</td>
</tr>
<tr>
<td>Weight</td>
<td>36.5kg</td>
</tr>
<tr>
<td>Maximum speed (electronically limited)</td>
<td>10 km/h</td>
</tr>
<tr>
<td>Operating time</td>
<td>$&gt;3$ hours</td>
</tr>
<tr>
<td>Wireless communication range to GCS in open field</td>
<td>200m</td>
</tr>
<tr>
<td>Wireless communication range to GCS in urban environment</td>
<td>50m</td>
</tr>
<tr>
<td>Maximum operating temperature</td>
<td>55ºC</td>
</tr>
<tr>
<td>Maximum turning torque</td>
<td>39.06Nm</td>
</tr>
<tr>
<td>Total maximum traction force</td>
<td>179N</td>
</tr>
<tr>
<td>Maximum object detection range</td>
<td>65m</td>
</tr>
<tr>
<td>Maximum horizontal/vertical scan range</td>
<td>360º/70º</td>
</tr>
<tr>
<td>Total cost to reproduce one UGV excluding labour</td>
<td>$26296$</td>
</tr>
<tr>
<td>Total expenses for one UGV excluding labour</td>
<td>$17500$</td>
</tr>
</tbody>
</table>
The Ground Control Station (GCS) provides a central coordination point for the main control of the team of Unmanned Ground Vehicles (UGVs) in the field. Through the assistance of a Human Machine Interface (HMI), users can view individual UGV actions or take control and manually operate the UGV’s actions, including the selection of waypoints for the UGV to follow. The GCS also provides the ability to view various maps, computer vision and localisation aspects of the UGVs as they explore the field.

The GCS achieves this by regularly receiving information from the team of UGVs on field through wireless communication. If a single UGV is out of the wireless communication range, the data is sent to the GCS through another UGV that is within range of both the out-of-range UGV and the GCS. This is shown in Figure 4.1.

![Figure 4.1: UGV 1 sending information to GCS through UGV 2](image)

### 4.1 GCS HARDWARE

The GCS has the primary function of controlling and displaying the various actions of the UGVs as well as displaying all the mapping information generated. In order to achieve these tasks, a means of communication between the GCS and the UGVs was required. A visual display and peripherals were also required, so that the user can interact with the UGVs through the HMI. Additionally, the GCS acts as a medium for all data transfer. Data is sent to the GCS and fused together prior to redistribution to the team of UGVs, so each UGV receives updated mapping, localisation and control data regularly. A further requirement was for the GCS to provide a base station that would transmit correction values to the GPS system as described in Section 2.3.1. This
Figure 4.2: Level Two GCS Hardware Flowchart

GCS Hardware Flowchart

GCS Computer

Remote E-Stop

SECTION: 4.1.2
LCD Screen (22 inch)

SECTION: 4.1.1
Touch Screen (22 inch)

SECTION: 4.1.1.2
Direct Camera Displays

SECTION: 4.1.1.1
Map Displays

SECTION: 4.1.3
Program/Code Display

SECTION: 4.1.4
User Input

SECTION: 4.1.5
Wireless Communication

dGPS

Waypoint Selection

Remote E-Stop

GCS Hardware Flowchart

Level 2 Flowchart
would enhance the ability of the GPS system to triangulate the position of the UGV accurately and is a necessary component of the differential global positioning system (dGPS) being utilised for localisation purposes. The dGPS relates the known fixed location of the GCS location to the less well known variable locations of the moving UGVs.

A selection of hardware was required to achieve these functions. Figure 4.2 shows how the GCS was implemented.

### 4.1.1 Visual Displays

Visual display of acquired data such as maps, machine vision and control Graphical User Interface (GUI) was a requirement of the MAGIC competition rules. Initially, the visual displays had to cater for 8-10 individual UGVs, hence, requiring a simultaneous display of 8 camera feeds. Failure to progress to the final stage of the MAGIC 2010 competition reduced the required total number of UGVs to two. Therefore, currently only two camera feeds need to be displayed on the GCS so the user no longer needs to switch between displays as frequently.

In addition to camera feeds, two physical and three conceptual, individual maps are created by each UGV which are then combined into global maps requiring display on the GCS (Refer to Chapter 8). With other requirements, such as having the capability of alternating between manual, semi- and fully autonomous control as well as being able to quickly apply a remote emergency freeze or stop functions for each UGV, the amount of display space required was significant. Hence a balance of size, cost and functionality was desired when selecting the most suitable visual display.

The desire for functionality, such as the ability to swiftly scan through maps or individual UGV camera feeds, made a touch screen a viable option. A touch screen would enable quick changing and viewing of multiple maps and camera feeds, toggle between different control types (manual, semi or fully autonomous control), as well as inherently provide an emergency stop functionality without the need for extra peripherals.

### 4.1.1.1 Touch Screen

The touch screen was purchased based upon the best available price for an industry standard piece of equipment as there were no special requirements. A compromise between size and cost was required, and hence the largest touch screen that could be purchased within a budget of $2000 was obtained. At the time of purchase this was found to be a 22inch NEC AS221 Capacitive Touch Screen (Shown in Figure 4.3) from Touchscreen Solutions.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions (mm)</td>
<td>505.2 × 390.9 × 221.8</td>
<td>&lt;600 × 400 × 250</td>
</tr>
<tr>
<td>Weight</td>
<td>5kg</td>
<td>&gt;5kg</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>27W</td>
<td>&lt;50W</td>
</tr>
<tr>
<td>Response time</td>
<td>5ms</td>
<td>&lt;10ms</td>
</tr>
<tr>
<td>Viewing Angle</td>
<td>176°(H) / 176°(V)</td>
<td>&gt;150°(H) / 130°(V)</td>
</tr>
<tr>
<td>Price</td>
<td>$1192.50</td>
<td>&lt;$2000</td>
</tr>
</tbody>
</table>
Not all of the display space was required to be interactive, so it was possible to supplement the touch screen with at least one Liquid Crystal Display (LCD) screen to provide additional visual display space at reduced cost. It was decided that at least two LCD screens were required; one to display the camera feeds and one to display miscellaneous items such as UGV status, error logs and other items required for UGV control. To retain uniformity with the display screens, the LCD screens were selected to match the size and style of the touch screen already purchased. Hence, two 22 inch NEC AS221 Non Touch Monitors (Shown in Figure 4.3) were also bought from Touchscreen Solutions.

Table 4.2: Specifications of the NEC AS221 Touchscreen (NEC n.d)

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions (mm)</td>
<td>505.2 × 390.9 × 221.8</td>
<td>&lt;600 × 400 × 250</td>
</tr>
<tr>
<td>Weight</td>
<td>5kg</td>
<td>&gt;5kg</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>24W</td>
<td>&lt;50W</td>
</tr>
<tr>
<td>Response time</td>
<td>5ms</td>
<td>&lt;10ms</td>
</tr>
<tr>
<td>Viewing Angle</td>
<td>176°(H) / 176°(V)</td>
<td>&gt;150°(H) / 130°(V)</td>
</tr>
<tr>
<td>Price</td>
<td>$379</td>
<td>&lt;$400</td>
</tr>
</tbody>
</table>

4.1.2 GCS COMPUTER

The GCS Computer was selected based on performance, the number and types of connections available, cost and past experience. The University of Adelaide has an agreement with DELL, and the IT support staff for the School of Mechanical Engineering desire uniformity among the university’s computers, hence a DELL computer was selected. The number of peripherals required a minimum of four USB2.0 ports, an ethernet port and a serial/RS232 port for the dGPS.

The selection criteria included efficient handling of camera and mapping data received through the network, and the associated calculations for mapping and image processing. Mapping and
computer vision can require intense processing capabilities. Although programming efficiency is a large part of the efficiency of the GCS computer, the limitations of the hardware capabilities would be a major restriction on the performance of the UGVs. Therefore, a computer with high computational capability was desired without spending extensive amounts of money on a super computer. Hence, the final choice was a Dell Precision™ T3500 (shown in Figure4.5) containing an Intel® Xeon® W3530 (2.8GHz) Quad-core CPU with a maximum Turbo frequency of 3.06GHz. The computer also has four gigabytes (GB) of Random Access Memory (RAM), to deal with the large amounts of incoming data (as well as 9.4GB of swap space). The 500 GB hard-drive was selected to store test run data and can be further augmented in the future. The screens are fed by dual ATI FirePro™ V5700 graphics cards with capabilities suitable for multiple screen outputs. The GCS was installed with the Ubuntu 10.04 (Lucid) operating system, selected for compatibility with the Ubuntu systems installed on each of the UGVs. The performance specifications of this computer were deemed to be sufficient, to ensure that hardware limitations would not be an issue when considering the overall efficiency of the network system. All further efficiency issues should be repairable through software implementation.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions (mm)</td>
<td>447 × 172 × 468</td>
</tr>
<tr>
<td>Weight</td>
<td>17.7kg</td>
</tr>
<tr>
<td>Processors</td>
<td>Intel® Xeon® W3530 (2.8GHz) Quad Core (Turbo 3.06GHz)</td>
</tr>
<tr>
<td>Memory</td>
<td>4GB</td>
</tr>
<tr>
<td>Hard Drive</td>
<td>500GB</td>
</tr>
<tr>
<td>Graphic Cards</td>
<td>2x Dual ATI FirePro™ V5700</td>
</tr>
<tr>
<td>Price</td>
<td>$2031.20</td>
</tr>
</tbody>
</table>
As with the use of most computers, a keyboard and mouse was required. In this case, no special specifications were desired, hence, a standard universal serial bus (USB) keyboard and optical mouse were used. However, due to some of the inherent functions of the GCS, other peripherals were required. One of these is the touch screen (discussed in Section 4.1.1), which enables interactive viewing and control of certain aspects of the HMI, including way-point selection. A further peripheral that was required for manual control was a joystick or game controller, for ease in manually controlling the UGVs. Initially, a standard joystick was used to develop the drivers; however, this was then superseded by a wired PlayStation™ controller for a more portable solution. Later, a wireless generic game pad was made available at no cost and hence, this replaced the PlayStation™ controller for increased portability. Unfortunately, further testing showed that the game controllers were inducing a lag in the system that could not be resolved, and hence, the controls reverted back to being keyboard control.

A wireless communication system was required in order to send and receive data from the UGVs. All hardware required for this aspect of the GCS is discussed in detail in Chapter 3 Section 2.

In order to achieve a differential global position system (dGPS) for localisation of the UGVs it was required to have a dGPS unit and antenna setup on the GCS. All hardware required for this aspect of the GCS is discussed in detail in B.1.7.

The hardware within the GCS required connections to operate. The GCS computer was the main hardware through which all the connections were input as it controlled many of these hardware. This can be seen in Figure 4.6.
4.2 GCS MECHANICAL CASE

The GCS was made up of a number of bulky items and a significant amount of wiring. It was desirable to be able to avoid set up and dismantling of all of this hardware every time it was transported, as this would be a time consuming and awkward process. Hence, a mechanical station was required for the protection and mounting of the selected equipment, as well as providing a suitable workstation for user operation. The workstation was designed to be compact and portable so that the user may quickly set up the GCS on field in a variety of situations. Hence, custom roadcases were explored for this application as a viable solution to the design and build of a GCS mechanical case.

4.2.1 REQUIREMENTS

The requirements of the GCS mechanical case were as follows:

- During GCS operation, all monitors are viewable and peripherals accessible
- Portable and easy to handle
- dGPS and wireless antennas located at positions that minimise signal interference
- Encases all hardware and allows fixed interior cabling
- Provides protection for all GCS equipment
• Easy to setup GCS on field in a short time
• Provides a comfortable workstation for the user

The most challenging of these requirements was mounting the three monitors compactly within the case during transport but allowing the user ease of access to them during operation. In order to achieve these two requirements simultaneously, adjustable monitor arms were explored.

In positioning the dGPS and wireless antennas, maximum distance between these two antennas was desired so that signal interference is minimised during operation. Additionally, the position must provide clearance from any other objects or walls to optimise reception.

To ensure all hardware may be packed in the case, the arrangement of the equipment was vital to the design process. Cabling and wires connecting hardware also needed to be considered. Equipment had to be sufficiently protected during transport; and therefore, foam and methods of fixing the hardware in place were investigated.

The GCS would need to be designed such that setting up would be easy and not time consuming. A multiple power port was considered, so that the user may just plug in a single power plug to start the GCS.

The GCS is a workstation and may be used for considerable amounts of time, hence, the user will need to be able to use the GCS comfortably. Ergonomics was considered in the design stages to ensure this requirement was met.

4.2.2 Final Design

There were many initial designs that were iterated to achieve the final successful design. They are discussed in further detail in Appendix A. To optimise space within the case, the computer was placed horizontally with the wireless and dGPS units. A slide-out tray was incorporated into the design for placement of peripherals. The three monitors employ a multiple monitor mount to enable compact positioning within the case when closed, and expansion for viewability when opened. A PRIMA mount was selected for this as it was highly customisable and provided the mounting of multiple monitors on a single arm (see Figure 4.7). The PRIMA mount was available.

Figure 4.7: Top view of the final design
in a number of configurations; however, the quad mount was purchased so that the dGPS may be mounted on the fourth arm. The touch screen is positioned in a portrait orientation on the centre arm during the closed state as this optimised space effectively within the case. When expanded, this touch screen can swivel to the landscape orientation if necessary. Both the wireless and dGPS antennas needed open space for better signal quality and hence, the wireless antennas are attached to the back of the side monitor, and the dGPS on the fourth arm can extend away from the wireless antennas providing maximum clearance.

The arrangement of the hardware optimised space within the case, creating a very portable design for the GCS. This can be seen in Figure 4.8a where the user is able to utilise the GCS in a cross-legged position. However, if the top lid was placed underneath the main frame of the case, the height is increased sufficiently for a user to sit on a chair and operate the GCS comfortably. This is shown in Figure 4.8b.

![Figure 4.8: Different positions for the user to operate the GCS comfortably](image)

Castor wheels were added for better ease in handling the GCS and acted to secure the main frame when placed on top of the inverted top lid. The wheels lie inside the inverted top lid, preventing sliding of the main frame whilst on top of the inverted top lid. The front two wheels allow braking to prevent undesired rolling when the main frame is on the ground. Side handles were also added for improved handling. Spanning valence latches (see Figure 4.9) were used to fasten the removal front panel and top lid on the main frame of the case.

![Figure 4.9: Spanning Valence Latch](image)
To protect equipment inside the case during travel, foam lining of 20mm thickness was placed in the interior sides. For ease of access to cables, a rear access hatch was designed through which a single power cable can exit. Plywood was used in the manufacture of the case as it provided sufficient strength for the purpose of protecting the hardware encased from handling during travel while reducing the overall weight when compared to various heavier materials. The total weight of the GCS was 74.7kg as shown in Table 4.4. More detailed accounting of the weight and costs may be found in Appendix B.

The total cost of the GCS was $10,060.34 which includes all hardware, case and monitor mount. The complete GCS was successfully built and is shown in Figure 4.10.

![Custom built GCS case](image)

**Figure 4.10: Custom built GCS case**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions when closed (mm)</td>
<td>726 × 658 × 1144</td>
</tr>
<tr>
<td>Dimensions when fully opened (mm)</td>
<td>800 × 1000 × 1600</td>
</tr>
<tr>
<td>Weight</td>
<td>74.7kg</td>
</tr>
<tr>
<td>Cost</td>
<td>$10060.34</td>
</tr>
</tbody>
</table>

**Table 4.4: Specifications of the GCS**
4.3 FUTURE WORK

The GCS may be further developed with the addition of new technology such as augmented reality, which can allow the user to take the perspective of the UGV through the camera view in an environment more stimulating and immersive than viewing on a conventional 2D display monitor. This is achieved through the use of Head Mounted Displays (HMD) that are goggles with an infrared detector attached. Johnny Chung Lee brings an example of a successful application of this with the use of a Wii remote and the sensor bar (Lee 2010). Fixing the sensor bar to the head of the user through glasses or goggles and placing the Wii remote at the base of the monitor, enables the user to immerse in a 3D view of the screen and closely emulate a real environment, so that moving left and right bends the view in those directions respectively.

For the application of the GCS, it is possible to develop it such that the user is able to view the environment on field through the perspective of the camera on the UGV. Mechanically, the GCS monitor mount may be improved. According to manufacturer’s specifications, the monitor mount was designed to allow for mounting up to three monitors on a single connector for monitors up to 5kg. However, in practice, sliding of the connector occurred, and hence, a bolt was drilled through the main mounting support pole to remedy this but disables the vertical adjustability feature of the mount. To allow the desired vertical adjustability of the monitor mount, grooves or bolt holes at equal distances along the bar should be made to allow the monitors to be mounted at those defined heights.
Part II

SOFTWARE
Software is an integral part of using the Unmanned Ground Vehicle (UGV) design to its maximum potential. It is key to ensuring a complete and effective system and is the means with which autonomous control, localisation, computer vision and mapping are implemented. As part of the system of flowcharts used for the MAGIC final year project, Figure 5.1 shows the high level software architecture for the UGV and shows the communication requirements between the UGVs and Ground Control Station (GCS). There are a mixture of functions performed by the UGV and GCS. The localisation, basic mapping and position control of the UGVs are all handled by the on-board computer. Systems that induce a large computational burden or rely on combining data from multiple UGVs are performed by the GCS. These systems include object detection and identification, mapping and high level path planning and collaboration. Unlike the sourcing of hardware for this project, most of the software required has been specially designed and implemented for the specific requirements of the MAGIC project. While the system is based on a common operating system, there is a large focus on creating efficient and effective software at all levels, from hardware drivers through to high level mapping and control.

Figure 5.1: Level one software flowchart
5.1 **SYSTEM SPECIFICATIONS**

The general specifications of each system to be implemented are developed out of the initial design work performed through the creation of flowcharts. The scope of each subsystem is defined so as to clearly define boundaries between each subsystem, while also completing all functional requirements of the overall system. Table 5.1 lists the scope of each sub-system required on the UGV and Table 5.2 lists the scope of the sub-systems implemented on the GCS.

Table 5.1: Key functions of the onboard computer.

<table>
<thead>
<tr>
<th>Function</th>
<th>Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Localisation</td>
<td>Kalman Filter, for outdoor localisation, using measurements from GPS and IMU. SLAM for indoor localisation, using measurements from the LiDAR and IMU.</td>
</tr>
<tr>
<td>Objects of Interest (OOI) Identification</td>
<td>Capture of images from the camera, for sending to the base station. Automatic exposure and gain control to ensure the images are clear and meaningful.</td>
</tr>
<tr>
<td>Mapping</td>
<td>Capture LiDAR readings to transmit to the base station.</td>
</tr>
<tr>
<td>Network management</td>
<td>Send and receive information required for UGV and system operation.</td>
</tr>
<tr>
<td>Control</td>
<td>Control the position of the UGV and the PTU, according to the instructions received from the GCS.</td>
</tr>
<tr>
<td>E-stop</td>
<td>Check for signal from base station and stop UGV if required.</td>
</tr>
<tr>
<td>Collision Prevention</td>
<td>Use measurements from LiDAR and Ultrasonic Sensors, to ensure that collisions do not occur in case of error by the control algorithm.</td>
</tr>
</tbody>
</table>

Table 5.2: Key functions of the ground control station.

<table>
<thead>
<tr>
<th>Function</th>
<th>Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping</td>
<td>Process LiDAR readings and use the data to generate maps for display on the HMI.</td>
</tr>
<tr>
<td>Objects of Interest (OOI) Identification</td>
<td>Detect, identify and locate desired objects within the images captured by the UGV and display images and map locations on the HMI.</td>
</tr>
<tr>
<td>Human Machine Interface (HMI)</td>
<td>Display maps and UGV status information. Provide an interface to control and track the UGVs.</td>
</tr>
<tr>
<td>Network management</td>
<td>Send and receive information required for UGV and system operation.</td>
</tr>
<tr>
<td>Control</td>
<td>Perform path planning, using information developed from the Mapping sub system or user input.</td>
</tr>
</tbody>
</table>
5.2 OPERATING SYSTEM

The operating system (OS) is the basis for all software onboard the UGV and also on the GCS. The operating system is required to provide a stable base for communications with sensors and other computers. For the requirements of a completely autonomous UGV, a real time operating system is ideal to give the developer full control over the allocation of resources and processing power. Key aspects of real time operating systems include the ability for multi-tasking, assigning priority levels to processes and thread-safe Inter Process Communication (IPC). Multi-tasking is required as a robust method of implementing multiple sub systems concurrently on board both the UGV and GCS and the ability to assign application’s priorities provides the developer control over the importance and scheduling of each task. Inter Process Communication is required to transfer information between different processes running on any computer.

Early on in the MAGIC project, Strategic Engineering chose to use QNX as the operating system on-board the UGV. QNX is a very scaleable, modular operating system that is targeted at the real-time embedded systems market. It offers a large range of features including POSIX compliance, a proprietory message passing system and very small processing requirements due to its inherent modularity. A short discussion of some more details of QNX are included in Appendix G. As this project offered the possibility for significant media exposure and was non-commercial in nature, a number of free licences for QNX and some training were provided by Symmetry Innovations. All initial prototyping of the UGV software was performed using QNX. The development environment and operating system seemed tricky to use and often posed issues when implementing software. Some of the provided system tools seemed to display erratic or unusual behaviour and the provided development environment did not generate projects suited to collaborative development, making this very difficult to use by anyone developing software (Marias 2010, pers. comm. 27 Sept). Also, all software that could be used on the system needed to be compiled specifically for QNX, and QNX drivers for sensors were rarely provided by the equipment manufacturers. Trials of the camera driver on the UGV on-board computer revealed that the Linux drivers supplied by Baumer would not work on the QNX OS. It was discovered that Prosilica seems to be the only company that supports GigE Vision cameras on QNX (Total Turnkey Solutions 2007) and the Prosilica drivers did not support Baumer cameras. For these reasons, QNX was not satisfactory for use in the project.

A new operating system was required for the UGVs that was supported by the camera driver, offered better compatibility with existing Linux tools and easier implementation through known software development tools. Ubuntu was chosen to run on-board the UGV computer, because it is compatible with the camera driver, but also was much better known amongst the people involved with the project (Marias 2010, pers. comm. 27 Sept). There are a large number of tools and tutorials available for Ubuntu systems, and hence, Ubuntu significantly reduced implementation time. Ubuntu has a very refined Kernel and is well supported by the open source community, as demonstrated by its relatively large user base (StatOwl 2010). Ubuntu does not, as standard, offer true real time capability, like that available in QNX, but this was deemed unnecessary after the June down selection. None of the capabilities for prioritisation had been required within QNX and the principle focus of the development was on academic research instead of the implementation of a complete system where the real-time issues would have greater relevance.

The standard installation of Ubuntu includes many features not required by the UGV, like a graphical user interface, some games and tools. As these are not required for the project, the server installation of Ubuntu was used. This has fewer features and consequently operates faster and has a smaller footprint on the harddrive of the computer. As the onboard computers only have 2GB harddrives, this was a significant concern. Further pruning of the system could remove some unnecessary tools, to increase hard drive space available for other purposes. It is also possible to use
a real-time kernel with Ubuntu, which would offer some improvements in performance, specifically related to speed of interaction with other hardware (Ubuntu 2010). The real-time kernel also allows for one process to pre-empt another. This means that a higher priority process will always stop a lower priority process, rather than the operating process only changing when the CPU is released by the lower priority process. This is not necessary for the current software used as no true real-time deadlines have been established.

Ubuntu was chosen early on in the project as the operating system to be used on the GCS. It was chosen primarily as it is a very stable Linux release and supports 64-bit processors. In desktop environments, Ubuntu is renowned for its ease of use, is similar to Windows for those with little experience of Unix systems and it has been used by members of the project team before. Ubuntu has support for both OpenCV and QT which are being used for Human Machine Interface (HMI) development. A more detailed discussion of the choice of operating system is included in Appendix G.

5.2.1 INTER PROCESS COMMUNICATION

One aspect that is key to the interoperability of sub systems on the UGV and for reasonable abstraction, between the device drivers and the sub systems, is Inter Process Communication (IPC). IPC is primarily for the transferring of information between the processes on board one computer but is also required to transfer data across the wireless network. The popular forms of IPC for real time operation are shared memory, message queues, pipes, file in file out and sockets (Yerraballi N.D., p3). As it is the standard method for communication across a network, sockets using the TCP (Transmission Control Protocol) was chosen for communication between the UGVs and the GCS. This has a proven ability in its use as the data transmission protocol for many services on the Internet. It is very reliable because all information sent needs to be acknowledged by the receiving computer.

During the initial prototyping of the system, message passing was used for inter process communication. As this is one of the key features of QNX and is a very capable system for stable and fast IPC (Yerraballi N.D., p3), this was appropriate. Message passing effectively transfers information as a single packet between a client and a server and is typically one directional. The packet is typically a data structure that contains all the related information. The key advantage of message passing for a real time system is that messages are inherently synchronised, meaning that there are never situations when different processes are trying to simultaneously read and write the same data. It is also a very effective method to implement an event driven software architecture and reduces overhead by removing any requirement for polling. Message passing was perceived to prevent easy communication between multiple processes in situations where more than one process wanted to access the same sensor data. No clear data structures were defined in the early stages of the project and hence these systems were setup by ad-hoc means. This hindered the integration of different project systems and hence a new solution to IPC was required.

To fulfill the requirements of IPC, Strategic Engineering developed a C++ library called mpi or message passing interface. This library uses a safe shared memory implementation for communication on one UGV and sockets for communication between the UGV and GCS. Shared memory systems directly access the computers RAM to store data and must be carefully managed to prevent errors due to simultaneous read and write attempts. The mpi also provides a common method for accessing data at both the UGV and GCS. All shared memory on each UGV is transferred to the GCS and stored in local shared memory on the GCS also. This requires the operation of a process onboard each UGV and the GCS to handle the transfer of information across the network. The mpi library provides the capability for a programmer with limited experience to utilise shared memory, without
having to worry about issues like synchronisation and scheduling. The library also handles all transfer of network traffic, significantly simplifying the design process for the students involved in the project. As such, all drivers for use onboard the UGV have now been written to use the mpi.

5.3 UGV SOFTWARE

The functions required in software on board the UGV relate to the low level information gathering and control processes required. These elements typically involve interacting directly with some pieces of hardware. Several subsystems operate onboard the UGV including localisation and control. These are explained in Sections 6 and 8. The other function of the software on the UGV is to receive and process sensor data, hence drivers have been written for each relevant piece of hardware and these are used to read the raw data from the sensors and write it into shared memory, using the mpi library. A full set of processes required is shown in Figure 5.2.

5.3.1 MOTOR CONTROL

The controllers used for all of the motors onboard the UGV were EPOS2 24/5 Programmable all-in-one positioning controllers from Maxon motors. The EPOS units are programmable positioning controllers and are compatible with DC brush motors with encoders and brushless EC motors with hall sensors and encoders (Maxon Motor, 2010). The EPOS controller offers current control, position control and velocity control. The EPOS controllers were set up to control the drive motors using velocity control and position control for the tilt and pan. This was implemented with Proportional, Integral and Derivative (PID) control. The EPOS controller comes with a Graphical User Interface (GUI) known as EPOS Studio.

There were a number of steps required to set up the motors. The first step involved the actual initialisation and designation, in firmware of the motor controllers, of what motors were being used, as different motors were being used for driving, panning and tilting. The second step involved tuning the motors and setting up what method of control was desired for each particular motor. The final step involved writing a driver onboard the UGV to be able to control all the motors together for manual and various levels of autonomous control.

5.3.1.1 STEP 1: MOTOR SETUP

There were three different motors being used on the UGV. They each required different setup which was enabled by using a setup wizard on EPOS studio. This included setting many parameters including the nominal and maximum current, the number of pole pairs and maximum speed. For further information on the specific setup of the motors and the setting required see Appendix G.

5.3.1.2 STEP 2: PID TUNING

In order to achieve accurate velocity and position control, the controllers needed to be tuned. Tuning accounts for the actual requirements of the installation of the motors and alters the gains used in the feedback controller. This ensures that the motors are driven with accurate velocity or position control, with a good compromise between overshoot and rise time. The controller was tuned using EPOS studio. EPOS studio offered two options to tune the controller. The first option was Auto tune; in this option, the EPOS controller uses a test signal to analyse the plant being controlled and attempts to automatically assign appropriate controller gains. The Auto tune option is a quick method which obtains reasonable control results. However, for finer tuning the second option was
Figure 5.2: Level Two Software Flowchart
available known as *Expert tune*. In this method, the user is able to manually tune according to specific requirements.

**Drive Motors**

The drive motors were tuned for velocity control. However, difficulty arose with tuning the drive motors from the fact that not all of the motors could be tuned simultaneously using EPOS studio and hence they had to be tuned individually and without any weight on them. This meant that the tuning was inherently going to be less effective than if the UGV was tuned by putting the UGV on the floor and auto tuning with the weight of the vehicle taken into account. Due to this issue the controller was auto tuned knowing that the plant response in real situations would be less than desired. Future work should be conducted to improve the tuning method and provide an expert tune with better gains however time did not allow this to be investigated.

**Tilt Motor**

The tilt motor was tuned for position control. However, due to the limited range of the tilt angle available with all the hardware attached, an auto tune could not be conducted without removing all the hardware and hence being presented with a similar problem as the drive motors whereby unrealistic PID values were acquired, because the weights and torques of the attached hardware could not be experienced while auto tuning. This meant that an expert tune was required. Unfortunately when testing was conducted on the tilt motor, it was unknown that the motor being used was damaged and did not possess enough torque to tilt with the hardware attached. Hence the hardware was removed and an auto tune was conducted before replacing the hardware. This enabled the hardware to be tilted although with significantly less torque than expected and with insufficient PID gains. Finally when the motor was discovered as being damaged time did not permit a specialised expert tune to be conducted however the auto tuning method had provided gains that, although not ideal, were sufficient to tilt.

**Pan Motor**

The pan motor is currently still waiting on specialised cable components from the motor providers. This was due to the supplier making a mistake and providing the wrong motor initially. The pan motor will be required to be set up in position mode and will, unlike the other motors, be able to be auto tuned with the actual load on it. This will require the cables attached to the pan/tilt unit (PTU) to be disconnected so that it may freely spin fully about its axis. This may still need further expert tuning to refine the PID gains however the auto tuning method is likely to be sufficient.

### 5.3.1.3 Step 3: EPOS Driver

The EPOS driver was primarily written by industry partners Strategic Engineering. It is written in C++ and is designed to directly interact with the eight motor controllers. Once the CAN network has been set up successfully between the motor controllers, they can all be interfaced through a single RS232 connection and the firmware on the EPOS controllers handles all flow control and signals on the CAN bus.

Once the RS232 port has been initialised, communication with the motor controller is achieved through the reading and writing of objects. An object is a packet containing a header byte, the node ID of the recipient, the data and the checksum of the object. The header typically contains a hexadecimal command byte and there are hundreds of different control commands available for setting different modes and methods of operation on the motor controllers. Before this packet can be sent, an acknowledgement is required from the EPOS that it is ready to receive data and this is repeated after the packet is sent also. Receiving data from the motor controllers is very similar,
an acknowledgement is sent from the motor controller, which must be responded to and then the packet is received before another acknowledgement is exchanged. This is shown in Figure 5.3.

The method of the communication used by the EPOS controllers is very robust but also somewhat complex and the use of hexadecimal control data packets, means it is not easily understood by humans. This significantly complicated the development of the driver and hence it is incomplete. It requires EPOS studio to enable the EPOS units before it is run for the first time. It is also not able to receive feedback information from the motor controllers, meaning there is no encoder feedback for localization purposes. Thirdly, the requirement to initially set-up and tune the motors in EPOS studio is bad practice because the driver is not robust to hardware failures and means that firmware changes are difficult to implement. By adding the capability to alter the firmware settings in the EPOS driver would allow for a complete manual tune of the drive motors so that the best possible tuning could be achieved.

The E-Stop was also achieved through settings in the EPOS driver and EPOS firmware. The emergency stop button (Section 3.3.2.2) is used in case of a loss of control by the human or autonomous operator and is connected to a digital pin on each motor controller. This pin is activated each time the EPOS driver is started, so that when the button is pushed, all motors stop operating.

The EPOS driver is designed to be utilized via another piece of software, dependent on the type of control being implemented. This is typically achieved by inheritance of the EPOS driver class but could also be performed by threading, where the EPOS driver is run in a separate thread to the calculation of control commands allowing for a somewhat faster response. To test the motors and movement of the vehicle and to assist in the testing of other subsystems, various forms of manual control have used the EPOS driver code. This has included implementation of:

- Control via keyboard commands. This allowed an operator, logged into the on-board computer, to control the velocity of the wheels and turn the vehicle.
- Control via a wireless gamepad. This was much more convenient to use than the keyboard commands but suffered from a very significant level of lag, which made it very difficult to control.
- Object detection using the LiDAR and concurrent response of slowing/stopping at designated intervals.

Further details of the planned control algorithms are included in Section 9.

### 5.3.2 Ultrasonic Driver

The ultrasonic driver is written in C++ and is used for RS232 serial port configuration, communication with the ultrasonic sensors, and checking for errors in the data transmitted from the ultrasonic sensors to the on-board computer. The serial ports used for communication with the two ultrasonic sensors are configured for read only data transmission as no data is sent to the sensors, and a data transmission rate of 9600 baud. The driver obtains data from the sensors at a rate of 10Hz in the form of ASCII packets containing range data in centimetres and start and stop characters. The data packets transmitted from the sensors are checked for errors by checking that the start and stop ASCII characters of every packet are the capital letter ‘R’ and a blank space respectively. If it is determined that a packet contains an error then the data received is not processed, otherwise the data is read and range values are determined. Out of range measurements are represented as either minimum or maximum range values depending on whether the object detected is too close or too far away respectively for the ultrasonics to give accurate range values. The firmware used within
the ultrasonic sensors chosen cannot be configured and the sensors do not need to be calibrated (MaxBotix, n.d). The operation process of the ultrasonic driver is displayed in Figure 5.4.

Although the basics of this driver has been implemented, no integration has occured with the control subsystems on board the vehicle. This has also become difficult as some of the serial ports on the onboard computer have failed. A possible future implementation would be to read the ultrasonic range measurements through the I/O Unit or a seperate microcontroller. This reduces the number of serial ports required and the computational burden on the on board computer.

5.3.3 LiDAR DRIVER

The LiDAR driver is used to communicate with the LiDAR and receives range data that is used by the mapping and indoor localisation processes. The LiDAR driver was written in C++ by Strategic Engineering but some debugging was also done by students in Adelaide. The driver receives data from the LiDAR through a TCP/IP Ethernet connection at a rate of 25 data scans per second. Once the data is received it is converted to a range in millimetres, which is stored in a 529 element array.
of unsigned short integers. When a complete data set has been scanned, it is written into shared memory, using the mpi library. This makes the data available both on the GCS for mapping and on the UGV for indoor localisation. The process of operation of the driver is shown in Figure 5.5 and the code to achieve this is include in Appendix H. Results of LiDAR testing are included in Section 8.

The firmware of the LiDAR provides on-board data processing and filtering to generate the distance measurements, in millimetres, from the raw sensor data. The firmware can be configured in regards to the data transfer protocol, measurement range, angular resolution, number of scans transferred per second and IP address. The data transmission protocols supported by the LiDAR’s firmware are ASCII and binary. The binary protocol is the default data transmission protocol stored in firmware and the LiDAR will return to using this every time it is turned off, hence to mitigate the effect of unexpected sensor reset or power failure during operation the binary protocol was chosen. The default measurement range of the LiDAR is the device’s maximum range of 65m. All the results scanned up to the 65m are used by the mapping processes. The default values of angular resolution and number of scans transferred per second are 0.36 degrees and 25 scans per second respectively, which are also the maximum values. The angular resolution and number of scans transferred per second were left as maximum values as a high resolution and data transfer rate were desired. The IP address of the LiDAR can be configured in firmware. Leuze publishes full specifications of their firmware (Leuze n.d.).

Figure 5.4: Operation of the MaxBotix ultrasonics driver.
5.3 UGV SOFTWARE

The IMU Driver is used to set up and receive data from the IMU, for use by the indoor and outdoor localisation processes to determine the UGV’s pose. The IMU is interfaced using an RS232 port on the Lex box. As dictated by the requirements of the Kalman filter, the IMU must be set in an appropriate mode to receive the data required and this can be set up to occur synchronously or asynchronously. When the data has been received from the serial port, it must be converted into a meaningful form before being written to shared memory.

The Microstrain IMU chosen has a large number of possible modes of operation depending on the requirements of the end user. These modes give many options in terms of the data returned, including accelerations and rotational velocities (Microstrain 2009, p7). There are also various options in terms of the extent of the correction or filtering performed on-board by the IMU. To reduce the development time, it was decided that a mode should be chosen that offers the most reliable and noise-free data, as this would require less tuning of the localisation processes to be sufficient. The data that is required to develop a dynamic model to be used by the localisation processes is the forward acceleration and the roll, pitch and yaw of the UGV. Some control functions are also improved by measurements of the angular rates of the vehicle and so it is preferable to track this data also. All modes include a timestamp using the clock on the IMU. The mode chosen was Acceleration, Angular Rate Vectors and the Orientation Matrix as the Orientation Matrix can very easily be converted to roll, pitch and yaw. This mode was the only mode that fulfilled the requirements of the localisation systems and therefore was the obvious choice.

The other principle design decision in the implementation of the IMU driver is whether to set up the IMU for continuous or polling-based operation. As the software architecture used is primarily event driven, the continuous mode is much more suitable. This means that when the IMU is turned
on, it is set to continuously send new data at a fixed data rate. When this new data is ready, it is then immediately available to any processes on the UGV. The other option is to poll the IMU every time a new reading is required. This will be slower than continuous mode and can waste useful data because the computer does not poll at the correct moment in time. As such this mode would always be slower than continuous operation but has the advantage that different data could be received every time the sensor is polled. The Microstrain IMU has a continuous mode, that will be used for this project to maximise the useful data received from the IMU.

Data communication with the IMU occurs in the form of binary strings of data sent to and from the IMU. As far as the end user is concerned these can be dealt with as hexadecimal numbers. For example, the command byte to send to the IMU is 0xC8 for Acceleration, Angular Rate Vectors and Orientation Matrix mode. Data is then returned as a series of bytes, of type char, that must then be converted in to the variable type that the data is stored in. All data returned from the IMU in the mode chosen is of type float and so each value returned has a length of four bytes. To convert these four characters to a float is straightforward. Each byte is cast as a float and then the sum of the four bytes is taken. Any data conversion requirements can be handled by the computer on board before the data is sent to the localisation processes. When data is ready from the IMU, it is written to shared memory as shown in Figure 5.

A very important part of the setup of the firmware of the IMU was to perform a hard and soft iron calibration. As the IMU combines measurements from a magnetometer and a gyroscope to measure orientation, any magnetic fields not caused by the earth prevent the IMU from operating correctly. Magnetic fields are typically caused by metallic objects and electricity. To reduce the effect of these, a calibration is performed to measure the fixed magnetic field created by the UGV. This needs to be performed in an area that is away from external metallic objects that would affect the calibration. This was performed by using Microstrain’s provided Hard and Soft Iron Calibration tool and the resulting correction calculated is stored on board the device. This calibration ensures that heading measurements are accurate when there are no other significant magnetic fields affecting the device.

As there are significant magnets fields generated by changes in electric current and running motors, the effect of turning the motors on and off upon the orientation estimate was also tested. The effect of the magnetic field created by any electronics other than the motors is mitigated by the iron calibration, as devices other than the motors draw a relatively fixed amount of current. To test the effect of the motors operating, the UGV was placed on a stand with the wheel raised. The motors were then operated as though the UGV was going forwards, backwards, turning left and turning right. The results of this are shown in Figure 5.7. The effect of the motors on the heading measurement from the IMU is noticeable but still insignificant for the requirements of the project. The standard deviation of this measurement when the motors are stationary is 0.02 degrees and when the motors are running is 0.05 degrees.

5.3.5 DGPS DRIVER

The dGPS driver is required to both receive corrections sent from the GPS base stations and also to receive pose data from the GPS. These two functions are relatively separate and use separate RS232 ports on the on-board computer. The information that is required from the GPS is the Easting, Northing, Velocity and Heading of the robot and also the variance of all of these measurements. This information is then available for localisation purposes. The differential correction must be received from the base station, through the wireless network, to the UGV, and then sent to the GPS.
The implementation of a dGPS network at the site of operation is significantly more complex than using a standard GPS receiver. It is not straightforward to accurately determine the position of a unit to use as the base station and a method of sending corrections is required. The position of the base station can be found by averaging the estimated position over a length of time but to achieve good accuracy this should be done for 24 hours before use (Novatel n.d., p73). This period of time was not available before the MAGIC challenge and so a compromise was required. If the time period is reduced, the position of the base station is known less accurately but this should not affect the precision of the GPS units. As most localisation and mapping occurs relative to the position of the base station, this was not be a problem. For the MAGIC UGVs designed, a wireless network was used for sending data. Since this is not the typical method of sending corrections, which are normally sent using a low frequency radio modem, there was an added level of complexity. The set up of the dGPS system is shown in Figure 5.8.

Receiving the POSE data from the GPS is relatively simple. Communications using the RS232 port require a string to be sent to the port and this will trigger a reply. This is the same as the IMU driver except the user has the option to send the string in ASCII or Binary (Novatel 2009, p19). The advantage of the binary strings are that fields contained within the messages are always a
fixed length, so they can be reliably interpreted from a fixed position in the string of data returned (Cullen et al. 2007, p113). The advantage of ASCII strings is that they can be easily understood by human and so are easier to implement in code. For this reason, ASCII strings will be used for this driver. As the UTM co-ordinate system has been chosen, it is easiest if data is returned in this form. All Novatel GPS units use a set of logs to describe the types of data available. Hence the BESTUTM log will be used, as this returns the GPS unit’s best estimate of position and variance, in three dimensions. The log for velocity and heading data is BESTVEL. These logs work independently of
the operation of the differential link but contain information about whether or not the differential correction has been successfully applied. These logs can be set up at a fixed data rate which is suited to the event driven software model utilised. These logs are a straightforward way to return pose information from the GPS and so are implemented in software on the on board computer.

The strings that are returned must then be split into the constituent fields and transferred to the localisation processes through the mpi, as seen in Figure 5.9. The logs contain both header and data fields, which are separated by a semi-colon while the individual fields are separated by a comma. The header field is used to check what data the string contains and also what method is currently being used to find position of the GPS. The data fields contain the actual position and variance information. Some fields must be converted to a different variable type, to be interpreted correctly. For example, all position and variance information is converted to type float or type double, which can be achieved using standard C++ libraries. When the data has been successfully received, it is written to a location in shared memory, such that it can be accessed by the localisation processes.
The code for the GPS driver, which was developed significantly both at the University of Adelaide and Strategic Engineering, is included in Appendix H.

![Novatel GPS Driver Flowchart](image)

**Figure 5.9:** Basics of operations of the GPS driver.

The differential is a set of corrections or biases that are calculated by the base station GPS and then sent to the mobile GPS units. A base station is able to generate a more accurate location because it is stationary for a long period of time and can be set up with a position that it knows to a high degree of accuracy. It then calculates the differences between its known location and the location calculated through the satellite signals it receives. It uses this information to generate biases for the satellite signals and for positions calculated. These biases are used by a mobile GPS to improve its accuracy and so must be transferred from the base station GPS to the mobile GPS units. These corrections should allow the GPS to calculate position to an accuracy of approximately 2 cm (Novatel 2009). The code is principally the work of Strategic Engineering but is based on the open source *serialoverip* software.

5.3.5.1 Testing

As introduced in Section 3.1.1, the accuracy of the position measurement from any UGV is dependent on the accuracy of the base station position and this has been confirmed in testing through the project. As the base station position can only reasonably be determined by collecting a
Comparison of position measurements using GPS and differential GPS

Figure 5.10: Comparison of position measurements using GPS and differential GPS. Red line shows nominal path, green dots are differential GPS measurements and blue dots are standard GPS measurements.

large number of GPS readings while the GPS is stationary, high accuracy can never be achieved in the situations of this project, where the ground control station is a mobile system. For the purposes of testing in this project, the base station GPS position was typically determined using averaging over the course of an hour. Novatel (2010) claim that this is equivalent to an accuracy of approximately 60cm. Appendix I includes more details on base station set up.

To test the accuracy of the GPS system, a test was performed in an outdoor area with a clear view of the sky, where the UGV was driven around a rectangular course or approximately known size. As the UGV was driven by a human operator and measurements of the course could not be achieved to a high degree of accuracy, the course was not entirely consistent every lap. The UGV was driven around the rectangle for four laps total, two measuring standard GPS and two using differential GPS. As can be seen in Figure 5.10, the reliability of the measurements from the differential system improved greatly on those from the standard GPS test, as did the consistency between laps. Typical error of the standard GPS position was approximately 2m, compared to an error of less than 1m for the differential system. Precision of the differential system is typically within 20cm, compared to 1m for the standard GPS. The accuracy compared to the measured path was not excellent but the precision of the differential GPS system was good. Precision is important in the MAGIC project, as localisation does not need to be accurate on a world scale, only accurate relative to other previously travelled areas.
5.3.6 Camera Driver

The camera driver is used to initialise the camera on each UGV such that it streams images to a storage location on the onboard computer. A Power over Ethernet (PoE) cable between the camera and the 4-port hub connects the camera to the network. The most important camera settings determined by the camera driver are the image format, the image size and the image clarity.

For much of the year, testing of the camera driver was done on computers with Windows Operating Systems (OSs). While testing the camera driver on the Windows OS, the Baumer Filter Driver was used, which lowers the CPU load caused by the Baumer-GAPI (Generic Application Programming Interface) to 20% of the load normally caused by using a normal Windows socket driver. This closely simulated the performance of the camera driver on a Linux system, because a Linux socket driver uses approximately 30% of the CPU load normally caused by using a Windows socket driver. Early on in the year, this was seen as a good indicator of the success of the driver on a Linux OS, but the problems with network bandwidth later made the in year made the performance redundant.

5.3.6.1 Image Format

An important camera driver requirement was to provide the images in a suitable digital format for analysis. The three main types of image format that could be delivered by the Baumer TXG20c-P camera were RGB (Red Blue Green), Monochrome and YUV with various sampling rates (a colour space consisting of one luma (Y) and two chrominance (UV) components). RGB was chosen as the format to be used, the reason being that monochrome was not useful for detecting the distinctive...
colour of the objects and the computer vision supervisor had substantial experience with the RGB colour space but very little with the YUV colour space. As discussed in Chapter 10, YUV offers little advantage over RGB. A number of formats for transferring the camera data were considered. Even before the wireless network bandwidth became a severe limiter on the size of the data stream, compression was the main factor in the choice of camera data format in order to reduce the load as much as possible on the data transmission services. The three main types of compressed visual data are JPEGs, MJPEGs and MPEGs, all discussed in Appendix G but JPEGs were used in this case due to the resulting small file sizes.

5.3.6.2 IMAGE SIZE

During testing of the wireless network, it was discovered that the bandwidth on the most reliable transmission standard for our hardware (802.11b) was approximately 5 Mb/s (or 0.63 MB/s), which was insufficient to carry the theoretical bandwidth of a single GigE camera, which Baumer specifies as taking up to 800 Mb/s (or 100 MB/s) for 16 frames per second at the maximum resolution of the cameras. The 802.11n Wi-Fi transmission standard was also tested, but the data transfer rate at 70 Mb/s (8.8 MB/s) was still insufficient, and proved to be less reliable than 802.11b. To significantly reduce the load on the wireless network from the cameras, the camera driver converts the format of the images from the raw RGB format (approximately 5.8 MB) to the JPEG format discussed above (approximately 600kB), which is a 90% decrease in file size. This reduction can be still insufficient to ensure network reliability at long range (e.g. more than 20 metres), so one technique used by the camera driver to reduce file size further is to set a partial scan area in the centre of the camera view to be extracted rather than the whole camera view. For example, the normal image size is 1624 x 1232, and the centre area extracted is 816 x 400, producing a file size of approximately 100 kB. The extracted area is easily adjustable in the camera driver code, so that if only one UGV is being used rather than multiple UGVs, the partial area scan may be made larger since the UGV does not have to share bandwidth with other UGVs. The other technique used is a bilinear image interpolation algorithm in the OpenCV library, where the whole camera view may be seen, but with reduced detail. Reducing the image size to a quarter of the original gives image dimension of 812 x 616, producing a file size of approximately 160 kB.

5.3.6.3 IMAGE REFRESH RATE

The update speed of the camera images to the GCS display depends mainly on the speed of the image file transfer across the network, and to a lesser extent, the frame-rate of the camera. The advantage of having a smaller image file size is that the rate of update of images received by the GCS is increased (because more files of smaller size can be transmitted in any given time), making it easier to track the mobile OOI as they move between images. To also ensure the images are the latest possible, the frame rate of the camera recording was optimised. The camera had two image types both having the same size but differing refresh rates and quality determine by exposure time. The format mode “full frame” with a maximum frame-rate of 16 frames per second was selected over the mode “full frame HQ” which halved the maximum frame-rate for an increased exposure time.

5.3.6.4 IMAGE CLARITY

The camera driver was required to provide sufficient clarity for the object finder colour detection as well as visual appearance for the GCS user. The clarity is mainly affected by the focus, exposure and gain of the camera. The camera focus is adjusted manually using a thumb screw on the Kowa lens.
The camera obtains a picture with sufficient contrast and light intensity to look visually correct and aesthetically pleasing to the GCS user by manipulation of the camera gain and exposure, as shown in Fig 5.12. To achieve this, the desired average brightness across the whole image was set to a constant value. Samples of pixel brightness from the latest saved image are taken from points in a 160 by 160 grid across the image. If the sample average is too bright or dark compared to the desired brightness, the gain is decreased or increased in order to match it. The gain control works by amplifying or scaling down the pixel values read from the CCD (Charged-Coupled Devices, or camera pixel sensor). The Baumer TXG20c-P is capable of a 0 to 20 dB gain range for scaling the pixel values to adjust the image light intensity. If the adjustment of the gain is not large enough to reach the desired average brightness, then the exposure of the camera is manipulated accordingly, increasing the exposure time for images that are too dark and vice versa. The gain is adjusted first, because it is desired that the exposure time is kept to a minimum in order to reduce blurry images.

5.4 GCS SOFTWARE

The Ground Control Station (GCS) is essentially the brain of the MAGIC 2010 system and fulfills the two roles of compiling data collected by the UGVs and planning the future actions of the UGVs. This includes sending high level instructions to the UGVs on the field and fusing mapping data collected by the UGVs. It has the requirement to consistently perform these functions so that the UGVs are used to their maximum potential. With the assistance of a Human Machine Interface (HMI), it is possible for commands to be sent from the GCS in events where the operator desires to alter or interrupt the actions of any robot. Most of the software used on the GCS is explained in later sections except for the development of the Human Machine Interface because it relates to all of the other subsystems of the vehicle. Figure 5.13 demonstrates how the GCS relates to the UGVs via software.

5.4.1 HUMAN MACHINE INTERFACE

The Human Machine Interface is required to display information gathered by the UGV and provide a method for the operator to remotely control it. It is the interface which is built upon other components of the system. It is required to display maps that have been generated, show the status and position of any vehicle and also display images transferred from the UGV’s cameras. An interface has been built on top of the mapping code, that provides the method to display the maps.
Figure 5.13: Level Two Software flowchart
generated. It was mainly developed by the Australian Centre for Visual Technologies (ACVT) at the University of Adelaide.

The HMI was developed using a development environment targeted at developing graphical interfaces, called QT. QT was chosen by Strategic Engineering early in the development of the GCS. It is a cross platform application primarily developed by Nokia but freely available. It is a simple and graphical means of creating a display and interface and can be used with C++. It allows a high degree of customisation yet still has a small learning curve and hence it is suitable for this project.

The HMI was developed prior to the finalisation of the mpi library and with the aim of receiving and sending all data via sockets. It is used in conjunction with a pre-processing application on board the GCS, which receives data from the UGVs and converts it into a form to send to the HMI. The use of a pre-processor makes the final HMI very flexible, as any changes to the structure or size of the UGV system only require changes to the pre-processor, while the HMI is developed to handle as many UGVs and as much data as is likely to ever occur. The downside of this system is that the pre-processor is required to convert the sensor data from the form dictated by the mpi to the form required for the HMI. It is also possible to have multiple pre-processing applications, for example there is one for tracking objects of interest and one for mapping and localisation.

As most of the development this year has focused on lower level platform development, the HMI is still very simple. It can display and generate maps and show the location of any UGVs and OOIs on these maps also. It is capable of displaying detailed position and mode of operation information about each UGV. An example output from the HMI is shown in Figure 5.14. Future development of the HMI could include integration with the touch screen, development of a method to display paths planned through the control algorithms (Section 9) and the ability to query the operator in case of decision that cannot be reliably made autonomous.
Figure 5.14: Example display from the HMI, showing simulated feature map.
LOCALISATION

Localisation is the ability of the UGV to determine its position and orientation in space. This is a feature required in order for the UGV to perform other tasks like position control, track objects of interest and perform accurate mapping. As introduced in Chapter 3.1.1, there are four pieces of hardware that can be used to perform localisation. There is the Global Positioning System (GPS) unit, the Inertial Measurement Unit (IMU), the wheel encoders mounted to the drive motors and the Light Ranging and Detection (LiDAR) unit. As the GPS unit is only useable in an outdoor environment (See Section 2.3.1 for details), the localisation problem is split into outdoor and indoor localisation, where outdoor localisation relies on the GPS, but indoor localisation relies on more complex and computationally intensive processes and is discussed in Chapter 7. While the GPS does accurately track the UGVs location, it is somewhat slow to update and prone to dropouts in areas close to buildings and so is used in conjunction with the IMU, which is much faster to update but not as accurate as the GPS unit. Wheel encoders can also be used to aid localisation, but have not been used to this point due to difficulties in implementation. A Kalman filter (Section 2.3.2) is used for the integration of these sensors as it is capable of mitigating the effect of sensor noise, combining measurements of different and varying accuracy and predicting the UGVs position between measurements. This produces effective results; however, these results are limited by the accuracy of the GPS unit and length of GPS dropouts experienced. At these occasions the Kalman filter relies heavily on the IMU, which is prone to drift.

6.1 FUNCTIONAL REQUIREMENTS

In order for an autonomous vehicle to be useful and functional, accurate and reliable localisation is required. There are several sensors used on board the UGV to provide measurements of the vehicle’s pose (position and orientation), however these need to be combined into an accurate estimate. Also, these measurements are typically noisy, so they must be filtered to be useable for a UGV application.

The Microstrain IMU and Novatel GPS chosen both include built in filtering, so the measurements from these are relatively stable, but still prone to noise from the system vibration. The Kalman filter must be able to combine these measurements into one estimate of pose, . There are some elements of pose that are only measured by one sensor and so these do not need to be part of the Kalman filter. Some of these elements are the height from the GPS, and the roll and pitch measurements from the IMU. The elements that are most important for the localisation are the position (Easting and Northing), velocity and heading or direction that the UGV is facing. The Easting and Northing are defined as the distances East and North respectively from the base station to the UGV. Hence, the output from the Kalman filter will be these four states (See Figure 6.1). It is possible to add extra states and this could be considered for improved accuracy at a later stage.

The IMU and GPS units operate at a fixed frequency, while the motor encoders and SLAM operate continuously. The SLAM and Kalman filter algorithms will operate at a frequency that balances the computational load and the accuracy of the calculations. The Kalman filter will operate in order to provide sufficient accuracy for mapping and tracking objects.

The design specification for localisation is an accuracy of 200mm (Section 1.3.3) and if the vehicle is moving at its maximum speed of 10km/h, then the dynamics of the vehicle update at up to 1.4Hz
SECTION: 7
Location (East, North)
Velocity
Yaw, Acceleration (x)
Location (East, North), Velocity, Heading, Pitch, Roll

SECTION: 6.5
Extended Kalman Filter

SECTION: 6.4
Dynamic Model

POSE (East, North, Velocity, Heading, Pitch, Roll, Vertical & Horizontal Angles of PTU)

Dashed Line: direct data feed into POSE

Future Work

Figure 6.1: Level 4 POSE Flowchart
6.2 System Model

As seen in Figure 6.2, the UGVs pose may be expressed as \( (E, N, \theta)^T \). These quantities are measured from the centre of the UGV. The derivatives of these quantities are required to maintain a more complete knowledge of the motion of the vehicle, however due to the non-holonomic constraint of the vehicle these derivatives are not independent. Based on the formulation of Kelly (1994, p. 29), only velocity and angular velocity states are added. The Microstrain IMU already uses the angular velocity information to track changes in heading, so this state does not need to be included in the system model. An acceleration state is included, while this could be considered an input to the system model, the formulation is simplified by tracking its value as a state. To avoid problems with a discontinuity of heading information at 180 and -180 degrees, heading is represented as a vector of the form \( (\sin \theta, \cos \theta)^T \). Hence the states of interest for the Kalman filter are:

\[
\begin{bmatrix}
E \\
N \\
\sin(\theta) \\
\cos(\theta) \\
v \\
a
\end{bmatrix}
= 
\begin{bmatrix}
\text{Easting}(m) \\
\text{Northing}(m) \\
\sin(\text{heading}) \\
\cos(\text{heading}) \\
\text{velocity}(m/s) \\
\text{acceleration}(m/s^2)
\end{bmatrix}
\]

(6.1)

The system is governed by the differential equations as follows, a simplification of Kelly (1994, p. 30).

\[
\begin{bmatrix}
\dot{E} \\
\dot{N} \\
\dot{\theta} \\
\dot{v}
\end{bmatrix}
= 
\begin{bmatrix}
v \sin \theta \\
v \cos \theta \\
\omega \\
a
\end{bmatrix}
\]

(6.2)

As angular velocity is not being included in the system model, \( \omega \) is set to zero. Hence the derivatives of both \( \sin \theta \) and \( \cos \theta \) will also be set to zero.

6.3 Types of Kalman Filters

Various types of Kalman filters have been compared extensively in published research and so a detailed comparison between them is not required. The fundamental aspects of each type of filter...
is explained in Section 2.3.2.1. The basic Kalman filter, relies on linear system dynamics (Welch and Bishop 2006, p. 7) and so is applicable in very few cases and is not useful for the development of a UGV. For cases where a non-linear system model is required, the system model is linearised to a form suited to the Kalman filter and this is named the Extended Kalman Filter (EKF). The Unscented Kalman Filter also has a similar functional result to the EKF. The TARGET project (Cullen et al, 2007, p. 33) compared the types of Kalman filter suited to an IMU/GPS data integration and is included in Appendix G. The EKF has become the de facto standard for this application (Orderud, 2005, p. 1) and for this reason is used for the UGV.

6.4 STATE SPACE MODEL

The Kalman filter is based around a state space model of the system that is one representation of the dynamic model. The state is chosen such that it is the elements of the pose that need to be combined or filtered. The choice of state is not unique, there are many possible choices and this one does have the deficiency that the states are not all independent. The two heading states are related. The state is defined by the vector \( x \) and is given by:

\[
x = \begin{bmatrix}
E \\
N \\
\sin \theta \\
\cos \theta \\
v \\
\alpha
\end{bmatrix}
\]  

(6.3)

The inputs to the system that are controlled by the motion of the UGV are the accelerations of the wheels on each side of the vehicle. These are generated by changing the current to the left and right drive motors. It is possible to measure this through the EPOS motor controller but these
measurements would be prone to errors caused by wheel slip, which is a regular occurrence because of the differential drive model utilised (Baumgartner et al. n.d., p. 5). The inputs of the dynamic model could also be measured by the IMU, these are forward acceleration and angular acceleration, which are the direct result of the current used by the drive motors. For the sake of simplicity these are not treated as separate inputs to the system model, and instead, acceleration is another state that is filtered. Filtering of the acceleration state gives a better estimate of actual acceleration than the raw sensor reading, due to filtering of external disturbances. Therefore, this information could be used for path planning and other control algorithms. This type of Kalman filter, where there are no deterministic inputs included in the system model, is referred to as an Unforced Kalman Filter.

The measurements are the values returned from the sensors. A Kalman filter can use measurements of any quantity that can be used to estimate the value of the states. There will be many other measurements available from the sensors, but only those that are useful to update the states are included. The measurements are defined by the \( y \) vectors, with one for each sensor.

\[
y_{\text{GPS}} = \begin{bmatrix} E_{\text{GPS}} \\ N_{\text{GPS}} \\ v_{\text{GPS}} \end{bmatrix}
\]

\[
y_{\text{IMU}} = \begin{bmatrix} \sin(\theta_{\text{IMU}}) \\ \cos(\theta_{\text{IMU}}) \\ a_{\text{IMU}} \end{bmatrix}
\]

\[
y_{\text{encoders}} = \begin{bmatrix} v_{\text{encoders}} \end{bmatrix}
\]

Using the results from Section 6.2, the differential equations are expressed in terms of the state chosen. The model is changed into State Space form.

\[
\dot{x} = \begin{bmatrix} v \sin \theta \\ v \cos \theta \\ 0 \\ 0 \\ a \\ 0 \end{bmatrix}
\]

The Jacobian, \( F \), for the system model relationship are then found. This will use the following useful result:

\[
\frac{d \sin \theta}{d \cos \theta} = \frac{d \sin \theta}{d \theta} \times \frac{d \theta}{d \cos \theta} = \cos \theta \times \frac{d \cos \theta}{d \theta} = \cos \theta \div (-\sin \theta) = -\cot \theta
\]

\[
F = \frac{\partial \dot{x}}{\partial x}
\]
The measurement model relates the measured quantities to the states of the Kalman filter. This is used to update the states from the measurements. It is represented by $\hat{y}$, meaning is it the estimated value of $y$.

$$\hat{y}_{GPS} = \begin{bmatrix} E \\ N \\ v \end{bmatrix}$$  \hfill (6.10)

$$\hat{y}_{IMU} = \begin{bmatrix} \sin \theta \\ \cos \theta \\ a \end{bmatrix}$$  \hfill (6.11)

$$\hat{y}_{encoders} = \begin{bmatrix} v \end{bmatrix}$$  \hfill (6.12)

The Jacobians for the measurement model relationship are also found:

$$C = \frac{\partial \hat{y}}{\partial x}$$  \hfill (6.13)

$$C_{GPS} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$  \hfill (6.14)

$$C_{IMU} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$  \hfill (6.15)

$$C_{encoders} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$  \hfill (6.16)
In the continuous time domain, the system can now be represented as:

\[ \dot{x} = Fx \]  \hspace{1cm} (6.17)  

\[ \hat{y}_{\text{GPS}} = C_{\text{GPS}}x \]  \hspace{1cm} (6.18)  

\[ \hat{y}_{\text{IMU}} = C_{\text{IMU}}x \]  \hspace{1cm} (6.19)  

\[ \hat{y}_{\text{encoders}} = C_{\text{encoders}}x \]  \hspace{1cm} (6.20)

As a system that works in the discrete time domain is required, this can be converted to a discrete form. The \( F \) matrix is non-constant and dependent on the states, but the measurement models are all constant. Hence, the system, where \( T_k \) is the time difference between the current and previous iteration, becomes:

\[ \frac{x_k - x_{k-1}}{T_k} = F_k x_{k-1} \]  \hspace{1cm} (6.21)  

\[ \hat{y}_{k,\text{GPS}} = C_{\text{GPS}}x_k \]  \hspace{1cm} (6.22)  

\[ \hat{y}_{k,\text{IMU}} = C_{\text{IMU}}x_k \]  \hspace{1cm} (6.23)  

\[ \hat{y}_{k,\text{encoders}} = C_{\text{encoders}}x_k \]  \hspace{1cm} (6.24)  

Equation 6.21 is simplified, using \( A_k = I + F_k T_k \), to

\[ x_k = A_k x_{k-1} \]  \hspace{1cm} (6.25)
A basic check of the dynamic model at this point shows that it is not useable as the resulting dynamic model is not equivalent to the initial dynamic model or the behaviour of the system.

\[
x_k = A_k x_{k-1} = \begin{bmatrix} 1 & 0 & v_T & -v_T \cot \theta & T \sin \theta & 0 \\ 0 & 1 & -v_T \tan \theta & v_T & T \cos \theta & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} E \\ N \\ \sin \theta \\ \cos \theta \\ v \\ a \end{bmatrix}
\]

This may occur because not all the states are independent. As this model is clearly incorrect, the dynamic model was simplified to:

\[
F_k = \begin{bmatrix} 0 & 0 & v_k & 0 & 0 & 0 \\ 0 & 0 & 0 & v_k & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\]

### 6.5 Extended Kalman Filter

After setting up the state space model, the algorithm for the Kalman filter is straightforward. According to Welch and Bishop (2006, p. 10) the extended Kalman filter is as follows. This is used to predict the new value of the state based on the previous state and inputs. This means that even the states that are not measured can be updated. This is the time update step and is achieved through the following two equations:

\[
\hat{x}_k = A_k \hat{x}_{k-1} \\
P_k = A_k P_{k-1} A_k^T + Q_k
\]
The hat represents an estimate of the true value. We cannot determine the true value and so the hat is used to reinforce this. The minus represents a predicted value, from the dynamic model. Note, that the $A_k$, $P_k$ and $Q_k$ matrices are all time dependent. $P_k$ is the matrix of estimated error covariance of the elements of $\hat{x}$, and $Q_k$ is the process noise covariance matrix. $Q_k$ is a diagonal matrix, containing the estimated variance of the difference between the true value and the value from the dynamic model.

The second step is the measurement update, where the predicted state is combined with the new measurements to produce a minimum variance estimate of the state. $P_k$ gives an estimate of the accuracy of the estimated state, $\hat{x}_k$.

$$K_k = P_k C_k^T (C_k P_k C_k^T + R_k)^{-1}$$  \hfill (6.30)

$$\hat{x}_k = \hat{x}_k + K_k (y_k - C\hat{x}_k)$$  \hfill (6.31)

$$P_{k} = (I - K_k C) P_k$$  \hfill (6.32)

$R_k$ is the estimated noise associated with the measurements. It is a diagonal matrix with the same number of entries as measurements used for this update step. $K_k$ is the gain term and is a measure of the relative accuracy of the previous estimate compared to the new measurement. The time update - measurement update operation of the Kalman filter is summarised in Figure 6.3.

![Figure 6.3: Time Update - Measurement Update cycle of a Kalman filter. (Welch and Bishop 2006)](image)

The values of $R$ and $Q$ are vital to the successful operation of the filter. They control the settling time and signal tracking of the filter. This allows for some capability to ‘tune’ the filter, which involves adjusting the usability of the filter by changing the weighting associated with each measurement. Those measurements with a high degree of accuracy are given a higher weighting by the filter and
those with a low degree of accuracy are given a lower weighting. This is achieved by the values set in the R and Q matrices. A very noisy set of measurements would be given a large covariance in the R matrix, meaning that the filter should track the true value of the state and not be affected by noise. Obviously, the values in R need to be carefully set for this to work effectively. A larger value of R will typically reduce the bandwidth of the response but a smaller value of R makes the filter more prone to perturbations of noise.

6.6 IMPLEMENTATION

The final implementation of this filter would draw inputs from the GPS, IMU, wheel encoders and SLAM algorithm. The SLAM algorithm is still in development and so was not available for integration over the course of this project. The driver software for the EPOS motor controllers written by Strategic Engineering did not successfully implement feedback from the motor drivers, hence, measurements from the wheel encoders are also not available. The only measurements available for the filter are from the IMU and the GPS. The drivers for the IMU and GPS write the measurements into shared memory, using the mpi library and the Kalman filter can read these measurements from shared memory when they are updated. As the platform development of the UGV took a large amount of the project’s time scale, initial testing was done using a trolley (Figure 6.4) but many details could not be finalised as the dynamics of the trolley were not representative of the actual behaviour of the UGV.

Initially, the values of the state and the error matrices must be estimated; however, these are not critically important, as the filter will converge to the correct values reasonably quickly. The initial state estimate is generated by reading the last measurements of those states from the IMU or GPS. If there have been no measurements, then these are set to zero. The value of the diagonal elements of P are initialised to be very large to ensure that the initial state is given a very low weighting by the filter.

The sensors (IMU and GPS) are set up to send data at a fixed frequency and when new measurements are received, any measurements that are not being used in the filter, are saved to be written to the pose. The EKF is then iterated, so that the estimate of the state is updated. The updated state estimate and the other measurements are then written to the pose within the
mpi library. This guarantees that it is accessible by several other processes that will use the same information.

The filter does not operate at a fixed frequency, but relies on the input of new data to start the update. The maximum output rate of the GPS is 20Hz (Novatel n.d.) but both the IMU and wheel encoders can operate significantly faster if required. Because of the use of the dynamic model, the states that are not measured are still updated. This is implemented through a multi-threaded process, where the measurements from each sensor have a separate thread and as soon as new data is received by any of the sensors, the filter is be iterated, only using the data from the new measurement. The C++ code to demonstrate the finer details of the filter’s operation is included in Appendix G.

6.7 Tuning

When the filter has been implemented in software, the process noise and estimated noise matrices can be tuned to achieve the best performance of the filter. This can be performed theoretically. The noise in each measurement is measured by running the UGV with that state constant. For example, the UGV can be driven North, to measure the noise in the measurement of the Easting from the GPS. The process noise can be estimated by considering the planned actions of the vehicle. The process noise associated with the acceleration can be determined by knowledge of the planned maximum acceleration of the vehicle. Details for this are included in Kelly (1994, p. 40). In practice, these error sources are only roughly known in magnitude and the combined influence of multiple random and systematic error means that the errors are not well enough understood to be modeled reliably (Kelly 1994, p. 35), and hence, be worthwhile for use within the filter. Thus, a better performing filter can typically be developed by selecting values for the Q and R matrices through trial and error or manual tuning.

6.7.1 Dynamic Model

Before any useful tuning of these parameters can be performed, the dynamic model must be checked. This can be done by comparing the results when the filter is only using one sensor, with the measurements from the other sensor. For example, the position of the UGV is updated by the acceleration and heading measurements from the IMU. Conversely, the heading estimate is updated from the changes in position, through the Kalman gain \( K \). As the GPS is the most accurate sensor, this was tested first, to check that the heading and acceleration estimates were being updated correctly.

It is worth noting, from Equation 6.27, that only one term in both the first and second row has been kept from the old dynamic model. The terms chosen were considered a pair, as they followed a similar format of the coefficients used. Any pair of terms can be used, where there is one from each row. All these pairs were tested. This does not affect the results from the time update step but measurement updates from the GPS will update the states related to the chosen term. Hence, for the dynamic model, the heading estimate is updated through the Kalman gain when a new GPS measurement is received. This is shown in Figure 6.5, where the new dynamic model is being tested and only measurements from the GPS are being used by the Kalman filter. The resulting heading estimate is compared to the heading measurement from the IMU. This clearly shows that the heading is updating correctly from GPS measurements.

The same test was performed for the acceleration estimate, which is updated from GPS velocity measurements and direct measurement from the IMU. A comparison of the two showed that that
Figure 6.5: Comparison of heading estimate, generated through the Kalman filter from GPS measurements and IMU heading measurements. In this example, multiple process noise weightings have been tested and the green line is chosen as the best.

the IMU acceleration measurements are extremely noisy and did not clearly represent the actual acceleration of the vehicle. This is caused by the vibration of the UGV when it is moving over bumpy ground but the noise in the measurement is often 50% of any possible acceleration value, making the two impossible to distinguish. When the acceleration measurement is filtered heavily by the Kalman filter, as shown in Figure 6.6, the update of the final value is extremely slow and still the result is not useable. As a Kalman filter operates according to the error in a measurement compared to the dynamic model and does not explicitly use pole placement, a pre filter was necessary that could be developed using pole placement. A low pass filter was implemented and the design of this was guided by a frequency spectral analysis, as shown in Figure 6.7. This one sided spectral analysis was performed with 50% overlap of each of 1024 Fast Fourier Transform sections.

It can be seen from the spectral analysis (Figure 6.7) that there is some low frequency noise that is within the range of interest of the localisation system. While a heavily weighted Kalman filter is capable of filtering this noise, a low pass filter was used as it offers direct control over the operation of the filter. A low pass filter was designed using MATLAB’s sptool. This tool is used to automatically generate the co-efficients of a filter, to be implemented as a difference equation, and significantly improves development time over traditional hand calculation techniques. To avoid the signal around 3Hz, the poles of the filter needed to be quite small and this was set through the use of the pass band and stop band frequencies within the design tool. The sampling rate was set to 90Hz, as this was the approximate successful sampling rate of the IMU in testing. The pass band frequency was 0.5Hz and magnitude 1dB and the stop band frequency was 2Hz and magnitude 10dB. The resulted is the following difference equation, the result of which is multiplied by a gain of 0.00048.

\[
\]  
(6.33)
Figure 6.6: Comparison of acceleration estimates from a Kalman filter using GPS measurements and filtered IMU acceleration measurements.

Figure 6.7: Frequency spectrum analysis of acceleration measurements from the IMU (in dB).
The resultant acceleration when using the low pass filter can be seen in Figure 6.8. The result is not entirely free of noise but it was decided not to filter the signal further, so it does not affect the parts of the signal that are needed for localisation.

![Figure 6.8: Resulting acceleration signal after low pass filtering.](image)

### 6.7.2 Measurement Noise

The effect of measurement noise is reduced through the correct selection of the \( R \) matrices. These are diagonal matrices, defined as:

\[
R_{GPS} = \begin{bmatrix}
\sigma_{\text{Easting}}^2 & 0 & 0 \\
0 & \sigma_{\text{Northing}}^2 & 0 \\
0 & 0 & \sigma_v^2
\end{bmatrix}
\]  

(6.34)

\[
R_{IMU} = \begin{bmatrix}
\sigma_{\sin(\theta_{\text{IMU}})}^2 & 0 & 0 \\
0 & \sigma_{\cos(\theta_{\text{IMU}})}^2 & 0 \\
0 & 0 & \sigma_a^2
\end{bmatrix}
\]  

(6.35)

The GPS unit provides an estimate of its accuracy, so this can be used for \( \sigma_{\text{Easting}} \) and \( \sigma_{\text{Northing}} \). These can be multiplied by a constant value if required to improve the tuning. The claimed accuracy of the velocity measurements from the GPS is 0.03 m\( \text{s}^{-1} \) but this value has been far too small in testing.
To determine the values to be used, the Kalman filter is run without the dynamic model, where the $F_k$ matrix is zero. Various values of sigma are tested and through a process of trial and error, the value is chosen that gives a good compromise between settling time and noise perturbation. An example of this is shown in Figure 6.9, demonstrating that for a resultant signal not to perturbate noise, it may be slower to update to the true values than some lower weighted filters.

This method is useful to find the amount of noise in a signal but the result is not independent of the process noise covariance matrix. Hence, this process may be repeated many times as part of an iterative tuning, along with the process noise covariance. Best results could be achieved by using this method with states whose true value is not changing, but this is not possible for all states. Ultimately, some final tweaking with a fully operational filter will produce the best result.

### 6.7.3 Process Noise

The process noise matrix estimates the variance in the error between the dynamic model and the true values. It is a diagonal matrix, defined as:

$$
Q = \begin{bmatrix}
\sigma_E^2 & 0 & 0 & 0 & 0 & 0 \\
0 & \sigma_N^2 & 0 & 0 & 0 & 0 \\
0 & 0 & \sigma^2_{\sin \theta} & 0 & 0 & 0 \\
0 & 0 & 0 & \sigma^2_{\cos \theta} & 0 & 0 \\
0 & 0 & 0 & 0 & \sigma_v^2 & 0 \\
0 & 0 & 0 & 0 & 0 & \sigma_g^2
\end{bmatrix}
$$

(6.36)
The process noise can be tuned using the same method that was used in Section 6.7.1 to prove the dynamic model. Once the measurement noise has been tuned to reasonable effectiveness, this allows the process noise to be tuned independently of the measurement noise. Hence the process noise is adjusted through a process of trial and error to choose the best compromise between noise and the speed of the update. This general procedure is shown in Figure 6.5.

### 6.7.4 Final Tuning Values

Using the iterative trial and error procedure shown in Sections 6.7.2 and 6.7.3, the following values were chosen:

\[
R_{\text{GPS}} = \begin{bmatrix}
50 \times S_{\text{GPS,E}} & 0 & 0 \\
0 & 50 \times S_{\text{GPS,N}} & 0 \\
0 & 0 & \frac{100}{v_{\text{GPS}}}
\end{bmatrix}
\]  

(6.37)

\[
R_{\text{IMU}} = \begin{bmatrix}
50 & 0 & 0 \\
0 & 50 & 0 \\
0 & 0 & 100
\end{bmatrix}
\]  

(6.38)

\[
Q = \begin{bmatrix}
0.01 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.01 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.05 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.05 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.2
\end{bmatrix}
\]  

(6.39)

### 6.8 Results

The test method for the Kalman filter is the same as the method used to test the GPS driver in Section 5.3. The UGV is driven around a flat rectangular course of approximately known dimensions, the results from the filter can then be compared to the measurements of the course. It is very difficult to accurately measure the rectangular course with a simple tool like a tape measure, and hence, this should not be treated as very accurate. Independent velocity and heading measurements during the test were also not measurable.

### 6.8.1 Full Differential GPS Reception

The results from the Kalman filter when operated using differential GPS measurements are shown in Figure 6.10. As the accuracy of measurements from the GPS is much greater than through the dynamic model and the overall accuracy of the system is dependent on the accuracy of the GPS, these results are very similar to those shown in Section 5.3.5, which only contain the GPS
measurements. The key difference is that full state tracking has been achieved of all the states of interest of the pose.

![Diagram showing position estimates and actual values](image.png)

Figure 6.10: Best estimate position from Kalman filter using Differential GPS measurements and IMU measurements.

6.8.2 DIFFERENTIAL GPS WITH SIMULATED GPS DROPOUT

One of the key requirements of the operation of the Kalman filter is the ability to cope with short GPS dropouts. It is worth noting that the ability of the localisation system is severely limited during these periods as there are no reliable velocity or position updates available. Figure 6.11 shows the performance of this to date. The estimated heading of the UGV changes at the point of dropout and this is a significant cause of error in the result and is likely to be caused by poor IMU calibration or a local magnetic field influencing the magnetometer used for the heading measurement. Other tests along this test track show an IMU error of approximately ten degrees, counter clockwise, at this point in the course.

6.8.3 FULL STANDARD GPS RECEPTION

The performance of the Kalman filter when the differential GPS is not operating is also of interest. This is reflective of the ability of the UGV to localise when it is out of the range of the wireless data communications. Figure 6.12 shows that the Kalman filter was able to accurately localise for one of the two laps of the test course. The cause of the erroneous estimate is bad GPS measurements, as shown in 6.13. The corresponding accuracy estimate from the GPS measurement is included. The accuracy estimate in this case is not reflective of the erroneous estimate. As the Kalman filter relies on the accuracy estimate from the GPS, it did not adapt to the erroneous measurements adequately because the accuracy measurement from the GPS was not updated. If the filter was tuned to remove
Figure 6.11: Best estimate position from Kalman filter using Differential GPS measurements and IMU measurements, with simulated GPS dropout.

erroneous measurements of this scale, then it would be extremely slow to update to changes in the true value of any state being measured.
Comparison of best estimate position to actual position

Figure 6.12: Best estimate position from Kalman using standard GPS and IMU measurements.

Best estimate easting and GPS measurement

Figure 6.13: Standard GPS measurements, showing erroneous measurements and corresponding accuracy estimates.
6.9 **FUTURE WORK**

It can be seen from the results of this work that the system used for Outdoor Localisation is moderately successful in determining and tracking the location of the vehicle. The complexity of the system, requiring interaction with several pieces of hardware as well as the requirement for a complete and functional platform to be able to complete some details of the system generally made it difficult to accomplish accurate localisation. Despite this, a system has been developed, that in the right conditions, will track the vehicle’s position to within 200 mm accuracy.

One of the key issues with this implementation of outdoor localisation is the poor quality of the acceleration measurements from the IMU. The addition of measurements from wheel encoders is one potential method to improve the ability of the system to track the true velocity and position of the vehicle. The problem with this solution is the difficulty in determining an accurate model for the behaviour of the vehicle in different terrain. For example, the wheels will typically slip less and travel faster on hard surfaces than soft surfaces. Also, tyre pressure changes will affect the usefulness of measurements from the wheel encoders. A dynamic model can be developed, using geometry, to describe the motion of the vehicle in terms of some parameter to represent changes in terrain and driving conditions (Wang et al. 2009, p. 4112). This could then be used with an adaptive controller to estimate the value of the parameter representing terrain conditions (Wellington & Stentz n.d., p. 1) and this should significantly improve the accuracy of the localisation during times of GPS dropout and across a variety of terrain (Wang et al. 2009, p. 4117). Wang et al.’s comparisons on method of localisation show that the accuracy of visual techniques (SLAM) would also return equal or better accuracy in the case of GPS dropouts to the proposed model including wheel slip.
The outdoor localisation system described in Chapter 6 suffers from errors due to drift when a GPS drop out occurs, and thus becomes inaccurate. In an indoor environment, the GPS signal is rarely useable, and hence, an alternative system of localisation must then be used that does not rely on the GPS. The problem itself is one of localisation, where a prior conceived map of the environment is unavailable. Given that an absolute location is required (rather than one dependent on the UGV’s previous location), it is sensible to utilise features in the environment to correct for estimations of the position of the UGV. The problem then becomes Simultaneous Localisation and Mapping (SLAM) and requires calculating the position of the UGV based on features in the environment, and calculating the location of the features based on the position of the UGV.

SLAM, as introduced in Section 2.3.3, is a popular technique used for autonomous localisation of a robot or group of robots. Its function is to localise the position of the robot, as well as creating a virtual map of the surroundings. This map is then used by the robot to plan and traverse an appropriate route based on predefined route strategy parameters. SLAM was coined around 1986 by Durrant-Whyte and Leonard (Risgaard & Blas 2005) in their seminal work based on ideas first proposed by Smith, Self and Cheeseman (1985).

### 7.1 BACKGROUND

The MAGIC 2010 competition requires teams to

"demonstrate autonomous indoor navigation tasks and static OOI neutralisation. For this demonstration, and without moving the GCS used for the outdoor demonstration, teams will be asked to task two sensor UGVs and one disruptor UGV to enter a building (via ramps if necessary). The building entrance should ideally be at least 100m from the GCS and GPS should not be available within the building. Teams will then be expected to explore and navigate a flat and uncluttered course or series of corridors” (MAGIC 2010 December Information Package - Final, page 21- see Appendix A).

Without the aid of the GPS indoors, the UGVs can estimate its position entirely from dead reckoning, where the change in position is updated entirely from IMU and wheel encoder measurements. While this can be accurate at slower speeds and in smooth areas, any errors will accumulate over time as there are no absolute measurements of position. This propagation of errors, or odometry shift, may be disastrous, particularly in the case of the failure of one of the sensors.

It has been decided to tackle the issue of accurate localisation indoors through the use of SLAM. The aim of the integration of the SLAM with other sensors that will be functional indoors, such as the IMU and the wheel encoders, is to reduce the propagation of uncertainty in position, and thus, provide the robot with an accurate localisation technique. Figure 7.1, adapted from the MAGIC competition information package gives an approximation to the shape and topology of what must be traversed indoors. This is an appropriate situation and location to employ SLAM as there are numerous clear landmarks that can be utilised for localisation, including door frames and wall edges.
The localisation system of the UGV plays an essential part in the workings of the software architecture of each UGV. The results of the localisation system feeds position and orientation information to the mapping software, so that the maps can be generated correctly as all features are calculated in reference to the position of the UGV. Moreover, the localisation system is required for accurate control of the position movement of the UGV. The computer vision software also uses position and information to accurately track the location of all objects-of-interest. Without an accurate estimate of position, the maps cannot be generated, control systems will not be able to generate possible paths for the UGVs and tracking of OOI becomes impossible. Hence, localisation underpins all the software being utilised on board the UGVs.

SLAM requires several sensors in order to be functional. First, a sensor is needed that can identify landmarks in the environment and a sensor is needed to provide odometry data. The data from these sensors would need to be combined into an accurate estimate of UGV position and orientation. Moreover, a method is required to filter noisy data from these sensors.

The Microstrain IMU and Leuze LiDAR are the chosen pieces of hardware (See Section 3.1.1). The IMU will provide values of heading and acceleration, while the LiDAR will provide range values of objects that it detects. This can then be used to identify landmarks. These measurements are likely to be noisy, due to external disturbances, even though both pieces of hardware contain internal filtering of data already. A more detailed analysis of system states is provided in Section 7.5.1.

As introduced in Section 6, the localisation subsytems should update at greater than 14 Hz so that the movement of the vehicle can be adequately tracked with sufficient accuracy for mapping and vision purposes. Thus, it is important for both the IMU and LiDAR to update at a rate that is faster than this. The pieces of hardware selected satisfy this functional requirement as shown in Section 3.
The SLAM indoor localisation system was developed to be used indoors as a replacement for the outdoor localisation system that depends on a GPS signal. There are four stages of development of the SLAM, that determine to what level the SLAM can be integrated into the UGV localisation system.

The first stage is the development of SLAM for a virtual environment, that simulates the types of environments likely to be encountered by the UGVs. The simulation developed used a simple map as shown in Figure 7.9. The UGV position was set to move through this map. The LiDAR feed was simulated as the distance between the UGV position and objects in the map with an additional noise value. Odometry data was simulated from the movement of the UGV without noise. The results of this simulation will be further described in Section 7.6.

The next step in the development of the SLAM system was to run the simulation SLAM on data that is recorded from an actual run of the UGV. In this case, coupled LiDAR and IMU data was recorded during a few test runs of the UGVs. This data was used to produce an estimated sequence of pose estimates. The process used is described in Section 7.6.

Once, the system has been tested and produces reasonable results based on data collected from a proper run of the UGV, the next step is for the system to be utilised while the UGV is running. The simplest way to integrate the SLAM with the outdoor localisation system is to have a trigger-based replacement system, where the trigger is a well-defined GPS outage. During a GPS outage or when the GPS signal is too weak to be useful for the outdoor localisation system, it would depend almost entirely on an IMU that inherently will suffer from odometry shift. An outage of approximately 10 seconds would signify a loss of GPS signal and the SLAM localisation system would be initiated and utilised. When the GPS has returned for at least five seconds, it would be worth switching back to the outdoor localisation system. Thus, at this stage, the two systems work in place of each other when triggered.

The final stage of SLAM system development is to combine it with the Kalman filter so that the strengths of both filters can be used at all times. In this scenario, the SLAM is actively extracting and storing landmarks all the time and not just when the GPS is unavailable. By varying the weighting of selected components based on the expected accuracy at each timestep, the system can be kept above a certain accuracy threshold. In particular, if there are few landmarks identified (because the UGV is in an open area, or the environment does not contain certain requisite properties), then we can have the system set the weighting of the SLAM low, so that the system depends more on the values found by the outdoor localisation system. Similarly, if the GPS signal is weak, the system can set the weighting value of the outdoor localisation system to a low value to give more weighting to the SLAM. The easiest way to accomplish this integration is to include the position and orientation information from the SLAM as a direct input into the outdoor localisation Kalman filter and alter the system model appropriately. The weighting issue can be resolved by including the expected noise values of the SLAM into the noise matrices. These values can be found by trial and error, but will likely be in terms of the number of landmarks being identified at each timestep.

Figure 7.2 displays the methods of integration with the outdoor localisation system. Figure 7.2a shows how the SLAM system would be utilised if used as a replacement for the outdoor Kalman filter when triggered by a sufficiently crippling GPS outage. The desired implementation is shown in Figure 7.2b where the SLAM is used as an input to the Kalman filter at all times with a varying accuracy based on the number and quality of landmarks in the environment.
160 | INDOOR LOCALISATION

There are many different SLAM algorithms implemented for robot localisation. See Section 2.3.3 for a list of these algorithms. The majority of these algorithms can be summarised into a four step process:

(i) Landmark Extraction - this is where the system finds landmarks in the environment
(ii) Data Association - this is where the system compares extracted landmarks with landmarks extracted earlier
(iii) Filtering - this is where the system uses the data association information to correct the estimated position of the robot calculated using odometry data
(iv) Navigation - this is where the system uses the position and orientation information to calculate a path for the robot.

The application of SLAM to the indoor localisation problem in this project does not require the SLAM to undertake any navigation, thus, only the other three steps were implemented and a discussion of the methods used will be detailed in this report.

Due to the complexity of all the parts of SLAM and the limited time available for its development, a standard approach to SLAM was adopted. For this reason, the SLAM algorithm implemented was the method called EKF-SLAM. This method uses the four standard stages of SLAM; but in particular utilises an extended Kalman filter (EKF) for the filtering step. EKF-SLAM has the advantage of being well-documented and researched. It is also fairly simple to implement compared to other SLAM techniques, and is more applicable than the simpler basic Kalman filter approach to SLAM, because it attempts to take into account nonlinearities in the system model.

Riisgaard and Blas (2005) give a very generic description of the standard EKF-SLAM approach. An adapted version of their flowchart of the processes involved in EKF-SLAM is shown in Figure 7.3.

The three processes of landmark extraction, data association and filtering can be developed separately, because the level of interaction between these stages of SLAM are limited to what inputs are needed from each of the stages to the other stages. None of the working involved in each stage needs to be used for any of the other stages. In particular, the filtering requires purely an up-to-date database of landmarks extracted and their positions as well as uncertainty values.

Figure 7.2: The different methods of integration of SLAM indoor localisation system with the outdoor localisation system.

7.4 SLAM Process

(a) SLAM used when a GPS outage trigger is activated (b) SLAM used as an input to Kalman filter with a varying accuracy value
associated with each. Data association requires just the database of landmarks extracted, any new landmarks and a special measure of distance from the filter based on uncertainty of each landmark position. Landmark extraction just requires the latest position and orientation estimate. Figure 7.4 demonstrates the interaction between the stages of the SLAM process.

Thus, with this separation of stages, this will be followed by an explanation of how filtering (Section 7.5), data association (Section 7.5.3) and landmark extraction (Section 7.5.4) were implemented, respectively. The code written to implement the strategies described in this chapter can be found in Appendix H.
7.5 FILTER

As explained in Section 7.4, the filter to be implemented is an extended Kalman filter. This EKF will be augmented with information about landmarks, in an application unique to SLAM. The EKF is an appropriate filter, as it is fairly simple to understand and takes into account non-linearities of the system model.

The EKF must perform the three following fundamental tasks (Riisgaard & Blas 2005):

- Update the current state estimate using the odometry data,
- Update the estimated state from re-observing landmarks,
- Add new landmarks to the current state.

The first step is simply the addition of input controls (odometry change) to the previous state estimation to get the new state estimate. In particular, if one of the states is the variable $k$, and one of the control inputs is $\Delta k$, then the result of the first step is $k + \Delta k$.

The second step considers the effect of reobserving landmarks. Using the current estimate of the position of the UGV yields an estimate for the position of a particular landmark, which may be different to where it is expected to be from previous observations. This difference is called innovation and is loosely defined as the difference between estimated and real landmark position, although this is relative to the accuracy of UGV position estimation. The uncertainty of each landmark with regards to the position of the UGV is also updated. This is done through updating the values in the system covariance matrix.

The third step, adds new landmarks to the system state and updates all filter matrices accordingly. In particular, information about the relation between new landmarks and old landmarks and the UGV position are stored.

7.5.1 STATE SPACE MODEL

As illustrated in Figure 6.2, the pose of the UGV is completely defined by $(E, N, \theta)^T$ in global coordinates. In this vector, the $E$ represents the Easting and $N$ the Northing of the UGV (defined in Section 6.2), while $\theta$ represents the heading of the UGV.

It is not necessarily important to have the positional coordinates of the UGV in global coordinates. It is more convenient to consider local coordinates based on the initial position of the UGV and convert to global coordinates when required by other software. Therefore, the system model of the UGV that will be utilised is $(x, y, \theta)^T$, where $x$ is the $x$-coordinate of the UGV in local coordinates and similarly for $y$. Note, that the value of $\theta$ has to be normalised to the fundamental domain of $-180^\circ$ to $180^\circ$.

As feedback from wheel encoders has not been implemented in this project, a UGV velocity value is not measured directly and has to be calculated from the acceleration measurement from the IMU, much like for the outdoor localisation system (see Section 6.2). This acceleration measurement suffers from noise and is better handled when filtered. For this reason, it is useful to include a velocity and acceleration state in the system state.

The system state has to also include the states of the landmarks. This is where EKF-SLAM differs from simple applications of an extended Kalman filter, as EKF-SLAM includes the states of the landmarks in the system state. The reason is because the states of the landmarks are also updated. This occurs, because every observance of a landmark may be inaccurate to some degree, so its
position changes as does the accuracy of that measurement. The states of the landmarks are purely their x-coordinates and y-coordinates. Thus, the state of landmark $i$ can be defined as $(x_i, y_i)^T$.

Combining these facts and the system state, $x$, is defined as:

$$
x = \begin{bmatrix}
x_{\text{UGV}} \\
y_{\text{UGV}} \\
\theta_{\text{UGV}} \\
v_{\text{UGV}} \\
a_{\text{UGV}} \\
x_1 \\
y_1 \\
\vdots \\
x_n \\
y_n
\end{bmatrix}
$$

(7.1)

where the subscript \text{UGV} represents the fact that the property is of the UGV, while a subscript $i$ denotes that it is a property of landmark $i$. All the coordinates are in millimetres, while $\theta$ is in radians.

The system is governed by the following differential equation.

$$
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta} \\
\dot{v}
\end{bmatrix} = \begin{bmatrix}
v \sin \theta \\
v \cos \theta \\
\omega \\
a
\end{bmatrix}
$$

(7.2)

Angular velocity is not included in the system model, thus, $\omega = 0$. The derivative of all other system states is also zero as they are fixed variables.

### 7.5.1.1 Measurement Model

It is worth discussing what measurements are available for this system. There are measurements from the IMU that provide heading and acceleration information and there is information from the landmark extraction process that provides the range and bearing of each landmark from the UGV. This is permissible, as the filter is able to utilise any quantity that can be used to estimate the system states. The measurement models for the IMU and each landmark (Riisgaard & Blas 2005):

$$
\hat{y}_{\text{IMU}} = \begin{bmatrix}
\theta \\
a
\end{bmatrix}
$$

(7.3)
\[
\hat{y}_{\text{landmark},i} = \begin{bmatrix} \text{range}_i \\ \text{bearing}_i \end{bmatrix} = \begin{bmatrix} \sqrt{(x_i - \hat{x}_{\text{UGV}})^2 + (y_i - \hat{y}_{\text{UGV}})^2} + v_r \\ \arctan \left( \frac{y_i - \hat{y}_{\text{UGV}}}{x_i - \hat{x}_{\text{UGV}}} \right) - \theta + v_\theta \end{bmatrix}
\] (7.4)

where \(v_r\) and \(v_\theta\) are measurement noise in the range and bearing, respectively. More on measurement noise is explained in Section 7.5.2.3. The measurement model provides a method of updating the system states based on the measurements. The Jacobians can be found for the measurement model:

\[
C = \frac{\partial \hat{y}}{\partial \mathbf{x}}
\] (7.5)

\[
C_{\text{IMU}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \ldots & 0 & 0 \end{bmatrix}
\] (7.6)

\[
C_{\text{landmark},i} = \begin{bmatrix} \frac{\partial \text{range}_i}{\partial x} & \frac{\partial \text{range}_i}{\partial y} \\ \frac{\partial \text{bearing}_i}{\partial x} & \frac{\partial \text{bearing}_i}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\hat{y}_{\text{UGV}} - y_i}{r^2} & \frac{\hat{y}_{\text{UGV}} - y_i}{r^2} \\ -\frac{\hat{x}_{\text{UGV}} - x_i}{r^2} & -\frac{\hat{x}_{\text{UGV}} - x_i}{r^2} \end{bmatrix}
\] (7.7)

where \(r\) is the range of the landmark from the UGV. For simplicity of understanding,

\[
C_{\text{landmark},i} = \begin{bmatrix} x_{\text{UGV}} & y_{\text{UGV}} & \theta_{\text{UGV}} & v_{\text{UGV}} & a_{\text{UGV}} & x_1 & y_1 & \ldots & x_i & y_i & \ldots & x_n & y_n \\ C_A & C_B & 0 & 0 & 0 & 0 & 0 & \ldots & -C_A & -C_B & \ldots & 0 & 0 \\ C_C & C_D & -1 & 0 & 0 & 0 & 0 & \ldots & -C_C & -C_D & \ldots & 0 & 0 \end{bmatrix}
\] (7.8)

where \(C_A, C_B, C_C,\) and \(C_D\) are as above in Equation 7.7. The top row is not part of the Jacobian; it is merely there, so that it is easier to tell in which column the elements are.

\(C_{\text{landmark},i}\) shows how much the range and bearing of landmark \(i\) changes as the \(x\)-coordinate, \(y\)-coordinate and heading of the UGV change. The partial derivatives with respect to all states except for the value of the coordinates of the \(i^{\text{th}}\) landmark are zero. The first row gives the change in the range of the landmark from the UGV with respect to a change in each of the system states, whereas the second row gives the change in the bearing of the landmark from the UGV with respect to a change in each of the system states. The elements in the \(x_i\) and \(y_i\) columns are unique to SLAM. A simple EKF application would not have any landmark states, and therefore, would not contain these elements.

7.5.1.2 Prediction Model

In this section, the prediction model for the system is described. Additionally, the prediction Jacobian is calculated.
The prediction model is the basis for estimating the new system state based on the old system state and a control input. It is done using the following formula:

\[ \dot{x}_k = \dot{x}_{k-1} + u_k = A_k \dot{x}_{k-1} \]  

(7.9)

where \( \dot{x}_k \) and \( u_k \) represent the system estimation and control input at the \( k^{th} \) timestep. Defining \( A_k = I + F_k T_k \), yields

\[ \frac{\dot{x}_k - \dot{x}_{k-1}}{T_k} = F_k \dot{x}_{k-1} \]  

(7.10)

where \( T_k \) is the time difference between the current and previous iteration. The left hand side can be rewritten as the time-derivative of the system state, thus yielding:

\[ \dot{x} = F_k x \]  

(7.11)

which can be linearised to:

\[ F_k = \frac{\partial \dot{x}}{\partial x} \]  

(7.12)

From Section 7.5.1,

\[
\begin{bmatrix}
\nu_{UGV} \sin \theta \\
\nu_{UGV} \cos \theta \\
0 \\
a_{UGV} \\
0 \\
0 \\
0 \\
\vdots \\
0 \\
0
\end{bmatrix}
\]  

(7.13)
Hence,

\[
F_k = \begin{bmatrix}
0 & 0 & \nu_k,\text{UGV} \cos \theta_k & \sin \theta_k & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 0 & -\nu_k,\text{UGV} \sin \theta_k & \cos \theta_k & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0
\end{bmatrix}
\]  

(7.14)

and thus the prediction model Jacobian, \(A_k\) equals

\[
A_k = \begin{bmatrix}
1 & 0 & \nu_k,\text{UGV} T_k \cos \theta & T_k \sin \theta & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 1 & -\nu_k,\text{UGV} T_k \sin \theta & T_k \cos \theta & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & \ldots & 0 & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 1
\end{bmatrix}
\]  

(7.15)

where \(x_k,\text{UGV}\) and \(y_k,\text{UGV}\) are the UGV position coordinates, \(\nu_k,\text{UGV}\) is the UGV velocity, \(\theta_k\) is the UGV heading and \(\Delta \theta_k\) is the change in UGV heading, all at the \(k^{\text{th}}\) timestep.

Two things can be seen from this Jacobian. The first is that it is time-dependent, which must be the case, as the update of the state is never going to be uniform. The amount that the filter should change the previous estimate, depends on the previous estimate and the varying control input. Secondly, beyond the the top-left \(5 \times 5\) sub-matrix in \(A_k\), the rest is just the identity matrix.
Furthermore, it is less computationally intensive and easier to implement if the covariance matrix, $P$, is updated in stages (Durrant-Whyte & Bailey 2006b). So reducing $A_k$ to a $5 \times 5$ matrix allows for separate updates of the UGV and landmark states. Let this new matrix be $A$. Thus,

$$
A = \begin{bmatrix}
1 & 0 & v_{k,UGV} T_k \cos \theta & T_k \sin \theta & 0 \\
0 & 1 & -v_{k,UGV} T_k \sin \theta & T_k \cos \theta & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & T_k \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix} \quad (7.16)
$$

In this linearised solution, process noise is ignored.

When updating the covariance sub-matrices for the landmarks, SLAM specific Jacobians are needed. These Jacobians are denoted $A_{xr}$ and $A_z$. $A_{xr}$ is used for landmark prediction, with respect to the robot state, whereas $A_z$ is used for landmark prediction with respect to $[\text{range}, \text{bearing}]^T$.

$A_{xr}$ contains just the first two rows of $A_k$. Thus,

$$
A_{xr} = \begin{bmatrix}
1 & 0 & v T_k \cos \theta & T_k \sin \theta & 0 \\
0 & 1 & -v T_k \sin \theta & T_k \cos \theta & 0 \\
\end{bmatrix} \quad (7.17)
$$

and $A_z$ is

$$
A_z = \begin{bmatrix}
\sin(\theta_k) & v T_k \cos(\theta_k) \\
\cos(\theta_k) & -v T_k \sin(\theta_k) \\
\end{bmatrix} \quad (7.18)
$$

### 7.5.2 Extended Kalman Filter

In the previous section, we have established the structure for the state space model. The next step required is the actual implementation of the extended Kalman filter process.

#### 7.5.2.1 Covariance Matrix

The structure of the covariance matrix, $P$, is shown in Figure 7.5. Cell A is a $5 \times 5$ covariance matrix for the UGV position. It contains a covariance value for all of the states of the UGV. The cells going down the diagonal are the covariance matrices of each landmark. B is the covariance of the first landmark and C is the covariance of the last landmark. They are all $2 \times 2$ matrices as each landmark is defined by two states. The cell D is the $2 \times 5$ covariance matrix between the UGV and the first landmark, while cell E is the $5 \times 2$ covariance matrix between the first landmark and the UGV. Thus, D and E are transposes of each other. Thus, cells along the top row and down the first column are covariance matrices between the appropriate landmark and the UGV and vice-versa respectively. All the other cells are covariance matrices between landmarks, for example cell F is the covariance matrix between the last and first landmarks.
At the initial timestep, the UGV has not seen any landmarks, so $P$ is only the matrix $A$ of Figure 7.5. The covariance matrix is usually initialised with values along the diagonal with default values that represent the uncertainty of the initial values of the states. At times, not including these uncertainties can produce singular errors in some calculations (Montemerlo & Thrun 2007). For the purpose of simplicity, the SLAM has been implemented with a null initial covariance.

### 7.5.2.2 Kalman Gain

The Kalman gain, $K$, is used as a measure of accuracy used to determine if we are to utilise the new information from new landmarks. In the simplest sense if a new landmark determines that the position of the UGV is $n$ mm in a particular direction, then this gain, $K$, will multiply $n$ to determine how much the position is altered.

In particular, the gain will depend on the uncertainty arising from the LiDAR as well as the uncertainty in the IMU. The uncertainty in the IMU is unlikely to vary, whereas the uncertainty of the LiDAR depends on the distance measurement received. The lower the uncertainties from the LiDAR, the higher the gain, $K$. The gain is shown below:

$$K = \begin{bmatrix} x_r & x_b \\ y_r & y_b \\ \theta_r & \theta_b \\ v_r & v_b \\ a_r & a_b \\ x_{1,r} & x_{1,b} \\ y_{1,r} & y_{1,b} \\ \vdots & \vdots \\ x_{n,r} & x_{n,b} \\ y_{n,r} & y_{n,b} \end{bmatrix}$$

(7.19)
The first column determines how much gain should be applied from the innovation in terms of range and the second column determines how much gain should be applied from the innovation in terms of bearing. Each row will act solely on the respective row of the state matrix.

7.5.2.3 MEASUREMENT NOISE

In the EKF, there is a term that compensates for measurement noise in the measurement of the LiDAR. This term is:

$$VRV^T$$

where $R$ and $V$ are both $2 \times 2$ matrices. $R$ represents the accuracy of the measurement devices. In this application it is of the form

$$R = \begin{bmatrix} rc & 0 \\ 0 & bd \end{bmatrix}$$

(7.20)

where $r$ represents range and $b$ represents bearing respectively. It is good rule of thumb to make $c = 0.01$ to account for an error of 1% in the range value and make $bd = 1$ for a fixed 1° error in the bearing reading (Riisgaard & Blas 2005). As there is no control input, $V$ is just an identity matrix. This means that measurement noise simplifies down to $R$.

7.5.2.4 PROCESS NOISE

The process noise, $Q$, tracks the accuracy of the odometry. It is usually of the form

$$Q = WCW^T$$

(7.21)

where $W = \begin{bmatrix} \Delta x & \Delta y & \Delta \theta & \Delta v & \Delta a \end{bmatrix}^T$ is the change in the state of the UGV. $C$ is a Gaussian sample that represents the accuracy of the IMU. Due to limited time, different values were never tested and the error values from the outdoor localisation were adopted. This did seem to work and so it was never changed.

Note, this means that

$$Q = \begin{bmatrix} 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & \frac{7}{60} & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0.2 \end{bmatrix}$$

(7.22)
7.5.2.5 **EKF UPDATE**

At the beginning of Section 7.5, the three step process for the EKF applied to SLAM was given. This section shall now explain how each update occurs.

### 7.5.2.5.1 Step 1: Update Current State Using the Odometry Data

This step is the prediction step as it aims to update the state matrix based on the odometry data. The odometry data gives an estimate of the change in position of the UGV rather than providing an actual measurement position. The odometry update is utilised to update the state. This is done through the use of the Jacobian. This yields:

\[
\hat{x}_k = A_k \hat{x}_{k-1}
\]  

(7.23)

Before, the covariance is also updated, the process noise, \( Q \), must be updated to take into account the change in input control. \( Q \) becomes (Riigaard & Blas 2005):

\[
Q = \begin{bmatrix}
    c\Delta x^2 & c\Delta x\Delta y & c\Delta x\Delta \theta & c\Delta x\Delta v & c\Delta x\Delta a \\
    c\Delta y\Delta x & c\Delta y^2 & c\Delta y\Delta \theta & c\Delta y\Delta v & c\Delta y\Delta a \\
    c\Delta \theta\Delta x & c\Delta \theta\Delta y & c\Delta \theta^2 & c\Delta \theta\Delta v & c\Delta \theta\Delta a \\
    c\Delta v\Delta x & c\Delta v\Delta y & c\Delta v\Delta \theta & c\Delta v^2 & c\Delta v\Delta a \\
    c\Delta a\Delta x & c\Delta a\Delta y & c\Delta a\Delta \theta & c\Delta a\Delta v & c\Delta a^2 \\
\end{bmatrix}
\]  

(7.24)

According to Welch & Bishop (2006), it is usually just as effective to have \( Q \) fixed. This is what has been implemented.

Utilising the notation used by Riigaard & Blas (2005), the top 5 × 5 sub-matrix of the covariance matrix \( P \), shall be denoted by \( P_{rr} \). The cross-covariance between the UGV and landmark \( i \) shall be denoted as \( P_{ri} \).

Thus, yields:

\[
P_{rr} = AP_{rr}A + Q
\]

(7.25)

and also

\[
P_{ri} = AP_{ri}
\]

(7.26)

and \( P_{ir} = P_{ri}^T \) as explained in Section 7.5.2.1.

### 7.5.2.5.2 Step 2: Update State from Re-observed Landmarks

Due to odometry errors known as odometry shift, the previous update will not be accurate all the time. An update based on re-observed landmarks must occur. Whenever, a new landmark is re-associated with a previously observed landmark, there is a displacement between the previous position of the landmark and the new position of the landmark. This displacement is used to update the state of the UGV and also all the landmarks. Note, that analysis of the new landmark is left to step 3, in order to decrease the computational difficulty by using smaller matrices at this step.
The position of the landmark \((x_i, y_i)\) is predicted using Equation 7.4, which we shall denote as \(\hat{y}_{\text{landmark},i}\). This is compared to the position of the previously observed landmark, \(y_{\text{landmark},i}\):

\[
y_{\text{landmark},i} = \begin{bmatrix} x_{i,\text{stored}} \\ y_{i,\text{stored}} \end{bmatrix}
\]  

(7.27)

Equation 7.7 is the Jacobian from the landmark prediction model. The error matrix, \(R\), should be updated. Riisgaard & Blas (2005) recommend multiplying the range error term of the landmark by 1\% and the bearing error term by 1 for a 1° error.

At this point the Kalman gain can be determined from the following formula (Montemerlo & Thrun 2007):

\[
K = P_{\text{landmark},i} \times (C_{\text{landmark},i} P_{\text{landmark},i}^T + R)^{-1}
\]  

(7.28)

This gain now provides a number indicating how to weight the correction of the system state based on landmark re-observations.

Note, that the term \((C_{\text{landmark},i} P_{\text{landmark},i}^T + R)\) is called the innovation covariance, \(S_i\). This will be used in Section 7.5.3.

Finally we can compute a new state vector using the Kalman gain:

\[
x = x + K \times (\hat{y}_{\text{landmark},i} - y_{\text{landmark},i})
\]  

(7.29)

where \(\hat{y}_{\text{landmark},i}\) and \(y_{\text{landmark},i}\) are the values of the range and bearing of the newly observed landmark and the previously observed landmark, respectively. The term \(z - h\) is called the innovation and is denoted \(v_i\). Equation 7.29 updates the state of the UGV and all the landmark positions that are in the landmark database, and hence, in the state matrix. Note, that this process is iterated for every new observed landmark.

**7.5.2.5 Step 3: Add New Landmarks to the Current State**

This is a matter of updating the state matrix, \(x\) and covariance matrix, \(P\).

To update the state matrix, \(x\), it is a matter of appending the states of the landmark to the state matrix. The states are the x-coordinate and the y-coordinate of the landmark, thus,

\[
x = \begin{bmatrix} x \\ x_{N+1} \\ y_{N+1} \end{bmatrix}
\]  

(7.30)

The covariance matrix, \(P\), needs to be updated. This involves adding a new row and a new column. There are three covariance sub-matrices to add: the covariance matrix, \(P_{N+1,N+1}\), of the new landmark, the covariance matrices, \(P_{r,N+1}\) and \(P_{N+1,r}\), between the UGV and the landmark and the covariance matrices, \(P_{N+1,i}\), between this new landmark and the other landmarks already included in the system state.

The formulas for these cells (adapted from Bailey, Durrant-White 2006a) are:
\[ p_{N+1,N+1} = A_{x_t} P_{A_{x_t}} + A_{z} R A_{z}^T \]
\[ p_{r,N+1} = p_{r} A_{r}^T \]
\[ p_{N+1} = p_{r} A_{N+1} \]
\[ p_{N+1,i} = A_{x_t} P_{r}^{T} \]
\[ p_{i,N+1} = p_{i,N+1} \]

At this step, the process is complete and the system awaits the next iteration. This may include updating the system due to odometry update or due to the next observed landmark.

### 7.5.3 Data Association

Data association is described in more detail in Section 2.3.2. There are many possible methods for associating landmarks to other previously observed landmarks. The simplest method to implement is a validation gate that is used to compare the positions of two landmarks, one of which is newly observed and one that has been previously observed and is currently in the landmark database.

A validation gate works according to the following algorithm (Riisgaard & Blas 2005):

- If the pair passes the validation gate it must be the same landmark we have re-observed so increment the number of times we have seen it in the database.
- If the pair fails the validation gate, add this landmark as a new landmark in the database and set the number of times we have seen it to 1.

The two methods of validation gates encountered in literature were the simple Euclidean distance gate, but also the Mahalanobis distance gate. The Euclidean distance gate simply checks whether or not there is a previously observed landmark within a particular distance. Graphically, it can be thought of as a circle drawn around the position of the new landmark. If a landmark lies within that circle, then they are the same landmark. If there are two or more, than it is assumed to be the closest one.

The Mahalanobis distance gate is a similar technique but an ellipse is drawn around the position of the landmark with a larger radius in the direction of less certainty and the shorter radius in the direction in which the position of the landmark is more certain. The standard technique is to utilise the innovation calculated in Section 7.5.2.5. If the innovation of the landmark is \( v_i \) and the innovation covariance to the other landmark is \( S_i \), then the validation gate is:

\[ v_i^T S_i^{-1} v_i \leq \lambda \]  \hspace{1cm} (7.31)

where \( \lambda \) is a predefined distance.

The distance, \( \lambda \), used in the SLAM was 200 mm. This worked well for the simulation and is yet to be trialled for real data. 200 mm is also the maximum desired localisation uncertainty.

### 7.5.4 Landmark Extraction

Landmark extraction is the process of identifying landmarks in the laser data according to a predefined definition of landmarks.
7.5.4.1 FUNCTIONAL REQUIREMENTS FOR LANDMARKS

According to Riisgaard and Blas (2005),

(i) Landmarks should be easily re-observable,
(ii) Individual landmarks should be distinguishable from each other,
(iii) Landmarks should be plentiful in the environment,
(iv) Landmarks should be stationary.

Examples of landmarks that are not suitable for the process of localisation are moving objects such as people. If landmarks are to be considered in the light of the MAGIC competition, it is found that in both indoor and outdoor scenarios, features in the environment will be mainly objects of interest (OOIs), which may be either static or mobile, as well as walls and doors. Ruling out mobile objects as landmarks as they contradict the fourth tenet of landmarks, it is clear that the majority of the landmarks that will be encountered will be walls and door frames. Thus, it is essential to utilise a technique that will recognise objects such as walls and door frames and classify them as landmarks, but ignore objects such as OOI or humans. Section 2.3.3.1 details the different types of landmarks and landmark extraction methods. The three that will be considered are spike, RANSAC and scan-matching techniques.

The spike landmark extraction uses the existence of extrema in the laser data. Extrema can be defined in numerous ways. A simplistic way of finding extrema is to find two consecutive or nearby data points whose distance value, as calculated by the LiDAR, differs by more than a specified value such as half a metre. This will not differentiate between practical extrema and permanent changes in distance such as openings in walls. A more sophisticated approach is to consider three nearby data points and sum the differences between the first two points and the last two points respectively. If this sum is above a certain value, it implies the existence of an extremum. However, in the situation at hand, there will not be many instances where such extrema can be found. Certain static OOI may be found as extrema, but this method will also store people and mobile OOI in memory leading to false conclusions, as well as being ineffective in extracting any information about walls.

The RANSAC (Random Sampling Consensus) extraction method is a generic method of establishing the location of landmarks that have a pre-defined shape pattern. The data is sampled and heuristically correlated with the predefined shape until either all data points are considered or there is a set of points that have a high enough correlation, called a consensus. This set of points is the landmark and it is converted into a single value that can later be re-observed and associated with. In particular, this method is robust for straight lines and computationally inexpensive.

A third extraction method is called scan-matching where landmarks are sets of points that have a high enough correlation value for successive scans. As this requires the use of all data points for many scans, it is highly computationally intensive and seems to be inappropriate for the level of complexity proposed for MAGIC 2010.

Table 7.1 outlines the criteria for selection of the extraction method and it is clear that RANSAC is the best method for this project. It is imperative that the methods have robustness against moving objects such as people, be accurate, not be computationally intensive, recognise walls, and be reliable. These criteria have been chosen for their direct applicability to the project goals and as a result of what is known about the terrain the robots will be in. Additionally, as the robot will be doing a lot of calculation at once (localisation, mapping, navigation and visual recognition), the computations can be taxing on the CPU of the robots. Of course, the extraction methods must be accurate and reliable.
Table 7.1: Decision matrix for the landmark extraction method

<table>
<thead>
<tr>
<th>Function</th>
<th>Spike</th>
<th>RANSAC</th>
<th>Scan-matching</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robustness against people</td>
<td>10</td>
<td>25</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>Accuracy</td>
<td>12</td>
<td>12</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Computational Intensity</td>
<td>15</td>
<td>12</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Recognises walls</td>
<td>10</td>
<td>25</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>Reliability</td>
<td>8</td>
<td>10</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>Total</td>
<td>55</td>
<td>84</td>
<td>78</td>
<td>100</td>
</tr>
</tbody>
</table>

It is clear from the matrix that the RANSAC landmark extraction method is the most desirable landmark extraction method for implementation in this project.

7.5.4.2 RANSAC METHOD FOR LANDMARK EXTRACTION

Figure 7.6 demonstrates the types of landmarks this process will identify. The red dots indicate the LiDAR scan. The blue lines represent the straight line landmarks that will be extracted. Note, that a landmark will be extracted even if the line is interrupted by an object. The RANSAC method is very robust against outliers.

Figure 7.6: An example of line landmarks that would be extracted by the RANSAC method.
The following is a more step-by-step approach to explaining the technicalities of the RANSAC method. The basic structure for implementing RANSAC to find lines in the data is graphically exhibited in Figure 7.7 (adapted from an algorithm presented by Riisgaard & Blas 2005).

As can be seen from Figure 7.7, as each data set is read from the LiDAR we begin the algorithm. While there are still unassociated laser readings, and the number of readings is more than a consensus amount \( C \), and the amount of trials done is less than an upper limit \( N \), we repeat the following process:

(i) Select a random laser reading
(ii) Randomly sample \( S \) readings within 10 degrees of this laser reading
(iii) Using all these laser readings construct a least squares line of best fit
(iv) Determine how many laser readings are within \( X \) cm of this line of best fit

If the number of these laser readings is above a consensus \( C \), we recalculate this line of best fit and update the landmark database and remove the readings that were used. Otherwise, repeat the whole process. Note, that in Figure 7.7, the pink box with ‘go to next set of LiDAR data’, implies this algorithm is finished for this set of LiDAR data and that the rest of this data is discarded and the next set will now be analysed.
Note, that when a landmark is extracted there are usually two ways of storing that landmark. Either a map is generated on which the landmark is placed or alternatively a simple landmark database is created. As mapping has already been implemented and takes up a good deal of memory, it was decided that the better storage technique is the use of a database. This satisfies the requirements completely, because as is explained in Section 7.5.3, all the landmarks are needed for is a simple position comparison. The implementation used in this project stored the positions and other relevant information such as how many times it has been observed, number of timesteps since last observation, latest heading and range seen at, range and bearing errors and line equation coefficients in a C++ container class called a vector.

Data association and filtering requires the position of the landmark as a point. Thus far, the landmark has been extracted as a line and the equation of this line has been stored in the database. In order to convert this to a point, the ‘position’ of the landmark is defined as the closest point on the line to the origin on the map. The advantage of this representation is that it is unique and really easily invertible to give the equation of the line if need be. Moreover, as this point is just a translation along the landmark line of where the landmark was observed the difference in landmark position observed at the point of observation will directly translate to this representation point. This can be observed in Figure 7.8.

![Figure 7.8: Representation position of line landmark](image)

7.5.4.3 Testing

By altering the values of S, C, X, and N from Section 7.5.4.2, this process can be refined to improve accuracy and reliability. The simulation used (See Section 7.6.) to test the SLAM process had input noise of size 200mm for each data scan. The major problem with most trialled values for S, C, X, and N was that landmarks would be split up into several landmarks due to the noise disturbances. Eventually, a set of values was found that worked to extract landmarks correctly.

The final values used for these variables are:
7.6 SIMULATION RESULTS

The SLAM process utilising all the work explained thus far, was tested using a simulation of the UGV moving through test environment. The environment is shown in Figure 7.9. It is a test map written in a text file utilising the numbers 0, 1, 2 and 3. The zeros represent empty space, a one represents a wall, a two is an object-of-interest and a three is any other object such as a person or a tree or anything of that sort. Figure 7.9 has been colour-coded for ease of viewing. The black arrows represent the path of the UGV. Note at the corners of this path, the UGV turns 90° on the spot.

A few simple observations were made about the test map, and therefore, of the value of the test. Firstly, the movement of the UGV is simple and in and of itself contains no disturbances. For this reason, the simulated IMU data had no noise present. At all times excluding turning, the acceleration was zero. Secondly, each number on the test map represents a 200mm × 200mm square of the environment. Thus, the simulated LiDAR data, does not give an exact value of the range to each point on a wall, for example, but gives the shortest distance to the square. This creates an inherent noise value in the LiDAR data of uncertainty 100 millimetres. It was determined, that it would be a great test to determine the accuracy of the landmark extraction and data association method.

For this reason, the simulation provided the means to test various values for variables such as X, C, S and N for the RANSAC landmark extraction method and also λ for the Mahalanobis validation gate in the data association. These will be further discussed in Sections 7.6.1 and 7.6.2, respectively.

7.6.1 TESTING LANDMARK EXTRACTION

The testing of landmark extraction was primarily through the variation of the parameters X, C, S and N used in the RANSAC landmark extraction process. X represents the proximity with which LiDAR data values must lie to the line extracted to be included in the data values on the new line. S represents number of data values used in initial calculation of line of best fit. C represents number of points needed for final line to be considered a landmark. N is number of iterations of the RANSAC method before the next data set is considered.

Because of the simplicity of the map, it is known what the landmarks are and what their positions are. Moreover, there should be three of them at most times, due to the three walls that are visible to the UGV. This information is used when trialling different values for these RANSAC parameters.

The first parameter that was tested was N. It needed to be large to make sure landmarks are extracted, but not too large as to lag the system. It was found that N = 100 gave a good compromise between speed and number of iterations.

The next variables tested were S and X. The smaller S, the more likely local disturbances are going to cause the initial line to be too far skewed from the true line. In such a case, X would have to be larger. Choosing X = 500mm, as an initial value, meant S could be no less than 7. Iterating values for S and seeing the allowable values for X found that setting S = 10, allowed X = 375mm. Thus, the
values chosen, expecting more noise in real environments were $S = 10$ and $X = 400\text{mm}$. This was done with $C = 30$, a reasonable value according to Durrant-Whyte and Bailey (2006a). Therefore, this was the final choice for $C$. 

Figure 7.9: The test map utilised to simulate UGV movement and for testing SLAM algorithm
7.6.2 TESTING DATA ASSOCIATION

Once landmarks were being identified regularly in the right numbers from the work in Section 7.6.1, the data association was tested. The value that needs to be tested here is $\lambda$ for the validation gate. This was done for both the Euclidean and Mahalanobis distance applications. It was first tested with Euclidean distance, as this was expected to be a lower bound for the Mahalanobis distance value.

The desired localisation accuracy is for the estimated position to be within 200mm of the actual position. So, $\lambda$ was tested from a value of 100mm upwards in increments of 20mm. The values that worked for the Euclidean distance was 180mm for the Euclidean distance in 92% of reobservations and 200mm in 99% of reobservations. Utilising Mahalanobis distance found that $\lambda = 200$mm, was a sufficient value with correct reobservations in all but two of a few thousand readings.

7.6.3 TESTING ENTIRE SLAM PROCESS

Because of the fact that there was no noise in the IMU data and uniform noise in the LiDAR data, the testing of Kalman filter specific values was left until the SLAM was tested in real-time on a UGV. These values will include the initial state matrix, $x_0$, and initial covariance matrix, $P$, and also the values in the measurement and process error matrix terms ($R$ and $Q$ respectively).

A graph of the accuracy of the testing with all selected values is shown in Figure 7.10. Note, that the biggest deviations from the true values occur when the UGV is turning.

The biggest deviation was a value of 280mm. This is a desirable result, so that after some more testing on real data that will be used to alter some of the EKF terms, this may be reduced further.

The SLAM program was used on a data set collected from a test run of the UGV that contained coupled IMU and LiDAR data, however, no real significant values for the position were recorded so there is nothing to correct against. The test was done to check that noisy IMU data will still allow the extended Kalman filter to converge. This was the case.

7.7 FUTURE WORK

As of the preparation of this report, the SLAM has been successfully written and adapted to a simulation of the UGV moving through an indoor environment. The results have been analysed and shown to localise the UGV to within 280mm. Moreover, a quick analysis of landmarks shows that they correspond to the walls modelled in the test map provided with the simulation. The SLAM has also been applied to a set of data that contains coupled IMU and LiDAR data. The results, however, are difficult to prove as there was only a general idea of the direction in which the UGV was moving, as well as the fact that the test was outdoors and very few landmarks were observed.

Further work will include proving the system on-board a UGV in real-time. Once the system has been adapted to the UGV and working reasonably accurately, it will be valuable to work on the integration of the outdoor and indoor localisation systems. Section 7.3 involves an explanation of the stages of integration that are planned for the localisation system.

Furthermore, as SLAM is a rich ground for research there are other directions in which future work could progress. One of the disadvantages of SLAM is the large amount of computational power required in order to utilise the system. There is a lot of opportunity to streamline the system. In particular, the system utilised for this project has the processes of landmark extraction, data association and filtering done in three separate algorithms and contains some of the same
Figure 7.10: The difference between true and estimated position for the SLAM simulation

information stored in various places. The system will be more efficient if these processes are integrated into one another. Other efficiency strategies are the dynamic resizing of the state matrix to only incorporate landmarks that may be reobserved and keep unobservable landmarks in memory until needed.

Another possible direction of future work is a different filter method. There have been advances in the area of SLAM utilising unscented Kalman filters or particle filters. The possibilities for developing this further are endless.
Mapping is an important software system that is used to create a digital representation of the environment in which the Unmanned Ground Vehicles (UGVs) are located. Mapping is used for navigation and control purposes, as well as displaying various characteristics of the environment on a Human Machine Interface (HMI) for humans to see and understand. For the purposes of the MAGIC project, mapping is divided into two main parts, namely Physical mapping and Conceptual mapping, and these parts will be discussed in more detail in the following sections.

### 8.1 Physical Mapping

The flowchart for the physical mapping system is displayed in Figure 8.1. The occupancy grid is the foundation of the physical mapping system with all physical maps deriving information from it. The physical maps that will be discussed are the elevation map, the local and global 3D physical maps, and the local and global visibility maps.

The flowchart for the physical maps systems is displayed in Figure 8.1. The occupancy grid is the essential foundation of the physical maps system with the elevation and feature maps deriving information from it. The physical maps that will be discussed are the occupancy grid, the elevation map, and the visibility map.

![Physical Map Flowchart](image-url)
8.1.1 OCCUPANCY GRID

The occupancy grid is a global three dimensional grid which stores data regarding the probability of the existence of obstacles in the environment along with their position relative to the sensor. The occupancy grid accepts position and orientation (pose) information from the localisation system and range readings from the LiDAR to construct a 3D probability map of the environment. The occupancy grid is an integral aspect of the UGV’s mapping system and is used to create a digital representation of the physical environment surrounding the UGV. The information stored within the occupancy grid is used to create physical and conceptual maps. The main role undertaken by the University of Adelaide in working on the occupancy grid program has been in debugging, testing and optimisation. The following section will discuss the work undertaken by the University of Adelaide on the occupancy grid program including the choice of the grid cell size used within the occupancy grid, a derivation of the coordinate transformations used to transform LiDAR coordinates into Ground Control Station (GCS) coordinates, program optimisation, and program testing.

8.1.1.1 OCCUPANCY GRID CELL SIZE

The choice of cell size in the occupancy grid is important as a smaller cell size will allow for a greater resolution, however, it is also associated with increased data storage and processing, and increased susceptibility to sensor errors. The resolution of the occupancy grid determines the degree to which the UGV can interpret and interact with the environment. Hence the choice of grid cell size is ultimately governed by the choice between maximising resolution and minimising data storage, processing, and sensor errors. The following section will discuss the choice of occupancy grid cell size with regard to the need of the UGVs to interact with the environment and limitations in resolution due to sensor errors.

The need of the UGVs to interact with the environment is governed by the desire to meet MAGIC competition requirements wherein the UGVs are required to pass through doorways of width 900mm and traverse ramps of height 320mm. In order to pass through a door of width 900mm it is desirable that the grid cells be a maximum of 300mm wide, as then an open doorway can be represented by three empty cells and the UGV will be directed towards the middle cell to ensure it does not collide with the door frames. A minimum of three cells is desired to represent open doors, because if a UGV is passing through a doorway and it detects that it is not in the middle cell then corrective action can be taken, whereas with less than three cells the UGV will collide with the door frame before corrective action can be taken. In order to negotiate ramps in the MAGIC arena it is desirable for the grid cells to have a maximum height of around 100mm so that ramps can be identified as a series of steps in the occupancy grid. Ultimately the identifiability of the ramp in the occupancy grid is determined by the grid cell sizes in both the horizontal and vertical planes. High resolution in the horizontal plane will allow for an accurate representation of where the ramp begins and ends, and high resolution in the vertical plane allows for easier ramp identification using height variations. Hence, the maximum grid cell dimensions set by the desire to meet the MAGIC competition objectives are 300mm wide by 100mm high, however this is the maximum resolution attainable for the computational ability available for the MAGIC project, and a higher resolution is attainable at the cost of increased computational expense.

The minimum grid cell size depends on the beam spread of the LiDAR, data storage, and processing. An extract from the Leuze Rod4-08 plus user manual detailing beam spread over distance is displayed in Figure 8.2. Whilst beam spread is only indicated up to a distance of 25 metres in Figure 8.2 a personal communication with Leuze (12 April 2010) confirmed that linear extrapolation can be done to obtain values outside of this range. At a distance of 50 metres, which is the maximum
useful range of the LiDAR, the beam spread in the vertical plane is approximately 240mm. Hence, for a cell size of 100mm, measurements taken at this distance will be using a LiDAR beam which overlaps three grid cells. The effect of the LiDAR beam overlapping cells is that, at long distance, errors will be produced in the occupancy grid, however the probability algorithm in the occupancy grid can correct this as the UGV approaches the affected area and beam spread becomes less pronounced. Whilst the effect of beam spread can be corrected, the degree of correction that can be achieved will depend on grid cell size and beam spread close to the UGV. For a cell size of 100mm in the vertical plane the grid will be relatively accurate for areas close to the UGV which is important as this is the environment that the UGV interacts with. Decreasing the grid cell size beyond this level could produce inaccurate data, hence, 100mm is the grid cell size limit set in the vertical direction to avoid errors in the occupancy grid from LiDAR data. It can be observed from Figure 8.2 that the beam spread of the LiDAR in the horizontal plane is much less pronounced than the beam spread in the vertical plane, therefore the grid cell width can be reduced to below 100mm without compromising the accuracy of the occupancy grid. Whilst it might be desirable to reduce the grid cell width below 100mm to increase resolution, the MAGIC competition requirements detailed previously can be met using a 100mm cube for the occupancy grid cells, hence increased resolution would result in unnecessary computational expense.

Figure 8.2: Beam Spread of LiDAR versus Distance (Leuze Electronic, p. 45)

8.1.1.2 COORDINATE TRANSFORMATIONS DERIVATION

To satisfy the objectives of the MAGIC competition the UGVs are required to cooperate as a team in locating and neutralising OOI and in producing a map of the surrounding environment. For the UGVs to cooperate as a team it is necessary that they know the position and state of one another to facilitate strategic planning and task handling. As the behaviour of the UGVs will be influenced by their interaction with the surrounding environment it is also necessary that the UGVs have knowledge of the environment surrounding each other. Hence to facilitate UGV cooperation, navigation and strategic control will need to be applied at a global level to take into account the
Mapping

states of all of the UGVs with respect to each other and the environment. Therefore, both the need for strategic interaction between UGVs and the desire to meet the MAGIC competition objectives drive the need for global maps. As all of the maps derive information at some level from the occupancy grid, the production of global maps requires the production of a global occupancy grid that encompasses the surrounding environment.

To create a global occupancy grid the LiDAR values obtained for each UGV need to be transformed so that the positions of obstacles in the occupancy grid are known relative to the Ground Control Station (GCS). To achieve this, coordinate frames are placed in known locations on the UGV and GCS and coordinate transformations are employed to express the positions of obstacles in terms of different frames. The steps involved in expressing the position of an obstacle seen by the UGV in terms of a position vector from the GCS are summarised as follows:

(i) The distance measurement obtained from the LiDAR is converted into a position vector in Cartesian coordinates relative to the centre of the LiDAR, $L^P$.

(ii) A coordinate transformation which maps LiDAR coordinates to pan-tilt coordinates is applied to the position vector of the obstacle so it can be written in terms of pan-tilt coordinates, $PT^P$.

(iii) A coordinate transformation which maps pan-tilt coordinates to pose coordinates is applied to the position vector of the obstacle so it can be written in terms of pose coordinates, $pose^P$.

(iv) A coordinate transformation which maps pose coordinates to GCS coordinates is applied to the position vector of the obstacle so it can be written in terms of GCS coordinates, $GCS^P$.

The coordinate frames assigned to each UGV can be seen in Figure 8.3.

The following section will detail each of these steps involved in transforming LiDAR coordinates to GCS coordinates in the order which the steps are done.

The values obtained from the LiDAR are distance measurements in polar coordinates. To change the LiDAR values from polar coordinates to Cartesian coordinates Equations 8.1, 8.2 and 8.3 are employed where $D$ is the distance measurement returned by the LiDAR, $\theta_L$ is the polar angle of the LiDAR measurement in the horizontal x-y plane, and $\theta_t$ is the tilt angle of the pan-tilt unit.

$$x = D \cos \theta_L$$  \hspace{1cm} (8.1)

$$y = D \sin \theta_L$$  \hspace{1cm} (8.2)

$$z = D \sin \theta_t$$  \hspace{1cm} (8.3)

In order to be compatible with transformation matrices, the LiDAR coordinates are modified into the form $L^P = (x, y, z, 1)^T$ where the added one is a global scale factor.

To express the position vector $L^P$ in terms of pan-tilt coordinates the LiDAR centred coordinate frame is rotated about its X axis by the tilt angle $\theta_t$, its Z axis by the pan angle $\theta_p$, and then translated by $(x_1, y_1, z_1)^T$ where $x_1$, $y_1$ and $z_1$ are the Cartesian coordinates specifying the origin.
of the LiDAR frame with respect to the centre of the pan-tilt unit. The transformation matrix used to transform LiDAR coordinates into pan-tilt coordinates is displayed in Equation 8.4, where $s\theta$ and $c\theta$ represent $\sin\theta$ and $\cos\theta$ respectively.

\[
PT_L = \begin{bmatrix}
\theta_p & -s\theta_p & 0 & x_1 \\
c\theta_t s\theta_p & c\theta_t c\theta_p & -s\theta_t & y_1 \\
s\theta_t s\theta_p & s\theta_p c\theta_p & c\theta_t & z_1 \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (8.4)

Therefore the position vector of the obstacle can be written in pan-tilt coordinates as $PT_P = PT_L PT_P$.

To express the position vector $PT_P$ in terms of pose coordinates the pan-tilt frame is translated into the pose frame by $(x_2, y_2, z_2)^T$ where $x_2$, $y_2$ and $z_2$ refer to the origin of the pan-tilt frame with respect to the centre of the pose frame. The transformation matrix used to transform pan-tilt coordinates into UGV pose coordinates is displayed in Equation 8.5.

\[
\text{pose}_{PT} = \begin{bmatrix}
1 & 0 & 0 & x_2 \\
0 & 1 & 0 & y_2 \\
0 & 0 & 1 & z_2 \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (8.5)
Hence, the position of the obstacle can now be written in pose coordinates as \( \mathbf{P} = \mathbf{P}_T^T \mathbf{L}_T \mathbf{P} \).

To express the position vector \( \mathbf{P} \) in terms of GCS coordinates the pose frame must be rotated about its X, Y, and Z axes by the pitch, roll and yaw angles obtained from the Inertial Measurement Unit (IMU). The pitch, roll and yaw angles are designated as \( \theta_x, \theta_y, \theta_z \) respectively where the subscript refers to the axis of rotation. Also, the frame must be translated by the pose coordinates \( (x_{\text{pose}}, y_{\text{pose}}, z_{\text{pose}}) \). The transformation matrix used to transform UGV pose coordinates into global coordinates relative to the GCS is displayed in Equation 8.6.

\[
\mathbf{GCS}_{\text{pose}}^T = 
\begin{bmatrix}
  c\theta_z c\theta_y & -s\theta_z c\theta_x - s\theta_y c\theta_z s\theta_x & s\theta_z s\theta_x - s\theta_y c\theta_x c\theta_z & x_{\text{GPS}} \\
  s\theta_z c\theta_y + c\theta_z s\theta_y & c\theta_x c\theta_z - s\theta_y s\theta_z s\theta_x & -s\theta_x c\theta_z - s\theta_y s\theta_z c\theta_x & y_{\text{GPS}} \\
  s\theta_y & c\theta_y s\theta_x & c\theta_y c\theta_x & z_{\text{GPS}} \\
  0 & 0 & 0 & 1
\end{bmatrix}
\] (8.6)

Hence the positions of the obstacles in the environment surrounding the UGVs can now be written in terms of GCS coordinates as \( \mathbf{GCS} = \mathbf{GCS}_{\text{pose}}^T \mathbf{P}_T^T \mathbf{L}_T \mathbf{P} \).

These transformations allow the positions of objects detected by the UGV to be known in a global context relative to the GCS, so that global maps can be produced. The production of global maps satisfied the objectives of the MAGIC competition by providing a complete map of the surrounding environment and facilitated strategic planning and UGV cooperation and task handling.

### 8.1.1.3 Occupancy Grid Program Optimisation

The beta occupancy grid program obtained from Strategic Engineering was functional upon receipt, however contained some undesirable characteristics which were altered to optimise the program in terms of its functionality and the results produced. The alterations done on the program involved integrating it into the libmagic framework used for mapping on the GCS, and changing the probability formula used to store and update values in the occupancy grid. The results of these changes made the program simpler and easier to debug, and also yielded significant improvements in processing speed, program size, and robustness with regard to UGV position and orientation errors.

The optimisation performed on the occupancy grid probability algorithm lead to the alteration of the data type used to store probability information and the incorporation of conditional statements to prevent variables from approaching probability limits. The probability algorithm that was originally used within the occupancy grid was Bayesian Inference which uses values between zero and one to represent the probability of the existence of objects in the environment where, theoretically, the probability value contained within a cell should never reach the limits of zero or one. The undesirable characteristics of this method were twofold. Firstly, an occupancy grid that is 2 metres high, spans 4.9 hectares and uses 64-bit double precision floating point data types to store probability values in order to encompass the proposed MAGIC competition area required approximately 6 gigabytes of RAM at the GCS, which was significant. Secondly, a double precision floating point data type has limited precision for incrementing (can only reach 0.999... to twelve decimal places before the number will overrun to one) and a limited range for decrementing (can...
only be decreased to $2.22507 \times 10^{-308}$ before the number will overrun to zero), hence probability values used within the Bayesian framework of the occupancy grid do not asymptote towards the extreme values of zero or one as desired, and instead can actually reach the limits of zero or one. Probability values reaching the limits of zero or one are undesirable since upon the limits being reached the Bayesian Inference algorithm cannot alter the probability value stored in the cell. Therefore if a UGV receives erroneous pose information for a period of time then the occupancy grid can be permanently updated with erroneous data. The solution to both of these problems was to replace the 64-bit double precision floating point data types used to represent probability values between zero and one with 8 bit unsigned integers that stored the probability as an integer between zero and 255. The reduction in RAM requirements due to employing 8 bit unsigned integers was 5.25 gigabytes, which was a much more sensible choice.

The Bayesian Inference algorithm could not be used with unsigned integers hence it was replaced with an integer increment or decrement depending on the detection of an obstacle or vacant space respectively. For example, the initial probability value stored in a cell is 128 as this is halfway between zero and 255 and thus represents a state of uncertainty. If an object is detected once in an occupancy grid cell then the value stored within the cell is incremented to become 129, and conversely if vacant space is detected then the value is decremented to become 127. Essentially, the probability scale between zero and one has been normalised to be between zero and 255 to compensate for the fact that unsigned integers cannot store decimal figures, however the overall idea of representing the environment surrounding the UGVs as values of probability has not changed. Therefore the integer increment/decrement algorithm performs essentially the same task as the Bayesian Inference algorithm in that multiple event observations are logged to determine object existence. The advantage of the integer increment/decrement algorithm is that faster map updating can be achieved due to the fact that character incrementation is faster to perform than float multiplication. In addition, as the integer increment/decrement algorithm also uses multiple event observations to infer object existence, sensor noise is filtered from data. Hence the integer increment/decrement algorithm is a robust and computationally inexpensive replacement of Bayesian Inference.

The improvement in performance achieved by employing 8 bit unsigned integers was quantified by using timer modules throughout the occupancy grid program to determine function times, where the use of 8 bit integers reduced the time taken to update the occupancy grid by 50%. The problem of permanent map updating was addressed by incorporating conditional statements into the probability updating algorithm to prevent probability values being updated beyond the limits of 255 or zero. Whilst the incorporation of conditional statements was necessary to prevent overrun errors the benefits produced by preventing probability values from reaching their limits are that the occupancy grid cannot have permanent errors embedded in it and it can deal with dynamic objects such as mobile OOI, which Bayesian Inference could not reliably achieve.

### 8.1.2 Occupancy Grid Testing

The primary role of the University of Adelaide with regards to the occupancy grid was debugging and testing. The testing of the occupancy grid involved testing of both the data obtained from the LiDAR and the accuracy of the occupancy grid in static and dynamic situations.

#### 8.1.2.1 Test Platform

The platform used for dynamic testing of software and LiDAR data was the Pioneer 3-AT mobile robot owned by the University of Adelaide. The code used to program the Pioneer robot was
written in the C++ programming language using the Mobile Robot’s ARIA library and compiled in MS Visual 2003. The robot was programmed to perform basic tele-operation which is the process of controlling the robot’s motion using input from a computer keyboard, and to display velocity and heading values to the screen every 100 ms. The setup of the hardware on the Pioneer robot that was used for testing is displayed in Figure 8.4.

The intentions of the first test were to validate LiDAR data obtained from the LiDAR driver, and test the rate at which the occupancy grid program processed data. To validate the data an environment that was relatively empty with well defined boundaries was desired. Also it was desired that the environment contain open doorways as one of the challenges of the MAGIC competition is for the robot to negotiate doorways. It was decided that the hallway by S237 in Engineering South met all of the requirements of being a suitable environment; hence this is where testing occurred. To test the rate at which the occupancy grid processed the data obtained from the LiDAR driver, the LiDAR data was written to a log file at the completion of every occupancy grid program cycle.

The method of the first test was to drive the robot down the hallway and log LiDAR data and camera video. The intention of logging both data sets was to create a video of what the LiDAR was looking at that could be synchronised with the LiDAR data. A MATLAB program capable of reading the LiDAR data from the log file and displaying it to screen in a graphical form was created by Chris Madden, the output of which is displayed in Figure 8.5.

The LiDAR data itself is fairly unintuitive, hence the desire for it to be synchronised to a video of what it is displaying. It can be observed however that the data set does contain well defined lines, specifically on the right and the top of the figure, and these lines meet in a well defined corner. The existence of well defined lines and readings on the vertical axis on the chart indicate that the LiDAR is approximately 1.5 metres away from the corner of a room. Also the lack of a defined line on the left side of the display represents vacant space, which indicates that at this point in time the robot was about to turn a corner in the hallway. While the data obtained was
representative of the surrounding environment, it was observed that there was significant variation in some of the recordings, specifically in the section marked by the red oval. While it was desirable to combine the LiDAR data with the video that was recorded to determine the cause of this, the error in this test session was that timestamps and robot velocities were not recorded with the LiDAR data, hence it could not be synchronised with the video taken as the position of the LiDAR could not be determined. As the LiDAR data could not be synchronised with the video, it could not be determined whether the variations in the data were due to specula reflection, a person walking past, or a problem with the LiDAR, hence a second testing session was arranged. The first testing session also indicated significant program lag. This was evident as only 98 scans were logged in a testing session of about four minutes, which is significantly less than the 25 scans per second output by the LiDAR.

8.1.2.3 Lidar Data Testing: Second Test
The intention of the second testing session was to log LiDAR data in a well defined environment and synchronise it with the camera feed. As the intention of the second test did not involve testing program cycle times, the communication between the LiDAR driver and the occupancy grid was removed and data was written to a log file from the LiDAR driver. The method of the second testing session was the same as the first, except that the Pioneer robot was programmed to produce a log file containing velocity and heading information with timestamps indicating when measurements were taken. This was produced in the following format:

Thu Apr 08 11:51:32 TransVel:224 RotVel:11 Heading:10 TransAcc:-5 RotAcc: -2

Velocity and acceleration measurements were logged in mms$^{-1}$ and mms$^{-2}$ respectively. The heading was obtained using encoder information from the Pioneer robot and logged in units of
degrees. Rotational velocity and acceleration refer to motion about the robot’s yaw axis. Data was logged at a rate of ten logs per second. The LiDAR data from the second test session was not logged in the same format as it was in the first test hence the MATLAB display program had to be edited. The change in logging format was an unintentional mistake, and to ensure that the problem would not re-occur logging standards were defined. The output from one of the LiDAR scans from the second test is displayed in Figure 8.6. In Figure 8.6, a hallway can clearly be defined by parallel lines on the sides of the scan. A doorway can also be clearly defined as indicated by the red oval. The above figure demonstrates that the data logged by the LiDAR driver is representative of the surrounding environment, indicating that accurate data could be obtained for software development.

Figure 8.6: MATLAB Graphical representation of one LiDAR scan obtained from the second testing session. The red line represents a doorway as seen by the LiDAR.

8.1.2.4 Static occupancy grid testing

The intention of the static occupancy grid tests were to test if the occupancy grid could use data from the LiDAR to produce maps that were an accurate interpretation of the static environment. Static testing was done by aiming the LiDAR at objects with well defined features and observing their representation in the occupancy grid produced. Figure 8.7 indicates the environment created for a static test along with the corresponding occupancy grid produced. Static occupancy grid testing was undertaken before the Bayesian Inference probability formula was replaced with an integer increment, hence the grid is comprised of floating point values where:

- 0 is the low limit of probability of an object existing in the grid cell,
- 0.5 is the starting value which indicates uncertainty about the contents of the grid cell,
- 1 is the high limit of probability of an object existing in the grid cell.
It can be observed that the occupancy grid produced was an accurate interpretation of the environment surrounding the LiDAR. Static testing was undertaken multiple times in varying situations to thoroughly test the accuracy of the occupancy grid.

Figure 8.7: Static occupancy grid testing conditions and occupancy grid produced.

### 8.1.2.5 Dynamic Occupancy Grid Testing

The intention of the dynamic occupancy grid tests were to test if the occupancy grid could use data from the LiDAR to produce an accurate interpretation of a changing environment. Static tests of the grid were performed to confirm accuracy of environmental interpretation, and dynamic tests were performed to determine if an accurate grid could be produced in a changing environment. A changing environment was produced by attaching the LiDAR to the Pioneer Mobile Robot in the same manner as indicated in Figure 8.8. The same method of driving the robot down the hallway in Engineering South used in the LiDAR data testing sessions was used for dynamic testing. At the time of testing neither GPS nor differential GPS were available for use, hence, real-time positional and velocity data were not available for use within the occupancy grid program. To overcome this limitation an approximate synchronisation between the program and the Pioneer robot was done by manipulating the GPS coordinates in the occupancy grid program. The \( y \) value of the GPS coordinate in the occupancy grid program was incremented by a set amount every program cycle to produce a simulated forward velocity in the program, i.e. if the cycle time of the program is 0.1 seconds and the \( y \) value of the GPS coordinate is incremented by 10mm every program cycle, then the program’s interpretation of this is that it is moving forward at 100mm/s. The simulated velocity in the occupancy grid program was chosen to be approximately 200mm/s hence the velocity of the Pioneer robot was set to 200mm/s during tests.

Dynamic testing was done after the Bayesian inference probability formula was replaced with an integer increment. Hence, the grid is comprised of unsigned character values where:

- 0 is the low limit of probability of an object existing in the grid cell,
- 128 is the starting value which indicates uncertainty about the contents of the grid cell,
- 255 is the high limit of probability of an object existing in the grid cell.

Figure 8.8 displays an occupancy grid of the hallway by S237 produced during a dynamic test. It can be observed that the occupancy grid produced well defined boundaries of the hallway indicated by maximum probability values of 255, and well defined vacant space indicated by minimum probability values of 0. Also as people were walking past the robot during testing it is evident that only the stationary physical environment is represented and slow moving objects are not embedded
in the grid, which is desired. The occupancy grid produced was an accurate interpretation of the hallway by S237 indicating that the program responded as desired to environmental changes. Dynamic testing was completed multiple times in varying situations to thoroughly test the accuracy of the occupancy grid and its response to environmental changes.
Figure 8.8: Occupancy grid of hallway by S237 produced during dynamic test
8.1.3 Visibility Map

The visibility map is a global two dimensional map which displays the collective sum of UGV visibility. The visibility map uses unsigned 8-bit integers to store data about the environment surrounding the UGVs regarding two states: what is visible to the UGVs and what is not visible to the UGVs. The visibility map uses information stored in the feature map combined with the poses of the UGVs and ray tracing techniques to determine the areas of the feature map which are visible to the UGVs. The areas of the feature map that are visible to the UGVs are assigned a value of one in the visibility map, and the areas that are not visible are assigned a value of zero. The visibility map is created from inference of UGV visibility rather than incoming LiDAR data as creating a cordon line of UGV visibility, is difficult using LiDAR data as it can be random and noisy. The visibility map is not a map that is continually updated, but instead is generated only when visibility information is required.

8.1.4 Elevation Map

The elevation map is a global two dimensional map which stores information regarding the elevation of the environment surrounding the UGVs. The elevation map is derived from the occupancy grid, by counting from the bottom of the grid upwards, the number of adjacent occupied cells within the occupancy grid columns and then using this number as a measure of ground or obstacle height. The elevation map is derived from the occupancy grid as opposed to LiDAR data as the occupancy grid provides data and noise filtering. The disadvantage of deriving the elevation map from the occupancy grid is that the resolution is limited to 100mm as opposed to 1mm if LiDAR data were used. However the advantage of a reduced resolution is that the elevation map can store values as 8 bit integers whereas 16 bit integers or 32 bit floating point values would be required to store elevation values in millimetres derived from LiDAR data. The advantages of an 8 bit elevation map as opposed to a 16 or 32 bit map are that the memory required to store the map is reduced, and the values stored within the elevation map are the same type as those stored in other maps, hence no type conversions are needed if elevation data is referenced by other maps. The elevation map is displayed on the HMI, as seen in a simulation in Figure 8.9 where the elevation of ground level in the occupancy grid is represented by blue and elevations above ground are represented by shades of red where a brighter red indicates a higher elevation. The lower and upper elevation limits of the elevation map are 0.5m below ground and 1.5m above ground respectively which are imposed due to the elevation map being derived from the occupancy grid.

Figure 8.9: The elevation map displayed on the HMI. The elevation of ground level in the occupancy grid is displayed as blue and elevations above ground are displayed as shades of red where a brighter red indicates a higher elevation.
8.2 CONCEPTUAL MAPPING

The second part of mapping is known as Conceptual Mapping. The fundamental difference between physical maps and conceptual maps is that conceptual maps are constructed in a such a way that humans can easily understand what the maps are showing. Whilst physical maps contain information about the environment, this information is not easily understood by humans and has to be interpreted in some useful way. Conceptual maps take information from the occupancy grid and using a set of pre-defined rules interpret this data to isolate particular features and create different conceptual maps. A conceptual map is designed to display one particular characteristic of the environment. All conceptual maps are designed to be visually easy to understand, and are displayed on a Human Machine Interface (HMI). For the MAGIC 2010 project, three conceptual maps were developed, namely the Exploration Map, Feature Map and Vantage Map. The following sections describe the motivation behind each map, a brief description of how the map works, the results obtained from testing and finally, future work for this section. Figure 8.10 shows an overall architecture of the conceptual maps.

![Conceptual Map Flowchart](image)

Conceptual maps are created and updated continuously as the field situation changes. The map information is generated locally by each UGV and sent dynamically to the Ground Control Station (GCS), where all the locally generated information will be combined to form a global map. Constant communication between the UGVs and the GCS ensures that all global conceptual maps are continually updated, and that all UGVs are aware of the approximate location of fellow UGVs, OOLs and other obstacles. All conceptual maps are programmed in C++, and store information in the form of numeric arrays. In order to make the maps visually easy to understand, the HMI was used to display the various maps (See Section 5.4.1). The HMI shades the squares on the map area being explored appropriately. For instance, if a map is stored as a value of 0 or 1, then the HMI would shade the square with the value of 0 as red and 1 as blue (which is easier for humans to understand as opposed to an array of 1s and 0s). Each conceptual map implemented has its own tab in the HMI and relevant capability to display that map through colour.

8.2.1 MAP TESTING

All conceptual maps were subject to the same testing procedure. The testing mainly consisted of two stages. In the first stage, the conceptual map was tested using a software simulated input. Since the input is simulated, the input data is one hundred percent accurate, and there is no accumulation of errors carried forward. Hence, this stage only checks the accuracy of the code and to verify if the
code is producing the desired output. Once the first stage was successfully passed, the second stage of testing was to be conducted. In the second stage of testing, the conceptual maps were to be tested in a real environment. The inputs are from real sensors and the occupancy grid. This stage is critical in order to determine and eliminate, where possible, sources of error and integration issues with other software. Given the inherent uncertainty in position, if the conceptual map results are within individually set acceptable error bounds, then they will be deemed to have passed the second stage of testing. However, due to time constraints and software integration issues, the second stage of testing could not be performed completely. The UGV was operated outdoors, and real time data was logged, however there were a few issues in communicating the data to the HMI, and these issues could not be resolved in time to get the desired results.

8.2.2 EXPLORATION MAP

The first conceptual map created was the exploration map. As indicated by the name, this map shows which areas of the field have been explored, and which areas have not been explored. The motivation behind this map is twofold. Firstly, prior to exploring an area, no information is known about that area, such as the features, obstacles and presence of OOi. Hence it is critical to know which areas have and have not been explored. Secondly, since the MAGIC project is a collaborative effort by a team of UGVs, it is essential that all UGVs be aware of areas that have been explored, and not explore the same area again, unless required. Having a global exploration map will aid in avoiding wasting time exploring already discovered areas.

8.2.2.1 HOW IT WORKS

The exploration map is completely based on the LiDAR readings. The LiDAR sends out multiple laser beams. The environment around the UGV is divided into a grid of 3D cubes, as explained previously. If a laser beam from the LiDAR passes through a cube repeatedly, it can safely be said that the cube has been explored. It is important to note that the laser beam must pass through the cube more than once in order to assume that the cube is explored. This is because a single pass of a laser beam may occur randomly, and is usually not enough to obtain information about the cube. This information includes probability of occupancy and any features that may be present. For the MAGIC project, it was decided that the laser beams must pass through any cube thirty times before the cube is said to have been explored. The exploration map is a 2D Boolean array of 1’s and 0’s, where a ‘1’ indicates that the square has been explored, while a ‘0’ indicates that the square has not been explored. Initially, all squares are set to 0 because no area has been explored yet. When a square has received thirty or more hits from the LiDAR, its value in the exploration map is set to 1. The HMI is then programmed to shade the explored areas with a certain colour, and the unexplored areas with another colour, in order to clearly differentiate between areas that have been explored; and have not been explored.

8.2.2.2 TESTING

The first stage of testing was done by loading a pre-defined static image of an environment, and running the code in order to see if the code works as desired. However, it was found that the code used in integrating the map and the real time data feed had a few bugs that were not all fully eliminated in the time constraint given. Hence, this stage could not be fully tested. Figure 8.11 shows an impression of how the map would appear.

The second stage of testing was incomplete at the required due date of the submission of this report, and hence cannot be discussed further.
Figure 8.11: Envisaged view of the exploration map where light blue regions represent unexplored areas, and white regions represent explored areas. The orange triangle represents the UGV position.

8.2.3 FEATURE MAP

The feature map is a map that displays the various features found in the environment. It is used to create a realistic digital representation of the environment. The main motivation behind the feature map is not only to display the features of the environment, but also to help the control system of the UGV in determining suitable way points for the UGV to reach, as discussed in Chapter 9: Control.

8.2.3.1 HOW IT WORKS

The feature map is a global two dimensional map which stores information about features in the environment surrounding the UGVs. The feature map is derived from the occupancy grid where information about occupied regions is used to determine the existence of selected features. For example if a vertical column of cells in the occupancy grid is determined to be occupied then this region in the feature map is labelled as a blocked region, which is a region that the UGVs cannot drive through or see over. Similarly if a column of the occupancy grid is occupied to a height of 60cm above ground level then this region is labelled semi-blocked, which is a region that the UGVs can see over but cannot drive over. Using similar reasoning the existence of ditches, ramps, curbs, un-navigable areas, unknown areas and open areas can be defined. The feature map stores information as 8 bit integers where each feature type is assigned an integer value between 0 and 255, hence up to 256 different features can be defined. Currently, the following features have been defined, and these are allocated the integer values given below:

- 0 represents a region that the UGV cannot see over
- 1 represents a region that is more than 60cm high
- 2 represents a region that the UGV can see over, but not drive over
- 3 represents a region that the UGV cannot drive over
• ‘6’ represents an unknown region in the environment
• ‘7’ represents a map region where the ground height changes suddenly by 10-20 cm
• ‘8’ represents a ramp. In order to determine the start and end points of the ramp, the elevation map is to be consulted
• ‘9’ represents a region of open space with unknown ground texture

The reason for defining these features is because these were the most important elements for the MAGIC competition specifications. Based on input from various sensors on board the UGV, the feature map determines the type of feature occupying a particular square in the map grid. Depending on the number assigned to a square, the HMI displays an appropriate colour on the map for that column of cells, representing the type of obstacle identified.

The feature map can be created from the occupancy grid, however the inverse is also possible where an occupancy grid can also be created from a feature map. A feature map can be written as an ASCII file with integers in the file representing different feature types, and this file can be read into the HMI. Once a feature map has been read in, it may be displayed to screen and can also be used to create an occupancy grid. The process of creating an occupancy grid from a feature map is simply the reverse of the process used to create a feature map from an occupancy grid. For example, if cell (x,y) of the feature map is characterised as blocked then column (x,y) of the occupancy grid must be completely occupied. Similarly if cell (x,y) of the feature map is semi-blocked then column (x,y) of the occupancy grid must be occupied up to 60cm above ground level and then unoccupied above this. The purpose of reading in feature maps is twofold. Firstly, the HMI map display can be tested for accuracy as a known map is available; hence it is known in advance what the HMI should display. Secondly, as an occupancy grid can be created from the feature map, elevation, configuration space, and visibility maps can be created from this occupancy grid and tested for accuracy as they will be derived from the known feature map.

### 8.2.3.2 Testing

The first stage of testing was performed by inputting a static map from a simulator in order to check the accuracy of the code. The feature map successfully passed this stage of testing, and Figure 8.12 shows a screen-shot of the image displayed by the HMI, where black represents open areas and blue represents blocked regions.

![Figure 8.12: An image from the Feature map simulations where black represents open areas and blue represents blocked regions.](image-url)
The second stage of testing was incomplete at the required due date of the submission of this report, and hence cannot be discussed further.

### 8.2.4 Vantage Map

The vantage map is designed to show areas of varying vantage on a map. Vantage essentially means 'field of view', in this case referring to a UGV’s field of view. It is desirable for the UGV to have maximum vantage at all times. The map is designed to show areas of high vantage with a lighter shade, tending to white for areas of highest vantage. The primary motivation of this map is for strategy development and navigation. The controller that directs where the UGV goes is programmed to tend to areas of high vantage, thereby decreasing the chance of the UGV colliding into obstacles.

#### 8.2.4.1 How it Works

The vantage map is created mainly through ray tracing techniques. In simple terms, rays from the LiDAR are sent out, and the occupancy of each grid square on the map is examined. The code counts how many empty squares it passes through before it hits anything (C Madden 2010, pers. comm. 14 October). There are 36 rays from the LiDAR that are analysed, each about 10 degrees apart to a range of 10m (C Madden 2010, pers. comm. 14 October). If the rays pass through empty squares without hitting any obstacle for the whole 10m range, then that area is assigned maximum vantage. As the rays hit obstacles, the vantage proportionately decreases. It should be noted that ditches do not affect the vantage map because the map searches at 50 or 60m high, and so while it hits OOI, little bumps do not affect the map (C Madden 2010, pers. comm. 14 October). The number of rays chosen, 36, is a trade off between accuracy and computational cost. Greater the number of rays chosen, more accurate the vantage map. However, with increasing number of rays, the points on the map multiply very quickly, and the computational cost can increase such that there are significant delays in generating the map. This is detrimental for the control algorithm, and hence is not preferred. On the other hand, choosing very few rays will result in a vantage map that may show areas having high vantage, when in actuality these areas may have an obstacle present due to it being in between the line of view of the rays. For the simulations, it was found that 36 rays is a reasonable trade-off between accuracy and computational cost, but this may be adjusted if actual testing proves otherwise.

#### 8.2.4.2 Testing

The first stage of testing was performed by inputting a static image of a map and testing the code for accuracy. Figure 8.13 shows a screen-shot of the image displayed by the HMI.

The second stage of testing was incomplete at the required due date of the submission of this report, and hence cannot be discussed further.
Figure 8.13: An image from the Vantage map simulations, where lighter areas tending to white represent increasing vantage; and darker areas tending to dark blue represent decreasing vantage

8.3 FUTURE WORK

Whilst the mapping system is operational and satisfies most of the objectives of the MAGIC project, it is also recognised that improvements can be made, and a few of the recommended improvements are given below.

OCCUPANCY GRID

At present the occupancy grid is a vital component in the mapping system, however the occupancy grid requires a large amount of memory for data storage and constantly updating probability information in a three dimensional grid is more computationally expensive than updating a two dimensional map. As the elevation map stores essentially the same information as the occupancy grid, future work with regards to the occupancy grid would be to replace it with the elevation map. The elevation map can store data regarding the height of objects and object shape can be determined by calculating the gradient between adjacent cells. This means that the feature map can be derived entirely from the elevation map, making the occupancy grid redundant. However, if the occupancy grid were to be superseded then filtering of noise would need to be incorporated with this process.

ELEVATION MAP

Future work regarding the elevation map would be to supersede the occupancy grid and to alter the elevation map so that it can be created from LiDAR data. Depending on the level of resolution and the elevation ranges desired the elevation map data type may have to be changed. For example, if the elevation map is an 8-bit map with a 2m elevation range then the resolution of the map will be approximately 8mm. However if the elevation map is changed to a 32-bit map then elevation values can be stored to a 1mm resolution up to a range in the order of thousands of metres. The implications of changing the data type of the elevation map would be that the HMI display of the elevation map would have to be changed, as would coded functions that reference elevation
map data as 8 bit integers, hence a cost benefit analysis should be performed before changing the elevation map data type.

**Visibility Map**

At present the visibility map is a global map which displays the collective sum of UGV visibility in a Boolean manner, meaning that if an area of the feature map is visible to a UGV the visibility value is one, and if an area is not visible to any UGVs then the visibility value is zero. Whilst this method does produce a satisfactory visibility map, it does not provide information about which areas are visible to each UGV. Hence future work for the visibility map would involve the implementation of binary masking where the visibility of UGV number N would be represented by bit N-1 of the visibility mask. For example if an area is visible to UGV number one, then the visibility value of the area would be 0x0001 which corresponds to the integer 1. Similarly an area is visible by UGV number two or by both UGVs one and two then the visibility masks would be 0x0010 and 0x0011 respectively, corresponding to the integers 2 and 3. Therefore by introducing masking into the visibility map complete information regarding UGV visibility can be obtained and more advanced control algorithms can be implemented.

**Exploration Map**

Currently, the exploration map uses a threshold of thirty hits of the LiDAR before a grid square is deemed to be explored. Since second stage of testing was not implemented, it is unclear if this number is sufficient for significant error elimination. Based on tests, this threshold may need to be modified. However, an optimal number must be chosen such that an effective compromise is made between the reliability and the speed with which the map is updated.

**Feature Map**

Future work for the feature map would be to include more features, depending on the terrain and the application in which the UGV may be used in. Since the feature map code allows for up to 255 features to be stored, this database may be expanded in the future.

**Vantage Map**

The vantage map uses 36 LiDAR rays in order to determine vantage areas. This number was chosen based on rough calculations. However, by choosing a more ‘optimum’ number, the map can be made more accurate, with less computational cost. Another area of research to be conducted in the future would be to study the effects of accuracy as the number of rays decreases. Determining a threshold before accuracy is completely deteriorated may be useful in cutting computational cost. Furthermore, based on this research, it may even be viable to generate a Voronoi map as well as a vantage map for better control (refer Chapter 9 Control), and still remaining within the computational power provided.

Overall, conceptual maps were primarily created for displaying useful information to humans. Hence, depending on the specific application that the UGVs may be used for in the future, additional conceptual maps may be developed. As far as the MAGIC project is concerned, the main future work is the testing of the three conceptual maps implemented.
The primary task of the Unmanned Ground Vehicles (UGVs) in the MAGIC project is mapping areas and identifying and neutralising Objects Of Interest (OOI). In order to accomplish this, various software systems have to work in tandem to produce useful and meaningful maps. This is the primary basis for control. Control is a critical component of the MAGIC project, and it provides commands for the UGV to execute. In simple terms, this involves moving the UGV to certain destinations, giving the UGV the means through which it can get to desired destinations, and providing instructions to perform tasks required of it. Control is significant because it integrates other software systems present in the UGV. For the purposes of the MAGIC project, control has been identified to have four main components. This chapter will discuss these four components of control, the accomplishments of the MAGIC project in the field of control and future work in this field.

9.1 FOUR COMPONENTS OF CONTROL

In order to implement control for the UGVs, four components of control have been identified. These four components are namely World Perception, Waypoint Generation, Path Generation and Path Tracking. This is illustrated in Figure 9.1.

![Figure 9.1: The four components of control](image)

9.1.1 WORLD PERCEPTION

The first component of control, world perception, is responsible for mapping the environment in which the UGV is in i.e. creating a digital representation of the environment. The main reason for having this component is that it enables the system to make informed and sensible decisions in
controlling the UGV. For instance, without world perception, the UGV is ‘driving blind’ and has no knowledge of which of the surrounding areas it will be able to travel to. In the MAGIC project, this component is essentially the mapping system of the UGV as discussed in Chapter 8: Mapping, and both the physical and conceptual maps are useful for world perception. However, of most importance to the world perception component of control is the Feature map and the Vantage map. The feature map contains information about the boundaries of the area that the UGV is in, and also contains information on walls, trees, terrain (which includes slopes and ditches) and other obstacles.

9.1.2 WAYPOINT GENERATION

One of main features of control is commanding the UGV to go to a certain location or destination. The destinations that the UGV must reach are called waypoints and the process of generating waypoints is known as waypoint generation. This component of control is closely related to World perception because the controller needs to be aware of the environment around the UGV in order to make informed decisions about where the UGV must go. It is clear that without having information from world perception, the waypoint generator would command the UGV to go to areas that are inaccessible or dangerous. This component of control was researched extensively by a researcher from the School of Computer Science but has not been implemented. A viable methodology to implement a waypoint generator was looked at and is described in the following section.

9.1.2.1 THEORY

For successful waypoint generation, two requirements need to be met. Firstly, the waypoint generated should be just out of sensor range of the UGV, thereby giving the UGV an ‘unexplored’ destination to reach, and secondly, the waypoint should be located in a region that is easily accessible to the UGV and avoid crashing into obstacles.

The first requirement is satisfied by choosing a waypoint that is just outside the LiDAR scan range. This ensures that the UGV has a reasonable range to cover in order to reach the waypoint, and hence a high level controller can ‘guide’ the UGV to go to a particular location.

The second requirement is satisfied by making use of one of two maps - the Vantage map or the Voronoi map. An example of a vantage map may be found in Chapter 8: Mapping, while Figure 9.2 shows an example of a voronoi map. The Voronoi map essentially plots lines that are equidistant from edges, and the controller can be programmed to generate waypoints along this line. The vantage map estimates areas of high and low vantage (where vantage is defined as the field of view of the UGV), and the controller generates waypoints that have a certain minimum value of vantage, in order to avoid crashing into obstacles. The Vantage map was created for the MAGIC project, and more detail on it may be found in Chapter 8, Section ??.

Depending on the application, any of these two maps may be used. One of the main deciding factors is the computational cost. Depending on the number of LiDAR rays used, the vantage map can quickly become very large and expensive to calculate. The Voronoi map is usually less expensive.

Another main deciding factor is accuracy. While the Voronoi map is reasonably accurate, it tends to draw lines towards corners as well, and the code used in the waypoint generator must account for this. Otherwise, the UGV will drive straight to a corner. The vantage map does not suffer from this fault, however it does not take into account the physical size limitations of the UGV. For instance, consider a narrow corridor without any obstacles in the way. The vantage map will indicate high
vantage, and if the waypoint generator code is not carefully implemented, the UGV will drive straight into the corridor walls. Hence, both maps have their merits and demerits, but the onus is on the waypoint generator code to make sensible decisions.

This system of waypoint generation is yet to be implemented but will be required to implement fully autonomous control of the vehicle.

9.1.3 PATH GENERATION

The third component of control, called path generation, tells the UGV how to get to a particular way point. Path generation creates a path between the UGVs current pose and the UGVs desired pose, so the controller can command the UGV to follow this path. As discussed in Chapter 2: Literature Review there are various methods that may be used to generate this path. However, given the complexity of these methods and the time constraints for the MAGIC project, it was difficult for the student responsible (who took on this aspect of the project as of August 2010) to become familiar with these methods. Hence a new algorithm was developed, which incorporated a simpler method involving cubic polynomials.

9.1.3.1 PATH REQUIREMENTS

Theoretically, there are a huge number of valid paths that can be generated between two given poses. However, not all theoretically valid paths can be followed due to the physical size limitations of the UGV, the obstacles that are present, and the limitations of the drive motors used in the UGV.
Hence, there are a few essential requirements that must be met in order for the path generated to be practical. These include:

**Continuity** - The path generated must be continuous throughout i.e. there should not be any breaks or jumps in the path. Having discontinuities in the path will result in the controller being unable to recognise the path generated, and produce errors. Furthermore, if the path generated is discontinuous, then at the points of discontinuity, the path tracking algorithm will fail, as it cannot obtain valid pose information from the generated path.

**Smoothness** - The path generated must be smooth so that the UGV can follow it easily. Having sharp bends or corners would result in difficulty for the UGV to traverse the area due to physical size and drive motor limitations. There would be a significant amount of tracking error that would be generated when the UGV encounters a sharp corner and would require a higher amount of energy use. This in turn, will cause the path tracking controller to aggressively alter the course of the UGV, leading to an accumulation of errors that will quickly become unmanageable.

**Accounting For Orientation** - While generating a path between two way points, the initial and final orientations of the UGV must be taken into consideration in order to generate a smooth path. If this is not done, then the UGV will reach the final position, and then drastically turn in order to satisfy the orientation requirement. This poses the same problems as discussed before when the UGV encounters sharp bends or cornering.

To illustrate the requirements given above, Figure 9.3 demonstrates the difference between good and bad path generation. The red dots represent way points, the black curve represents the path and the green arrows represent the direction in which the UGV is facing in. The picture on the left shows a good path, while the one on the right shows a bad path. Table 9.1 gives a brief summary comparing the two paths.

![Good Path vs Bad Path](image)

**Table 9.1: Comparison of good path and bad path example shown in Figure 9.3.**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Good Path</th>
<th>Bad Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuity</td>
<td>Satisfies</td>
<td>Satisfies</td>
</tr>
<tr>
<td>Smoothness</td>
<td>Satisfies</td>
<td>Does not satisfy</td>
</tr>
<tr>
<td>Accounting for orientation</td>
<td>Satisfies</td>
<td>Does not satisfy</td>
</tr>
</tbody>
</table>
The left picture in Figure 9.3 is an example of a good path because it is continuous and has a smooth path, that has taken initial and final orientation into account. This is evident by the absence of sharp turns and corners. This path also has a sufficiently large radius at all bends, that will easily allow the UGV to follow it. The bad path on the other hand, even though is continuous, has sharp corners at multiple locations. The initial and final orientations have also not been taken into account, which is evident by the sudden turns.

9.1.3.2 THEORY

The algorithm used for path generation was developed by generating a cubic polynomial between two given poses. This algorithm only takes into account the position and orientation of the UGV for sake of simplicity. In order to be more robust, the algorithm should consider curvature as a state variable as well, however, this significantly complicates the algorithm. Furthermore, considering position and orientation is sufficient to meet all the path requirements discussed in Section ?? . In this section, the theory behind the algorithm used will be explained.

The algorithm uses a standard Cartesian coordinate system. Figure 9.4 shows the coordinate system used for path generation.

![Coordinate system and state variables used in path generation](image)

As mentioned before, the path generated is a simple cubic polynomial which follows the form of:

\[ y(x) = ax^3 + bx^2 + cx + d \] (9.1)

where \( a, b, c, d \in \mathbb{R} \).

The algorithm solves the unknown constants \( a, b, c, d \) in order to determine the path. The values of these constants change with the initial and final poses. Prior to explaining the mathematics behind the algorithm, a few assumptions that were made are listed below.

9.1.3.3 ASSUMPTIONS

(i) The UGV is assumed to operate only on the \( x - y \) plane, and the \( z \) component is zero everywhere i.e. the algorithm assumes that the terrain that the UGV will traverse is sufficiently flat, and that all minor slopes and ditches are negligible (this was largely based on the MAGIC competition specifications).
(ii) The Ground Control Station (GCS) is taken as the origin \((0, 0)\) of the coordinate system, and
the \((x, y)\) position of the UGV is with reference to this.

(iii) The orientation, \(\theta\) is measured in the anti-clockwise direction from the positive \(x\) axis.

\subsection*{9.1.3.4 \textbf{Problem Set-Up}}

Let the initial pose of the UGV be \((x_o, y_o)\) at an orientation of \(\theta_o\).

Let the desired or final pose of the UGV be \((x_f, y_f)\) at an orientation of \(\theta_f\).

\subsection*{9.1.3.5 \textbf{Solution}}

In order to determine the four constants, four equations are required. These equations were obtained as follows:

\begin{equation}
  y_o = a(x_o)^3 + b(x_o)^2 + c(x_o) + d
\end{equation}

\begin{equation}
  y_f = a(x_f)^3 + b(x_f)^2 + c(x_f) + d
\end{equation}

Using the mathematical relation that the slope of a curve is given by the tangent of the angle between the curve and the positive \(x\)-axis, and also by the first derivative, two more equations may be obtained.

\begin{equation}
  \frac{dy}{dx} = 3ax^2 + 2bx + c
\end{equation}

\begin{equation}
  \left.\frac{dy}{dx}\right|_{(x=x_o)} = 3ax_o^2 + 2bx_o + c = \tan \theta_o
\end{equation}

\begin{equation}
  \left.\frac{dy}{dx}\right|_{(x=x_f)} = 3ax_f^2 + 2bx_f + c = \tan \theta_f
\end{equation}

Equations 9.2 through 9.5 are solved simultaneously to determine the constants \(a, b, c\) and \(d\). Once these constants have been determined, the cubic polynomial may be found. This is the path that the UGV will follow.

\subsection*{9.1.3.6 \textbf{Algorithm Example}}

An example showing how this algorithm works is given below.

Let the path polynomial be

\begin{equation}
  y(x) = ax^3 + bx^2 + cx + d
\end{equation}
Let initial pose be \((x_0, y_0) = (3, 5)\) at orientation \(\theta_o = 10^\circ = 0.175\text{rad}\).

Let final pose be \((x_f, y_f) = (10, 8)\) at orientation \(\theta_f = 60^\circ = 1.047\text{rad}\).

Using the algorithm to find a suitable path yields:

\[
y(x) = (0.02144)x^3 + (0.30716)x^2 + (1.44077)x + 2.86316
\]

The complete working of this solution is given in Appendix G.

In order to perform this in a repetitive fashion, it was desired that this algorithm be implemented in code so that the GCS can perform calculations automatically, and provide the required data to the UGV. The algorithm was implemented in MATLAB because simultaneous equation solving is relatively easy in MATLAB. This code was extensively checked for different initial and final poses, and orientations and can be seen in Appendix G. A future improvement to this system would involve implementation in C++ and consequent integration with the mapping application on the GCS. Figure 9.5 shows MATLAB’s generated output for the above example.

![Figure 9.5: Generated path for example given using MATLAB](image)

In order to provide an intuitive Graphical User Interface (GUI) to display on the HMI, MATLAB was used to create the GUI shown in Figure 9.6.
Figure 9.6: Path generation GUI created using MATLAB
9.1.4 PATH TRACKING

Path tracking is the final component of control, and it is responsible for ensuring that the UGV follows the path generated between two waypoints. A reasonably accurate path tracking algorithm can be implemented by using a simple controller. For the MAGIC2010 project, the path tracking algorithm is shown in Figure 9.7.

![Path tracking algorithm diagram](image)

The algorithm is designed to run in a loop, thereby comparing current and desired poses at regular intervals. It was decided that a fuzzy logic controller be designed to implement the path tracking algorithm. This was due to the fact that a fuzzy controller is relatively easy to design and implement when compared to traditional controllers, given the short time available for implementing path tracking.

9.1.4.1 THEORY

Fuzzy logic was used in designing the controller and this section will briefly describe the theory behind this technique. Fuzzy logic is essentially computing with words as opposed to numbers. While the design is intuitive and simple, it is still mathematically robust. A fuzzy controller takes numerical inputs and converts these to linguistic terms. Then, the controller applies fuzzy inference i.e. a set of rules that are pre-programmed into the controller, and finally defuzzifies this information (converts linguistic variables to numerical values) in order to control actuators.

Figure 9.8 shows the breakdown structure of the controller used in path tracking.

(i) In the first stage, the algorithm reads the desired UGV pose from the path generated, and compares this with the pose provided by the localisation systems. This results in two numerical variables, $\Delta x$ (difference in the x coordinate) and $\Delta y$ (difference in the y coordinate).

(ii) Next, the controller converts these variables to linguistic variables, based on a few defined fuzzy sets. This process is known as Fuzzification.
(iii) Once all the variables are in linguistic form, the controller then applies certain rules that have been pre-programmed into it. This stage is called *Fuzzy inference*. This results in linguistic command variables.

(iv) Finally, in order to operate actuators, the controller needs to convert linguistic variables to numerical variables. This process is known as *Defuzzification*.

![Diagram](image)

Figure 9.8: Fuzzy controller break down structure

### 9.1.4.2 IMPLEMENTATION

This section describes how the theory given above is implemented in the fuzzy controller designed for the MAGIC project. The fuzzy controller was designed and tested using MATLAB.

**PROBLEM SETUP**

**Input Variable:**
- $\Delta x$
- $\Delta y$

**Input Fuzzy Sets**

The fuzzy sets used for the controller here are:
- “Zero” - No difference
- “Possmall” - Small difference in the positive direction
- “Poslarge” - Large difference in the positive direction
- “Negsmall” - Small difference in the negative direction
- “Neglarge” - Large difference in the negative direction

The directions are defined according to the standard Cartesian coordinate system, as shown in Figure 9.4.
For instance, “Negsmall” means that the UGV is a small way away from the desired point, in the negative direction. Similarly, “Poslarge” means that the UGV is a large distance away from the desired point, in the positive direction, and so forth. These fuzzy sets are used for both the x and y coordinates. In the context of the x coordinate, positive is to the right and negative is to the left; while for the y coordinate positive is upwards, and negative is downwards.

**Output Variables**

The output variables are:

- VLR (Velocity Left/Right)
- VFB (Velocity Forward/Backward)

Currently, these output variables do not have any units. The EPOS driver converts these to wheel rpm in order for useful control.

**Output Fuzzy Sets**

Similar to input fuzzy sets, the output variables also have their own fuzzy sets, and these are listed below:

*For VLR:*

- “Zero” - Apply zero output
- “Rightsmall” - Small output in right direction
- “Rightlarge” - Large output in right direction
- “Leftsmall” - Small output in left direction
- “Leftlarge” - Large output in left direction

*For VFB:*

- “Zero” - Apply zero output
- “Forsmall” - Small output in forward direction
- “Forlarge” - Large output in forward direction
- “Bacsmall” - Small output in backward direction
- “Baclarge” - Large output in backward direction

**MEMBERSHIP FUNCTIONS**

**Input membership functions**

Fuzzy logic extensively uses membership functions, which indicate the extent to which a linguistic variable belongs to a certain fuzzy set, and the area of operation of each fuzzy set. MATLAB was used to define these membership functions, and Figure 9.9 shows the membership functions for both input variables.

From Figure 9.9, it can be seen that a variable can easily be part of more than one fuzzy set, and this is where the role of membership functions is crucial. For instance, consider \( \Delta x \) to be \(-60\text{cm}\), it is part of both the “Neglarge” and “Negsmall” fuzzy sets. Membership functions provide the weighting for each set. In this example, “Negsmall” will have a higher weighting than “Neglarge”.
Figure 9.9: $\Delta x$ and $\Delta y$ membership function plots defined in MATLAB
Output membership functions

Similar to the input membership function plots, the output variables also have their own corresponding membership function plots. This is shown in Figure 9.10.

Figure 9.10: Output variables (VLR and VFB) membership function plots defined in MATLAB
FUZZY INFERENCE

As stated before, fuzzy inference is the process of providing rules for the controller to apply. For instance, if the fuzzy sets for the UGV are “Neglarge” for $\Delta x$ and “Negsmall” for $\Delta y$, then the rule that the controller would apply is:

If ($\Delta x = \text{"Neglarge"}$) and ($\Delta y = \text{"Negsmall"}$) THEN (VLR = "Poslarge") AND (VFB = "Possmall")

in order to move the UGV to the desired position.

For the controller in the MAGIC project, it was decided that twenty five such rules would be sufficient to provide reasonable control for the UGV. However, writing out all twenty five of these rules would be tedious and time consuming. In order to save time and space, it is convenient to assign integer values to each of the fuzzy sets, and represent the various rules in the form of two numerical arrays. Each array is a Multiple Input Single Output (MISO) array. The two output variables defined earlier, VLR and VFB, both require two inputs ($\Delta x$ and $\Delta y$). Table 9.2 shows integer assignment for the fuzzy sets. The entire rule set may now be conveniently represented as shown in Tables ?? and ??.

Table 9.2: Assignment of integer values for fuzzy sets

<table>
<thead>
<tr>
<th>$\Delta x$ and $\Delta y$ (Inputs)</th>
<th>VLR (Output)</th>
<th>VFB (Output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Zero”</td>
<td>0</td>
<td>“Zero”</td>
</tr>
<tr>
<td>“Possmall”</td>
<td>1</td>
<td>“Rightsmall”</td>
</tr>
<tr>
<td>“Poslarge”</td>
<td>2</td>
<td>“Rightlarge”</td>
</tr>
<tr>
<td>“Negsmall”</td>
<td>-1</td>
<td>“Leftsmall”</td>
</tr>
<tr>
<td>“Neglarge”</td>
<td>-2</td>
<td>“Leftlarge”</td>
</tr>
</tbody>
</table>

Table 9.3: MISO Fuzzy inference tables

<table>
<thead>
<tr>
<th>VLR</th>
<th>$\Delta x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>-2 -1 0 1 2</td>
</tr>
<tr>
<td>-1</td>
<td>-1 -1 -1 -2</td>
</tr>
<tr>
<td>0</td>
<td>2 2 1 0 -1 -2</td>
</tr>
<tr>
<td>1</td>
<td>1 1 0 -1 -2</td>
</tr>
<tr>
<td>2</td>
<td>2 1 0 -1 -2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VFB</th>
<th>$\Delta y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>-2 2 2 2 2</td>
</tr>
<tr>
<td>-1</td>
<td>-1 1 1 1 1</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 -1 -1 -1 -1</td>
</tr>
<tr>
<td>2</td>
<td>2 -2 -2 -2 -2</td>
</tr>
</tbody>
</table>

(a) MISO Fuzzy inference table for VLR

(b) MISO Fuzzy inference for VFB

From Tables 1.3a and 1.3b, the controller is able to apply rules for any situation that may arise.

For instance, let $\Delta x = 2$ and $\Delta y = -1$. The rule applied would be:

If ($\Delta x = 2$) AND ($\Delta y = -1$) THEN (VLR = 2) AND (VFB = 1), which is in agreement with common intuition.

These rules were defined in the Rule editor in MATLAB, and Figure 9.11 shows the rule editor window.
There are two main fuzzy inference methods that are usually used in designing fuzzy controllers. These are the Mamdani and Sugeno methods. Fuzzy inference methods are used when a variable is a member of more than one fuzzy set, and these methods determine the ‘final’ value of the variable. Mamdani is a weighted average method while Sugeno is a linear combination method. For the MAGIC project, the Mamdani inference method was chosen due to its simplicity and low computational cost. The mamdani method performs a weighted average of the value of the fuzzy set, and determines how much the variable belongs to that set.

**DEFUZZICATION**

There are various methods that may be used for defuzzification of the linguistic command variables. Some of these methods include Aggregation, Centre of Mass and Centroid. For the MAGIC project, the centroid method was chosen.

**9.1.4.3 TESTING AND RESULTS**

As stated before, MATLAB was used to design the controller, and Figure 9.12 shows the entire controller.

In order to simulate controller response, a value for the two input variables, \( \Delta x \) and \( \Delta y \) was inputted into MATLAB. MATLAB then simulated the controller response, and a screenshot of this is shown in Figure 9.13.

In Figure 9.13, it can be seen that MATLAB displays all four variables and the twenty five fuzzy inference rules that were defined. When an input is given, the relevant rules apply, and MATLAB
indicates which rules have been applied by shading the corresponding membership functions. The red line in the figure shows where the value lies. At the top of the figure, the values of each of the variables are shown. In Figure 9.13, the test value inputted was $\Delta x = -30$ and $\Delta y = 10$ i.e. the UGV is 30 cm to the left and 10 cm to the top of the desired coordinate. The controller applies relevant
rules and has determined that the left/right velocity must be $V_{LR} = 0.4$ and a forward/backward velocity must be $V_{FB} = -0.115$ in order to reach the desired point from the current position. It should be noted that $V_{FB}$ and $V_{LR}$ are merely output variables, and do not represent the actual wheel velocity value. The motor driver performs transformation calculations on these variables to determine actual wheel velocities, as discussed in Section 9.3.2.

**CONTROL SURFACE**

In order to see how the controller responds to changes in values of the input variables, a *control surface* was generated. A control surface shows the full operation behaviour of the controller as the input variables change. Since the system is a two input, two output system, a 3D plot is not enough to show all four variables. Hence, two plots have been generated - each showing a particular output with the two inputs. Figure 9.14 shows the control surfaces.

![Figure 9.14: Control surfaces of the controller generated using MATLAB (Top: VFB; Bottom: VLR)](image)
Hence, the fuzzy controller has been successfully designed and tested using simulation on MATLAB. Real time implementation of the controller requires changes to the motor driver of the UGV, and this has been left to future work due to time limitations.

9.2 CONTROL IMPLEMENTATION

The control section for the MAGIC project was undertaken in three stages, in the following order:

- First stage: Manual Control
- Second stage: Semi-Autonomous Control
- Third stage: Autonomous Control

A brief explanation of the stages as well as the level of completion of each stage is given below.

9.2.1 MANUAL CONTROL

Manual control is the first stage of implementation of control because this is relatively the easiest to achieve. In manual control, the UGV is essentially remote controlled or tele-operated. Figure 9.15 illustrates the process of manual control using the concept of the four components of control looked at in Section 9.1.

As shown in Figure 9.15, all four components of control are grayed out, indicating that these are not automated. The human operator controlling UGV takes the place of the control system, and performs all four components of control. This is evidenced by the fact that when an operator is tele-operating the UGV, it is assumed that the operator can clearly see the surrounding environment, which is equivalent to the world perception component. It is also assumed that the operator makes informed decisions on choosing the destinations or waypoints that the UGV must reach, given their understanding of the environment. This is equivalent to the waypoint generation component of control. The last two components of control, namely Path Generation and Path Tracking, are irrelevant for manual control, since the operator is manually driving the UGV. There is no need for a
path to be generated, which in turn nullifies the need for path tracking. Hence, all four components of control are performed by the human operator.

Manual control was successfully implemented and tested. The UGV was tele-operated by manually issuing commands to the EPOS driver, which in turn regulated the wheel velocities accordingly. For further information on how the EPOS driver works, please refer Chapter 5: Software.

9.2.2 SEMI-AUTONOMOUS CONTROL

The second stage of implementation of control is called Semi-Autonomous Control. Figure 9.16 illustrates semi-autonomous control in the context of the four components of control.

![Figure 9.16: Semi-autonomous control in the context of the four components of control](image)

As shown in Figure 9.16, two components of control are grayed out, while the other two are active. The grayed out components indicate that these components are performed manually. For the MAGIC project, the semi-autonomous type of control involves manual world perception and manual waypoint generation, while path generation and tracking is performed autonomously. In reality, this is performed by the human operator having a reasonable idea of the environment, and selecting desired waypoints on the HMI for the UGV to reach. Once the desired waypoint has been entered, then the path generation and path tracking algorithms are activated, and the UGV performs these autonomously.

The second stage of implementation of control i.e. semi-autonomous control succeeds manual control because the successful completion of the manual control confirmed that the UGV drive assembly, motors and motor controllers along with various other hardware components were working normally, and as desired. Semi-autonomous control then tests other systems of the UGV such as localisation, path generation and path tracking algorithms. If semi-autonomous control was implemented prior to manual control, then troubleshooting would be significantly more difficult as the fault could be anywhere in the hardware or software. This difficulty is eliminated by first checking manual control, which significantly narrows the areas where a fault can occur.

Both components of control required for semi-Autonomous control (Path Generation and Path Tracking) were implemented, but not integrated to the UGV. These two components of control were
designed and tested in MATLAB, but integrating them to communicating with the UGV has not yet been achieved, and this is left for future work.

### 9.2.3 Autonomous Control

The final stage of implementation of control is known as *Autonomous Control*. Figure 9.17 illustrates autonomous control within the context of the four components of control.

![Autonomous control within the context of the four components of control](image)

As shown in Figure 9.17, all four components of control are active, indicating that the entire control system is fully autonomous, and there is no human involvement. The UGV obtains information about its environment through world perception, which is performed via mapping. In the MAGIC project, this information is obtained not only by the UGV concerned, but also by other UGVs as well since the maps are updated by the data from all UGVs by a process running on the GCS. Based on this information and strategy control, the waypoint generator defines suitable waypoints for the UGV to reach. This is the most complex aspect of the autonomous control, especially considering the inclusion of multiple UGVs all requiring waypoints to be generated that enable them to collaboratively cover their environment. Finally, the path generation and path tracking components become active, and function as discussed previously.

As described earlier in this section, a successful waypoint generator was not designed in time for the project, and hence full autonomous control has not been achieved.

### 9.3 Implementation

The three types of control were implemented in software, as shown in Figure 9.18. The commands resulting from any control algorithm, feed into the EPOS driver, which controls the drive and pan and tilt motors.

When invoked, the EPOS driver displays two options that the operator can choose, and these are Manual and Semi-Autonomous, as shown in Figure 9.19. The operator chooses the desired option by pressing the key corresponding to the option. For instance, for manual control, the operator would
press ‘M’. It should be noted that because fully autonomous control has not been implemented, this option is not displayed currently.

### 9.3.1 Manual Control

Manual control can be used to command both the drive motors as well as the pan and tilt motors. The human operator uses a keyboard to input desired velocity increments or decrements. The driver is structured in such a way that it treats the three wheel motors on the left side of the UGV as one set, and the three motors on the right side of the UGV as another set. Henceforth, these will be referred to as 'Left side velocity' and 'Right side velocity'. When the operator presses ‘M’ at the startup screen, the manual option is chosen, and the program displays a screen containing a list of different buttons that may be used to accomplish different tasks. Figure 9.20 shows a screenshot of this.

As shown in Figure 9.20, in order to move the UGV forward, the operator presses the key 'W', and similarly to move the UGV backward, the operator presses the key 'S'. The left and right keys work in the same way. It should be noted that the program functions in an incremental fashion. For instance, pressing the forward key once moves the UGV forward, however repeatedly pressing the forward key increases the speed of the UGV proportionately. This applies to all commands. The increments associated with each key is as follows:
• Forward, ‘W’ key - Increments both ‘Left side velocity’ and ‘Right side velocity’ by 100rpm
• Backward, ‘S’ key - Decrements both ‘Left side velocity’ and ‘Right side velocity’ by 100rpm
• Left, ‘A’ key - Increments ‘Right side velocity’ by a 100rpm and decrements ‘Left side velocity’ by 100rpm
• Right, ‘D’ key - Decrements ‘Right side velocity’ by a 100rpm and increments ‘Left side velocity’ by 100rpm
• Tilt up, ‘I’ key - Tilts the Pan Tilt Unit (PTU) up by 5°
• Tilt down, ‘K’ key - Tilts the PTU down by 5°
• Pan left, ‘J’ key - Pans the PTU left by 5°
• Pan right, ‘L’ key - Pans the PTU right by 5°

It is a requirement that the operator be aware of the number of increments given, in order to counter the effect when needed. For instance, if the operator hit the ‘A’ key (for turning left) three times, then to stop the UGV from turning left and go straight, the operator has the press the ‘D’ (for turning right) three times. The same applies for forward and backward motion as well. However, it was realised that given the maximum speed that the UGV can reach, there must be provision to stop the UGV immediately when the need arises. This is done by the operator pressing either the ‘0’ or the ‘O’ key. The UGV comes to a sudden and complete halt. This however, is not preferred as it quickly wears out the motors and gearbox assemblies, but it is there as a failsafe.

The limits for the pan tilt motors are as follows:
• Pan motor: 90° to the left and 90° to the right
• Tilt motor: 20° to the top and 35° to the bottom
9.3.2 SEMI-AUTONOMOUS CONTROL

When the semi-autonomous control option is selected, the program awaits for an input from the path tracking fuzzy controller. When an input from the controller is received, a transformation calculation is performed in order to convert the controller output value to a velocity increment value that can be fed into the wheel motors. The transformation calculation for converting between angular and linear velocities is given below:

\[
\text{Velocity (m/s)} = \text{Motor speed (rpm)} \times \text{gearbox ratio} \times 2\pi \times \frac{\text{Wheel radius}}{60}
\]

where Gearbox ratio could equal \(1/18\), for example. The complete transformation from the fuzzy controller to the EPOS driver has been researched into a bit, but not extensively. This has been left for future work.

It should be noted that the fuzzy controller only controls the drive motors, and hence the pan and tilt motors need to be controlled separately, possibly by the human operator. This is evidenced by the fact that the PTU motor is a separate box in Figure 9.18.

9.3.3 AUTONOMOUS CONTROL

At the time the report was written, this section of control could not be implemented due to time constraints. Hence, this section has been left for future work.

9.4 FUTURE WORK

Future work for this section includes:

- Successful implementation of Semi-Autonomous control by integrating the path generation and path tracking components of control with the UGV via the EPOS driver
- Implementation of the autonomous section of control
- Presently, for turning the UGV, all the wheels on the left side of the UGV have the same velocity and all the wheels on the right side of the UGV have the same velocity. This however, is not desirable as it means that the UGV cannot turn on the spot. In order to correct this, the outer wheels ought to have a higher velocity than the centre wheels.
- The path generation algorithm assumes that the UGV is traversing a relatively smooth plane (ignoring small bumps and ditches), and the z-component has been assumed to be zero everywhere. This however is not very robust, especially if the bumps or ditches are significant. Hence, a better algorithm that accounts for the z-component as well ought to be developed.
According to the MAGIC 2010 competition rules, the team of UGVs must be able to “detect, locate, classify, recognise, track and neutralise a number of static and mobile objects of interest (OOI)” (MAGIC 2010 Guidelines 2009). The three types of objects in the project were red bins of approximately half a metre height, and humans dressed in navy blue or red coveralls that walk around, all shown in Figure 10.1. The colour of the objects signified the ‘friendliness’ of the objects, where red represented dangerous objects, and navy blue represented civilians. The form of the objects signified mobility, where bins were stationary and humans dressed in coveralls were able to walk around at no more than 5 km/h. As discussed in the Literature Review (Section 2.6), the most logical technique to accomplish this is camera vision coordinated with the other UGV sensors for position information. A detailed outline of the program flow for the computer vision part of the project is displayed in Figure 10.2. Since the contract change, the objects are no longer required to be tracked in a video feed for the judges and do not require neutralising with the laser. The camera driver and object finder programs work together in acquiring the latest images and corresponding LiDAR data for analysis and object locating on the GCS. Once the camera driver has saved the images and the corresponding LiDAR data in a text file at the same time the image is captured to a known location on the UGV hard disk, the object finder requests the images and LiDAR data as required for analysis and subsequent displaying on the GCS. Since the images are named in the format “image_####.jpg” where the hashes represent the image number taken in the recording session, the latest image may be found by sorting the directory of the images on the UGV. Using the image number of the latest image found, the text file containing the corresponding LiDAR data may also be found. The latest text files and images are sent across the network whenever the object finder program is ready to analyse another image to detect, identify and locate the objects seen in the image.

Figure 10.1: The MAGIC competition objects of interest.
Figure 10.2: Flowchart of the camera hardware and software interaction.
10.1 OBJECTS OF INTEREST FOR COMPUTER VISION

For this project, object recognition and tracking using a digital camera consists of three steps, namely, detection, identification and location. For object detection, the most distinguishing feature of the MAGIC competition target objects is colour, as shown in Figure 10.1. Once the camera images are gathered by the camera driver, and sent back to the Ground Control Station (GCS), the camera images are scanned for the desired colours and the pixels detected that are the desired colours of navy blue or red are grouped into large continuous “blobs”. For object identification, the objects are then distinguished from other similar-coloured objects by specific characteristics, such as relative dimensions and a second colour analysis, which confirms the correct colours. Finally, the location of the object in each image is translated to a map location by knowing the the angle to the object from the camera perspective, and extracting the distance in the LiDAR scan data corresponding to that same angle to find the distance between the UGV and the object. These tasks are completed in the order shown on the flowchart of the computer vision program in Figure 10.3.

Figure 10.3: Flowchart of the computer vision program flow.
10.2 OBJECT DETECTION

Object detection is achieved by scanning every pixel of the image to see if the pixels pass certain thresholds that correspond to the desired navy blue and red colours. Clumps of the pixels that pass the thresholds are grouped together if close enough to represent the one object via a process called morphology.

10.2.1 COLOUR THRESHOLDING

Colour thresholding is a technique used to extract regions of desired colour with set boundaries of detection called thresholds. As discussed in the Literature Review (Section 2.6) and the Camera Driver discussion (Section 2?), the three contending colour spaces that could be used for the object finding code were Red Green Blue (RGB), YUV (a colour space consisting of one luma (Y) and two chrominance (UV) components) and Hue Saturation Value (HSV). The Baumer camera outputs images in either RGB or YUV, and a conversion calculation may be used to produce the image in HSV. The advantage of the HSV and YUV colour spaces is that the same colours can be more easily recognised under light intensity changes which is more of a problem with RGB (C Madden 2010, pers. comm. 22 April). However, calculations with the HSV colour space are more complicated in terms of programming since they use polar coordinates, requiring greater time for programming. Most importantly however, RGB was chosen due to the computer vision supervisor having much more experience with RGB (C Madden 2010, pers. comm. 22 April).

10.2.1.1 THEORY

There is a range of colours that would be considered by humans to be red or a close derivative of red. This is illustrated in Figure 10.4 below. A human would label the bin as a red bin, since humans recognise that the sections of the bin lid near white in colour are reflections, and the section of the bin that are a very dark red are in shadow. To a computer however, these colours are seen as very different colours in the RGB colour model because the red, green and blue channel numbers are so different, as illustrated by the examples given. Additionally, if two images of exactly the same scene were captured one after the other, even more variation is found as the colour channel contributions vary by small amounts due to electronic noise.

In the RGB colour space, thresholding helps to overcome these variations, and extract regions of desired colour, accounting for shadows and reflections. The most effective method in testing various kinds of colour thresholds was found to be ratios between different colour channels. This significantly reduced the problem of light intensity changes affecting the usefulness of the RGB colour space. In Figure 10.4, various relationships between the red, green and blue channels may be observed. First, the red channel is always greater than the green and blue channels regardless of the lighting intensity of the region. Also, the greater the ratio between red and the other two channels, the more likely it is to be red. Thus, the thresholds to be fulfilled by pixels that are likely to be part of the red desired objects are ratios of \( \frac{\text{red}}{\text{green}} > 1.5, \frac{\text{red}}{\text{blue}} > 1.4 \) and a red threshold value of \( \text{red} > 40 \). Note that not all samples in Figure 10.4 fulfil these thresholds, but a compromise has to be made at some point in order to avoid collecting many other pixels (such as browns and oranges that have similar ratios between channels). This point was chosen by testing the threshold values on different image samples that included indoors under light, indoors under dim light, outdoors in the sun and outdoors in shadow. Since the colour of the red bin is indistinguishable from the red human object colour due to changing lighting conditions, both red types of objects are processed with the same threshold. For the majority of cases, the blue channel is slightly larger than the green channel, the difference mainly due to the surrounding environment. For example
the blue channel is larger if the surface is reflecting a clear blue sky, but the green channel will be larger if reflecting nearby grass. Similar types of thresholds are used for the navy blue colour. These thresholds are then applied to each incoming image from the camera driver by methodically comparing every pixel against the thresholds. The pixels that satisfy the thresholds are made white, and the remainder black, thus creating a binary image as shown in Figure 10.5. Since the process of thresholding is essentially a quantisation process, i.e. a colour photograph with a large amount of information is reduced to just black and white in a binary image, the red and blue patches are indistinguishable in the resultant binary image. Thus, the red and blue objects are detected and information extracted in two different thresholding passes on the image.
10.2.2 IMAGE MORPHOLOGY AND CONTOURS

Morphological transformations on an image are used to remove noise, separate components, find image holes or regions of unusual intensity and merge disparate parts of common characteristics together (Bradski & Kaehler 2008, p. 119). The result of the thresholding operation in Figure 10.5 was quite “noisy”, as seen by the large number of small black spots representing pixels that did not pass the thresholds. To remove the noise and obtain a clearer idea of how many objects are really being observed, morphological transformations are used to reduce the noise. The two basic morphological transformations are dilation and erosion. To dilate or erode an image, a kernel must be defined. A kernel is a group of pixels in some shape, usually a square or circle, with a defined anchor point. For dilation, the function of the kernel is a local maximum operator. As the anchor point of the kernel passes across each pixel in an image, the maximum pixel value in the kernel is substituted as the existing pixel value of the anchor point. Thus, dilation causes lighter regions of the image to expand. Erosion has a similar operation, except the kernel function is a minimum operator, causing darker regions to grow (Bradski & Kaehler 2008, p. 119). A visualisation of both of these transformations are given in Figure 10.6.

In this project, the morphological transformations are operated on a binary image. Dilation expands the white areas, and erosion expands the black areas. The order in which these operations are applied is important. For the project purposes, the closing operation is used, since it reduces undesirable holes or noisy segments in the object. An example of a morphological closing operation is shown in Figure 10.7. First the white area is expanded by dilation in the left frame, eliminating any holes and noise in the object, produced by shadows or reflections outside of the threshold boundaries. The pixels that are changed to white in the dilation operation are coloured grey in the middle frame. An erosion operation is made, reducing the object back to the original size, minus the noisy pixels. The kernel used in the closing operation is represented by the 3x3 red square with yellow anchor point.

10.2.2.1 TESTING AND RESULTS

Testing of the computer vision code revealed that reflections and shadows are especially prominent on sunny days, and created larger holes and noise in the objects than on overcast days. Thus a suitably large kernel was needed to overcome the noisy areas. The kernel used is a 5x5 pixel elliptical kernel with central anchor point, and the result is shown in Figure 10.8 for the previously shown red bin video frame sample. Note that in the image processing program, the morphology
closing operation is done before the contour finding operation to reduce the number of contours to be found, thus saving processing time and resources.

Using the boundaries created by the black and white pixels, an algorithm is run on the image that creates a list of the contours that separate the black and white regions. As depicted in Figure 10.8, the contour sizes vary in size from the perimeter of the large area of most of the bin to the small perimeters of shadows or reflections. The small contours often represent noise, as small areas of the targeted range of colour can appear and disappear from video frame to frame. To remove this noise before any further image processing, only contours of perimeter greater than 30 pixels are identified and listed in a contour database. Although the longest around the whole bin shown in Figure 10.8 is in the hundreds of pixels, this is a result of the close proximity of the bin to the camera. Care has to be taken that the minimum contour length for an object detection is not increased too much, or objects of interest at further distances from the camera may not be detected.

10.2.3 OTHER OBJECT DETECTION TECHNIQUES CONSIDERED

There were a number of computer vision techniques proposed during the year that were not used due to a number of reasons:

- **Face detection:** If a face were detected above a blue or a red object, then it would most definitely be a desired human object. Unfortunately, face recognition was deemed to be too computationally expensive (Madden, C 2010, pers. comm., 14 September).
• **Shape detection:** If the outline of objects were able to be detected, then it would be a matter of matching the known perimeter shapes of the bins and humans with the outlines detected in the image. As seen in Section 10.2.2, the contour around the edge of the detected region of desired colour is very prone to lighting noise such as reflections and shadows. For humans, there are less lighting noise problems, but since humans are mobile and of varying sizes, the shapes required to match vary considerably.

• **Thermal or stereo cameras:** A thermal camera would make the detection of the humans considerably easier and the shape matching idea would work far better in this scenario (Madden, C 2010, pers. comm., 14 September). Stereo cameras would also help to show the desired objects are disconnected from objects in the background with similar colours, because stereo cameras have an additional dimension of depth. Using stereo camera, a similar coloured background would be seen to be behind the detected object and not part of the object, and detected blobs of colour could be grouped according to distance from the cameras, reducing the chance of identifying a red brick wall as part of a red bin for example. Thermal and stereo cameras were deemed to be prohibitively expensive, especially since the partner Strategic Engineering desired images of high resolution. In hindsight, since low resolution images are capable of identifying the required objects, stereo vision cameras were probably a viable option.

• **Other common segmentation techniques:** Segmentation is the basis of most image processing, and is a viable option for this project with a few restrictions. The more simple segmentation used in this project utilises fixed thresholds rather than the many different types of segmentation via mode-seeking such as the K-means algorithm, Mixture of Gaussians and mean shift methods commonly used that employ an adjustable threshold. The decision to use a fixed set of thresholds was primarily because of speed, as adjusting the adaptable threshold can be computationally intensive and difficult, especially since the desired objects are only occasionally observed. To segment single images, the usual parametric and non-parametric methods mentioned above require several iterations to achieve correct clustering of colour patches, the computation of which cannot be afforded on the base station. Faster surveillance methods using background subtraction could have been used to make detecting moving humans much easier as the pixels for moving objects would change from image to image. However, the project partner Strategic Engineering made a “no stopping of the UGVs” rule that prevented this method from being employed from the start of competition planning, and so the method was not considered from that point onwards. The final choice of thresholding with fixed values proved to be an efficient means of object detection computation for the highly mobile UGVs.

### 10.3 OBJECT IDENTIFICATION

Computers do not have the many past associations with objects as humans have learned from a young age, making object recognition for computers a difficult task. Since the competition objects of interest (OOI) are few and distinctive, the process of object identification is largely reduced in complexity by recognising the main features of the required OOI. The three types of OOI to be located and classified, as previously outlined, are static red bins, and humans walking around dressed in either red or navy blue coveralls. The three main features that are used to distinguish the three types of objects from each other are dimensions, colour and movement. Dimensions and movement are used in parallel to distinguish between the bins and humans, while colour is used to differentiate between the red and blue objects. So far in this discussion colour has been used to detect any of the desired objects, but at this stage no distinction has been made between the two types of red objects.
10.3.1 IDENTIFICATION BY DIMENSIONS

The colours of the red bins can be very difficult to distinguish from the red human colour due to noise and lighting discussed above. Thus the next most obvious way to differentiate between the bins and humans is physical size. The dimensions of the located object may be found through the use of “bounding rectangles”. In the OpenCV programming language, a rectangle can be constructed that bounds the greatest vertical and horizontal dimensions of a contour. As discussed earlier, the contours in this project are the sets of points that define the boundaries between the black and white regions in the morphology binary image. Examples of bounding rectangles are shown in Figure 10.9, where the rectangles bound the contour of greatest dimensions. However, note that the dimensions of the objects are highly dependent on distance. A human would be seen to be the same height as a bin if the human was far enough away from the camera, but the widths would distinguish between them. Bins and humans may then be distinguished by the different ratios of the height divided by the width. It was found the ranges of dimensions ratio values for bins and humans was larger than expected due to lighting induced noise, different human statures and the perspective of the humans (when humans are observed from the side, the ratio was smaller when humans were mid-stride when walking, and larger if they were stationary). The range of values corresponding to object types used in the object finder program is 0.8 to 1.5 for bins and 1.6 to 4.6 for humans. Note that if a human crouches, the object recognition program will not work with the dimension ratio, as a red crouched human would appear to the program as a red bin, and the blue crouched human would not satisfy the dimension ratio at all. This problem was overlooked because the MAGIC rules specified that the objects of interest would not change their behaviour (i.e. the humans would always walk about upright). Dimensions will distinguish between the bins and humans, but other features are necessary to distinguish between the navy and red mobile OOI, particularly colour.

\[
\text{Dimensions ratio of bounding box} = \frac{\text{Height}}{\text{Width}}
\]

Figure 10.9: OpenCV output of a purple bounding rectangle around the greatest dimensions of the group of contours on the bin.

10.3.2 IDENTIFICATION CONFIRMATION BY COLOUR

To distinguish between the red and blue objects, colour is the most obvious choice. It has already been shown that the object locations are detected by the methods of thresholding and morphology. However, although humans might be able to distinguish the objects from their surrounding
environments easily, similar colours that fulfil the thresholds are often found in urban environments that the UGVs are supposed to work in. For example, red brick walls under certain lighting conditions satisfied the thresholds for the red objects, and dark shadows, especially on overcast days satisfied the thresholds used for the navy blue objects. These regions of similar colour can be identified falsely as one of the objects of interest, or if they are close to one of the objects, expand the object size as seen by the binary image to be outside the expected range of the dimensions ratio for the actual object. Thus, more exacting colour comparisons were required after the initial thresholding operation to ensure the region identified as a possible object did not give a false positive for areas of the image with very similar colours. This is achieved by analysing the colours in a sample region of interest of the object. The region of interest is a rectangle scaled to the dimensions of the bounding box that defines the dimensions of the largest contour of the detected object, shown in Figure 10.10. A scaled region of interest is used (rather than a fixed size) to achieve the best representation of the object’s colour regardless of the distance the camera is from the object. The means of the red, green and blue channels are calculated and compared with a desired threshold. Tests on a range of examples in varying lighting conditions were done in MATLAB to find the most likely values for the three object means and their order in value, as seen in Appendix ?? . The results of the tests enabled better initial thresholds to be made as well as determining more exact requirements the suspected object regions had to fulfil. For example, a region of interest sampled from a suspected desired red object section needs to have the calculated red mean above a minimum channel value of 100 and the red mean should also be greater than both the blue and green means. The blue mean is not necessarily always above the green mean as in the example in Figure 10.10, but depends mainly on the colour of the ambient lighting, and the colours of the surrounding objects that reflect light onto the object’s surface. The size of the sampling region was also found to have a significant effect on the accuracy of the object identification. If the sampling region is too large, the region has more chance of including noise or objects that will distort the measurements of the red, green and blue channel means. For example, for a bin, inclusion of the reflections on the bins surface in the sampling region can skew the means to be much larger than the actual object colour. For humans, if the sampling rectangle is too low down, the sampling region may include the gap between their legs which also distorts the measured means. A test was run comparing different sampling rectangle sizes and positions, and by reducing the original chosen size by 50% and moving the sampling region up relative to the object bounding box, the object identification accuracy was increased from 42% to 87%.

10.3.3 IDENTIFICATION CONFIRMATION BY CONTINUITY

Another way to reduce the occurrence of the detection of false objects (i.e. clumps of pixels that satisfy all of the detection conditions but are not objects of interest) is by continuity checks. This is achieved by using the information about the detected objects in a database within the object finder program. As new objects are detected, they are added to the database of objects. The main continuity check relies on the locations of the objects to be known. To calculate the object locations in map coordinates, the camera must first be calibrated which is discussed in the following section. Once the location of the detected object is known it may be compared with other objects of the same type to determine whether it is a new object, or if it is an already detected object that has changed location slightly. The threshold used to determine if it is the same object is currently a 2 metre radius from the last sighted location of the object. Of course, assuming the object is in view all of the time, the accuracy of this technique is greatly dependent on the update frequency of the images to the object finder program, so the radius threshold may be increased or decreased manually depending on the framerate of the images displayed on the GCS to account for object movement more appropriately.
Figure 10.10: Scaled to the dimensions of the bounding box (green rectangle), a region of interest (yellow rectangle) is analysed for the mean channel values as well as the order of the means as represented in the histogram of RGB values to the right.

10.4 OBJECT LOCATION

The image returned by the camera contains 2-dimensional information about the scene it is viewing. In order to give objects spotted a physical location, the image information must be combined with another sensor, in this case being the Light Detection and Ranging (LiDAR) sensor, to give a third dimension of distance to what the camera is viewing. Camera calibration is required to accurately relate the location of detected objects in the images to the LiDAR distance scans in order to give each detected object a physical location on the maps displayed on the GCS.

10.4.1 CAMERA CALIBRATION

To understand how camera calibration works, the basic principles of cameras and lenses need to be understood. The simplest form of camera is the pinhole camera model, which is often used to explain the geometric aspects of vision. The pinhole camera model is an imaginary wall that blocks all light rays except those that pass through a tiny aperture or pinhole in the wall. Unfortunately, a pinhole is a poor way to make images because of the long exposure time required to form an image of sufficient brightness. Human eyes and cameras use lenses to gain a more rapid exposure time. Nevertheless, lenses not only introduce more complex geometry but distortion to the problem. Distortion in lenses due to their manufacture is countered by camera calibration. Calibration of the image produced by the camera is particularly important for this project, since angles and positions of objects are estimated from the image data. The two main camera lens distortions are radial and tangential. Radial distortions result from the shape of the lens, and are revealed by curved line in the periphery of the image. An example is shown in Figure 10.11 by the room-divider lines at the top of the distorted image. Tangential distortions arise from manufacturing assembly, when the lens is not placed exactly parallel to the imaging plane of the pixel die. A schematic in Figure 10.12 depicts how tangential distortion can occur.
The camera being used in the project underwent a calibration process using a checkerboard pattern on a flat surface and a tape measure. A checkerboard pattern is often used in camera calibration since it is a distinctive planar object with easily detectable lines and corners. As expected, the lens has a relatively wide field of view but is a quality-made camera, so the predominant distortion was radial with hardly observable tangential distortion. For the purposes of object locating calculations, the resultant image outputted by the camera has the property of 20 pixels per angle degree. The object finder code has been programmed such that for the images that are scaled down, the angle is also appropriately scaled. Since many different sizes of images were tested, the scale factor was programmed to be calculated automatically. Upon detecting an object, using the pixel number to angle relationship found by camera calibration, the angle to the object from the camera’s perspective is calculated. This angle may then be used to find the distance to the object for the closest corresponding angle in the latest LiDAR reading. Since the position of the sensors on the UGV is known relative to the UGV chassis itself, and the location of the UGV is known from localisation techniques, the object may be given map coordinates via a number of matrix transformations relating the UGV position in real world coordinates to map coordinates.

![Camera image before undistortion (left) and after undistortion (right)](image)

Figure 10.11: Camera image before undistortion (left) and after undistortion (right) (Bradski & Kaehler 2008, p. 396)

### 10.4.2 **Final Testing and Results**

The final computer vision program has reasonable accuracy in identifying each of the desired object, both inside and out, with the exception of the navy blue objects outside. The latest object finder program was run on some past test image sequences captured through the year for the relevant objects, with the results shown in Table 10.1. For most objects, plenty of indoor data was taken, because the majority of testing was done during the winter, when it was raining or too dark to be of practical use much of the time when it was desired to do testing. For the outside tests, other factors often contributed to poor numbers of useful images captured, such as the failure for the automatic gain to work properly, meaning the images would have a ‘washed-out’ appearance.

The main factors that limit the accuracy of the object recognition program are image blur, object occlusion and lighting effects. Image blur occurs when the UGV is in motion, causing the desired objects to have a “fuzzy” appearance. The problem is mainly prevalent in low lighting conditions, such as outside at night. The automatic gain of the camera increases the camera exposure in order to obtain an image with reasonable lighting levels for the computer vision program (and for the GCS operators’s view). Subsequently, objects at night are seen to take up larger portions of the images, and distort the colour and bounding box dimensions ratio readings, failing to register as an object.
Figure 10.12: Tangential distortion occurs because the pixel die is not exactly parallel to the lens plane. For a cheap camera, this occurs because the pixel die is roughly glued to the back of the camera (Bradski & Kaehler 2008, p. 377).

Table 10.1: Accuracy results for each type of object, both inside and outside, from running the object finder program on several image sequences captured through the year.

<table>
<thead>
<tr>
<th>Test Details</th>
<th>Number of relevant images in test sequence</th>
<th>Percentage of specified target objects identified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red bin - inside</td>
<td>23</td>
<td>96</td>
</tr>
<tr>
<td>Red bin - outside</td>
<td>44</td>
<td>76</td>
</tr>
<tr>
<td>Red human - inside</td>
<td>40</td>
<td>95</td>
</tr>
<tr>
<td>Red human - outside</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Blue human - inside with poor lighting</td>
<td>69</td>
<td>81</td>
</tr>
<tr>
<td>Blue human - inside with good lighting</td>
<td>57</td>
<td>87</td>
</tr>
<tr>
<td>Blue human - outside on overcast day</td>
<td>12</td>
<td>None</td>
</tr>
</tbody>
</table>

Object recognition accuracy is also affected by occlusion, where an object in the environment might partially obscure the camera’s view of the object. Generally the object blocking the desired object will cause the dimensions ratio to miss the desired range for that particular object, or may obstruct the sampling region in the middle of the object causing false colour values to be read. Finally, lighting effects also have a significant effect on the accuracy of the object recognition program. For the blue human objects this is especially prevalent, and a reliable method to overcome shadow detection has not yet been found for outdoor scenarios. As shown in Figure 10.13, the blue hue in many shadows and on many dark or grey objects in an urban environment caused by the reflection of the sky satisfy the target ratios for the navy blue objects. Even indoors, the lighting can affect the object identification accuracy when the camera view turns into or away from a light source such...
as a window or ceiling light. The camera driver usually takes a few frames to adjust the gain and exposure to suitable levels of brightness and contrast for computer analysis and human viewing, and in the period of adjustment the images may have a darkened or “washed out” appearance which may cause the colour values of the desired objects to fail the criteria for object detection.

A typical analysis of a indoors room reveals the scenario in Figure 10.14, where it can be seen that the tasks of detection and identification by the computer vision techniques are by no means as simple as a human finds it to be. The various coloured contours in the picture represent the types of objects that are identified, where in this case blue is a success, but pink and black are failures. The red object does not have a bounding box around it because the image is displaying the result of a blue thresholding and morphology pass. The blue rectangle represents a successful identification of a navy blue human, where the small white square just above the centre of the blue rectangle is the sampling region for the second colour check. The pink bounding boxes satisfy the colour thresholds, since the dark coloured floor contains a blue hue from the light coming through a window to the right of the image. However, the pink boxes do not satisfy the bounding box dimensions ratio criteria for one of the three desired objects. The black bounding box satisfies the blue colour thresholds, and fits the dimensions ratio for a blue bin, but since a blue bin is not one of the three desired objects, fails to recognise any object. Other colours not shown in this example are used in testing to represent successful detection of red humans, red bins and the failure to pass the second colour test but with the correct dimensions for a desired object. Some incorrect blobs may be removed by using the LiDAR data to compare the expected distance with the dimensions of the bounding boxes, but this image anlaysis was made using colour features only, so all noise is displayed. The accuracy of the object location techniques have not been compared numerically with actual measurements, but from a knowledge of the testing areas, the locating of the objects by corresponding the LiDAR data with the object location in the image seems to be accurate to plus or minus two metres.

10.5 FUTURE WORK

Computer vision is a very useful tool for robots in enhancing awareness of their surroundings. Some of the features that it could provide for the robot system in the future are:
• **Detection and identification of other objects:** The three objects detected, identified and located in this project were only chosen because of competition specifications. Given time, many other objects could be added to the recognition library using other computer vision techniques mentioned in Section 10.2.3 for more complex objects than those given. The reliability of the current computer vision program in recognising the objects and minimising noise could also be boosted by the use of these techniques.

• **Ground texture recognition:** Using pattern recognition techniques, terrain texture could be determined (such as mud, grass, rocks or concrete). The camera program could then pass this information to the mapping system, and the waypoint planning could be modified to avoid potentially hazardous areas for the robot in autonomous mode, such as mud which could bog the UGVs or rocky ground that the UGVs would struggle to traverse.

• **Edge detection:** Using feature recognition and edge detection techniques mentioned in the Literature Review, the edges and corners of walls, buildings, doors and ramps could be identified. These techniques have been used in “visual SLAM” (Simultaneous Localisation and Mapping). Combining the edge and feature detection with the indoor localisation techniques would make the current form of SLAM more robust. Even if only a simple form of edge detection was used, the information could be used to pass through doorways more reliably.

• **Object tracking:** Detected objects were required to be tracked in the original MAGIC competition rules, but the concept was much less of a priority after failing to reach the final round of the competition. It was planned to link the pan motor on the pan-tilt unit to rotate the camera and keep a detected object in view. This could be achieved by a simple Kalman filter to predict the movement of an object, and track the object even when it is temporarily blocked from view by another object.
Part III

CONCLUSION
CONCLUSION

The aim of the MAGIC 2010 project was to research, design, construct and test a team of Unmanned Ground Vehicles (UGVs) that was capable of completely exploring and accurately mapping a given area. The UGV team was also required to correctly locate, classify and recognise all defined Objects Of Interest (OOI) within the competition area and undertake appropriate responses. All of this was desired to be conducted in a short period of time and with minimal or no human interaction. This required a stable and robust UGV platform with appropriate sensors to map the physical environment and actuators to mobilise the platform. It also required a system of centralised control, which was able to send commands to multiple UGVs so that the team could collaboratively and effectively achieve the goals of the mission. The project also had to be completed within budget, including all equipment purchase and labour hire.

These objectives have been achieved with the successful design and build of two UGVs and a ground control station (GCS). The two UGVs, in cooperation with the software and hardware capabilities of the GCS have been able to achieve localisation, mapping (both physical and conceptual), as well as machine vision to locate and identify OOIs, and there has been success in implementing the first two stages of the control system. A significant number of the contract goals have been fully achieved and work still continues to complete the last of these as well as the extended goals.

A thorough review of literature relevant to the MAGIC 2010 project has been instrumental in motivating many of the solutions implemented to achieve the goals. The literature considered various relevant concepts needed to successfully complete this project including methods of localisation, such as Kalman filters, and Simultaneous Localisation and Mapping (SLAM); methods used for mapping, including creating maps such as the exploration map, vantage map, elevation map and occupancy grid; methods used for control including manual control as well as the various components of the control system such as world perception, waypoint generation, path generation and path tracking; and also methods for machine vision, including analysis of colour and techniques of adjusting camera feed. Previous attempts at projects like these have had varied successes, but the goals of this project differed, because it was desired to have the UGVs working collaboratively in a variety of environments. This is something many other implementations have not achieved in the past and something which this project has achieved some success in. Although not all goals were fully met, a reasonable foundation for significant future work in these areas has been built.

The approach to tackling this project involved working on developing the complete system architecture for both hardware and software in order to eliminate the need to modify completed parts to make way for others. The system architecture was developed as a series of detailed flowcharts displaying the inter-connections and hierarchy of the various subsystems involved.

The development began in cooperation with project partner, Strategic Engineering and consisted of developing the mechanical design of the UGVs, the selection of hardware and the development of sensor drivers. This was eventually completed after several iterations to increase the internal accessibility, decrease weight and improve the torque transfer of the UGV design.

For outdoor environments, where Global Positioning System (GPS) signals are available, an extended Kalman filter is utilised to filter and combine data from the GPS (for position) and the Inertial Mass Unit (IMU) (for heading and odometry). This has been successfully achieved and tested on the UGV in real time. Indoor localisation is achieved through the use of SLAM and this has been successfully simulated, however has not yet been tested in real time.
Mapping has been successfully achieved and many maps have been created through the use of mapping techniques applied to data from the LiDAR. The maps that have been created successfully include an occupancy grid, a vantage map, an exploration map, a feature map, a UGV visibility map and an elevation map.

Control is a fundamental aspect of this project and has been developed in three stages, manual control, semi-autonomous control (where waypoints are chosen by the user), and fully autonomous control which includes cooperation between the UGVs. The first two stages have been completed and tested since path generation and path tracking have been successfully developed. Fully autonomous control is in development using a Vantage map for waypoint generation. This is yet to be integrated and time did not allow the implementation of this.

Computer vision is used to detect, identify and locate OOI, and has been successfully completed. OOI of different colours have been tracked even when lighting changes the hue of the object.

The final aspect of the project is the development of the GCS and the inherent communication solutions involved. The GCS has been designed as a stand alone station with the ability to display the detail of each UGV and the system as well as allow the human operator to interact with the UGV through a Human Machine Interface (HMI). The UGV is an integral component in the wireless communication network with much of the calculation occurring on the GCS computer. Additionally, the GCS is being used to house a GPS receiver in order to correct GPS measurements, thereby increasing the accuracy of GPS data.

Safety has been taken into account in the design through the inclusion of an emergency stop feature, both at the GCS (remotely) and also on the UGVs themselves (locally). Risk assessment, safe operating procedures and a detailed Failure Modes and Effects Analysis (FMEA) were performed and documented.

The individual future work for this project has been discussed in each chapter where a particular system concerned.
REFERENCES


Bradski, G & Kaehler, A 2008, Learning OpenCV: computer vision with the OpenCV library, O’Reilly Media, Beijing, China.


Foran, C n.d., QNX real-time course, Algonquin College.


Leuze Electronic 2010, *rotoScan ROD4plus/ ROD4-08plus area scanning distance sensor technical description*, p. 45.


Novatel 2010a, *FlexPak-G2*.


Shin, E 2001, *Accuracy improvement of low cost INS/GPS for land applications*, University of Calgary, Department of Geomatics Engineering.


Welch, G & Bishop, G 2006, An introduction to the Kalman filter, University of North Carolina, Department of Computer Science.

Yerraballi, B, n.d., Real-time operating systems: an ongoing review, Dept. of Computer Science and Engineering University of Texas at Arlington, viewed 26 September 2010,

1025: MAGIC 2010

Multi Autonomous Ground International Challenge

Volume II: Appendices
22\textsuperscript{nd} October 2010

\textbf{Supervisors}
Associate Professor Ben Cazzolato
Dr Steven Grainger
Dr Chris Madden

\textbf{Students}
Mark Baulis (a1147334)
Adam Cundy (a1162416)
Nigel Gaskin (a1160814)
Peter Hardy (a1149159)
Phuong Huynh (a1160609)
Sundar Komandurelayavalli (a1154422)
Konrad Pilch (a1160221)
Anton Skeketee (a1148726)
Benjamin Quast (a1162326)
Stella Wong (a1159937)
APPENDIX A : ADMINISTRATION

A.1 MAGIC 2010 INFORMATION PACK

This is the information pack published by the competition organisers detailing the requirements of the competition.
Multi Autonomous Ground-robotic International Challenge 2010

A US/Australia Initiative

8 -13 November 2010, Australia
# Document Change Summary

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial Release</td>
<td>30th June, 2009</td>
</tr>
</tbody>
</table>
Table of Contents

1. Introduction
2. Program Goals
3. Challenge Overview
4. Down Selection Process
5. Information Dissemination Process
6. Timeline
7. Location of Event
8. Expenses and Reimbursements
9. General Rules of Entry
10. Challenge Rules
11. An Example Timeline
   11.1. Preparation by Challenge Support Staff
   11.2. Challenge Start
   11.3. Teams Create an Operational Picture for Planning
   11.4. Initial Planning for First Challenge Phase
   11.5. UGVs Begin Executing Plan
   11.6. OOI Detection
   11.7. Communications Network Dropout
   11.8. Phase Completion
   11.9. Servicing of Vehicles
   11.10. Successive Phases
12. Additional Information
   12.1. Preparation
   12.2. Servicing, Repairs, DSL and DSZ
   12.3. Challenge Data
   12.4. UAV Imagery and Metadata
   12.5. Ground Truth
   12.6. UGV Identification
   12.7. UGV Endurance
   12.8. End of Phase
   12.9. Speed Limits
   12.10. Obstacles and Collision Avoidance
   12.11. GPS and DGPS
   12.12. GPS/DGPS & Outages
   12.13. UGV Mobility
   12.14. Disruptor and Sensor UGVs
   12.15. Narrow or Restricted Access
   12.16. Queueing
   12.17. Objects of Interest (OOI) and Signatures
   12.18. Sniper Action
   12.20. Automatic Target Detection/Recognition
12.21. Tracking and Locating Objects of Interest
12.22. In-Brief, Setup and Rehearsal
12.23. Sacrificial Use of UGVs
12.24. Indoor Lighting and Infrastructure
12.25. Cognitive Loads on Operators
12.26. Classified Data
12.27. Vehicle Safety
12.28. Autonomous Vehicle Configuration
12.29. Team Disqualification and Technology Demonstration

13. Guidelines for Submissions
13.1. Initial Submissions – 2nd October, 2009
13.2. Site Visits & Technical Presentations (2-13 November, 2009)
13.3. Site Visits (June 2010)

14. Evaluation of Initial Submissions
15. Judging and Determination of a Winner
15.1. Final Technical Paper and Presentation – 2010

16. Eligibility
16.1. Team Membership
16.2. Operations Team Membership
16.3. Support Team Membership
16.4. Participation & Sponsorship
16.5. Team Funding & Support
16.6. Definitions

17. Rules & Interpretation
17.1. Complaints Process

Figure 1: MAGIC 2010 – Phase I Graphic
Figure 2: MAGIC 2010 – Phase II Graphic
International Challenge (MAGIC) 2010 Down Under

1. Introduction
The Multi-Autonomous Ground-robotic International Challenge (MAGIC) 2010, is jointly sponsored by the US and Australian Departments of Defence; it will be conducted in Australia in conjunction with the Land Warfare Conference 2010. Research awards of US$750,000, US$250,000, and US$100,000 will be provided to the top three finalists that complete a series of increasingly complex assigned autonomous unmanned multi-vehicle intelligence, surveillance and reconnaissance (ISR) tasks within an allotted time limit.

The event, which is designed to demonstrate emerging unmanned technologies necessary to meet a range of current and future requirements, is open to national and international organisations within industry and academia. It will be conducted in two phases: a two step down selection to five teams; followed by a multi-Unmanned Ground Vehicle challenge that features multiple autonomous unmanned vehicles conducting coordinated ISR missions in a mock urban environment.

The ultimate challenge will take place during the week commencing 8-14 November 2010 at a location to be announced (date) within the state of South Australia. The five teams selected to compete will also be invited to attend and present at the Australian Land Warfare Conference in Brisbane, Australia commencing 15-19 November 2010. Down selection will take place in two phases: (i) an initial phase, which will be completed by 16th November 2009, in which ten teams will be initially selected and out of these up to five research awards of $100,000 granted for the maturation of the technology; and (ii) a final phase that takes place in June 2010, when the best five teams regardless of funding are selected to compete.

At the completion of the challenge, the successful teams will have the opportunities to seek funding through the Capability Technology Demonstrator (CTD) programs of Australia and the US.

The challenge is designed to test the ability of the multi-vehicle cooperatives to autonomously and dynamically coordinate, plan and re-plan their task allocation and execution strategies against a changing environment while simultaneously providing a unified situational awareness picture.

To succeed, a minimum of three unmanned ground vehicles (UGVs) supervised by a maximum of two operators must autonomously coordinate their activities to safely, efficiently and effectively explore and map their environment and detect, locate, classify, recognise, track and neutralise a number of static and mobile objects of interest (OOI).

Potential participants are encouraged to register their interest via the website www.dsto.defence.gov.au/MAGIC2010/ where more detailed information will be posted. A series of participants’ conferences will also be held in Adelaide, Australia, Frankfurt,
Germany and Warren MI, USA in July/August 2009 to brief interested competitors on all aspects of the challenge. Subject to interest other locations will be considered.
2. Program Goals
This challenge aims to improve the ability of dismounted ground forces to conduct zone reconnaissance in an urban environment by increasing safety and efficiency and reducing cost via the integration of autonomy. It is not about sensor development or vehicle mobility, but about shifting the Unmanned Vehicle System (UVS) state-of-the-art from manual and tele-operation to partial or full autonomy.

It anticipates integration of humans and multiple unmanned vehicle systems (UVS) to autonomously and dynamically coordinate, plan, and re-plan multi-UVS task allocation and execution strategies against changing environments whilst simultaneously providing effective situational awareness to potential users of the information. Increased autonomy, robot-to-robot collaboration and reduced operator workload are rewarded.

Specifically, the program aims to:

- Accelerate the development of autonomous and unmanned vehicle technology in areas that include: task allocation, multi-UVS control, tactical behaviour, machine intelligence, dynamic planning and re-planning, data and sensor fusion, human-machine interfaces, multi-aspect situational awareness, and systems integration.

- Improve the current human-to-vehicle control ratios by demonstrating that multi-UVS cooperatives can operate effectively with limited supervision by humans in realistic environments.

- Shift the perceptions within the technical and operational communities regarding the state-of-the-art of autonomous multi-vehicle control by demonstrating the augmentation of realistic military activity in changing environments.

- Attract and energise a wide community of participants to bring fresh insights to the problem of developing robust autonomous multi-vehicle cooperatives and to identify and transition technologies to meet emerging requirements.

3. Challenge Overview
Teams are challenged to develop a cooperative of unmanned ground vehicles that can coordinate their activities such that they safely, efficiently and effectively explore and map an environment while detecting, locating, classifying, recognising, tracking and neutralising a number of potential Objects Of Interest (OOI). Key information concerning the environment and OOI should then be combined for transmission to the UGV operators and judges for evaluation relative to ground truth information.

A maximum of two operators will be allowed to supervise at least three UGVs. The team will be allowed a total of 3 hours 30 minutes to complete a series of increasingly complex phases, comprising a number of mission tasks designed to test the ability of the UGVs to autonomously and dynamically coordinate, plan and re-plan their task allocation, execution and sensing strategies in changing environments.
The challenge will be conducted in a mock urban environment of the order of 500m x 500m and will comprise sealed roads, paths, buildings, trees, grassed areas, sandy ground, trenches, holes, safety barriers, curbs, and fences. In addition to any natural obstacles, additional obstacles may also be inserted. The buildings will be mainly single or double-storey and of brick construction, with tiled or tin roofs. A representative layout of the area depicting open, restricted and wooded areas, street and road layout, general topography, the location, number, and area of buildings, etc will be provided prior to the challenge in December 2009. However the precise coordinates of the area and location of the obstacles will not be available for distribution. The challenge will be conducted in daylight and illuminated indoor conditions and precipitation of as much as 1mm/hour.

Line of sight between the operators and the unmanned vehicles will not be possible at all times and teams may need to consider communications relay. The terrain may undulate slightly, but will essentially be flat. The UGVs may encounter positive and negative obstacles inside and outside buildings, but will not require special mobility or manipulation characteristics. Access to buildings will be through open doorways at least 0.9m wide. GPS will be available outside the buildings, but subject to the usual physical restrictions imposed by urban infrastructure. GPS will not be relayed inside buildings.

The location of OOI will not be known to teams a priori. These objects may be static or mobile and may be located inside or outside buildings. OOI will include humans. Mobile OOI will manoeuvre outdoors in unscripted patterns. Mobile OOI may be either hostile or non-combatants. All hostile OOI have the potential to “damage” and “destroy” a team’s vehicles if the UGVs are within their lethality zone. High resolution EO images of all OOI signatures will be made available to teams prior to the event.

During the challenge, the location of all detectable OOI outside buildings will be provided via a real time data feed to simulate the information provided by an Unmanned Aerial Vehicle (UAV). Each UGV cooperative will contain two categories of vehicles: sensor UGVs, which can explore and map the area and detect OOI, and disruptor UGVs, which have the ability to neutralise static OOI.

To complete the challenge the teams must: (i) accurately and completely explore and map the challenge area; (ii) correctly detect, locate, classify, recognise and neutralise all hostile OOI; and (iii) finish all tasks within 3 hours and 30 minutes.
4. Down Selection Process
The down selection process will involve a series of increasingly detailed technical reviews over a one-year period. To be considered for selection, teams are required to submit their initial proposals in electronic form by 2nd October 2009. The initial proposal requires competitors to submit a DVD or video and a written description of their current robotics research programs, experience with development of autonomous systems, search and tracking, multi-UAVS concepts, and a technical paper describing the likely capabilities, platforms, architecture, sensors, processing, testing plan, technical specifications, supervision and task allocation strategies, etc that they plan to use in the challenge. Forms and approved formats are available from the website, www.dsto.defence.gov.au/MAGIC2010/ and in Section 13: Guidelines for Submissions.

In October 2009 the initial proposals will be assessed by members of a Technical Assessment Panel (TAP) against criteria that are specified in Section 14: Evaluation of Initial Submissions.

Ten teams will be selected from the applications received and notified of their selection by 19th October 2009. These ten “semi-finalists” will then be asked to submit phased expenditure proposals for the $100,000 awards by 26th October and host site visits by the TAP in November 2009 where they will be invited to provide detailed presentations and demonstrations of capability. Finally, up to five research awards of US$100,000 per team will be granted to the most promising proposals. These will be payable 50% on commencement and 50% following a progress assessment approximately six months later (June 2010). The semi-finalists that do not receive funding may participate in the June 2010 progress assessment using their own resources, but must notify organisers of their intention by 4th December 2009. The TAP will conduct site visits to all competing teams in June 2010 for progress assessment and final down selection to five teams who will compete in MAGIC 2010 in Australia.

If the TAP determines that an insufficient number of initial proposals and/or interim progress demonstrations have satisfied the evaluation criteria, the challenge event will be re-scheduled.

5. Information Dissemination Process
A series of participants’ conferences will be held in Adelaide, Australia (31st July 2009), Frankfurt, Germany (4th August) and Warren MI, United States of America (6th August 2009) to discuss all aspects of the challenge. Subject to interest, locations in Asia are also being considered. These conferences are open to any interested parties that may wish to participate in the challenge and are aimed at allowing the potential participants to meet directly with US and Australian Defence representatives to discuss all aspects of the challenge. In particular, suggestions and comments on the draft set of rules will be invited to ensure they are both clear and consistent. The conference will comprise an overall description of the challenge, a rules briefing, and a simulation of the event mission scenario, as well as Q&A sessions on FAQ, R&D proposals, the website, information dissemination, the CCR/DUNS process, and funding. It will also permit networking, teaming and information-sharing between attendees.
A website [www.dsto.defence.gov.au/MAGIC2010/](http://www.dsto.defence.gov.au/MAGIC2010/) and an email distribution list for teams submitting applications will be established as part of the meeting. The website will be the forum through which all information concerning rules, rule interpretations, and informational updates are communicated to participants. The email list will simply assist in information dissemination (i.e. remaining up to date with the latest rules, etc is the responsibility of the participants). Those teams that are unable to attend the conference can download a video of the event and register for the email distribution list at the MAGIC 2010 website.

### 6. Timeline

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 July 2009</td>
<td>Official Announcement of Competition</td>
</tr>
<tr>
<td>31 July 2009</td>
<td>Participants Conference [DSTO – Adelaide]</td>
</tr>
<tr>
<td>4 Aug 2009</td>
<td>Participants Conference [Frankfurt, Germany]</td>
</tr>
<tr>
<td>6 Aug 2009</td>
<td>Participants Conference [TARDEC, Warren, MI]</td>
</tr>
<tr>
<td>4 Sept 2009</td>
<td>Submission of Intention to Compete forms</td>
</tr>
<tr>
<td>2 Oct 2009</td>
<td>Technical submissions (video/DVD, technical paper, etc)</td>
</tr>
<tr>
<td>19 Oct 2009</td>
<td>Teams notified of down selection and scheduling of site visits</td>
</tr>
<tr>
<td>26 Oct 2009</td>
<td>Ten down selected teams submit budget proposals</td>
</tr>
<tr>
<td>26 Oct 2009</td>
<td>CCR/DUNS application &amp; registration submitted</td>
</tr>
<tr>
<td>2-13 Nov 2009</td>
<td>Site visits by TAP – detailed technical assessments</td>
</tr>
<tr>
<td>16 Nov 2009</td>
<td>Five $100,000 research awards granted (50% funding paid)</td>
</tr>
<tr>
<td>7-21 Jun 2010</td>
<td>Site visits by assessment panel</td>
</tr>
<tr>
<td></td>
<td>Progress assessments (50% funding paid)</td>
</tr>
<tr>
<td>22 Oct 2010</td>
<td>Submission of Challenge Technical Paper</td>
</tr>
<tr>
<td>8-14 Nov 2010</td>
<td>MAGIC 2010 Grand Challenge Event</td>
</tr>
<tr>
<td>15-19 Nov 2010</td>
<td>Land Warfare Conference, Brisbane (winners announced)</td>
</tr>
</tbody>
</table>

### 7. Location of the Event

MAGIC 2010 will be held in the State of South Australia. The organisers will announce the exact location of the event in June 2010. No entrants will be permitted to use or access the site until the challenge commences.

### 8. Expenses and Reimbursements

The organisers will fund economy class air travel to/from the location of the challenge event and the Land Warfare Conference in Brisbane, Australia for three members of each competing team. Hotel accommodation and meals or meals allowance for three people will also be provided for the duration of their participation in accordance with Joint Federal Travel Regulations. Organisers will reimburse freight costs up to a maximum of US$3,000 per team. Organisers will also reimburse the hire of one rental van per team for the period of the challenge event.
9 General Rules of Entry

9.1 All UGVs must be autonomous, weigh less than 40kg (including fuel). For reasons of safety, maximum vehicle speed will be limited to 10 km/hr.

9.2 Communications between the operations team and their UVS may be by radio, infra-red, acoustic or other means so long as they are considered safe by the Technical Assessment Panel (TAP). No umbilical links are permitted.

9.3 Any sort of propulsion for the UGVs is acceptable if the design is deemed safe by the TAP during the review process.

9.4 After considering an application organisers may direct safety improvements that must be made in order to compete in the challenge.

9.5 Each UGV must be equipped with a termination mechanism (“E-Stop”) that can be triggered remotely on a judge’s command.

9.6 Each UGV must be equipped with a “Freeze” (i.e. administrative stop) mechanism that can be triggered remotely on the judges’ command. Freezing a vehicle means stopping the vehicle and immediately disabling any contribution that the vehicle can make to its own or the operators’ overall situational awareness picture. The vehicle may continue to draw and generate power.

9.7 In the event of equipment failure, UGVs must be able to be easily and safely disabled mechanically and electrically by organisers so that they may be lifted and transported out of the challenge area using four-wheel push carts.

9.8 The perimeter of the challenge area will be designated by a series of waypoints provided to teams immediately before the challenge. Vehicles manoeuvring outside the designated challenge area or in an unsafe manner must be brought back under control or terminated on the command of the judges.

9.9 Teams must supply all of their own vehicles and infrastructure such as ground control stations, communications, etc. 240VAC ~50Hz power will be available on site.

9.10 Organisers will provide teams with passive RF tags for each vehicle. These will be used by judges for determining true vehicle position during the challenge. The location information will be used for adjudication, safety and spectator-display purposes. Any use of this ground truth information for navigation or other situational awareness tasks is strictly forbidden.

9.11 Teams must record the navigation and mapping solutions of their UGVs at 1 Hz and provide it to facilitators in the appropriate format in WGS84 coordinates. Gaps in any mapping information will be considered to be areas that were not properly explored. This information may be provided to judges at the end of each phase, in real time, with latency, or intermittently, as appropriate.
9.12 Teams must also provide an output feed for each operator display used during the challenge. These feeds will be used to capture and display the operator interface to the judges and spectators on organizer provided displays. This output feed should contain all content on the operator display in standard SVGA format.

9.13 All participants are asked to complete an Intention to Compete form and submit it to the organisers by Friday 4 September 2009, preferably or as soon as possible. This form is available at www.dsto.defence.gov.au/MAGIC2010/ and should be transmitted by email to [MAGIC2010@dsto.defence.gov.au]. Submission of the Intention to Compete is not a pre-requisite for application submission.

9.14 All teams must submit a completed MAGIC 2010 Application Form on or before 2 October 2009 electronically by e-mail [MAGIC2010@dsto.defence.gov.au]. The application form is available at www.dsto.defence.gov.au/MAGIC2010/ website. The exact requirements of the submission are detailed on the website, but include a DVD/video and a written description of their current research programs and a technical paper describing the vehicles, architecture, sensors, processing, testing plan, technical specifications, supervision and task allocation strategies, etc.

9.15 Challenge finalists must submit a follow-on Technical Paper by 22 October 2010 describing the technical details of their proposal.

9.16 Teams must comply with all relevant national technology import/export policies. For Australia these are available from http://www.customs.gov.au/site/page.cfm?u=4226

9.17 All communications with organisers and judges must be in English.
10. **Challenge Rules**

10.1 To complete each challenge phase a team must declare that it has:
   a. Accurately and completely explored and mapped the entire phase area; and
   b. Correctly located, classified, recognised and neutralised all OOI in the phase.

10.2 To complete the entire challenge a team must declare that it has completed all three phases of the challenge within the 3 hours 30 minute time limit.

10.3 Teams may use a maximum of two operators to control or supervise the multi-UGV cooperatives, which must contain a minimum of three UGVs.

10.4 Each UGV cooperative must comprise a ratio of at least two “sensor” UGVs to one “disruptor” UGV. Teams are limited to a maximum of three disruptor UGVs, but may have as many sensor UGVs as they wish.

10.5 A sensor UGV may carry sensors and payloads that contribute to the overall mapping, object location, classification, recognition and operator situational awareness process and they may transmit information that may be shared between the other UGVs.

10.6 A disruptor UGV must be used to neutralise static OOI. It must carry an eye-safe laser pointer for designating these objects. A disruptor UGV may also carry sensors and payloads that contribute to its own navigation and situational awareness and this information may be transmitted to the operators. However, this information may not be transmitted to or shared with other UGVs and it may not contribute to the overall exploration and mapping process.

10.7 The disruptor and sensor functions may not be transferred between UGVs. Teams must make each class of UGV readily discernible to judges, both on the situational awareness displays and visually in the challenge area.

10.8 Mobile OOI and non-combatants will manoeuvre outside buildings at a maximum velocity of 6km/hour. They may stop, turn, about face, reverse or continue manoeuvring. They may also be located inside buildings, but when inside will remain stationary.

10.9 Mobile OOI has detection and lethality zones of 10m diameter, subject to building occlusion. The detection/lethality zone of mobile OOI will also be bounded by the walls of buildings such that entities inside a building cannot be detected or damaged by an OOI outside and vice versa. The zones are similarly bounded by the internal structure of buildings.

10.10 Any UGV entering the lethality zone of a mobile OOI will be deemed to have been detected and damaged. Judges will immediately command teams to “freeze” the UGV until the end of the phase.
10.11 Mobile OOI must be neutralized to complete the challenge. To neutralize a mobile OOI two UGVs must simultaneously view it. The lead operator must then clearly and correctly identify its location to judges and request that the OOI be neutralised. The OOI must then be continuously viewed and tracked by both UGVs for a period of 15 seconds. If this operation is carried out successfully the judges will announce that the OOI has been neutralised.

10.12 During one or more phases of the challenge at least one UGV will be lost to enemy “sniper” action. Snipers cannot be detected. The affected UGVs will be deemed to have been damaged and judges will command teams to “freeze” the UGV until the end of the phase.

10.13 Static OOI have a lethality zone of 20m diameter, subject to building occlusion. If detonated, any UGV, OOI or non-combatant within the lethality zone will be deemed to have been damaged or killed. The lethality zone of static OOI is bounded by the walls of the buildings such that entities inside a building cannot be damaged by an OOI outside and vice versa. The lethality zone is similarly bounded by the internal structure of buildings.

10.14 Static OOI have an activation zone of 5m diameter, subject to building occlusion. Any UGV entering the activation zone of a static OOI will cause it to detonate. The activation zone of static OOI is bounded by the walls of buildings such that a UGV inside a building cannot activate an OOI outside and vice versa. The activation zone is similarly bounded by the internal structure of buildings.

10.15 Any UGV entering the activation zone of an OOI that has not already detonated or been neutralised will cause the OOI to detonate and the UGV will be deemed to have been damaged. Judges will immediately command teams to “freeze” the UGV until the end of the phase.

10.16 To neutralise an OOI one or more sensor UGVs must locate and classify it and the team leader must communicate this information to judges. A disruptor UGV must then approach the OOI to within 2m of its activation zone and remain stationary while designating the OOI with an eye-safe laser “pointer” continuously for at least 30 seconds. The team leader must then request that the object be neutralised and the judges must concur before the OOI is deemed neutralised.

10.17 When static OOI detonate they affect all entities within their lethality zone, which is a cylinder not a sphere. OOI do not pose a threat once they detonate or have been neutralised. Mobile OOI do not activate static OOI.

10.18 False attempts to neutralise static OOI incur only the time penalty of the neutralisation process. Penalties will be imposed if non-combatants are in the lethality zone of a stationary OOI when it is activated or are incorrectly identified as a mobile OOI and neutralised.
10.19 The team leader must inform judges when they have completed each phase. Teams may not score points for any tasks associated with a phase once it has been declared concluded. At the conclusion of each phase, UGVs may access the next designated servicing zone (DSZ).

10.20 The challenge will be conducted under daylight conditions and precipitation of as much as 1mm/hour. Should the weather fall outside these conditions, the challenge will be interrupted (challenge time will stop) until conditions are once again within these limits. Teams will not be penalised for such outages.

10.21 Powering up, re-charging or replacing energy sources are permitted only at the completion of each phase. However, this may only be undertaken by the operations or support staff and within the designated servicing zones.

10.22 Minor repairs such as re-attaching a cable, electronic re-booting, or replacing a faulty payload module are permitted within the DSZ, but only after consultation with and under the direct supervision of the judges. Items must be of an identical configuration to those being replaced. Major repairs or modifications may not be carried out at any stage during the 3.5 hour challenge.

10.23 Any number of computers and communications sites may be set up at the ground control station (GCS). Computers and communications relay sites may not be set up elsewhere, but may be carried by UGVs participating in the challenge. Only the operations team may be located at the GCS. The support team may only be located at the DSL and/or DSZ.

10.24 Operators and support staff may not leave their designated zones during the challenge without permission from the judges. Operators and support staff may not exchange roles during the challenge.

All communications with judges must be via the team leader, except when a member of the service team wishes to service or affect a minor repair to a UGV within a DSZ or at the DSL. Under these circumstances, the designated representative within these areas may ask permission of the local/supervising judge.
11. An Example Timeline
In conjunction with Figures 1 & 2, which graphically represents a potential challenge area (together with OOI and non-combatants) the following describes a possible scenario. The pre-briefing and rehearsal sessions are assumed to have already taken place.

11.1 Preparation by Challenge Support Staff: Static OOI will be deployed inside and outside buildings prior to the commencement of the 3.5 hour challenge. Mobile OOI and non-combatants will start manoeuvring through the streets, or may already be located in buildings.

11.2 Challenge Start: The judges will call “start.”

11.3 Teams Create an Operational Picture for Planning: Based on information from the simulated UAV feed, teams might generate a registered overhead situational awareness picture containing the location and activity of any potential mobile OOI relative to the known infrastructure of the challenge area and any other a priori information provided by facilitators during the pre-briefing session, such as building access points.

11.4 Initial Planning for First Challenge Phase: Using this aerial situational awareness picture, autonomous mission planning and task allocation software might generate mission guidance for the UGV collective such that the finite resources of the UGV cooperative are tasked to explore and map the challenge area to locate, track, recognise, identify and neutralise OOI. The guidance might be optimised based on constraints such as: the location, orientation and type of terrain, accessible buildings and OOI present within the environment; the observed and potential motion of OOI; the robustness of the proposed solution to OOI and/or environmental uncertainties; the need to enter buildings; the individual capabilities of the participating UGVs; the benefits that derive from the association of UGVs into teams; communications or sensor scheduling requirements between the UGVs to enable this cooperation; any “no-go” or “difficult-to-go” zones; the need to manage power and access to servicing zones; UGV safety and deconfliction requirements; the prospect of losing particular classes of UGVs; the need to continually monitor specific areas or access points for other UGVs to carry out their missions, etc.

11.5 UGVs Begin Executing Plan: The sensor UGVs might then begin to explore their environment, searching for static and mobile OOI inside and outside buildings. As the sensor UGVs progressively explore and map their environment the aerial and ground situational awareness views could be fused to provide a single, more complete picture. This information might also be fused and integrated with applications such as geospatial information, track data, imagery and visualisation tools to provide enhanced situational awareness to the team leader.

11.6 OOI Detection: A sensor UGV might autonomously detect a static OOI and coordinate with a disruptor UGV to neutralise it. Based on the simulated UAV feed, another sensor UGV might be cross-cued to approach a potential mobile OOI and, while remaining at a safe distance, discriminate it from a non-combatant. Once this UGV has
performed this task, it might then continue to track and possibly pursue the mobile OOI while simultaneously disseminating this information throughout the cooperative in order to task other sensor UGVs to confirm its identity and location for the purposes of neutralisation. While either or both of these activities are taking place a mobile OOI or a non-combatant may be detected by a sensor UGV (or the UAV) having emerged from a location previously unobservable by the cooperative’s sensors. The system might then respond by autonomously and dynamically re-tasking all of the UGVs, re-calculating their objectives, re-directing payload activity based on the automatic manipulation and fusion of the data in order to classify the nature of the OOI and its trajectory. Alternatively, a series of feasible options might be presented to the operators who might select one.

11.7 Communications Network Dropout: A UGV might detect that it is losing communications with its operators and/or other UGVs and another UGV might be tasked autonomously to act as a temporary radio relay station. Alternatively, a UGV might be lost to enemy sniper fire or a static or mobile OOI that it did not detect. The cooperative might then be autonomously re-tasked to accommodate the loss of this unit.

11.8 Phase Completion: Over time, the UGV cooperative might progressively explore and map the entire phase area, ensuring that static and mobile OOI are neutralised. When teams believe they have fully explored and mapped all of the phase area (inside and outside buildings) and detected, recognised, classified and neutralised (as appropriate) all OOI the team leader will notify the judges that the phase is complete.

11.9 Servicing of Vehicles: All UGVs, including the frozen ones, may then be “un-frozen” and manoeuvred to the DSL/DSZ for servicing within either the DSL or the newly achieved DSZ. The team leader might also request that organizers collect some of the UGVs that have unexpectedly stopped working so that they may be serviced in the DSZ.. Alternatively, teams may immediately task some or all of their UGVs to continue with the next phase without servicing.

11.10 Successive Phases: The next phase can be expected to be more complex than the previous one. For example, the number of OOI may increase, their location may be more difficult to detect and the environments may become more navigationally complex and cluttered.
Figure 1: MAGIC 2010 – Phase I Graphic

Figure 2: MAGIC 2010 – Phase II Graphic
12. Additional Information
In cases of discrepancy, the Challenge Rules takes precedence over this information.

12.1 Preparation
Teams will receive warning calls of 30 minutes, 10 minutes and 5 minutes to start. They will then be notified when the challenge has commenced. The challenge clock will then run for 3 hours and 30 minutes or until the team leader states that the team has ceased to compete.

12.2 Servicing, Repairs, DSL and DSZ
A UGV may only enter the challenge for the first time from the DSL. Once it has crossed the DSL the UGV will be deemed to have entered the phase, although it may enter subsequent phases from either the DSL or DSZ to which the team has earned access. To enter the challenge from a DSZ, the UGV must first have exited the phase area at this DSZ. When a team earns the right to access a DSZ from the phase area obstacles blocking its access will be removed.

Vehicles may only be serviced within these zones. Servicing is defined as powering up or powering down the UGV and/or re-fuelling, replacing or re-charging batteries, etc. Teams may also affect minor repairs such as electronic re-booting, cable re-attachment and/or the replacement of damaged or ineffective external payload modules.

As a guide, minor repairs are defined as those that are quick to enact (< 60 seconds of manipulation), require no specialist equipment, and may easily be performed by untrained staff. “Opening up” or modifying a UGV or its configuration for any reason constitutes a major repair and is not permitted. All servicing and repair activities must be conducted in consultation with and under the direct supervision of the judges, who must be notified in advance and agree to any procedures being carried out.

Transferring data between UGVs entering and leaving the DSZ is also permitted, but both UGVs and their software systems must be configured to allow automatic pre-processing of any data without the need for human inspection of the data. The data may be copied to/from a flash drive onto the incoming UGV or via a communications protocol such as 802.11g or Bluetooth immediately prior to its entrance into the competition. No software may be transferred and no computer or other diagnostic hardware may be connected to either UGV until the challenge is declared complete. Other than that specified above or expressly permitted by the judges, UGV hardware and software may not be repaired or modified during the challenge.

12.3 Challenge Data
An approximate layout of the challenge area, including topography, nature of accessible areas, streets, the number and size of buildings, entry/exit points, etc will be provided to teams prior to the challenge.

Precise data and coordinates will be delivered to teams during their pre-brief on a USB 2.0 flash drive in formats previously described to teams. The individual vehicles,
cooperative, and software systems must be configured to allow automatic pre-processing of any data without the need for human-inspection of the data. The data may be copied from the flash drive onto team computers and/or disseminated among the vehicles. The data must not be removed from the challenge site.

Most data (challenge area information, required data formats, the signatures of OOI, etc) will be released to teams in or around December 2009.

12.4 UAV Imagery and Metadata
A static, geo-registered aerial image of the challenge area will be provided to teams at the pre-brief in a format previously released to teams.

During the challenge the location of mobile OOI and non-combatants that are visible to the UAV will also be provided to teams in real time in WGS84 coordinates at an update rate of approximately 1Hz.

The feed may be unintentionally interrupted for short periods and teams should have the capacity to cope with these outages and adjust their schemes when the signal is re-acquired.

Longer interruptions, for example that might result from equipment failure, are not anticipated, but should they occur the challenge will be suspended until the metadata feeds are restored. Penalties will not be imposed for such outages.

12.5 Ground Truth
The ground truth data will be superimposed by the organisers onto an accurately geo-registered map display and used by the judges for the purposes of adjudication, spectator-display and safety. Teams will not have access to the ground truth information. Interruptions in the flow of ground truth data caused by teams or their UGVs may result in penalties or operations being suspended for safety reasons.

12.6 UGV Identification
Teams must provide a simple UGV identification scheme that visually, uniquely, clearly and continuously identifies each UGV and its class to the judges in the challenge area and observing the situational awareness displays. Teams will be expected to demonstrate this during the mid-term review in June 2010. The scheme must be consistent across any situational awareness and ground truth displays.

12.7 UGV Endurance
The target endurance of any single UGV (i.e. the duration of the longest phase) should be about 90 minutes, but may be less. UGVs with longer endurance will obviously provide teams with a potential time advantage.
12.8 End of Phase
To complete a phase of the challenge a team must declare that it has accurately and completely explored and mapped the entire phase area and correctly located, classified, recognised and neutralised all OOI in the area.

Teams may also use their discretion and declare that they are moving onto the next phase before they complete the current phase. However, teams may not return to an earlier phase at a later stage of the challenge. For instance, a team may declare phase end when all UGVs have broken down or been damaged.

Once a phase is declared complete, all OOI within that phase are considered inert and they do not impact on the next phase.

At the end of each phase, teams may elect to service their UGVs at the DSL (or within the DSZ that they have just achieved), to continue with the next phase without stopping, or some combination of the above. At this point, all UGVs will also be un-frozen. These UGVs must be returned to the DSL or a DSZ. It is permissible for teams to use tele-operation for this muster process without penalty or to request that facilitators collect any broken or frozen UGVs and bring them to the DSZ’s, whereupon they may be serviced. However, teams may not enter the course themselves. Challenge time will continue to run during this period.

Organisers will lift and transport the UGVs out of the challenge area using four-wheel push carts. All attempts will be made to ensure that the UGVs are not damaged in any way, but ultimate responsibility for the design of a UGV that is robust to this process rests with teams.

12.9 Speed Limits
Maximum speed limits are imposed for safety reasons. Minor and inadvertent excursions may be tolerated (i.e. < 10% for periods of 5sec or less and twice per phase), but major or repeated excursions (i.e. > 10%, periods longer than 5sec or more than twice per phase) will result in a UGV being “frozen” or disqualified. Judges will use ground truth data to determine this information. Judges will inform teams of any speeding violations. Teams are able to complete the challenge without exceeding the speed limit.

12.10 Obstacles and Collision Avoidance
Effective navigation while avoiding collisions with obstacles and other UGVs is one of the primary guiding principles for this challenge. UGVs are expected to continuously monitor the path ahead and surrounding area for OOI, other UGVs and obstacles. Vehicles should not assume that either the UGV ahead or the mobile OOI are entirely predictable.

Collision with an obstacle is defined as the main body of a UGV contacting an obstacle. “Feelers” that sense and make contact with an obstacle do not constitute a collision.
12.11 GPS and DGPS
GPS is expected to be continuously available within the DSL and DSZ. GPS will also be available within the challenge area, but subject to the naturally occurring physical restrictions and environmental conditions imposed by buildings, trees, etc. No GPS signals will be relayed inside buildings.

Teams are permitted to set up their own DGPS service or use a commercially available one, but must declare to organisers that they intend to do so in their proposals and provide details of the service, any spectrum allocation requirements, etc. DGPS ground stations may only be set up during the set-up and rehearsal phases and in the OCS. Once the challenge has commenced teams may not interact with their DGPS infrastructure.

Sufficient GPS waypoints will be provided to allow UVS to uniquely identify key aspects of the challenge area (boundaries, buildings, etc). However, UGVs should also use perception-based navigation to negotiate obstacles and the inside of buildings, to traverse areas where the waypoints are not dense, and to detect variations in ground conditions.

12.12 GPS/DGPS & Outages
Vehicles should not exhibit excessive delays or erratically vary their direction of travel due to the intermittent loss of GPS/DGPS signals. Operation with degraded GPS/DGPS due to foliage or buildings is a requirement of the challenge. An inertial navigation system or another technique should be used to “fly-wheel” the UGVs safely and effectively through any such outages. A vehicle should attempt to re-acquire GPS rather than allow its map or derived situational awareness picture to “drift” or become distorted.

12.13 UGV Mobility
It will not be necessary for the UGVs to negotiate obstacles such as steps or doors, but it may be an advantage for them to be able to mount small discontinuous obstacles (e.g. 10cm curbs) as this may provide teams with a time advantage. Similarly, UGVs that cannot demonstrate adequate mobility across the different types of terrain (which will include tarmac roads, gravel paths, sandy areas and grass) may be disadvantaged.

12.14 Disruptor and Sensor UGVs
The sensor and disruptor UGVs must be easily identifiable to judges. For example, they might be discriminated electronically through the use of colour coding on the situational awareness display and physically using clearly visible different coloured flags on top of the different classes of UGV. The discrimination technique must be demonstrated to the TAP during the relevant site visits.

Disruptor UGVs must carry an eye-safe laser pointer for designating static OOI in order to simulate their neutralisation. This payload may be manually controlled by operators through tele-operation, but more points will be awarded if the process is automated.
12.15 Narrow or Restricted Access
Access to all buildings and other challenge areas will be through entrances that are at least 0.9m wide. For safety reasons UGVs should not touch the sides of doorways with the main body of the UGV and will be penalised if they do so.

During the pre-brief all potential access points to buildings will be identified. However, not all of the doors will necessarily be open or be able to be opened. Similarly, a number of obstacles will be put in place prior to the commencement of the challenge. These obstacles will not be moved or altered dynamically during the challenge.

12.16 Queueing
If multiple UGVs need to pass through a narrow gap they should demonstrate sufficient collaborative behaviour to form a physical or virtual queue. UGVs should not collide with one another while negotiating the gap.

12.17 Objects of Interest (OOI) and Signatures
High resolution images of all classes of OOI and organizer obstacles will be provided to the teams well in advance of the challenge (date).

Static OOI will be placed on the ground throughout the challenge area, both inside and outside buildings. They will not be reported in the simulated UAV feed, but are expected to be detectable by the UGVs using commercially available software and electro-optic sensors. For example, they may be represented by a backpack or a 15cm coloured disk that is readily distinguishable from its background.

Mobile OOI and non-combatants will manoeuvre throughout the outdoor challenge area. Where their position will be captured and transmitted by the simulated UAV feed (i.e. subject to building eave or tree-canopy occlusion) Mobile OOI and non-combatants will be detectable, classifiable and recognisable from a ground perspective using commercially available software and electro-optic sensors. Distinctive colour and pattern markings will distinguish mobile OOI from non-combatants and human inside buildings. Judges and safety officers will also wear distinctive attire.

OOI and non-combatants may be located inside buildings, but they will not be mobile. Humans inside buildings may not be neutralised, but must be located and positively identified.

During the neutralisation process for mobile OOI, UGVs are themselves vulnerable to detection and damage. If one or both UGVs taking part in the process are damaged during the 20sec neutralisation process, the neutralisation process must be re-started.

12.18 Sniper Action
At some point during the challenge at least one UGV will be lost to enemy “sniper” fire. Teams will know that their UGVs have been damaged when a judge declares that “UGV X has been damaged by sniper fire. Freeze UGV X.” This UGV must then immediately be frozen and may take no further part in the phase.
The performance of the UGV cooperative within the phase will then be evaluated from this point forward. Snipers cannot be detected by UAVs or UGVs.

12.19 Operator-to-Vehicle Interaction
Teams may not tele-operate any UGV in regard to their key low-level or moment-to-moment control tasks such as navigation or collision avoidance unless they first request permission from the judges that they be allowed to perform such actions. Significant point penalties will be applied for any such intervention.

Once judges have acceded to a tele-op request, operators are permitted to tele-operate their UGV. However, it is anticipated that such requests will only be to extract a UGV from a difficult or ambiguous situation and/or to assist it in resolving some conflict. Any such actions must be performed remotely (i.e. through the use of tele-operation as team members are not permitted to leave or enter the challenge area). UGVs that are unresponsive to operator commands and cannot be extracted by tele-operation must remain in situ until the end of the phase.

Teams are permitted to manually control the payloads and sensors onboard their UGVs and may assist UGVs with higher-order tasks such as mission allocation, multi-UGV coordination, trajectory optimisation, route/path deconfliction, and OOI detection and recognition. However, they will not score as highly as teams that carry out the challenge fully autonomously.

Human-UGV interaction on the higher-order tasks is expected to be limited to an operator selecting from or confirming a number of alternative strategies presented to him. At a physical level, such interactions are expected to take place (for example) through map-based interfaces in a “point-and-click” fashion, through interactive voice commands, or other modalities. The effectiveness of the modality of this interaction will be assessed in the judging criteria.

12.20 Automatic Target Detection/Recognition
Operators may assist UGVs in regard to initiating or confirming their detection, recognition and classification of OOI offered by automatic target detection/recognition (ATD/ATR) software. The aim of the challenge is not to test the detection or processing capabilities of the sensors per se, but ATD/ATR will be rewarded as this will reduce the cognitive load on the operators and allow superior autonomous tracking of static and mobile OOI.

12.21 Tracking and Locating Objects of Interest
Tracking an OOI is defined as keeping a sensor oriented on it for a continuous period of time and performing feature extraction and registration processing sufficient to continuously locate the OOI. It does not imply a need to control a vehicle’s sensor head in a pan-tilt sense. It does, however, mean that the relevant features of an OOI are recognised and tracked within the field of view of the sensor; panning and tilting the
sensor is optional. Teams will be expected to explain and demonstrate their detection, location and tracking strategies during the site visits.

Tracking an OOI is not expected to involve operator interaction beyond any initial detection or confirmation that detection has taken place. Automatic target re-acquisition will be rewarded relative to manual re-acquisition.

OOI must be located, recorded and identified to judges using WGS84 coordinates.

12.22 In-Brief, Setup and Rehearsal
Teams will be required to give a technical presentation to judges outlining their technology development and mission plans. They will then receive a detailed in-brief and the UGVs will be weighed. The following morning, teams will set up and test their equipment at the challenge site. During this period, in addition to their own system tests, teams will be expected to perform a number of safety, “freeze” and ground truth tests for judges. There will also be a period set aside for a rehearsal, followed by a short period of “reset” where teams may re-fuel, replace energy sources, etc. The trial run will then commence.

<table>
<thead>
<tr>
<th>Day</th>
<th>Time</th>
<th>Team 1</th>
<th>Team 2</th>
<th>Teams 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>06:30-08:00</td>
<td>08:00-09:00</td>
<td>09:00-09:30</td>
<td>09:30-13:00</td>
</tr>
<tr>
<td></td>
<td>Tech presentation</td>
<td>In-brief/weigh-in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mon</td>
<td>06:30-08:00</td>
<td>08:00-09:00</td>
<td>09:00-09:30</td>
<td>09:30-13:00</td>
</tr>
<tr>
<td></td>
<td>Setup/system test</td>
<td>Trial run/rehearsal</td>
<td>Final reset period</td>
<td>Challenge run</td>
</tr>
<tr>
<td>Tues</td>
<td>06:30-08:00</td>
<td>08:00-09:00</td>
<td>09:00-09:30</td>
<td>09:30-13:00</td>
</tr>
<tr>
<td></td>
<td>Setup/system test</td>
<td>Prelim. rehearsal</td>
<td>Final reset period</td>
<td>Challenge run</td>
</tr>
<tr>
<td>Wed</td>
<td>06:30-08:00</td>
<td>08:00-09:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Setup/system test</td>
<td>Prelim. rehearsal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
12.23 Sacrificial Use of UGVs
The use of UGVs to intentionally or sacrificially detonate mines is permitted, but risks killing non-combatants, which will be penalised.

12.24 Indoor Lighting and Infrastructure
Lighting conditions inside buildings are expected to be brighter than 100lux [TBC], whereas lighting conditions outside buildings are expected no brighter than 75klux [TBC].

Buildings may be expected to contain a range of internal infrastructure such as walls and other fixed obstacles, but no doors that require opening. All OOI will be placed on the ground, not on walls or on or under any infrastructure.

12.25 Cognitive Loads on Operators
Systems with autonomous navigation capabilities that coordinate and evenly distribute their mapping and OOI servicing tasks within the UGV cooperative and that impose low physical and cognitive demands on their operators will generate most points in this evaluation area.

12.26 Classified Data
No classified data or devices may be used in preparation for or during the challenge.

12.27 Vehicle Safety
RF safety standards, laser safety standards, acoustic safety standards and wireless E-stop units will be assessed by the TAP during down selection in June 2010. The TAP may suggest safety improvements that must be made in order to compete.

12.28 Autonomous Vehicle Configuration
All vehicles must be unmanned and no animals are permitted onboard. Only independent, untethered ground vehicles are eligible to participate in the challenge. The ground vehicles must be propelled and steered principally by traction with the ground. The type of ground contact (e.g. tyres, tracks, legs, etc or hovercraft) is not restricted. If hovercraft are used the maximum hover height is less than or equal to 1 “effective” rotor disk off the ground. Additional rules with regard to the safety of jumping and hovering UGV will be developed in the event that a team using this technology is selected.

The vehicles must not damage the environment or infrastructure in the challenge area and vehicle operation must conform to any regulations or restrictions imposed by the
applicable land-use authority. The vehicle must be able to travel on asphalt, concrete and paved surfaces without damaging these surfaces.

A system comprising UGVs and one or more sub-systems that are not physically tethered to the UGVs is permitted provided the sub-systems are not propelled or manoeuvred independently of the UGVs (as would be the case with a UAV). For safety reasons, the maximum height of any UGV (including any tethered or extending sub-system) must not extend more than 2m above the surface.

Any aspect of vehicle activity or operation that has an unacceptable impact on the environment is prohibited. Such activities might include destructive vehicle behaviour, the use of abnormally hazardous materials, and/or generally reckless operation. All potentially hazardous equipment or activities must be identified to organisers for review during site visits and the pre-mission-brief.

12.29 Team Disqualification and Technology Demonstration
In the event that a team is disqualified or the rules forbid it from meaningfully continuing to compete (e.g. equipment failure precipitating a major repair or modification) the team leader may request permission to be allowed to repair/modify its equipment and compete against the phase/challenge tasks. However, the team will not be eligible for any of the cash awards.
13. **Guidelines for Submissions**

13.1 **Initial Submissions – 2nd October 2009**
All initial applications must be received by challenge organisers no later than 5:00pm EST on Friday, 2nd October 2009. All parts of the application must be received by challenge organisers before the specified deadlines for a team to become eligible for participation in the challenge. Materials received after their respective deadlines will not be considered and will be destroyed.

The initial submission should consist of five (5) parts and be in the following format:
Technical Submission
For
[Title, Organization]

Point of contact details (Team Leader):
Name, email address, phone number, mailing address
Alternate POC, email address, phone number, mailing address

1 Technical Proposal (not to exceed 15 pages).
   a. Abstract
   b. Introduction
      i. Statement of the problem.
      ii. Conceptual solution proposed
      iii. Graphic overview of overall systems architecture
      iv. Work breakdown and milestones
   c. Ground vehicle component & systems
   d. UVS autonomy & coordination strategy (by task)
   e. Sensors, processing & mapping for UGVs
   f. Operations in GPS-denied environments
   g. Processing and fusion of provided Metadata (from UAV)
   h. Human-machine interface (HMI)
   i. Operational approach/missions operations strategy
   j. Risk reduction strategy
      i. EMI/RFI & electrical
      ii. Vibration & physical
      iii. Modelling & Simulation
      iv. Safety, E-Stop, Freeze & lost-link
      v. Communications architecture
      vi. Spectrum plan & usage
      vii. Test Plan
   k. Summary.

2 A written summary describing [Not to exceed 5 pages]:
   Source of resources and budget, sponsorship (if applicable), team composition (organisations), personnel and experience, and facilities

3 Enclosure A: A video showing current research projects

4 Enclosure B: Certificate of Team Funding & Support

5 Enclosure C: Site Visit Liability Statement.

Signature of Team Leader & Date
13.2 Site Visits & Technical Presentations (2-13 November 2009)
Teams will be notified of their down selection to host a site visit by 19 October 2009.

Ten teams will be asked to host site visits in November 2009. They will be expected to provide more detailed technical presentations lasting an hour. This will be followed by up to an hour of questions. Teams will also be allowed a further two hours to demonstrate any current autonomous or UGV research/technology relevant to MAGIC 2010.

Presentations are expected to focus on convincing the TAP that teams have a good extant knowledge of: autonomous unmanned ground vehicle design; multi-UGV autonomy and coordination; signal, sensor and image processing; navigation and mapping; processing and fusion from the simulated UAV feed; situational awareness and information fusion; human-machine interfaces; and mission operations relevant to the challenge.

In addition to the information in Initial Submission, the presentation should also include:

- Justification for (phased) expenditure of the $100,000 from the contract
- A description of a phased plan of execution for the challenge mission
- An explanation of the key operational tasks/techniques for:
  - Level of human UGV interaction should be highlighted for each task
  - Autonomous target hand-over between sensor & disruptor UGVs
  - Automatic target detection, classification and recognition
  - Collaborative target tracking and target cross-cueing
  - Determination of cleared buildings and phase areas
  - Automatic/optimal route planning for the UGVs
  - Communications relay (if appropriate)

All teams invited to give technical presentations to the TAP in November 2009 must also apply for a DUNS (Data Universal Number System) number (available from http://fedgov.dnb.com/webform) and once received complete CCR (Central Contracts Registration) registration (available from http://www.ccr.gov/), which will enable them to become a research contractor to the organisers should they be short listed. The CCR Registration should be submitted no later than 26 October 2009. Any questions on the CCR/DUNS registration or contracting process should be directed to Ms. Michi Sawa at the US Army International Technology Center Pacific (ITC-PAC), Akasaka Press Center, Tokyo, Japan (Telephone: +81-3-6385-3453 and Email: michi.sawa@us.army.mil).

13.3 Site Visits (June 2010)
At the site visit in June 2010 the TAP will evaluate progress towards proposed objectives. During this visit, teams will also be expected to demonstrate a range of UGV functionality that includes the integration of ground truth systems, E-stop and “freeze” termination mechanisms for their UGVs. Emphasis will be given to partial or complete demonstrations of each of the detection, recognition, identification, localisation and neutralisation tasks. Partially or fully functional user interfaces will also be evaluated for integration of the mobile OOI indicator data into a situational awareness display. Method of multi-UGV control and UGV collaboration will also be discussed in detail.
14. Evaluation of Initial Submissions
The following equally weighted factors will be used in evaluating the initial submissions:

- Completeness of application and conformity to the Guidelines for Submissions outlined in this document.
- Demonstration of a strong grasp of the problems associated with the challenge and an articulation of a clear, credible and complete approach to overcoming them.
- Demonstration of a deep quantitative understanding of the key design points relating to autonomous multi-UVS cooperatives and presentation of a reasoned, optimised design based on preliminary results.
- Description of a testing methodology that will be used to demonstrate the performance of the proposed multi-UVS cooperative.
- Team composition, resources, facilities and experience capable of addressing the multiple and interdependent human, technology and systems integration problems posed by this challenge.

The papers will be evaluated by multiple reviewers, who will provide a quantitative score for each evaluation factor using the following scale:

- 4 Points – No weaknesses; all elements are presented clearly and convincingly.
- 3 Points – Minor weaknesses or shortfalls; all of the required elements are covered, but theory or justification is not entirely clear or complete.
- 2 Points – Modest weaknesses or multiple minor shortfalls; the required elements are substantially covered, but there are a number of theories or justifications that are not entirely clear or complete.
- 1 Point – Major weaknesses or multiple serious shortfalls; the submission is not complete and does not fully explain or justify several assertions.
- 0 Points – Sections are entirely missing; technical assertions are not supported.

The reviewer’s score for each submission is computed as the total score for each of the five criteria (i.e. maximum value 20 points). The composite score is the average of the reviewer’s scores.

The Technical Assessment Panel will then meet to review, compare and adjudicate the relative experience of the teams and the technical merits of the most successful proposals relative to the challenge goals.
15. Judging and Determination of a Winner
An international panel of judges will verify compliance with all rules.

Teams must provide the navigation and mapping solutions of their UGVs as well as the identity and locations of all OOI at the end of each phase. This information may be provided in real time, with latency, or intermittently, as appropriate, although real time at an update rate of approximately 1 Hz or faster is preferred and rewarded.

To complete the challenge the judges must consider that a team:
   a. Has accurately and completely explored and mapped each phase area
   b. Correctly located, classified, recognised and neutralised all OOI, and
   c. Achieved the above within a total time of 3 hours 30 minutes.

Teams will be awarded a maximum of 800 points for their challenge operations and a further 200 points for their technical submission and presentations to judges, making a total possible score of 1,000 points.

The challenge operations will be judged against three over-arching sets of criteria: mission level, systems level and technical success. Broadly, high levels of individual and collaborative autonomy will be rewarded and human intervention penalised. Also, the degree to which the UGV cooperative is able to share the workload successfully will be rewarded (i.e. a cooperative of 10 UGVs each mapping 10% of the area will score more points than one in which one UGV maps 90% of the area).

At a mission level, criteria such as the percentage of targets correctly and incorrectly detected, located, recognised, and neutralised, the percentage of phase area explored and mapped, and the accuracy and timeliness with which this is achieved will be used. Teams will receive a maximum of 400 points for their mission level assessment. Greater weight will be placed on the later phases, which will be more complex.

At a systems level, criteria such as the number of UGVs handled by teams, the workload experienced by teams, the human-machine interface, the amount of time spent interacting with the cooperative, the number and nature of these interventions, and the degree to which UGVs autonomously and successfully share and coordinate their various activities will be used. Teams will receive a maximum of 300 points for their systems level assessment.

At a technical level, criteria such as the robustness, reliability and survivability of a team’s UGVs, the capacity of the UGVs to autonomously plan, re-plan, and then execute their tasks against dynamically changing priorities, and the navigation and mobility capabilities of the individual UGVs will be used. Teams will receive a maximum of 100 points for their systems level assessment.

In the event that no team achieves adequate performance the cash awards may not be made. However, the judges may also present non-monetary awards for: innovation, best individual UGV performance and best multi-UGV coordination strategy.
15.1 Final Technical Paper and Presentation – 2010
All teams down selected to compete in the challenge must also submit a detailed technical paper by 22 October 2010 [Max 25 pages]. This paper will serve as an outline for the technical presentations given by each team the evening prior to their demonstration and should describe the detailed technical approach of their solution. Judges will be anticipating this paper and the presentation to cover all of the sections above (see Initial Submissions and Technical Presentations).

A maximum of 200 points will be awarded for the technical paper and final presentation and should be framed such that a reader can understand the team’s entire approach/design without having seen previous papers or designs.

Based on a team’s technical presentation judges may award 100 points for:
- Overall format, completeness and readability/clarity (10 points)
- Innovation and elegance of overall technical solution (35 points)
- Mission strategy, vehicle-operator ratio and user workload (25 points)
- Craftsmanship, durability, portability of UGVs (10 points)
- Systems integration and testing (20 points)

The Final Technical paper should include [Max 25 pages]

Abstract
Introduction
- Statement of the problem
- Conceptual solution proposed
- Overall systems architecture
Ground vehicle component & systems
UVS autonomy & coordination strategy (by task)
Sensors, processing & mapping for UGVs
Operations in GPS-denied environments
Processing & fusion (from the simulated UAV feed)
Situational awareness tools
Human-machine interface
Mission operations strategy
Risk reduction strategy
- EMI/RFI & electrical
- Vibration & physical
- Modelling & Simulation
- Safety, E-stop, Freeze & lost link
- Communications architecture
- Spectrum plan & usage
- Test plan

Summary
All papers submitted in October 2010 will be published on the website and as part of the Proceedings of the Land Warfare Conference and potentially in a special issue of an archival robotics journal.

16   Eligibility

16.1 Team Membership
A team comprises individuals identified to the challenge organisers on the team roster. There are no limits to the number of individuals that may be listed on the team. However, there are limits to the number of individuals allowed into the designated areas of the challenge. The team roster may be changed, but organisers must be notified in writing that this has occurred.

Each team must designate an individual to serve as a Team Leader. The team leader must be at least 21 years of age and will act as the primary point of contact with challenge organisers during the challenge. The team leader will sign the application, is responsible for providing a Certificate of Team Funding and Support and Site Visit Liability Statement (available from www.dsto.defence.gov.au/MAGIC2010/), and must be present at the Technical Presentation and November/June Site Visits.

The team leader is the only person authorized to directly communicate with the judges and organizers with regards to the selection process, funding, and to discuss judging decisions.

Team leadership may be transferred from the team leader to another eligible individual, although there may only be one team leader at a time. Transfer of leadership occurs when the organisers have received and acknowledged transfer of team leadership in writing.

During the challenge there will be two participating components of the challenge team: an operations team, and a support team. Team leadership and membership of operations and support teams may not be changed (other than in exceptional circumstances) after June 2010. Any such applications must be made in writing, with strong justification, to the challenge organisers.

16.2 Operations Team Membership
The operations team comprises up to 2 individuals identified to the challenge organisers on the team roster. Only these individuals are allowed to enter the designated operator’s zone and/or operate/supervise the multi-UVS cooperative during the challenge. They may not leave the operator’s zone, but are permitted to power up, re-charge or replace energy sources on the UGVs that are in the operator’s zone. Operations team members may also affect minor repairs after permission has been granted by the judges as per normal repair judging procedures.

Each team shall designate an individual to serve as its Operations Team Leader during the actual challenge event. The team leader must be at least 21 years of age and is the
only member of the operations team permitted to interact with the judges during the event. The operations team leader may be the team leader.

16.3 Support Team Membership
The support team comprises up to 3 individuals identified to the challenge organisers on the team roster. Only these individuals are allowed to enter the designated servicing zones during the challenge. They may not leave the designated servicing zones during the challenge, but are permitted to power up, re-charge or replace energy sources on a UGV. Teams may also affect minor repairs such after permission has been granted by the judges. Support teams must designate a single individual in each DSZ/DSL to serve as the support team leader to communicate with judges.

16.4 Participation & Sponsorship
Corporations and non-government organisations may participate as teams or as sponsors.

Universities, university research centres, colleges of learning, polytechnics, schools, their employees and graduate and undergraduate students may participate in this competition unless they receive direct funding or support through a grant, contract or other transaction for the purposes of participating in or developing equipment for this challenge. This does not prohibit support or funding provided by the organisers of this challenge.

Teams receiving direct funding or support for this competition from national or state governments, government agencies or other state or federal government organisations are not eligible to participate. However, individuals employed by these agencies or organisations may participate as team members as long as they do so outside their official responsibilities and not as part of some work-related duty or assignment. Government travel funds and government-related travel may not be used to support the challenge team.

16.5 Team Funding & Support
The cost of developing, fielding, and insuring entered vehicles is the sole responsibility of the individual teams.

Teams must submit formal proposals to receive the $100,000 research awards granted by the organisers during the down selection process. Teams may use additional funds, but must identify the amount and source of this funding during the technical presentations and down selection process.

Each team leader must sign and submit a Certificate of Team Funding and Support which will contain the following certifications (forms are available from the challenge website at www.dsto.defence.gov.au/MAGIC2010/):

- No funding used in the foreground design, development, construction or operation of the systems employed in this challenge has been or will be charged to a grant, contract or other transaction from any national government, either directly through such work or indirectly through government-reimbursable R&D, government-funded independent R&D, overhead or general and administrative accounts. This restriction includes funding to pay for labour, travel, equipment...
leases, or other services that are applied directly to the design, development, construction or operation of the challenge vehicles.

- No portion of the Foreground IP used in the UGV systems (including the GCS) has been paid for or will be paid for, wholly or in part, using direct government funding. This exclusion does not apply to Third Party IP.
- Government-owned equipment or facilities have not been used and will not be used in the design, development or operation of the UGVs unless the equipment or facilities are made available on a cost-reimbursable basis.

This certification does not prohibit:

- The use of government-sponsored information such as GPS signals, cartographic products, or government-developed software routines that are openly available.
- The use of any technologies that are commercially available to all teams.
- The use of facilities, services, equipment or funding supplied to the teams by the organisers of the challenge.
- The use of paid vacation time by government employees and contract employees to support a challenge team.

Each team leader must also sign and submit a Site Visit Liability Statement which will contain the following certifications (forms are available on the website):

- The location chosen for the site visit complies with all site visit specifications,
- The team holds harmless and indemnifies the US and Australian Governments and their employees and any MAGIC 2010 contractors for all claims of liability arising from the site visit, and
- The test location, test UGV, and test activities are in accordance with national, state and local laws and regulations.

16.6 Definitions

**Government Funding** – Government funding refers to compensation in the form of salary or travel expenses and any form of funding, supplies, equipment or reimbursement that is paid for by a national government, direct contractual efforts or through any form of overhead account, independent R&D grant, general and administrative account or other similar means funded by any national or international government entity. Funds received in the form of a grant that originated with a government shall also be considered government funding. Prize money awarded in a government-sponsored, publicly open competition shall not be considered government funding. This definition covers national and international government organisations, whether located in the United States, Australia, or in any other country or territory.

**Commercially Available** – Commercially available refers to services and materials sold, leased or licensed to the general public.

**Openly Available** – Openly available refers to services and materials that are available to anyone without charge, such as software that is available for public download or GPS signals.
**Intellectual Property** - Intellectual Property or IP means all copyright (including moral rights) and all rights in relation to inventions (including patent rights), registered and unregistered trademarks (including service marks), registered and unregistered designs, confidential information (including trade secrets and know-how), and circuit layouts, and any other rights resulting from intellectual activity in the industrial, scientific, literary and artistic fields recognised in domestic law anywhere in the world.

**Foreground IP** – Foreground IP means IP which is not Background IP and which is created under or otherwise in connection with this challenge or any approved subcontract, other than Third Party IP.

**Third Party IP** - Third Party IP means that IP which is owned by a party other than the Commonwealth Government of Australia, the Federal Government of the United States, the Challenge Participants or their approved sub-contractors.

**Background IP** – Background IP means IP, other than Third Party IP that is in existence at the date of the challenge announcement or is subsequently brought into existence other than as a result of a Team’s participation in this challenge.

**Team** – A team comprises two parts: a qualified team leader and any other individuals who may have been appropriately designated by the team leader as team members in the application. Team members may contribute their individual labour, individually-owned materials and equipment, and individual funds to a team.

**Team Leader** – A team leader is an individual identified to the challenge organisers on the team roster responsible for the following: primary point of contact for team communication with challenge organisers, signatory of the Certificate of team funding and Support, signatory of the Site Visit Liability Statement, [contract signature], and who will be present at all site visits.

**Team Member** – A team member is a team leader or individual who has been identified on the team roster by the team leader as a team member during the submission process.

**Team Sponsor** – a team sponsor is an organisation that contributes labour, materials, services, facilities, equipment or funds to a team.

**17 Rules & Interpretations**

The rules posted on the website [http://www.dsto.defence.gov.au/MAGIC2010/](http://www.dsto.defence.gov.au/MAGIC2010/) are the official governing set of regulations and guidelines of the Joint US-Australian Autonomous UVS Challenge 2010. At any time prior to the event, requests for rules clarifications should be sent to [MAGIC2010@dsto.defence.gov.au](mailto:MAGIC2010@dsto.defence.gov.au). In general, these clarifications will be placed on the website. However, organisers will hold confidential any questions that are designated as team-proprietary.

The chairman of the judges is the final authority on all rules and mission execution of MAGIC 2010. The chairman of the judges has the authority to change aspects of the
mission execution and provide interpretation of the rules at any time and in any manner that is required. The chairman will ensure that all interpretations are made available to all teams to maximum extent possible under the team-proprietary guidelines. The MAGIC 2010 organizing IPT retains the decision-making authority on all prize and contract awards pending recommendations of the chairman of the judges. The chairman’s decisions regarding the rules are expected to be based on a number of factors including: safety, compliance, fairness, challenge goals, environmental protection and efficiency.

17.1 Complaints Process
All complaints will be investigated by a committee of judges, which must include at least three judges who are not involved with the specific issue or any persons in any complaint.

Teams raising legitimate complaints will not be disadvantaged in any way. However, teams falsely or deliberately fabricating a complaint may be penalised or disqualified.

There are three approaches available when a team has firmly established in its mind that it has reason to complain. These are:

- Direct approach which, if unresolved, can lead to an informal complaint which, if unresolved, can lead to a formal complaint, or
- An informal complaint which, if unresolved, can lead to a formal complaint.
- A formal complaint.

The direct approach, which is from one team leader to another, is preferred. However, if either party is not comfortable with this approach for any reason, then it may seek to have the matter addressed by either of the other two approaches.

An informal complaint may be made by any of the team leaders to any of the judges. It is the responsibility of the judge on the receipt of the complaint to discuss the concerns and issues involving the incident(s) and explore possible options with both parties for resolving the complaint informally. If informed that a team wishes to make an informal complaint, the judge should make brief notes of the following:

- Date and time
- Name of team lodging complaint
- Names of the team or person(s) alleged to be offending
- Brief points relating to the alleged incident

If, after mediation, the team that has complained determines the outcome to be acceptable and indicates that the complaint will not progress any further the judge’s notes are to be kept confidential and retained until the end of the competition. In the event that the complaint proceeds, the judge and the notes may be called upon as evidence.

If the above process fails to satisfy the team which has made the complaint and the problem remains unresolved, then the team lodging the complaint should be informed
that they may lodge a formal complaint [and has a further 24 hours to decide whether to proceed with that complaint].

A formal complaint may only be made by a team leader to the chairman of the judging committee in writing. The chairman of the judges must then decide whether the matter is one for investigation. The formal complaint should outline:

- Particulars relating to date, time, place, etc
- Names and individuals involved, judge who conducted mediation, etc
- Nature of the incident(s), preferably in chronological order
- Any witnesses, supporting evidence, other matters, etc

The chairman of the judges will then

- Convene a suitable panel of judges
- Collate all material pertinent to the complaint
- Disseminate all this information to all parties
- Arrange and liaise with all parties to arrange a suitable meeting time/location

Only the complaints committee, the team that is alleging the offence has occurred and the parties alleged to have committed the offence may attend the hearing.

The investigation by the committee of judges will establish whether the complaint is:

- Substantiated (i.e. the event did happen)
- Unsubstantiated (i.e. insufficient evidence for a determination)
- Proven false (i.e. the event did not happen)

Depending upon the outcome of the investigation the chairman of the judges may choose to disqualify, penalise or counsel the parties involved. Any penalty imposed is entirely at the discretion of the chairman and should be based on the recommendation of the committee.
A.2 MAGIC 2010 PROJECT CONTRACT

This is the contract between the student team and the School of Mechanical Engineering at the University of Adelaide.
Project Definition, Specification and Contract 2010

An important part of any engineering project is setting goals for the project, working towards achieving these goals, and evaluating your performance against these goals. As an engineer you will be evaluated on whether you have completed all of the goals of a project on time and to budget, and this may even determine whether or not you are paid for the work. Hence your performance against the goals for your project forms 15% of the total assessment for the Level 4 Design Project. You, your supervisors, and a moderator will assess your project performance against the goals you set below, as well as the significance of the achievements (difficulty of the goals) and the technical merit of outcomes.

Please note that if the scope of work changes during the course of the project, another of these forms must be completed, detailing and justifying the change in scope.

Part A to be completed and submitted to the principal project supervisor by 19th March.
Part B to be completed and submitted to the principal project supervisor by 29th October.

PART A

Project Number & Title: MAGIC 2010

Project Definition: (Briefly outline the project background, what the project aims to achieve, and other information relevant to achieving the expected project outcomes.)

Although the combined Strategic Engineering and University of Adelaide entry into the MAGIC 2010 has ultimately been unsuccessful, it still provides the basis for the development of a robotic platform that can be used for research. The original project definition is included below to outline the background and aims of the project.

This project will involve the design and build of multiple autonomous robots with the intention of competing in the MAGIC 2010 competition.

“The Multi Autonomous Ground-robotic International Challenge (MAGIC 2010) is jointly sponsored by the Australian and US Departments of Defence to attract innovative proposals from worldwide research organizations to develop next-generation fully autonomous ground vehicle systems that can be deployed effectively in military operations and civilian emergency situations.”

The application for MAGIC 2010 project was prepared and submitted by a Sydney based engineering consultancy, Strategic Engineering. They have been short listed

for the competition with the University of Adelaide on-board as a partner in the project. The international challenge requires competitors to use multi-vehicle robotic teams that can execute an intelligent surveillance and reconnaissance mission in a simulated dynamic urban environment. To complete the challenge, the constructed Unmanned Ground Vehicles (UGV’s) must:

(i) Completely explore and accurately map the challenge areas;
(ii) Correctly locate, classify and recognise all simulated threats and undertake appropriate response protocols; and
(iii) Complete all phases of the competition within a given time limit and with minimal or no human interaction.

As a requirement for participation in the MAGIC 2010 finals, the project must be ready for extensive demonstrations to the competition judges in June, after which the number of competing teams will be reduced to five. If the team is included in the final five, then the final competition occurs in November. Regardless of the June down-select outcome, the project goal is to have autonomous, coordinated mapping and identification of specifically defined objects by a team of UGV’s.

Currently a primitive chassis design exists from the work of Strategic Engineering, but the design only incorporates the wheels and motors. This will need extensive design work to incorporate all hardware required for operating and competing in the MAGIC 2010 competition.

Project Specification: (Specify all the items that affect the outcomes of the project and must be considered to ensure successful completion of the project. Also document any constraints on the project, e.g. budget)

The focus of the MAGIC 2010 project is to build multiple autonomous vehicles that are able to compete in the MAGIC 2010 competition. Should this be successful, three large monetary prizes are available and the University will receive international recognition for the project. Nevertheless, due to the international status of this project, many sponsors have already offered quality hardware and software at significantly reduced prices or at no cost. Costs of hardware and software will be split between Strategic Engineering and the University of Adelaide. Thus, some of the hardware will be available for this year’s honours project and also in the future for either continuing development of the UGV designs or for other honours and research projects.

To complete the challenge, the UGV’s need to be effective at identifying objects, determining their own position and generating maps of the real world from the sensor data available. The development of the software required is largely an area of continuing research. The development of machine awareness using a range of sensors including a Light Detection And Ranging unit (LIDAR), differential Global Positioning System (dGPS), Inertial Measurement Unit (IMU), camera and wheel encoders is an area that will be particularly challenging. Additionally, the development of complex maps and localisation appropriate for autonomous control is an area with numerous opportunities for further extension including further development in integrating these sensors using Simultaneous Localisation and Mapping (SLAM) and a Kalman Filter. The initial focus of the project is to compete in June and continue to the November competition. If this aim is not achievable due to circumstances out of the University and the project team’s control, then the
project focus will be on developing advances in machine vision, autonomous machine mapping, high accuracy autonomous localisation, autonomous object identification and high level autonomous machine strategy and interaction between multiple unmanned ground vehicles.

There are various objects required to be identified for the MAGIC challenge and these will be the focus of the computer vision implementation. It is expected that scans from the LIDAR will be able to detect most physical objects and so will be sufficient for most mapping purposes. This will include buildings, walls, doors, windows, trees, curbs and inclines. The machine vision will be designed to identify the objects of interest (OOI) in the MAGIC challenge. There are both static OOI and mobile OOI. The static OOI are large red bins and the mobile OOI are people wearing single colour overalls. Mobile OOI must be identified correctly, because there will be judges and observers on the course who are not OOI. Development of the capability to identify and track OOI will be important for proving the usefulness of the UGV’s.

This project does not aim to design and build all elements required to compete in the MAGIC 2010 challenge. It will have goals and deliverables focused on development of some aspects required for competition, while still offering a system that is useful and functional. This project will look at the following areas as outlined in the technical specification.

I. Technical Specifications

1. The design and construction of at least 2 mobile autonomous units. Each vehicle must:
   a. weigh less than 40 kg,
   b. be able to fit through doorways 900mm wide,
   c. be able to travel at speeds up to 10km/h,
   d. be able to operate for a minimum of 45mins,
   e. be able to travel in a variety of terrain, including but not limited to, sealed and unsealed roads and paths, grassed and sandy areas,
   f. have an effective and easily accessible emergency stop mechanism.
   g. be able to mount hardware to function as desired and provide impact protection for sensitive hardware,
   h. have an effective and easily accessible emergency stop mechanism,
   i. be able to mount hardware to function as desired and provide impact protection for sensitive hardware.

2. The vehicles must be able to communicate remotely with a central database (computer) and be able to work collaboratively to achieve their goals.
   a. The central computer should display mapping and object detection data as it is generated.
   b. The central computer must have the ability to freeze UGV’s in position in case of an emergency,
   c. The UGVs should be controllable from the central computer, with simple human interaction,
d. The vehicles must be able to communicate wirelessly with a range of up to 250m. This may be achieved by relaying the signal between UGVs.
e. Maximum rate of data transfer will be 100MBps combined for all communication. All required mapping and control data must fit within this data rate.

3. The vehicles must be able to determine their position to an accuracy of 200mm or better in outdoor environments, while moving and stationary. This will require each UGV to be fitted with:
   a. A differential GPS (Global Positioning System) unit,
   b. An Inertial Measurement Unit (IMU),
   c. Encoders connected to the drive motors,
   d. A laser scanner (LIDAR).

4. The vehicles need to completely map an urban environment (simulated or real) in 3D, at least 0.5m above and below the horizontal, to a resolution of at least 300mm. This will allow for accurate path planning and motion by the UGV’s.

5. The system needs to be able to achieve simple location and classification of the objects of interest defined by MAGIC 2010 competition.
   a. This will require a colour camera that can represent a 0.5m height at 10m away, using at least 70 pixels. The camera driver will output colour images in an image format compatible with the used computer vision and image manipulation software.

6. To complete the functional requirements, in addition to any other hardware outlined, the UGV’s will require:
   a. An on-board computer.
   b. High speed wireless communication capability.
   c. 24V nominal capacity rechargeable batteries and power electronics for a variety of power demands.

II. **Critical Components of the Project**

1. *Time Management* – for competing in MAGIC 2010 the project needs to be operational by May so testing can be completed before the June down-selection. However this is a considerably large and challenging task and was already significantly behind schedule before the students were given the project. Should the June down-selection be successfully completed the required improvements will be made to the existing robots and, if sufficient sponsorship is found, more robots will be created to expand the team by November.

2. *Communication* – the project aims to integrate the work of Strategic Engineering and staff and students at the University of Adelaide. This makes communication of significant importance, so that all areas of the project are completed and interact efficiently.
3. **Hardware and software** – most of the hardware and software requirements are being supported, in kind, by sponsors. All hardware will require significant testing and will only be available with a significant lead-time, possibly causing delays in the projects goals.

4. **Software/Programming** – This project has a large software focus, so that vehicles are able to operate autonomously. This will be challenging for the students involved because of limited previous programming experience.

5. **Integration** – integration of Software and Hardware between the different systems required for the project will be important for the UGV’s and the overall system to work efficiently. A comprehensive plan for the software architecture will be required to achieve this.

6. **Hardware Limitations** - all of the localisation and mapping is limited by the accuracy of the data obtained from the hardware available. A large portion of the project depends on how successfully, real, noisy data is analysed and converted into understandable information. Furthermore the useful results could be limited by the processing power of the hardware available.

### III. Project Constraints

1. **Budget** – There is currently $85K in funding made available by The University of Adelaide. Further funding of $50K has been made available by the MAGIC competition organisers to Strategic Engineering due to their submission being shortlisted. Further funding will be provided by Strategic Engineering for hardware, software and expert consultants. Initial estimates suggest the project cost will be around $300K, depending on the amount of in-kind support and not including the cost of student labour.

2. **Experience** – Students have limited or no experience in C++ and QNX programming languages, as well as SLAM, Kalman filters and integration of the hardware. Thus this may take quite some time for students to become proficient enough in these areas to start producing significant results.

3. **Time** – An operational system desired by May means that quality of work leading up to June may be affected by the time taken to achieve the set goals

4. **Location** – The University of Adelaide in Adelaide and Strategic Engineering in Sydney means communication is mainly constrained to telephone and email contact.
CONTRACT CHANGE:
Due to the unsuccessful efforts of Strategic Engineering and the University of Adelaide to progress to the final round of the MAGIC 2010 competition, the goals required for the students doing their honours project at the University need to be revisited. This has resulted since all of the goals were set with extremely short time frames that were required due to the Adelaide/Strategic team desiring to achieve as much as possible to show good functionality for the June down selection. In addition to this, the rapid development in the early stages of the project, has resulted in the need for some aspects of the project to be redone after the June down selection. As this rapid development was the will of our sponsors, we obliged and hence are submitting this adjusted contract for the MAGIC 2010 Honours project.

Project Goals: (Summarise the major goals of the project (usually 3 to 4) and include a description of how successful completion of each goal will be verified or measured at the end of the project. The project goals should be specific, measurable, quantifiable, and have an associated date of completion. Make sure the goals are reasonable given the time and resources available.)

Since our focus for the first half of this project was to achieve the requirements for the MAGIC 2010 honours project, many of the goals will still be related to this competition.

1. Mechanical and Electrical Design of the UGVs
   a) Conceptual Design
   The mechanical and electrical design of the UGVs was intended to involve a significant focus on the conceptual design and the process of design alteration to optimise physical result. However due to the very short time frame available our Sponsors Strategic Engineering pushed through a mechanical build which was suitable though not optimal. The students found that modifications were possible however they were limited. Hence the desired improvements and optimisation of the UGV will be developed into a conceptual design to replace the original conceptual design aspect of this project goal. This goal can be measured by the comparison of the optimised conceptual design and a discussion on the benefits it has over the actual mechanical design both for the UGV and PTU unit.

   b) Detailed UGV Design
   i) Design and build at least one Unmanned Ground Vehicle (UGV) in conjunction with our sponsors Strategic Engineering, this will include optimising the placement and arrangement of hardware in and on the UGV. This goal can be measured first of all by the successful completion of one or more UGV's. The secondly part of this goal, the optimisation, can be measured on the basis of successful weight distribution, relative ease of access to hardware for testing purposes and the fact that all necessary hardware was successfully and tidily placed in or on the UGV with internal space for access still available. Furthermore optimisation can be measured by specific modifications made by students on the original mechanical design built by our sponsors SE and whether these modifications are beneficial to the design and operation of the UGV.

   ii) Other aspects required for successful completion of this goal must include verification of a robust design such that it is not highly susceptible to
overheating or vibration damage.

iii) The UGV will conform with the still relevant MAGIC competition specifications, namely, entire weight less than 40kg, able to fit through a door of 900mm, able to travel up to 5km/h but restricted to no more than 10km/h and able to operate for a minimum of 45 minutes without requiring a battery recharge.

iv) PTU design. Successful completion will be measured provided it can enable the camera and LIDAR to view a 360 degree range horizontally, panned in less than 10 seconds and a 90 degree range vertically which must be scanned in less than 5 seconds.

2. Central Computer Database
One of the requirements for the MAGIC 2010 competition was that the UGVs must have a central governing command centre or Ground Control Station (GCS):

a) The creation of a central database with software that provides a Human Machine Interface (HMI). This goal can be seen to be successful, provided the GCS has the ability to remotely control the actions of the UGVs.

b) A further goal of the HMI is that it is able to displays mapping and camera vision data. It should also be able to monitor the status of each UGV, including elements such as their positions and basic fault detection.

c) It was decided that the University of Adelaide would like to design a portable version of this GCS even thought this was not required by the MAGIC 2010 competition. Hence a conceptual portable GCS design incorporating at least two LCD screens, a computer and peripherals will be designed.

3. Localisation
A self-localisation system will be developed, involving the integration of at least four sensors (GPS, IMU, incremental encoders and LIDAR). Two sets of algorithms will be developed to achieve this. When a GPS signal is available (such as in most outdoor areas), the vehicle position will be determined mostly by the GPS measurement because it is the most accurate of the sensors available. When a GPS signal is not available (such as indoors) or the variance of the signal is too large, the localization of each UGV will be determined using the IMU, wheel encoders and SLAM techniques. SLAM uses the physical features of the environment to create a virtual map that can be continuously calibrated by using the prominent features of the map. The measurements from all the sensors will be combined using a Kalman filter.

a) An Extended Kalman Filter will be developed to perform localisation using the GPS and IMU. This will use a very simple system model and the focus will be on investigating the accuracy of the different sensors. The accuracy of each sensor will be determined by trial and error and used to tune the filter to be most effective. The system will be capable of outdoor localisation to within 200 mm accuracy and able to cope with up to 20 second GPS outages.

b) The SLAM algorithm must also be added to the Kalman filter. Initial development of this will focus on identifying and tracking objects that are scanned from the LIDAR. When this is achieved effectively, the position of the robot can be interpolated from this data and hence added to the Kalman filter.

4. Physical and Conceptual Mapping
The vehicles will completely map a 3D urban environment. Two different categories
of maps (each with a number of variations) will be developed using this data. The maps will be used to show the operators locations of physical objects, the UGVs, danger areas and the extent of the mapping so far.

a) Physical Map
This will be a fusion of the following two maps that manipulate the sensor data in different ways:

i) 3D Occupancy Map – probability map of obstacle existence, which is used to create the Elevation Map and the Feature Map. This will use integer incrementing as opposed to traditional Bayesian Inference to create a 3D probability map of the surrounding environment. Coordinate transformations will be used to convert data input from local coordinates into global coordinates. The 3D Occupancy Map will not be available on the HMI.

ii) UGV Visibility – A UGV Visibility map will be created on demand, which will use the Feature Map and the positions of UGVs to infer the sum of UGV Visibility. The Visibility map will not be displayed on the HMI, however for the purpose of producing measurable project goals it will be able to be printed to screen in a numerical form.

ii) Elevation Map – The Elevation Map will be a 2D map containing height information about the environment surrounding the UGVs. The Elevation Map will be created from the 3D Occupancy Map and will give elevation to a resolution of 10cm. The Elevation Map will be displayed on the HMI.

b) Conceptual Maps
The successful completion of these maps will be determined by displaying these maps on the HMI so they can easily be read and understood.

i) The creation of an Explored/Unexplored Terrain Map that reveals the extent of UGV coverage of the map. Explored areas and unexplored areas will be clearly identifiable on the HMI.

ii) The development of a Feature Map that displays features in the UGVs environment such as obstacles that prevent the UGV from seeing an area, ditches, curbs, unknown and unnavigable areas.

iii) Creation of a vantage map that calculates the highest vantage position for the UGV to be in. This will further aid the UGV from avoiding obstacles and OOI.

iv) Path generation between waypoints. When two way points are inputted, a valid path for the UGV to follow will be generated between the two way points. The path generated will be displayed on the HMI.

5. Computer Vision
Computer vision algorithms will be used to locate and identify basic objects of interest. A fusion of the laser scanner and camera will be used to accomplish this. The camera software will scan and analyse the real-time picture feed from the camera in order to identify both mobile and static items of interest. Signatures of the objects of interest will be used to compare the findings and accurately inform the base station of the type of object observed.

The goals for the computer vision are:

a) Cooperation between the LIDAR and the camera to locate OOI achieved by:

i) Calculating the position of the OOI relative to the camera, and reading the appropriate LIDAR scan angle to retrieve the distance to the OOI within ±2 metres, or at least an estimation of the distance within ±5 metres using homography techniques if the LIDAR section fails.
ii) Sending the OOI location information to the GCS to be displayed on the created maps.

b) Accurate identification of both mobile and static OOI achieved by:
   i) Matching OOI known features (dimensions, colour, mobility, etc) with the types of OOI that need to be detected.
   ii) Classifying objects by colour (blue or red) and type (static or mobile) with a 75% success rate in live tests.

6. Software
Software for certain low levels aspects of the project is required for basic functionality. Software for some systems is provided with the hardware and so only limited work will be required in those cases. The two cases where software will be developed are:

a) A driver is required for the GPS unit to start the unit operating and then read data sent as an ASCII string from the unit over a RS232 connection and convert this to data formats suited to localisation purposes. This will be proven by successful operation to read data from the GPS, at a high (20Hz) data rate.

b) A driver is required for the IMU to start the unit and then read data in the form of hexadecimal strings. These strings arrive at a high data rate (50Hz) and must be converted to float format numbers. This is proven by successful operation to read the data and convert to the appropriate format.

c) Construct a driver to initialise all UGV cameras reliably from the Ubuntu systems achieved by:
   i) The detection and initialisation of multiple cameras simultaneously
   ii) Successful passing back of images to the GCS for display on the HMI.

Extension Goals: (In addition to the project goals described above that are essential for satisfactory completion of the project, include at least one additional extension goal that will be a challenge and that would exceed expectations for the project.)

(Completion by: 24/7/2010)

1. Generate complex visual maps, incorporating all UGV’s and the provided simulated UAV feed data
2. Implementation of high level control algorithms which use map data to create intelligent machine strategy.
3. Develop software that enables multiple UGV’s to engage in complex collaborative behaviour, in the form of working towards common goals.
4. Development of software that enables UGV’s to continue effective operation in the case of long term loss of communication with the central computer. This could include intelligent relaying of the signal between UGV’s to extend the networks effective range or the ability to continue operation entirely without the central computer.
5. Extension of the object identification algorithms to more complex objects and areas than those required for the MAGIC challenge.
6. Be able to operate for 3.5 hours continuously (entire November competition time frame).
7. An improvement of the accuracy of the localisation to within 150mm at all
times, both indoor and outdoor.
8. If funds allow build the designed portable GCS.
9. Implement texture information from the camera into the feature map to indicate terrain types (i.e. sand, grass etc).
A.3 PROJECT EXPENDITURE AND STUDENT HOURS

Table A.1 details the value of all hardware contained within one complete UGV.

Table A.2 details the value of all hardware contained within one complete GCS.

Table A.3 provides a breakdown of the weekly contribution to the project in terms of time spent per student.

Table A.4 provides a breakdown of estimated cost of each project member if they were employed by the University throughout the duration of the project.

A breakdown of the workshop time used per month is displayed in Table A.5.

A.4 RISK ASSESSMENT
A5  SAFE OPERATING PROCEDURE (SOP)
SAFE OPERATING PROCEDURE:
MAGIC UGV Operation

LOCATION DETAILS
School/Branch: Mechanical Engineering

TASK/ACTIVITY
MAGIC UGV Operation | Date: 30, June, 2010

PREPARED BY Name, Position and Signature (insert names of the supervisor, HSR, HSO and operator involved)

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Konrad Pilch</td>
<td>Operator</td>
<td></td>
</tr>
<tr>
<td>Ben Quast</td>
<td>Operator</td>
<td></td>
</tr>
<tr>
<td>Ben Cazzolato</td>
<td>Supervisor</td>
<td></td>
</tr>
</tbody>
</table>

HAZARD IDENTIFICATION: RISK ASSESSMENT
See Risk Assessment dated 29 / June / 2010
LOW

SAFE OPERATING PROCEDURE DETAILS

STOP

DO NOT OPERATE PLANT IF YOU HAVE NOT COMPLETED (1) THE COMPULSORY UNIVERSITY OF ADELAIDE OCCUPATIONAL HEALTH AND SAFETY INDUCTION COURSE, AND; (2) DO NOT POSSESS THE REQUISITE QUALIFICATIONS OR TRAINING FOR THIS PIECE OF PLANT.

Preparation – work area check:

☐ Ready access to and egress from the MAGIC UGV (min of 600mm clearance required)
☐ Area is free from grease, oil, debris and objects, which can be tripped over.
  (Use diatomaceous earth ("kitty litter") or absorption pillow to soak up grease, coolant, oil and other fluids)
☐ Area is clear of unauthorised people before commencing work.
☐ Be aware of other activities happening in the immediate area.
☐ Ensure that no slip and/or trip hazards are present.

Personal Attire & Safety Equipment:

☐ Approved closed toe type shoes must be worn at all times.
☐ Clothing must be tight fitting.
☐ Long hair must be confined close to the head by an appropriate restraint.
☐ Finger rings and exposed loose jewellery (eg bracelets and necklaces) must not be worn. Medic Alert bracelet must be taped if exposed.
☐ Gloves must not be worn when operating machine, excepting where specifying otherwise.
MAGIC UGV Pre-operational Safety Checks – Safety Precautions that MUST be Observed:

- Visual inspection of MAGIC UGV to verify it is in good operational order, ensuring no damage to any stationary or moving parts, electrical cords etc. Any unsafe equipment is to be reported to an authorised staff member and tagged out.
- Visual inspection of MAGIC UGV to verify that any local machine movement, such as of the pan-tilt unit, is not obstructed.
- Check that all hardware is properly connected.

**IF IN DOUBT, ASK**

- Ensure all sensors and peripherals, such as camera, LIDAR, laser pointer, ultrasounds, GPS are turned on prior to UGV initialisation.
- Ensure wheels are properly inflated.
- Ensure MAGIC UGV lid is closed and securely fastened.
- Locate and be familiar with the operation of the ON/OFF starter switch.
- Locate and be familiar with the operation of the Emergency Stop button.

**Operation Of Emergency Stop Button:**

- Switch lights and power on at their respective main switches, where required.
- Start machine using the Start Button.
- Check that the local Emergency Stop Button is working.
- Check that the remote Emergency Stop Button is working.
- Release the Emergency Stop button in preparation for next machine use.

**Operation:**

- Ensure light and power is switched on at their respective switches.
- Ensure batteries are connected properly before start up.
- Proceed to initiate MAGIC UGV operation, being mindful of the location of the local and remote Emergency Stop Buttons.
- **NEVER LEAVE MAGIC UGV RUNNING WHILST UNATTENDEd.**

**Turning Off:**

- Switch off UGV at the ON/OFF starter switch.
- Visual inspection to ensure no faults or mechanical damage has occurred during operation including disconnection of hardware or damage to wires.
- Clean up MAGIC UGV and surrounding area.
- Store MAGIC UGV safely and securely in the designated MAGIC UGV storage area.
- Ensure there are no hazards surrounding the UGV upon storage, including trip hazards, loose items on shelves etc.
- Check that there are no loose items sitting on the MAGIC UGV; clean up surrounding area.
General Safety

- Visual inspection of MAGIC UGV prior to use. Unsafe components to be tagged out and reported to Workshop Manager.
- Keep all parts of your body and attire safely clear of the rotating and moving parts, at all times.
- Ensure scheduled maintenance for this MAGIC UGV has been carried out; including scheduled testing of Emergency Stop.
- Assistance from other staff or students to be sought and support frames used when handling or transporting large, long and/or heavy sections of apparatus.
- **DO NOT** attempt to open lid or disconnect any hardware while MAGIC UGV is in use.
- NEVER LEAVE MAGIC UGV RUNNING WHILST UNATTENDED.
- Closed Toe Type Shoes must be worn during the operation of this machine.
- Loose hair to be securely tied back, loose clothing to be rolled up and/or secured, loose jewellery to be removed.
- Leather Safety Gloves to be worn, when carrying and transporting the MAGIC UGV, and;
- Switch Off machine before leaving it unattended.

**Note:** This Safe Operating Procedure must be reviewed:

a) after any accident, incident or near miss;

b) when training new staff;

c) if adopted by new work group;

d) if equipment, substances or processes change;

e) within 1 year of date of issue.
## Failure Mode, Effects, and Criticality Analysis for the MAGIC 2010 UGV’s

<table>
<thead>
<tr>
<th>Function</th>
<th>Failure Mode</th>
<th>Effect Of Failure</th>
<th>S(^1)</th>
<th>Potential Cause(s) Of Failure</th>
<th>O(^2)</th>
<th>Current Process Controls</th>
<th>D(^3)</th>
<th>CRIT(^4)</th>
<th>RPN(^5)</th>
<th>Recommended Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batteries</td>
<td>Mains electrical power failure</td>
<td>Amp will shut down and motor will become inoperable.</td>
<td>1</td>
<td>Grid failure.</td>
<td>2</td>
<td>Inspection and regular testing of mains leads.</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unlikely to cause damage of device unless other motors continue to operate, driving specimen or end effector beyond limits.</td>
<td></td>
<td>Earth leakage.</td>
<td></td>
<td>Two amplifiers run of a single mains phase. Should have more than adequate current supply to avoid tripping mains circuit.</td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Excessive current draw.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>Motor Position error (Bit 0)</td>
<td>Battery failure</td>
<td>Battery failure.</td>
<td>1</td>
<td>Grid failure.</td>
<td>2</td>
<td>Inspection and regular testing of mains leads.</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Two amplifiers run of a single mains phase. Should have more than adequate current supply to avoid tripping mains circuit.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Excessive current draw.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>

1 Severity Rating from 1 (no danger) to 10 (important)
2 Occurrence Rating from 1 to 10
3 Detection Rating. This number represents the ability of planned tests and inspections at removing defects or detecting failure modes.
4 Criticality = S x O
5 Risk Priority Number = S x O x D

Note: To perform this action, the bits must be set to 1 in the FaultMaskGlobal parameter. If a fault occurs that is set in both this fault mask and the FaultMaskDisable parameter, a
<table>
<thead>
<tr>
<th>Function</th>
<th>Failure Mode</th>
<th>Effect Of Failure</th>
<th>S&lt;sup&gt;1&lt;/sup&gt;</th>
<th>Potential Cause(s) Of Failure</th>
<th>O&lt;sup&gt;2&lt;/sup&gt;</th>
<th>Current Process Controls</th>
<th>D&lt;sup&gt;3&lt;/sup&gt;</th>
<th>CRIT&lt;sup&gt;4&lt;/sup&gt;</th>
<th>RPN&lt;sup&gt;5&lt;/sup&gt;</th>
<th>Recommended Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overcurrent (RMS) (Bit 1)</td>
<td>Damage to the Motor can occur if this parameter is not set with the continuous current rating required by the load motor that is used. An additional safety check prevents damage to the amplifier by dictating that the amplifier cannot output its peak current for more than one second</td>
<td>4</td>
<td>Attempting to drive motor beyond rated limit.</td>
<td>4</td>
<td>Use the ThresholdAvgIAmp parameter to specify the peak continuous current level at which an RMS current trap occurs. A fault is generated when the continuous current feedback from the controller exceeds this continuous current threshold for a period longer than the time specified by the ThresholdAvgITimeMsec parameter (set to 4000ms), setting the overcurrent bit in the fault status register (Bit 1). This parameter is intended to protect the motor from being driven too hard. For this reason, the amount of time it takes to trip this fault is decreased as the amount of current that is passed through the motor is increased. If the continuous current stays below the threshold, this fault never occurs. It does, however, allow for brief accelerations and decelerations where the current can deceleration occurs first, followed by a disable. The deceleration rate associated with a fault is specified by the DecelRateMoveAbort parameter, unless the fault is caused by the tripping of a hardware or software limit. In that case, a deceleration rate is calculated by the limits set in the LimitDecelDistCnts parameter and is compared to the DecelRateMoveAbort parameter. The faster deceleration rate is used.</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>Ensure that the ThresholdAvgIAmp parameter is set to the average current limit on the BM250 which 10A continuous.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>S1</td>
<td>Potential Cause(s) Of Failure</td>
<td>O2</td>
<td>Current Process Controls</td>
<td>D3</td>
<td>CRIT4</td>
<td>RPN5</td>
<td>Recommended Action(s)</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>----</td>
<td>------------------------------</td>
<td>----</td>
<td>--------------------------</td>
<td>----</td>
<td>-------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CW hardware limit active (Bit 2)</td>
<td>If fault flag correctly activated then amplifier will simply decelerate the actuator safely.</td>
<td>8 Actuator driven to limit.</td>
<td></td>
<td></td>
<td>8</td>
<td>Fault flag should be identified. Normally high has been used for this flag. If this fault flag fails then the CW software limit flag (Bit 4) should become active.</td>
<td></td>
<td></td>
<td></td>
<td>Ensure that CW and CCW Hall effect limit switches are at the appropriate ends of the actuator. Ensure that the Soloist amp is set up for normally high for hardware limit switches. Use the LimitLevelMask parameter to specify the hardware level of the limit inputs that indicates an active limit state. A value of 1 for a bit indicates that the limit is active high (which is our case).</td>
</tr>
<tr>
<td>CCW hardware limit active (Bit 3)</td>
<td>If fault flag correctly activated then amplifier will simply decelerate the actuator safely.</td>
<td>8 Actuator driven to limit.</td>
<td></td>
<td></td>
<td>8</td>
<td>Fault flag should be identified. Normally high has been used for this flag. If this fault flag fails then the CCW software limit flag (Bit 5) should become active.</td>
<td></td>
<td></td>
<td></td>
<td>Ensure that CW and CCW Hall effect limit switches are at the appropriate ends of the actuator. Ensure that the Soloist amp is set up for normally high for hardware limit switches. Use the LimitLevelMask parameter to specify the hardware level of the limit inputs that indicates an active limit state. A value of 1 for a bit indicates that the limit is active high (which is our case).</td>
</tr>
<tr>
<td>CW software limit active (Bit 4)</td>
<td>If CW Hall effect limit switch fault flag is not activated at end-of-travel, and this fault flag also fails to become active, then actuator will be damaged.</td>
<td>10 If the CW hall effect limit fails, then this fault flag should protect act to prevent actuator.</td>
<td></td>
<td></td>
<td></td>
<td>If the CW hall effect limit fails, then this fault flag should protect act to prevent actuator. Use the ThresholdSoftCW parameter to specify the minimum position set by the software. A fault is generated when the servo position command from the controller exceeds this parameter, setting the CW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>S1</td>
<td>Potential Cause(s) Of Failure</td>
<td>O2</td>
<td>Current Process Controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------</td>
<td>--------------------------------------------------------</td>
<td>----</td>
<td>-------------------------------</td>
<td>----</td>
<td>------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>software limit bit in the fault status register (Bit 4).</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The reference point for this fault is 0 from the absolute axis position. The fault is</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>generated only after a return to home position.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCW software limit</td>
<td>If CCW Hall effect limit switch fault flag</td>
<td>If CCW Hall effect limit switch fault flag is not</td>
<td>10</td>
<td></td>
<td></td>
<td>If the CCW hall effect limit fails, then this fault flag should protect act to prevent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>active (Bit 5)</td>
<td>active at end-of-travel, and this fault</td>
<td>activated at end-of-travel, and this fault flag also</td>
<td></td>
<td></td>
<td></td>
<td>actuator will be damaged.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>flag also fails to become active, then</td>
<td>fails to become active, then actuator will be</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>actuator will be</td>
<td>damaged.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amplifier fault (Bit</td>
<td>Motor will not operate - likely cause due to</td>
<td>The amplifier fault is caused by a failure of the</td>
<td>1</td>
<td></td>
<td></td>
<td>If the amplifier disables as soon as it is enabled and indicates a hardware failure,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6)</td>
<td>faulty motor connection.</td>
<td>hardware power stage at a low level.</td>
<td></td>
<td></td>
<td></td>
<td>a hardware failure occurred. Check for a short circuit on the motor by disconnecting the</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>There are two potential causes of this fault. The</td>
<td></td>
<td></td>
<td></td>
<td>motor power leads and attempting to enable the amplifier again. If the amplifier still</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>first cause is that the intelligent power module on</td>
<td></td>
<td></td>
<td></td>
<td>does not enable, there is a hardware failure on the board. This failure can also appear</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>the controller detects a short circuit across its</td>
<td></td>
<td></td>
<td></td>
<td>when bus power is disconnected from a Soloist with the AUXPWR option. In this case, connect bus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>phases or experiences an internal failure that</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**D3** | **CRIT** | **RPN** | **Recommended Action(s)** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>S¹ Potential Cause(s) Of Failure</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Position feedback fault (Bit 7)</td>
<td>Failure of encoders will disable amp and motors will not turn.</td>
<td>Prevents normal operation. The second cause is a hardware protection algorithm that shuts down the amplifier if peak current is commanded for more than 1 second. This trap increases the amount of time required to trip as the current command reduces and takes four seconds to shut down when 70.7% of peak current is commanded.</td>
<td></td>
</tr>
<tr>
<td>Velocity feedback fault (Bit 8)</td>
<td>Failure of encoders will disable amp and motors will not turn.</td>
<td>Position feedback faults indicate feedback device connection problems. These errors are usually caused when one or more signal connections to the Soloist are disconnected. If these faults occur, inspect the wiring of the specific feedback source.</td>
<td></td>
</tr>
</tbody>
</table>

If these faults occur when a system is initially wired, inspect the system for proper wiring and connections. Encoders require differential line-controller outputs.

If the amplifier enables for a second or two, but disables with an amplifier fault, the fault is caused by the hardware protection algorithm. The system can recover and continue to run; however, it indicates severe instability in the control loop, and you must verify that your system is properly set up and the motor and feedback configurations are correct.

power and attempt to enable the amplifier.

If these faults occur when a system is initially wired, inspect the system for proper wiring and connections. Encoders require differential line-controller outputs.
<table>
<thead>
<tr>
<th>Function</th>
<th>Failure Mode</th>
<th>Effect Of Failure</th>
<th>S</th>
<th>Potential Cause(s) Of Failure</th>
<th>O</th>
<th>Current Process Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hall effect input fault (Bit 9)</td>
<td>Failure of Motor Hall effect sensors will disable amp and motors will not turn.</td>
<td></td>
<td></td>
<td>Hall feedback faults indicate feedback device connection problems. These errors are usually caused when one or more signal connections to the Soloist are disconnected. If these faults occur, inspect the wiring of the specific feedback source.</td>
<td></td>
<td>Configure the Hall effect switches so that the switches are not all set high or low at the same time. If Hall effect signals are not being used, disable this fault by setting the <code>FaultMaskGlobal</code> parameter.</td>
</tr>
<tr>
<td>Max velocity command fault (Bit 10)</td>
<td>Running motor beyond rated velocity limit will cause overheating due to friction and eddy-current losses.</td>
<td></td>
<td></td>
<td>Velocity command too high.</td>
<td>Use the <code>ThresholdVelCmd</code> parameter to specify the maximum software commanded velocity. A fault is generated when the servo velocity command from the controller exceeds this parameter, setting the maximum velocity command bit in the fault status register (Bit 10). Setting this parameter to a value of 0 (default) disables the maximum velocity command check. This has been done here as the servo is operating in Torque Command mode and not Velocity Command mode.</td>
<td></td>
</tr>
<tr>
<td>Emergency stop (EStop) fault (Bit 11)</td>
<td>Unable to kill Soloist amplifiers via Estop.</td>
<td></td>
<td></td>
<td>Failure of Estop button on J-board. Unlikely given E-stop signal normally high. Only way fault can occur if output of button remains high.</td>
<td>Use the <code>FaultEstopInput</code> parameter to specify the physical hardware input that generates the emergency stop (EStop) fault. When the EStop input is opened (there is no connection and the system is in a fail-safe state), the EStop bit is set in the Fault Status register in Bit 11.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>S&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Potential Cause(s) Of Failure</td>
<td>O&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Current Process Controls</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>-------------------------------</td>
<td>---------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Velocity error fault</td>
<td>Bit 12</td>
<td></td>
<td>1</td>
<td>1</td>
<td>Use the <em>ThresholdVelError</em> parameter to specify the maximum servo velocity error. A fault is generated when the servo velocity error on the controller exceeds this parameter, setting the velocity error bit in the fault status register (Bit 12). Setting this parameter to a value of 0 disables the velocity error check. This has been done here as the servo is operating in Torque Command mode and not Velocity Command mode.</td>
<td>1</td>
</tr>
<tr>
<td>Task fault (Bit 13)</td>
<td>N/A</td>
<td></td>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved (probe fault)</td>
<td>(Bit 14)</td>
<td></td>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved (auxiliary fault)</td>
<td>(Bit 15)</td>
<td></td>
<td>4</td>
<td>8</td>
<td>When auxiliary input is opened (there is no connection and the system is in a fail-safe state), the auxiliary bit is set in the Fault Status register in Bit 15. Use the <em>FaultAuxInput</em> parameter to specify the physical hardware input that generates the auxiliary fault. A value of 0 for the <em>FaultAuxInput</em> parameter specifies that input from an optically isolated device (pin 17 on the J104) is used.</td>
<td>2</td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>S1</td>
<td>Potential Cause(s) Of Failure</td>
<td>O2</td>
<td>Current Process Controls</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>------------------</td>
<td>----</td>
<td>-------------------------------</td>
<td>----</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td>Safe zone fault (Bit 16)</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Failure of this Fault Status register will be picked up by a VI which monitors voltage from the 6 load cells. If the threshold exceeds +9.8V then the VI will shut down the torque command to the Soloist amps.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Motor temperature (°C) fault (Bit 17)</td>
<td>Motor will overheat and possible damage.</td>
<td>Attempting to drive motor with too much current or too fast.</td>
<td>Motor thermocouple was not purchased with motors so this fault flag has been disabled. Motor should not over heat since Peak and Average current limits have been set in the Soloist as has the peak Velocity command.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Amplifier temperature (°C) fault (Bit 18)</td>
<td>The amplifier temperature fault occurs when the amplifier power stage heats up, causing the temperature of the heat sink to exceed the maximum temperature setting (by default, 90°C).</td>
<td>This fault is usually caused because the amplifier is enclosed in a warm environment or is run at peak capacity (outputting too much current).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instantaneous current limit exceeded on motor</td>
<td>Motor coils will be damaged.</td>
<td>Peak current limit into motor exceeded.</td>
<td>Use the ThresholdClampIAmp parameter to specify the peak current command that the servo loop generates to the current loop. Caution: Damage to the equipment can occur if this parameter is not set with the continuous current rating required by the BM250 which is 20A continuous.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>Potential Cause(s) Of Failure</td>
<td>Current Process Controls</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet</td>
<td></td>
<td>Motor will run at maximum torque until fault flag initiation or failure.</td>
<td>Torque command from the FPGA gets pulled to one of the rails from either software fault, or possible hardware fault such as user driving the torque command output BNC on J-board with signal.</td>
<td>Motor will initially drive with maximum torque until EOT limits are encountered or current limits are exceeded.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wheel Motors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pan Tilt Unit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ultrasonics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lex Box</td>
<td>Mains electrical power failure</td>
<td>Grid failure.</td>
<td>Earth leakage.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Excessive current draw.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPOS Board</td>
<td>Mains electrical power failure</td>
<td>Loss of communications to Target computer. Loss of communications to Sokoist amplifiers</td>
<td>Grid failure.</td>
<td>Earth leakage.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Excessive current draw.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>Potential Cause(s) Of Failure</td>
<td>Current Process Controls</td>
<td>D³</td>
<td>CRIT⁴</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------</td>
<td>------------------------------------------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>Failure of Wireless</td>
<td>Unable to remotely control the VI</td>
<td>Failure to connect the device.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>remote control</td>
<td></td>
<td>Loss of power to the device.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>External interference.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LiDAR</td>
<td>Load cell limit circuit</td>
<td>Soloist amps will not be directly disabled via aux input fault flag.</td>
<td>Load cell limit detection circuit failure.</td>
<td>Secondary protection may be afforded by</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>failure</td>
<td></td>
<td></td>
<td>Target VI. If load cell voltage exceeds</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>user defined range (9.5V) then initiate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>fault flag in VI and halt.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Emergency stop button</td>
<td>Soloist amps will not be directly disabled via Estop fault flag.</td>
<td>Failure of output signal to fall low depressed.</td>
<td>Signal output normally high for fail safe</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>failure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jogging joystick failure</td>
<td>If voltage is tied high, then jogging will be at maximum as soon as mode entered.</td>
<td>Possibly overdrive actuators.</td>
<td>Jogging speed will be set low (maximum</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>on 10mm/sec).</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If analog input to FPGA from jogging</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>joystick not within acceptable deadzone</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.1V) then jogging mode cannot be</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>entered.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect of Failure</td>
<td>Potential Cause(s)</td>
<td>Recommended Action(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>If joystick fails when in jogging mode, then user will need to activate E-stop.</td>
<td>No communication. Camera is not detected on the network</td>
<td>Flat batteries, PoE cable unplugged or loosely connected.</td>
<td>Supply camera with power appropriately. Ensure the connecting system (e.g. GCS computer/laptop) is on the same network (172.16.0.##). Taking note of the UGV number, set the camera IP address appropriately (172.16.0.43 for UGV no. 4). Change the correct image format to RGB (RGB8packed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Camera will not send or receive data. No camera LEDs will be lit.</td>
<td>Camera operation failure values -1 to -4</td>
<td>Memory not allocated to save images or lack of memory (e.g. full hard drive on UGV).</td>
<td>Appropriate camera error numbers displayed in driver command line. Any errors are not dangerous, but will be readily noticeable on the GCS display (e.g. no images displayed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Driver</td>
<td>Camera driver cannot save images</td>
<td>Camera systems not created. (IP address possibly incorrect).</td>
<td>System Errors (21 to 26)</td>
<td>Read more detail about the error in the Baumer BGAPI Manual and adjust code accordingly. Check the camera is the correct Baumer model.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>If joystick fails when in jogging mode, then user will need to activate E-stop.</td>
<td>No communication. Camera is not detected on the network</td>
<td>Flat batteries, PoE cable unplugged or loosely connected.</td>
<td>Supply camera with power appropriately. Ensure the connecting system (e.g. GCS computer/laptop) is on the same network (172.16.0.##). Taking note of the UGV number, set the camera IP address appropriately (172.16.0.43 for UGV no. 4). Change the correct image format to RGB (RGB8packed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Camera will not send or receive data. No camera LEDs will be lit.</td>
<td>Camera operation failure values -1 to -4</td>
<td>Memory not allocated to save images or lack of memory (e.g. full hard drive on UGV).</td>
<td>Appropriate camera error numbers displayed in driver command line. Any errors are not dangerous, but will be readily noticeable on the GCS display (e.g. no images displayed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Driver</td>
<td>Camera driver cannot save images</td>
<td>Camera systems not created. (IP address possibly incorrect).</td>
<td>System Errors (21 to 26)</td>
<td>Read more detail about the error in the Baumer BGAPI Manual and adjust code accordingly. Check the camera is the correct Baumer model.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>If joystick fails when in jogging mode, then user will need to activate E-stop.</td>
<td>No communication. Camera is not detected on the network</td>
<td>Flat batteries, PoE cable unplugged or loosely connected.</td>
<td>Supply camera with power appropriately. Ensure the connecting system (e.g. GCS computer/laptop) is on the same network (172.16.0.##). Taking note of the UGV number, set the camera IP address appropriately (172.16.0.43 for UGV no. 4). Change the correct image format to RGB (RGB8packed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Camera will not send or receive data. No camera LEDs will be lit.</td>
<td>Camera operation failure values -1 to -4</td>
<td>Memory not allocated to save images or lack of memory (e.g. full hard drive on UGV).</td>
<td>Appropriate camera error numbers displayed in driver command line. Any errors are not dangerous, but will be readily noticeable on the GCS display (e.g. no images displayed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Driver</td>
<td>Camera driver cannot save images</td>
<td>Camera systems not created. (IP address possibly incorrect).</td>
<td>System Errors (21 to 26)</td>
<td>Read more detail about the error in the Baumer BGAPI Manual and adjust code accordingly. Check the camera is the correct Baumer model.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>If joystick fails when in jogging mode, then user will need to activate E-stop.</td>
<td>No communication. Camera is not detected on the network</td>
<td>Flat batteries, PoE cable unplugged or loosely connected.</td>
<td>Supply camera with power appropriately. Ensure the connecting system (e.g. GCS computer/laptop) is on the same network (172.16.0.##). Taking note of the UGV number, set the camera IP address appropriately (172.16.0.43 for UGV no. 4). Change the correct image format to RGB (RGB8packed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Camera will not send or receive data. No camera LEDs will be lit.</td>
<td>Camera operation failure values -1 to -4</td>
<td>Memory not allocated to save images or lack of memory (e.g. full hard drive on UGV).</td>
<td>Appropriate camera error numbers displayed in driver command line. Any errors are not dangerous, but will be readily noticeable on the GCS display (e.g. no images displayed).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Driver</td>
<td>Camera driver cannot save images</td>
<td>Camera systems not created. (IP address possibly incorrect).</td>
<td>System Errors (21 to 26)</td>
<td>Read more detail about the error in the Baumer BGAPI Manual and adjust code accordingly. Check the camera is the correct Baumer model.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>S&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Potential Cause(s) Of Failure</td>
<td>O&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Current Process Controls</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>--------------------------------</td>
<td>---------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Camera Errors (-30 to -39)</td>
<td>Camera driver cannot save images</td>
<td>Camera cannot be opened (IP address possibly incorrect). Error in device while creating a camera. Camera is busy. Trying to release a camera object while the camera is running.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature Errors (-50 to -59)</td>
<td>Camera driver may not save images correctly or at all. Images may not be refreshing on the GCS screen, or not displaying at all.</td>
<td>Features locked, not implemented, not available. No features or sub-features available. Feature not supported by BGAPI object.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value and Image Errors (-70 to -101)</td>
<td>Camera driver may not save images correctly or at all. Images may not be refreshing on the GCS screen, or not displaying at all.</td>
<td>Memory not allocated to save images or lack of memory (e.g. full hard drive on UGV). Incorrect programming.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIDAR</td>
<td>Mains electrical power failure to hub</td>
<td>Host computer unable to connect to Soloist amps. Host computer unable to communicate with Target computer.</td>
<td>Grid failure. Earth leakage. Excessive current draw.</td>
<td>Ethernet communications essential for Real-Time Labview operation. Suspect fault flag generated by Real-Time Labview.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Disconnected or damaged Ethernet lead</td>
<td>Host computer will be unable to connect to device.</td>
<td>Disconnected or damaged Ethernet lead</td>
<td>Host computer will be unable to communicate with the device. A fault flag will be issued.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Failure Mode</td>
<td>Effect Of Failure</td>
<td>S</td>
<td>Potential Cause(s) Of Failure</td>
<td>O</td>
<td>Current Process Controls</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---</td>
<td>-------------------------------</td>
<td>---</td>
<td>-------------------------</td>
</tr>
<tr>
<td>The Brain</td>
<td>Move outside map boundary</td>
<td>UGV data dramatic change</td>
<td>GPS reading is low</td>
<td>Incorrect positioning of OOI</td>
<td>Ultra sonics warning</td>
<td>Different size map exception thrown</td>
</tr>
</tbody>
</table>
Table A.1: UGV hardware value

<table>
<thead>
<tr>
<th>HARDWARE</th>
<th>COST</th>
<th>QUANTITY</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PTU Assembly</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RODi4 plus RoloScan LiDAR and cables</td>
<td>$3,728.50</td>
<td>1</td>
<td>$3,728.50</td>
</tr>
<tr>
<td>Baumer TXG20c-P Camera</td>
<td>$1,713.66</td>
<td>1</td>
<td>$1,713.66</td>
</tr>
<tr>
<td>Kowa LM 3NC1M Lens</td>
<td>$535.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTU Harness</td>
<td>$1,311.67</td>
<td>1</td>
<td>$1,311.67</td>
</tr>
<tr>
<td>Maxon Motor Tilt Motor assembly</td>
<td>$506.71</td>
<td>1</td>
<td>$506.71</td>
</tr>
<tr>
<td>24/5 EPOS2 Motor Controller</td>
<td>$491.55</td>
<td>6</td>
<td>$2,949.30</td>
</tr>
<tr>
<td>Maxon Motor Pan motor assembly</td>
<td>$237.18</td>
<td>1</td>
<td>$237.18</td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
<td></td>
<td></td>
<td>$8,480.82</td>
</tr>
<tr>
<td><strong>Drive Assembly</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24/5 EPOS2 Motor Controller</td>
<td>$491.55</td>
<td>6</td>
<td>$2,949.30</td>
</tr>
<tr>
<td>Maxon Motor Drive Motor Assembly</td>
<td>$280.00</td>
<td>6</td>
<td>$1,680.00</td>
</tr>
<tr>
<td>MBS T3 and Roadie Tyres</td>
<td>$12.49</td>
<td>6</td>
<td>$74.93</td>
</tr>
<tr>
<td>MBS RockStar Pro Wheel</td>
<td>$24.99</td>
<td>6</td>
<td>$149.93</td>
</tr>
<tr>
<td>Coupling</td>
<td>$11.00</td>
<td>6</td>
<td>$66.00</td>
</tr>
<tr>
<td>Cabling</td>
<td>$28.53</td>
<td>1</td>
<td>$28.53</td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
<td></td>
<td></td>
<td>$5,572.68</td>
</tr>
<tr>
<td><strong>Chassis</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panels</td>
<td>$500.00</td>
<td>1</td>
<td>$500.00</td>
</tr>
<tr>
<td>Screws</td>
<td>$20.00</td>
<td>1</td>
<td>$20.00</td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
<td></td>
<td></td>
<td>$520.00</td>
</tr>
<tr>
<td><strong>Power Management</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sanyo UR16650W cell Battery</td>
<td>$336.00</td>
<td>6</td>
<td>$2,016.00</td>
</tr>
<tr>
<td>P-DUKE 24V DC converter</td>
<td>$250.00</td>
<td>1</td>
<td>$250.00</td>
</tr>
<tr>
<td>P-DUKE 12V DC converter</td>
<td>$150.00</td>
<td>1</td>
<td>$150.00</td>
</tr>
<tr>
<td>P-DUKE 5V DC converter</td>
<td>$150.00</td>
<td>1</td>
<td>$150.00</td>
</tr>
<tr>
<td>Cabling</td>
<td>$100.00</td>
<td>1</td>
<td>$100.00</td>
</tr>
<tr>
<td>Innovative Energies SR100Li SmartCharger</td>
<td>$1,200.00</td>
<td>1</td>
<td>$1,200.00</td>
</tr>
<tr>
<td>Terminal Block</td>
<td>$6.62</td>
<td>29</td>
<td>$191.98</td>
</tr>
<tr>
<td>Circuit Breakers</td>
<td>$20.00</td>
<td>1</td>
<td>$20.00</td>
</tr>
<tr>
<td>E-stop button</td>
<td>$60.00</td>
<td>1</td>
<td>$60.00</td>
</tr>
<tr>
<td>On/off Switch</td>
<td>$150.00</td>
<td>1</td>
<td>$150.00</td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
<td></td>
<td></td>
<td>$4,287.98</td>
</tr>
<tr>
<td><strong>Communication and Computation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lex box Computer</td>
<td>$689.07</td>
<td>1</td>
<td>$689.07</td>
</tr>
<tr>
<td>Baumer GigE 4-port PoE Switch</td>
<td>$480.93</td>
<td>1</td>
<td>$480.93</td>
</tr>
<tr>
<td>Ubiquiti RouterStation Pro Wireless Modem</td>
<td>$160.00</td>
<td>1</td>
<td>$160.00</td>
</tr>
<tr>
<td>9dBi 2.4GHz Omni Antenna</td>
<td>$19.90</td>
<td>3</td>
<td>$59.70</td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
<td></td>
<td></td>
<td>$1,269.70</td>
</tr>
<tr>
<td><strong>Localisation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLEXG2-2-Generic GPS unit</td>
<td>$1,940.00</td>
<td>1</td>
<td>$1,940.00</td>
</tr>
<tr>
<td>GPS-702-GG Antenna</td>
<td>$1,000.00</td>
<td>1</td>
<td>$1,000.00</td>
</tr>
<tr>
<td>3DM-GX3-25 IMU</td>
<td>$1,421.25</td>
<td>1</td>
<td>$1,421.25</td>
</tr>
<tr>
<td>MB7060-XL-MaxSonar-WR 1 Ultrasonic Sensor</td>
<td>$62.51</td>
<td>2</td>
<td>$125.01</td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
<td></td>
<td></td>
<td>$4,486.26</td>
</tr>
<tr>
<td><strong>TOTAL PER UGV</strong></td>
<td></td>
<td></td>
<td>$24,617.44</td>
</tr>
</tbody>
</table>
Table A.2: GCS hardware value

<table>
<thead>
<tr>
<th>HARDWARE</th>
<th>COST</th>
<th>QUANTITY</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chassis</td>
<td>$2,197.46</td>
<td>1</td>
<td>$2,197.46</td>
</tr>
<tr>
<td>22&quot; NEC AS221 Capacitive Touch Screen Monitor</td>
<td>$1,192.50</td>
<td>1</td>
<td>$1,192.50</td>
</tr>
<tr>
<td>22&quot; NEC AS221 Non Touch Monitor</td>
<td>$379.00</td>
<td>2</td>
<td>$1,026.75</td>
</tr>
<tr>
<td>A PRIMA Monitor Mounting Arms</td>
<td>$625.00</td>
<td>1</td>
<td>$625.00</td>
</tr>
<tr>
<td>GPS-702 GG Antenna</td>
<td>$1,000.00</td>
<td>1</td>
<td>$1,000.00</td>
</tr>
<tr>
<td>FLEXG2-2 Generic GPS unit</td>
<td>$1,940.00</td>
<td>1</td>
<td>$1,940.00</td>
</tr>
<tr>
<td>Wireless antenna</td>
<td>$180.18</td>
<td>1</td>
<td>$180.18</td>
</tr>
<tr>
<td>Ubiquiti RouterStation Pro Wireless Modem</td>
<td>$160.00</td>
<td>1</td>
<td>$160.00</td>
</tr>
<tr>
<td>Dell T3600 Precision Desktop</td>
<td>$2,031.20</td>
<td>1</td>
<td>$2,031.20</td>
</tr>
<tr>
<td>Mouse</td>
<td>$50.00</td>
<td>1</td>
<td>$50.00</td>
</tr>
<tr>
<td>Keyboard</td>
<td>$60.00</td>
<td>1</td>
<td>$60.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$10,454.09</strong></td>
<td><strong>1</strong></td>
<td><strong>$10,454.09</strong></td>
</tr>
</tbody>
</table>

Table A.3: Weekly student project hours as of the 17th October

<table>
<thead>
<tr>
<th>Week</th>
<th>Student</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mark</td>
<td>201.25</td>
</tr>
<tr>
<td></td>
<td>Baulis</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Adam</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cundy</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Nigel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gaskin</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Peter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardy</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Phuong</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Huynh</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sundar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Komandurelyalavalli</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Konrad</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pitch</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Benjamin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quant</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Anton</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stoeckeler</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Stella</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wong</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Break 1</td>
<td></td>
<td>227.17</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exam 1</td>
<td></td>
<td>135.67</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exam 2</td>
<td></td>
<td>234.81</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Break 1</td>
<td></td>
<td>223.42</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Break 2</td>
<td></td>
<td>214.83</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Break 3</td>
<td></td>
<td>214.83</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub Total (Hrs)</td>
<td>850.477</td>
<td>874.7</td>
</tr>
</tbody>
</table>
Table A.4: Student labour costs based on hours worked as of the 17th October

<table>
<thead>
<tr>
<th>Costs YTD</th>
<th>Student</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mark Baulis</td>
<td>$20,444</td>
</tr>
<tr>
<td></td>
<td>Adam Cundy</td>
<td>$13,815</td>
</tr>
<tr>
<td></td>
<td>Nigel Gaaklin</td>
<td>$18,329</td>
</tr>
<tr>
<td></td>
<td>Peter Hardy</td>
<td>$12,572</td>
</tr>
<tr>
<td></td>
<td>Phuong Huynh</td>
<td>$10,072</td>
</tr>
<tr>
<td></td>
<td>Sundar Komandurelayavalli</td>
<td>$17,138</td>
</tr>
<tr>
<td></td>
<td>Konrad Pitch</td>
<td>$16,983</td>
</tr>
<tr>
<td></td>
<td>Benjamin Quast</td>
<td>$13,221</td>
</tr>
<tr>
<td></td>
<td>Anton Stekotee</td>
<td>$19,038</td>
</tr>
<tr>
<td></td>
<td>Stella Wong</td>
<td>$14,423</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>$156,037</strong></td>
</tr>
</tbody>
</table>

Table A.5: Total workshop hours used by project 1025 as of the 17th October

<table>
<thead>
<tr>
<th>Total Hours</th>
<th>Month</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrical Workshop Hours</td>
<td>Jan 0, Feb 0, Mar 1.5, Apr 0, May 13.33, Jun 27.83, Jul 5.83, Aug 0, Sep 0, Oct 48.5</td>
<td></td>
</tr>
<tr>
<td>Mechanical Workshop Hours</td>
<td>Breakdown of hours not provided</td>
<td>70</td>
</tr>
</tbody>
</table>
APPENDIX B: ADDITIONAL UGV AND GCS DESIGN INFORMATION

B.1 ADDITIONAL HARDWARE SELECTION INFORMATION

This section contains additional detail on the selection of the on-board computer, ethernet switch, wireless communications, LiDAR, camera, dGPS, IMU and material for the chassis and PTU structure. As well as information on some of the universal decision factors. These details were omitted from the main report due to the excessively long length of the discussion.

B.1.1 GENERAL DECISION FACTORS

There are several factors that continuously influenced the decision making process when selecting hardware for this project. These factors were cost, size, weight, reliability, and efficiency.

Cost was the most significant factor in deciding the majority of the hardware. Although quality hardware was desired, whenever discount or sponsorship was offered this was usually the deciding factor in whether that particular piece of hardware was chosen; provided it met the minimum requirements for functionality. The next two most influential decision factors were hardware size and weight. The UGV chassis size was limited due to the nature of the MAGIC competition course obstacles through which the UGV was designed to navigate. Due to this limited UGV total size there was a finite space inside the UGV, hence trying to minimise the amount and size of the chosen hardware was an important factor. Furthermore, the power required to move the UGV is highly influenced by weight. For these reasons the weight of all hardware needed to be taken into account and where possible kept to a minimum. The UGVs were designed to run autonomously, hence the need for reliability and efficiency was introduced as another governing factor when selecting the hardware although this was carefully balanced with cost as the main limiting factor.

B.1.2 ON-BOARD COMPUTER SYSTEM

Initially the three systems considered for data processing and storage were a microcontroller, a Lex Box and a laptop. A normal desktop computer was not considered as the large size would have taken up too much space inside the UGV chassis. Furthermore, the power requirements were generally in the order of at least 200 Watts, which would have required a cumbersome battery storage array to power each UGV for the hours of operation and as far as features were concerned, a laptop possesses all, if not more, features than most desktop computers. The major requirements on the selected hardware were, processor ability, number of relevant ports, power, weight, and memory.

PROCESSOR

The UGV processor was required to manage numerous inputs and outputs simultaneously. These include data feeds from seven sensors, local mapping calculations, local storage of some data as backup, and sending and receiving files and instructions over the wireless network. The provided
Lex boxes had Intel Atom 1.6Ghz processors inside which was adequate for the on-board processing desired. The presence of multi-core processors on the Lex Box was desirable in order to be able to perform more calculations (such as processing high resolution images from the camera), but these were not provided however there is the ability to upgrade to a dual core solution in the future if the current hardware is not sufficient.

**PORTS**

The majority of the sensors being used on the UGVs required either serial or USB connection hence the fact the the Lex box possessed six serial ports and two USB ports as well as ethernet and VGA ports was ideal. Even before the Lex box was made available the ability to find a portable processor option with six serial ports would have required significant customisation which was not required due to the fact that the Lex box was so highly suitable in this area.

**POWER**

The power requirements of the UGV computer were of medium importance but straightforward - the computer should use as little power as possible, without encroaching on the performance. The Lex Box uses even less than most laptops at 60 watts as opposed to approximately 90W for most laptops.

**WEIGHT & SIZE**

Similar to the power requirements, the weight and size of the UGV computer were to be minimised while not hindering other computer functions. Microcontrollers were easily the smallest and lightest, resulting from their low processing power. On the other hand, laptops were bulky in comparison, and carried the weight of a screen. The Lex Box took the middle ground in weight and size. Furthermore, the light aluminium case around the Lex Box offered a protected and ruggedised solution for the fragile electronics and also simplified attachment to the chassis.

**MEMORY**

The memory required for the practical functionality of the UGV was considered to be minimum 512MB RAM and 512MB of storage. The Lex Box had 1GB RAM options of 1, 2, 4 or 8 GB of Solid State Drive memory, or SATA hard disks.

From what has been stated previously it is clear that the Lex Box meets all the computational requirements onboard the UGV.

**B.1.3 Ethernet Switch**

When choosing the ethernet switch, the options available were to either source an industrial switch (which included Power over Ethernet functionality), or the addition of a small adapter to an existing single network cable. The decision matrix shown in Table B.1, compares the options that are viable to meet the necessary requirements.

Industrial switches are designed for permanent installations in inaccessible locations. These units are not regularly monitored and maintained, they are built to be able to withstand vibrations and
come in an easy to mount, small form factor. This makes the units generally more robust but also smaller and more energy efficient. Units that use the same power supply for the switch and PoE requirements will be more efficient than using a separate PoE adapter unit. It can be seen from Table B.1 that an industrial switch with built in PoE is the best option. The selected ethernet switch is of this type and has PoE functionality.

### B.1.4 WIRELESS COMMUNICATION

The common wireless communication methods are wireless USB, Wireless LAN (Local Area Network) based on IEEE 802.11, Ethernet Radio Modems, Bluetooth, and Zigbee, a fairly new and relatively unknown communication system which is similar to Bluetooth except with a lower data transfer rate and higher range. A decision matrix was not deemed necessary for this choice as the transmission ranges of wireless USB, Bluetooth and ZigBee are approximately ten metres, ten metres, and 100 metres respectively (ZigBee, 2010), which are inadequate to handle the UGVs communication requirements. Radio modems offer the best range of any of the options, as they use a 900MHz radio band which penetrates objects better than higher frequency signals. This low frequency also restricts the data transfer rate to be typically less than 200 kbps (Data-Linc Group 2009). Therefore Wireless LAN, which can have a bandwidth of up to 300Mb/s (ZigBee, 2010) and, under correct conditions, a range of kilometres was chosen for this project. Wireless LAN can be used to reliably transfer large data packets such as camera images from the UGVs to the GCS over reasonable long distances.

While Wireless LAN is a very common place technology, there are numerous standards and variations, all based on the IEEE 802.11 standards. The primary choice for the MAGIC project was between the 2.4GHz and 5GHz spectrum’s and between a traditional single antenna versus newer multi antenna systems. The 5GHz spectrum does offer slightly higher bandwidth than 2.4Ghz but also has the advantage of a frequency spectrum that suffers from less interference. Most Wireless LAN’s, such as those run by Universities, and mobile phones operate in a 2.4 GHz frequency band and this would significantly reduce the performance of another network in the frequency range. The key advantages of the 2.4 GHz frequency range are the accessibility of compatible hardware, which is within most new computers sold, and also an increased range compared to 5GHz options. There is a new 802.11n standard, which allows for multi antenna operation of compatible wireless interfaces. Multiple antennas systems reduce overall range but improve performance in areas with a large amount of reflections, which are typically caused by walls and buildings. Multiple antenna

---

**Table B.1: Decision matrix for ethernet switch selection**

<table>
<thead>
<tr>
<th>Function</th>
<th>Consumer switch with PoE adapter</th>
<th>Industrial switch with PoE adapter</th>
<th>Industrial switch including PoE</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Ports</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>Robustness</td>
<td>8</td>
<td>16</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>Weight, Size</td>
<td>10</td>
<td>14</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>Power Requirements</td>
<td>8</td>
<td>8</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>Cost</td>
<td>16</td>
<td>6</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>58</td>
<td>60</td>
<td>74</td>
<td>100</td>
</tr>
</tbody>
</table>
options typically have less line of sight range than single antenna options because the antennas are prone to generate interference with each other.

### B.1.5 Light Detection and Ranging Sensor (LiDAR)

The range sensors that were considered were 2D and 3D Light Detection and Ranging Sensors (LiDARs), ultrasonic sensors, and stereo vision camera systems, all of which are readily available and are commonly found in mobile robotics applications. The decision matrix which was used to evaluate sensor choice is displayed in Figure B.2.

<table>
<thead>
<tr>
<th>Function</th>
<th>2D LiDAR</th>
<th>3D LiDAR</th>
<th>Ultrasonics</th>
<th>Stereo Vision</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range and Accuracy</td>
<td>28</td>
<td>28</td>
<td>20</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>Information Rate</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Resolution</td>
<td>24</td>
<td>24</td>
<td>15</td>
<td>18</td>
<td>25</td>
</tr>
<tr>
<td>Cost</td>
<td>7</td>
<td>0</td>
<td>10</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Reliability</td>
<td>14</td>
<td>14</td>
<td>15</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Weight</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Power Supply</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>88</td>
<td>73</td>
<td>76</td>
<td>70</td>
<td>100</td>
</tr>
</tbody>
</table>

**Range and Accuracy**

Range and accuracy were given a high importance in the selection process, since an accurate interpretation of the surrounding environment is required to produce accurate maps of the environment used for navigation, strategy planning and viewing. The range and accuracy of the chosen sensor will also affect the accuracy to which the position of the UGV can be determined using simultaneous localisation and mapping (SLAM) methods, which is required for areas where the differential GPS (dGPS) is unavailable. Both range and accuracy are determined by the actual method used by each sensor to extract information about the environment. In this context, accuracy refers to the accuracy with which the distance between the sensor and the object can be determined. The LiDARs and ultrasonic sensors produce range information by emitting an energy pulse of known velocity and measuring the reflection time. The 2D and 3D LiDARs will have approximately the same range and accuracy as both emit high frequency light pulses. As LiDARs emit a high frequency light
pulse they will produce more accurate distance information than ultrasonic sensors which use a low frequency sound pulse. The typical range of a LiDAR is about 30 to 50 metres, whereas the range of ultrasonic sensors range up to about 15 metres. Distance information is obtained from the stereo vision system by using multiple cameras to take grey-scale pictures of the environment where the shade of the pixel can be used to interpret distance, with lighter pixels being closer to the camera than darker pixels. Once a picture has been taken from multiple angles the image pixels corresponding to the same point in space are identified, and the distance information obtained from the pixel shade is triangulated to identify the position of the pixel in three dimensional space relative to the camera. Stereo vision cameras can produce accurate range information for objects at a close distance, however at large distances the triangulation angle is decreased and the region of overlap between pixels is increased, causing the accuracy of measurements to be less reliable. To give an approximation of the range and accuracy of a stereo vision system, a data sheet for a Point Grey 3.8mm Bumblebee2 camera with a stereo resolution of 512x384 was consulted (PointGrey, n.d). It was found that for a range of 4m, the measurement error is approximately 7cm.

INFORMATION RATE

The rate at which information about the environment can be returned to the UGV was given moderate importance since it determines the rate at which the UGV can learn about its surroundings. The rate at which the UGV can learn is important, however not as important as the accuracy of the information obtained. A 2D LiDAR can return many scans per second with two examples of this being the SICK LMS200 (SICK, n.d) and the Leuze ROD4-08 plus (Leuze Electronic, n.d) 2D LiDARs which have scanning frequencies of 75 and 40Hz respectively. In contrast, a 3D LiDAR will return a scan every few seconds, with an example of this being the SICK 3DLS 3D LiDAR which has a scanning time of 3.2 - 26.64 seconds depending on the angular resolution chosen (SICK, n.d). The rate at which an ultrasonic sensor produces information will depend on the range of the sensor. For a range of 15 metres and using the speed of sound to be 343m/s the ultrasonic sensor can return information at a rate of about 10Hz however this rate will be increased if closer objects are detected. The information rate of a stereo vision system will depend on the resolution of the image and the data transfer rate from the camera. The Triclops SDK driver that comes with all Point Grey camera systems can perform up to 1,000,000 distance calculations per second. Using a low pixel resolution of 640x480 range information can be calculated at a rate of approximately 3Hz.

RESOLUTION

Resolution was given high importance since the resolution of the sensor used determines how much information can be obtained every time a sensor is read, and how accurate this data may be regarded. In the case of the LiDARs and sonar the resolution refers to the angular spread between the width of emitted pulses, and in the case of stereo vision refers to the number of pixels in the images. The angular resolution of both 2D and 3D LiDARs is typically between 0.25° and 1° and the beam spread is approximately 8cm over 50m, quantified for a LEUZE Rod4-08 plus LiDAR (Leuze Electronic, n.d). The resolution of a typical ultrasonic sensor can be adjusted and can range from a wide angle to a narrow spread. Stereo vision systems can produce high resolution images of the environment, however higher resolution images will take longer to process and hence will reduce the rate of information updating.

COST

The cost of the sensor was given moderate weighting. This was because the MAGIC 2010 project had a limited budget and a large amount of hardware was required. It was however a significant
requirement of the project to have high quality sensors for precise localisation to be possible. Taking into account the need to buy multiple pieces of the same hardware it was decided that 2D LiDARs were suitable in terms of cost. They are typically in the order of thousands of dollars, while 3D LiDARs commonly cost in the order of tens of thousands of dollars, hence the purchasing of multiple 3D LiDARs was not feasible and this was reflected on the decision matrix. Ultrasonic sensors are generally cheap to purchase, typically around a hundred dollars each for high quality sensors, hence they were scored high. Stereo vision systems range in cost from a few hundred to thousands of dollars depending on system quality. It was assumed that the cost of a high quality system capable of meeting our requirements would be approximately the same as that of a 2D LiDAR.

RELIABILITY

The reliability of the sensor chosen was of high importance, since the control algorithms used by the UGV derive information from the maps produced and a high level of accuracy was desired at all times. In terms of accuracy of data, when subjected to vibration, it is likely that the 2D and 3D laser range finders will perform the best out of all of the sensors. This is due to their high frequency pulse, which results in measurement times that are in the order of microseconds, and hence measurements are unlikely to be significantly affected by the low to medium frequency vibrations that the UGVs are subjected to in most running conditions. LiDARs can be purchased with IEC 68 vibration and shock ratings meaning that they can be operated in vibrational environments. In terms of reliability against dust and water, LiDARs can be purchased with IP65 environmental ratings, meaning they are protected from dust infiltration and limited water exposure. Ultrasonic sensors are relatively reliable sensors since their wide beam angle and low resolution means that they are not likely to be significantly affected by noise and vibrations. In terms of reliability against dust and water, ultrasonic sensors can be purchased with environmental ratings of IP67, meaning that they are protected from dust infiltration and can be immersed in low pressure water. Stereo vision cameras may be significantly affected by vibration. However, the severity of the effect will depend on shutter speed. A stereo vision camera with a low shutter speed will receive blurred images, which will not be useful in distance calculations. Stereo vision shutter speeds can generally be configured to be between 0.01 and 50ms and therefore can be resistant to low frequency vibrations. This however, does not exclude the effect of sudden shocks to the system, which will result in erroneous data. Whilst research was undertaken to find environmental protection ratings for stereo vision systems, none could be found. Given that stereo vision systems are common in outdoor mobile robotics applications, it will be assumed that a system with at least an IP64 rating could be obtained, meaning that it would be protected from dust infiltration and splashed water.

WEIGHT

Sensor weight was given low weighting relative to other sensor requirements. However, since the MAGIC competition guidelines stipulate a weight limit on the UGV, sensor weight must be considered. The weight of a 2D or 3D LiDAR is generally around 2-5 kilograms, which for a 40 kilogram UGV weight limit contributes up to 12% of the total UGV weight, thereby making it quite significant. Ultrasonic sensors typically weigh under 100 grams per sensor, which is light in comparison to the LiDARs. Stereo vision cameras generally weigh in the order of a few hundred grams, however this will vary depending on the model chosen.
POWER SUPPLY

The power supply needed to operate the sensor was given a relatively low importance since voltage converter circuits and batteries are easily attainable, thereby making most power requirements catered for. 2D and 3D LiDARs generally consume around 25W of power at 24V, however some scanners come equipped with heating elements to prevent lens fog, which can increase the power requirement to around 75W. The power requirements of a LiDAR can easily be supplied by a battery or battery array. Ultrasonic sensors will usually require a power supply voltage in the order of about 2 to 5 volts with a power draw in the order of about 0.1W, hence ultrasonic sensors are an attractive option for range sensors used in power conscious applications. Stereo vision cameras are generally low power consumption devices and may use up to around 20W of power. The advantage of stereo vision systems is that in addition to the ability to be powered by batteries, their low power consumption and high data transfer requirements means that POE stereo vision cameras are common, meaning separate power and communication cables are generally not required.

From previous discussion it is clear that the 2D LiDAR is the most suitable option for the MAGIC project.

B.1.6 CAMERA

Due to the environmental awareness requirements of having to locate and classify OOI in the unexplored territory, it was found that a camera was necessary. A number of different cameras were compared to determine the most suitable for this project. The types of cameras compared were webcams, consumer quality still and video cameras, professional quality still and video cameras, industrial monitoring cameras and security cameras. These camera types were compared with each other by the project requirements of image quality, cost, communication connections, physical connections, reliability, weight, size and power supply.

The first choice to be made was the camera type, as this would determine the variability of the other factors. The main types of digital cameras considered were those that were easily available including webcams, consumer still/video cameras, professional still/video cameras, industrial monitoring cameras and security cameras. Each requirement for the UGV’s was weighted in a decision matrix shown in Table B.3, so that the most important factors influenced the scores the most. As shown in the table, industrial monitoring cameras satisfied the criteria the best, followed by security cameras. The brief reasons for the requirement rating for each type of camera are discussed below.

IMAGE QUALITY

Image quality was given a high importance in the camera selection process, because the effectiveness of computer vision software is highly dependent on the clarity and resolution of the working image. The quality of an image produced by a digital camera depends on two main factors – resolution and pixel size. Resolution is the number of pixels in the image, which is determined by the number of light sensors on the pixel die of the camera. As the resolution grows, the image quality improves. The pixel size is the size of each pixel on the die. As the pixel size increases, the dynamic range and signal-to-noise ratio of each sensor increases, thus improving image quality (Chen et. al., 2000, p.1). Thus, an increase in the size of the camera die would improve resolution and pixel size, increasing image quality. This effectively cancels out the use of webcams, since their resolution and sensor quality is poor in comparison to the other four types of cameras. The remaining four types of
cameras score closely in image quality since they have relatively large dies, the professional cameras being of the best quality and size. A balance of image quality and cost was the major focus for selecting the camera.

**Cost**

Cost was of medium importance in the camera selection process, because it was understood that while it was desired to minimise cost used to purchase the cameras, it was also realised that some companies may sponsor in kind or dramatically reduce purchase prices. Poor quality cameras such as webcams and consumer still cameras sell for relatively cheap prices when compared to off-the-shelf professional, security or industrial monitoring cameras.

**Communication Connections**

It was understood quite early in the project that the most effective means of communication between UGV’s themselves, and between UGVs and the Ground Control Station was a wireless network connection. The majority of network switches use ethernet connections, so the presence of these became important. Some Ethernet switches also supported PoE plugs, which would benefit the UGV design by reducing the number of cables required. Other options were FireWire or USB for data transfer. Since most security and industrial monitoring cameras had a choice of two or three of these data transfer protocols, they were favoured in the scores. The other cameras were usually limited to USB data transfer only. As part of each score, a further consideration in communication connections was the robustness of the connections. Since the security and industrial monitoring cameras are designed for long durations of use, their connections were generally of higher quality and more durable than the other cameras in the selection.

**Reliability**

The reliability of the camera was important in the selection process, because of the significance of the object identification in the MAGIC competition with which the camera was initially selected.

---

### Table B.3: Decision matrix for camera selection

<table>
<thead>
<tr>
<th>Function</th>
<th>Webcam</th>
<th>Consumer Still/Video</th>
<th>Pro Still/Video</th>
<th>Industrial Monitoring</th>
<th>Security</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Quality</td>
<td>9</td>
<td>25</td>
<td>30</td>
<td>27</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>Cost</td>
<td>20</td>
<td>13</td>
<td>8</td>
<td>14</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Comm. conns</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Reliability</td>
<td>1</td>
<td>8</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Weight/ size</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Power supply</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>54</td>
<td>65</td>
<td>67</td>
<td>86</td>
<td>70</td>
<td>100</td>
</tr>
</tbody>
</table>
Camera reliability for this application is highly dependent on the vibration level that the camera is subjected to and the environmental conditions. Webcams, consumer and professional still and video cameras are not designed for vibration, or exposure to dust and moisture. Outdoor security cameras would cope with weather conditions such as rain and dust, but are generally designed for static applications. Hence, industrial monitoring cameras that are often mounted above vibrating conveyor belts and subjected to grime and light spray from fluids were ideal for the desired application.

**WEIGHT AND SIZE**

The weight and size of a camera are closely related, so these two requirements were grouped together. Since the camera was to be mounted on a swiveling pan-tilt unit, reduced size and weight were highly desirable. Decreased size means both a lower height for the UGV’s, reducing the complexity of the pan-tilt unit design and diminishing the chances of the camera being caught by a low branch. Decreased weight is desirable since this reduces the amount of mass that the pan-tilt unit motors need to move, and reduces the height of the centre of mass of the UGV, which in turn reduces the chances of rolling. Webcams and many consumer still and video cameras are generally light and small, subsequently scoring well in this requirement. Many industrial monitoring cameras are also small and light since they need to fit in confined spaces. On the other hand, security cameras are often bulky and comparatively heavy in order to be able to sustain weather.

**POWER SUPPLY**

Since a camera was to be mounted on each UGV, the power supply systems required needed to be convenient. In this case, the power required by the camera needed to be minimised to increase battery life, and the voltage required should be a common level used by other system parts if possible, to avoid having to accommodate the camera with extra power adjustments. It was desired that the number of wires be minimised in order to increase the mobility of the pan-tilt unit, and to improve connections convenience. All of these features were satisfied the best by the industrial monitoring and security cameras, especially the standard voltages and compatibility with PoE cables. PoE cables carry data as a normal Ethernet cable while also supplying power with one internal wire. However, this in turn means that the connecting Ethernet hub must also support PoE cables. Webcams, consumer and professional still and video cameras satisfied most of these desirable features, and often offered an alternative single power and data cable in the form of a USB cable. FireWire was another common camera data transfer cable type, but was not considered due to the lack of FireWire ports on the QNX box which had been acquired before the camera selection.

From what has been stated above it is clear that industrial monitoring cameras were the most ideal choice and hence were selected for this project.

### B.1.7 DIFFERENTIAL GLOBAL POSITIONING SYSTEM (DGPS)

A basic comparison of standard and differential GPS receivers is included in Table B.4. An example of a low cost GPS is like those available from Sparkfun (2010) and the Novatel unit chosen is considered Tactical Grade. There are two key differences between the systems. The first is the capability for differential GPS, which is only offered on Tactical Grade models. The second is that the tactical grade units are multiple channel units, which reduces the effect that interference can have on the unit but also improves the overall accuracy. These features were discussed in Section 2.3.1. The table shows that all options are comparable and so the Tactical Grade set up with a differential network was chosen as it offered the best accuracy.
As discussed in Section 2.3.1 a GPS unit is most useful for determining position in an earth fixed co-ordinate system. There are several possible co-ordinate systems that could be used, however the Universal Transverse Mercator (UTM) system of co-ordinates is suited to localisation applications. Using this system, position is returned in metres in Easterly and Northerly directions and these then require no transformation to be used with the localisation system. More details of the UTM co-ordinate frame can be found in publication by the National Geospational Intelligence Agency (2009). A more comprehensive discussion of various co-ordinate systems and frames can be found in Cullen et al. (2007 pp32-34) and Kaplan and Hegarty (2006, pp 26-34). Hence it was required supported returning data in UTM co-ordinates.

Table B.4: Decision matrix for dGPS selection

<table>
<thead>
<tr>
<th>Function</th>
<th>Tactical Grade</th>
<th>Budget</th>
<th>Tactical Grade without Differential</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>22</td>
<td>8</td>
<td>13</td>
<td>25</td>
</tr>
<tr>
<td>Speed</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Complexity</td>
<td>10</td>
<td>15</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>Serial Support</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Weight and size</td>
<td>6</td>
<td>10</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Power requirements</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Cost</td>
<td>4</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>67</td>
<td>55</td>
<td>65</td>
<td>100</td>
</tr>
</tbody>
</table>

B.1.8 INERTIAL MEASUREMENT UNIT (IMU)

There is a large range of IMU’s available, which vary greatly in cost and functionality. These range from those designed for hobbyist applications, through to expensive tactical grade units. One example is manufactured by SparkFun and is called the Razor IMU (SparkFun Electronics, 2010). The cost is only US$125 and it offers a 3 axis accelerometer and gyroscope. The board comes uncased and requires separate power regulation. This means that it is somewhat complicated to implement and will need some sort of casing added. A middle of the range unit is something like the Microstrain 3DM GX series chosen, which offers improved accuracy, through a faster sampling rate, slightly improved sensors with temperature compensation, a better developed filtering system and the addition of a magnetometer to measure the direction of the earth’s magnetic field. Hobby level IMUs are only able to determine heading through the integration of the gyroscope measurements. This is prone to drift and so can be used in conjunction with a magnetometer to account for the drift. The gyroscope has short term accuracy and a fast update speed and is then coupled with the magnetometer, which is more accurate in the long term but slow to update. The unit is also much more flexible in terms of the possible modes of operation, comes in a smaller form factor than the SparkFun and is reasonably weather proof but has a cost of approximately $2000. Moving up to a tactical grade unit will improve the performance another step. For example the iMAR iIMU-FSAS is a ruggedised unit, that can operate on a wide variety of input voltages, interfaces using several
different communications protocols and is designed to be very robust (IMAR, 2010). This type of unit can cost up to $100,000 and this price is unrealistic for the MAGIC project. The range of IMUs available offer many options in terms of possible configurations. These options are considered representative of the relative grades considered and so are compared in a decision matrix. From Table B.5 it can be seen that a mid range IMU is best suited for the purposes of the MAGIC project. The difficulty in fusing the data from the IMU with other sensors was not considered because there is no standard solution available for the range on sensors used on the MAGIC UGV. For this reason, there is a very limited advantage in choosing a unit like the Integrated IMU/GPS (eg Smiths Industries), discussed in Section 2.3.2.

Table B.5: Decision matrix for inertial sensor selection

<table>
<thead>
<tr>
<th>Function</th>
<th>Hobby</th>
<th>Mid Range</th>
<th>Tactical Grade</th>
<th>Integrated IMU/GPS Unit (See Section 2.3)</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Drift</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Range of available functions</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Durability</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>Weight, size</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Power Requirements</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Cost</td>
<td>10</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>55</td>
<td>72</td>
<td>63</td>
<td>59</td>
<td>100</td>
</tr>
</tbody>
</table>

From what has been stated previously it is clear that the Mid Range IMU was the best choice and hence was selected for this project.

B.1.9 MATERIAL TYPES AND SELECTION CRITERIA

Several types of material were investigated to determine the suitability. A number of material properties were identified as being important. Satisfying these properties would give the chassis the desired characteristics and functionality. The material properties were cost, specific strength, specific stiffness, and manufacturability. Some common chassis materials were evaluated with respect to these criteria. These criteria were each weighted in a decision matrix shown in Table B.6, so that the most important factors influenced the overall scores the most. The materials assessed were steel, aluminium, titanium and carbon fibre. As shown in table B.6, aluminium satisfied the criteria the best, followed by steel. The brief reasons for the material rating and requirement weighting are discussed below.
Table B.6: Decision matrix for material type

<table>
<thead>
<tr>
<th>Function</th>
<th>Steel</th>
<th>Aluminium</th>
<th>Titanium</th>
<th>Carbon Fibre</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>16</td>
<td>20</td>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Specific Strength</td>
<td>18</td>
<td>19</td>
<td>22</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Specific Stiffness</td>
<td>16</td>
<td>17</td>
<td>14</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Manufacturability</td>
<td>30</td>
<td>30</td>
<td>25</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>Total</td>
<td>80</td>
<td>86</td>
<td>62</td>
<td>62</td>
<td>100</td>
</tr>
</tbody>
</table>

Table B.7: Cost and density comparison of chassis materials (Callister 2007, Calm Aluminium Pty Ltd 2009)

<table>
<thead>
<tr>
<th>Material</th>
<th>Cost (US$/kg)</th>
<th>Density (kg/m^3)</th>
<th>Specific Cost (US$/m^3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steel (Aloy 4340)</td>
<td>0.5 - 0.6</td>
<td>7850</td>
<td>4710 - 3925</td>
</tr>
<tr>
<td>Aluminium (Alloy 6061)</td>
<td>6.10</td>
<td>2700</td>
<td>16470</td>
</tr>
<tr>
<td>Titanium (Alloy Ti-5Al-2.5V)</td>
<td>55.00 - 130.00</td>
<td>4480</td>
<td>246,400 - 582,400</td>
</tr>
<tr>
<td>Carbon Fibre</td>
<td>100.00 - 130.00</td>
<td>1700</td>
<td>170,000 - 221,000</td>
</tr>
</tbody>
</table>

**COST**

Owing to the fact that the project has a considerable budget the weighting value placed on cost was not as high as some of the other categories. However the budget is not unlimited so the cost must still be minimised where possible. In order to evaluate the respective material costs, the cost per kilogram of a material was considered. Table B.7 shows the cost per kilogram, density and cost per volume of each of the four materials considered.

It is evident from Table B.7 that steel is the least expensive material of the four considered, followed by aluminium, titanium and carbon fibre respectively on a per mass basis.

**SPECIFIC STRENGTH**

As there was a total weight restriction on the UGV, it was required that aspects of the UGV be as light as possible without sacrificing performance, hence this criterion carried a large weighting. The comparative strength property chosen was yield strength, as permanent plastic deformation of the chassis was to be avoided. Since carbon fibre composites do not undergo plastic deformation, tensile strength was used instead. Table B.8 shows yield strength and subsequent specific strength of each of the materials considered.

As there was a total weight restriction on the UGV, it was required that aspects of the UGV be as light as possible without sacrificing performance, hence this criterion carried a large weighting. Table B.8 shows yield strength and subsequent specific strength of each of the materials considered.
It can be seen from Table B.8 that carbon fibre has the highest specific strength, and hence receives the maximum value. Titanium, aluminium, and steel all have significantly smaller specific strengths and receive lower scores.

**MANUFACTURABILITY**

As the chassis had to be created in the University of Adelaide mechanical workshop, the manufacturability was considered purely from the capabilities of the workshop. Consequently this carried the largest weighting, for if the chassis could not be manufactured in the workshop then the chassis manufacture would have to be outsourced, which costs significantly more. The workshop contains a large variety of metalworking equipment and there are many staff with the expertise and experience in working with steel and aluminium, hence why both these materials received a high score. Titanium and receives a lower score as it is not commonly used in the workshop. Carbon fibre received the lowest score as there is less equipment and knowledge; it is also time consuming to work with.

**SPECIFIC STIFFNESS**

A high specific stiffness was required as excessive deflection of the chassis panels was to be minimised. This was because precision tracking equipment e.g. IMU were to be mounted inside the chassis and excessive deflection can cause errors. Also having a stiff chassis will reduce the chance of component failure due to vibration. Table B.9 shows Young’s modulus and subsequent specific stiffness of each of the materials considered.

From Table B.9 it can be seen that carbon fibre has a significantly higher specific stiffness than all the other materials evaluated. Titanium, aluminium, and steel all have virtually the same specific stiffness, which is much less, and hence receive lower scores.

Based on the stated selection criteria for the chassis material, aluminium proved to be the best option. The specific type of aluminium used was 5083 cast tooling plate. Series 5000 aluminium was chosen for its high strength and good corrosion resistance (Aluminium Metallurgy 2006). A cast aluminium plate was chosen for its high dimensional accuracy.
Table B.9: Specific stiffness of materials (Callister 2007, Calm Aluminium Pty Ltd 2009)

<table>
<thead>
<tr>
<th>Material</th>
<th>Yield Strength (MN/m²)</th>
<th>Density (kg/m³)</th>
<th>Specific Stiffness (Nm/kg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steel (1020)</td>
<td>207,000</td>
<td>7850</td>
<td>26.37</td>
</tr>
<tr>
<td>Aluminium (5083)</td>
<td>70,000</td>
<td>2660</td>
<td>26.32</td>
</tr>
<tr>
<td>Titanium (Ti-6Al-4V)</td>
<td>114,000</td>
<td>4480</td>
<td>25.45</td>
</tr>
<tr>
<td>Carbon Fibre Composite</td>
<td>220,000</td>
<td>1700</td>
<td>129.4</td>
</tr>
</tbody>
</table>

B.2 OBSOLETE HARDWARE SELECTION

This section contains the selection detail of two piece of hardware which were not used in the final UGV design. These two piece of hardware are the laser pointer, originally required in the MAGIC 2010 competition to neutralise static objects of interest (OOI), and the I/O board which was originally required for the LiDAR alarm, PTU limit switches and laser pointer all of which which are no longer being used.

B.2.1 LASER POINTER

The laser pointer was originally required to neutralise static OOI. The neutralisation of static OOI is accomplished by the OOI being located by a sensor UGV. After which a disruptor UGV must approach the OOI to within seven metres, but no closer than its detonation zone of two metres and target it with a laser pointer for a period of 30 seconds (Defence, Science & Technology Organisation, 2009).

According to the MAGIC2010 competition, the laser pointer could not be no greater than class II, hence eye safe (Defence, Science & Technology Organisation, 2009). It was decided by project partners Strategic Engineering that for the laser to be visible on a red background corresponding to the colour of static OOI then the laser should not be red. Hence this ruled out blue and red lasers since red would not show up clearly on a red background and blue lasers are not eye safe due the increased energy required at that end of the spectrum.

As shown in Table B.10 the chosen laser pointer is of the required class which is visible on a red background since it is a green laser. The laser chosen is a JLP series green laser pointer from Telescopes and Astronomy

As this project is no longer contributing to the MAGIC2010 competition, the need for a laser pointer is no longer a requirement. However, it was decided by the University of Adelaide MAGIC group that the laser pointer would not be removed, due to the fact that it was already purchased and implemented at the point when the competition was no longer a goal.

B.2.2 INPUT/OUTPUT BOARD

The input/output board (I/O board) was originally required to provide additional inputs and ports for various sensors and hardware and to then transfer this additional data to the Lex box.

The I/O board chosen was an Ocean Controls KTA-224 Modbus IO Module. This I/O board contains four opto-isolated digital inputs/outputs, eight relay input/outputs and three analog...
input/outputs. The I/O board communicates with the onboard computer via a USB connection. The chosen I/O board requires a 24V power source. The Ocean Controls KTA-224 Modbus IO Module can be seen in figure B.1. The minimum requirements for the I/O board is given in Table B.11.

![Ocean Controls KTA-224 Modbus IO Module](image)

Figure B.1: Ocean Controls KTA-224 Modbus IO Module

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Actual Specifications</th>
<th>Acceptable Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Digital I/Os</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Number of Relay I/Os</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Onboard Computer connection</td>
<td>USB</td>
<td>Serial, USB</td>
</tr>
<tr>
<td>Power Supply</td>
<td>12V</td>
<td>5V, 12V or 24V</td>
</tr>
</tbody>
</table>

The I/O board was required to support 3 devices with a digital input, this included LiDAR alarm and pan/tilt position sensors. The LiDAR alarm is used to control indicator lamps or relays which indicate the device status and is used for error diagnostics. The pan/tilt sensors are used to as a reset
value for determining the position of the PTU. The selected I/O board has a total of 4 digital inputs. Two relay inputs were also needed to actuate the laser pointer and the LiDAR restart command. The laser circuit was controlled using a relay to switch it on or off. The LiDAR restart command was used to reboot the LiDAR. This restart command is achieved by applying a +24VDC input to the LiDAR, which was achieved using a relay switch. The selected I/O board has a total of 8 relay inputs/outputs. The selected I/O board has more digital and relay outputs than required, to cater for possible additional hardware equipment that may need to be linked to the onboard computer in the future. The I/O board was required to have a compatible communication connection type with that of the onboard computer in order to be able to communicate with the onboard computer, the selected I/O board has USB communications which enable direct communication with the onboard computer. It was also important that the I/O board be powered with the commonly used voltage in the UGV, in order to avoid purchase of additional power management equipment. The selected I/O board requires a 12V power supply which can be provided by the UGV power management setup without the purchase of additional equipment.

B.3 PTU AND GCS OBSOLETE DESIGNS

B.3.1 PTU OPERATIONAL DESIGN NOT USED: STATIONARY POSITIONS

The LiDAR has a 190 degree viewing range. For this reason it is possible to view the surrounding area using only two pan angles, 180 degrees apart. In Figure B.2 an above view of the LiDAR sensor can be seen with the dotted red line showing its viewing range. It was proposed that the unit’s starting point be centred forward as indicated by the black arrow. From this point it could rotate 180 degrees in either direction (left or right). The blue arrows demonstrate the extent of movement in either direction of the laser module. Hence a 360 degree viewing range (due to a 20 degree overlap) would be achievable with only two stationary viewing positions.

This idea was decided against in favour of a more continuous range of motion giving increased flexibility to other aspects of the UGV such as tracking and continuous mapping using the camera (situated on top of the LiDAR sensor).

![Figure B.2: Top view of LiDAR sensor showing its viewing range and the proposed rotations](image-url)
B.3 PTU and GCS Obsolete Designs

B.3.2 Secondary Pan/Tilt Unit Design

This section details the design of the secondary PTU, which was produced in parallel with the primary design. This design was produced at a time when how the PTU would be manufactured was unknown. The primary design could not be manufactured at the Adelaide University hence a second, easier to manufacture design was required in the event that an outsourced manufacture could not be funded. This design sacrificed durability to reduce its manufacturing difficulty.

B.3.2.1 LiDAR Harness Design

The LiDAR harness joins the LiDAR to the tilt shaft. This design is required to be light-weight, occupy minimal volume, minimise vibration and be easily manufactured. Due to the geometry of the LiDAR sensor this design is reasonably restricted; the LiDAR can only be attached to the PTU structure via four 4mm holes on the rear of the LiDAR as can be seen in Figure B.3.

All designs consist of two parts; the LiDAR mount which is fastened to the LiDAR and the tilt shaft mount which is connected to the LiDAR mount and the tilt shaft. During the design process for this component many designs were brainstormed, modeled and compared to generate the final design. Owing to the minimal difference between a large number of the design iterations only the greatly differing designs will be discussed and compared.

LiDAR Harness Design One

LiDAR Harness Design One was formed early in the design phase; hence it is very complex and difficult to manufacture. The LiDAR mount design consists of two braces on either side of the LiDAR shaped to fit snugly onto contours of the LiDAR as shown in Figure B.4a. The tilt shaft mounts were not designed for this system before the LiDAR mount was deemed unusable.

LiDAR Harness Design Two

LiDAR Harness Design Two was produced in an effort to further cut down the manufacturing difficulty of the design. The LiDAR mount has been moved to the rear of the LiDAR and made completely flat with through holes to be bolted to the LiDAR sensor, see Figure B.4b. Also this design uses four slot holes rather than circle holes, as the torque required to tilt the LiDAR can be minimised by running the horizontal axis through its center of gravity. As the center of gravity of the LiDAR is not known the connection was made adjustable to locate it during testing. This design is considerably easier to manufacture and will also help to remove stress concentrators that are clear in the first design. The tilt shaft mount consists of a U-shaped hollow bar which wraps around the LiDAR mount. It is fastened to the LiDAR harness with two bolts via the slot-shaped
through hole. This was done to make the support adjustable for reasons previously mentioned. Two bolts are used to resist rotation and distribute force. This design can be seen in Figure B.4b.

**LiDAR Harness Design Three**

LiDAR Harness Design Three is an extension upon the second design. Instead of the two LiDAR mounts on either side of the LiDAR there is a single flat plate with the same slot-shaped through holes. The tilt shaft mount is similar to design two; however an additional component is attached to the LiDAR mount. The tilt shaft mounts are fastened to this with the same slot-shaped through holes (see Figure B.4c). This design is heavier, however it is easier to manufacture and allows for greater resistance to rotation about the join.

**LiDAR Harness Design Four**

LiDAR Harness Design Four was formed after the camera harness was changed. The LiDAR mount is similar to the second design however the unnecessary material between the holes has been removed. Also the top holes are connected to the camera mount and not the tilt section. This design is shown in Figure B.4d. This tilt shaft mount utilises the camera harness running along the top. It is fastened to both the camera harness and the LiDAR mount. Although this design is not adjustable, this is no longer of high importance due to the strength of the motors. This design will be manufactured using a laser cutter to save on time and manufacturing costs.

![Designs](image)

Figure B.4: Harness designs considered for mounting LiDAR sensor

Each of the requirements were weighted in a decision matrix (see Table B.12), where the most important factors influenced the scores the most. Design three was found to be the superior design. The reasons for the function ratings and requirement weights are below.

**Weight**
### Table B.12: Motor side tilt design decision matrix.

<table>
<thead>
<tr>
<th>Function</th>
<th>Two</th>
<th>Three</th>
<th>Four</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Volume Occupied</td>
<td>10</td>
<td>15</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>Vibration Resistance</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>Manufacturability</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Total</td>
<td>40</td>
<td>55</td>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>

The size of the components in this section are relatively small, hence its influence is minimal. Designs one and two use the same parts just spaced differently, however design three uses less parts of similar size hence the higher rating.

#### Design Width

Design width is a relatively important factor as this section has the greatest impact on the horizontal envelope. The designs get progressively thinner starting at one and ending at three, hence the rating values given.

#### Vibration Resistance

Vibration resistance is relatively important as any vibration in the structure will cause the results from both the camera and the LiDAR to lose accuracy. Also any vibration could harm the motors internal bearings. The shorter the shaft in the design the greater the vibration reduction hence design three is superior in this area.

#### Manufacturability

 Manufacturability is of similar significance as manufacture will be the chief cost for the PTU. All parts in these designs are rather simple to design, however design three requires less parts, hence it will be quicker and cheaper to make.

### B.3.2.2 Pan Design

The pan design is the interaction between the outer supports of the PTU and the base which the PTU is mounted on. This design must allow the motor to freely turn the PTU about the vertical shafts axis. It is important to keep vibration to a minimum and reduce the weight.

Two designs were produced and considered for this section. For both designs there is a lower support running between the two outer supports. This support is fastened to the bottom of the two outer supports to make a U-shaped structure which supports the tilt shafts and is supported by the pan shaft.

#### Pan Design One

Pan Design One has the motor mounted on top of the lower support and the bearing mounted below. The motor shaft is attached to another shaft via a coupling. This shaft passes through the bearing and into the base. The shaft is designed with three different diameters such that it will rest on the inside ring of the bearing and the base. The weight of the PTU applies a force where the shaft rests on the inside ring of the bearing and the base. The motor mount and bearing mounts are held in place via two and four bolts respectively. This design can be seen in Figure B.5a.
Pan Design Two

Pan Design Two has the motor and bearing mounted under the base. Similar to design one the motor shaft is attached to the pan shaft via a coupling. This shaft passes through the bearing and into the lower support. The shaft is designed with three different diameters such that it will rest on the inside ring of the bearing and the lower support. The PTU weight force is applied to the inside ring of the bearing and the lower support. The motor and bearing mounts are both fastened to the base via four bolts running through the corners of the three parts. This design allows the PTU to sit as close to the base as possible (reducing the length of the pan shaft) and the length of the outer supports to be reduced. This design can be seen in Figure B.5b.

A decision matrix is not required for this section as design two is superior to design one in every important area. Design two will have less vibration due to the shorter side supports and short pan shaft. Also due to the motor and bearing being mounted to the base rather than the PTU this reduces the weight. It also removes these items from visible sight, which is not a necessity but a bonus.

B.3.2.3 FINAL DESIGN

This PTU design has never finalised as it was unnecessary once it was decided that the other design (the primary design) was to be manufactured. The final point of this design can be seen in Figure B.6. This design includes all features that were chosen to be incorporated. Several sections were not considered in design choice due to being extremely trivial.

Bearings

The bearings for this design were chosen with a safety factor of 2 due to vibration forces, possible additional forces during the UGVs operation and weight approximations.

The bearings for the tilt shaft support the tilt section of the PTU which weights approximately 3.5kg. This load can be magnified by up to ten times during strong vibration, hence each bearing must support a radial load of up to 350N. Any axial loading will be due to vibration (minimal) or an external source. A NTN single row deep groove bearing with a dynamic basic load rating of 1,070N was chosen for the motor side. For the no-motor side a NTN micro ball bearing with a dynamic basic load rating of 830N was chosen.

The bearing for the pan shaft supports the section of the PTU above the chassis lid. The total weight of this is approximately 4.5kg. Due to vibration the bearing will be required to support a dynamic
axial load of up to 450N. The same bearing used on the motor side tilt shaft was used for the pan shaft.

**Couplings**

The coupling for the pan shaft is not going to be subjected to any high speeds due to the relatively slow turning speed of the PTU. The couplings only requirement is to be a 6mm to 6mm connector. A Duratech straight coupling was used.

### B.3.3 Initial GCS Designs

The initial designs of the GCS mechanical case were widely rejected due to manufacture difficulties or costs. The variations of the designs were predominantly in the ways in which the three monitors were positioned and mounted in the case as this was the driving factor in the design process of the GCS mechanical case.

#### B.3.3.1 Concept Design One

The first design utilised 7-flex adjustable monitor mounts that allowed the monitors to be compactly placed in the case (Refer to Figure B.7) or extended out during field use (Refer to Figure B.8).

The computer was placed upright on the bottom shelf with the ubiquiti board and dGPS unit. The three small wireless antennas may be placed at the back of one of the monitors, taped on the arm of the 7-flex. The dGPS antenna is manually taken out and placed outside of case during operation so that signal interference is prevented. At the rear of the case is a small drop down flap that enables a power cord to go through. During field use, the top lid is used as a seat while the front flap allows extra space to place peripherals (See figure B.9).

This design was rejected as consultation with manufacturers revealed that the front flap of the case will not be able to drop down as desired.
Following the consultation from manufacturers in Concept Design One, alternative options of placing the peripherals were required. The peripherals were not able to be placed on the flat surface of the top section as the walls will hinder the user during operation where the arms will need to outstretch in an awkward position. The height of these walls were required to be at a minimum of 150mm to provide sufficient strength for mounting of the 7-flex arms.

Further investigation of different mounting options for the 7-flex arms found that the 7-flex arms can be mounted through grommet holes on the top surface directly. Hence, the need for side walls can be removed and consequently the user will be able to use the top surface for peripherals freely (See Figure B.10). The removal of the side walls also allowed the dGPS antenna to be placed in the corner during operation as walls were no longer interfering with the signals.

This design was rejected as manufacturers were unable to build the case without compromising the strength of the top surface which needs to hold a minimum of 15kg weight from the monitors.
For manufacturability, the design had to revert back to the first concept design but with the front flap fixed instead. However, as discussed in the Concept Design Two section, this made positioning of the peripherals for ease of use difficult. To remedy this, the front was redesigned so that it
consisted of one entire front panel. This front panel is removed before GCS use and allows the user to access all components inside the case with ease. However, even though the peripherals may be placed on the top surface as previously suggested, this did not provide the best ergonomics and so a keyboard tray was incorporated to the design. This is shown in Figure B.11.

![Concept Design Three](image)

**Figure B.11: Concept Design Three**

Manufacturers quoted a price of approximately $3000 for the case with the size of the case design being the major contributor to cost. The three monitor arms came to a total of $750 and hence, the design was rejected as it was outside the budget.

**Table B.13: Weights of GCS components**

<table>
<thead>
<tr>
<th>Item</th>
<th>Weight (kg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
<td>17.7kg</td>
</tr>
<tr>
<td>dGPS unit and antenna</td>
<td>0.5kg</td>
</tr>
<tr>
<td>Wireless unit and antennas</td>
<td>0.5kg</td>
</tr>
<tr>
<td>Peripherals</td>
<td>1kg</td>
</tr>
<tr>
<td>Monitors</td>
<td>15kg</td>
</tr>
<tr>
<td>Monitor mount</td>
<td>20kg</td>
</tr>
<tr>
<td>GCS Case</td>
<td>20kg</td>
</tr>
<tr>
<td><strong>Total mass</strong></td>
<td><strong>74.7kg</strong></td>
</tr>
</tbody>
</table>
Table B.14: Costs of GCS components

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
<td>$2031.20</td>
</tr>
<tr>
<td>dGPS unit and antenna</td>
<td>$2975</td>
</tr>
<tr>
<td>Wireless unit and antennas</td>
<td>$180.18</td>
</tr>
<tr>
<td>Peripherals</td>
<td>$100</td>
</tr>
<tr>
<td>Monitors</td>
<td>$1950.50</td>
</tr>
<tr>
<td>Monitor mount</td>
<td>$626</td>
</tr>
<tr>
<td>GCS Case</td>
<td>$2197.46</td>
</tr>
<tr>
<td>Foam lining</td>
<td>$18.95</td>
</tr>
<tr>
<td>Total Cost</td>
<td>$10079.29</td>
</tr>
</tbody>
</table>
APPENDIX C : DRAWINGS

C.1 CHASSIS DRAWINGS

This section of the Appendices contains all manufacturing drawings of parts required for the construction of the Chassis. Only the latest revised versions are contained in this section.
All dimensions in mm unless otherwise stated

3rd Angle

Ben Quast 15/7/10

 MAGIC 1025

GPS mount

Aluminium

Version 4 Quantity 1 Scale 1:2 Size A4 Part Number 2 of 2
3 off \( \phi \) 3.4 Thru, CSK \( \phi \) 6.3 x 90°

All dimensions in mm unless otherwise stated

3rd Angle

General Tolerances

<table>
<thead>
<tr>
<th>Dim</th>
<th>Tol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5&lt;3</td>
<td>±0.1</td>
</tr>
<tr>
<td>3&lt;6</td>
<td>±0.1</td>
</tr>
<tr>
<td>6&lt;30</td>
<td>±0.2</td>
</tr>
<tr>
<td>30&lt;120</td>
<td>±0.3</td>
</tr>
<tr>
<td>120&lt;400</td>
<td>±0.5</td>
</tr>
<tr>
<td>400&lt;1000</td>
<td>±0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drawn</th>
<th>Date</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben Quast</td>
<td>29/9/10</td>
<td>Magic 1025</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checked</th>
<th>Date</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr. Michael Riese</td>
<td>30/9/10</td>
<td>Prismatic Shim</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authorised</th>
<th>Date</th>
<th>Material</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr. Steven Grainger</td>
<td>6/10/10</td>
<td>Aluminium</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Quantity</th>
<th>Scale</th>
<th>Size</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3 : 2</td>
<td>A4</td>
<td>1025-DWG-08</td>
</tr>
</tbody>
</table>
C.2 PTU DRAWINGS

This section of the Appendices contains all manufacturing drawings of parts required for the construction of the PTU. Only the latest revised versions are contained in this section.
Notes:
- All measurements are in MM
- All tolerances are +/- 0.1MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°

Designed by: R. BERTINATO

Checked by:  

Approved by: Date: Material: Stainless Steel Date: 26/05/2010

Title: Camera - Laser Shield Folded QTY: 6

File name:  

Revision Sheet 1 / 1
4 off ∅3.5 THRU

Notes:
- All measurements are in MM
- All tolerances are +/- 0.1MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°
Notes:

- All measurements are in MM
- All tolerances are +/- 0.1MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°

Designed by: R. Bertinato

Checked by: 

Approved by: 

Date: 11/05/2010

Material: Aluminium 6061 T6

QTY: 5

STRATEGIC ENGINEERING Pty Ltd

Robotics, Advanced Sensing & Automation

Unit 4, 25 Monro Avenue

Kirrawee, N.S.W. 2232

Ph: 1300 88 66 83

Fax: (02) 9521 5685

Richard@StrategicEng.com.au

Title: Elevation Pipe - Sheet 1

Revision A

Sheet 1/1
Material: Aluminium 6061 T6

Title: Motor Head Cap

Date: 12/05/2010

QTY: 5

Revision A

Sheet 1/1

Notes:
- All measurements are in MM
- All tolerances are +/- 0.1MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°

SCALE 3 : 2

SECTION A-A

0.25MM Chamfer

Φ 25

4 off M3 10
@ 14 PCD

Φ 35

Φ 6 15.5

2 off M3 12

A

A

3

12.5

31.5

6.5

0.25MM Chamfer

STRATEGIC ENGINEERING Pty Ltd
Robotics, Advanced Sensing & Automation
Unit 4, 25 Monro Avenue
Kirrawee, N.S.W. 2232
Ph: 1300 88 66 83
Fax: (02) 9521 5685
Richard@StrategicEng.com.au

Designed by: R. Bertinato

Checked by: 

Approved by: Date: 

File name: 

Revision A
Notes:
- All measurements are in MM
- All tolerances are +/- 0.1MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°

Material: Aluminium 6061 T6
Date: 11/05/2010
QTY: 6

Title: Pan Base - Sheet 1
File name:
Revision A
Sheet 1/1
All measurements are in MM
All tolerances are +/-0.1MM unless stated otherwise
All countersunks are recessed below the surface by 0.5MM
All countersunks are 90°

8 off M3 \( \varnothing 10 \) @ \( \varnothing 57 \) PCD

7 off M4 \( \varnothing 10 \) (8 hole spacing)

0.5MM Chamfer

0.2MM Undercut

SECTION C-C
SCALE 1:1

Material: Aluminium 6061 T6
Date: 11/05/2010

Title: Bearing Housing - Sheet 1
QTY: 5
Notes:
- All measurements are in MM
- All tolerances are +/- 0.1MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°

Designed by: R. Bertinato
Checked by: 
Approved by: 
Date: 10/05/2010
Material: Aluminium 6061 T6

Title: Pan Motor Mount
QTY: 5

File name: 
Revision: C
Sheet: 1/1

STRATEGIC ENGINEERING Pty Ltd
Robotics, Advanced Sensing & Automation
Unit 4, 25 Monro Avenue
Kirrawee, N.S.W. 2232
Ph: 1300 88 66 83
Fax: (02) 9521 5685
Richard@StrategicEng.com.au
Notes:
- All measurements are in MM
- All tolerances are +/- 0.1MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°

Designed by: R. Bertinato
Checked by: [Checked by]
Approved by: [Approved by] Date: [Date]
Material: Aluminium 6061 T6
Date: 10/05/2010
Title: SHAFT BASE PLATE
QTY: 5

STRATEGIC ENGINEERING Pty Ltd
Robotics, Advanced Sensing & Automation
Unit 4, 25 Monro Avenue
Kirrawee, N.S.W. 2232
Ph: 1300 88 66 83
Fax: (02) 9521 5685
Richard@StrategicEng.com.au

File name: [File name]
Revision: A Sheet: 1/1

All dimensions are in MM. All tolerances are +/- 0.1MM unless otherwise stated. All countersinks are recessed below the surface by 0.5MM and are 90°. M4 bolts are used throughout.
Notes:
- All measurements are in MM
- All tolerances are +/- 0.1MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Checked by:</th>
<th>Approved by:</th>
<th>Material:</th>
<th>Date:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R. BERTINATO</td>
<td></td>
<td></td>
<td>Stainless Steel</td>
<td>27/05/2010</td>
</tr>
</tbody>
</table>

STRATEGIC ENGINEERING Pty Ltd
Robotics, Advanced Sensing & Automation
Unit 4, 25 Monro Avenue
Kirrawee, N.S.W. 2232
Ph: 1300 88 66 83
Fax: (02) 9521 5685
Richard@Strategiceng.com.au

Title: Tilt Motor Cover - Folded
QTY: 6
All measurements are in MM
All tolerances are +/- 0.1MM unless stated otherwise
All countersunks are recessed below the surface by 0.5MM
All countersunks are 90°
C.3 COUPLING DRAWINGS

This section of the Appendices contains all manufacturing drawings of parts required for the construction of the couplings. Only the latest revised versions are contained in this section.
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES: +/- 0.1 mm
LINEAR:
ANGULAR:

Material: Aluminium 6061T6

Checked by: Richard Aplin

Title: Bearing Hub

Drawn by: J.K.

Date: 28/04/2010

SolidWorks Student License Academic Use Only
All measurements are in MM
All tolerances are +/- 0.1MM unless stated otherwise
All countersunks are recessed below the surface by 0.5MM
All countersunk are 90°

DESIGNER: R. Bertinato
CHECKED BY: Approved by: Date: 21/04/2010

Material: Aluminium 6061 T6

STRATEGIC ENGINEERING Pty Ltd
Robotics, Advanced Sensing & Automation
Unit 4, 25 Monro Avenue
Kirrawee, N.S.W. 2232
Ph: 1300 88 66 83
Fax: (02) 9521 5685
Richard@Strategiceng.com.au

Title: GearBox Mount
QTY: 6

File name:
Revision D
Sheet 1/1
M6, 18 deep

SECTION A-A

Material: Stainless Steel

Notes:
- All countersunks are recessed below the surface by 0.5mm
- All countersunks are 90°

All countersunks are 90° surface by 0.5mm

Stainless Steel

Material:

Weight:

Checked by: Richard Aplin

Title: Sleeve Shaft

Drawn by: J.K.

Date: 22/04/2010

SLEEVE SHAFT

M3 x 0.5

0.5mm Chamfer

H7 (+0.015)

0.5mm Chamfer

0.5mm Chamfer

0.5mm Chamfer

10

18

SECTION A-A

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES: +/- 0.1mm
LINEAR:
ANGULAR:

SolidWorks Student License
Academic Use Only
Notes:
- All measurements are in MM
- All tolerances are set to +/- 0.1 MM unless stated otherwise
- All countersunk are recessed below the surface by 0.5MM
- All countersunks are 90°

Material: Aluminium 6061T6

Title: Wheel Hub Inside Cap

QTY: 6

Revision D
Sheet 1/1
R. Bertinato

Wheel Hub Inside Cap

Aluminium 6061T6

All measurements are in MM

All tolerances are set to +/- 0.1 MM unless stated otherwise

All countersunks are recessed below the surface by 0.5MM

All countersunks are 90°

0.2MM Undercut

0.5MM Chamfer

5 off φ 3.5 CSK @ φ 32 PCD

0.5MM Chamfer

0.5MM Chamfer

28.2

Material:

Aluminium 6061T6

Date: 21/04/2010

Title: Wheel Hub Inside Cap

QTY: 6

Notes:

- All measurements are in MM
- All tolerances are set to +/- 0.1 MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5MM
- All countersunks are 90°
Notes:
- All measurements are in MM
- All tolerances are set to +/- 0.1 MM unless stated otherwise
- All countersunk are recessed below the surface by 0.5 MM
- All countersinks are 90°
5 off M3 THRU @ Ø 32 PCD

- All measurements are in MM
- All tolerances are set to +/- 0.1 MM unless stated otherwise
- All countersunks are recessed below the surface by 0.5mm
- All countersunks are 90°

Material: Aluminium 6061T6
Date: 22/04/2010
Title: Wheel Hub Spacer
QTY: 6
C.4 ELECTRICAL DRAWINGS
C.5 UGV ASSEMBLY GUIDE

This guide discusses how the components required are used to assemble a UGV in an efficient way that was determined from experience. This guide will be useful for future project teams that want to assemble additional UGVs. Wiring and cabling has been omitted since this can become extremely difficult to display in photographs. For this reason the wiring requirements can be found in the Electrical Drawings section and the cabling requirements can be determined from the data connections flowchart in the UGV Design chapter.

C.5.1 ON/OFF SWITCH SUBASSEMBLY

This section discusses how to assemble the On/Off Switch subassembly.

Parts required

- On/Off switch, complete with screws
- Top front panel
- 5mm Phillips head screwdriver
- Flat head screwdriver

Method

1. Unbolt the switch handle using the Phillips head screwdriver.

2. Now using the flat head screwdriver remove the grey part with on and off printed on it. The screws holding the top part of the on/off switch to the bottom part should now be visible.
3. Unscrew the four screws holding the top and bottom parts of the on/off switch

4. Locate the on/off switch on the front lid panel using the already drilled holes. Screw the four screws previously removed as shown to reconnect the top and bottom parts of the on/off switch.
5. Reconnect the piece with on/off printed on it and screw back on the switch handle.
C.5.2 WIRELESS UNIT SUBASSEMBLY

This section discusses how to assemble the wireless unit subassembly.

Parts required

- wireless unit
- 4 wireless unit feet
- wireless unit mounting bracket
- rear panel
- 2 M3 15mm hex socket countersunk bolts
- 4 M3 6mm hex socket head cap bolt
- Allen key set

Method

1. Attach the wireless unit to the wireless unit’s mounting bracket using the wireless unit feet and fasten the two parts together using the Allen key and the M3 6mm hex socket bolts as shown below.

2. The orientation of the wireless unit board is important so make sure that the orientation is the same as shown above.

3. Now fasten the mounting bracket assembly to the back panel using the M3 15mm countersunk bolts and the Allen key.
4. Attach the wireless unit antenna stabilising bracket to the rear panel.

5. Slide the wireless unit’s antennas through the holes in the stabilising bracket and fasten the wireless antennas to the wireless unit via the wireless unit’s cables.
C.5.3 DRIVE ASSEMBLY

This section discusses how to assemble the drive assembly. Essentially each drive assembly is the same and the process is just rinse and repeat for all six wheels.

Parts required (for just one drive assembly)
- key
- bearing hub
- sleeve shaft
- wheel motor
- wheel hub spacer
- gearbox mount
- wheel hub inside cap
- wheel hub outside cap
- wheel
- left or right side panel
- 4mm grub screw
- Allen key set

Method
1. Place the sleeve shaft over the motor shaft as shown below.

2. Fasten the sleeve shaft to the motor shaft using the 4mm grub screw with an Allen key set.
3. Put the bearing hub over the sleeve shaft with the side panel in between the bearing hub and the gearbox mount as shown below.
4. Slide the bearing hub along the sleeve shaft until the bearing hub and gearbox mount faces meet.

5. It is important that the plug for the motor cable is facing upwards as shown above before fastening the bearing hub and the gearbox mount together.

6. Fasten the bearing hub and the gearbox mount together with the M5 countersunk bolts using a 3mm Allen key.

7. Attach the key to the sleeve shaft as shown below.

8. Attach the wheel and fasten the wheel to the sleeve shaft with an Allen key as shown below. The wheel has a keyway slot where the key on the sleeve shaft will lock onto.

9. Repeat steps 1-7 for the other 5 drive assemblies. It may be helpful to jack the UGV frame up with some wood planks to make attaching the wheels easier.
C.5.4 E-STOP SUBASSEMBLY

Parts required
- E-stop button
- lid front panel
- Phillips head screwdriver

Method
1. Separate the E-stop by twisting the top part clockwise.

2. Locate the E-stop on the front top lid panel as shown below.

3. Now fasten the bolts shown below on each side of the E-stop to secure the E-stop in place.
C.5.5 LEX BOX SUBASSEMBLY

Parts required
- Lex box
- Lex box mounting brackets
- 4 M4 10mm countersunk bolts
- Allen key set

Method
1. Make sure that the connection ports are facing upwards and that the M4 bolt holes on the Lex box are facing the rear panel.
2. Fasten the LEX box to the Lex box mounting bracket with the M4 countersunk bolts and Allen key set.
3. Fasten the mounting bracket to the bottom panel with the M4 countersunk bolts and Allen key set.

C.5.6 IMU SUBASSEMBLY

Parts required
- IMU
- 2 M3 5mm countersunk bolts
- Allen key set

Method
1. Fasten the IMU to the top front panel using the countersunk bolts and an Allen key. The IMU should be fastened with the orientation shown below.
C.5.7 DC CONVERTER SUBASSEMBLY

Parts required

- 24V DC-DC converter
- 12V DC-DC converter
- 5V DC-DC converter
- 2 M4 6mm hex socket head cap bolts
- 14 M4 flat washer
- 4 M4 16mm hex socket head cap bolts
- 4 M4 4mm hex socket head cap bolts
- Allen Key set

Method

1. The orientation of the DC-DC converters is important and make sure that the orientation is as shown below before fastening them to the side panels.

2. Fasten the 12V DC-DC converter to the left side panel using the M4 16mm hex socket head cap bolts.

3. Fasten the 5V DC-DC converter to the right side panel using M4 4mm hex socket head cap screws and a Allen key set.

4. Fasten the 24V DC-DC converter to the right side panel using the M4 6mm hex socket head cap screws as shown below.
C.5.8 DIN RAIL SUBASSEMBLY

Parts required

- 2 23cm DIN rails
- Left side panel
- Right side panel
- Allen key set
- 4 port Ethernet switch
- 6 M3 6mm hex socket bolts

Method

1. Fasten a 23cm DIN rail to the left side panel using the M3 6mm hex socket bolts and the Allen key set as shown below.

2. Fasten the 23cm DIN rail to the right side panel using the countersunk bolts and the Allen key set as shown below.

3. Attach the connector blocks, circuit breakers and 4 port Ethernet switch as shown below. The connector blocks and 4 port Ethernet switch snap onto the DIN rail.
C.5.9 GPS ANTENNA SUBASSEMBLY

Parts required

- M16 50mm countersunk bolt
- GPS antenna
- GPS antenna mount
- 4 M5 15mm countersunk bolts
- top rear panel

Method

1. Attach the GPS antenna mount to the top rear panel with the M4 15mm countersunk bolts as shown below.

2. Fasten the M16 bolt to the GPS antenna.
### C.5.10 GPS UNIT SUBASSEMBLY

**Parts required**

- GPS unit
- GPS unit mounting bracket
- rear panel
- 2 M5 20mm bolts
- 2 M3 15mm countersunk bolts
- Allen key set

**Method**

1. Fasten the GPS unit to the GPS mounting bracket using the M5 20mm bolts with an Allen key as shown below.

![Image 1](image1.jpg)

2. Fasten the GPS bracket assembly to the back panel using the M3 15mm countersunk bolts and the Allen key set.

![Image 2](image2.jpg)
C.5.11 ULTRASONIC SENSOR SUBASSEMBLY

Parts required
- 2 ultrasonic sensors
- UGV front panel
- 2 28mm diameter fasteners

Method
1. Push the ultrasonics through the holes on the front panel and fasten them in place with the 28mm fasteners.
2. The holes are cut so that the ultrasonics can be aligned with 30 degrees of separation for a wider field of vision.
3. The ultrasonics should be orientated as shown below.

C.5.12 EPOS UNITS SUBASSEMBLY

Parts required
- 8 EPOS2 units
- 16 M3 5mm countersunk bolts
- Allen key set

Method
1. It is important to make sure that the EPOS cards are in the orientation shown below before fastening them to the front panel.
2. Fasten the 8 EPOS2 units to the front panel using the countersunk bolts and the Allen key set.
C.5.13 PAN TILT UNIT SUBASSEMBLY

**Parts Required**

- Tilt motor with cabling
- Worm gearbox
- 3 M3 5mm hex socket head cap bolts
- Allen key set
- 12 M3 5mm countersunk bolts
- PTU main bracket
- LiDAR
- left stabiliser
- left stabiliser bracket
- right stabiliser bracket
- 3 M5 16mm countersunk bolts
- 3 M4 15mm countersunk bolts
- 4 M5 70mm hex socket head cap bolts
- 4 M5 hexagon nut
- 4 M5 flat washers
- 4 M3 16mm countersunk bolts
- 4 M4 20mm countersunk bolts
- 8 M3 10mm countersunk bolts
- Camera with lens
- camera housing
- tilt motor housing

**Method**

1. Connect the cables for the tilt motor as shown below.
2. Fasten the tilt motor housing to the main bracket using the M3 5mm countersunk bolts and an Allen key. Make sure the key is keyway shown below.

3. Now fasten the three M5 16mm countersunk bolts and an Allen key as shown below.

4. Now slot the right stabiliser bracket onto the tilt motor housing and attach the LiDAR right stabiliser bracket as shown below.
5. Now attach the left stabiliser bracket to the LiDAR and fasten the two stabiliser brackets to the LiDAR using the M5 70mm hex socket head cap bolts.

6. Fasten the left stabiliser to the pan tilt unit main bracket using the three M4 15mm countersunk bolts and an Allen key.
7. Attach the camera harness to the camera using the M3 10mm countersunk bolts as shown below.

8. Attach the camera harness to the two stabiliser brackets with the M3 16mm countersunk bolts as shown below.
9. Attach the current assembly to the Pan motor housing using the M4 20mm countersunk bolts as shown below.
C.5.14  OVERALL ASSEMBLY PROCEDURE

Please refer to the individual sections for each subassembly if more information is required. It would be advantageous to have each subassembly fully assembled before going through this section of the assembly guide. The chassis screws are mostly M3 15mm hex socket countersunk bolts with the only exception which is the top rear panel which uses hex socket head cap bolts.

1. Attach the side panels to the bottom panels as shown below. The orientation of the side panels is important.

2. Attach the drive assembly to the side panels as shown below. Looking at the photo above the front end is the end further away.

3. Attach the EPOS subassembly to the front panel and then fasten the front panel to the side panels.

4. Attach the ultrasonic subassembly to the front panel.

5. Attach the DIN rail subassembly to the side panels.

6. Attach the DC converter subassembly to the side panels.

7. Attach the rear panel to the side panels.

8. Attach the GPS unit subassembly to the rear panel.

9. Attach the wireless unit subassembly to the rear panel.

10. Attach the Lex box subassembly to the bottom panel.

11. Attach the external USB and Ethernet plug to the rear panel as shown below.
12. Attach the Pan Tilt Unit subassembly to the top front panel using M4 12mm countersunk bolts.
13. Attach the E-stop subassembly to the top front panel.
14. Attach the On/Off switch to the top front panel.
15. Attach the top front panel to the side and front panel.
16. Place the batteries resting sideways at the rear of the UGV in between the GPS mounting bracket and the wireless unit’s mounting bracket.
17. Attach the GPS antenna subassembly to the top rear panel then attach the top rear panel to the side panels and rear panel.
APPENDIX D : DATASHEETS

This part of the Appendice contains relevant datasheets.

D.1 UGV DRIVE SYSTEM
D.1.1 UGV DRIVE MOTOR
D.1.2 UGV DRIVE MOTOR ENCODER
D.1.3 UGV DRIVE MOTOR GEARBOX
### Motor Data

<table>
<thead>
<tr>
<th>Values at nominal voltage</th>
<th>329285</th>
<th>251601</th>
<th>329286</th>
<th>329287</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Nominal voltage (V)</td>
<td>18.0</td>
<td>24.0</td>
<td>24.0</td>
<td>36.0</td>
</tr>
<tr>
<td>2 No load speed (rpm)</td>
<td>6710</td>
<td>6700</td>
<td>4700</td>
<td>3360</td>
</tr>
<tr>
<td>3 No load current (mA)</td>
<td>294</td>
<td>201</td>
<td>116</td>
<td>46.9</td>
</tr>
<tr>
<td>4 Nominal speed (rpm)</td>
<td>5250</td>
<td>5260</td>
<td>3410</td>
<td>2320</td>
</tr>
<tr>
<td>5 Nominal torque (mNm)</td>
<td>96.9</td>
<td>84.3</td>
<td>64.2</td>
<td>94.3</td>
</tr>
<tr>
<td>6 Nominal current (mA)</td>
<td>3.54</td>
<td>3.36</td>
<td>1.32</td>
<td>0.861</td>
</tr>
<tr>
<td>7 Stall torque (mNm)</td>
<td>1100</td>
<td>822</td>
<td>337</td>
<td>488</td>
</tr>
<tr>
<td>8 Starting current (A)</td>
<td>43.6</td>
<td>24.5</td>
<td>7.10</td>
<td>4.86</td>
</tr>
<tr>
<td>9 Max. efficiency (%)</td>
<td>84.5</td>
<td>83</td>
<td>77</td>
<td>82</td>
</tr>
</tbody>
</table>

### Characteristics

- Terminal resistance phase to phase (Ω): 0.413, 0.978, 3.38, 7.41
- Terminal inductance phase to phase (mH): 0.322, 0.573, 1.15, 5.15
- Torque constant (mNm/A): 25.1, 33.5, 47.5, 101
- Speed constant (rpm/V): 380, 285, 201, 95.0
- Speed / Torque gradient (rpm/mNm): 6.24, 8.32, 14.3, 7.00
- Mechanical time constant (ms): 8.82, 11.8, 20.3, 9.90
- Rotor inertia (gcm²): 135, 135, 135, 135

### Specifications

#### Thermal data
- Thermal resistance housing-ambient: 4.25 K/W
- Thermal resistance winding-housing: 4.5 K/W
- Thermal time constant motor: 16.6 s
- Ambient temperature: -40°C to +120°C

#### Mechanical data (preloaded ball bearings)
- Max. permissible winding temperature: +120°C

### Operating Range

<table>
<thead>
<tr>
<th>rpm</th>
<th>329285</th>
<th>251601</th>
<th>329286</th>
<th>329287</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>2500</td>
<td>2500</td>
<td>2500</td>
<td>2500</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### Other specifications
- Number of pole pairs: 6
- Number of phases: 3
- Weight of motor: 110 g

### Connection
- Pin 1: Hall sensor 1
- Pin 2: Hall sensor 2
- Pin 3: 4.5...18 VDC
- Pin 4: Motor winding 3
- Pin 5: Hall sensor 3
- Pin 6: GND
- Pin 7: Motor winding 1
- Pin 8: Motor winding 2
- Internal pull-up (7...13 kΩ) on pin 3
- Wiring diagram for Hall sensors see page 29

### Cable
- Connection cable Universal, L = 500 mm: 339380
- Connection cable to EPOS, L = 500 mm: 354045

---

**Order Number**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>329285</td>
<td>3260619</td>
</tr>
<tr>
<td>251601</td>
<td>8407235</td>
</tr>
<tr>
<td>329286</td>
<td>3360619</td>
</tr>
<tr>
<td>329287</td>
<td>3360620</td>
</tr>
</tbody>
</table>

---

**Continuous operation**

In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.

#### Assigned power rating

- Short term operation
  - The motor may be briefly overloaded (recurring).

---

**maxon Modular System**

### Planetary Gearhead

- Ø45 mm
- 3 - 15 Nm
- Speed: 1:1

### Spur Gearhead

- Ø45 mm
- 0.5 - 2.0 Nm

---

**Recommended Electronics**

- DECS 50/5
- Page 244
- DECS 24/3
- DECV 50/5
- DECV 24/3
- EPOS 24/5
- EPOS P 24/5

---

**Notes**

- 20
**Encoder MR, Type L, 256 - 1024 CPT, 3 Channels, with Line Driver**

**Order Number**

<table>
<thead>
<tr>
<th>Encoder Type</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR 256</td>
<td>225783</td>
</tr>
<tr>
<td>MR 500</td>
<td>228452</td>
</tr>
<tr>
<td>MR 1024</td>
<td>225785</td>
</tr>
<tr>
<td>MR 2000</td>
<td>228456</td>
</tr>
<tr>
<td>MR 3200</td>
<td>225787</td>
</tr>
</tbody>
</table>

**Type**

<table>
<thead>
<tr>
<th>Counts per turn</th>
<th>Number of channels</th>
<th>Max. speed (rpm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 - 1024</td>
<td>3</td>
<td>80 - 18750</td>
</tr>
</tbody>
</table>

**maxon Modular System**

<table>
<thead>
<tr>
<th>Motor</th>
<th>Page</th>
<th>Gearhead</th>
<th>Page</th>
<th>Brake</th>
<th>Page</th>
<th>Overall length [mm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE 30, 60 W</td>
<td>80</td>
<td>GP 32, 0.75 - 6.0 Nm</td>
<td>232/233</td>
<td></td>
<td></td>
<td>79.4</td>
</tr>
<tr>
<td>RE 30, 60 W</td>
<td>80</td>
<td>GP 32 S</td>
<td>249-251</td>
<td></td>
<td></td>
<td>79.4</td>
</tr>
<tr>
<td>RE 30, 60 W</td>
<td>80</td>
<td>GP 32, 0.75 - 4.5 Nm</td>
<td>230</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>RE 35, 90 W</td>
<td>81</td>
<td>GP 32, 0.75 - 4.5 Nm</td>
<td>230</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>RE 35, 90 W</td>
<td>81</td>
<td>GP 32, 0.75 - 6.0 Nm</td>
<td>232/233</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>RE 35, 90 W</td>
<td>81</td>
<td>GP 32, 4.0 - 8.0 Nm</td>
<td>235</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>RE 35, 90 W</td>
<td>81</td>
<td>GP 42, 3 - 15 Nm</td>
<td>238</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>RE 35, 90 W</td>
<td>81</td>
<td>GP 32 S</td>
<td>249-251</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>RE 40, 150 W</td>
<td>82</td>
<td>GP 42, 3 - 15 Nm</td>
<td>238</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>RE 40, 150 W</td>
<td>82</td>
<td>GP 52, 4 - 30 Nm</td>
<td>241</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>A-max 32</td>
<td>110/112</td>
<td>GP 32, 0.75 - 6.0 Nm</td>
<td>232/234</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>A-max 32</td>
<td>110/112</td>
<td>GS 38, 0.1 - 0.6 Nm</td>
<td>237</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>A-max 32</td>
<td>110/112</td>
<td>GP 32 S</td>
<td>249-251</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>EC-max 40, 70 W</td>
<td>168</td>
<td>GP 42, 3 - 15 Nm</td>
<td>239</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>EC-max 40, 70 W</td>
<td>168</td>
<td>GP 52, 4 - 30 Nm</td>
<td>242</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>EC-i 40, 50 W</td>
<td>190</td>
<td>GP 32, 1 - 6 Nm</td>
<td>234</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>EC-i 40, 50 W</td>
<td>190</td>
<td>GP 32 S</td>
<td>249-251</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>EC-i 40, 70 W</td>
<td>191</td>
<td>GP 32, 1 - 6 Nm</td>
<td>234</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
<tr>
<td>EC-i 40, 70 W</td>
<td>191</td>
<td>GP 32 S</td>
<td>249-251</td>
<td></td>
<td></td>
<td>82.4</td>
</tr>
</tbody>
</table>

**Technical Data**

- Supply voltage: 5 V ± 5%
- Output signal: TTL compatible
- Phase shift: 90° ± 45°
- Index pulse width: 90° ± 45°
- Operating temperature range: -25°C to +85°C
- Moment of inertia of code wheel: 1.7 gcm²
- Output current per channel: max. 5 mA
- Terminal resistance R: typical 120 Ω
- Capacitor C: max. 0.1 nF per m line length

The index signal I is synchronised with channel A or B.

May 2010 edition / subject to change
Spur Gearhead GS 45 A Ø45 mm, 0.5 - 2.0 Nm

Technical Data
Spur Gearhead straight teeth
Output shaft stainless steel, hardened
Bearing at output ball bearing
Radial play, 10 mm from flange max. 0.15 mm
Axial play 0.02 - 0.2 mm
Max. radial load, 10 mm from flange 180 N
Max. permissible axial load 60 N
Max. permissible force for press fits 60 N
Recommended input speed < 6000 rpm
Recommended temperature range -15 ... +80°C
Extended range as option -40 ... +100°C

Order Number
301177 301178 301181 301186 301191

Order Number
301179 301180 301182 301187 301192

Order Number
301189 301190 301193 301194

Gearhead Data
1 Reduction 5:1 18:1 212:1 732:1
2 Reduction absolute 1/11 1/111 1/1111 1/11111
10 Mass inertia gcm² 3.7 1.6 1.0 0.8 0.8
3 Max. motor shaft diameter mm 3 3 3 3 3

Order Number
301177 301178 301181 301186 301191

Order Number
301179 301180 301182 301187 301192

Order Number
301189 301190 301193 301194

Gearhead Data
1 Reduction 7:1 26:1 89:1 310:1 1072:1
2 Reduction absolute 203/28 1805/236 6662/245 18248/252 50375/252
10 Mass inertia gcm² 3.1 1.4 1.0 0.8 0.8
3 Max. motor shaft diameter mm 3 3 3 3 3

Order Number
301179 301180 301182 301187 301192

Order Number
301189 301190 301193 301194

Gearhead Data
1 Reduction 14:1 47:1 163:1 564:1 1952:1
2 Reduction absolute 247/123 6227/402 141115/601 161890/673 1929023/287
10 Mass inertia gcm² 2.1 1.4 0.6 0.5 0.4
3 Max. motor shaft diameter mm 3 3 3 3 3

Order Number
301180 301181 301185 301189 301194

Technical Data
Spur Gearhead Output shaft stainless steel, hardened
Bearing at output ball bearing
Radial play, 10 mm from flange max. 0.15 mm
Axial play 0.02 - 0.2 mm
Max. radial load, 10 mm from flange 180 N
Max. permissible axial load 60 N
Max. permissible force for press fits 60 N
Recommended input speed < 6000 rpm
Recommended temperature range -15 ... +80°C
Extended range as option -40 ... +100°C

Spur Gearhead GS 45 A Ø45 mm, 0.5 - 2.0 Nm

Technical Data
Spur Gearhead Output shaft stainless steel, hardened
Bearing at output ball bearing
Radial play, 10 mm from flange max. 0.15 mm
Axial play 0.02 - 0.2 mm
Max. radial load, 10 mm from flange 180 N
Max. permissible axial load 60 N
Max. permissible force for press fits 60 N
Recommended input speed < 6000 rpm
Recommended temperature range -15 ... +80°C
Extended range as option -40 ... +100°C

Spur Gearhead GS 45 A Ø45 mm, 0.5 - 2.0 Nm

Technical Data
Spur Gearhead Output shaft stainless steel, hardened
Bearing at output ball bearing
Radial play, 10 mm from flange max. 0.15 mm
Axial play 0.02 - 0.2 mm
Max. radial load, 10 mm from flange 180 N
Max. permissible axial load 60 N
Max. permissible force for press fits 60 N
Recommended input speed < 6000 rpm
Recommended temperature range -15 ... +80°C
Extended range as option -40 ... +100°C

Spur Gearhead GS 45 A Ø45 mm, 0.5 - 2.0 Nm

Technical Data
Spur Gearhead Output shaft stainless steel, hardened
Bearing at output ball bearing
Radial play, 10 mm from flange max. 0.15 mm
Axial play 0.02 - 0.2 mm
Max. radial load, 10 mm from flange 180 N
Max. permissible axial load 60 N
Max. permissible force for press fits 60 N
Recommended input speed < 6000 rpm
Recommended temperature range -15 ... +80°C
Extended range as option -40 ... +100°C

Spur Gearhead GS 45 A Ø45 mm, 0.5 - 2.0 Nm

Technical Data
Spur Gearhead Output shaft stainless steel, hardened
Bearing at output ball bearing
Radial play, 10 mm from flange max. 0.15 mm
Axial play 0.02 - 0.2 mm
Max. radial load, 10 mm from flange 180 N
Max. permissible axial load 60 N
Max. permissible force for press fits 60 N
Recommended input speed < 6000 rpm
Recommended temperature range -15 ... +80°C
Extended range as option -40 ... +100°C
D.2 UGV PTU TILT SYSTEM

D.2.1 TILT MOTOR

D.2.2 TILT MOTOR PLANETARY GEARHEAD

D.2.3 TILT MOTOR WORM GEARHEAD

D.2.4 TILT MOTOR ENCODER
## EC 32 flat Ø32 mm, brushless, 15 Watt

<table>
<thead>
<tr>
<th>Values at nominal voltage</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Nominal voltage V</td>
<td>9.0</td>
<td>12.0</td>
<td>24.0</td>
<td>48.0</td>
</tr>
<tr>
<td>2 No load speed rpm</td>
<td>3590</td>
<td>4460</td>
<td>4390</td>
<td>4630</td>
</tr>
<tr>
<td>3 No load current mA</td>
<td>152</td>
<td>150</td>
<td>73.4</td>
<td>39.4</td>
</tr>
<tr>
<td>4 Nominal speed rpm</td>
<td>2120</td>
<td>2850</td>
<td>2800</td>
<td>2980</td>
</tr>
<tr>
<td>5 Nominal torque (max. continuous torque) mNm</td>
<td>22.5</td>
<td>22.8</td>
<td>23.3</td>
<td>22.4</td>
</tr>
<tr>
<td>6 Nominal current (max. continuous current) A</td>
<td>1.05</td>
<td>0.981</td>
<td>0.490</td>
<td>0.251</td>
</tr>
<tr>
<td>7 Stall torque mNm</td>
<td>70.0</td>
<td>84.1</td>
<td>85.8</td>
<td>84.1</td>
</tr>
<tr>
<td>8 Starting current A</td>
<td>3.13</td>
<td>3.49</td>
<td>1.75</td>
<td>0.906</td>
</tr>
<tr>
<td>9 Max. efficiency %</td>
<td>61</td>
<td>63</td>
<td>64</td>
<td>63</td>
</tr>
</tbody>
</table>

### Characteristics
- **10 Terminal resistance phase to phase Ω:** 3.87, 3.43, 13.7, 53.0
- **11 Terminal inductance phase to phase mH:** 1.61, 1.87, 7.73, 27.8
- **12 Torque constant mNm / A:** 22.4, 24.1, 49.0, 92.8
- **13 Speed constant rpm / V:** 427, 397, 195, 103
- **14 Speed / torque gradient rpm / mNm:** 54.9, 56.6, 54.5, 58.7
- **15 Mechanical time constant ms:** 20.1, 20.7, 20.0, 21.5
- **16 Rotor inertia gcm²:** 35.0, 35.0, 35.0, 35.0

### Mechanical data (preloaded ball bearings)
- **23 Max. permissible speed 1000 rpm**
- **24 Axial play at axial load < 5.0 N:** 0 mm typ., 0.6 mm
- **25 Radial play preloaded**
- **26 Max. axial load (dynamic):** 4.8 N
- **27 Max. force for press fits (static):** 50 N (static, shaft supported)
- **28 Max. radial loading, 7.5 mm from flange:** 5.5 N

### Other specifications
- **29 Number of pole pairs:**
- **30 Number of phases:** 3
- **31 Weight of motor:** 46 g

### Connection with Hall sensors
- **Pin 1:** 3.5 ... 24 VDC Motor winding 1
- **Pin 2:** Hall sensor 3 Motor winding 2
- **Pin 3:** Hall sensor 1 Motor winding 3
- **Pin 4:** Hall sensor 2 neutral point
- **Pin 5:** GND
- **Pin 6:** Motor winding 3
- **Pin 7:** Motor winding 2
- **Pin 8:** Motor winding 1

### Adapter
- **Order number:** see p. 310 220000

### Recommended Electronics:
- **DEC 50/5** Page 289
- **DEC 24/1** Page 290
- **DEC Module 24/2** Page 291
- **DEC Module 50/5** Page 291
- **DEC Module 50/5** Page 297
- **EPOS2 Module 36/2** Page 304
- **EPOS 24/1** Page 304

### Notes
- **20**

---

**Continuous operation**

In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.

**Thermal limit.**

**Short term operation**

The motor may be briefly overloaded (recurring).

**Assigned power rating**

---

**maxon Modular System**

Planetary Gearhead Ø32 mm 0.75 - 6 Nm Page 232 / 234

Spur Gearhead Ø38 mm 0.1 - 0.8 Nm Page 237

---

**maxon flat motor**

May 2010 edition / subject to change
Planetary Gearhead GP 32 A Ø32 mm, 0.75 - 4.5 Nm

Technical Data
Planetary Gearhead
Output shaft: stainless steel
Shaf diameter as option: 8 mm
Bearings at output: ball bearing
Radial play, 5 mm from flange: max. 0.14 mm
Axial play: max. 0.4 mm
Max. radial load, 10 mm from flange: 140 N
Max. permissible axial load: 120 N
Max. permissible force for press fits: 120 N
Sense of rotation, drive to output: < 6000 rpm
Recommended input speed: -40 ... +100°C

Gearhead Data

<table>
<thead>
<tr>
<th>Order Number</th>
<th>166155</th>
<th>166158</th>
<th>166163</th>
<th>166164</th>
<th>166169</th>
<th>166174</th>
<th>166179</th>
<th>166184</th>
<th>166187</th>
<th>166192</th>
<th>166197</th>
<th>166202</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduction</td>
<td>3:7:1</td>
<td>14:1</td>
<td>33:1</td>
<td>51:1</td>
<td>111:1</td>
<td>246:1</td>
<td>492:1</td>
<td>762:1</td>
<td>118:1</td>
<td>197:2</td>
<td>282:9</td>
<td>4380:1</td>
</tr>
<tr>
<td>Max. motor shaft diameter</td>
<td>mm</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Order Number</td>
<td>166156</td>
<td>166159</td>
<td>166165</td>
<td>166168</td>
<td>166173</td>
<td>166176</td>
<td>166181</td>
<td>166186</td>
<td>166191</td>
<td>166196</td>
<td>166201</td>
<td>166206</td>
</tr>
<tr>
<td>Reduction</td>
<td>4:8:1</td>
<td>18:1</td>
<td>68:1</td>
<td>123:1</td>
<td>295:1</td>
<td>537:1</td>
<td>913:1</td>
<td>1414:1</td>
<td>2189:1</td>
<td>3052:1</td>
<td>4254:1</td>
<td></td>
</tr>
<tr>
<td>Max. motor shaft diameter</td>
<td>mm</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Order Number</td>
<td>166157</td>
<td>166160</td>
<td>166165</td>
<td>166170</td>
<td>166173</td>
<td>166176</td>
<td>166181</td>
<td>166186</td>
<td>166191</td>
<td>166196</td>
<td>166201</td>
<td>166206</td>
</tr>
<tr>
<td>Reduction</td>
<td>5:8:1</td>
<td>21:1</td>
<td>79:1</td>
<td>132:1</td>
<td>318:1</td>
<td>589:1</td>
<td>1093:1</td>
<td>1526:1</td>
<td>2362:1</td>
<td>3389:1</td>
<td>6258:1</td>
<td></td>
</tr>
<tr>
<td>Reduction absolute</td>
<td></td>
<td>0.005</td>
<td>0.005</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>Max. motor shaft diameter</td>
<td>mm</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Order Number</td>
<td>166161</td>
<td>166162</td>
<td>166165</td>
<td>166168</td>
<td>166173</td>
<td>166176</td>
<td>166181</td>
<td>166186</td>
<td>166191</td>
<td>166196</td>
<td>166201</td>
<td>166206</td>
</tr>
<tr>
<td>Max. motor shaft diameter</td>
<td>mm</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Planetary Gearhead straight teeth

Order Number

Planetary Gearhead
Output shaft: stainless steel
Shaf diameter as option: 8 mm
Bearings at output: ball bearing
Radial play, 5 mm from flange: max. 0.14 mm
Axial play: max. 0.4 mm
Max. radial load, 10 mm from flange: 140 N
Max. permissible axial load: 120 N
Max. permissible force for press fits: 120 N
Sense of rotation, drive to output: < 6000 rpm
Recommended input speed: -40 ... +100°C

Max. continuous torque Nm

Max. efficiency %

Weight g

Average backlash no load °

Mass inertia gcm²

Gearhead length L1 mm

Gearhead length L2 mm

Overall length [mm] = Motor length + gearhead length + (sensor / brake) + assembly parts

M:1:2

Option: Low-noise version

Max. permissible force for press fits: 120 N

Special program (on request)

Overall length [mm]

Overall length [mm]

Planetary Gearhead straight teeth

Output shaft: stainless steel
Shaf diameter as option: 8 mm
Bearings at output: ball bearing
Radial play, 5 mm from flange: max. 0.14 mm
Axial play: max. 0.4 mm
Max. radial load, 10 mm from flange: 140 N
Max. permissible axial load: 120 N
Max. permissible force for press fits: 120 N
Sense of rotation, drive to output: < 6000 rpm
Recommended input speed: -40 ... +100°C

May 2010 edition / subject to change
**Technical and performance data:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gear unit weight</td>
<td>kg</td>
<td>0.620</td>
</tr>
<tr>
<td>Dimension L min.</td>
<td>mm</td>
<td>16</td>
</tr>
<tr>
<td>Dimension L max.</td>
<td>mm</td>
<td>32</td>
</tr>
<tr>
<td>Dimension l</td>
<td>mm</td>
<td>5</td>
</tr>
<tr>
<td>Ø d standard mount</td>
<td>mm</td>
<td>6h6</td>
</tr>
<tr>
<td>Lk Ø max.</td>
<td>mm</td>
<td>32</td>
</tr>
<tr>
<td>Centre distance</td>
<td>mm</td>
<td>25</td>
</tr>
<tr>
<td>Average mass inertia at drive</td>
<td>kg cm²</td>
<td>0.060</td>
</tr>
<tr>
<td>No-load backlash Type Standard</td>
<td>arcmin</td>
<td>30</td>
</tr>
<tr>
<td>No-load backlash Type R Reduced</td>
<td>arcmin</td>
<td>15</td>
</tr>
<tr>
<td>Self-locking whenever step-down ratio</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>Max. radial load, poor fastening</td>
<td>N</td>
<td>150</td>
</tr>
<tr>
<td>Max. axial load</td>
<td>N</td>
<td>200</td>
</tr>
<tr>
<td>Torsional stiffness</td>
<td>Nm/arcmin</td>
<td>1.3</td>
</tr>
<tr>
<td>Rated speed at drive</td>
<td>min⁻¹</td>
<td>4'000</td>
</tr>
<tr>
<td>Max. rated speed at drive</td>
<td>min⁻¹</td>
<td>6'000</td>
</tr>
<tr>
<td>Temperature range</td>
<td>°C</td>
<td>-30° / +90°</td>
</tr>
<tr>
<td>Protection class</td>
<td>IP</td>
<td>44</td>
</tr>
<tr>
<td>Service life</td>
<td>h</td>
<td>~ 5000</td>
</tr>
</tbody>
</table>

Stock programme • Standard programme • Special programme (on request)
# Encoder MR, Type L, 256 - 1024 CPT, 3 Channels, with Line Driver

## Technical Data

- **Supply voltage**: 5 V ± 5%
- **Output signal**: TTL compatible
- **Phase shift φ**: 90° ± 45°
- **Operating temperature rang**: -25...+85°C
- **Moment of inertia of code wheel**: 1.7 g cm²
- **Output current per channel**: max. 5 mA

## Pin Allocation

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N.C.</td>
</tr>
<tr>
<td>2</td>
<td>VCC</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
</tr>
<tr>
<td>4</td>
<td>N.C.</td>
</tr>
<tr>
<td>5</td>
<td>Channel A</td>
</tr>
<tr>
<td>6</td>
<td>Channel B</td>
</tr>
<tr>
<td>7</td>
<td>Channel C</td>
</tr>
<tr>
<td>8</td>
<td>Channel D</td>
</tr>
<tr>
<td>9</td>
<td>Channel E</td>
</tr>
<tr>
<td>10</td>
<td>Channel F</td>
</tr>
<tr>
<td>11</td>
<td>Channel G</td>
</tr>
<tr>
<td>12</td>
<td>Channel H</td>
</tr>
<tr>
<td>13</td>
<td>Channel I</td>
</tr>
</tbody>
</table>

The index signal I is synchronised with channel A or B.

---

## Encoder MR, Type L, 256 - 1024 CPT, 3 Channels, with Line Driver

### Order Number

<table>
<thead>
<tr>
<th>Type</th>
<th>Counts per turn</th>
<th>Number of channels</th>
<th>Max. operating frequency (kHz)</th>
<th>Max. speed (rpm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE 30, 60 W 80</td>
<td>256</td>
<td>3</td>
<td>80</td>
<td>18750</td>
</tr>
<tr>
<td>RE 30, 60 W 80</td>
<td>500</td>
<td>3</td>
<td>200</td>
<td>24000</td>
</tr>
<tr>
<td>RE 30, 60 W 80</td>
<td>512</td>
<td>3</td>
<td>200</td>
<td>18750</td>
</tr>
<tr>
<td>RE 30, 60 W 80</td>
<td>1000</td>
<td>3</td>
<td>320</td>
<td>12000</td>
</tr>
<tr>
<td>RE 30, 60 W 80</td>
<td>1024</td>
<td>3</td>
<td>320</td>
<td>18750</td>
</tr>
</tbody>
</table>

### Encoder, Line Driver

- **Cycle C = 360°**
- **Pulse P = 180°**

### Connection example

- **Encoder MR**
  - **Channel A**
    - **Channel B**
    - **Channel C**
    - **Channel D**
    - **Channel E**
    - **Channel F**
    - **Channel G**
    - **Channel H**
    - **Channel I**

- **DIN Connector 41651**
  - **Line receiver**
  - **Recommended IC’s:**
    - MC 3486
    - SN 75175
    - AM 26 LS 32

---

## maxon Modular System

### Type

- **Counts per turn**
- **Number of channels**
- **Max. operating frequency (kHz)**
- **Max. speed (rpm)**

### Overall length

<table>
<thead>
<tr>
<th>Type</th>
<th>Counts per turn</th>
<th>Number of channels</th>
<th>Max. operating frequency (kHz)</th>
<th>Max. speed (rpm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE 30, 60 W 80</td>
<td>256</td>
<td>3</td>
<td>80</td>
<td>18750</td>
</tr>
<tr>
<td>RE 30, 60 W 80</td>
<td>500</td>
<td>3</td>
<td>200</td>
<td>24000</td>
</tr>
<tr>
<td>RE 30, 60 W 80</td>
<td>512</td>
<td>3</td>
<td>200</td>
<td>18750</td>
</tr>
<tr>
<td>RE 30, 60 W 80</td>
<td>1000</td>
<td>3</td>
<td>320</td>
<td>12000</td>
</tr>
<tr>
<td>RE 30, 60 W 80</td>
<td>1024</td>
<td>3</td>
<td>320</td>
<td>18750</td>
</tr>
</tbody>
</table>

### Technical Data

- **Supply voltage**: 5 V ± 5%
- **Output signal**: TTL compatible
- **Phase shift φ**: 90° ± 45°
- **Operating temperature rang**: -25...+85°C
- **Moment of inertia of code wheel**: 1.7 g cm²
- **Output current per channel**: max. 5 mA

---

## maxon sensor

- **Stock program**
- **Standard program**
- **Special program (on request)**
D.3 UGV PTU PAN SYSTEM

D.3.1 PAN MOTOR

D.3.2 PAN MOTOR PLANETARY GEARHEAD

D.3.3 PAN MOTOR ENCODER
Operating Range

In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.

Thermal limit.

Short term operation

The motor may be briefly overloaded (recurring).

Thermal Data

| Thermal resistance housing-ambient | 13.2 K/W |
| Thermal resistance winding-housing | 3.2 K/W |
| Thermal time constant winding | 12.4 s |
| Thermal time constant motor | 785 s |
| Ambient temperature | -30 ... +85°C |
| Max. permissible winding temperature | +125°C |

Mechanical Data (ball bearings)

| Max. permissible speed | 10400 rpm |
| Axial play | 0.1 - 0.2 mm |
| Radial play | 0.025 mm |
| Max. axial load (dynamic) | 5 N |
| Max. force for press fits (static) | 75 N |
| Max. radial loading, 5 mm from flange | 20.5 N |

Mechanical data (sleeve bearings)

| Max. permissible speed | 10400 rpm |
| Axial play | 0.1 - 0.2 mm |
| Radial play | 0.012 mm |
| Max. axial load (dynamic) | 1.7 N |
| Max. force for press fits (static) | 80 N |
| Max. radial loading, 5 mm from flange | 5.5 N |

Other specifications

| Number of pole pairs | 1 |
| Number of commutator segments | A |
| Weight of motor | 119 g |

Specifications

<table>
<thead>
<tr>
<th>Pole pair 1</th>
<th>Pole pair 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10400 rpm</td>
<td>10400 rpm</td>
</tr>
<tr>
<td>0.1 - 0.2 mm</td>
<td>0.1 - 0.2 mm</td>
</tr>
<tr>
<td>0.025 mm</td>
<td>0.012 mm</td>
</tr>
<tr>
<td>5 N</td>
<td>1.7 N</td>
</tr>
<tr>
<td>75 N</td>
<td>80 N</td>
</tr>
<tr>
<td>20.5 N</td>
<td>5.5 N</td>
</tr>
</tbody>
</table>

Order Number

with terminals

<table>
<thead>
<tr>
<th>Model</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>110968</td>
<td>353608</td>
</tr>
<tr>
<td>110969</td>
<td>353607</td>
</tr>
<tr>
<td>110970</td>
<td>353606</td>
</tr>
<tr>
<td>110971</td>
<td>353605</td>
</tr>
<tr>
<td>110972</td>
<td>353610</td>
</tr>
<tr>
<td>110973</td>
<td>353611</td>
</tr>
<tr>
<td>110974</td>
<td>353612</td>
</tr>
<tr>
<td>110975</td>
<td>353613</td>
</tr>
<tr>
<td>110976</td>
<td>353614</td>
</tr>
<tr>
<td>110977</td>
<td>353615</td>
</tr>
<tr>
<td>110978</td>
<td>353616</td>
</tr>
</tbody>
</table>

with cables

<table>
<thead>
<tr>
<th>Model</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>353606</td>
<td>353607</td>
</tr>
<tr>
<td>353608</td>
<td>353609</td>
</tr>
<tr>
<td>353610</td>
<td>353611</td>
</tr>
<tr>
<td>353612</td>
<td>353613</td>
</tr>
<tr>
<td>353614</td>
<td>353615</td>
</tr>
<tr>
<td>353616</td>
<td>353617</td>
</tr>
<tr>
<td>353618</td>
<td>353619</td>
</tr>
</tbody>
</table>

Motor Data

Values at nominal voltage

| Nominal voltage V | 6.0 | 7.2 | 12.0 | 15.0 | 18.0 | 24.0 | 30.0 | 36.0 | 42.0 | 48.0 | 48.0 |
| No load speed rpm | 9110 | 9850 | 8010 | 8320 | 7950 | 8820 | 7020 | 7250 | 7850 | 7450 | 6000 |
| No load current mA | 140 | 128 | 56.7 | 37.0 | 31.7 | 18.9 | 16.3 | 15.5 | 12.7 | 9.67 |
| Nominal torque (max. continuous torque) mNm | 5.53 | 6.30 | 14.2 | 16.5 | 17.2 | 17.2 | 17.6 | 17.7 | 17.6 | 17.5 | 17.7 |
| Nominal current (max. continuous current) A | 1.08 | 1.08 | 1.08 | 1.03 | 0.846 | 0.704 | 0.455 | 0.394 | 0.365 | 0.300 | 0.244 |
| Stall torque mNm | 54.4 | 59.6 | 66.6 | 70.3 | 69.8 | 77.6 | 60.5 | 63.1 | 68.4 | 63.2 | 50.6 |
| Starting current A | 9.32 | 9.07 | 4.80 | 4.19 | 3.30 | 3.04 | 1.51 | 1.35 | 1.36 | 1.04 | 0.672 |
| Max. efficiency % | 68 | 70 | 76 | 77 | 78 | 79 | 79 | 79 | 79 | 79 | 77 |

Characteristics

| Terminal resistance Ω | 0.644 | 0.794 | 2.50 | 3.58 | 5.46 | 7.90 | 9.19 | 26.9 | 30.9 | 46.0 | 71.4 |
| Terminal inductance mH | 0.0402 | 0.0509 | 0.227 | 0.332 | 0.528 | 0.770 | 1.90 | 2.57 | 2.99 | 4.34 | 6.68 |
| Torque constant mNm/A | 5.84 | 6.57 | 13.9 | 16.8 | 21.2 | 25.5 | 40.1 | 46.7 | 50.3 | 60.6 | 75.2 |
| Speed constant rpm/V | 1640 | 1450 | 689 | 569 | 451 | 374 | 238 | 205 | 190 | 158 | 127 |
| Speed / torque gradient rpm/mNm | 180 | 176 | 124 | 116 | 116 | 116 | 118 | 117 | 116 | 120 | 121 |
| Mechanical time constant ms | 27.1 | 23.6 | 17.6 | 16.7 | 16.2 | 15.8 | 15.4 | 15.3 | 15.2 | 15.2 | 15.2 |
| Rotor inertia gcm² | 14.3 | 12.8 | 13.5 | 13.2 | 13.0 | 12.5 | 12.5 | 12.5 | 12.1 | 12.0 | 12.0 |

Assigning power rating

Continuous operation

In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.

- Thermal limit.

Short term operation

The motor may be briefly overloaded (recurring).

Motor modular system

Overview on page 16 - 21

<table>
<thead>
<tr>
<th>Planetary Gearhead</th>
<th>Spur Gearhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>26 mm</td>
<td>32 mm</td>
</tr>
<tr>
<td>0.5 - 20 Nm</td>
<td>0.4 - 6.0 Nm</td>
</tr>
<tr>
<td>Page 228</td>
<td>Page 229</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Planetary Gearhead</th>
<th>Spur Gearhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 mm</td>
<td>0.07 - 0.2 Nm</td>
</tr>
<tr>
<td>Page 229</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoder MR</th>
<th>Encoder Enc</th>
</tr>
</thead>
<tbody>
<tr>
<td>128 - 1000 Imp.</td>
<td>22 mm</td>
</tr>
<tr>
<td>3 channels</td>
<td>100 Imp.</td>
</tr>
<tr>
<td>Page 262</td>
<td>Page 265</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoder HED</th>
<th>Encoder MEnc</th>
</tr>
</thead>
<tbody>
<tr>
<td>5540</td>
<td>13 mm</td>
</tr>
<tr>
<td>500 Imp.</td>
<td>16 Imp.</td>
</tr>
<tr>
<td>3 channels</td>
<td>2 channels</td>
</tr>
<tr>
<td>Page 267</td>
<td>Page 275</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoder LSC</th>
<th>Encoder ADS</th>
</tr>
</thead>
<tbody>
<tr>
<td>30/2</td>
<td>50/5</td>
</tr>
<tr>
<td>Page 282</td>
<td>Page 283</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoder EPOS2</th>
<th>Encoder EPOS2P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 36/2</td>
<td>24/5</td>
</tr>
<tr>
<td>304</td>
<td>308</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoder EPOS2</th>
<th>Encoder EPOS2P</th>
</tr>
</thead>
<tbody>
<tr>
<td>50/5</td>
<td>24/5</td>
</tr>
<tr>
<td>305</td>
<td>308</td>
</tr>
</tbody>
</table>
Planetary Gearhead GP 32 A !32 mm, 0.75 - 4.5 Nm
straight teeth
stainless steel
8 mm
ball bearing
max. 0.14 mm
max. 0.4 mm
140 N
120 N
120 N
=
< 6000 rpm
-40 ... +100°C

M 1:2
Option: Low-noise version

Stock program
Standard program
Special program (on request)

Order Number
166155 166158 166163 166164 166169 166174 166179 166184 166187 166192 166197 166202

Gearhead Data
1 Reduction
2 Reduction absolute
3 Max. motor shaft diameter
mm
Order Number
1 Reduction
2 Reduction absolute
3 Max. motor shaft diameter
mm
Order Number
1 Reduction
2 Reduction absolute
3 Max. motor shaft diameter
mm
Order Number
1 Reduction
2 Reduction absolute
3 Max. motor shaft diameter
mm
Order Number
1 Reduction
2 Reduction absolute
3 Max. motor shaft diameter
mm
4 Number of stages
5 Max. continuous torque
Nm
6 Intermittently permissible torque at gear output Nm
7 Max. efficiency
%
8 Weight
g
9 Average backlash no load
°
10 Mass inertia
gcm2
11 Gearhead length L1
mm

3.7 : 1
14 : 1
26/
676/
7
49
6
6
166156 166159
4.8 : 1
18 : 1
24/
624/
5
35
4
4
166157 166160
5.8 : 1
21 : 1
23/
299/
4
14
3
3
166161
23 : 1
576/
25
4
166162
28 : 1
138/
5
3
1
2
0.75
2.25
1.1
3.4
80
75
118
162
0.7
0.8
1.5
0.8
26.5
36.4

33 : 1
529/
16
3

2
2.25
3.4
75
162
0.8
0.8
36.4

51 : 1
17576/

343

6
166165
66 : 1
16224/
245
4
166166
79 : 1
3887/
49
3
166167
86 : 1
14976/
175
4
166168
103 : 1
3588/
35
3
3
4.50
6.5
70
194
1.0
0.7
43.1

111 : 1
13824/

125

4
166170
123 : 1
6877/
56
3
166171
132 : 1
3312/
25
3
166172
159 : 1
1587/
10
3
166173
190 : 1
12167/
64
3
3
4.50
6.5
70
194
1.0
0.7
43.1

246 : 1

762 : 1 1181 : 1 1972 : 1 2829 : 1 4380 : 1
10123776/
8626176/
495144/
109503/
8575
4375
25
175
25
4
3
3
4
4
3
3
166175 166180 166185 166188 166193 166198 166203
295 : 1 531 : 1 913 : 1 1414 : 1 2189 : 1 3052 : 1 5247 : 1
2425488/
1907712/
536406/
839523/
101062/
331776/
36501/
1715
625
343
625
40
245
160
3
4
3
3
3
3
3
166176 166181 166186 166189 166194 166199 166204
318 : 1 589 : 1 1093 : 1 1526 : 1 2362 : 1 3389 : 1 6285 : 1
389376/
9345024/
6436343/
2066688/
20631/
474513/
279841/
1225
6125
875
1024
35
256
140
4
3
3
4
3
3
3
166190 166195 166200
166177 166182
411 : 1 636 : 1
1694 : 1 2548 : 1 3656 : 1
1162213
359424/
79488/
/686 7962624/3125 457056/125
875
125
4
3
3
4
3
166178 166183
166191 166196 166201
456 : 1 706 : 1
1828 : 1 2623 : 1 4060 : 1
2238912/
2056223/
3637933/
89401/
158171/
1225
784
896
196
224
3
3
3
3
3
4
4
4
5
5
5
5
4.50
4.50
4.50
4.50
4.50
4.50
4.50
6.5
6.5
6.5
6.5
6.5
6.5
6.5
60
60
60
50
50
50
50
226
226
226
258
258
258
258
1.0
1.0
1.0
1.0
1.0
1.0
1.0
0.7
0.7
0.7
0.7
0.7
0.7
0.7
49.8
49.8
49.8
56.5
56.5
56.5
56.5
421824/

1715

492 : 1

86112/

175

19044/

overall length

overall length

maxon Modular System
+ Motor
RE 25
RE 25
RE 25
RE 25
RE 25
RE 25, 20 W
RE 25, 20 W
RE 25, 20 W
RE 25, 20 W
RE 25, 20 W
RE 25, 20 W
RE 25, 20 W
RE 25, 20 W
A-max 26
A-max 26
A-max 26
A-max 26
A-max 26
RE-max 29
RE-max 29

Page
77/79
77/79
77/79
77/79
77/79
78
78
78
78
78
78
79
79
101-108
102-108
102-108
102-108
102-108
131-134
132/134

+ Sensor / Brake

Page

MR
Enc 22
HED_ 5540
DCT 22

262
264
266/268
276

MR
HED_ 5540
DCT22
AB 28
HED_ 5540 / AB 28
AB 28
HED_ 5540/AB 28

262
267/270
276
318
267/318
318
266/318

MEnc 13
MR
Enc 22
HED_ 5540

275
262
265
267/269

MR

262

May 2010 edition / subject to change

Overall length [mm] = Motor length + gearhead length + (sensor / brake) + assembly parts
81.1
91.0
91.0
97.7
97.7
104.4
104.4
104.4
111.1
92.1
102.0
102.0
108.7
108.7
115.4
115.4
115.4
122.1
95.2
105.1
105.1
111.8
111.8
118.5
118.5
118.5
125.2
101.9
111.8
111.8
118.5
118.5
125.2
125.2
125.2
131.9
103.4
113.3
113.3
120.0
120.0
126.7
126.7
126.7
133.4
69.6
79.5
79.5
86.2
86.2
92.9
92.9
92.9
99.6
80.6
90.5
90.5
97.2
97.2
103.9
103.9
103.9
110.6
90.4
100.3
100.3
107.0
107.0
113.7
113.7
113.7
120.4
91.9
101.8
101.8
108.5
108.5
115.2
115.2
115.2
121.9
103.7
113.6
113.6
120.3
120.3
127.0
127.0
127.0
133.7
120.9
130.8
130.8
137.5
137.5
144.2
144.2
144.2
150.9
115.2
125.1
125.1
131.8
131.8
138.5
138.5
138.5
145.2
132.3
142.2
142.2
148.9
148.9
155.6
155.6
155.6
162.3
71.3
81.2
81.2
87.9
87.9
94.6
94.6
94.6
101.3
78.4
88.3
88.3
95.0
95.0
101.7
101.7
101.7
108.4
80.1
90.0
90.0
96.7
96.7
103.4
103.4
103.4
110.1
85.7
95.6
95.6
102.3
102.3
109.0
109.0
109.0
115.7
90.1
100.0
100.0
106.7
106.7
113.4
113.4
113.4
120.1
71.3
81.2
81.2
87.9
87.9
94.6
94.6
94.6
101.3
80.1
90.0
90.0
96.7
96.7
103.4
103.4
103.4
110.1

111.1
122.1
125.2
131.9
133.4
99.6
110.6
120.4
121.9
133.7
150.9
145.2
162.3
101.3
108.4
110.1
115.7
120.1
101.3
110.1

111.1
122.1
125.2
131.9
133.4
99.6
110.6
120.4
121.9
133.7
150.9
145.2
162.3
101.3
108.4
110.1
115.7
120.1
101.3
110.1

111.1
122.1
125.2
131.9
133.4
99.6
110.6
120.4
121.9
133.7
150.9
145.2
162.3
101.3
108.4
110.1
115.7
120.1
101.3
110.1

maxon gear

231

maxon gear

Technical Data
Planetary Gearhead
Output shaft
Shaft diameter as option
Bearing at output
Radial play, 5 mm from flange
Axial play
Max. radial load, 10 mm from flange
Max. permissible axial load
Max. permissible force for press fits
Sense of rotation, drive to output
Recommended input speed
Recommended temperature range


Encoder Enc 22, 100 Counts per turn, 2 Channels

Stock program
Standard program
Special program (on request)

Order Number

<table>
<thead>
<tr>
<th>Type</th>
<th>103937</th>
<th>143330</th>
<th>103935</th>
<th>110520</th>
<th>168045</th>
<th>110521</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts per turn</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Number of channels</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Max. speed (rpm)</td>
<td>12000</td>
<td>12000</td>
<td>12000</td>
<td>12000</td>
<td>12000</td>
<td>12000</td>
</tr>
<tr>
<td>Shaft diameter (mm)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Orientation encoder to motor mounting defined</td>
<td>± 5°</td>
<td>± 5°</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**maxon Modular System**

<table>
<thead>
<tr>
<th>Type</th>
<th>Motor</th>
<th>Gearhead</th>
<th>Overall length [mm] / see Gearhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-max 19, 1.5 W</td>
<td>94</td>
<td>43.3</td>
<td></td>
</tr>
<tr>
<td>A-max 19, 1.5 W</td>
<td>94</td>
<td>GP 19, 0.1 - 0.3 Nm</td>
<td>218</td>
</tr>
<tr>
<td>A-max 19, 1.5 W</td>
<td>94</td>
<td>GS 20, 0.06 - 0.25 Nm</td>
<td>219</td>
</tr>
<tr>
<td>A-max 19, 1.5 W</td>
<td>94</td>
<td>GP 22, 0.1 - 2.0 Nm</td>
<td>220-223</td>
</tr>
<tr>
<td>A-max 19, 1.5 W</td>
<td>94</td>
<td>GS 24, 0.1 Nm</td>
<td>227</td>
</tr>
<tr>
<td>A-max 19, 1.5 W</td>
<td>94</td>
<td>GP 32 S</td>
<td>247/248</td>
</tr>
<tr>
<td>A-max 19, 2.5 W</td>
<td>96</td>
<td>GP 22 S</td>
<td>247/248</td>
</tr>
<tr>
<td>A-max 19, 2.5 W</td>
<td>96</td>
<td>GS 24, 0.1 Nm</td>
<td>227</td>
</tr>
<tr>
<td>A-max 19, 2.5 W</td>
<td>96</td>
<td>GP 32 S</td>
<td>247/248</td>
</tr>
<tr>
<td>A-max 22</td>
<td>98/100</td>
<td>GP 22, 0.1 - 0.3 Nm</td>
<td>220</td>
</tr>
<tr>
<td>A-max 22</td>
<td>98/100</td>
<td>GP 22, 0.2 - 0.6 Nm</td>
<td>221</td>
</tr>
<tr>
<td>A-max 22</td>
<td>98/100</td>
<td>GP 22, 0.1 - 2.0 Nm</td>
<td>220-223</td>
</tr>
<tr>
<td>A-max 22</td>
<td>98/100</td>
<td>GS 24, 0.1 Nm</td>
<td>227</td>
</tr>
<tr>
<td>A-max 22</td>
<td>98/100</td>
<td>GP 22 S</td>
<td>247/248</td>
</tr>
<tr>
<td>A-max 26</td>
<td>102-108</td>
<td>GP 26, 0.5 - 2.0 Nm</td>
<td>228</td>
</tr>
<tr>
<td>A-max 26</td>
<td>102-108</td>
<td>GP 30, 0.07 - 0.2 Nm</td>
<td>229</td>
</tr>
<tr>
<td>A-max 26</td>
<td>102-108</td>
<td>GP 32, 0.75 - 4.5 Nm</td>
<td>230</td>
</tr>
<tr>
<td>A-max 26</td>
<td>102-108</td>
<td>GP 32, 0.75 - 4.5 Nm</td>
<td>231</td>
</tr>
<tr>
<td>A-max 26</td>
<td>102-108</td>
<td>GP 32, 1.0 - 6.0 Nm</td>
<td>234</td>
</tr>
<tr>
<td>A-max 26</td>
<td>102-108</td>
<td>GS 38, 0.1 - 0.6 Nm</td>
<td>237</td>
</tr>
<tr>
<td>A-max 26</td>
<td>102-108</td>
<td>GP 32 S</td>
<td>247/248</td>
</tr>
</tbody>
</table>

**Technical Data**

- Supply voltage: 5 V ± 10 %
- Output signal: TTL compatible
- Phase shift: 90° ± 45°
- Signal rise time: 200 ns
- Signal fall time: 50 ns
- Operating temperature range: -20...+65°C
- Moment of inertia of code wheel: ≤ 0.05 gcm²
- Output current per channel: min. -1 mA, max. 5 mA

**Pin Allocation**

- Micromodule contact strip
- Type Lumberg MICS 4
- Pin 4 GND
- Pin 3 Channel A
- Pin 2 VCC
- Recommended connectors: Micromodule connector Type Lumberg MICA 4

**Connection example**

- VCC 5 VDC ± 5 %
- Rpull-up 3.3 kΩ
- Ambient temperature range Aa = 22 - 25°C
D.4 GPS SYSTEM

D.4.1 GPS RECEIVER

D.4.2 GPS RECEIVER ENCLOSURE

D.4.3 GPS ANTENNA
Receivers

OEMV-2

Compact, Dual Frequency
GNSS Receiver Delivers Robust
RTK Functionality

Benefits

Proven OEMV® technology

Lowest power consumption in the market for a dual frequency receiver

Application Programming Interface (API) reduces hardware requirements and system complexity

Easy to integrate

Features

L1, L2 and L2C signal tracking

Increased satellite availability with GLONASS tracking

RT-2™, RT-20®, ALIGN™ and GL1DE firmware options

Designed for Efficiency

The OEMV-2 sports low power consumption and a small form factor for ease and efficiency in integration. The modular nature of OEMV-2 firmware allows the user the flexibility to configure the receiver from a basic GPS L1-only to a dual frequency receiver with RTK functionality.

Greater Performance with GNSS Functionality

The OEMV-2 is configurable with GPS or GPS+GLONASS real-time capabilities. The GPS+GLONASS option increases available positions in obstructed sky conditions and allows users to work more often.

Enhanced, Flexible Firmware Features

With L2C tracking capabilities, the OEMV-2 is ideal for low signal strength applications, providing stronger signal tracking and better cross correlation protection. The OEMV-2 provides decimetre-level pass-to-pass accuracy with NovAtel’s GL1DE™ technology. NovAtel’s optional Advance™ RTK technology is available for centimetre-level real-time position accuracy. ALIGN™ technology is available for heading and position outputs.

Customization With The API

The Application Programming Interface (API) functionality is available on the OEMV-2. Using a recommended compiler with the API library, an application can be developed in a standard C/C++ environment to run directly from the receiver platform; eliminating system hardware, reducing development time and resulting in faster time to market.

If you require more information about our receivers, visit novatel.com/products/receivers.htm
## Performance

### Channel Configuration
- 14 GPS L1, 14 GPS L2
- 12 GLONASS L1, 12 GLONASS L2
- 2 SBAS

### Horizontal Position Accuracy (RMS)
- Single Point L1: 1.5 m
- Single Point L1/L2: 1.2 m
- SBAS: 0.6 m
- DGPS: 0.4 m
- RT-2: 0.2 m
- RT-2: 1 cm +/- 1 ppm

### Measurement Precision (RMS)
- L1 C/A Code: 4 cm
- L1 Carrier Phase: 0.5 mm
- L2 P(Y) Code: 8 cm
- L2 Carrier Phase: 1 mm

### Data Rate
- Measurements: 20 Hz
- Position: 20 Hz

### Time to First Fix
- Cold Start: 60 s
- Hot Start: 35 s

### Signal Reacquisition
- L1: 0.5 s (typical)
- L2: 1.0 s (typical)

### Time Accuracy
- 20 ns RMS

### Velocity Accuracy
- 0.03 m/s RMS

### Velocity
- 515 m/s

## Physical and Electrical

### Dimensions
- 60 x 100 x 13 mm

### Weight
- 56 g

### Power
- Input Voltage: +3.3 to +5.0 +/- 3% VDC
- Power Consumption: 1.2 W (GPS only)
- 1.6 W (GPS & GLONASS)

### Antenna LNA Power Output
- Output Voltage: +5.1 VDC
- Maximum Current: 100 mA

### Communication Ports
- 1 RS-232 or RS-422 capable of 300 to 921,600 bps
- 2 LV-TTL serial port capable of 300 to 230,400 bps
- 1 CAN Bus serial port capable of 1 Mbps
- 1 USB port capable of 5 Mbps

### Input/Output Connectors
- Main: 24-pin dual row male header
- Antenna Input: MMCX female
- External Oscillator Input: MMCX female

### Environmental
- Temperature
  - Operating: -40°C to +85°C
  - Storage: -45°C to +95°C
- Humidity: 95% non-condensing

### Options and Accessories
- GPS-700 series antennas
- ANT series antennas
- RF Cables: 5, 10 and 30 m lengths
- 50 Hz output rate
- Right angle RF connector

### Additional Firmware Features
- RT-20
- RT-2
- ALIGN
- GL1DE
- Pseudo Range/Delta-Phase (PDP) Positioning

### Additional Features
- Common, field-upgradeable software for all OEMV family receivers
- Auxiliary strobe signals, including a configurable PPS output for time synchronization and mark inputs
- Outputs to drive external LEDs
- External oscillator input

---

*Typical values. Performance specifications subject to GPS system characteristics, US DOD operational degradation, ionospheric and tropospheric conditions, satellite geometry, baseline length, multipath effects and the presence of intentional or unintentional interference sources.*

*GPS only.*

*Expected accuracy after static convergence.*

*Typical value. No almanac or ephemerides and no approximate position or time.*

*Typical value. Almanac and recent ephemerides saved and approximate position and time entered.*

*Time accuracy does not include biases due to RF or antenna delay.*

*Export licensing restricts operation to a maximum of 515 metres per second.*

*GLONASS is not supported at 50 Hz.*

---

For the most recent details of this product: [novatel.com/Documents/Papers/OEMV-2.pdf](http://novatel.com/Documents/Papers/OEMV-2.pdf)
**Enclosures**

<table>
<thead>
<tr>
<th>FlexPak-G2™</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compact Enclosure Supports a Range of NovAtel GNSS Receivers Offering Flexible Performance for Any Application</strong></td>
</tr>
</tbody>
</table>

## Benefits
- Proven NovAtel GNSS technology
- Easy to integrate
- Ideal for low-payload UAV and robotics applications
- Field upgradable to support all OEMStar, OEMV-1, OEMV-1G and OEMV-2 functionality

## Features
- Metre to centimetre-level accuracy
- Auxiliary strobe signals with configurable PPS output
- Shock and dust resistant; waterproof to IPX7
- Rugged DB-9 connectors with power in/out support
- Active antenna support

---

### Scalable Functionality
The FlexPak-G2 is available in four variants, all of which are software upgradable in the field to provide the custom performance required for your application demands.

- **FlexPak-G2-V2**: Offers dual frequency GPS+GLONASS tracking, modernized to support GPS L2C, allowing stronger signal tracking. Available with NovAtel's AdVance™ RTK for centimetre-level accuracy with fast initialization over extended baselines.
- **FlexPak-G2-V1G**: Provides GPS+GLONASS L1 tracking for reliable positioning even in obstructed sky conditions. NovAtel’s RT-2 L1TE L1-only RTK algorithm allows reliable centimetre level accuracy for high precision applications.
- **FlexPak-G2-V1**: Delivers GPS-only L1 tracking, and supports OmniStar® VBS, CDGPS and SBAS corrections for accurate and reliable DGPS positioning. NovAtel’s RT-20® L1 carrier-phase positioning is available for increased accuracy with a 20 Hz data rate.
- **FlexPak-G2 with OEMStar™**: Offers L1 GPS+GLONASS positioning and measurements in combination with GPS data to provide increased satellite availability for positioning in challenging environments.

### Base Station or Rover
All FlexPak-G2 models are capable of base station or rover operation. Using standardized RTCM 2.3, RTCMV3 and CMR+ message types, the FlexPak-G2 is compatible with all NovAtel and third party GNSS receivers.

### Enhanced Connectivity
Two standard DB-9 communication ports support power in and out; one port may be dedicated to powering and communicating with a radio, while the other may be dedicated to your host application. Independent input/output and USB ports may be used simultaneously for time synchronization and direct connection to your laptop for field operation.

---

*If you require more information about our enclosures, visit novatel.com/products/enclosures.htm*
### FlexPak-G2 with OEMStar

- **Dimensions**: 147 x 113 x 45 mm
- **Weight**: 313 g
- **Power Consumption**: 1.1 W
- **Channel Configuration**:
  - 10 GPS L1 + 2 SBAS
  - 8 GPS L1 + 4 GLO L1 + 2 SBAS
- **Altitude**: 18 288 m

### FlexPak-G2-V1

- **Dimensions**: 147 x 113 x 45 mm
- **Weight**: 314 g
- **Power Consumption**: 1.4 W
- **Channel Configuration**:
  - 12 GPS L1 + 2 SBAS
  - 8 GPS L1 + 6 GLO L1
- **Altitude**: 18 288 m

### FlexPak-G2-V1G

- **Dimensions**: 147 x 113 x 45 mm
- **Weight**: 331 g
- **Power Consumption**: 1.4 W
- **Channel Configuration**:
  - 14 GPS L1 + 4 GLO L1 + 2 SBAS
  - 12 GLONASS L1
- **Horizontal Position Accuracy (RMS)**
  - Single Point L1: 1.8 m
  - SBAS: 0.6 m
  - CDGPS: 0.6 m
  - OmniSTAR VBS: 0.7 m
  - DGPS: 0.45 m
  - RT-20: 0.2 m
- **Communication Ports**:
  - 1 RS-232 serial port (230,400 bps)
  - 1 RS-232 or RS-422 (230,400 bps) serial port with power in/out
  - 1 input/output port (PPS, Event 1, Event 2)
  - 1 USB port
- **Environmental Temperature**
  - Operating: -40°C to +75°C
  - Storage: -40°C to +85°C
- **Humidity**: 95% non-condensing
- **Altitude**: 18 288 m

### FlexPak-G2-V2

- **Dimensions**: 147 x 113 x 45 mm
- **Weight**: 338 g
- **Power Consumption**: 1.4 W
- **Channel Configuration**:
  - 12 GPS L1 + 2 SBAS
  - 10 GPS L1 + 4 GLO L1
- **Horizontal Position Accuracy (RMS)**
  - Single Point L1: 1.8 m
  - SBAS: 0.6 m
  - CDGPS: 0.6 m
  - OmniSTAR VBS: 0.7 m
  - DGPS: 0.45 m
  - RT-20: 0.2 m
- **Communication Ports**:
  - 1 RS-232 serial port (230,400 bps)
  - 1 RS-232 or RS-422 (230,400 bps) serial port with power in/out
  - 1 input/output port (PPS, Event 1, Event 2)
  - 1 USB port
- **Environmental Temperature**
  - Operating: -40°C to +75°C
  - Storage: -40°C to +85°C
- **Humidity**: 95% non-condensing
- **Altitude**: 18 288 m

### Additional Features

- Common, field-upgradeable software for all OEMV family receivers
- Auxiliary strobe signals, including a configurable PPS output for time synchronization and event inputs

### Additional Accessories

- Serial cable
- I/O cable
- USB cable
- Automotive 12 VDC power adapter with 6A slow-blow fuse

### Enclosures

- RoHS COMPLIANT

---

Version 1a - Specifications subject to change without notice. ©2010 NovAtel Inc. All rights reserved. NovAtel, RT-20 and OEMV are registered trademarks of NovAtel Inc. GL1DE, AdVance, RT-2, ALIGN, OEMStar and FlexPak-G2 are trademarks of NovAtel Inc. OmniSTAR is a registered trademark of OmniSTAR Inc. Printed in Canada, D13879 FlexPak-G2 February 2010 For the most recent details of this product: novatel.com/Documents/Papers/FlexPak-G2.pdf
## Antennas

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choke ring antenna performance without size and weight</td>
<td>L1 or L1/L2 options</td>
</tr>
<tr>
<td>Reduces equipment costs</td>
<td>GPS+GLONASS signal reception</td>
</tr>
<tr>
<td>Placement flexibility and precision positioning, even on long baselines</td>
<td>Excellent multipath rejection</td>
</tr>
<tr>
<td>Eliminates need for future redesign</td>
<td>Highly stable phase center</td>
</tr>
<tr>
<td></td>
<td>RoHS compliant</td>
</tr>
</tbody>
</table>

### Dual Constellation For Enhanced Positioning

The GPS-701-GG uses the L1 frequency while the GPS-702-GG uses the L1 and L2 frequencies. Both antennas offer combined GPS+GLONASS signal reception. Customers can use the same antenna for GPS-only or dual constellation applications, resulting in increased flexibility and reduced equipment costs.

### Stable Phase Center

The phase center of these two antennas remains constant as the azimuth and elevation angle of the satellites change. Signal reception is unaffected by the rotation of the antenna or satellite elevation, so placement and installation of the antennas can be completed with ease. With the phase center in the same location for both the L1 and L2 signals, and with minimal phase center variation between the two antennas, these antennas are ideal for baselines of any length.

### Durable, Future-Proof Design

These rugged antennas are enclosed in a durable, waterproof housing and meet MIL-STD-202F for vibration and MIL-STD-810F for salt spray. Sharing the same form factor as other NovAtel GPS-700 series antennas, the GPS-701-GG and GPS-702-GG antennas are compact and lightweight, making them highly portable and suitable for a wide variety of environments and applications.

Both antennas meet the European Union’s directive for Restriction of Hazardous Substances (RoHS), so integrators can be confident these antennas can be used in system designs for years to come.

---

If you require more information about our antennas, visit novatel.com/products/antennas.htm
Elevation Gain Patterns

The plots below represent the typical right-hand circular polarized (RHCP) and left-hand circular polarized (LHCP) normalized radiation patterns for GPS L1/L2 and GLONASS L1/L2 frequencies, respectively.
D.5 PAN–TILT UNIT

D.5.1 CAMERA

D.5.2 CAMERA LENS

D.5.3 LIDAR
### Digital Color Matrix Camera, 2 Megapixels, Power over GigE

#### Sensor Information

<table>
<thead>
<tr>
<th>Model Name</th>
<th>SONY® ICX274</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>1/1.8&quot; interline progressive scan CCD</td>
</tr>
<tr>
<td>Native Resolution</td>
<td>1624 × 1232 pixels</td>
</tr>
<tr>
<td>Exposure Time</td>
<td>4 µsec ... 60 sec</td>
</tr>
<tr>
<td>Color Filter</td>
<td>RGB Bayer Mosaic</td>
</tr>
</tbody>
</table>

#### Acquisition Formats

<table>
<thead>
<tr>
<th>Image Formats</th>
<th>Full Frame HQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full Frame</td>
</tr>
<tr>
<td></td>
<td>1624 × 1232 pixels</td>
</tr>
<tr>
<td></td>
<td>1624 × 1232 pixels</td>
</tr>
<tr>
<td>Max.</td>
<td>8.0 fps</td>
</tr>
<tr>
<td>Max.</td>
<td>16.0 fps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pixel Formats</th>
<th>Mon0, BayerRGB, BayerRG12, BayerRG8 Packed, BGR8 Packed, YUV411 Packed, YUV422 Packed, YUV444 Packed</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Partial Scan</th>
<th>True Partial Scan, Region of Interest (ROI) arbitrary</th>
</tr>
</thead>
</table>

#### Image Pre-processing

| Analog Controls | Gain (0 ... 20 dB), Offset (0 ... 255 LSB | 12 bit) |
|-----------------|-------------------------------------------|
| Color Models    | RGB, YUV, Mono |
| Color Processing| Integrated color processor for high quality color calculation |
| Color Adjustment| One Push White Balance, User-specific color adjustment |

#### Camera Features

<table>
<thead>
<tr>
<th>Internal Buffer</th>
<th>32 MBytes (max. 3 images)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronization</td>
<td>Free running, Trigger</td>
</tr>
<tr>
<td>Trigger Sources</td>
<td>Hardware, Software, ActionCommand</td>
</tr>
<tr>
<td>Trigger Delay</td>
<td>0 ... 2 sec, Tracking and buffering of up to 512 trigger signals</td>
</tr>
<tr>
<td>Digital I/Os</td>
<td>1 input line (with Debouncer), 1 output line</td>
</tr>
</tbody>
</table>

#### Interfaces and Connectors

<table>
<thead>
<tr>
<th>Data Interface</th>
<th>Gigabit Ethernet, Transfer rate 1000 Mbits/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Interface</td>
<td>M8 / 4 pins</td>
</tr>
<tr>
<td>Power Interface</td>
<td>via 8PBC Modular Jack (RU45)</td>
</tr>
</tbody>
</table>

#### Mechanical Data

<table>
<thead>
<tr>
<th>Housing</th>
<th>Aluminum, IP40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lens Mount</td>
<td>C-Mount</td>
</tr>
<tr>
<td>Dimensions</td>
<td>36 × 36 × 58 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>&lt; 110 g</td>
</tr>
</tbody>
</table>

#### Electrical Data

<table>
<thead>
<tr>
<th>Power Supply</th>
<th>Power over Ethernet, Class 0, 20 ... 57 V DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Consumption</td>
<td>approx. 4.3 Watts</td>
</tr>
</tbody>
</table>

#### Environmental Data

<table>
<thead>
<tr>
<th>Operating Temperature</th>
<th>+5 °C ... +50 °C (+41 °F ... +121 °F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humidity</td>
<td>10% ... 90% non condensing</td>
</tr>
</tbody>
</table>

---

Technical data has been fully checked, but accuracy of printed matter not guaranteed. Subject to change without notice. Printed in Germany 12/09
### LM3NCM

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>LM3NCM</td>
</tr>
<tr>
<td>Focal Length (mm)</td>
<td>3.5</td>
</tr>
<tr>
<td>Image Size (mm)</td>
<td>1/2” (Ø8)</td>
</tr>
<tr>
<td>Iris Range (F-Stop)</td>
<td>F2.4~F14</td>
</tr>
<tr>
<td>Focusing Range (m)</td>
<td>0.1~∞</td>
</tr>
<tr>
<td>Iris Control</td>
<td>Manual</td>
</tr>
<tr>
<td>Focus Control</td>
<td>Manual</td>
</tr>
<tr>
<td>Shooting Range at M.O.D.</td>
<td>201 (H) X 151 (V)</td>
</tr>
<tr>
<td>Angle of View</td>
<td></td>
</tr>
<tr>
<td>2/3”</td>
<td></td>
</tr>
<tr>
<td>1/2”</td>
<td>82.4X66.9</td>
</tr>
<tr>
<td>1/3”</td>
<td>66.9X52.7</td>
</tr>
<tr>
<td>Resolution (Center, Corner)</td>
<td>120lp/mm, 100lp/mm</td>
</tr>
<tr>
<td>Distortion (TV)</td>
<td>0.40%</td>
</tr>
<tr>
<td>Back Focus in Air (mm)</td>
<td>9.7</td>
</tr>
<tr>
<td>Mount</td>
<td>C-Mount</td>
</tr>
<tr>
<td>Filter Size (mm)</td>
<td>M40.5XP0.5</td>
</tr>
<tr>
<td>Size (mm)</td>
<td>Ø42X38.2</td>
</tr>
<tr>
<td>Weight (g)</td>
<td>75</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>-10°C ~ 45°C</td>
</tr>
</tbody>
</table>
**Measurement data transmission via Fast Ethernet 100MBit/s**

**Measurement data transmission via RS 232/422 serial interface**

**Measurement data reduction, measurement data processing and measurement data filtering in the interface box**

**Service interface for configuring measurement applications and detection fields.**

**With the ROD 4plus, it is possible to save and switch between 4 detection fields for object detection.**

**ROD 4-08plus with heating, dust-insensitive version.**

**RODplussoft software for configuring measurement applications**

**RODsoft software for configuring detection fields**

---

**Accessories:**

*(available separately)*

- Mounting system
- Configuration software
- Various connection cables

---

**Measurement principle**

- **190° working range**
- **529 measurement sectors, 0.36° each**

---

**Electrical connection**

- **Y1 Logic**
- **Y2 Ethernet**
- **Y3 Service**
- **Y4 RS 232/422**
**Specifications**

**Optical data**
- Measurement range: 0 … 65m (ROD 4-08plus … 25m)
- Angular range: max. 190°
- Angular resolution: 0.36°
- Scanning rate: 25 scans/s or 40ms/scan
- Transmitter: infrared laser diode, laser class 1 (EN 60815-1), wavelength = 905nm, $P_{\text{max}} = 15$W, pulse duration: 3ns, average output power: 12µW

**Detection fields**
- Reflectivity: from min. 1.8% (matte black), ROD 4-08plus from 6% (dark grey)
- Object size: > 20mm at distance of 4m, > 100mm at distance of 15m
- Response time: at least 40ms (corresponds to 1 scan)
- Number of detection field pairs: 4 (selectable via switching inputs)
- Switching inputs: 4x +24 VDC
- Switching outputs: 4x PNP transistor output 24V/250mA
- Measurement value resolution per sector: 5mm
- Repeatability: 10 … 90% diffuse refl. at op. range of 4m: ±15mm / ±20mm

**Electrical data**
- Voltage supply: +24 VDC +20 % / -30 %
- Overcurrent protection: via fuse 2.5A semi-time-lag in the switching cabinet
- Current consumption: approx. 1A (use power supply with 1.5A), approx. 2.5A with heating
- Power consumption: < 75W at 24V including the outputs
- Overvoltage protection: overvoltage protection with protected limit stop

**Mechanical data**
- Housing: diecast aluminium, plastic
- Weight: 2.3kg
- Connection type: 4 connectors (can be plugged from above)

**Environmental data**
- Ambient temp. (operation/storage): -0°C … +50°C / -20°C … +50°C
- VDE safety class: II, all-insulated
- Protection class: IP 65
- Laser class: 1 (acc. to EN 60825-1)
- Standards applied: IEC 60947-5-2

**Interface assignments**

<table>
<thead>
<tr>
<th>Y1 Logic</th>
<th>V2 Ethernet</th>
<th>Y3 Service</th>
<th>Y4 RS 232/422</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN</td>
<td>Function</td>
<td>PIN</td>
<td>Function</td>
</tr>
<tr>
<td>A</td>
<td>+Ug</td>
<td>1</td>
<td>Tx+</td>
</tr>
<tr>
<td>C</td>
<td>GND_IN</td>
<td>2</td>
<td>Rx+</td>
</tr>
<tr>
<td>E</td>
<td>FPP1</td>
<td>3</td>
<td>Tx-</td>
</tr>
<tr>
<td>G</td>
<td>FPP2</td>
<td>4</td>
<td>Rx-</td>
</tr>
<tr>
<td>J</td>
<td>FPP3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>FPP4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Restart_IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Near field 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>Near field 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>Alarm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Warn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Din</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Out1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>Out2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Order guide**

With heating/dust-insensitive

- Designation: ROD 4plus
- Part No.: 501 06481
- Designation: ROD 4-08plus
- Part No.: 501 06480

**Remarks**

"RODplussoft" Configuration Software

The configuration software runs under Windows 2000/XP and offers the following features:
- Interface configuration of the ROD 4plus...
- Measurement data visualisation
- Configuration of measurement segments
- Filtering data output
- Start of RODsoft for definition of detection fields

There are a variety of options available for defining detection fields. These include e.g.:
- "Teach-In" function
- Numeric and graphical input of the detection fields
- "Edit" function

Approved purpose:
The ROD 4 distance sensors are optical electronic sensors for the optical, contactless measurement of distance to objects.
D.6 OTHER UGV SENSORS

D.6.1 IMU

D.6.2 ULTRASONICS
Technical Product Overview

3DM-GX3™-25

Introduction
3DM-GX3™-25 is a high-performance, miniature Attitude Heading Reference System (AHRS), utilizing MEMS sensor technology. It combines a triaxial accelerometer, triaxial gyro, triaxial magnetometer, temperature sensors, and an on-board processor running a sophisticated sensor fusion algorithm to provide static and dynamic orientation, and inertial measurements.

The system offers a range of output data quantities, including fully calibrated inertial measurements: acceleration, angular rate, and magnetic field; or deltaAngle & deltaVelocity vectors. It can also output computed orientation estimates: pitch, roll, and heading (yaw) or rotation matrix. All quantities are fully temperature compensated and are mathematically aligned to an orthogonal coordinate system. The angular rate quantities are further corrected for G-sensitivity and scale factor non-linearity to third order.

The 3DM-GX3™-25 AHRS is available with RS-232, USB 2.0 and TTL serial communication interfaces and is a member of the 3DM-GX3™ family of inertial sensors.

Features & Benefits
- smallest and lightest AHRS available on the market, with versions weighing only 11.5 grams
- fully temperature compensated over entire -40°C to 75°C operational range
- calibrated for sensor misalignment, gyro G-sensitivity, and gyro scale factor non-linearity
- improved navigation under vibration, as sensors are sampled at 30 kHz and digitally filtered and scaled into physical units; coning and sculling integrals are computed at 1 kHz
- available with RS-232, USB 2.0 and TTL serial communication interfaces
- user adjustable data rate, 1 to 1,000 Hz
- outputs Euler angles, rotation matrix, deltaAngle & deltaVelocity, acceleration angular rate and magnetic field

Applications
- inertial aiding of GPS
- location tracking of personnel
- unmanned vehicles, navigation, artificial horizon
- computer science, biomedical animation, linkage free tracking/control
- platform stabilization
- antenna and camera pointing
- robotics

MicroStrain® Little Sensors. Big Ideas.®

www.microstrain.com
3DM-GX3™ -25 Miniature Attitude Heading Reference Sensor

The system architecture has been carefully designed to substantially eliminate common sources of error such as hysteresis induced by temperature changes and sensitivity to supply voltage variations. On-board coning and sculling compensation allows for use of lower data output rates while maintaining performance of a fast internal sampling rate.

Weighing only 11.5 grams, the OEM version of the 3DM-GX3™ -25 AHRS

Specifications

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orientation range</td>
<td>360° about all axes</td>
</tr>
<tr>
<td>Accelerometer range</td>
<td>± 5 g standard</td>
</tr>
<tr>
<td></td>
<td>± 2 g, ± 18 g and ± 50 g also available</td>
</tr>
<tr>
<td>Accelerometer bias stability</td>
<td>± 0.005 g for ± 5 g range</td>
</tr>
<tr>
<td></td>
<td>± 0.003 g for ± 2 g range</td>
</tr>
<tr>
<td></td>
<td>± 0.010 g for ± 18 g range</td>
</tr>
<tr>
<td></td>
<td>± 0.050 g for ± 50 g range</td>
</tr>
<tr>
<td>Accelerometer nonlinearity</td>
<td>0.2 %</td>
</tr>
<tr>
<td>Gyro range</td>
<td>± 300°/sec standard, ± 1200°/sec, ± 600°/sec, ± 150°/sec, ± 75°/sec also available</td>
</tr>
<tr>
<td>Gyro bias stability</td>
<td>± 0.2°/sec for ± 300°/sec</td>
</tr>
<tr>
<td>Gyro nonlinearity</td>
<td>0.2 %</td>
</tr>
<tr>
<td>Magnetometer range</td>
<td>± 2.5 Gauss</td>
</tr>
<tr>
<td>Magnetometer nonlinearity</td>
<td>0.4 %</td>
</tr>
<tr>
<td>Magnetometer bias stability</td>
<td>0.01 Gauss</td>
</tr>
<tr>
<td>A/D resolution</td>
<td>16 bits (SAR) oversampled to 17 bits</td>
</tr>
<tr>
<td>Orientation Accuracy</td>
<td>± 0.5° typical for static test conditions</td>
</tr>
<tr>
<td></td>
<td>± 2.0° typical for dynamic (cyclic) test conditions</td>
</tr>
<tr>
<td></td>
<td>&amp; for arbitrary orientation angles</td>
</tr>
<tr>
<td>Orientation resolution</td>
<td>&lt;0.1°</td>
</tr>
<tr>
<td>Repeatability</td>
<td>0.2°</td>
</tr>
<tr>
<td>Output modes</td>
<td>acceleration, angular rate and magnetic field</td>
</tr>
<tr>
<td></td>
<td>deltaAngle and deltaVelocity</td>
</tr>
<tr>
<td></td>
<td>Euler angles</td>
</tr>
<tr>
<td></td>
<td>rotation matrix</td>
</tr>
<tr>
<td>Interface options</td>
<td>standard: USB 2.0 or RS232</td>
</tr>
<tr>
<td></td>
<td>OEM: USB 2.0 / TTL serial (3.3 volts)</td>
</tr>
<tr>
<td>Data rate</td>
<td>1 Hz to 1,000 Hz</td>
</tr>
<tr>
<td>Filtering</td>
<td>sensors sampled at 30 kHz, digitally filtered</td>
</tr>
<tr>
<td></td>
<td>(user adjustable) and scaled into physical units;</td>
</tr>
<tr>
<td></td>
<td>coning and sculling integrals computed at 1 kHz.</td>
</tr>
<tr>
<td>Baud rate</td>
<td>115,200 baud to 921,600 baud</td>
</tr>
<tr>
<td>Supply voltage</td>
<td>standard: 4.4 to 6 volts [up to 15 volts operation</td>
</tr>
<tr>
<td></td>
<td>possible at limited temp range or low duty cycle]</td>
</tr>
<tr>
<td></td>
<td>OEM: 3.2 to 5.5 volts</td>
</tr>
<tr>
<td>Power consumption</td>
<td>80 mA @ 5 volts with USB</td>
</tr>
<tr>
<td>Connectors</td>
<td>micro-D9,</td>
</tr>
<tr>
<td></td>
<td>OEM: Samtec FTSH-105-01-F-D-K</td>
</tr>
<tr>
<td>Operating temp.</td>
<td>-40°C to +75°C (consult factory for higher</td>
</tr>
<tr>
<td></td>
<td>temperature operation)</td>
</tr>
<tr>
<td>Dimensions</td>
<td>44 mm x 25 mm x 11 mm - excluding mounting tabs,</td>
</tr>
<tr>
<td></td>
<td>width across tabs 37 mm, OEM: 38 mm x 24 mm x 12 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>18 grams RS-232 and USB, 11.5 grams OEM</td>
</tr>
<tr>
<td>Shock limit</td>
<td>1000 g (unpowered), 500g (powered)</td>
</tr>
</tbody>
</table>

*Accuracy and stability specifications obtained over operating temperatures of -40 to 70°C with known sine and step inputs, including angular rates of ± 300° per second.*
XL- MaxSonar® - WR1™ (MB7060)
XL- MaxSonar® - WRA1™ (MB7070)
Weather Resistant (IP67) Sonar Range Finder with High Power Output, Noise Rejection, Auto Calibration & Long-Range Narrow Detection Zone

This sensor provides very short to long-range detection and ranging, in a compact, robust PVC housing, designed to meet IP67 water intrusion, and matches standard electrical ¼” PCV pipe fittings. This sensor has a new high power output along with real-time auto calibration for changing conditions (temperature, voltage or acoustic or electrical noise) that ensure you receive the most reliable (in air) ranging data for every reading taken. The low power 3.3V – 5V operation detects objects from 0-cm to 765-cm (25.1 feet) and provides sonar range information from 20-cm out to 765-cm with 1-cm resolution. Objects from 0-cm to 20-cm range as 20-cm. The interface output formats included are pulse width output (MB7060), real-time analog voltage envelope (MB7070), analog voltage output, and serial digital output.

Features
- High acoustic power output
- Real-time auto calibration and noise rejection for every ranging cycle
- Precise narrow beam
- Continuously variable gain
- Object detection includes zero range objects
- 3.3V to 5V supply with very low average current draw
- Readings can occur up to every 100mS, (10-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- All interfaces are active simultaneously
  - Serial, 0 to Vcc
  - 9600Baud, 81N
  - Analog, (Vcc/1024) / cm
  - Pulse Width (MB7060)
  - Real-time analog envelope (MB7070)
  - Sensor operates at 42KHz

Benefits
- Acoustic and electrical noise resistance
- Reliable and stable range data
- Sensor dead zone virtually gone
- Low cost IP67 sensor
- Quality narrow beam characteristics
- Very low power ranger, excellent for multiple sensor or battery based systems
- Can be triggered externally or internally
- Sensor reports the range reading directly, frees up user processor
- Fast measurement cycle
- User can choose any of the sensor outputs
- No calibration requirement is perfect for when objects may be directly in front of the sensor during power up
- Easy hole mounting or mating with standard electrical fittings

Applications and Uses
- Tank level measurement
- Bin level measurement
- Proximity zone detection
- People detection
- Robot ranging sensor
- Autonomous navigation
- Environments with acoustic and electrical noise
- Multi-sensor arrays
- Distance measuring
- Long range object detection
- Wide beam sensitivity
- Users who prefer to process the analog voltage envelope (MB7070)
- Industrial sensor
- Physical drop-in upgrade for LV-MaxSonar-WR1 product, part numbers: MB7001

MaxBotix® Inc.
MaxBotix, MaxSonar, WR1 & WRA1 are trademarks of MaxBotix Inc.
XL-WR1™ XL-WRA1™ • v1.1b • 07/2009 • Patents 7,679,996
MB7060 & MB7070 Pin Out

Pin 1 - Leave open (or high) for serial output on the Pin 5 output. When Pin 1 is held low, the Pin 5 output sends a pulse (instead of serial data), suitable for low noise chaining.

Pin 2 - MB7060 (PW) This pin outputs a pulse width representation of range. To calculate distance, use the scale factor of 58uS per cm.

Pin 3 - (AE) This pin outputs the analog voltage envelope of the acoustic wave form.

Pin 4 - (RX) This pin is internally pulled high. The MB7060 & MB7070 will continually measure range and output if the pin is left unconnected or held high. If held low the MB7060 & MB7070 will stop ranging. Bring high 20uS or more for range reading.

Pin 5 - (TX) When Pin 1 is open or held high, the Pin 5 output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII character “P”, followed by three ASCII character digits representing the range in centimeters up to a maximum of 765, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232. When Pin 1 is held low, the Pin 5 output sends a single pulse, suitable for low noise chaining (no serial data).

V+ Operates on 3.3V - 5V. The average (and peak) current draw for 3.3V operation is 2.1mA (50mA peak) and 5V operation is 3.4mA (100mA peak) respectively. Peak current is used during sonar pulse transmit.

GND Return for the DC power supply. GND (& V+) must be ripple and noise free for best operation.

MB7060 & MB7070 Real-time Operation & Timing

175ms after power-up, the XL-MaxSonar® is ready to begin ranging. If Pin-4 is left open or held high (20uS or greater), the sensor will take a range reading. The XL-MaxSonar® checks the Pin-4 at the end of every cycle. Range data can be acquired once every 99mS. Each 99mS period starts by Pin-4 being high or open, after which the XL-MaxSonar® calibrates and calculates for 20.5mS, and after which, thirteen 42KHz waves are sent. Then for the MB7060, the pulse width (PW) Pin-2 is set high and when an object is detected the PW pin is pulled low, yet the PW pin can be held high for up to 44.4mS (i.e. 58uS * 765cm) if no target is detected. The most accurate range data presented to the user is taken from the PW output of the MB7060 product. For the MB7070 with analog envelop output, Pin-2 will show the real-time signal return information of the analog waveform. The remainder of the 99mS time (less 4.7mS) is spent adjusting the analog voltage to the correct level, (and allowing the high acoustic power to dissipate). During the last 4.7mS, the serial data is sent.

MB7060 & MB7070 Real-time Auto Calibration

Each time after the XL-MaxSonar® takes a range reading it calibrates itself. The sensor then uses this data to range objects. If the temperature, humidity, or applied voltage changes during sensor operation, the sensor will continue to function normally. The sensor does not apply compensation for the speed of sound change versus temperature to any range readings.
MB7060 & MB7070 Real-time Noise Rejection

While the XL-MaxSonar® is designed to operate in the presence of noise, best operation is obtained when noise strength is low and desired signal strength is high. Hence, the user is encouraged to mount the sensor in such a way that minimizes outside acoustic noise pickup. In addition, keep the DC power to the sensor free of noise. This will let the sensor deal with noise issues outside of the users direct control (in general, the sensor will still function well even if these things are ignored). Users are encouraged to test the sensor in their application to verify usability.

For every ranging cycle, individual filtering for that specific cycle is applied. In general, noise from regularly occurring periodic noise sources such as motors, fans, vibration, etc., will not falsely be detected as an object. This holds true even if the periodic noise increases or decreases (such as might occur in engine throttling or an increase/decrease of wind movement over the sensor). Even so, it is possible for sharp non-periodic noise sources to cause false target detection. In addition, *(because of dynamic range and signal to noise physics,) as the noise level increases, at first only small targets might be missed, but if noise increases to very high levels, it is likely that even large targets will be missed.

*HINT: In high noise environments, if needed, use 5V power to keep acoustic signal power high.

MB7060 & MB7070 Beam Characteristics

People detection requires high sensitivity, yet minimal side-lobes requires low sensitivity. The MB7060 and MB7070 balances the detection of people with minimal side-lobes. Sample results for measured beam patterns are shown below on a 30-inch grid. The detection pattern is shown for;
(A) 0.25-inch diameter dowel,
(B) 1-inch diameter dowel,
(C) 3.25-inch diameter dowel,
(D) 11-inch wide board moved left to right with the board parallel to the front sensor face and the sensor stationary. This shows the sensor’s range capability.

Note: The displayed beam width of (D) is a function of the specular nature of sonar and the shape of the board (i.e. flat mirror like) and should never be confused with actual sensor beam width.

MB7060 & MB7070 Mechanical Dimensions

MaxBotix® Inc.
MaxBotix, MaxSonar, WR1 & WRA1 are trademarks of MaxBotix Inc.
XL-WR1™ XL-WRA1™ • v1.1b • 07/2009 • Patents 7,679,995

Email: info@maxbotix.com
Web: www.maxbotix.com

Data Sheet Release: 04/28/10, pg. 3
D.7 COMMUNICATION AND COMPUTER
D.7.1 UGV ONBOARD COMPUTER
D.7.2 WIRELESS ROUTER
D.7.3 UGV FOUR-PORT ROUTER
Embedded System

### BRIK Series

Compact design for 3.5" SBC, excellent cooling function

**Application:**
Network device, VPN, Firewall

**Technical Data:**

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3V700A</td>
<td>1 LAN / VGA / USB</td>
</tr>
<tr>
<td>3V700D</td>
<td>3 LAN / VGA or COM</td>
</tr>
<tr>
<td>3I270A</td>
<td>Intel Atom N270, 1 LAN / VGA / COM</td>
</tr>
<tr>
<td>3I270D</td>
<td>Intel Atom N270, 4 LAN / VGA</td>
</tr>
</tbody>
</table>

**M/B Information:** (Please refer to M/B specification)

- **M/B Model:**
  - 3V700A
  - 3V700D
  - 3I270A
  - 3I270D

- **Description:**
  - 1 LAN / VGA / USB
  - 3 LAN / VGA or COM
  - Intel Atom N270, 1 LAN / VGA / COM
  - Intel Atom N270, 4 LAN / VGA

**Optional Accessory:**
- Wall mount kit

**Material:** Aluminum

**Color:** Blue & Silver

**VESA Mounting:** 75 x 75 mm

**Weight (Incl. M/B):**
- 1.3 Kgs (2.5" HDD ver.)
- 1.6 Kgs (3.5" HDD ver.)
- 1.8 Kgs (6 COM ver.)

**Storage Space:**
2.5" HDD or 3.5" HDD

**Operation Temperature:**

- 0 ~ 45 °C (with 2.5" HDD)
- 0 ~ 55 °C (with CF Card)

---

**Dimension:**

- **2.5" HDD ver.:**
  - LCM ver. (3V700D / 3I270D)
- **3.5" HDD ver.:**
  - 6 COM ver. (3I270A)

---

**Material:** Aluminum

**Color:** Blue & Silver

**VESA Mounting:** 75 x 75 mm

**Weight (Incl. M/B):**
- 1.3 Kgs (2.5" HDD ver.)
- 1.6 Kgs (3.5" HDD ver.)
- 1.8 Kgs (6 COM ver.)

**Storage Space:**
2.5" HDD or 3.5" HDD

**Operation Temperature:**

- 0 ~ 45 °C (with 2.5" HDD)
- 0 ~ 55 °C (with CF Card)

---

**Dimension:**

- **2.5" HDD ver.:**
  - 52 H x 187 W x 130 D (2.5" HDD ver.)
- **3.5" HDD ver.:**
  - 73 H x 187 W x 130 D (3.5" HDD ver.)
  - 80 H x 187 W x 130 D (6 COM ver.)

---

**Material:** Aluminum

**Color:** Blue & Silver

**VESA Mounting:** 75 x 75 mm

**Weight (Incl. M/B):**
- 1.3 Kgs (2.5" HDD ver.)
- 1.6 Kgs (3.5" HDD ver.)
- 1.8 Kgs (6 COM ver.)

**Storage Space:**
2.5" HDD or 3.5" HDD

**Operation Temperature:**

- 0 ~ 45 °C (with 2.5" HDD)
- 0 ~ 55 °C (with CF Card)

---

**Optional Accessory:**
- Wall mount kit
## Standard Spec.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU type</td>
<td>Intel Atom N270 1.6GHz processor</td>
</tr>
<tr>
<td>Front Side Bus</td>
<td>533 MHz</td>
</tr>
<tr>
<td>MB Chipset</td>
<td>Intel 945GSE + ICH7M (82801GBM)</td>
</tr>
<tr>
<td>Graphics</td>
<td>Integrated with Intel 945GSE Dynamic Video Memory Technology(DVMT 3.0)</td>
</tr>
<tr>
<td>Memory</td>
<td>On board DDRII module 1GB</td>
</tr>
<tr>
<td>NAND flash memory</td>
<td>Support Compact Flash card type II for ATA interface On board SSD 1/2/4/8Bytes(Option)</td>
</tr>
<tr>
<td>SATA</td>
<td>Two SATA ports with independent DMA operation supported</td>
</tr>
<tr>
<td>Audio</td>
<td>Intel High Definition Audio Specification Rev.1.0 Compliant Line-out / Mic-in / Lline-in support</td>
</tr>
<tr>
<td>LAN</td>
<td>1* Intel 82574L or Realtek RTL8111C 10/100/1000 Mbps</td>
</tr>
<tr>
<td>IO function</td>
<td>5* RS232 or RS485(option) + 1 RS422/485 or RS232(option)</td>
</tr>
<tr>
<td>USB</td>
<td>4* USB 2.0(external) &amp; 3 USB 2.0(internal)</td>
</tr>
<tr>
<td>DIO &amp; WDT</td>
<td>Hardware digital Input &amp; Output, 8xDI / 8xDO Hardware Watch Dog Timer, 0~255 sec programmable</td>
</tr>
<tr>
<td>Expansion interface</td>
<td>1* Mini PCI socket for only PCI rev 2.2 interface 2* PCIe mini card support USB and PCIe interface</td>
</tr>
<tr>
<td>Power</td>
<td>On board DC +12V±5% convert to +3V/+5V for system</td>
</tr>
<tr>
<td>Dimension</td>
<td>145 x 102 mm (3.5 inch) (MB) 80H x 187W x 130D mm(Chassis)</td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>0~50°C Please upgrade the hard drive to industrial grade for better temperature performance</td>
</tr>
<tr>
<td>Operating Humidity</td>
<td>5~95%, non-condensing</td>
</tr>
</tbody>
</table>

## Optional Choice

2. PCIe mini card wireless LAN
3. 1* Intel 82574L Giga LAN
4. Audio Amplifier 6-W/CH Into an 8-Q Load from a 12-V Supply
Ubiquiti RouterStation Pro

The embedded wireless dream machine

1 Pro Version Enhancements:

- 48V 802.3af Power Over Ethernet
- 4-Port Gigabit Ethernet Switch
- 128MB RAM
- On Board SDIO Support
- On Board, USB2.0, RS232/dB9
- On Board DC power jacks
- Breakthrough $79 USD MSRP!

2 Ships with Open-WRT Standard

RouterStation Pro ships with Open-WRT Kamikaze, the de facto standard for open-source wireless router software.

3 Features

- 802.3af 48V compatible
- Gigabit Ethernet Switch (4 ports)
- 128MB RAM
- 16MB Flash
- USB 2.0 Support
- SDIO Support
- 680MHz CPU (Option to overclock to 800)
- Robust ESD/Surge Protection for Industrial Applications
- Support for 3 Ubiquiti SR,XR,or SR71 radios
- Built in dB9 connector
- Easy Access to full GPIO signals

4 Pro Version Enhancements

The RouterStation Pro features several on board input enhancements including SDIO Support, USB2.0, RS232/dB9, and DC power jacks.

Source: http://www.ubnt.com/routerstation
# Switch Facts and Data

## Industrial 4-Port PoE Gigabit Ethernet Switch

### Ethernet Interface

<table>
<thead>
<tr>
<th>Media Interface</th>
<th>1000base-T, 100base-T, 10base-T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>IEEE802.3</td>
</tr>
<tr>
<td>Transmission Length</td>
<td>max. 100 m</td>
</tr>
</tbody>
</table>

### Connectors

<table>
<thead>
<tr>
<th>Connector Type</th>
<th>4x 8P8C Modular Jack (RJ45)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoE Injection</td>
<td>all 4 ports</td>
</tr>
</tbody>
</table>

### Features

<table>
<thead>
<tr>
<th>Supported Packet Size</th>
<th>Jumbo Frames up to 10 kByte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Negotiation</td>
<td>yes</td>
</tr>
<tr>
<td>Auto Crossover</td>
<td>yes</td>
</tr>
<tr>
<td>Auto Polarity</td>
<td>yes</td>
</tr>
</tbody>
</table>

### Electrical Data

<table>
<thead>
<tr>
<th>Power Supply</th>
<th>24 - 48 V DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Consumption</td>
<td>approx. 6 W</td>
</tr>
<tr>
<td>Supported Powered Devices</td>
<td>Class 0 (on 48 V PoE)</td>
</tr>
<tr>
<td>Feeding (per Port)</td>
<td>up to 15.4 W</td>
</tr>
<tr>
<td>Protective Functions</td>
<td>Protection against polarity reversal Disconnection of powered devices on  • Overload  • Overtemperature</td>
</tr>
</tbody>
</table>

### Mechanical Data

<table>
<thead>
<tr>
<th>Housing</th>
<th>Plastic, IP20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions</td>
<td>22.5 x 99 x 113.5 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>&lt; 150 g</td>
</tr>
</tbody>
</table>

### Environmental Data

<table>
<thead>
<tr>
<th>Operating Temperature</th>
<th>+5 °C ... +55 °C (+41 °F ... +131 °F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humidity</td>
<td>10% ... 90% non condensing</td>
</tr>
</tbody>
</table>

---

Due to technical data being fully checked, but accuracy of printed matter not guaranteed.

Baumer Optronic GmbH
Badstrasse 30
DE-01454 Radeberg, Germany
Phone +49 (0)3528 4386 0
Fax +49 (0)3528 4386 86
sales@baumeroptronic.com
www.baumer.com/cameras
D.8 POWER SOURCE AND DISTRIBUTION

D.8.1 24V DC-DC CONVERTER

D.8.2 5V AND 12V DC CONVERTERS

D.8.3 BATTERIES
Data Sheet
24V Input Maxi Family
DC-DC Converter Module

Features
- DC input range: 18 – 36 V
- Operation to 16 V at 75% power after startup
- Input surge withstand: 50 V for 100 ms
- DC output: 3.3 – 48 V
- Programmable output: 10 to 110%
- Regulation: ±0.25% no load to full load
- Efficiency: Up to 88%
- Maximum operating temp: 100°C, full load
- Power density: up to 80 W per cubic inch
- Height above board: 0.43 in. (10.9 mm)
- Parallelable, with N+M fault tolerance
- Low noise ZCS/ZVS architecture
- Isolated output

Product Overview
These DC-DC converter modules use advanced power processing, control and packaging technologies to provide the performance, flexibility, reliability and cost effectiveness of a mature power component. High frequency ZCS/ZVS switching provides high power density with low noise and high efficiency.

Applications
Industrial and process control, distributed power, medical, ATE, communications, defense, aerospace

Part Numbering
e.g. V24A12T400BL2

Absolute Maximum Ratings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Rating</th>
<th>Unit</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>+In to –In voltage</td>
<td>-0.5 to +53</td>
<td>Vdc</td>
<td></td>
</tr>
<tr>
<td>PC to –In voltage</td>
<td>-0.5 to +7.0</td>
<td>Vdc</td>
<td></td>
</tr>
<tr>
<td>PR to –In voltage</td>
<td>-0.5 to +7.0</td>
<td>Vdc</td>
<td></td>
</tr>
<tr>
<td>SC to –Out voltage</td>
<td>-0.5 to +1.5</td>
<td>Vdc</td>
<td></td>
</tr>
<tr>
<td>-Sense to –Out voltage</td>
<td>1.0</td>
<td>Vdc</td>
<td></td>
</tr>
<tr>
<td>+Out to –Out, -Sense to –Out</td>
<td></td>
<td></td>
<td>See Module Output Specifications</td>
</tr>
<tr>
<td>Isolation voltage</td>
<td>3000</td>
<td>Vrms</td>
<td>Test voltage</td>
</tr>
<tr>
<td>in to out</td>
<td>1500</td>
<td>Vrms</td>
<td>Test voltage</td>
</tr>
<tr>
<td>out to base</td>
<td>500</td>
<td>Vrms</td>
<td>Test voltage</td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>-55 to +100</td>
<td>°C</td>
<td>M-Grade</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>-65 to +125</td>
<td>°C</td>
<td>M-Grade</td>
</tr>
<tr>
<td>Pin soldering temperature</td>
<td>500 (280)</td>
<td>°F (°C)</td>
<td>&lt;5 sec; wave solder</td>
</tr>
<tr>
<td></td>
<td>750 (380)</td>
<td>°F (°C)</td>
<td>&lt;7 sec; hand solder</td>
</tr>
<tr>
<td>Mounting torque</td>
<td>5 (0.57)</td>
<td>in-lbs (N·m)</td>
<td>6 each</td>
</tr>
</tbody>
</table>

V24A

Output Voltage
3 V = 3.3 V
5 V = 8 V
9 V = 12 V
15 V = 24 V
25 V = 36 V
48 V = 48 V

Product Grade Temperatures (°C)

<table>
<thead>
<tr>
<th>Grade</th>
<th>Operating Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>-10 to +100 -20 to +125</td>
</tr>
<tr>
<td>C</td>
<td>-20 to +100 -40 to +125</td>
</tr>
<tr>
<td>T</td>
<td>-40 to +100 -60 to +125</td>
</tr>
<tr>
<td>H</td>
<td>-60 to +100 -80 to +125</td>
</tr>
<tr>
<td>M</td>
<td>-80 to +100 -100 to +125</td>
</tr>
</tbody>
</table>

Output Power

<table>
<thead>
<tr>
<th>Vout</th>
<th>Pout</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3 V</td>
<td>300 W, 264 W</td>
</tr>
<tr>
<td>5 V</td>
<td>300 W, 400 W</td>
</tr>
<tr>
<td>6 V</td>
<td>300 W, 400 W</td>
</tr>
<tr>
<td>8 V</td>
<td>300 W, 400 W</td>
</tr>
<tr>
<td>12 V</td>
<td>300 W, 400 W</td>
</tr>
<tr>
<td>15 V</td>
<td>300 W, 400 W</td>
</tr>
<tr>
<td>24 V</td>
<td>300 W, 400 W</td>
</tr>
<tr>
<td>28 V</td>
<td>300 W, 400 W</td>
</tr>
<tr>
<td>36 V</td>
<td>300 W, 400 W</td>
</tr>
<tr>
<td>48 V</td>
<td>300 W, 400 W</td>
</tr>
</tbody>
</table>

Pin Style
Blank: Short Tin/Lead
L: Long Tin/Lead
S: Short Modular
N: Long Modular
F: Short RoHS
G: Long RoHS

Baseplate
Blank: Slotted
3: Threaded
3: Through-hole

For a description of pin options, see page 11. Baseplate options include slotted flanges, threaded and through-hole. See page 12 for dimensions. For other package sizes and power levels, see the Mini (half size) and Micro (quarter size) datasheets.
LM2576/LM2576HV Series
SIMPLE SWITCHER® 3A Step-Down Voltage Regulator

General Description
The LM2576 series of regulators are monolithic integrated circuits that provide all the active functions for a step-down (buck) switching regulator, capable of driving 3A load with excellent line and load regulation. These devices are available in fixed output voltages of 3.3V, 5V, 12V, 15V, and an adjustable output version.

Requiring a minimum number of external components, these regulators are simple to use and include internal frequency compensation and a fixed-frequency oscillator.

The LM2576 series offers a high-efficiency replacement for popular three-terminal linear regulators. It substantially reduces the size of the heat sink, and in some cases no heat sink is required.

A standard series of inductors optimized for use with the LM2576 are available from several different manufacturers. This feature greatly simplifies the design of switch-mode power supplies.

Other features include a guaranteed ±4% tolerance on output voltage within specified input voltages and output load conditions, and ±10% on the oscillator frequency. External shutdown is included, featuring 50 µA (typical) standby current. The output switch includes cycle-by-cycle current limiting, as well as thermal shutdown for full protection under fault conditions.

Features
- 3.3V, 5V, 12V, 15V, and adjustable output versions
- Adjustable output voltage range, 1.23V to 37V (57V for HV version) ±4% max over line and load conditions
- Guaranteed 3A output current
- Wide input voltage range, 40V up to 60V for HV version
- Requires only 4 external components
- 52 kHz fixed frequency internal oscillator
- TTL shutdown capability, low power standby mode
- High efficiency
- Uses readily available standard inductors
- Thermal shutdown and current limit protection
- P+ Product Enhancement tested

Applications
- Simple high-efficiency step-down (buck) regulator
- Efficient pre-regulator for linear regulators
- On-card switching regulators
- Positive to negative converter (Buck Boost)

Typical Application (Fixed Output Voltage Versions)

![Figure 1](example_image.png)

SIMPLE SWITCHER® is a registered trademark of National Semiconductor Corporation.

© 1999 National Semiconductor Corporation
### Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

- **Maximum Supply Voltage**
  - LM2576: 45V
  - LM2576HV: 63V
- **ON/OFF Pin Input Voltage**
  - $-0.3 \leq V \leq +V_{IN}$
- **Output Voltage to Ground**
  - (Steady State)
  - $-1V$
- **Power Dissipation**
  - Internally Limited
- **Storage Temperature Range**
  - $-65^\circ C$ to $+150^\circ C$
- **Maximum Junction Temperature**
  - $150^\circ C$

### Minimum ESD Rating

($C = 100 \mu F, R = 1.5 k\Omega$)

- **2 kV**

### Operating Ratings

- **Temperature Range**
  - LM2576/LM2576HV: $-40^\circ C \leq T_J \leq +125^\circ C$
- **Supply Voltage**
  - LM2576: 40V
  - LM2576HV: 60V

### LM2576-3.3, LM2576HV-3.3 Electrical Characteristics

Specifications with standard type face are for $T_J = 25^\circ C$, and those with **boldface type** apply over full Operating Temperature Range.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Typ</th>
<th>Limit (Limits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage</td>
<td>$V_{IN} = 12V, I_{LOAD} = 0.5A$ Circuit of Figure 2</td>
<td>3.3</td>
<td>3.234 V (Min) 3.368 V (Max)</td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage LM2576</td>
<td>$6V \leq V_{IN} \leq 40V, 0.5A \leq I_{LOAD} \leq 3A$ Circuit of Figure 2</td>
<td>3.3</td>
<td>3.168/3.135 V (Min) 3.432/3.465 V (Max)</td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage LM2576HV</td>
<td>$6V \leq V_{IN} \leq 60V, 0.5A \leq I_{LOAD} \leq 3A$ Circuit of Figure 2</td>
<td>3.3</td>
<td>3.168/3.135 V (Min) 3.450/3.482 V (Max)</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Efficiency</td>
<td>$V_{IN} = 12V, I_{LOAD} = 3A$</td>
<td>75</td>
<td>%</td>
</tr>
</tbody>
</table>

### LM2576-5.0, LM2576HV-5.0 Electrical Characteristics

Specifications with standard type face are for $T_J = 25^\circ C$, and those with **boldface type** apply over full Operating Temperature Range.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Typ</th>
<th>Limit (Limits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage</td>
<td>$V_{IN} = 12V, I_{LOAD} = 0.5A$ Circuit of Figure 2</td>
<td>5.0</td>
<td>4.900 V (Min) 5.100 V (Max)</td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage LM2576</td>
<td>$0.5A \leq I_{LOAD} \leq 3A$, $8V \leq V_{IN} \leq 40V$ Circuit of Figure 2</td>
<td>5.0</td>
<td>4.800/4.750 V (Min) 5.200/5.250 V (Max)</td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage LM2576HV</td>
<td>$0.5A \leq I_{LOAD} \leq 3A$, $8V \leq V_{IN} \leq 60V$ Circuit of Figure 2</td>
<td>5.0</td>
<td>4.800/4.750 V (Min) 5.225/5.275 V (Max)</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Efficiency</td>
<td>$V_{IN} = 12V, I_{LOAD} = 3A$</td>
<td>77</td>
<td>%</td>
</tr>
</tbody>
</table>
**LM2576-12, LM2576HV-12**

Electrical Characteristics

Specifications with standard type face are for $T_J = 25^\circ$C, and those with **boldface** type apply over full Operating Temperature Range.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>LM2576-12</th>
<th>LM2576HV-12</th>
<th>Units (Limits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Typ</td>
<td>Limit</td>
<td>Typ</td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage</td>
<td>$V_{IN} = 25V, I_{LOAD} = 0.5A$</td>
<td>12</td>
<td>11.76</td>
<td>12.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circuit of Figure 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage</td>
<td>$0.5A \leq I_{LOAD} \leq 3A$</td>
<td>12</td>
<td>11.52/11.40</td>
<td>11.52/11.40</td>
</tr>
<tr>
<td></td>
<td>LM2576</td>
<td>$15V \leq V_{IN} \leq 40V$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circuit of Figure 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage</td>
<td>$0.5A \leq I_{LOAD} \leq 3A$</td>
<td>12</td>
<td>11.52/11.40</td>
<td>12.54/12.66</td>
</tr>
<tr>
<td></td>
<td>LM2576HV</td>
<td>$15V \leq V_{IN} \leq 60V$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circuit of Figure 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\eta$</td>
<td>Efficiency</td>
<td>$V_{IN} = 15V, I_{LOAD} = 3A$</td>
<td>88</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LM2576-15, LM2576HV-15**

Electrical Characteristics

Specifications with standard type face are for $T_J = 25^\circ$C, and those with **boldface** type apply over full Operating Temperature Range.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>LM2576-15</th>
<th>LM2576HV-15</th>
<th>Units (Limits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Typ</td>
<td>Limit</td>
<td>Typ</td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage</td>
<td>$V_{IN} = 25V, I_{LOAD} = 0.5A$</td>
<td>15</td>
<td>14.70</td>
<td>14.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circuit of Figure 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage</td>
<td>$0.5A \leq I_{LOAD} \leq 3A$</td>
<td>15</td>
<td>14.40/14.25</td>
<td>14.40/14.25</td>
</tr>
<tr>
<td></td>
<td>LM2576</td>
<td>$18V \leq V_{IN} \leq 40V$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circuit of Figure 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Output Voltage</td>
<td>$0.5A \leq I_{LOAD} \leq 3A$</td>
<td>15</td>
<td>14.40/14.25</td>
<td>15.60/15.75</td>
</tr>
<tr>
<td></td>
<td>LM2576HV</td>
<td>$18V \leq V_{IN} \leq 60V$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circuit of Figure 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\eta$</td>
<td>Efficiency</td>
<td>$V_{IN} = 18V, I_{LOAD} = 3A$</td>
<td>88</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LM2576-ADJ, LM2576HV-ADJ**

Electrical Characteristics

Specifications with standard type face are for $T_J = 25^\circ$C, and those with **boldface** type apply over full Operating Temperature Range.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>LM2576-ADJ</th>
<th>LM2576HV-ADJ</th>
<th>Units (Limits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Typ</td>
<td>Limit</td>
<td>Typ</td>
</tr>
<tr>
<td>$V_{OUT}$</td>
<td>Feedback Voltage</td>
<td>$V_{IN} = 12V, I_{LOAD} = 0.5A$</td>
<td>1.230</td>
<td>1.217</td>
<td>1.243</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V_{OUT} = 5V$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Circuit of Figure 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifications</td>
<td>2500mAh</td>
<td>3.7V</td>
<td>Constant Current Std. 1750mA</td>
<td>4.2V</td>
<td>Constant Voltage -20°C</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>-------</td>
<td>------------------------------</td>
<td>-------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Nominal Capacity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nominal Voltage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charging Method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charging Voltage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charging Current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charging Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ambient Temperature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weight (Max)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimensions (Max)</td>
<td>D</td>
<td>(D)</td>
<td></td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>Volumetric Energy Density</td>
<td>64.7mm</td>
<td>18.05mm</td>
<td>9.0mm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Dimensions (Typ.) of Bare Cell**

Discharged State after Assembling

Maximum size without tube
APPENDIX E : HAND CALCULATIONS

E.1 DRIVE ASSEMBLY MOTORS

E.1.1 SPEED REQUIREMENTS FOR WHEEL MOTORS

Assumptions
- UGV mass is 40kg
- UGV speed is 10km/h

Motor speed required

\[ 10 \text{km/h} = 2.78 \text{ m/s} \]

Speed of rotation of wheels = \((\text{speed of UGV})/(\text{circumference of wheels})\) = \(2.78/(2\pi(0.1))\) = 4.42 rotations per second = 265 rpm

Gear ratio is 18:1

Required motor speed = \(265 \times 18 = 4778 \text{ rpm}\)

E.1.2 BEARING LOAD CALCULATION

Assumptions
- Dynamic loading 10x greater than static loads
- UGV mass is 40kg
- Loading is evenly distributed between the 12 wheel motor bearings

Static load of bearings = 40 \times 9.81 = 392.4 N

static load of one bearing = 392.4 / 12 = 32.7 N

Dynamic load of one bearing = 327 N

BATTERY CAPACITY CALCULATION

This shows how the battery capacity was estimated using back of the envelope calculations.

Maximum Current Draw = \(x = 30\text{Amps}\)

Time = \(T = 45\text{ Minutes} = 0.75\text{ hours}\)

Capacity = \(xT = 30(0.75) = 22.5\text{ Amp hours}\)

However discharging batteries fully can damage the battery. To allow for this extra capacity is needed. The batteries will be 80% discharged after use.

New capacity = \(C' = C/0.8 = 22.5/0.8 = 28.125\text{ Amp hours}\)

Hence the batteries selected must have this capacity as a minimum.
Wheel Motor Torque Calculation

\[ \theta = \tan^{-1} \left( \frac{\theta}{\frac{L}{2}} \right) \]
\[ = 43.6^\circ \]

The middle wheels are at a pivot point so all torque from the gearbox is used to turn the vehicle. The corner wheels are not at a pivot point so not all the torque from the gearbox is used to turn the wheels.

\[ L = \sqrt{(0.2)^2 + (0.1)^2} \]
\[ = 0.24 \text{ m} \]
\[ \theta = \tan^{-1} \left( \frac{0.1}{0.2} \right) = 43.6^\circ \]

- \( F_m \) = motor force
- \( F_c \) = turning force
- \( R \) = radius of wheel = 0.1 m

\[ \frac{F_m}{F_c} > \cos \theta \]
Turning force measured using a spring gauge.

<table>
<thead>
<tr>
<th>Surface</th>
<th>Old Wheels</th>
<th>2 Old wheels + 4 Slick wheels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linoleum</td>
<td>117.2 N</td>
<td>98.1 N</td>
</tr>
<tr>
<td>Carpet</td>
<td>245.3 N</td>
<td>181.5 N</td>
</tr>
<tr>
<td>Concrete</td>
<td>196.2 N</td>
<td>137.5 N</td>
</tr>
</tbody>
</table>

Turning torque from corner wheels: \( F \times L = F_m \times \theta \times 1 \)

Turning torque from middle wheels: \( F_m \times (0.2 \times 1) \)

Total turning torque: \( 4F_m \times \theta + 2F_m \times (0.2) \)

\[ = F_m (4 \times 0.2 + 2 \times 0.2) \]

\[ = F_m (0.8 + 0.42) \]

\[ = 1.268F_m \]

For 2 old wheels and 4 slick tyre configuration:

<table>
<thead>
<tr>
<th>Surface</th>
<th>Turning Torque</th>
<th>Required Motor Force</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linoleum</td>
<td>128.2 N</td>
<td>158.7 N</td>
</tr>
<tr>
<td>Carpet</td>
<td>36.3 N</td>
<td>28.81 N</td>
</tr>
<tr>
<td>Concrete</td>
<td>271.5 N</td>
<td>318 N</td>
</tr>
</tbody>
</table>

For 32:1 Gearbox:

Motor force = \( \frac{\text{Motor torque} \times \text{gear ratio}}{\text{radius of wheel}} \)

\[ = \frac{96.9 \times 10^2 \times (32)}{0.1} \]

\[ = 3.1 N \]

This result shows that the selected gearbox will supply the required torque for skid steering on carpet.
For 18:1 gearbox:

\[
\text{motor force} = \frac{1.98}{0.1} = 19.8 \text{ N}
\]

For old six type configuration with old high grip wheels:

<table>
<thead>
<tr>
<th>Surface</th>
<th>Turning Torque</th>
<th>Required Motor Force</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linoleum</td>
<td>23.34 Nm</td>
<td>18.68 N</td>
</tr>
<tr>
<td>Carpet</td>
<td>40.95 Nm</td>
<td>38.94 N</td>
</tr>
<tr>
<td>Concrete</td>
<td>39.24 Nm</td>
<td>31.19 N</td>
</tr>
</tbody>
</table>
Maximum turning torque = 39.06 Nm

E.2 GPS

<table>
<thead>
<tr>
<th>Title: GPS Requirements</th>
<th>Project: MAGIC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed: For mapping accuracy of 300 mm, assume pos accuracy needs to be ≈ 100 m</td>
<td></td>
</tr>
<tr>
<td>⇒ AT 10 km/h = 10×10/3.6 = 2.78 m/s</td>
<td></td>
</tr>
<tr>
<td>&amp; 0.1 m accuracy</td>
<td></td>
</tr>
<tr>
<td>⇒ 2.8 Hz update</td>
<td></td>
</tr>
<tr>
<td>⇒ GPS must run approx 10×2.8 Hz = 28 Hz ( \geq 28 \text{ Hz} )</td>
<td></td>
</tr>
<tr>
<td>- 20 Hz is sufficient as it is rule of thumb and 20 Hz is the limit of what is reasonably available</td>
<td></td>
</tr>
<tr>
<td>Accuracy: assume ( \approx 100 \text{ mm} ) for sufficient accuracy</td>
<td></td>
</tr>
<tr>
<td>- Reasonable velocity accuracy is unlikely to be achievable</td>
<td></td>
</tr>
<tr>
<td>- Only required for control if system model</td>
<td>Not essential</td>
</tr>
</tbody>
</table>

E.3 INERTIAL MEASUREMENT UNIT
**IMU Requirements**

Assume:
- Want to perform mapping, accurate to 300mm cubes.
- Max range of lidar is 65m.
- Each lidar segment is 0.36° (no leeway).

**Orientation Accuracy:**

Assume 300mm = arclength

\[ 2\pi \frac{65}{360} = 1.015 \]

\[ \theta = \pm 1.9° \]

\[ \leq 2.0° \]

Accuracy improves for objects that are closer than max lidar range.

⇒ OK.

**Update Speeds**

Assume max turn rate of \( \pm 300°/\text{sec} \)

⇒ To be accurate to 2.0°,

Sample rate ≥ 180°/Hz.

**But**

Lidar scan rate is 25 Hz.

⇒ Orientation update should be 250 Hz (onboard IMU).

⇒ But only 25 Hz–50 Hz (to be safe) to onboard computer required.
Accelerometer Range

If accelerate in 2 seconds,

\[
\begin{align*}
\alpha &= \frac{10 \times 3600}{1000} \\
\alpha &= \frac{110 \times 1000}{3600 \times 2} = 1.39 \text{ m/s}^2 \\
\text{or} 0.14\, \text{g}
\end{align*}
\]

But accelerometer will be very noisy due to vibrations of moving platform. Measuring this is useful for control of mapping, but its effect can only be found using T & E.

Gyro Range

Requirements unknown.

If Lidar updates at 25 Hz

⇒ Max accurate turn rate for heading accuracy of 2° is 50°/sec.

Magnetometer Reqs

Earth's Magnetic Field: 0.31-0.6 gauss (Wikipedia)

Magnetic Field of UGV. Only determined by experiment. Too many variables:
- chassis
- motor
- wiring & shielding
- positioning...

⇒ Reqs unknown, ...
APPENDIX F : TEST PROCEDURES

This part of the Appendix includes the test procedures.

F.1 TEST PROCEDURES

Tests were required to ensure that the UGVs met the mechanical requirements set by the project contract. This section discusses briefly the test procedures that were carried out and the results. The design changes as a result of what was found from testing is also discussed briefly.

F.1.1 INITIAL DRIVE TEST

This test is to test if the UGV can skid steer and to test the overall maneuverability of the UGV.

Apparatus
-1 fully assembled and functional UGV
-Computer with EPOS studio installed

Method
1. Command the UGV to travel forwards along a painted straight line and check if the UGV turns to the right or left at all while moving forward.
2. Check how effectively the UGV turns on the spot and how maneuverable the UGV is overall.
3. Check if the Pan Tilt Unit Aim: (PTU) experiences much vibration when traveling on bumpy ground.

Results/Discussion

After long periods of testing it was found that the UGV was having trouble skid steering on carpet and concrete. The UGV would stop moving if skid steering was implemented by the operator since there was too much friction between the wheels and carpet surface. On lino and concrete surfaces the UGV could barely skid steer and for this reason new slick tyres were purchased and trailed to reduce the friction between the tyres and the ground. This will reduce the amount of torque required from the drive assembly for skid steering. Another problem that was discovered from testing the drive assembly is that the grub screws used to fasten the sleeve shafts to the motor shafts were failing in shear. The option that was chosen to solve this problem was to use a larger diameter grub screw. The grub screw diameter increased from 3mm to 4mm and has proven to be sufficient after subsequent tests. When driving over rough terrain or over bumps the PTU and UGV experienced significant vibrations. This is because there has not been any suspension implemented to dampen out vibrations or to isolate the UGV from vibrations. This is an area for future teams to investigate.

F.1.2 TILT MOTOR TEST

The aim of this test is to determine if the tilt motor is capable of tilting the PTU without exceeding the recommended current input.
**Apparatus**
- 1 fully assembled UGV
- computer with EPOS studio installed

**Method**
1. Using EPOS studio start with low current inputs of say 1A and check the response of the tilt motor.
2. Increase the current if it seems that the tilt motor is not producing enough power to tilt the PTU.
3. Be careful not to exceed the recommended current input for the tilt motor to avoid causing damage to the motor.

**Results/Discussion**
It was found that one tilt motor was faulty and after replacing the faulty tilt motor with a working tilt motor the response of the tilt motor was greatly improved. The faulty tilt motor was not producing enough torque even at high current inputs as high as 5A. After a working tilt motor was installed the tilt unit turned a lot smoother and required lower current inputs. With careful calibration the speed of rotation can be controlled and 90 degree tilt range can be achieved within a time less than 5 seconds.

**F.1.3 UGV WEIGHT TEST**
This test aims to determine the weight of the UGVs to see if the weight limit of 40kg has been achieved or not.

**Apparatus**
- 1 fully assembled and functional UGV
- 5m length of rope
- 100kg capacity spring gauge

**Method**
1. Strap the UGV up with the rope so that the spring gauge can be hooked onto the rope and hoist the UGV up.
2. Hoist the UGV up and take the reading on the spring gauge.
3. Since the UGV is quite heavy it would probably require two people to hoist the UGV up and another person to take the reading.

**Results/Discussion**
Weight of UGV sent down from Sydney = 42kg
UGV mass after milling of panels and removal of unnecessary hardware = 36.5 kg

This shows that after all the design improvements and removal of unnecessary hardware the mass limit of 40kg was achieved successfully since the UGV mass has been reduced from 42kg to 36.5kg.
F.1.4 UGV TURING FORCE TEST

The aim of this test is to determine the required force to skid steer the UGV.

Apparatus
- 1 fully assembled UGV
- 4 slick tyres
- 100kg capacity spring gauge

Method
1. Make sure that the old high grip tyres are secured firmly to the UGV chassis.
2. Hook the spring gauge to the wheel outside hub and pull on the spring gauge at 90° until the vehicle starts to rotate.
3. Test first on lino, then on carpet and concrete.
4. It is helpful to do 3 or more measurements and take an average or hold the reading by rotating the UGV on the spot by applying a constant force to the spring gauge.
5. Now removed the 4 corner wheels and replace with the 4 slick tyres that were purchased for trial.
6. Repeat steps 2-4 for the new wheel configuration.

Results/Discussion
The turning force was measured three times and was averaged. This is shown in Table F.1.

<table>
<thead>
<tr>
<th>Surface</th>
<th>Turning force (N)</th>
<th>Turning force (N)</th>
<th>% reduction in turning force required</th>
</tr>
</thead>
<tbody>
<tr>
<td>lino</td>
<td>117.7</td>
<td>98.1</td>
<td>16.7</td>
</tr>
<tr>
<td>carpet</td>
<td>245.5</td>
<td>181.5</td>
<td>26.1</td>
</tr>
<tr>
<td>concrete</td>
<td>196.2</td>
<td>137.5</td>
<td>29.9</td>
</tr>
</tbody>
</table>

From Table F.1 it can be seen that the turning force required is reduced significantly with the addition of slick tyres. Only one set of slick tyres were purchased for trial and with confirmation that these new tyres reduce the turning force enough to allow smoother turning a second set will be ordered for the second UGV.

F.1.5 DRIVE TEST (WITH DESIGN CHANGES)

The aim of this test to test the effectiveness of the new slick tyres and larger grub screws.

Apparatus
-1 fully assembled and operational UGV
- Computer with EPOS studio installed

Method
1. Using EPOS studio to navigate the UGV test the maneuverability of the UGV and whether smooth skid steering can be achieved on carpet, lino and concrete.
2. Record how the UGV responds to the difference surfaces.
3. Record the maximum speed of the UGV that can be determined using the encoders.

Results/Discussion
The UGV can easily skid steer on lino with minimal effort from the drive assembly and can rotate on the spot at a rate of roughly 15rpm. On carpet it is more difficult to skid steer and the drive assembly struggles the most when sharp turns are implemented by the operator. On concrete there is less resistance however, the motors require a higher current input than on lino. No grub screws failed during the test which has confirmed the larger grub screws are of a sufficient size. The maximum speed recorded was 10km/h.

F.1.6 BATTERY RUNTIME TEST
This test is used to determine how long the batteries can run for with the UGVs driving at 5km/h and all sensors recording data.

Apparatus
-1 fully assembled and operational UGV
-Ground Control Station

Method
1. Drive the UGV around with all the main sensors such as the LIDAR, camera and GPS recording data.
2. Stop the UGV when the batteries have been discharged to a level that the UGV can no longer operate.

Results/Discussion
After 3 hours of operation the UGV did not function properly anymore and testing was concluded. The batteries did not require charging at this time, however the wheel motors failed during the test and new motors have been ordered as a result. Since the batteries can last for periods of up to 3 hours the design requirement that the batteries had to last for a minimum time of 45 minutes was achieved.
APPENDIX G: SOFTWARE DEVELOPMENT

G.1 QNX OPERATING SYSTEM

The Operating System (OS) is the basis for all software that is implemented on board the UGV. The software is required to run with a low latency and so an OS must be chosen that allows for software to be implemented in an efficient manner suited to real time systems. A real time OS was chosen, which means that the latency of any process implemented is regular and predictable even though the system does not truly run in real time. For this purpose, Symmetry Innovations have sponsored this project and provided numerous free copies of QNX. QNX is an OS that is adept at this type of application. QNX has a unique microkernel and message passing architecture that makes it suitable for robust and stable real time implementations.

Some of the benefits of using an OS like QNX for a real time robot system, include the fact that QNX has low latency, is event driven, has low resource requirements and is stable. As is explained in Section ??, because QNX is a real-time OS, latency of interrupts and thread switching is minimised and there is sophisticated control over the scheduling of processes. QNX utilizes message passing to a great extent, which is an event driven type of control. This means that rather than polling or constantly checking for data or process updates, processes send a message and get reactivated upon the successful operation of the message. This significantly reduces the computational load. QNX has a very small memory and resources footprint for operation of the most basic OS processes. This allows the end user to choose the processes that they require, leaving more resources for key project processes. Additionally, due to its microkernel structure, QNX is a very stable system. A malfunction of a key process does not affect the functioning of the kernel and the process can be restarted automatically. This prevents process malfunctions from crashing other key processes or the entire system.

G.1.1 REAL-TIME OPERATING SYSTEM

A real-time OS is specifically designed to be set up for systems requiring the response of the system to occur within a known time, that must be controllable by the programmer. Various inputs must be responded to with different time periods, depending on the importance of that input for the operation of the overall system. Since some application and process requests are served almost immediately, there is a significant level of control available to the developer in terms of process priority. There are a vast number of priority levels, each of which can be set for any possible process. In particular, applications can have a higher priority than system processes. Real-time OSs accomplish this by making critical parts of the system code as minimal as possible, allowing for interrupts to processes or applications to have a critical status (Foran n.d.).

Scheduling is an important part of a real-time OS. It is usually more complex than for other OSs and allows for a highly sophisticated orchestration of priorities giving detailed control of how the applications are to function. Latency of interrupts and thread switching are minimal, however, the most important aspect of a real-time OS is its response speeds.
G.1.2 QNX

QNX is a microkernel-based OS. Thus the OS functions like a collection of small servers, whose functionality can be turned on or off depending on the situation. Most standard OSs have a so-called monolithic kernel, where the whole system is essentially one huge program whose parts have particular abilities. Thus, QNX can be tailored to the needs of this project and the OS utilises minimal resources on the Lex computer being used.

QNX is designed to be used in the embedded systems market because of its size and the fact that its a very efficient real-time OS. It works using a system of processes and threads that can run in parallel. QNX also has the ability to run on numerous modern CPUs manufactured for the embedded system market.

In QNX, all applications are made up of sequential processes, each of which is made up of any number of threads. A process is the actual execution of application instructions. The priority level of each process can be set and modified allowing very tight control over how applications function in QNX. A thread is an independent, schedulable stream of instructions that exist within a process. Threads within a process share the same memory space as the process. Each thread has its own function call stack. This is necessary so each thread can independently call functions and have their own local variables. (Foran n.d.)

G.1.3 MESSAGE PASSING

Communication between the different software functions is important to ensure the useability of any OS. Because QNX is designed to operate as a multi threaded system, communication cannot occur between different functions using normal methods such as global data structures. Instead a system is required that can operate with a time delay between the data being available and the recieving process being ready to accept it. This means that one process can continue working or stop operating while it is waiting for data, without the need for polling. Hence processes are event driven. An event is the occurrence of new data being available to a process and so when the data is available, then the process performs its function. QNX achieves this through the use of a proprietary system of message passing. (Foran, n.d.)

Figure G.1: Send - receive reply messaging (Foran, n.d.)
Message passing involves passing information from the client to the server, which then sends an acknowledgment message back (See Figure G.1). When the client has new information to send to the server, it sends a message which joins the servers message queue. The server may have multiple messages in its queue and these are read by the server according to their priority. When the message is read by the server, it sends an acknowledgment message to the client. In the time between the client sending the message and receiving its response, it becomes blocked. This means that it does not consume any CPU resources and does not continue operation until it has received the acknowledgment message from the server. This also occurs the other way around, such that when the server is ready to receive a message but the message has not been sent yet, it becomes blocked until it has a message join its message queue.

G.2 UBUNTU

The GCS requires an OS capable of handling the data received and the management requirements for up to eight UGV's, especially given that the UGVs have a limited processing capability on board. It must also be capable of running the HMI and the visualization of the progress of the UGVs.

Thus, the major requirements of the OS for the GCS are stability, reliability, ease of use, appropriate graphical capabilities, portability and cost. There are many available OSs that could be used, including Windows, MAC, Linux distributions and some other less well known systems such as QNX.

G.2.1 EASE OF USE

An important aspect to a project of this size is that each design requirement be completed as effectively and quickly as possible. An OS that is familiar to most people working on the project would reduce the development and set up time of some components of the GCS. Windows is a very familiar OS, however; for embedded applications, it is complex. MAC OS is very specific to Macintosh computers and experienced users are not as common. QNX is an option because it is being used on the UGV's however it would require many more drivers to be written and would involve significantly more time spent in coding especially for the HMI. Linux is a logical alternative to QNX for the GCS since it is based on a stable Unix kernel, however many of the available distributions have a much wider source of drivers and open source support already available than QNX. Ubuntu in particular is a very highly supported distribution of Linux and renowned for its ease of use, similarities to Windows for those with little experience of Unix systems and it has been used by members of the project team before. Ubuntu has support for both OpenCV and QT which are being used for machine vision and HMI development.

G.2.2 GRAPHICAL APPLICATIONS

Most OSs nowadays have sophisticated capabilities in the field of graphical applications. Of particular note, Windows, MAC OS and Ubuntu have regularly been used for this sort of an application. QNX, due to its focus on embedded systems does not have sophisticated graphical capabilities and so graphical applications are much harder to implement.

G.2.3 PORTABILITY

As the choice of hardware for the GCS may be restricted by certain factors such as sponsorship or expense, it is important to utilize an OS that is portable and can be utilized by numerous
types of hardware. For this reason, Ubuntu and QNX are definitely most appropriate. Windows is fairly portable and, MAC OS runs on machines manufactured only by Apple, limiting the options significantly.

G.2.4 COST

Cost is an important factor to consider in any project. Ubuntu is free and open source with good product support. QNX is available freely to academic and research groups and reasonable support is available while Windows and MAC OS are not free, but do have good support.

Table G.1 shows the decision matrix used to determine the most suitable OS for the GCS. It demonstrates these characteristics as well as taking into account weightings for each characteristic.

It is clear that Ubuntu is the most logical choice in this instance and hence will be used as the OS for the GCS.

Ubuntu is a popular computer OS. It is based on the original Debian GNU/Linux distribution and it is estimated to account for approximately 50% of Linux desktop usage and more for web server use (StatOwl 2010). Ubuntu is free and distributed as open source, and it has support provided for by a UK based company Canonical Ltd. The focus of Ubuntu is on ease of usability and other key functions such as installation. Updates are scheduled every half a year, making it a very stable system. Ubuntu is made up of numerous software packages, which are distributed as open source software according to the GNU General Public License (GNU GPL) and the GNU Lesser General Public License (GNU LGPL).

G.3 OPENCV

The number of software options available for computer vision processing is extremely large. Subsequently, the following discussion will examine the prerequisites in brief and outline which of these influences the final selection the most.

The prerequisites were as follows:

(i) Compatibility - The most prominent limitation on the software was that it must be compatible with the QNX Neutrino Real Time OS 6.4 on each UGV and the Ubuntu 10 OS used on the

<table>
<thead>
<tr>
<th>Function</th>
<th>Window</th>
<th>MAC</th>
<th>Ubuntu</th>
<th>QNX</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease Of Use</td>
<td>25</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>Graphical Capability</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>Portability</td>
<td>15</td>
<td>5</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Cost</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>80</td>
<td>65</td>
<td>100</td>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>
GCS. It was also desired to have the software compatible with the Windows OS, so that vision code could be learned and tested while the installation and setting up of the UGV and GCS computers was progressing. Thus, the software needed to be highly portable with minimal incompatibilities.

(ii) Language and Format – For the easiest integration into the UGV and GCS program code, the vision section needed to be the same programming language, C++, as the rest of the project code. A C++ computer vision library is the easiest option to achieve the programming language compatibility, rather than standalone programs, which may not include the desired image processing functionality. Furthermore, libraries are often far more efficient than standalone programs, because the only code that is included in the constructed program is what is essential.

(iii) Functionality – To locate, recognise and classify the OOI, the software must be capable of certain image processing functionalities. These include morphology, dilation, erosion, image and video extraction, codec compatibility, image and video saving capabilities, thresholding, contour plotting and other mathematical processes and transformations.

(iv) Experience – Once the previous requirements had been established, the experience of the programmer became the main selection factor. If a programmer has had experience with software before, significant amounts of time are saved because of familiarity with support resources, previously compiled code and particular subroutines. For the student with no experience in computer vision, the choice became that of the computer vision supervisor who had had experience with different computer vision software.

(v) Support Resources – The computer vision library should also have a large support base if possible, resulting in adequate and easily accessible examples, tutorials and help forums.

(vi) Cost – If possible, it was desired that the computer vision library would be available a public or open source license, so a license did not have to be purchased.

(vii) Size and System Requirements – Since the size of a UGV hard-drive was two gigabytes (GB), the computer vision library needed to be as small as possible to keep the memory requirements low. To a certain extent, some of the code will be split between the GCS and UGVs, the more processor intensive code being dealt with by the powerful GCS server. The code sections to be processed on UGVs and the GCS are yet to be decided.

OpenCV, an open source GNU licensed computer vision library, was the computer vision library that satisfied the previous factors most adequately. Programmed in C++, OpenCV code can be assembled by compilers in both QNX, Ubuntu and Windows OSs. Chris Madden had used the OpenCV library in the United Kingdom MOD Grand Challenge in 2008, subsequently having a good grasp of the library and the associated functionalities, as well as some sections of relevant code. From this experience, it was known that OpenCV had excellent support resources. An additional excellent resource was “Learning OpenCV”, the only known official guide for the OpenCV language, which has proven to be indispensable. The size of the library is approximately 30 megabytes, so this fits on the hard-drive more than adequately.

A further consideration was the use of the Intel Integrated Performance Primitives (IPP). The OpenCV library was originally developed by Intel for use on their computer processors as a marketing venture. To improve the processing efficiency, a number of low-level routines have been developed by Intel in a number of computer applications, including computer vision. The IPP functions are specifically designed to optimise performance on Intel-based chips (Intel, 2009), such as the 1.6 GHz Atom processor in the UGV computers. Unfortunately, the IPP functions need a Windows OS for the functions to be useful, whereas the actual OS used, QNX and Ubuntu, are both
Linux-based. It was decided the efficiency advantages provided by the IPP were insignificant in comparison to the advantages of both Linux-based OS.

G.4 CAMERA DATA FORMAT

A number of formats for transferring the camera data were considered. Even before the wireless network bandwidth became a severe limiter on the size of the data stream, compression was the main factor in the choice of camera data format in order to reduce the load as much as possible on the data transmission services. There were three main types of compression for visual data commonly used:

(i) JPEGs (Joint Photographic Experts Group standard format) are individually encoded frames. This means JPEGs have the lowest latency of the three common image streams, as the frame can be processed as soon as it is transferred (rather than having to wait for a video sequence to completely transfer) (Madden C 2010, pers. comm. 10 May 2010). The latency of a video stream is the time taken for a change in the recorded scene to change on the observer screen. The quality may be adjusted even further using JPEG compression artefacts to reduce the file sizes if necessary.

(ii) MJPEGs are Motion encoded JPEGs. Video frames are stored as JPEGs that are stitched to form a complete video sequence, where each frame is a keyframe. For low bandwidths, image resolution is given the highest priority, so images are dropped rather than reducing the quality of the images. This introduces latencies in the order of 100 ms.

(iii) MPEGs (Moving Picture Experts Group format) save memory space by using time encoding so only regions of the image that change significantly are encoded. Throughout most MPEG streams, a keyframe is created usually once every five to ten seconds (Page 2000). Only the pixel values that change significantly in the ensuing frames are recorded for those frames until the next keyframe is made. Iframes (Intermediate frames) are also used for further compression. The main advantage of MPEGs is the good quality to memory space ratio, but the latency is often in the order of 300 ms. In practice the MJPEG and MPEG streams are often done on the same chip, so often both have a delay in the order of 300 ms (Madden C 2010, pers. comm. 10 May 2010).

Due to the wireless bandwidth limitations, it was decided to use the JPEG format, as the bandwidth contribution used was highly controllable due to compression options as well as being able to have the most up-to-date display on the GCS due to the send-on-demand method used in the GCS image display.

G.5 IMAGE ANALYSIS USING MATLAB
Random samplings of pixel values were used to achieve recognition of objects of interest (OOI) earlier in the year. A more detailed examination of the colours associated with the objects of interest was required to ensure correct OOI detection in the various conditions observed by the camera, including sunlight, shadows and indoor lighting. The red objects were always much easier to distinguish from the natural background colours because of the distinctive colour that is not normally found in urban environments. The red bins and red humans feature a particularly large contribution to the red channel of colour in the RGB format. The navy blue coveralls are much more difficult to distinguish, because urban environments feature many similar colours. Many shadows have colours of similar darkness, and these shadows often contain a blue component due to ambient blue light from the sky.

In order to examine the particular characteristics of the colours of the OOI, it was decided to use the software Matlab, as this has a computer vision toolbox (from which various functions were used), is also highly recommended for visualising data for analysis and also the student had a reasonable amount of experience with the use of the software previously.

**General Procedure of Analysis**

The general procedure was to analyse images obtained from tests using regions of interest, or small test samples extracted from the image. The regions of interest examined were the sections of the image that were desired to be detected, as well as sections of images that produced ‘false positives’ or were incorrectly identified as desired objects during testing. These regions were compared for patterns and similarities in order to be able to generalise some rules to distinguish between the desired objects, and similar coloured objects that were found in the tested urban environments.

For example, the picture below from outdoor tests was used to examine the outdoor blue human in sunlight.
Taking a rectangle around the blue OOI, the following Region of Interest (ROI) was extracted:

Using Matlab, a histogram of the green, red and blue channel values was constructed:
The statistics of the RGB channels were also calculated, presented in the table below:

Table 1: Means and variances of the channel values for the sample region.

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>145.9</td>
<td>135.8</td>
<td>160.8</td>
</tr>
<tr>
<td>Variance</td>
<td>6801</td>
<td>7763</td>
<td>6048</td>
</tr>
</tbody>
</table>

The histogram shows many of the pixels had maximum channel values of 255 (since the channel values are 8-bit – $2^8-1 = 255$). Due to there being so much white in the ROI, the mean and variance are not very useful, since the mean averages between the bright white and dark navy colour. A smaller sample region needed to be examined to obtain the true characteristics of the blue.
Figure 1: A smaller sampling region of just the navy blue.

Figure 2: RGB histograms for each of the channels for the sample region.

These histograms supply much more useful information about the specific blue colour than the previous histograms because the vast majority of the sampling region is the target navy blue. The histograms show the variances differ as well as the means for each of the three channels, tabulated below. Note that the histograms have different scales making them difficult to compare. This was fixed in the final Matlab program.

Table 2: The means and variances for the smaller “navy blue only” sample region.

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>57.8</td>
<td>43.4</td>
<td>75.8</td>
</tr>
<tr>
<td>Variance</td>
<td>128.4</td>
<td>248.4</td>
<td>487.9</td>
</tr>
</tbody>
</table>
**Testing Results**

The sampling technique demonstrated has given a summary of the concepts required to be understood during testing. The following section records tests on a number of other samples of the objects in order to come to a conclusion on how best to distinguish the objects from other similarly coloured objects in an urban environment.

**Blue Objects of Interest**

**Blue in sunlight**

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>57.2</td>
<td>41.8</td>
<td>73.2</td>
</tr>
<tr>
<td><strong>Std. Dev.</strong></td>
<td>4.9</td>
<td>4.0</td>
<td>6.7</td>
</tr>
</tbody>
</table>

**Blue in shadow**

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>39.4</td>
<td>33.8</td>
<td>62.9</td>
</tr>
<tr>
<td><strong>Variance</strong></td>
<td>5.1</td>
<td>4.9</td>
<td>7.6</td>
</tr>
</tbody>
</table>
Blue Inside (Average Lighting)

Blue OOI inside (average lighting) | Red | Green | Blue
Mean | 30   | 27    | 57
Std. Dev | 8.6  | 7.7   | 13.4

Blue OOI inside (good lighting)

Blue OOI inside (good lighting) | Red | Green | Blue
Mean | 35   | 31    | 42
Std. Dev | 16.2 | 14.8  | 11.4
Blue inside (good lighting) (2)

Blue OOI inside (good lighting) (2) | Red | Green | Blue
--- | --- | --- | ---
Mean | 35 | 32 | 45
Std. Dev | 8.1 | 8.2 | 9.7

Blue OOI indoors dark leg

Blue OOI indoors dark leg | Red | Green | Blue
--- | --- | --- | ---
Mean | 17 | 15 | 33
Std. Dev | 3.2 | 3.2 | 6.4
Outside and Overcast

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>28</td>
<td>23</td>
<td>62</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>5.6</td>
<td>6.3</td>
<td>15.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outside with gain turned up</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>47</td>
<td>51</td>
<td>136</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>9.9</td>
<td>11.6</td>
<td>25.2</td>
</tr>
</tbody>
</table>

The mean values for each colour are higher as expected, but the variance is higher in all channels for blue in shadow than blue in the sun. Note that in the sun, each of the peaks are separated cleanly, but in the shadow, the red and green spectrums have similar values. This is due to the surrounding environment in some cases.
**Blue False Positives**

**Blue window** (special sun tinting gave it a dark blue colour)

<table>
<thead>
<tr>
<th>Blue Window</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>19.4</td>
<td>25.5</td>
<td>49.3</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>1.6</td>
<td>1.8</td>
<td>2.6</td>
</tr>
</tbody>
</table>

**Underside of Tree in Shadow**

<table>
<thead>
<tr>
<th>Underside of Tree in Shadow</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>26</td>
<td>21</td>
<td>35</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>5.7</td>
<td>5.8</td>
<td>6.6</td>
</tr>
</tbody>
</table>

**Black wrought iron fence**

<table>
<thead>
<tr>
<th>Wrought Iron Fence</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>20</td>
<td>19</td>
<td>33</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>4.1</td>
<td>4.0</td>
<td>6.4</td>
</tr>
</tbody>
</table>
## Final Results Comparison – Blue Actual Objects vs Blue False Positives

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Means</th>
<th>Ratios</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>Outside: in sun</td>
<td>57.2</td>
<td>41.8</td>
<td>73.2</td>
</tr>
<tr>
<td>Outside: in shadow</td>
<td>39.4</td>
<td>33.8</td>
<td>62.9</td>
</tr>
<tr>
<td>Inside: window lighting</td>
<td>30</td>
<td>27</td>
<td>57</td>
</tr>
<tr>
<td>Inside: good lighting</td>
<td>35</td>
<td>31</td>
<td>42</td>
</tr>
<tr>
<td>Inside: good lighting (2)</td>
<td>35</td>
<td>32</td>
<td>45</td>
</tr>
<tr>
<td>Inside: dark leg</td>
<td>17</td>
<td>15</td>
<td>33</td>
</tr>
<tr>
<td>Outside: overcast</td>
<td>28</td>
<td>23</td>
<td>62</td>
</tr>
<tr>
<td>Outside: gain turned up</td>
<td>47</td>
<td>51</td>
<td>136</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>17.00</td>
<td>15.00</td>
<td>33.00</td>
</tr>
<tr>
<td>Average Value</td>
<td>36.08</td>
<td>31.83</td>
<td>63.89</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>57.20</td>
<td>51.00</td>
<td>136.00</td>
</tr>
</tbody>
</table>

| Compared to false              |       |        |        |        |        |        |        |
|                                | Min <  | Min <  | Similar | Similar | Min <  | Min <  | Min <  |
|                                | Max    | Max    |         |         | Max    | Max    | All >  |
| Blue window                    | All >  | All >  |         |         |        |        |        |
| Underside of Tree in Shadow   | All >  | All >  |         |         |        |        |        |
| Wrought Iron Fence (black)    |        |        |         |         |        |        |        |
| Minimum Value                 | 19.40  | 19.00  | 33.00   | 1.35    | 1.67    | 0.76    | 1.60   |
| Average Value                 | 21.80  | 21.83  | 39.10   | 1.85    | 1.78    | 1.02    | 3.80   |
| Maximum Value                 | 26.00  | 25.50  | 49.30   | 2.54    | 1.93    | 1.24    | 5.70   |

From this table, the values for the code were determined. At this stage, any choices chosen for the blue objects outside have produced poor results. Note that in both the red and the blue colours, due to the noise of varying lighting conditions, the standard deviations recorded for each sample were found to give little distinction between the correctly detected and the false positives. The values that work well for the blue OOI indoors are summarised by this code:

```cpp
if ( (red > 15) && (green > 14) && (blue > 30) 
  && ((float)blue/red > 1.1) && ((float)blue/red < 2) 
  && ((float)blue/green > 1.2) && ((float)blue/green < 2.2) 
  && ((float)red/green > 0.9) && ((float)red/green < 1.5)) {
  temp_ptr[0]=255;  //White for greater than threshold
}
else{
  temp_ptr[0]=0;     //Black other
}
```

For the purposes of the second colour pass for colour confirmation, the thresholds used were that the sample blue mean must be greater than 35, and the sample mean should be greater than the means of the other two channels.
Red Objects of Interest

Red Bin

Red Bin Inside

<table>
<thead>
<tr>
<th>Red Bin Inside</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>204</td>
<td>73</td>
<td>73</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>18.2</td>
<td>15.4</td>
<td>15.8</td>
</tr>
</tbody>
</table>

Red bin inside good lighting

<table>
<thead>
<tr>
<th>Red bin inside good lighting</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>196</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>34.3</td>
<td>7.2</td>
<td>5.1</td>
</tr>
</tbody>
</table>
**Red Bin Outside Dark**

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Red</td>
<td>63</td>
<td>16</td>
<td>41</td>
</tr>
<tr>
<td>Mean Green</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean Blue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std. Dev Red</td>
<td>10.3</td>
<td>5.8</td>
<td>11.7</td>
</tr>
<tr>
<td>Std. Dev Green</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std. Dev Blue</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Red Bin Outside Sunny**

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Red</td>
<td>212</td>
<td>45.3</td>
<td>42</td>
</tr>
<tr>
<td>Mean Green</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean Blue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std. Dev Red</td>
<td>66.3</td>
<td>24.8</td>
<td>5.8</td>
</tr>
<tr>
<td>Std. Dev Green</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std. Dev Blue</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Red Human in Coveralls

Red Mobile OOI Sunny

Red Mobile OOI Sunny | Red | Green | Blue
---|---|---|---
Mean | 216 | 101 | 130
Std. Dev | 45.7 | 86.0 | 62.8

Red Mobile OOI Shadow

Red Mobile OOI Shadow | Red | Green | Blue
---|---|---|---
Mean | 142 | 34 | 79
Std. Dev | 46.7 | 7.8 | 13.2
Conclusions from this set of samples:

- The variance is not always a deciding factor. Note the extremely bright red bin in the sunlight, as many pixels seemed to have almost maximum brightness on the red channel, but also a large tail falling away, causing a larger variance.
- As discovered earlier in the year, the red component is always separate from the blue and green components which are usually very close.
- Note the wider spread of the green in the sunny sample. This is most probably because the bin was situated on grass (also in sunlight), and the grass reflected green light on the bin.
Red False Positives

Brick wall (1)

![Brick wall (1) image]

<table>
<thead>
<tr>
<th>Brick Wall (1)</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>199</td>
<td>92</td>
<td>78</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>25.6</td>
<td>16.0</td>
<td>11.3</td>
</tr>
</tbody>
</table>

Brick wall (2)

![Brick wall (2) image]

<table>
<thead>
<tr>
<th>Brick wall (2)</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>234</td>
<td>117</td>
<td>116</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>18.6</td>
<td>12.7</td>
<td>8.6</td>
</tr>
</tbody>
</table>

Red Downpipe

![Red Downpipe image]

<table>
<thead>
<tr>
<th>Red Downpipe</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>92</td>
<td>34</td>
<td>48</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>7.5</td>
<td>10.4</td>
<td>12.5</td>
</tr>
</tbody>
</table>
### Final Results Comparison – Red Actual Objects vs Red False Positives

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Means</th>
<th>Ratios</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>Red Bin Inside bright lighting</td>
<td>204</td>
<td>73</td>
<td>73</td>
</tr>
<tr>
<td>Red Bin Inside good lighting</td>
<td>196</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>Red Bin Outside Dark</td>
<td>63</td>
<td>16</td>
<td>41</td>
</tr>
<tr>
<td>Red Bin Outside Sunny</td>
<td>212</td>
<td>45.3</td>
<td>42</td>
</tr>
<tr>
<td>Red Mobile: Sunny</td>
<td>216</td>
<td>101</td>
<td>130</td>
</tr>
<tr>
<td>Red Mobile: Shadow</td>
<td>142</td>
<td>34</td>
<td>79</td>
</tr>
<tr>
<td>Red Mobile: Dark Shadow</td>
<td>101</td>
<td>27</td>
<td>68</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>63.00</td>
<td>16.00</td>
<td>26.00</td>
</tr>
<tr>
<td>Average Value</td>
<td>162.00</td>
<td>45.90</td>
<td>65.57</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>216.00</td>
<td>101.00</td>
<td>130.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compared to false</th>
<th>Similar</th>
<th>Similar</th>
<th>Similar</th>
<th>Ave &gt; Max</th>
<th>Ave &gt; Max</th>
<th>Ave &gt; Max</th>
<th>Ave &gt; Max</th>
<th>Ave &gt; Max</th>
</tr>
</thead>
</table>

### Red values for false positives

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Means</th>
<th>Ratios</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>Brick Wall (1)</td>
<td>199</td>
<td>92</td>
<td>78</td>
</tr>
<tr>
<td>Brick Wall (2)</td>
<td>234</td>
<td>117</td>
<td>116</td>
</tr>
<tr>
<td>Red downpipe</td>
<td>92</td>
<td>34</td>
<td>48</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>92.00</td>
<td>34.00</td>
<td>48.00</td>
</tr>
<tr>
<td>Average Value</td>
<td>175.00</td>
<td>81.00</td>
<td>80.67</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>234.00</td>
<td>117.00</td>
<td>116.00</td>
</tr>
</tbody>
</table>

The comparison of the actual red objects versus the red false positives shows there is very little difference in the raw channel values. However, with the means and variances, the average value for the actual objects was greater than the maximum values seen in the red false positives. Thus, in the code, no thresholds are set for the minimum red, green and blue values as was done for the blue, but just the ratios between channels were used. The variance differences between actual and false positive have not currently been exploited in the code. The values that work well for the blue OOI indoors are summarised by this code:

```c
if ( ((float)red/green > 2) && ((float)red/blue > 1.5) && ((float)blue/green > 1.05)){
    temp_ptr[0]=255;     //White for greater than threshold
} else{
    temp_ptr[0]=0;        //Black other
}
```

For the purposes of the second colour pass for colour confirmation, the thresholds used were that the sample blue mean must be greater than 35, and the sample mean should be greater than the means of the other two channels.
G.6 KALMAN FILTER COMPARISON

The following comparison of Kalman filters for IMU/GPS integration is included from Cullen et al. (2007, p33):
Table 3.1: Kalman Filter comparisons

<table>
<thead>
<tr>
<th>Kalman Filter Type</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>• Provides an <em>optimal</em> estimate of the states, minimising sensor noise; (Kelly, 1994b).</td>
<td>• Only applicable to systems of purely linear relationships; (Conner, 2000).</td>
</tr>
<tr>
<td></td>
<td>• Pre-calculation of filter matrices reduces computational burden; (Carlson, 2004; Rezaei and R., 2003).</td>
<td>• Noise must be zero-mean, white and Gaussian; (Rezaei and R., 2003).</td>
</tr>
<tr>
<td></td>
<td>• Easily understood and implemented</td>
<td></td>
</tr>
<tr>
<td>Extended</td>
<td>• Applicable to both linear and non-linear systems; (Conner, 2000).</td>
<td>• Not a formally optimal solution; (Zhao et al., 2003).</td>
</tr>
<tr>
<td></td>
<td>• Relatively easily understood and implemented</td>
<td>• Only reliable for systems which are almost linear on the time scale of update intervals; (Julier and JK, 1997).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Noise errors accumulate gradually over time; (Matia et al., 2006).</td>
</tr>
<tr>
<td>Unscented</td>
<td>• Noise is required to be Gaussian; (Julier and JK, 1997).</td>
<td>• Difficult to understand and implement</td>
</tr>
<tr>
<td></td>
<td>• Superior performance to the Extended Kalman Filter; (Julier and JK, 1997).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The mean and covariance properties of the noise are calculated through the unscented process and therefore need not be known in advance</td>
<td></td>
</tr>
<tr>
<td>Fuzzy</td>
<td>• Noise is not required be symmetrically distributed about the mean; (Matia et al., 2006).</td>
<td>• The process is computationally expensive</td>
</tr>
<tr>
<td></td>
<td>• Noise errors may be forced not to accumulate over time; (Matia et al., 2006)</td>
<td></td>
</tr>
</tbody>
</table>
MATLAB code for sample region analysis

%% Matlab Image Analysis
% Author: Mark Baulis
% Project: MAGIC 2010
clc
clear all

%% Read files to matrix
image = imread('image_02454.jpg');
% image = imread('RGBTest.bmp')
figure
imagesc(image)  % Display the original image
title('image')
whos('image') % Examine image properties (resolution and file size)

%% Extract Region of Interest (ROI)
% There are 3 sections of code for different situations

% (1) Put in actual values for ROI if you are analysing the same portion
%ROI = imcrop(image,[1110 720 55 75]); %Automatic crop of Redbin.jpg
% rect = [10 10 75 75];
% ROI = imcrop(image,rect);

% (2) Drag the cross-hair around the desired ROI and double click the left
% mouse button to define the ROI
[ROI rect] = imcrop(image);   % Used for manual definition of ROI
rect = round(rect);

% (3) If the whole image is to be analysed.
% ROI = image; % No crop needed for Test Images made in Paint with known colours

% Display the ROI
figure;
imagesc(ROI);    % Display the Region of Interest
title('ROI');

%% Histogram Analysis
ROI = double(ROI); % First convert to double precision format

[rows cols channels] = size(ROI);   % Extract matrix dimensions [could have used numel()]
channelCount = rows*cols;   % Extract number of entries given to each channel (R,G or B)
totalROI = ROI(:);   %All colour values listed sequentially (R, then G, then B in 1 column)
Bins = 0:1:255;
figure

%Display red histogram
redEntries = totalROI(1:channelCount,1);
Rcounts = hist(redEntries, Bins); % Extract R channel only
% subplot(3,1,1)
bar(Bins,Rcounts,1,'FaceColor', 'r', 'EdgeColor', 'r')
title('ROI Red Histogram')
hold on;
% Display green histogram
greenEntries = totalROI(channelCount+1:2*channelCount,1);
Gcounts = hist(greenEntries, Bins); % Extract G channel only
% subplot(3,1,2)
bar(Bins,Gcounts,1,'FaceColor', 'g', 'EdgeColor', 'g')
title('ROI Green Histogram')
hold on;

% Display blue histogram
blueEntries = totalROI(2*channelCount+1:3*channelCount,1);
Bcounts = hist(blueEntries, Bins); % Extract B channel only
% subplot(3,1,3)
bar(Bins,Bcounts,1,'FaceColor', 'b', 'EdgeColor', 'b')
title('ROI Blue Histogram')
hold off;

%% Mean and Variance
% Compute the mean and variance (std^2) of the formation

% Red Channel Mean and variance
mean_R = mean(redEntries);
var_R = var(redEntries);
StdDev_R = sqrt(var_R)
% sum_R_hist = sum(Rcounts.*Bins);
% mean_R_hist = sum_R_hist./channelCount % Mean is m = sum(x)/N
% var_R_hist = (sum(Rcounts.*Bins.^2)-channelCount*mean_R_hist^2)/(channelCount-1)
% Variance s^2 = sum((x-m)^2)/(N-1) = (sum(x^2)-N*m^2)/(N-1), x contains the data and
% N is the number of the data.

% Green Channel Mean and variance
mean_G = mean(greenEntries);
var_G = var(greenEntries);
StdDev_G = sqrt(var_G)
% sum_G_hist = sum(Gcounts.*Bins);
% mean_G_hist = sum_G_hist./channelCount
% var_G_hist = (sum(Gcounts.*Bins.^2)-channelCount*mean_G_hist^2)/(channelCount-1)

% Red Channel Mean and variance
mean_B = mean(blueEntries);
var_B = var(blueEntries);
StdDev_B = sqrt(var_B)
% sum_B_hist = sum(Bcounts.*Bins);
% mean_B_hist = sum_B_hist./channelCount
% var_B_hist = (sum(Bcounts.*Bins.^2)-channelCount*mean_B_hist^2)/(channelCount-1)
G.7 MAPPING

The mapping section went through many iterations prior to having the present form. In the early days of the project, the first map written was the exploration map. After this map was written, the entire software architecture of the MAGIC project was defined, and hence a new exploration map was written. However, the original exploration map displays the original thought and idea of the student responsible, and hence is given in this section.

G.7.0.1 ORIGINAL EXPLORATION MAP CONSTRUCTION METHODOLOGY

The original exploration map read the occupancy grid values generated from the physical maps and created an array that shows the areas that the UGV has explored and hasn’t explored. The criterion used to determine if a square has been explored or not was by checking the occupancy probability value of that square. If the probability value was between 194 and 255 or between 0 and 64, then that square was considered to have been explored. This is because probability values in the ranges mentioned are significantly greater than or less than the initial value of 128. Hence, it may be said with a fair amount of confidence that these have been explored. Squares that have a probability value in between the said ranges were considered unexplored, and data from further sweeps of the LiDAR is awaited. The probability values change with increase in confidence. Explored squares were assigned a value of ‘1’ and unexplored squares were assigned a value of ‘0’ in the exploration map.

G.7.0.2 ORIGINAL EXPLORATION MAP RESULTS

There were two versions of the original exploration Map code that were written. The first version did not cater for communication with the base station and hence it did not generate a global map. However, that was the first attempt at the code, and it worked well for generating a local exploration map. The results from this version are shown in Figure 6.1. When the original exploration map code was written, given the fact that the occupancy grid program was itself being tested, the LiDAR readings were not readily available for use in the testing of the Global Exploration Map code. In order to determine the accuracy of the Global Exploration Map code, it was tested using a simulated LiDAR reading. The result from the test performed using the first version of the code, that produced a local exploration map is shown in Figure G.2.

From Figure G.2, it can be seen that for a given input from the LiDAR, the code converts it into an array of 1’s and 0’s indicating areas that have been explored and areas that have not yet been explored. It should be noted that during the time this code was written, the LiDAR code was using a float array for the occupancy grid. Hence, the simulated LiDAR array was also of type float.

G.8 CONTROL

In this section, the full working of the path generation example given in the control chapter is given.

Let initial pose be \((x_o, y_o) = (3, 5)\) at orientation \(\theta_o = 10^\circ = 0.175\text{rad}\).

Let final pose be \((x_f, y_f) = (10, 8)\) at orientation \(\theta_f = 60^\circ = 1.047\text{rad}\).

Substitute initial position in Equation (G.1) to give:
Figure G.2: Test output from original exploration map

\[ 5 = a(3)^3 + b(3)^2 + c(3) + d \]

\[ \Rightarrow 27a + 9b + 3c + d = 5 \quad (G.1) \]

Substitute final position in Equation (9.6) to give:

\[ 8 = a(10)^3 + b(10)^2 + c(10) + d \]

\[ \Rightarrow 1000a + 100b + 10 + d = 8 \quad (G.2) \]

Now, the slope (or first derivative) of the curve is equal to the tangent of the angle between the curve and the positive x-axis.

\[ \frac{dy}{dx} = 3ax^2 + 2bx + c = \tan(\theta) \quad (G.3) \]

Substitute initial pose information in Equation (G.3) to give:

\[ 3a(9) + 2b(3) + c = \tan(0.175) \]
Substitute final pose information in Equation (G.3) to give:

\[ 3a(100) + 2b(10) + c = \tan(1.047) \]

\[ \Rightarrow 300a + 20b + c = 1.7312 \]  \hspace{1cm} (G.5)

Solving Equations (G.1),(G.2),(G.4) and (G.5):

\[
\begin{bmatrix}
  a \\
  b \\
  c \\
  d
\end{bmatrix}
= \begin{bmatrix}
  0.02144 \\
  -0.30716 \\
  1.44077 \\
  2.86316
\end{bmatrix}
\]
APPENDIX H : SOFTWARE CODE

This part of the Appendix includes the code written for this project.

H.1 UGV CODE

H.1.1 DRIVERS

H.1.1.1 LIDAR

UGV-DriverLIDAR.cpp

```cpp
#include "DriverLIDAR.h"

int main(int argc, char *argv[]) {
    // start the laser driver
    if (argc == 2) {
        DriverLIDAR driverLIDAR(argv[1]);
        driverLIDAR.run();
    }
    else {
        std::cout << "Usage ./lidarDriver <IPAddress>" << std::endl;
        std::cout << "EG: ./lidarDriver 172.16.0.52" << std::endl;
    }
    return 0;
}
```
DriverLIDAR.h

/*
 * DriverLIDAR.h
 *
 * Created on:  2/07/2010
 * Author:     Luke Sowter
 * Others:     Kim Bedson, Peter Hardy
 * Using:      libMPI
 * By:         Matthew Sanderson, Luke Sowter
 * Purpose:    Leuze rod4plus LIDAR Driver.
 *             Reads data to buffer.
 *             Extracts values from the buffer.
 *             Saves to values to shared memory.
 */

#ifndef DRIVERLASER_H_
#define DRIVERLASER_H_

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <iostream>
#include <string>
#include <unistd.h>
#include "data/DataLidarShared.h"
#include "TimeStamp.h"

#define PACKET_SIZE_LIDAR 1081
#define PORT 9008

using namespace std;
using namespace mpi;
using namespace mpi::data;

class DriverLIDAR
{
    public:
        DriverLIDAR(char * IP){ openDevice(IP); }
        ~DriverLIDAR() { close(fd); }
        public:
            void run();

    private:
        int openDevice(char * IP);
        int readDevice();
        int extractData();
        int writeToSharedMemory();
        private:
static DataLidarShared sharedLidar;
static DataLidar myLidar;
int fd;
unsigned char rawdata[PACKET_SIZE_LIDAR];
};

#endif /* DRIVERLASER_H */
DriverLIDAR.cpp

#include <assert.h>
#include "DriverLIDAR.h"

using namespace mpi::data;
using namespace mpi::msg;

DataLidar DriverLIDAR::myLidar;
DataLidarShared DriverLIDAR::sharedLidar;

void DriverLIDAR::run()
{
    while(1)
    {
        if(this->readDevice())
        {
            myLidar.setTimeStamp(TimeStamp::now());
            if(this->extractData())
            {
                //if we got here then new data is ready
                writeToSharedMemory();
                //cout << "Shared Memory written";
            }
        }
    }
}

int DriverLIDAR::openDevice(char * IP)
{
    struct sockaddr_in serverAddress;

    fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (fd < 0 ) {
        perror("Socket error" );
        cout<<"Socket error"<<endl;
        return(1);
    }

    memset( &serverAddress, 0, sizeof(serverAddress) );
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(PORT );

inet_pton(AF_INET, IP, &serverAddress.sin_addr);

if (connect(fd, (struct sockaddr *)&serverAddress,
    sizeof(serverAddress)) < 0 ) {
    perror("connect");
    cerr<<"Can not connect to LIDAR on IP "<<IP<<endl;
    cerr<<"Please check the IP of the LIDAR and restart this process"<<endl;
    return(1);
} else {
    cerr<<"Connected to LIDAR on "<<IP<<endl;
    return 1;
}

/**
 * Return <0 in failure;
 * Return 0 when record incomplete but no error;
 * Return 1 when successfully read complete record.
 */
int DriverLIDAR::readDevice(void)
{
    unsigned int zeroes = 0;
    unsigned int i = 0;
    unsigned char c;

    for (;;) {
        int ret = recv(fd, &c, 1, 0);
        if (ret < 0) {
            perror("recv");
            return ret;
        }
        if (c) {
            /* read a non-zero byte */
            if (2 == zeroes) {
                /* read a non-zero byte following two zero bytes */
                if (0xFF == c) {
                    /* read two zero bytes followed by a 0xFF byte */
                    /* output two bytes with value 0; eat 0xFF */
                    if (i + 2 > PACKET_SIZE_LIDAR) {
                        return -1; /* packet too long */
                    }
                    rawdata[i++] = 0;
                    rawdata[i++] = 0;
                } else {
                    /* read two zero bytes followed by a non-zero, non-0xFF byte */
                    /* we are at start of record, and just-read byte is first data byte */
                    i = 0;
                    rawdata[i++] = 0;
                    rawdata[i++] = 0;
                    rawdata[i++] = c;
                }
                zeroes = 0;
            } else {
                /* read a non-zero byte not following two zero bytes */
                assert(0 == zeroes || 1 == zeroes);
                /* must flush buffered-up zeroes */
            }
        }
    }
}
if (i + 1 + zeroes > PACKET_SIZE_LIDAR) {
    return -1; /* packet too long */
}
if (zeroes) {
    rawdata[i++] = 0;
} else {
    /* read a zero byte */
    zeroes++;
    assert(zeroes <= 3);
    if (3 == zeroes) {
        zeroes = 0;
        if (PACKET_SIZE_LIDAR == i) {
            /* read a record of the correct length */
            return 1;
        }
    /* read a record with incorrect length */
    return -1;
    }
}
assert(0 == zeroes);
}
return 0;

int DriverLIDAR::extractData()
{
    // TODO: can all of these ints be uchar?
    unsigned int hexc1,hexc2;
    unsigned int i,count;
    //Skip header (19 bytes)
    i=19;
    for(count=0; count < myLidar.SAMPLES_PER_SCAN; count++){
        // TODO: faster but possibly incorrect to:
        // myLidar[count] = ntohs(*((uint16_t *)&rawdata[i]));
        // TODO: Even better, do this on the GCS in class accessor methods
        hexc1=(unsigned int)rawdata[i];
        hexc1=hexc1<<8;
        hexc2=(unsigned int)rawdata[i+1];
        myLidar[count]=hexc1 | hexc2;
        i +=2;
    }
    return 1;
}

int DriverLIDAR::writeToSharedMemory()
{
    //write to sharedMemory
    sharedLidar.write(myLidar);
    return 0;
}
# H.1.1.2  EPOS

UGV-TestAppManualControl.cpp

```cpp
/*
 * ************************************************************************************
 * NOTE: THIS IS A TEST APPLICATION AND HAS NOT BE CREATED TO BE USED AS ANYTHING
 * MORE
 * ************************************************************************************
 */

/* UGV-TestAppManualMovement.cpp
 * Created on:  13/07/2010
 * Lasted updated: 4/10/2010
 * Author:     Luke Sowter & Chris Madden
 * Purpose:    Move UGV using keyboard. Adapted so that it is no longer just at a set
 * speed, but increments in 250 units
 */
*/
#include "DriverEPOS.h"
#include <termios.h>

void manualControlTest(DriverEPOS & driverEPOS);

char getmych(void) {
    char buf = 0;
    struct termios old = {0};
    if (tcgetattr(0, &old) < 0)
        perror("tcgetattr()");
    old.c_lflag &-= ~ICANON;
    old.c_lflag &-= ~ECHO;
    old.c_cc[VMIN] = 1;
    old.c_cc[VTIME] = 0;
    if (tcsetattr(0, TCSANOW, &old) < 0)
        perror("tcsetattr ICANON");
    if (read(0, &buf, 1) < 0)
        perror("read()");
    old.c_lflag |= ICANON;
    old.c_lflag |= ECHO;
    if (tcsetattr(0, TCSADRAIN, &old) < 0)
        perror("tcsetattr ~CANON");
    return (buf);
}

int main(int argc, char *argv[]) {
    DriverEPOS driverEPOS;
    //set operational mode of the Wheels (Velocity) and Pan and Tilt (position)
```
H.1 ugv code
driverEPOS.setOperationModeWheels(VELOCITY_MODE);
driverEPOS.setOperationModePanTilt(POSITION_MODE);

45
46

// Pan and Tilt should only be

operated in position mode
// Note that the Pan mode is currently commented out until the motor is fixed.

47
48

manualControlTest(driverEPOS);

49
50

return 0;

51
52

}

53
54
55

void displayOptions()

56

{

57

cout << " −−−−−−− Select a movement option : −−−−−−−" << endl;

58

cout << " −−−− Current speed : −−−−−−−−−−−−−−−−−−−−−" << endl;

59

cout << " −−−− Key: Movement option

<< endl;

60

cout << "

<< endl;

61

cout << "

62

cout << "

63

cout << "

64

cout << "

65

cout << "

66

cout << "

67

cout << "

68

cout << "

69

cout << "

70

cout << "

71

cout << "

72

"
W: INCREMENT SPEED
"
S : DECREMENT SPEED (REVERSE)
"
A: TURN MORE LEFT
"
D: TURN MORE RIGHT
"
0: STOP
"
N: NEW SPEED
"
I : TILT UP
"
K: TILT DOWN
"
H: DISPLAY HELP
"
Q: QUIT
"
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−"
−− Enter Command: " << endl;

<< endl;
<< endl;
<< endl;
<< endl;
<< endl;
<< endl;
<< endl;
<< endl;
<< endl;
<< endl;

}

73
74

void getNewSpeed(int &lspeed, int &rspeed)

75

{

76

int newSpeed;

77

cout << endl << " −− SET NEW SPEED −− Current Speeds : " << lspeed << " , " << rspeed <<

78

cout << " −− Enter new speeds ( l e f t then right ) from 100 to 3000: " << endl;;

79

cin >> newSpeed;

80

if(newSpeed<3000 && newSpeed>100) {

endl;

lspeed = newSpeed;

81

cout << " −− NEW SPEED SET −− " << lspeed << " −−" << endl;

82

} else {

83

cout << " −− INVAILD SPEED −− " << endl;

84
85

}

86

cin >> newSpeed;

87

if(newSpeed<3000 && newSpeed>100) {
rspeed = newSpeed;

88

cout << rspeed << " −−" << endl;

89

}

90

cout << " −− INVAILD SPEED −− " << endl;

91

}

92
93

else {

}

94
95

void manualControlTest(DriverEPOS &driverEPOS)

96

{

285


//test app to use manual control taking
//default speed
int lspeed = 0;
int rspeed = 0;
double tilt = 0; // These defaults need to be checked
double pan = 0; // These defaults need to be checked
int ret;
displayOptions();
while(1)
{
    //cin >> command; // should this changed to be scanf so you don't need to press enter?
    //system( "stty raw" ); // Changes the input to raw mode so no end of line needed
    command = getmych();
    //system( "stty cooked" ); // Returns to cooked mode in case system breaks
    switch(command)
    {
    //forward
        case 'W':
            case 'w': lspeed = lspeed + 200;
            rspeed = rspeed + 200;
            driverEPOS.setVelocityWheelsRPM(lspeed,rspeed);
            cout << 'w';
            //cout << endl << " -- Increase FORWARD speed --" << endl;
            break;
        case 'S':
            case 's': lspeed = lspeed - 200;
            rspeed = rspeed - 200;
            driverEPOS.setVelocityWheelsRPM(lspeed,rspeed);
            cout << 's';
            //cout << endl << " -- Decrement FORWARD speed --" << endl;
            break;
    //left
        case 'A':
            case 'a': lspeed = lspeed - 100;
            rspeed = rspeed + 100;
            driverEPOS.setVelocityWheelsRPM(lspeed,rspeed);
            cout << 'a';
            //cout << endl << " -- Turning LEFT --" << endl;
            break;
    //right
        case 'D':
            case 'd': lspeed = lspeed + 100;
            rspeed = rspeed - 100;
            driverEPOS.setVelocityWheelsRPM(lspeed,rspeed);
            cout << 'd';
            //cout << endl << " -- Turning RIGHT --" << endl;
            break;
    //stop
        case 'o':
```cpp
    case 'O':
    case 'o':
        driverEPOS.setVelocityWheelsRPM(0, 0);
        rspeed = 0;
        lspeed = 0;
        cout << endl << "— STOPPING —" << endl;
        break;
    //New speed
    case 'N':
    case 'n':
        getNewSpeed(lspeed, rspeed);
        break;
    //Tilt Up
    case 'I':
    case 'i':
        tilt = tilt + 5120;
        if (tilt > 20000) {
            tilt = MAXTILT;
            std::cout<<"limited to max TILT angle"<<std::endl;
        }
        driverEPOS.setPositionTilt(tilt); // Check if this works
        break;
    //Tilt Down
    case 'K':
    case 'k':
        tilt = tilt - 5120;
        if (tilt < -35000) {
            tilt = MINTILT;
            std::cout<<"limited to min TILT angle"<<std::endl;
        }
        driverEPOS.setPositionTilt(tilt); // Check if this works
        break;
    // Pan Left
    case 'J':
    case 'j':
        pan = pan + 1.0;
        if (pan > MAXPAN) {
            pan = MAXPAN;
        }
        //driverEPOS.setPositionPan(pan); // Check if this works
        break;
    // Pan Right
    case 'L':
    case 'l':
        pan = pan - 1.0;
        if (pan < MINPAN) {
            pan = MINPAN;
        }
        //driverEPOS.setPositionPan(pan); // Check if this works
        break;
    // Quit
    case 'Q':
    case 'q':
        driverEPOS.setVelocityWheelsMPS(0, 0);
        cout << endl << "— EXITING... —" << endl;
        return;
        break;
    case 'H':
    case 'h':
        displayOptions();
        break;
```
default: driverEPOS.setVelocityWheelsMPS(0,0);
    cout << endl << "— INVALID OPTION — STOPPING —" << endl << endl;
    break;
}

// Apply limiting to the speed values
if (rspeed > 5000) { rspeed = 5000;}
if (lspeed > 5000) { lspeed = 5000;}
if (rspeed < -5000) { rspeed = -5000;}
if (lspeed < -5000) { lspeed = -5000;}

std::cout << "New Speed (l,r): " << lspeed << ", " << rspeed << " RPM" << std::endl;
std::cout << "New Tilt Angle: " << tilt/1024 << " deg" << std::endl;
DriverEPOS.h

/*
 *********************************************************************************************************************/
/* NOTE: THIS IS A TEST APPLICATION AND HAS NOT BEEN CREATED TO BE USED AS ANYTHING MORE */
/* *********************************************************************************************************************/
/* DriverEPOS.h */
/* Created on: 8/07/2010 */
/* Author: Luke Sowter, Brendan O'Connell */
/* Others: Jourdain Bonfante, Richard Aplin */
/* Purpose: Maxon EPOS2 24/5 Driver: using RS232 to CAN Gateway */
/* */
#ifndef DRIVEREPOS_H_
#define DRIVEREPOS_H_

#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <iostream>
#include <math.h>
#include <unistd.h>
#include <sys/ioctl.h>

//Serial Port Settings
#define BAUDRATE B115200
#define DEVICE "/dev/ttyS5"
#define MPS_TO_RPM 1729.0442338267445

//Mode ID's
#define VELOCITY_MODE 0x00FE
#define PROFILE_VEL_MODE 0x0003
#define CURRENT_MODE 0x00FD
#define POSITION_MODE 0x00FF
#define PROFILE_POS_MODE 0x0001
#define HOMING_MODE 0x0006
#define PAN_MODE 0x00AA

//Pan Tilt
#define MAXPAN 8826.9230769 // Counts (90 degrees)
#define MINPAN -8826.9 // Counts (-90 degrees)
#define MAXTILT 20480 // 20 degrees in tilt counts
#define MINTILT -35840 // -35 degrees in tilt counts
#define DEG2COUNTERS 1024 // The amount of counts for ONE degree
#define DEG2COUNTERS_PAN 30.666666667

// RS232 constants
#define FRAMEHEADER 0x1103 // RS232
#define OPCODE 0x11 // RS232
#define OPLENGHT 0x03 // RS232

// UGV specific data
#define NUM_OF_NODES 8
enum NodeID {ERROR, RIGHT_FRONT, RIGHT_MID, RIGHT_REAR, LEFT_FRONT, LEFT_MID, LEFT_REAR, TILT, PAN};

using namespace std;

class DriverEPOS
{
    public:
        DriverEPOS();
        virtual ~DriverEPOS();

    public:
        int setVelocityWheelsRPM(int left, int right);
        int setVelocityWheelsMPS(double left, double right);
        int setOperationModeWheels(unsigned long opmode);
        int setOperationModeNode(int nodeID, unsigned long opmode);
        int setOperationModePanTilt(unsigned long opmode);

    private:
        int EPOSlink;

    private:
        // Communication
        int openDevice();
        int AcknowledgmentReceiveFromEPOS(void);
        int AcknowledgmentSendToEPOS(unsigned char* opcode);
        int writeObject(int index, int subindex, char nodeID, unsigned long data);
        unsigned short computeChecksum(unsigned short* pDataArray, unsigned short numberOfWords);

        // Device setup
        int enableEPOSNode(int nodeID);
        int enableEPOSAll();
        int disableEPOSNode(int nodeID);
        int disableEPOSAll();
        int clearFaultAll();
        int clearFaultNode(char nodeID);

        // Velocity Mode
        int setVelocityNode(int nodeID, unsigned long v);
        int setVelocityLeftSide(unsigned long v);
        int setVelocityRightSide(unsigned long v);
public: // Position Mode
   int setPositionNode(int nodeID, unsigned long pos);
   int setPositionPan(unsigned long pos);
   int setPositionTilt(unsigned long pos);

private: // E-Stop
   int setQuickStopNode(int nodeID);
   int setQuickStopWheels();
   int setQuickStopAll();
   int enableQuickStopAll();
};

#endif /* DRIVEREPOS_H_ */
DriverEPOS.cpp

/*
 * NOTE: THIS IS A TEST APPLICATION AND HAS NOT BE CREATED TO BE USED AS ANYTHING
 * MORE
 */

# include "DriverEPOS.h"

DriverEPOS::DriverEPOS(){
    openDevice();
}

DriverEPOS::~DriverEPOS(){
    close(EPOSlink);
}

int DriverEPOS::openDevice(){
    EPOSlink = open(DEVICE, O_RDWR | O_NOCTTY );
    if (EPOSlink <0) {perror("Error opening EPOS"); return(-1); } }

    // Get the current options for the port...
    tcgetattr(EPOSlink, &options);
    // Set the baud rates to 115200...
    cfsetispeed(&options, B115200);
    cfsetospeed(&options, B115200);
    // Enable the receiver and set local mode...
    options.c_cflag |= (CLOCAL | CREAD);
    options.c_cflag &=- (CRTSCTS); //~(IHFLOW|OHFLOW);
    options.c_cflag &=- CSTOPB;
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;
options.c_iflag &= ~(ICANON | ECHO | ECHOP | ISIG);
options.c_iflag &= ~(IXON | IXOFF | IXANY);
options.c_iflag &= ~(INLCR | IGNCR | ICRNL);
options.c_oflag &= ~OPOST;
options.c_cc[VMIN] = 1;
options.c_cc[VTIME] = 100;
tcflush(EPOSlink, TCIFLUSH); // RA Added by RA to flush serial port before use
// Set the new options for the port...
tcsetattr(EPOSlink, TCSANOW, &options);
// END of old Port Settings */

struct termios settings;
fcntl(EPOSlink, F_SETFL, 0);
memset(&settings, 0, sizeof(settings));
// Set the baud rate of the port
if(cfsetispeed(&settings, B115200) < 0)
{
    std::cout << "Failed to set the ispeed of the port" << std::endl;
    if(close(EPOSlink) < 0)
        std::cout << "Failed to close serial port because - " << strerror(errno) << std::endl;
        return(-30);
}

if(cfsetsospeed(&settings, B115200) < 0)
{
    std::cout << "Failed to set the ospeed of the port" << std::endl;
    if(close(EPOSlink) < 0)
        std::cout << "Failed to close serial port because - " << strerror(errno) << std::endl;
        return(-30);
}

// Enable the receiver and set local mode
settings.c_cflag |= (CLOCAL | CREAD);
settings.c_cflag &= ~PARENB;
settings.c_cflag &= ~CSTOPB;
settings.c_cflag &= ~CSIZE;
settings.c_cflag |= CS8;
settings.c_cc[VTIME] = 1; // Set the timeout. If no data is read in this time, then exit.
settings.c_cc[VMIN] = 100; // This sets the minimum number of bytes before returning

// Apply the new settings
if(tcsetattr(EPOSlink, TCSANOW, &settings) < 0)
{
    std::cout << "Failed to apply the new serial port settings because - " << strerror(errno) << std::endl;
    if(close(EPOSlink) < 0)
        std::cout << "Failed to close serial port because - " << strerror(errno) << std::endl;
```cpp
// Verify that the port settings is what we expect it to be
std::cout << "Checking baud rate is set to 115200 ..... ";

// Read the port information back
memset(&settings, 0, sizeof(settings));
if(tcgetattr(EPOSlink, &settings) < 0)
{
    std::cout << "Failed to read the serial port settings" << std::endl;
    if(close(EPOSlink) < 0)
    {
        std::cout << "Failed to close serial port because - " << strerror(errno) << std::endl;
        return(-30);
    }
}

// Check if the baud rate is what we set it too
if(cfgetispeed(&settings) != B115200)
{
    std::cout << "FALSE" << std::endl;
    return(-20);
} else {
    std::cout << "TRUE" << std::endl;
}

std::cout << "Enabling EPOS units and clearing any Faults" <<endl;
//enable devices and Quickstop
enableEPOSAll();
clearFaultAll();
cout << "Enabling QuickStop" << endl;
enableQuickStopAll();

return 1;

/* writeObject - This function writes commands to the EPOS units
* inputs: int index and int subindex - These denote the EPOS register to be written to
* nodeID - denotes which EPOS unit is going to be written to
* data - the data to be sent to the EPOS register
*/
int DriverEPOS::writeObject(int index, int subindex, char nodeID, unsigned long data)
{
    // Write information accross to a given buffer on the EPOS unit at nodeID
    unsigned short frame[OPLENGTH*2] = {0}; //command frame
    unsigned char sendBuf[OPLENGTH*4] = {0}; //send buffer
    unsigned char recvBuf[8] = {0}; //reply buffer
    //create frame for checksum calculation
    frame[0] = FRAMEHEADER; // header (LEN,OPCODE)
    frame[1] = index; // data
    frame[2] = ((nodeID) << 8) | subindex;
    frame[3] = data;
    frame[4] = data >> 8;
    frame[5] = computeChecksum(frame,OPLENGTH*2);
```

---

**APPENDIX H: SOFTWARE CODE**

```
//Convert to chars for transmission
memcpy(&sendBuf, &frame, OPLENGTH*4);

//TODO swap header and opcode..... dont know why....
sendBuf[0] = OPCODE;
sendBuf[1] = OPLENGTH;

//send read to write to EPOS
if (AcknowledgmentSendToEPOS(&sendBuf[0]) < 0) {return -1;}
//send command
int bytes2 = write(EPOSlink,&sendBuf[1], sizeof(sendBuf)-1);
tcflush(EPOSlink, TCIFLUSH);
//send write complete
if (AcknowledgmentSendToEPOS(&sendBuf[0]) < 0) {return -1;}

//Wait for EPOS ready to send byte
int bytesRead = read(EPOSlink,&revcBuf[0],1);
//send driver ready to receive
if (AcknowledgmentReceiveFromEPOS() < 0){return -1;}
//read reply
read(EPOSlink,&revcBuf[1],7);
//send read complete
if (AcknowledgmentReceiveFromEPOS() < 0){return -1;}
//TODO: nothing is done with this reply at this time
return 1;
}

unsigned short DriverEPOS::computeChecksum(unsigned short* pDataArray, unsigned short numberOfWords)
{
    unsigned short shifter, c;
    unsigned short carry;
    unsigned short CRC = 0;

    //Calculate pDataArray Word by Word
    while(numberOfWords--)
    {
        shifter = 0x8000; //Initialize BitX to Bit15
        c = *pDataArray++; //Copy next DataWord to c
        do
        {
            carry = CRC & 0x8000; //Check if Bit15 of CRC is set
            CRC <<= 1; //CRC = CRC * 2
            if(c & shifter) CRC++; //CRC = CRC + 1, if BitX is set in c
            if(carry) CRC ^= 0x1021; //CRC = CRC XOR G(x), if carry is true
            shifter >>= 1; //Set BitX to next lower Bit, shifter = shifter/2
        } while(shifter);

        return CRC;
    }

    return CRC;
}

int DriverEPOS::AcknowledgmentSendToEPOS(unsigned char* opcode)
{
    // Sends an Acknowledgement to the EPOS unit
//send opcode
if (write (EPOSlink, (const char *)&opcode, 1) < 0)
{
    perror("write serial port");
    return -1;
}

//wait for reply
unsigned char reply = NULL;
int ret = read(EPOSlink, &reply, 1);
if (ret < 0)
{
    perror("read");
    return -1;
}
if (reply == 'F') //Slave not ready to receive Data (Failed state).
{
    cerr<<"EPOS not ready to receive data Reply: "<<reply<<endl;
    return -1;
}
return 1;

int DriverEPOS::AcknowledgmentReceiveFromEPOS()
{
    // Recieve an Acknowledgement from the EPOS unit and return if it worked
    unsigned char ack = 0x4F;
    if (write (EPOSlink, &ack, 1) < 0)
    {
        perror("write");
        cerr<<"Sending acknowledgment to EPOS Failed"<<endl;
        return -1;
    }
    return 1;
}

int DriverEPOS::enableEPOSNode(int nodeID)
{
    // This function Shutsdown the EPOS unit before restarting it and setting the Quickstop functionality
    int ret;
    ret = writeObject(0x6040, 0x00, nodeID, 0x00000006); //Shutdown
    if (ret<0){cerr<<"enableEPOSNode (Shutdown) "<<nodeID<<" failed"<<endl;return ret;}
    ret = writeObject(0x6040, 0x00, nodeID, 0x0000000F); //Switch-On
    if (ret<0){cerr<<"enableEPOSNode (Switch-On) " <<nodeID<<" failed"<<endl;}
    ret = writeObject(0x6040, 0x00, nodeID, 0x0000);
    if (ret<0){cerr<<"clearFaultNode " <<nodeID<<" failed"<<endl;}
    ret = writeObject(0x6040, 0x00, nodeID, 0x0000010F);
    if (ret<0){cerr << "enable Quickstop" << nodeID << " failed" << endl;}
    return ret;
}

int DriverEPOS::enableEPOSAll()
{
    for(int i = 1;i<=NUM_OF_NODES;i++)
    {
        if(enableEPOSNode(i)<0){return -1;}}
```cpp
int DriverEPOS::disableEPOSNode(int nodeID)
{
    // Sets the EPOS unit at nodeID to shutdown and become inactive. This removes all power locking.
    int ret;
    ret = writeObject(0x6040, 0x00, nodeID, 0x00000006); // Shutdown
    if (ret<0) {cerr<<"disableEPOSNode "<<nodeID<<" failed"<<endl;}
    return ret;
}

int DriverEPOS::disableEPOSAll()
{
    // This loops through to disable all the EPOS units.
    for(int i=1; i<=NUM_OF_NODES; i++)
    {
        if(disableEPOSNode(i)<0) {return -1;}
    }
    return 1;
}

int DriverEPOS::setOperationModeNode(int nodeID, unsigned long opmode)
{
    // Set the Operational node for a specified node
    int ret;
    // cout << "Setting operation mode for " << nodeID << endl;
    ret = writeObject(0x6060, 0x00, nodeID, opmode);
    // cout << "Write complete" << endl;
    if (ret<0) {cerr<<"setOperationModeNode "<<nodeID<<" to opmode "<<opmode<<" failed"<<endl;}
    return ret;
}

int DriverEPOS::setOperationModeWheels(unsigned long opmode)
{
    // Passes the operational mode to all the wheels
    if(setOperationModeNode(LEFT_FRONT, opmode)<0) {return -1;}
    if(setOperationModeNode(RIGHT_FRONT, opmode)<0) {return -1;}
    if(setOperationModeNode(LEFT_MID, opmode)<0) {return -1;}
    if(setOperationModeNode(RIGHT_MID, opmode)<0) {return -1;}
    if(setOperationModeNode(LEFT_REAR, opmode)<0) {return -1;}
    if(setOperationModeNode(RIGHT_REAR, opmode)<0) {return -1;}
    return 1;
}

int DriverEPOS::setOperationModePanTilt(unsigned long opmode)
{
    // Sets the Operational mode of the Pan and Tilt Motors to
    // if(setOperationModeNode(PAN, opmode)<0) {return -1;}
    if(setOperationModeNode(TILT, opmode)<0) {return -1;}
    return 1;
}

int DriverEPOS::clearFaultNode(char nodeID)
{
    // Writes to the EPOS node to clear all the faults. Note this does not enable Quickstop
    int ret;
```
ret=writeObject(0x1003,0x00, nodeID, 0x0000);  
if(ret<0){cerr<<"clearFaultNode "<<nodeID<<" failed"<<endl;}
return ret;
}

int DriverEPOS::clearFaultAll()
{ // Clears any Fault state for all nodes
  for(int i = 1;i<=NUM_OF_NODES;i++)
    {
      if(clearFaultNode(i)<0){return -1;}
    }
  return 1;
}

int DriverEPOS::setVelocityWheelsRPM(int left, int right)
{ // Sets the Velocity of the wheels in RPM mode
  tcflush(EPOSlink, TCIOFLUSH); // Flush the input buffer before setting the latest wheel velocities
  if(setVelocityRightSide(right)<0){return -1;}
  if(setVelocityLeftSide(left)<0){return -1;}
  return 1;
}

int DriverEPOS::setVelocityWheelsMPS(double leftSpeed,double rightSpeed)
{ // Converts the Metre per second MPS wheel velocities to revolutions per minute RPM
  leftSpeed*=MPS_TO_RPM;
  rightSpeed*=MPS_TO_RPM;
  if(setVelocityWheelsRPM((int)leftSpeed,(int)rightSpeed)<0){return -1;}
  return 1;
}

int DriverEPOS::setVelocityNode(int nodeID,unsigned long v)
{ // Sets the given velocity v to node nodeID
  int ret;
  ret=this->writeObject(0x206B,0x00, nodeID,v);
  if(ret<0){cerr<<"setVelocityNode "<<nodeID<<" failed"<<endl;}
  return ret;
}

int DriverEPOS::setVelocityRightSide(unsigned long v)
{ // Sets the EPOS units controlling the Right side to Velocity mode
  if(setVelocityNode(RIGHT_FRONT,v)<0){return -1;}
  if(setVelocityNode(RIGHT_MID,v)<0){return -1;}
  if(setVelocityNode(RIGHT_REAR,v)<0){return -1;}
  return 1;
}

int DriverEPOS::setVelocityLeftSide(unsigned long v)
{ // Sets the EPOS units controlling the Right side to Velocity mode
  if(setVelocityNode(LEFT_FRONT,-v)<0){return -1;}
  if(setVelocityNode(LEFT_MID,-v)<0){return -1;}
  if(setVelocityNode(LEFT_REAR,-v)<0){return -1;}
  return 1;
}


// Position Mode
int DriverEPOS::setPositionNode(int nodeID, unsigned long pos)
{
    // Sets the position of the Node - note that double to long might not work properly.
    int ret;
    ret = this->writeObject(0x2062, 0x00, nodeID, pos); // Is this correct?
    if (ret < 0) { cerr << "setPositionNode " << nodeID << " failed" << endl; }
    return ret;
}

int DriverEPOS::setPositionPan(unsigned long int pos)
{
    // Note that pos is defined in encoder counts with rates per degrees defined in the header
    if (pos > MAXPAN) {
        pos = MAXPAN;
        std::cout << "Limited to max PAN angle" << std::endl;
    }
    if (pos < MINPAN) {
        pos = MINPAN;
        std::cout << "Limited to min PAN angle" << std::endl;
    }
    if (setPositionNode(PAN, pos) < 0) { return -1; }
    return 1;
}

int DriverEPOS::setPositionTilt(unsigned long int pos)
{
    // Note that pos is defined in encoder counts with rates per degrees defined in the header
    //if (pos < MAXTILT) {
    //    pos = MAXTILT;
    //    std::cout << "Limited to max TILT angle" << std::endl;
    //}
    //if (pos < MINTILT) {
    //    pos = MINTILT;
    //    std::cout << "Limited to min TILT angle" << std::endl;
    //}
    if (setPositionNode(TILT, pos) < 0) { return -1; }
    return 1;
}

int DriverEPOS::setQuickStopNode(int nodeID)
{
    // Sets the nodes into powered quickstop state. This stops any motion until cleared
    int ret;
    ret = writeObject(0x6040, 0x00, nodeID, 0x000B);
    if (ret < 0) { cerr << "setQuickStopNode " << nodeID << " failed" << endl; }
    return ret;
}

int DriverEPOS::setQuickStopWheels()
{
    // Sets all wheels to a powered stop. This will stop the wheels rolling on slopes
    tcflush(EPOSlink, TCI0FLUSH); // Flush the input buffer then stop all Wheel Nodes
    if (setQuickStopNode(LEFT_FRONT) < 0) { return -1; }
    if (setQuickStopNode(RIGHT_FRONT) < 0) { return -1; }
    if (setQuickStopNode(LEFT_MID) < 0) { return -1; }
    if (setQuickStopNode(RIGHT_MID) < 0) { return -1; }
    if (setQuickStopNode(LEFT_REAR) < 0) { return -1; }
}
int DriverEPOS::setQuickStopAll()
{
    // Applies a powered stop to all nodes stopping them moving
    tcflush(EPOSlink, TCIOFLUSH); // Flush the input buffer then stop all Nodes
    for(int i = 1; i <= NUM_OF_NODES; i++)
    {
        if(setQuickStopNode(i) < 0){
            return -1;
        }
    }
    return 1;
}

int DriverEPOS::enableQuickStopAll()
{
    // Enable the Quickstop functionality on all EPOS units
    int ret;
    tcflush(EPOSlink, TCIOFLUSH);
    for (int i = 1; i <= NUM_OF_NODES; i++)
    {
        ret = writeObject(0x6040, 0x00, i, 0x010F); // Does this enable Quickstop?
        if (ret < 0)
        {
            cerr << "enableQuickStopNode " << i << " failed" << endl;
            return -1;
        }
    }
}
H.1.1.3 GPS

UGV-DriverGPS.cpp

/*
 * UGV-DriverGPS.cpp
 *
 * Created on: 7/07/2010
 * Author: Luke Sowter
 * Modified: Anton Steketee (to add device path options ...)
 * Purpose: Simple program to run the GPS driver.
 * Last Changed: 2/10/2010
 */

#include "DriverGPS.h"

int main(int argc, char *argv[])
{
    // start the IMU driver
    DriverGPS myGPS;

    // If there is an input argument, use that as the device path, otherwise use a "standard" one
    if (argc != 2) {
        cerr << "Input Device required" << endl;
        cerr << "Assuming /dev/ttyS" << endl;
        cerr << "Usage: ./gps_Driver devicePath" << endl;
        cerr << "eg: ./gps_Driver /dev/ttyS3" << endl;
        myGPS.device = "/dev/ttyS3";
    } else {
        // set the device path
        myGPS.device = argv[1];
    }

    myGPS.start();
    myGPS.run();

    return 0;
}
/*
* DriverGPS.h
*
* Created on:  2/07/2010
* Author: Luke Sowter, Anton Steketee
* Using: libMPI
* By: Matthew Sanderson, Luke Sowter
* Purpose: NovAtel OEMV GPS Driver.
* Reads data to buffer.
* Extracts values from the buffer.
* Saves to values to shared memory.
* Modified: 5/8/10, AS: Change and update for Ubuntu
*/

#ifndef DRIVERGPS_H_
#define DRIVERGPS_H_

#include <stdlib.h>
#include <termios.h>
#include <stdio.h>
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <string>

#include "data/DataGPSShared.h"

#define VERBOSE 1

using namespace std;
using namespace mpi::data;
using namespace mpi::msg;

//Serial Port Settings
#define BAUDRATE B115200 //YOU NEED TO CHANGE THE BAUDRATE ON THE NOVATEL TO BE 115200 FOR FAST LOGS (eg 20Hz)

#define PACKET_SIZE_GPS 355
#define HEADERLENGTH 28 ; //length of ASCII header

//Driver specify struct's - All user data is stored in the DataGPS class object;
struct LogASCIIHeader { //This structure is common to all logs returned
    string logname;
    string port ;
string sequencenum;
string idletime;
string timestatus;
string week;
string GPSec;
string status;
string reserved;
string recvers;

};

struct LogUTM { //This is the specific information in a BestUTM log
    string status;
    string postype;
    int zonenum;
    string zoneletter;
    double northing;
    double easting;
    double hgt;
    float undulation;
    string datumid;
    float Nstdev;
    float Estdev;
    float Hstdev;
    string baseStationID;
    unsigned char NumberOfSatelliteInfo[4];
    unsigned char CRC[4];
};

struct LogVelocity { //This is information in BestVel log
    string status;
    string veltype;
    float latency;
    double age;
    double horspd;
    double trkgnd;
    double vertspd;
    unsigned char CRC[4];
};

//used to store the positions of each log in rawdata[]
struct logIndex{
    unsigned int startIndex;
    unsigned int stopIndex;
};

//Specific GPS commands
static char cmdUnLog[] = "unlog all\n";
static char cmdPosition[] = "log com1 bestutma ontime 0.1 \n";
static char cmdVelocity[] = "log com1 BESTVELA ontime 0.1 \n";
static char cmdRoverSetup[] = "interfacemode com1 novatel novatel off \n interfacemode com2 rtca rtca off \n"; //Recieve rtca and send novatel formats
```cpp
class DriverGPS {
public:
    DriverGPS() {};
    virtual ~DriverGPS();
    void start();
    public:
    void run();
    char * device;
private:
    int openDevice();
    int readDevice();
    int extractData();
    int writeToSharedMemory();
private:
    static DataGPSShared sharedGPS;
    static DataGPS myGPS;
    int fd;
    unsigned char rawdata[PACKET_SIZE_GPS];
private:
    int flagReadComplelteUTM;
    int flagReadComplelteVel;
    int flagExtractComplelteUTM;
    int flagExtractComplelteVel;
    logIndex rawdataIndexUTM;
    logIndex rawdataIndexVel;
    LogASCIIHeader HeaderUTM;
    LogASCIIHeader HeaderVel;
    LogUTM logUTM;
    LogVelocity logVel;
private:
    int extractDataHeader (LogASCIIHeader & header, char ** str);
    int extractDataUTM();
    int extractDataVel();
    string extractNextField (char ** currentBufferPostion);
    timespec timeOfRead;
};
#endif /* DRIVERGPS_H_ */
```
DriverGPS.cpp

#include "DriverGPS.h"

using namespace std;

DataGPS::shared = DriverGPS::sharedGPS;
DataGPS DriverGPS::myGPS;

void DriverGPS::start() {
  openDevice();
  flagReadCompleteUTM=0;
  flagReadCompleteVel=0;
  flagExtractCompleteUTM=0;
  flagExtractCompleteVel=0;
}

DriverGPS::~DriverGPS() {
  close(fd);
}

void DriverGPS::run() {
  while(1) {
    if(this->readDevice()) {
      if(this->extractData()) {
        if (writeToSharedMemory()) {
          static int i=0;
          //if (0 == (++i % 10)) {
            cerr.setf(ios_base::fixed);
            cerr << "DriverGPS: Updating shared memory; N,E,H=
            << myGPS.position.northing << "",
            << myGPS.position.easting << "",
            << myGPS.position.height << endl;
            //}
          }
        }
      }
    }
  }
}

int DriverGPS::openDevice()
{ // Wynand Marias put a large amount of error checking in at this point because the serial ports were giving us heaps of trouble.

  // Open Port
  fd = open(device, O_RDWR | O_NOCTTY);
  if (fd < 0) {
    perror(DEVICE);
    std::cout<<"Serial port did not open"<<std::endl;
  }
  // END Open Port

  // Get the current options for the port
  struct termios options;
  tcgetattr(fd, &options);
  // Set the baud rates to baudrate
  cfsetispeed(&options, BAUDRATE);
  cfsetospeed(&options, BAUDRATE);
  options.c_cc[VTIME] = 1; // Set the timeout. If no data is read in this time, then exit.
  options.c_cc[VMIN] = 255; // This sets the minimum number of bytes before returning

  // Set the new options for the port
  tcsetattr(fd, TCSANOW, &options);
  // END Port Settings

  int writecount = write(fd, cmdUnLog, strlen(cmdUnLog));
  if (writecount != (signed int)strlen(cmdUnLog)) {
    perror("write");
    cerr<<"Send cmdUnLog failed"<<endl;
    exit(1);
  }

  // Clear GPS buffer
  tcflush(fd, TCIOFLUSH);
  cout<<"GPS buffer flushed "<<endl;
  // END clear

  // Set GPS settings
  writecount = write(fd, cmdPosition, strlen(cmdPosition));
  if (writecount != (signed int)strlen(cmdPosition)) {
    perror("write");
    cerr<<"Send cmdPosition failed"<<endl;
    exit(1);
  }
  writecount = write(fd, cmdVelocity, strlen(cmdVelocity));
  if (writecount != (signed int)strlen(cmdVelocity)) {
    perror("write");
    cerr<<"Send cmdVelocity failed"<<endl;
    exit(1);
  }
  writecount = write(fd, cmdRoverSetup, strlen(cmdRoverSetup));
  if (writecount != (signed int)strlen(cmdRoverSetup)) {
perror("write");
cerr<<"Send roversetup failed"<<endl;
exit(1);

//END GPS settings

sleep(2);

//cout<<"GPS Device open and started successfully"<<endl;

return 1;

int DriverGPS::readDevice()
{
    #ifdef VERBOSE
    cout<<"readDevice Started"<<endl;
    #endif
    int readcount = 0;
    readcount = read(fd, rawdata, 1);

    //Look for the header byte
    while (rawdata[0] != '#') {
        readcount = read(fd, rawdata, 1);
    }

    //Then read the packet
    do {
        readcount += read(fd, rawdata+readcount, PACKET_SIZE_GPS -readcount -1); //This allows
        for space for the null terminater in response.
        if (!readcount) {
            perror("read");
cerr<<"Read during main read loop failed"<<endl;
            return -1;
        }
    } while (readcount < PACKET_SIZE_GPS-1);

    //Add EOF to end to we can find this later;
    rawdata[readcount] = '\0';

    // Get the time as soon as reading from serial is done. Not precise but$
    if( clock_gettime( CLOCK_MONOTONIC , &timeOfRead) == -1 ) {
        perror("clock_gettime");
        return 0;
    }

    #ifdef VERBOSE
    cout<< "ReadCount: " << readcount <<endl;
cout<<rawdata<<endl;
    #endif

    //Get log indexes - logs separated by '\n'
    char * str = (char*)&rawdata[0];
```c
int spanFirstLog = strcspn(str, "\n");
int spanSecondLog = strcspn(str + spanFirstLog, "\0");

#ifdef VERBOSE
    cout << "Second Log Size: " << spanSecondLog << endl;
#endif

// Check to see if two logs were received
if (spanFirstLog > 0)
{
    #ifdef VERBOSE
        cout << "First log before newline found – thinks it is a vel log" << endl;
    #endif

    // Set index’s
    if (spanFirstLog < spanSecondLog)
    {
        rawdataIndexVel.startIndex = 0;
        rawdataIndexVel.stopIndex = rawdataIndexVel.startIndex + spanFirstLog;
        rawdataIndexUTM.startIndex = spanFirstLog + 1;
        rawdataIndexUTM.stopIndex = rawdataIndexUTM.startIndex + spanSecondLog;
    }
    else if (spanSecondLog < spanFirstLog)
    {
        #ifdef VERBOSE
            cout << "second Log found" << endl;
        #endif

        rawdataIndexUTM.startIndex = 0;
        rawdataIndexUTM.stopIndex = rawdataIndexUTM.startIndex + spanFirstLog;
        rawdataIndexVel.startIndex = spanFirstLog + 1;
        rawdataIndexVel.stopIndex = rawdataIndexVel.startIndex + spanSecondLog;
    }
    else
    {
        // If one log is not bigger that the other than we did not get the right logs
        return -1;
    }

    // If we got here then we have TWO logs with one smaller that the other so test to ensure they are the correct logs
    // TEST header Vel
    str = (char *)&rawdata[rawdataIndexVel.startIndex + 1];
    if ("BESTVELA" == extractNextField(&str))
    {
        flagReadCompleteVel = 1;
    }

    // TEST header UTM
    str = (char *)&rawdata[rawdataIndexUTM.startIndex + 1];
    if ("BESTUTMA" == extractNextField(&str))
    {
        flagReadCompleteUTM = 1;
    }
}
else
{
    // If neither log is the one that we want, neither flagRead will be set and hence the read will be dumped
    // in the call to extract data which is next, hence return successful
    return 1;
}
```

```cpp
else {
    // could not find log delimiter
    return -1;
}
}

int DriverGPS::extractData() {
    #ifdef VERBOSE
    cout<<"EXTRACTING DATA "<<endl ;
    #endif
    int ret = -1;
    // Select which correct packet
    char * str;
    if (flagReadComplelteVel == 1) {
        // check CRC
        // TODO - CRC
        // if (CRCCal = CRCRead)
        // extract UTM data
        if (extractDataVel()) {
            // if extract is successful, set flag
            flagExtractComplelteVel = 1;
            ret = 1;
        }
        // reset Read flag as data has been used
        flagReadComplelteVel = 0;
    }
    if (flagReadComplelteUTM == 1) {
        // check CRC
        // TODO - CRC
        // if (CRCCal = CRCRead)
        // extract UTM data
        if (extractDataUTM() > 0) {
            // if extract is successful, set flag
            flagExtractComplelteUTM = 1;
            ret = 1;
        } else { ret = 0 ; }
        // reset Read flag as data has been used
        flagReadComplelteUTM = 0;
    }
    return ret;
}

int DriverGPS::writeToSharedMemory() {
    if ((flagExtractComplelteVel == 1) && (flagExtractComplelteUTM == 1)) {
        ...
    }
```

//Seeing as we are not yet utilising ALL GPS data, population the small bits that we need here
#ifdef verbose
    cout << "Updating shared mem values " << endl;
#endif
myGPS.position.northing = logUTM.northing;
myGPS.position.easting = logUTM.easting;
myGPS.position.height = logUTM.hgt;

myGPS.position_stdev.northing_stdev = logUTM.Nstdev;
myGPS.position_stdev.easting_stdev = logUTM.Estdev;
myGPS.position_stdev.height_stdev = logUTM.Hstdev;

myGPS.velocity.speed = logVel.horspd;
myGPS.velocity.heading = logVel.trkngnd;

//myGPS.velocity_stdev.speed_stdev =
//myGPS.velocity_stdev.heading_stdev =
myGPS.setTimeStamp(timeOfRead);

//write to sharedMemory
sharedGPS.write(myGPS);
//reset flags
flagExtractCompleteVel = 0;
flagExtractCompleteUTM = 0;

return 1;
}
//Else no complete data to write
return 0;

string DriverGPS::extractNextField (char ** currentBufferPostion) {
    /* Search through the input string, and find the next field before the comma.
       * Write this to output. Note the size of the output string must be declared */
    int j=0;
    string result;
    j=strcspn(*currentBufferPostion,"",");
    result.assign(*currentBufferPostion),j);
    *currentBufferPostion += j;
    //skip the limiting char ready for next time.
    (*currentBufferPostion)++;
    return result;
}

int DriverGPS::extractDataHeader (LogASCIIHeader & header, char ** str) {
    /* Work through the string given and split header bytes into result struct */
    header.logname = extractNextField(str);
    header.port = extractNextField(str);
    header.sequencenum = extractNextField(str);
    header.idletime = extractNextField(str);
    header.timestatus = extractNextField(str);
    header.week = extractNextField(str);
header.GPSec = extractNextField(str);
header.status = extractNextField(str);
header.reserved = extractNextField(str);
header.recvers = extractNextField(str);

return 1;
}

int DriverGPS::extractDataUTM () {
    /* Work through the string given and split header bytes into result struct */
    char *str = (char *)&rawdata[rawdataIndexUTM.startIndex+1];
    extractDataHeader(HeaderUTM, &str);

    //UTM log
    //Check to see if GPS has signal.
    logUTM.status = extractNextField(&str);
    if (logUTM.status.compare("SOL_COMPUTED") != 0)
        {
        std::cout<<"GPS: " << logUTM.status <<endl;
        return -1;
        }
    //If so extract log
    logUTM.postype = extractNextField(&str);
    logUTM.zonenum = atoi(extractNextField(&str).c_str());
    logUTM.zoneletter = extractNextField(&str);
    logUTM.northing = atof(extractNextField(&str).c_str());
    logUTM.easting = atof(extractNextField(&str).c_str());
    logUTM.hgt = atof(extractNextField(&str).c_str());
    logUTM.undulation = atoi(extractNextField(&str).c_str());
    logUTM.datumid = extractNextField(&str);
    logUTM.Nstdev = atof(extractNextField(&str).c_str());
    logUTM.Estdev = atof(extractNextField(&str).c_str());
    logUTM.Hstdev = atof(extractNextField(&str).c_str());
    logUTM.baseStationID = extractNextField(&str);
    logUTM.dif_age = atof(extractNextField(&str).c_str());
    logUTM.sol_age = atof(extractNextField(&str).c_str());

    //After this point we don't care about the rest of the fields. Could do something clever with this information.

    // temp = extractNextField(str);
    // logUTM.NumberOfSatelliteInfo = temp.data();
    // unsigned char CRC[4] ;

    // str = str + nextfield (str, NULL) + 1 ; //num sat
    // str = str + nextfield (str, NULL) + 1 ; //num sats in solution
    // str = str + nextfield (str, NULL) + 1 ; //num GPS plus GLONASS L1
    // str = str + nextfield (str, NULL) + 1 ; //num GPS plus GLONASS L1 and L2
    // str = str + nextfield (str, posresult.solstatus) + 1 ; //ext sol status
    // str = str + nextfield (str, NULL) + 1 ; //Reserved
    // str = str + nextfield (str, NULL) + 1 ; //sig mask
    // str = str + nextfield (str, posresult.CRC) + 1 ; */

    return 1;
int DriverGPS::extractDataVel (){
    /* Work through the string given and split header bytes into result struct */
    char * str = (char *)&rawdata[rawdataIndexVel.startIndex+1];
    extractDataHeader(HeaderVel, &str);

    // vel log
    logVel.status = extractNextField(&str);
    logVel.veltype = extractNextField(&str);
    logVel.latency = atof(extractNextField(&str).c_str());
    logVel.age = atof(extractNextField(&str).c_str());
    logVel.horspd = atof(extractNextField(&str).c_str());
    logVel.trkgnd = atof(extractNextField(&str).c_str());
    logVel.vertspd = atof(extractNextField(&str).c_str());

    // str = str + nextfield (str, junk) + 1 ;
    // str = str + nextfield (str, velresult.CRC) + 1 ;

    return 1;
}

H.1.1.4 IMU

UGV_DriverIMU.cpp

1 /*
2 * UGV-DriverIMU.cpp
3 *
4 * Created on: 2/07/2010
5 * Author: Luke Sowter
6 *
7 * Purpose: Simple program to run the IMU driver.
8 *
9 * Modified: AS - 6/8/10
10 */
11
12 #include "IMU_Driver.h"
13
14 DriverIMU * myIMU ;
15
16 void * updateSHMWrapper ( void * ){
17     myIMU->updateSHM () ;
18 }
19
20 int main(int argc, char *argv[])
21 {
22     pthread_t updateSHMThread ;
23     pthread_attr_t attr;
24
25     // start the IMU driver
26     myIMU = new DriverIMU;
27
28     // The input argument is the path of the device (eg /dev/tty ...)
29     // If no argument given, assume it is some standard
30     if (argc != 2 ) {
31         cerr << "Input Device required" << endl ;
32         cerr << "Number of Arguments " << argc << endl ;
33         cerr << "Usage: ./imu_Driver devicePath" << endl ;
34         cerr << "eg: ./imu_Driver /dev/ttyS5" << endl ;
35         myIMU->device = "/dev/ttyS1" ;
36     }
37     else {
38         // set the device path
39         myIMU->device = argv[1] ;
40     }
41
42     // open the serial port and start the IMU
43     myIMU->start();
44
45     // This should ensure that the thread is joinable. Doesn’t seem to work, so just start
46     // a thread to prevent error ...
47     pthread_attr_init(&attr) ;
48     pthread_attr_setdetachstate( &attr , PTHREAD_CREATE_JOINABLE) ;
49     pthread_create(&updateSHMThread, NULL, updateSHMWrapper, NULL );
50     while (1) {
51         //
// Read data and send to SHM
myIMU->getData();

return 0;
}
#ifndef IMU_DRIVER_H_
#define IMU_DRIVER_H_ 
#define SUCCESS 1
#define SYSTEM_ERROR -1 //use GetLastError to retrieve status
#define CHECKSUM_ERROR -10
#define SERIAL_READ_ERROR -20
#define SERIAL_WRITE_ERROR -30

#define IMU_COMMAND (unsigned char) 0x48
#define BAUDRATE B115200
#define PACKET_SIZE 67
#define VERBOSE 0

#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <fcntl.h>
#include <termios.h>
#include <iostream>
#include <sys/ioctl.h>
#include <pthread.h>

#include "data/IMU.h"
#include "data/IMUShared.h"

using namespace mpi::data :
using namespace std ;

class DriverIMU {
public:
    DriverIMU(void) { }
    ~DriverIMU(void); //Close IMU
    // Function to initialise IMU
    void start(void) ;
    // Function to get data from IMU
    bool getData(void) ;
    //Process reading
    int updateSHM(void) ;
}
316 | APPENDIX H: SOFTWARE CODE

```c

// Address of the device
char * device;

private:

// Convert set of bytes to float
float floatFromBytes(const unsigned char* pBytes);
// Calculate checksum
unsigned short Checksum(const unsigned char* pBytes, int count);
// Open serial port
int openSerialPort();
// read string from serial port
int readStr(int);
// reset IMU
int reset();

/** The serial port to read / write from */
int fd;
/* message passing channel id */
int coid;

// Result structures
static DataIMUShared sharedIMU;
static DataIMU myIMU;
timespec timeOfRead;

unsigned char response[255]; // response string
```

#endif // IMU_DRIVER_H_
** IMU_Driver.cpp

---

```cpp
#include "IMU_Driver.h"

DataIMU DriverIMU::myIMU;
DataIMUShared DriverIMU::sharedIMU;

// Setup mutex. This is used to stop the response being written or read to at the same time.
pthread_mutex_t aMutex = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t responseReady = PTHREAD_COND_INITIALIZER;

int DriverIMU::openSerialPort()
{
    // Open the serial port
    int port = open(device, O_RDWR | O_NOCTTY);

    // Check if the port was opened correctly
    if (port < 0)
    {
        std::cerr << "Cannot open serial port: " << device << ", because - " << strerror(errno) << std::endl;
        exit(EXIT_FAILURE);
    }

    // Clear all the serial port flags
    fcntl(port, F_SETFL, 0);

    // Create a structure to keep the port settings
    struct termios settings;
    memset(&settings, 0, sizeof(settings));

    // Read the current port configuration
    if(tcgetattr(port, &settings) < 0)
    {
        // Change serial port settings so that all the settings are cleared and then a set the
        // settings needed.

        // -------------------------------------------
        // CHANGELOG
        // 13/8/10 - AS
        // Decided to split into threaded implementation. I think it improved speed a little bit
        // but not massively.
        // The reliability of the program seems to suffer with too much computational load. Could
        // be a problem?
        // 25/8/10 - AS and WM
        // Changed serial port settings so that all the settings are cleared and then a set the
        // settings needed.
        */
```
```cpp
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  

std::cerr << "Failed to read the serial port settings because - " << strerror(errno) << std::endl;
if(close(port) < 0)
    std::cout << "Failed to close serial port because - " << strerror(errno) << std::endl;
exit(EXIT_FAILURE);
}

// Set the baud rate of the port
if(cfsetispeed(&settings, B115200) < 0)
{
    std::cerr << "Failed to set the ispeed of the port because - " << strerror(errno) << std::endl;
    if(close(port) < 0)
        std::cout << "Failed to close serial port because - " << strerror(errno) << std::endl;
        exit(EXIT_FAILURE);
}
if(cfsetospeed(&settings, B115200) < 0)
{
    std::cerr << "Failed to set the ospeed of the port because - " << strerror(errno) << std::endl;
    if(close(port) < 0)
        std::cout << "Failed to close serial port because - " << strerror(errno) << std::endl;
        exit(EXIT_FAILURE);
}

// Enable the receiver and set local mode
settings.c_cflag |= (CLOCAL | CREAD);
settings.c_cflag &= ~PARENB;
settings.c_cflag &= ~CSTOPB;
settings.c_cflag &= ~CSIZE;
settings.c_cflag |= CS8;
settings.c_cc[VTIME] = 1; // Set the timeout. If no data is read in this time, then exit.
settings.c_cc[VMIN] = 255; // This sets the minimum number of bytes before returning

// Apply the new settings
if(tcsetattr(port, TCSANOW, &settings) < 0)
{
    std::cout << "Failed to apply the new serial port settings because - " << strerror(errno) << std::endl;
    if(close(port) < 0)
        std::cout << "Failed to close serial port because - " << strerror(errno) << std::endl;
        exit(EXIT_FAILURE);
}

// Verify that the port settings is what we expect it to be
std::cout << "Checking baud rate is set to 115200 ..... ";
```
// Read the port information back
memset(&settings, 0, sizeof(settings));

// Read the current port configuration
if(tcgetattr(port, &settings) < 0)
{
    std::cerr << "Failed to read the serial port settings because − " << strerror(errno) << std::endl;
    if(close(port) < 0)
        std::cout << "Failed to close serial port because − " << strerror(errno) << std::endl;
    exit(EXIT_FAILURE);
}

// Print the message to indicate it works well
std::cout << "TRUE" << std::endl;

// Check if the baud rate is what we set it too
if(cfgetispeed(&settings) != B115200)
{
    std::cout << "FALSE" << std::endl;
    exit(EXIT_FAILURE);
}

// Return the created port
return port;

int DriverIMU::reset() {
    //Stop Continuous Mode - incase it is already running
    unsigned char sendReset[3] = {0xFE, 0x9E, 0x3A};
    if (write(fd, sendReset, 3) != 3)
    {
        perror("write");
        cerr<<"Can not reset IMU: Can not write to fd ("<<fd<<")"<<endl;
        cerr<<"Please check the IMU is plugged in and restart this process"<<endl;
        exit(1);
    }
    //END Reset

    //Closing and re-opening the port here seems to help. Not sure why?
    close(fd);
    //Small delay to allow time for the IMU to reset.
    sleep(1);
    //Open Port
    fd = open(device, O_RDWR | O_NOCTTY);
    std::cout <<"Device fd: " <<fd <<std::endl ;
    return 1;
}
float DriverIMU::floatFromBytes(const unsigned char* pBytes)
{
    float f = 0;
    ((unsigned char*)&f)[0] = pBytes[3];
    ((unsigned char*)&f)[1] = pBytes[2];
    ((unsigned char*)&f)[2] = pBytes[1];
    ((unsigned char*)&f)[3] = pBytes[0];
    // pBytes must point to an array of 4 bytes representing a floating point value
    return f;
}

unsigned short DriverIMU::Checksum(const unsigned char* pBytes, int count)
{
    unsigned short checksum = 0;
    int i = 0;
    for(i = 0; i < count; ++i)
    {
        checksum+=pBytes[i];
    }
    return checksum;
}

int DriverIMU::readStr(int hComPort)
{
    unsigned char responseRaw[255] ;
    unsigned long dwBytesRead = 0; //length of response
    int bytesAvail ;

    // Find out how much data is in the buffer
    ioctl(fd, FIONREAD, &bytesAvail) ;
    while(bytesAvail<PACKET_SIZE) {
        if (VERBOSE) {
            // This should be 0 if we are keeping up with the read. If it is larger then the
            // programme is behind.
            cout << "Bytes Avail: " << bytesAvail << endl ;
        }
        usleep(5000) ;
        ioctl(fd, FIONREAD, &bytesAvail) ;
    }
    // If there is no data or more than 67 bytes, change the value to 1 or 67
    if (bytesAvail<1) {
        bytesAvail = 1 ;
    } else if (bytesAvail>255) {
        bytesAvail = 255 ;
    }

    // Read the amount of data available
    dwBytesRead = read(hComPort, responseRaw, bytesAvail) ;
    if (VERBOSE) {
        cout << "Bytes Read: " << dwBytesRead << endl ;
    }
}
if (dwBytesRead > bytesAvail || dwBytesRead == 0) {
    cout << "Read error" << endl;
    return (0);
}

// Check that it is the correct header byte.
// Start searching for header in the last 2 lots of PACKET_SIZE. This should get the latest packet read.
//unsigned int i = dwBytesRead - 2*PACKET_SIZE ;
//if (i < 0) { i = 0 ; }
int i = 0 ;
while (responseRaw[i] != IMU_COMMAND) {
    i++ ;
    if (i == dwBytesRead) {
        if (VERBOSE) {
            std::cout << "Header Byte Not Found " << std::endl ;
        }
        return 0;
    }
}
// When the header byte has been found, copy that data to the buf for processing
pthread_mutex_lock(&aMutex) ;
memcpy(response, responseRaw+i, dwBytesRead-i ) ;
dwBytesRead -= i ;

// Read up to 67 bytes if this has not already happened
while (dwBytesRead < PACKET_SIZE) {
    dwBytesRead += read(hComPort, response+dwBytesRead, PACKET_SIZE - dwBytesRead) ;
}

// Get the time as soon as reading from serial is done. Not precise but consistent
if(clock_gettime( CLOCK_MONOTONIC , &timeOfRead) == -1 ) {
    perror("clock_gettime");
    return 0;
}

// Signal to say response is ready.
pthread_cond_signal(&responseReady) ;
pthread_mutex_unlock(&aMutex) ;

return 1 ;

int DriverIMU::updateSHM( ){

// When the data has been found, this then converts and writes to shared memory
unsigned short wChecksum = 0; //Value of checksum from result
unsigned short wCalculatedCheckSum = 0; // Value of checksum calculated by the program
while(1) {
    pthread_mutex_lock(&aMutex) ;
    pthread_cond_wait(&responseReady, &aMutex) ;
    wChecksum = (unsigned int)response[PACKET_SIZE-2]<<8;
    wChecksum = wChecksum | (unsigned int)response[PACKET_SIZE-1];
}
wCalculatedCheckSum = Checksum(&response[0], PACKET_SIZE-2); //calculate the checksum, 77 = 79-2 don't include the checksum bytes
if(wChecksum != wCalculatedCheckSum)
{
    if (VERBOSE) {
        std::cout << "Checksum Error " << wChecksum << " " <<wCalculatedCheckSum<<
        std::endl ;
    }
    pthread_mutex_unlock(&aMutex) ;
    //return (CHECKSUM_ERROR) ;
} else if(response[0] != IMU_COMMAND){
    if (VERBOSE) {
        std::cout << "Does not match required func " <<std::endl ;
    }
    pthread_mutex_unlock(&aMutex) ;
    //return(0) ;
}
else {
    //Acceleration (m/s^2)
    myIMU.acceleration.x = floatFromBytes(&response[1]); //bytes 1..4
    myIMU.acceleration.y = floatFromBytes(&response[5]); //bytes 5..8
    myIMU.acceleration.z = floatFromBytes(&response[9]); //bytes 9..12
    //Angular Rate
    myIMU.acceleration_angular.apitch = floatFromBytes(&response[13]); //bytes 13..16
    myIMU.acceleration_angular.aroll = floatFromBytes(&response[17]); //bytes 17..20
    myIMU.acceleration_angular.ayaw = floatFromBytes(&response[21]); //bytes 21..24
    //Orientation (rad)
    myIMU.orientation.pitch = (asin(-floatFromBytes(&response[33])));
    myIMU.orientation.roll = (atan2(floatFromBytes(&response[45]),floatFromBytes(&
        response[57])));
    myIMU.orientation.yaw = (atan2(floatFromBytes(&response[29]),floatFromBytes(&
        response[25])));
    myIMU.setTimeStamp(timeOfRead);
    pthread_mutex_unlock(&aMutex) ;
    //Timer
    /*unsigned long tempTimer = ((((((response[61]) << 8) & response[62]) << 8) &
        response[63]) << 8) & response[64];
     //divide by timer constant to give result in seconds
    myIMU.timer = tempTimer / 19660800 ;
    */
    sharedIMU.write(myIMU);
    //write to sharedMemory
    //if (0 == (++printCount % 100)) {
    cerr << "DriverIMU: Updating shared memory; yaw = "
    << myIMU.orientation.yaw << endl;
    //}
}
```cpp
323
324  
325 return(SUCCESS);
326  
327 
328 void DriverIMU::start(void) {
329  
330  unsigned char response[8];
331  
332  // Open the serial port and configure it as required
333  fd = openSerialPort();
334  
335  // Reset IMU
336  this->reset();
337  
338  // Flush IMU buffer
339  tcflush(fd, TCIOFLUSH);
340  cout << "IMU Reset and buffer flushed " << endl;
341  
342  // Put IMU in continuous mode
343  unsigned char temp[4] = {0xC4, 0xC1, 0x29, IMU_COMMAND};
344  
345  if (write(fd, temp, 4) != 4) {
346      std::cout << "Unable to send continuous mode command: " << strerror(errno) << "Trying again" << std::endl;
347      if (write(fd, temp, 4) != 4) {
348          exit(EXIT_FAILURE);
349      }
350  }
351  
352  // Small delay so that IMU can respond
353  sleep(1);
354  
355  // Read response from IMU
356  if (read(fd, response, 8) < 0) {
357      std::cout << "Unable to read response to continuous mode command: " << strerror(errno) << std::endl;
358      exit(EXIT_FAILURE);
359  }
360  
361  if (response[0] == (unsigned char)0xC4 && response[1] == IMU_COMMAND) {
362      std::cout << "Continuous mode successfully started" << std::endl;
363      // exit(EXIT_FAILURE);
364  } else {
365      std::cout << "Unexpected response to continuous mode command: " << (unsigned long) response << std::endl;
366      // exit(EXIT_FAILURE);
367  }
368  
369 bool DriverIMU::getData(void) {
370```

// Read data and if it fails then print error. Should block at this point if there is no data.

static int i = 0;
if (readStr(fd)<1) {
    if (VERBOSE) {
        // std::cout << "IMU read error" << std::endl;
    }
    return 0;
} else {
    if (VERBOSE) {
        // if (75 == ++i) {
            // Print the output if verbose is on.
            // printf("\n");
            // printf("Angular Rate x:%4.4f \t y:%4.4f \t z:%4.4f\n", 
            // pRecord.angRateX, pRecord.angRateY, pRecord.angRateZ);
            // printf("roll: %4.4f\n", pRecord.roll);
            // printf("pitch: %4.4f\n", pRecord.pitch);
            // printf("yaw: %4.4f\n", myIMU.orientation.yaw);
            // i = 0;
            // }
        }
    }
    return 1;
}

DriverIMU::~DriverIMU() {
    unsigned char temp[3] = {0xFA, 0x75, 0xB4};
    write(fd, temp, 3);
    close(fd);
}
H.1.1.5  CAMERA

camera_driver.cpp

/**
 * camera_driver.cpp
 * Author: Chris Madden and Mark Baulis
 * Purpose: Save images from camera and save them to a defined spot on the
 * UGV (mounted USB). Also ensure clarity of image by adjusting gain and
 * exposure automatically
 */

#include <stdlib.h>
#include <stdio.h>
#include <string>
#include "bgapi.hpp"
#include <vector>

using namespace std;

// Adding in the OpenCV library to perform image saving
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include <iostream>
#include <sstream>
#include <fstream>

bool enable_tracing = 0;

// Definitions for getting the camera workin on the limited bandwidth
int BGR_FORMAT = 35127317; // BGR format
int DELAY = 2000;
int LEFT = 400;
int RIGHT = 1216;
int TOP = 410;
int BOTTOM = 810;

// Parameters that can be adjusted for Automatic software gain control
float initial_gain = 4.0f; // Level of initial camera gain
int initial_exposure = 1000; // Level of initial camera exposure (for starting outside)
int x_samples = 160; // Number of sample points across the image
int y_samples = 160; // Number of sample points down the image
double adjustment_amount = 85.0; // Level of Gain adjustment change per 1 unit currently
                                 // (256/9)*3
double adjustment_level = 15.0; // Level of difference before changing
double expected_level = 115.0; // Expected level of image brightness

/*******************************************************
 * This Class that stores the camera information and performs the callback functionality
 * to do the Image saving to disk
 */
*/
class GigECamera {
public:

    /* This function determines how much the image needs to be adjusted to provide a good image 
* INPUTS IplImage* cur_im This is the image to analyse 
* OUTPUTS double This is the amount to adjust the camera gain by 
*/
    double get_gain_adjustment(IplImage * cur_im) {
        // Debug only
        //cvNamedWindow("Current Image");
        //cvShowImage("Current Image", cur_im);
        //cvWaitKey(1);

        char * im_data = cur_im->imageData;
        double average_diff = 0.0;
        double pixel_data_sum = 0.0;
        int pixel_count = 0;
        int width_step = (int)floor((double)cur_im->width/x_samples);
        int height_step = (int)floor((double)cur_im->height/y_samples);
        for (int i=1; i<cur_im->width; i+=width_step) {
            for (int j=1; j<cur_im->height; j+=height_step) {
                pixel_data_sum = pixel_data_sum + (uchar)im_data[j*cur_im->widthStep+i] + (uchar)im_data[j*cur_im->widthStep+i+1] + (uchar)im_data[j*cur_im->widthStep+i+2];
                pixel_count+=3;
            }
        }
        average_diff = pixel_data_sum/pixel_count - expected_level;
        //printf("avg_diff=%.2f, sum=%.1f, pixels=%d, expected=%.2f\n", average_diff, pixel_data_sum, pixel_count, expected_level);

        // Only adjust the gain if it is out by more than the threshold level
        if (abs(average_diff) > adjustment_level ) {
            return -average_diff/adjustment_amount; // returns the amount to adjust gain by
        }
        return 0.0;
    }

    GigECamera( BGAPI::Camera* c, BGAPI::Image* i ) :
        camera( c ),
        image( i )
    {
        reduction = 2; // Size factor to reduce the image by (4 recommended for streaming up to 10fps)
        adjusting_gain = true; // Allow the camera to automatically adjust the gain and exposure settings
        //Setup Feature status and IP variables
        state.cbSize = sizeof(BGAPI_FeatureState);
        enable.cbSize = sizeof(BGAPI_TypeBOOL);
        ip.cbSize = sizeof(BGAPI_TypeArrayBYTE);
        subnetmask.cbSize = sizeof(BGAPI_TypeArrayBYTE);
        gateway.cbSize = sizeof(BGAPI_TypeArrayBYTE);
    }
H1 ugv code | 327

dataLen.cbSize = sizeof ( BGAPI_TypeRangeINT );

//Acquire IP Address of camera
BGAPI_RESULT res = camera->getGVSPersistentIP(&state, &enable, &ip, &subnetmask, &gateway);
if (res != BGAPI_RESULT_OK ) {
    printf("Camera->getGVSPersistentIP Errorcode: %d\n", res);
}
camera_number = (ip.array[3]-ip.array[3]%10)/10;
camera_count = 0; // Our own counter of saved images
printf("Camera: %d acquired ", camera_number);

// Set Image format to Full Frame non-HQ and change it to BGR format ofr OpenCV
int formatindex = 1;
res = camera->setImageFormat( formatindex );
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::Camera::setImageFormat Errorcode: %d\n", res);
res = camera->setPixelFormat( BGR_FORMAT );
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::Camera::setPixelFormat Errorcode: %d\n", res);
res = camera->setGain(intial_gain);
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::Camera::setGain Errorcode: %d\n", res);
res = camera->setExposure(intial_exposure);
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::Camera::setExposure Errorcode: %d\n", res);
// These settings are important for reducing frame size and rate for wireless from the camera
res = camera->setPartialScan( true, LEFT, TOP, RIGHT, BOTTOM ); // Set if you want limited scan
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::Camera::setPartialScan Errorcode: %d\n", res);
res = camera->setGVSPacketDelay(DELAY);
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::Camera::setGVSPacketDelay Errorcode: %d\n", res);
res = camera->getDataLength( &state, &dataLen );
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::Camera::getDataLength for Image 0 returned with errorcode %d\n", res);

// Any adjustment to images should be made before here
res = camera->setImage( image );
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::Camera::setImage Errorcode: %d\n", res);
image->getSize( &width, &height );
if ((width == 0) || (height == 0)) { // Set to the standard full resolution if there is no current image size
    width = 1624;
    height = 1232;
}
img = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);
if (reduction != 1.0) { // reduce the image size;
```c
reduced = cvCreateImage(cvSize((int)floor((double)width/reduction), (int)floor((double)height/reduction)), IPL_DEPTH_8U, 3);
}

// Resume the creation of an image and registering its callback function to save the image
res = camera->registerNotifyCallback( this, (BGAPI::BGAPI_NOTIFY_CALLBACK) &imageCallback );
if( res != BGAPI_RESULT_OK ) {
printf("BGAPI::Camera::registerNotifyCallback Errorcode: %d\n", res );
}
}

BGAPI::Camera* getCamera() {
return camera;
}

static BGAPI_RESULT BGAPI_CALLBACK imageCallback( void * callBackOwner, BGAPI::Image *pImage ) {
GigECamera* c = (GigECamera *) callBackOwner;
assert( pImage == c->image );
return c->callback();
}

BGAPI_RESULT callback() {
BGAPI_RESULT res = BGAPI_RESULT_OK;
printf("Enter Callback for camera %i.\n", camera_number);
int buf_width, buf_height;//, t_high, t_low;
char imFile[100];
image->get( &imagebuffer );
image->getSize( &buf_width, &buf_height ); // Check what the current image size really is
//image->getTimeStamp( &t_high, &t_low );
if ((width != buf_width) || (height!=buf_height)) {
// the current image is the wrong size, so recreate the OpenCV image file
width = buf_width;
height = buf_height;
cvReleaseImage(&img);
img = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);
}
//Processing and saving the Imagebuffer
img->imageData = (char *)imagebuffer;
#ifdef _WIN32
sprintf(imFile, "C:\\Data\\UGV\%d\image_%05d.jpg", camera_number, camera_count);
#else
sprintf(imFile, "/UGV/image_%05d.jpg", camera_number, camera_count);
#endif
printf("Image Number is %d\n", camera_count);
```
printf("Image size is (%d,%d), reduced (%d,%d)\n", img->width, img->height, reduced->width, reduced->height);
if (reduction >1) { // reduce the image size;
    cvResize( img, reduced, CV_INTER_LINEAR );
    printf("Saving reduced image: %s ", imFile);
    reduced->height;
    cvSaveImage(imFile, reduced);
} else {
    cvSaveImage(imFile, img);
}
camera_count++;

if (adjusting_gain) {
    BGAPIX_TypeRangeFLOAT cur_gain;
    cur_gain.cbSize = sizeof( BGAPIX_TypeRangeFLOAT );
    BGAPIX_TypeRangeINT cur_exposure;
    cur_exposure.cbSize = sizeof( BGAPIX_TypeRangeINT );

    float change = (float) get_gain_adjustment(img);
    res = camera->getGain(&state, &cur_gain);
    if (!res)
        printf("Error getting Gain %d \n", res);

    float new_gain = cur_gain.current + change;
    //printf("curG=%.2f, newG=%.2f, ", cur_gain.current, new_gain);
    int adjust_exposure = 0;
    if (new_gain < cur_gain.minimum) {
        new_gain = 2.5*cur_gain.minimum;
        // Adjust exposure to be longer
        adjust_exposure = 1;
    } else {
        if (new_gain > 0.7*cur_gain.maximum) {
            new_gain = 0.8*cur_gain.maximum;
            // Adjust exposure to be shorter
            adjust_exposure = 2;
        }
    }
    //printf("finalG=%.2f\n", new_gain);
    if (new_gain!=cur_gain.current) {
        // Adjust the gain if it is within limits
        printf("Adjusting gain by %.2f\n", new_gain);
        res = camera->setGain( new_gain );
        if ( res != BGAPI_RESULT_OK )
            printf( "Error setting the Gain to %.2f: failed with %d\n", new_gain, res );
    }
    if (adjust_exposure) {
        int new_exposure = 0;
        // Adjust the exposure (within limits) if the gain cannot be adjusted
        res = camera->getExposure(&state, &cur_exposure);
        if (!res)
            printf("Error getting Exposure: %d\n", res);
        if (adjust_exposure == 1){
new_exposure = cur.exposure.current - 250;
if (new.exposure < cur.exposure.minimum) {
    new.exposure = cur.exposure.minimum;
}
else {
    new.exposure = cur.exposure.current + 250;
    if (new.exposure > cur.exposure.maximum) {
        new.exposure = cur.exposure.maximum;
    }
}

res = camera->setExposure(new.exposure);
if (res != BGAPI_RESULT_OK)
    printf("Error setting the exposure to %i: failed with %d\n", new.exposure, res);

// after you are finished with this image, return it to the camera for the next image
res = camera->setImage(image);
if (res != BGAPI_RESULT_OK)
    printf("setImage at the end of the callback failed with %d\n", res);
printf("Returning callback\n");
return res;

// Define private class
private:
    BGAPI::Camera* camera;
    BGAPI::Image* image;
    unsigned char* imagebuffer;
    BGAPI::FeatureState state;
    BGAPI::TypeBOOL enable;
    BGAPI::TypeArrayBYTE ip;
    BGAPI::TypeArrayBYTE subnetmask;
    BGAPI::TypeArrayBYTE gateway;
    BGAPI::TypeRangeINT dataLen;
    int camera.number;
    int camera.count;
    int width;
    int height;
    double reduction;
    bool adjusting_gain;
    IplImage* img;
    IplImage* reduced;
};

/*
 * This is the main function which sets up the GigE based system to access and save images from
the camera
*/
int main() {
    // System Initialisation variables
int system_count = 0;
int camera_count = 0;
vector<BGAPI::System*> ppSystem;
vector<BGAPI::System*>::iterator i;
int cam = 0;
int input = 0;
int camera_running;
BGAPI::System * pSystem = NULL;
vector<GigECamera*> pCamera;
//BGAPIX_CameraInfo cameradeviceinfo;
BGAPI_RESULT res = BGAPI_RESULT_FAIL;
BGAPI::TraceObject * pTrace = NULL;

// Camera system Variables
BGAPIX_CameraImageFormat cformat;
cformat.cbSize = sizeof (BGAPIX_CameraImageFormat);
BGAPIX_TypeListINT formatlist;
formatlist.cbSize = sizeof(BGAPIX_TypeListINT);

//Camera system initialisation
//Count systems and create the first one.
res = BGAPI::countSystems( &system_count );
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::countSystems Errorcode: %d system_count: %d\n", res, system_count);
int sys = 0;
res = BGAPI::createSystem( sys, &pSystem );
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::createSystem Errorcode: %d System index: %d\n", res, sys);
res = pSystem->open();
if (res != BGAPI_RESULT_OK )
    printf("BGAPI::System::open Errorcode: %d System index: %d\n", res, sys);
if (res != BGAPI_RESULT_OK ) {
    printf("System Initialisation error");
    return -1;
}
printf("Camera system successfully created and opened\n");

// Initialise the file to write the log
ofstream log;
log.open(“camera_lidar_log.txt”);

if (enable_tracing) {
    // This code sets up the Tracing object, and pronts an error at the point of failure
    res = pSystem->createTraceObject( &pTrace );
    if (res != BGAPI_RESULT_OK )
        printf("BGAPI::System::createTraceObject Errorcode: %d\n", res);
    res = pTrace->setTarget( BGAPI_TRACETARGET_FILE );
    if (res != BGAPI_RESULT_OK )
        printf("BGAPI::TraceObject::setTarget Errorcode: %d\n", res);
    res = pTrace->setTraceFileName( "GCS_Camera_Tracefile.log", 25 );
    if (res != BGAPI_RESULT_OK )
        printf("BGAPI::TraceObject::setTraceFileName Errorcode: %d\n", res);
res = pTrace->setOutputOptions( BGAPI_TRACEOUTPUTOPTION.NEWLINE | 
    BGAPI_TRACEOUTPUTOPTION.TIMESTAMP | BGAPI_TRACEOUTPUTOPTION.TRACELEVEL );
if( res != BGAPI_RESULT_OK )
    printf("BGAPI: :TraceObject::setOutputOptions Errorcode: %d\n", res);
res = pTrace->setLevel( BGAPI_TRACELEVEL.ALL );
if( res != BGAPI_RESULT_OK )
    printf("BGAPI: :TraceObject::setLevel Errorcode: %d\n", res);
res = pTrace->setMask( BGAPI_TRACEMASK_ERROR | BGAPI_TRACEMASK.WARNING | 
    BGAPI_TRACEMASK_INFORMATION );
if( res != BGAPI_RESULT_OK )
    printf("BGAPI: :TraceObject::setMask Errorcode: %d\n", res);
res = pTrace->enableLogging( true );
if( res != BGAPI_RESULT_OK )
    printf("BGAPI: :TraceObject::enableLogging Errorcode: %d\n", res);

if (res != BGAPI_RESULT_OK ) {
    enable_tracing = false;
} else {
    printf("Logfile ‘GCS_Camera_Tracefile.log’ created! Logging enabled!\n");
}

//Count cameras
res = pSystem->countCameras( &camera_count );
if( res != BGAPI_RESULT_OK ) {
    printf("BGAPI: :System::countCameras Errorcode: %d camera_count: %d\n", res, 
        camera_count);
    return -2;
}
printf("Detected Cameras: %d \n", camera_count);

//Loop to create and initialise each camera
for( cam = 0; cam < camera_count; cam++ )
{
    BGAPI::Camera * tmp_camera = NULL;
    BGAPI::Image * tmp_image = NULL;

    // Create and open the current camera
    res = pSystem->createCamera( cam, &tmp_camera );
    if( res != BGAPI_RESULT_OK )
    {
        printf("BGAPI: :System::createCamera Errorcode: %d Camera index: %d\n", res, cam);
        break;
    }
    res = tmp_camera->open();
    if( res != BGAPI_RESULT_OK )
    {
        printf("BGAPI: :Camera::open Errorcode: %d Camera index: %d\n", res, cam);
        break;
    }
    printf("Created and Opened Camera: index %d\n", cam);

    res = BGAPI::createImage( &tmp_image );
    if( res != BGAPI_RESULT_OK )
    {
        printf("BGAPI: :createImage Errorcode: %d\n", res);
break;
}

// Put the camera into the vector of our custom Camera class
pCamera.push_back( new GigECamera( tmp_camera, tmp_image ) );
printf("Camera %d create with an allocated Image allocated\n", cam);
}

// Cameras have been created and are ready to be started
camera_running = 0;
for( cam = 0; cam < camera_count; cam++ )
{
    printf("Setting Camera %d active\n", cam);
    // Start the camera stream
    res = pCamera[cam]->getCamera()->setStart( true );
    if( res != BGAPI_RESULT_OK ) {
        printf("BGAPI::Camera::setStart Errorcode: %d\n", res );
    } else {
        camera_running = 1;
    }
}

input = 0;
while (camera_running) {
    // Wait while images are being saved. This needs to be expanded to consider conditions
    // for stopping cameras
    //printf("Inside loop\n");
    cvWaitKey(100); // sleep for at least 1ms whilst waiting for next image
}

log.close();
//Release the cameras and the camera system
printf( "Shutting down.\n" );
for( cam = 0; cam < camera_count; cam++ )
{
    res = pCamera[cam]->getCamera()->setStart( false );
    if( res != BGAPI_RESULT_OK )
        printf("BGAPI::Camera::setStart Errorcode: %d\n", res);
    res = pSystem->releaseCamera( pCamera[cam]->getCamera() );
    if( res != BGAPI_RESULT_OK )
        printf("BGAPI::System::releaseCamera Systemnumber %d Errorcode: %d\n", 0, res );
}
res = pSystem->release();
if( res != BGAPI_RESULT_OK )
    printf("BGAPI::System::release Errorcode: %d System index: %d\n", res, sys);
printf("System released: System index %d\n", sys);
H.1.1.6  ULTRASONICS

UltrasonicDriver.cpp

/*
 * UltrasonicDriver.cc
 *
 * Created on: May 18, 2010
 * Updated on: June 10, 2010
 * Author: Peter Hardy
 * Description - connects to Ultrasonic sensor through RS-232 serial,
 * baud rate 9600, does not block whilst reading
 * Returns distance if everything worked properly
 * Returns -1 if problem opening serial port
 */

#include <cstdlib>
#include <iostream>
#include <fstream>
#include <cmath>
#include <stdio.h>

#define DEVICE "/dev/ser3"

int fd; //Serial port var

int openSerialPort()
{
    //Open Port
    int port;
    port = open(DEVICE, O_RDONLY | O_NOCTTY | O_NONBLOCK);
    if (port <0) {cout<<"No Connection"<<endl; return(-1); }

    //END Open Port

    //Port Settings
    struct termios options;
    // Get the current options for the port...
tgetattr(port, &options);
    // Set the baud rates to 9600...
cfsetispeed(&options, B9600);
cfsetospeed(&options, B9600);
    // Enable the receiver and set local mode...
options.c_cflag |= (CLOCAL | CREAD);
    //options.c_cflag &= ~PARENB;
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;

    // Set the new options for the port...
tcsetattr(port, TCSANOW, &options);
    //END Port Settings

    return port;
```c
int main() {
    char buffer[5];

    //Set up messaging to waypoint program
    //Channel used is "Ult_Way", set up in waypoint program
    int coid = name_open("Ult_Way", 0);

    //Open serial port
    fd = openSerialPort();

    if (fd == -1) {
        cout << "Problem opening serial port for ultrasonics" << endl;
        cout << "Return -1" << endl;
        return fd;
    }

    while(1) {
        //Read serial port - 5 characters are returned, Capital letter 'R' followed by
        //3 characters (hundreds, tens, units) for distance, then ' ' is the final bit.
        read(fd, buffer, sizeof(buffer));

        //Check that first character is 'R', or 82 in integer form, and last character
        //is a white space, integer 13
        if (((int(buffer[0]) == 82) && int(buffer[4]) == 13)) {
            //Convert from char to int, subtract character '0' to convert from ASCII
            int bit1 = buffer[1] - '0';
            int bit2 = buffer[2] - '0';
            int bit3 = buffer[3] - '0';

            //Add it all together to get distance
            unsigned short distance = 100 * bit1 + 10 * bit2 + bit3;
            //Send data to waypoint program via message passing
            MsgSend(coid, &distance, sizeof(distance), NULL, 0);
        }

        //Ultrasonic frequency is 10Hz, so delay 100ms before reading again
        delay(100);
    }
}
```
H.1.2 LOCALISATION

H.1.2.1 OUTDOOR LOCALISATION

kalman.h

/**
 * Kalman.h
 * This header file contains the output structure from the Kalman filter. This contains all pose
 * information that other processes required and is written to shared memory.
 * Author: Anton Steketee
 * Last Modified: 16 October 2010
 * ----------CHANGELOG------------------
 * 30 June 2010 (AS) : Added yawdot, timestamp and header comments
 * 24 August 2010 (AS) : Added all the stuff from the Kalman.cpp file for simplicity ...
 * September 2010 (AS) : Convert to mpi shared memory useage
 * 16 October 2010 (AS) : Tidy
 */

#include <data/DataIMU.h>
#include <data/DataIMUShared.h>
#include <data/DataGPS.h>
#include <data/DataGPSShared.h>
#include <msg/MessageUGVPose.h>
#include <msg/MessageUGVPoseShared.h>

#include <time.h>
#include <iostream>
#include <fstream>
#include <string>
#include <complex>
#include <pthread.h>

#include "EKF.h"
#include "Matrix/Matrix.h"
#include <errno.h>

#include "lp_filter.h"

#ifndef Kalman
#define Kalman
#define PI 3.1415
#define G9 0.8065

using namespace std;


// Define base station location - GPS position. Position is determined relative to this
position if a GPS signal is available. Otherwise it will be relative to 0,0 and the
finding of GPS position during operation could cause an error.

double baseStationE, baseStationN;
float baseStationH;

double startTime; //seconds
double CurrentTime; //change in time from start

// These variable contain the data sent from the driver processes
static mpi::data::DataGPSShared gpsShm;
static mpi::data::DataGPS gpsData;
static mpi::data::DataIMUShared imuShm;
static mpi::data::DataIMU imuData;

// This is where the data goes to after filtering.
static mpi::msg::MessageUGVPoseShared poseShm;
static mpi::msg::MessageUGVPose pose;
static mpi::msg::MessageUGVPose unfilteredPose;

// Thread related stuff
pthread_t IMU_thread;
pthread_t GPS_thread;

// Setup mutex. This mutex prevents separate thread from trying to update the pose (through
the KF) at the same time.

pthread_mutex_t ekf_mutex = PTHREAD_MUTEX_INITIALIZER;

// Define states and covariance here so that they have file scope.
// State Vector. This vector contains the best estimate of current pose
// Easting(m), Northing(m), Heading(radians, -pi to pi), velocity (m/s)
Matrix x;
int lx; //size of x

// Estimate covariance. This is the estimated covariance in the best estimate of current pose
Matrix P;

// Process disturbance covariance
Matrix Q;

// Objects used for EKF
EKF an_ekf;

// Output file
char outputFile[] = "kalmanLogs/KFOutput.text"; // File that results are written to. This
grows in size very quickly.
std::ofstream* fileOutput;

// Low pass filter
lp_filter * lp;

#endif
Based on a whole bunch of different previous work.

You need to start this process, then start the device drivers, namely the GPS and IMU drivers. It is better to start the GPS driver first as it returns absolute position.

See Kalman Position Filter.doc for some details.

--------------CHANGELOG---------------------------
2 July 2010 (AS): Change from a 5 state with 2 input to a 6 state, 0 input model. This allows for a distinction between process noise and actual measurement noise and is a simpler final model. Some tuning also performed on individual sensors but not combined.
1 July 2010 (AS): Change of Q from 2 states into 5. This is a representation of actual process noise rather than input noise. Seems to give much better tuneability. Also, split thread handling for system state into a separate function.
30 June 2010 (AS): Change of representation of heading to a vector, to avoid discontinuity at -180/180.

Tuning of error estimates to suit this. Added a constant scaling factor to the accuracy from the GPS, which seems to have helped. Split source files into folders.

11 August 2010 (AS): Begin change to Ubuntu
24 August 2010 (AS): Move stuff to separate header files.
25 August 2010 (AS): Multiple Kalman filters to compare them.
30 August 2010 (AS): Added some log file stuff - mainly for basestation position.
28 September 2010 (AS): tuning from log file. Have changed to an improved dynamic model - re-tune. Logging system has been changed slightly also.
16 October 2010 (AS): General tidy up and finalisation. Should all be working reasonably at this point. Drift of IMU solution still bad but limited by acceleration and heading measurement accuracy.

-----------------TO DO --------------------------
++Stop using matrix class as it is very clunky.
++Develop so that all weightings and system model are read from a file.
*/

#include "kalman.h"

void updateFileGPS(mpi::msg::MessageUGVPose * posePtr, std::ofstream * aLog) {
  // The current pose and the measurements are also written to a file
  // The format is x y yaw vel acc gpsE gpsN gpsV 0 0 0 time GPSheading E_stdev N_stdev 0 0 0 0
  *aLog<<"GPS " <<(*posePtr).pos_ugv.x << " "
}
void updateFileIMU(mpi::msg::MessageUGVPose * posePtr, std::ofstream * aLog) {
    // The format is x y yaw vel acc 0 0 0 sin(yaw) cos(yaw) IMUacc time 0 0 0 IMUacc_y IMUacc_z IMU_roll IMU_pitch
    *aLog<<"IMU "<<(*posePtr).pos_ugv.x << " "
    << (*posePtr).pos_ugv.y << " "
    << (*posePtr).or_ugv.yaw << " "
    << (*posePtr).vel_ugv.speed << " "
    << (*posePtr).acc_ugv.x << " 0 0 0 "
    << (float)(sin(imuData.orientation.yaw+PI)) << " "
    << (float)(cos(imuData.orientation.yaw+PI)) << " "
    << imuData.acceleration.x<<" ";
    timespec TimeStamp = imuData.getTimeStamp() ;
    *aLog<< (0.000000001*TimeStamp.tv_nsec) + TimeStamp.tv_sec- starttime " "
    << gpsData.velocity.heading << " "
    << (float)(gpsData.position_stdev.easting_stdev*gpsData.position_stdev.easting_stdev)
    << " "
    << (float)(gpsData.position_stdev.northing_stdev*gpsData.position_stdev.
    northing_stdev) << " "
    << " 0 0 0 0 "
    <<endl ;
}

void convertStateToPose(Matrix x, timespec timeStampTemp , mpi::msg::MessageUGVPose * posePtr) {
    //This function converts the state as tracked by the kalman filter to pose.
    (*posePtr).pos_ugv.x = x[0][0] ;
    (*posePtr).pos_ugv.y = x[1][0] ;
    (*posePtr).pos_ugv.z = gpsData.position.height ;
    (*posePtr).or_ugv.yaw = (atan2f(x[2][0],x[3][0])+PI) ;
    (*posePtr).or_ugv.roll = -(imuData.orientation.roll) ;
    (*posePtr).or_ugv.pitch = -(imuData.orientation.pitch) ;
    (*posePtr).vel_ugv.speed = x[4][0] ;
    (*posePtr).acc_ugv.x = x[5][0] ;
    (*posePtr).acc_ugv.y = G*imuData.acceleration.y ;
    (*posePtr).acc_ugv.z = G*imuData.acceleration.z ;
    (*posePtr).timestamp = timeStampTemp ;
void * updateFromIMU(void *) {
  /**<
   * This function blocks while waiting for an IMU update. When it is recived, it updates
   * the
   * Kalman Filter using the heading and forward acceleration. The result is then
   * written to a file for post plotting and into shared memory for access by other
   * processes...
   */

  int num_Measurements = 3;

  //Covariance of measurement noise
  Matrix R(num_Measurements, num_Measurements);
  R.Zero();
  Matrix R1, R2, R3;
  R[0][0] = 50; //sin(theta) covariance
  R[1][1] = 50; //cos(theta) covariance
  R[2][2] = 100; //acceleration covariance

  //Measurement Noise Covariance Matrix
  Matrix C(num_Measurements, lx);
  C.Zero();
  C[0][2] = 1; //sin(theta)
  C[1][3] = 1; //cos(theta)
  C[2][5] = 1; //Acceleration

  //Measurements vector
  Matrix y(num_Measurements, 1);

  while(1) {
    /**< Read data from shared mem and set values of y */
    imuShm.waitForUpdate();
    imuShm.read(imuData);
    //Set the inputs and measurements from this sensor
    //TODO: Check orientation of IMU on robot to see if this is ok.
    y[0][0] = (float)(sin(imuData.orientation.yaw));
    y[1][0] = (float)(cos(imuData.orientation.yaw));
    //Acceleration: we just want the forward acceleration but the x acceleration is
    //influenced by orientation
    y[2][0] = (float)lp->update(-G*imuData.acceleration.x);
    //Low pass filter the acceleration updates.
    y[2][0] = (float)lp->update(G*(-imuData.acceleration.x*cos(imuData.orientation.
      pitch) + imuData.acceleration.z*sin(imuData.orientation.pitch))) ;

    pthread_mutex_lock(&ekf_mutex);

    timespec timeStampTemp = imuData.getTimeStamp();
    // The ekf is run and pose updated using the new measurements.
    an_ekf.run(x, y, P, Q, R, C, timeStampTemp);
void* updateFromGPS(void*) {

    // This thread blocks while waiting for new GPS readings. It uses Easting, Northing and
    // Velocity from the GPS. Heading information is included in the output file for
    // information.
    int num.Measurements = 3;

    //Covariance of measurement noise ( E, N)
    Matrix R(num.Measurements,num.Measurements);
    R.Zero();

    //Measurement Matrix
    Matrix C(num.Measurements,lx);
    C[0][0] = 1; //Easting
    C[1][1] = 1; //Northing
    C[2][4] = 1; //Velocity

    Matrix y(num.Measurements,1);

    while(1) {
        // Read data from sensors and set values of and y
        gpsShm.waitForUpdate();
        gpsShm.read(gpsData);

        //Set the measurements from this sensor
        y[0][0] = (float)(gpsData.position.easting - baseStationE);
        y[1][0] = (float)(gpsData.position.northing - baseStationN);
        y[2][0] = (float)gpsData.velocity.speed;

        R[0][0] = (float)(50*gpsData.position_stdev.easting_stdev*gpsData.position_stdev.
                        easting_stdev);
        R[1][1] = (float)(50*gpsData.position_stdev.northing_stdev*gpsData.position_stdev.
                        northing_stdev);
        R[2][2] = (float)(5/gpsData.velocity.speed); //The datasheet lists this as 0.03 m/s

        pthread_mutex_lock(&ekf_mutex);
    }
}

timespec timeStampTemp = gpsData.getTimeStamp();
an_ekf.run(x, y, P, Q, R, C, timeStampTemp);

// The results are written to the output structure
convertStateToPose(x, timeStampTemp, &pose);
updateFileGPS(&pose, fileOutput);
pthread_mutex_unlock(&ekf_mutex);
poseShm.write(pose);
}
return (NULL);
}

int main ( )
{
    // Create low pass filter instance
    lp = new lp_filter;
    // Load up a config file. This contains the base station position but I should add the
covariances used by the kalman filter also.
    // Could be complex with the matrix class?
    ConfigFile config("config.inp");
    baseStationE = config.read<double>("baseStationEasting");
    baseStationN = config.read<double>("baseStationNorthing");
    baseStationH = config.read<float>("baseStationHeight");
    // Initialise the state vector, if there is any data in the shm, use that. If there is not
    // it will be zero anyway.
    x = "[0; 0; 0; 1; 0; 0]";
    lx = x.nrows(); // Number of actual states
    imuShm.read(imuData);
gpsShm.read(gpsData);
    x[2][0] = sin(imuData.orientation.yaw);
    x[3][0] = cos(imuData.orientation.yaw);
    if (gpsData.position.easting != 0) {
        x[0][0] = gpsData.position.easting - baseStationE;
        x[1][0] = gpsData.position.northing - baseStationN;
    } else {
        x[0][0] = 0;
        x[1][0] = 0;
    }
// Initialise the estimated accuracy. These are very large to account for bad initialisation
P = "[100 0 0 0 0; 0 100 0 0 0; 0 0 100 0 0; 0 0 0 10 0; 0 0 0 0 10]";

// Process noise covariance. How much is the state expected to change?
Q = "[0.01 0 0 0 0; 0.01 0 0 0 0; 0 0.05 0 0 0; 0 0 0.05 0 0; 0 0 0 0 0.02]";

// Initialise EKF
starttime = an_ekf.init(lx);

fileOutput = new std::ofstream(outputFile);

//Start thread to deal with GPS readings
pthread_create(&GPS_thread, NULL, updateFromGPS, NULL);

//Start thread to deal with IMU readings
pthread_create(&IMU_thread, NULL, updateFromIMU, NULL);

while (1) {
    //print to screen at approximately 1Hz
    cerr<<"----------------Filtered vs Unfiltered Results----------------" <<endl;
    pthread_mutex.lock(&ekf_mutex);
    cerr<<"Easting(m): " <<pose.pos_ugv.x <<" " <<unfilteredpose.pos_ugv.x <<endl;
    cerr<<"Northing(m): " <<pose.pos_ugv.y <<" " <<unfilteredpose.pos_ugv.y <<endl;
    cerr<<"Yaw(deg): " <<pose.or_ugv.yaw <<" " <<unfilteredpose.or_ugv.yaw <<endl;
    cerr<<"Velocity(m/s): " <<pose.vel_ugv.speed <<" " <<unfilteredpose.vel_ugv.speed <<endl;
    cerr<<"Acceleration(m/s): " <<pose.acc_ugv.x <<" " <<unfilteredpose.acc_ugv.x <<endl;
    cerr<<"-----------------------------" <<endl;
    P.PrintStdout(4);
    pthread_mutex.unlock(&ekf_mutex);
    usleep(999999);
}

return(NULL);
#ifndef EKF_H_
#define EKF_H_

#include "Matrix/Matrix.h"
#include <time.h>
#include <cmath>

#define PI 3.1415

class EKF {
public:
    /** Initialize this EKF. */
    * Inputs:
    * lx - number of states
    */
    double init( unsigned int lx ) ;

    /** Run the filter after initialization. */
    * Inputs:
    * x - state vector (not augmented)
    * u - input vector
    * y - measurements
    * P - state covariance (not augmented)
    * Q - input noise
    * R - measurement noise
    * C - measurement matrix
    */
    public:
        bool run(Matrix &x, Matrix &y, Matrix &P, Matrix &Q, Matrix &R, Matrix &C, timespec) ;

private:

    /** Length of the (possibly augmented) state vector (L) */
    unsigned int m_StateLength ;

    /** Number of actual states (not augmented) */
    unsigned int m_numofStates ;

    /** Number of measurement */
    unsigned int m_numofMeasurements ;
/** State mean and covariances estimates */
Matrix m_Xhat; /// state estimate Vector
Matrix m_Px; /// state covariance

/** Posteriori statistics used in the time update */
Matrix m_Xhat_minus; /// state estimate Vector
Matrix m_Px_minus; /// state covariance

/** Posteriori statistics used in the measurement update */
Matrix m_Yhat_minus; /// measurement estimate Vector

/** Kalman gain matrix */
Matrix m_KalmanGain;

/** Update matrix **/
Matrix A ;
Matrix At;
Matrix F ;
Matrix Ft ;

/** Input matrix **/
Matrix B ;

/** Measurement Matrix **/
Matrix C;
Matrix Ct;

/* Timestamp (eg time of latest iteration */
timespec timestamp ;

};

#endif /* EKF_H_ */
EKF.cpp

/*
 * EKF.cpp
 *
 * Created on: Mar 14, 2010
 * Author: ajs
 */

#include "EKF.h"

double EKF::init( unsigned int lx ){
    m_numofStates = lx ;
    m_StateLength=lx;
    m_Xhat.Redim(lx,1);
    // m_measurements(l),
    m_Px.Redim(lx, lx) ;
    m_Xhat_minus.Redim(lx, 1) ;
    m_Px_minus.Redim(lx, lx) ;
    F.Redim(lx,lx) ;
    //Generate start time of the filter
    if( clock_gettime( CLOCK_MONOTONIC, &timestamp) == -1 ) {
        std::cout<<
"clock gettime failed"
        return -1;
    }
    double temp = timestamp.tv_sec + 0.000000001*timestamp.tv_nsec ;
    return temp;
}

bool EKF::run(Matrix &x, Matrix &y, Matrix &P, Matrix &Q, Matrix &R, Matrix &C, timespec newtimestamp) {
    m_numofMeasurements = y.ncols() ;
    m_Yhat_minus.Redim(1,m_numofMeasurements) ;
    m_KalmanGain.Redim(m_numofStates, m_numofMeasurements) ;
    double T = newtimestamp.tv_sec - timestamp.tv_sec ; //Calculate the change in whole
    seconds. Typically this value is zero.
    T += 0.000000001*(newtimestamp.tv_nsec-timestamp.tv_nsec) ; //Add the nanosecs
    timestamp = newtimestamp ; //save the timestamp for next time
    // Prediction
    float v = x[4][0] ;
    //Generate F matrix
    // See derivation in report
    F.Zero() ;
    F[0][2] = -v*T ;
    //F[0][3] = T*v*x[3][0]/x[2][0] ;

//F[0][4] = -T*x[2][0] ;
//F[1][2] = -T*v*x[2][0]/x[3][0] ;
F[1][3] = -v*T ;
//F[1][4] = -T*x[3][0] ;
F[4][5] = T ; //multiplied by a

// A Matrix
A.Identity(m_numofStates) ;
A.Inplace_Add(F) ;
//Prediction
m_Xhat_minus = A*x ;

At = A.Transpose() ;
m_Px_minus = A*P ;
m_Px_minus.Inplace_PostMultiply(At);
m_Px_minus.Inplace_Add(Q) ;

/*This would set the system model to 1*/
//A.Identity(m_numofStates) ;
//m_Px_minus = P ;
//m_Px_minus.Inplace_Add(A) ;
//m_Xhat_minus = x ;

//Update
/* Calculate Kalman Gain */
Matrix Ct = C.Transpose() ;
Matrix temp = C*m_Px_minus ;
temp.Inplace_PostMultiply(Ct) ;
temp.Inplace_Add(R) ;
temp.Inplace_Invert() ;
Matrix temp2 = m_Px_minus*Ct ;
m_KalmanGain = temp2*temp;
//std::cout << " Kalman Gain" << std::endl ;
//m_KalmanGain.PrintStdout() ;

// New x estimate
temp = C*m_Xhat_minus ;
temp = y-temp ;
#endif VERBOSE
std::cout<<"Measurement difference: " ;
temp.PrintStdout() ;
#endif

if temp = m_KalmanGain + temp ;
m_Xhat = m_Xhat_minus + temp ;
//New P matrix
m_Px.Identity() ;
m_KalmanGain = m_KalmanGain*C;
m_Px = (m_Px-m_KalmanGain); m_Px = m_Px+m_Px_minus ;
x = m_Xhat ;
P = m_Px ;

#endif VERBOSE
std::cout<< "Measurement noise estimate"<< std::endl ;
```cpp
R.PrintStdout(6) ;

std::cout<<"A: ";  
A.PrintStdout(4) ;
std::cout<<"B: ";  
B.PrintStdout(4) ;
std::cout<<"C: ";  
C.PrintStdout(4) ;

std::cout<<"Xhat: ";
m_Xhat_minus.PrintStdout(5) ;

std::cout<<"x: ";
m_Xhat.PrintStdout(5) ;
std::cout<<"Kalman Gain:" ;
m_KalmanGain.PrintStdout(5) ;

#endif
return 1 ;}
```
/* A really simple low pass filter to filter the acceleration measurements
Written by Anton Steketee, 10 October 2010

TODO: Read the coefficients in from a file */

ifndef LP
#define LP

class lp_filter {
public:
    lp_filter() ;
~lp_filter() {}
    float update(float) ;
private:
    float X[3] ;
    float Y[3] ;
    int count ;
};
#endif


#include "lp_filter.h"

lp_filter::lp_filter() {
  // This checks all the values are initialized zero
  X[0] = 0;
  X[1] = 0;
  X[2] = 0;
  Y[0] = 0;
  Y[1] = 0;
  Y[2] = 0;
  count = 1;
}

float lp_filter::update(float new_X) {
  // X and Y are used a cyclic buffers, so determine where K, K-1, and K-2 are in the buffer at this point
  int K, K1, K2;
  if (count == 1) {
    K = 0;
    K1 = 2;
    K2 = 1;
  }
  if (count == 2) {
    K = 1;
    K1 = 0;
    K2 = 2;
  }
  if (count == 3) {
    count = 0;
    K = 2;
    K1 = 1;
    K2 = 0;
  }
  X[K] = new_X;
  // Run the filter and save the result
  // Increment the counter so we know where we are in the buffer next time around
  count = count + 1;
  // Multiply the return value by a very small gain so that the result has not been affected by scale.
  return 0.0004797*Y[K];
}
### H.1.2.2 INDOOR LOCALISATION

landmark.h

```c
#ifndef _LANDMARK_H_
#define _LANDMARK_H_

#include "stdafx.h"
#include <vector>

// Set up landmark structure
class Ransac_Landmark
{
    public:

    Ransac_Landmark(Line_Coord AB1, UGV_Pose Now_Pose) : AB(AB1)
    {
        float Angle = Now_Pose.Cur_Angle.Yaw + Now_Pose.PT_Angles.Pan;
        // our goal is to calculate point on line closest to origin (0,0)
        // calculate line perpendicular to input line. A*Ao = -1
        float Ao = -1.0/AB1.A;

        // landmark position
        float X = AB1.B/(Ao - AB1.A);
        float Y = (Ao*AB1.B)/(Ao - AB1.A);

        // now do same calculation but get point on wall closest to robot instead
        // y = Ao*x + Bo => Bo = y - Ao*x

        // get intersection between y = A*x + B and y = Ao*x + Bo
        // so Ao*x + Bo = A*x + B => Ao*x - A*x = B - Bo => x = (B - Bo)/(Ao - A), y = Ao*(B - Bo)/(Ao - A) + Bo
        float Px = (AB1.B - Bo)/(Ao - AB1.A);
        float Py = ((Ao*(AB1.B - Bo))/(Ao - AB1.A)) + Bo;

        // initiate other variables
        Pos.X = X;
        Pos.Y = Y;
        Range_Error = points_distance(Now_Pose.Cur_Loc.X, Now_Pose.Cur_Loc.Y, 0, Px, Py, 0);
    }

    ~Ransac_Landmark(){};
```
Real_Coord get_landmark_pos()
{
    return Pos;
};

int get_landmark_life()
{
    return Life;
};

int get_landmark_times_observed()
{
    return Total_Times_Observed;
};

float get_landmark_range()
{
    return Range;
};

float get_landmark_range_error()
{
    return Range_Error;
};

float get_landmark_bearing()
{
    return Bearing;
};

float get_landmark_bearing_error()
{
    return Bearing_Error;
};

Line_Coord get_landmark_line_coord()
{
    return AB;
};

void reset_landmark_life();  // Resets landmark Life to max value
void decrement_landmark_life();  // Decrement landmark Life
void increment_landmark_times_observed();  // Increment total times observed
void initialise_landmark_times_observed();  // Initialise total times observed
void update_landmark_bearing(float Bearing);  // Update landmark Bearing
void update_landmark_range(float Range);  // Update landmark Range
void Ransac_Landmark::get_association(std::vector<Ransac_Landmark> Landmark_Database, int* ID);  // compare landmarks

private:
    Real_Coord Pos;  // Landmark's (x,y) position relative to map
int Life; // A Life counter used to determine whether to discard a landmark
int Total_Times_Observed; // The number of times we have seen landmark
float Range; // Last observed Range to landmark
float Bearing; // Last observed Bearing to landmark
float Range_Error; // Distance from robot position to the wall we are using as a landmark (to calculate error)
float Bearing_Error; // Bearing from robot position to the wall we are using as a landmark (to calculate error)
Line_Coord AB; // RANSAC: Now store equation of a line

void update_landmarks(int Indices[], int n, std::vector<Ransac_Landmark> Landmark_Database);

void remove_landmark(int i);

int remove_bad_landmarks(); // Remove Landmarks found outside bounded area

#include "H1_ugv_code.h"
landmark.cpp

```cpp
#include "stdafx.h"
#include <vector>

// resets landmark Life to max value
void Ransac_Landmark::reset_landmark_life()
{
    Ransac_Landmark::Life = LIFE;
}

// decrement landmark Life
void Ransac_Landmark::decrement_landmark_life()
{
    Ransac_Landmark::Life--;
}

// increment total times observed
void Ransac_Landmark::increment_landmark_times_observed()
{
    Ransac_Landmark::Total_Times_Observed++;
}

// initialise total times observed
void Ransac_Landmark::initialise_landmark_times_observed()
{
    Ransac_Landmark::Total_Times_Observed = 1;
}

// update landmark Bearing
void Ransac_Landmark::update_landmark_bearing(float Bearing)
{
    Ransac_Landmark::Bearing = Bearing;
}

//update landmark Range
void Ransac_Landmark::update_landmark_range(float Range)
{
    Ransac_Landmark::Range = Range;
}

// compare landmarks
void Ransac_Landmark::get_association(std::vector<Ransac_Landmark> Landmark_Database, int* ID)
{
    //this method needs to be improved so we use innovation as a validation gate
    //currently we just check if a landmark is within some predetermined distance of a
    // landmark in DB
    bool Seen = false;
```
```cpp
for(int i=0; i < Landmark_Database.size(); i++)
{
    if(points_distance(Ransac_Landmark::get_landmark_pos().X, Ransac_Landmark::
                        get_landmark_pos().Y, 0, Landmark_Database[i].get_landmark_pos().X,
                        Landmark_Database[i].get_landmark_pos().Y, 0) < MAX_ERROR)
    {
        Landmark_Database[i].reset_landmark_life(); // Landmark seen so reset its Life counter
        Landmark_Database[i].increment_landmark_times_observed(); // Increase number of times we seen landmark
        Landmark_Database[i].update_landmark_bearing(Ransac_Landmark::
            get_landmark_bearing()); // Set last bearing seen at
        Landmark_Database[i].update_landmark_range(Ransac_Landmark::get_landmark_range
            ()); // Set last range seen at
        Seen = true;
        *ID = i;
    }
}

if (!Seen)
{
    Ransac_Landmark::reset_landmark_life(); //set landmark Life counter
    Ransac_Landmark::initialise_landmark_times_observed(); //initialise number of times we’ve seen landmark
    // cout << this->get_landmark_life() << endl;
    // cout << Landmark_Database[0].get_landmark_life() << endl;

    *ID = Landmark_Database.size();
    cout << "hello" << endl;
}

// update landmarks that haven’t been seen
void update_landmarks(int Indices[], int n, std::vector<Ransac_Landmark> Landmark_Database)
{
    bool Update;
    for (int i=0; i < Landmark_Database.size(); i++)
    {
        Update = true;
        for (int j=0; j < n; j++)
        {
            if (i == Indices[j])
            {
                Update = false;
            }
        }
        if (Update)
```
(Landmark_Database[i]).decrement_landmark_life(); // landmark seen so reset its Life counter

if (Landmark_Database[i].get_landmark_life() == 0)
{
    // remove_landmark(i);
}

// remove landmark
void remove_landmark(int i, std::vector<Ransac_Landmark> Landmark_Database)
{
    for (; i < Landmark_Database.size()-1; i++)
    {
        Landmark_Database.erase(Landmark_Database.begin() + i - 1);
    }
}
#ifndef _RANSAC_H_
#define _RANSAC_H_

#include "stdafx.h"
#include <vector>
#include "landmark.h"

struct Lidar_Readings_Left { Real_Coord Readings[LIDAR_NUMBER]; int Number; };
struct Selected_Readings { int Indices[LIDAR_NUMBER]; int Number; };

//RANSAC Method
void RANSAC(UGV Robot, std::vector<Ransac_Landmark> Landmark_Database);

//Least Squares Line Estimate
Line_Coord least_squares_line_estimate(Lidar_Readings_Left Data_R, Selected_Readings Selection);

// Selecting points for RANSAC method
Selected_Readings selecting_points(int n);

// Updating remaining readings
Lidar_Readings_Left update_ransac_data(Lidar_Readings_Left Data_R, Selected_Readings Selection);

// Choosing centre
int centre(int n);

// Create a file with landmarks
void write_landmarks(UGV Robot, std::vector<Ransac_Landmark> Landmark_Database);

/* THE METHOD:

While
There are still unassociated laser readings,
And the number of readings is larger than the consensus,
And we have done less than N trials.

do
Select a random laser data reading. */
Randomly sample $S$ data readings within $D$ degrees of this laser data reading (for example, choose $S$ sample readings that lie within 10 degrees of the randomly selected laser data reading).

Using these $S$ samples and the original reading calculate a least squares best fit line.

Determine how many laser data readings lie within $X$ centimeters of this best fit line. If the number of laser data readings on the line is above some consensus $C$ do the following:

Calculate new least squares best fit line based on all the laser readings determined to lie on the old best fit line.

Add this best fit line to the lines we have extracted.

Remove the number of readings lying on the line from the total set of unassociated readings.

*/

#ifndef
RANSAC.cpp

```cpp
#include "RANSAC.h"
#include "stdafx.h"
#include <ctime>
#include <vector>

void RANSAC(UGV Robot, std::vector<Ransac_Landmark> Landmark_Database)
{
    int Num_Trials = 0; // Number of trials done on this data
    int IDs[MAX_TRIALS]; // IDs of landmarks found
    int IDs_Number = 0; // Number of landmarks found

    srand(time(NULL)); // Seeds the random function

    // Data_R stores remaining Lidar points - initialise Data_R with all readings and total
    // number of readings
    Lidar_Readings.Left Data_R;
    for (int i = 0; i < LIDAR_NUMBER; i++)
    {
        Data_R.Readings[i].X = Robot.get_last_scan().Cur_Scan[i].X;
        Data_R.Readings[i].Y = Robot.get_last_scan().Cur_Scan[i].Y;
    }
    Data_R.Number = LIDAR_NUMBER;

    // Main Ransac loop
    while (Num_Trials < MAX_TRIALS && Data_R.Number > MIN_LINE_POINTS)
    {
        // Select which points to apply method on
        Selected_Readings Selected = selecting_points(Data_R.Number);

        // Compute model M1: y = a + bx
        Line_Coord M1 = least_squares_line_estimate(Data_R, Selected);

        // Determine the consensus set of points compatible with M1 (within some error
        // tolerance)
        int Total_Points_On_Line = 0; // points close enough to line
        float X = 0, Y = 0, Distance = 0;

        for (int i = 0; i < Data_R.Number; i++)
        {
            // Find points close to line
            X = Data_R.Readings[i].X;
            Y = Data_R.Readings[i].Y;

            Distance = distance_to_line(X, Y, M1.A, M1.B);

            if (Distance < RANSAC_TOLERANCE)
            {
                //add points which are close to line
            }
        }
    }
}
```
```cpp
    Total.Points.On.Line++;

    cout << "( " << Data.R.Readings[i].X << " )";
}

// Debugging
if (Total.Points.On.Line > 0 )
{
    cout << Total.Points.On.Line << endl;
}

// Add line as landmark if there are enough
if(Total.Points.On.Line >= RANSAC_CONSENSUS)
{
    Selected.Number = Total.Points.On.Line;  // Update number of selected points
    Selected.Number = Total.Points.On.Line;  // Update number of selected points

    // Calculate updated line equation based on consensus points
    M1 = least_squares_line_estimate(Data.R, Selected);
    Ransac_Landmark Landmark(M1, Robot.get_pose());  // Initialise landmark
    Landmark.remove_bad_landmarks();  // Remove bad landmarks
    Landmark.get_association(Landmark.Database, &IDs[IDs_Number]);  // Associate landmark
    if (IDs[IDs_Number] == Landmark.Database.size())
    {
        Landmark.Database.push_back(Landmark);  // Update number of landmarks found
        IDs_Number++;  // Update number of landmarks found
    }
    Data.R = update_ransac_data(Data.R, Selected);  // Updates readings left
    Num_Trials++;  // Increments number of trials
}

write_landmarks(Robot, Landmark.Database);
update_landmarks(IDs, IDs_Number, Landmark.Database);  // Updates landmarks
while(1);
```
//Least Squares Line Estimate
Line_Coord least_squares_line_estimate(Lidar_Readings_Left Data_R, Selected_Readings Selection)
{
    float Y;  // y coordinate
    float X;  // x coordinate
    float Sum_Y=0; // sum of y Readings
    float Sum_YY=0; // sum of y^2 for each coordinate
    float Sum_X=0; // sum of x Readings
    float Sum_XX=0; // sum of x^2 for each coordinate
    float Sum_YX=0; // sum of y*x for each point

    Line_Coord AB;

    for(int i=0; i < Selection.Number; i++)
    {
        X = Data_R.Readings[Selection.Indices[i]].X;
        Y = Data_R.Readings[Selection.Indices[i]].Y;
        Sum_Y += Y;
        Sum_YY += pow(Y,2);
        Sum_X += X;
        Sum_XX += pow(X,2);
        Sum_YX += Y*X;
    }

    AB.B = (Sum_Y*Sum_XX - Sum_X*Sum_YX)/(Selection.Number*Sum_XX - pow(Sum_X, 2));
    AB.A = (Selection.Number*Sum_YX - Sum_X*Sum_Y)/(Selection.Number*Sum_XX - pow(Sum_X, 2));
    return AB;
}

Selected_Readings selecting_points(int n)
{
    Selected_Readings Selection;
    Selection.Number = MAX_SAMPLE + 1;
    Selection.Indices[0] = centre(n);
    for(int i = 1; i <= MAX_SAMPLE/2; i++)
    {
        Selection.Indices[i] = Selection.Indices[0] + i;
    }
    for(int j = MAX_SAMPLE/2 + 1; j<=MAX_SAMPLE; j++)
    {
        Selection.Indices[j] = Selection.Indices[0] - j + MAX_SAMPLE/2;
    }
    return Selection;
}
Lidar_Readings_Left update_ransac_data(Lidar_Readings_Left Data_R, Selected_Readings Selection)
{
    for(int i=0; i<Selection.Number; i++)
    {
        for(int j=Selection.Indices[i]; j < Data_R.Number - 1; j++)
        {
            Data_R.Readings[j]=Data_R.Readings[j+1];
        }
        Data_R.Number--;
    }
    return Data_R;
}

// Choosing centre
int centre(int n)
{
    int centre = rand() % (n - 10) + 5;
    return centre;
}

void write_landmarks(UGV Robot, std::vector<Ransac_Landmark> Landmark_Database)
{
    ofstream Landmark_DB;
    int i;
    Landmark_DB.open("LandmarkDataBase.txt", ios::app);
    if (Landmark_DB.is_open())
    {
        Landmark_DB << "Robot " << Robot.get_robot_ID() << " time " << Robot.get_last_scan().Timestamp << " Pose " << Robot.get_pose().Cur_Loc.X << " " << Robot.get_pose().Cur_Loc.Y << " " << Robot.get_pose().Cur_Angle.Yaw<< " " << endl;
        cout << Landmark_Database.size() << endl;
        for (i=0; i<Landmark_Database.size(); i++)
        {
            Landmark_DB << "( " << i << " , " << Landmark_Database[i].get_landmark_pos().X << " , " << Landmark_Database[i].get_landmark_pos().Y << " ) ";
        }
        Landmark_DB << endl;
    }
    else
    {
        cout << "Could not open LandmarkDataBase.txt" << endl;
    }
}
RobotSim.cpp

#include "stdafx.h"
#include "RANSAC.h"
#include <vector>

Global_Map Cur_Map;
float Global_Time;

std::vector<Ransac_Landmark> Landmark_Database; // Vector containing LMDB information
std::vector<UGV> UGVs; // Vector containing UGV information

//int _tmain(int argc, _TCHAR* argv[]) {
int _tmain(int argc, char *argv[]) {
  int i, j, Tmp;
  Real_Loc Initial;
  string Map_File = "";
  int Robots; // How many UGVs are in the system
  int Stop = 120; // Number of steps to simulate
  Global_Time = 0;
  Grid_Loc UGV_Loc;

  // Read arguments to obtain the number of UGVs and the appropriate Map file
  if (argc == 2)
    {
      Map_File = argv[1];
    }
  else
    {
      if (argc == 3)
        {
          Map_File = string(argv[1]); // Why does this not work?
          Robots = atoi(argv[2]);
        }
      else
        {
          cout << argc << " " << argv[0] << " " << argv[1] << " " << argv[2];
          printf("Usage: RobotSim mapFile\n");
          return -1;
        }
    }

  // Code to read map from file
  Map_File = "TestMap.txt";
  //mapFile = "IndoorMap.txt";
  //mapFile = "OutdoorMap.txt";
cout << "Reading Map" << endl;
//curMap.resetAll();
if (Cur_Map.read_map_from_file(Map_File))
{
    // Initialise the robot positions
    for (i=0; i < Robots; i++)
    {
        Initial.X = 4.4;
        Initial.Y = 4.8;
        Initial.Z = 0.6;
        float Heading = 90.0*(PI/180); // angle in radian;
        UGV Tmp_UGV(i, Initial, Global_Time, Heading);
        UGVs.push_back(Tmp_UGV);
    }
} else
{
    // Map reading failed
    cout << "Map reading from " << Map_File << " failed." << endl;
}
Cur_Map.print_true_feature_map();
//curMap.generateVisibility(UGVs);

// Run simulation for a number of steps
cout << "Starting Simulation" << endl;
cout << UGVs.size()<< endl;
for (i = 0 ;i<Stop ;i++)
{
    cout << "printing True Feature Map" << endl;
    //curMap.printTrueFeatureMap();
    cout << "printing Estimated Feature Map" << endl;
    //curMap.printFeatureMap();
    for (j = 0 ;j<UGVs.size(); j++)
    {
        // Generate Lidar sweep for each robot
        UGVs[j].get_lidar_scan(Global_Time, Cur_Map.True_Feature_Map, Cur_Map.Map_X,
        Cur_Map.Map_Y);
        //UGVs[j].write.scan();
    }
    // Move robot if necessary
    for (j=0; j < UGVs.size(); j++)
    {
        if (Global_Time <= 3.9)
        {
            UGVs[j].increment_location(0, -0.1); // move up
} else
{
    if (Global_Time < 4.8)
    {
        UGVs[j].turn(10*(PI/180)); // turn left
    } else
    {
        if (Global_Time <= 8.6)
        {
            UGVs[j].increment_location(-0.1, 0); // move left
        } else
        {
            if (Global_Time < 9.5)
            {
                UGVs[j].turn(10*(PI/180)); // turn left
            } else
            {
                if (Global_Time <= 13)
                {
                    UGVs[j].increment_location(0, 0.1); // move down
                } else
                {
                    if (Global_Time < 13.9)
                    {
                        UGVs[j].turn(10*(PI/180)); // turn left
                    } else
                    {
                        // Will need to sort out moving up the corridor properly
                    }
                }
            }
        }
        Global_Time = Global_Time + 0.1;
        RANSAC(UGVs[0], Landmark_Database);
    }
    UGV_Loc = UGVs[0].get_grid_location();
    cout << "UGV loc=" << UGV_Loc.I << ", " << UGV_Loc.J << ") Heading=" << UGVs[0].get_heading() << endl;
}
while(1);
return 0;
#ifndef _UTILS_H_
define _UTILS_H_

// Standard System definitions and include file
#include "stdafx.h"
define PI 3.14159265

#define PI 3.14159265

#define PI 3.14159265

#define PI 3.14159265

#define PI 3.14159265

static const int MAX_LANDMARKS = 3000;  // a landmark is within 20 cm of another
static const int MAX_ERROR = 200;  // if a landmark is the same landmark
static const int MIN_OBSERVATIONS = 15;  // Number of times a landmark must be observed to
static const int LIFE = 40;  // be recognized as a landmark
static const int MAX_RANGE = 1;
static const int MAX_TRIALS = 100;  // RANSAC: max times to run algorithm
static const int MAX_SAMPLE = 10;  // RANSAC: randomly select X points
static const int MIN_LINE_POINTS = 40;  // RANSAC: if less than N points left don’t bother
static const int RANSAC_TOLERANCE = 400;  // RANSAC: if point is within D distance of line
static const int RANSAC_CONSENSUS = 30;  // RANSAC: at least C votes required to determine
if a line
static const int LIDAR_LIMIT = 60;
static const int LIDAR_NUMBER = 528;  // Number of scans per laser data run (from
static const float LIDAR_SHIFT = 0.36*(PI/180);  // 0.36 degrees per scan in radians
static const float LIDAR_START = 95.0*(PI/180);  // angle in radian

///////////////////////////////////////////////////////

struct Grid_Loc{ int I, J; };  // Relative to the top left corner
struct Real_Loc{ float X, Y, Z; };  // Relative to the base station (X
struct Angles{ float Roll, Pitch, Yaw; };  // Relative to the base station (top
struct GPS_Data{ float X, Y, Z, Timestamp; };  // Relative to the base station ??
static const int GRID_X = 50;
static const int GRID_Y = 55;
static const int GRID_Z = 200;
static const int X_OFFSET = 0;
static const int Y_OFFSET = 0;

///////////////////////////////////////////////////////

struct PT_Data{ float Pan, Tilt, Timestamp; };  // PanTilt unit angles (radians)
struct Real_Coord{ float X, Y; };  // 2D coordinates
struct Line_Coord { float A, B; }; // gradient and y-intercept of line

struct UGV_Pose
{
    Grid_Loc Cur_Grid_Loc; // Best estimate of grid position
    Real_Loc Cur_Loc; // Best estimate of real position
    Angles Cur_Angle; // Best estimate of pitch roll and yaw
    PT_Data PT_Angles; // Angle of PanTilt unit (radians)
    float Speed; // Speed of travel (m/s)
    float Timestamp; // Time at which pose was determined
};

struct Lidar_Scan
{
    int Robot_ID; // ID of robot
    UGV_Pose Robot_Pose; // Pose of robot
    float Cur_Scan_Dist[LIDAR_NUMBER]; // Current Lidarscan
    Real_Loc Cur_Scan[LIDAR_NUMBER]; // Current Lidarscan positions
    float Timestamp; // Time of scan

    void convert_scan();
};

struct Occ_Map
{
    float Map[GRID_X][GRID_Y][GRID_Z];
    static const int X = GRID_X;
    static const int Y = GRID_Y;
    static const int Z = GRID_Z;
};

struct Object_Info
{
    Real_Loc Est_Location;
    int Type;
    float Certainty;
    float Timestamp;
};

struct Frame_Objects
{
    int Robot_ID;
    UGV_Pose Robot_Pose;
    float Timestamp;
};

inline float points_distance(float X1, float Y1, float Z1, float X2, float Y2, float Z2)
{  return sqrt(pow(X1-X2, 2) + pow(Y1-Y2, 2) + pow(Z1-Z2, 2));
};

inline float location_distance(Real_Loc A, Real_Loc B)
{
  return points_distance(A.X, A.Y, A.Z, B.X, B.Y, B.Z);
};

inline float distance_to_line(float X, float Y, float A, float B)
{
  //our goal is to calculate point on line closest to x,y
  //then use this to calculate distance between them.
  //calculate line perpendicular to input line. a*ao = -1
  float Ao = -1.0/A;
  //y = aox + bo => bo = y - aox
  float Bo = Y - Ao*X;
  //get intersection between y = ax + b and y = aox + bo
  //so aox + bo = ax + b => aox - ax = b - bo => x = (b - bo)/(ao - a), y = ao*(b - bo)/(ao - a) + bo
  float Px = (B - Bo)/(Ao - A);
  float Py = ((Ao*(B - Bo))/(Ao - A)) + Bo;
  //calculate distance
  return points_distance(X, Y, 0, Px, Py, 0);
};

inline Grid_Loc real_to_grid(Real_Loc Loc)
{
  // Convert a real location to a grid location using rounding
  Grid_Loc Tmp_Loc;
  Tmp_Loc.I = (int)floor(Loc.X+0.5+X_OFFSET);
  Tmp_Loc.J = (int)floor(Loc.Y+0.5+Y_OFFSET);
  return Tmp_Loc;
};
```cpp
#include "stdafx.h"
#include "Utils.h"

void Lidar_Scan::convert_scan()
{
    // This function converts a Lidar Scan into coordinates
    float Start_Angle = Lidar_Scan::Robot_Pose.Cur_Angle.Yaw + Lidar_Scan::Robot_Pose.
        PT_Angles.Pan - 185.04*PI/180.0;
    float Angle;
    float Robot_X = Lidar_Scan::Robot_Pose.Cur_Loc.X;
    float Robot_Y = Lidar_Scan::Robot_Pose.Cur_Loc.Y;
    for(int i = 0; i < LIDAR_NUMBER; i++)
    {
        Angle = Start_Angle + i*LIDAR_SHIFT;
        float X_Temp = Robot_X + Lidar_Scan::Cur_Scan_Dist[i]*cos(Angle)+5;
        float Y_Temp = Robot_Y + Lidar_Scan::Cur_Scan_Dist[i]*sin(Angle)+5;
        Lidar_Scan::Cur_Scan[i].X = X_Temp-5;
        Lidar_Scan::Cur_Scan[i].Y = Y_Temp-5;
    }
};
```
myUtils.h

#ifndef _MYUTILS_H_
#define _MYUTILS_H_

// Standard System definitions and include file
#include <iostream>
#include <sstream>
#include <fstream>
#include <math.h>
#include "stdafx.h"
#include "Utils.h"

#endif
augment.h

#ifndef _AUGMENT_H_
#define _AUGMENT_H_

#include "stdafx.h"

StateMat augment(StateMat S, Matrix z, Matrix R);
StateMat add_one_z(StateMat S, Matrix z, Matrix R);

#endif
# include "stdafx.h"

// x, P - SLAM state and covariance  
// z, R - range-bearing measurements and covariances, each of a new feature

StateMat add_one_z(StateMat S, Matrix z, Matrix R);

StateMat augment(StateMat S, Matrix z, Matrix R){

    // add new features to state
    Matrix tempZ(z.RowNo(), 1);
    for (int i=0; i<z.ColNo(); i++) {
        for (int j=0; j<z.RowNo(); j++) {
            tempZ(j,0) = z(j,i);
        }
        S = add_one_z(S, tempZ, R);
    }
    return S;
}

StateMat add_one_z(StateMat S, Matrix z, Matrix R){

    int len = S.x.RowNo();
    float r = z(0,0);
    float b = z(1,0);
    float s = sin(S.x(2,0)+b);
    float c = cos(S.x(2,0)+b);

    // augment x
    S.x.SetSize(len+2, S.x.ColNo());
    S.x(len+1,0)=S.x(0,0) + r*s;
    S.x(len+2,0)=S.x(1,0) + r*c;

    // jacobians
    Matrix Axr(2,5);
    Axr(0,0)=1;Axr(0,1)=0;Axr(0,2)=r*c;Axr(0,3)=s;Axr(0,4)=0;
    Axr(1,0)=0;Axr(1,1)=1;Axr(1,2)=-r*s;Axr(1,3)=c;Axr(1,4)=0;
    Matrix Az(2,2);
    Az(0,0)=c;Az(0,1)=-r*s;
    Az(1,0)=s;Az(1,1)=r*c;

    // augment P
    Matrix P_15_15(5,5); // the first 5x5 matrix of P
    for (int i_1=0; i_1<5; i_1++) {
        for (int j_1=0; j_1<5; j_1++) {
            // code continues...
P_15_15(i_1,j_1)=S.P(i_1,j_1);
}
}
Matrix new_row1row2_row1row2(Axr*P_15_15*~Axr+Az*R*~Az); // P(len+1:len+2,len+1:len +2)= Gv*P(1:5,1:5)*Gv' + Gz*R*Gz'
for(int i_2=0; i_2<2; i_2++){  
  for(int j_2=0; j_2<2; j_2++){  
    S.P(len+i_2,len+j_2)= new_row1row2_row1row2(i_2, j_2); // feature cov
  }
}
Matrix new_row1row2_15(Axr*P_15_15); // P(len+1:len+2,1:5)= Gv*P(1:5,1:5)
for (int i_3=0; i_3<2; i_3++){  
  for (int j_3=0; j_3<5; j_3++){  
    S.P(len+i_3,j_3)= new_row1row2_15(i_3, j_3); // vehicle to feature xcorr
  }
}
Matrix new_15_row1row2(~new_row1row2_15); // P(1:5,len+1:len+2)= P(len+1:len+2,1:5)'
for(int i_4=0; i_4<2;i_4++){  
  for(int j_4=0;j_4<len-5;j_4++){  
    S.P(len+i_4,5+j_4)= new_row1row2_6row(i_4,j_4); // map to feature xcorr
  }
}
for(int i_5=0; i_5<len-5;i_5++){  
  for(int j_5=0;j_5<2;j_5++){  
    S.P(5+i_5,len+j_5)= new_6row_row1row2(i_5,j_5); // map to feature xcorr
  }
}
if (len>5){
Matrix P_15_6row(5, len-3); // P(1:5,6:len)
  for (int i_5=0;i_5<5;i_5++){  
    for (int j_5=0; j_5<len-5; j_5++){  
      P_15_6row(i_5,j_5)=S.P(i_5, 5+j_5);
    }
  }
Matrix new_row1row2_6row(Axr*P_15_4row); // P(len+1:len+2,6:len)= Gv*P(1:5,6:len)
Matrix new_6row_row1row2(~new_row1row2_6row); // P(6:len,len+1:len+2)= P(len+1:len+2,6:len)'
  for(int i_6=0; i_6<2;i_6++){  
    for(int j_6=0;j_6<len-5;j_6++){  
      S.P(len+i_6,5+j_6)= new_row1row2_6row(i_6,j_6); // map to feature xcorr
    }
  }
  for(int i_7=0; i_7<len-5;i_7++){  
    for(int j_7=0;j_7<2;j_7++){  
      S.P(5+i_7,len+j_7)= new_6row_row1row2(i_7,j_7); // map to feature xcorr
    }
  }
}
return S;
}
#ifndef _KFSIMPLEUPDATE_H_
define _KFSIMPLEUPDATE_H_

#include "stdafx.h"

// Calculate the KF (or EKF) update given the prior state [x,P]
// the innovation [v,R] and the (linearised) observation model H.

StateMat KF_simple_update(StateMat State, Matrix v, Matrix R, Matrix H);

#endif
#include "stdafx.h"

StateMat KF_simple_update(StateMat State, Matrix v, Matrix R, Matrix H)
{
    Matrix PHt = State.P * ~H;
    Matrix S = H * PHt + R;
    Matrix Si = S.Inv();
    Si = make_symmetric(Si);
    Matrix W = PHt * Si;
    State.x = State.x + W * v;
    State.P = State.P - make_symmetric(W * S * ~W);
    return State;
}
```
#ifndef _UPDATE_H_
#define _UPDATE_H_

#include "stdafx.h"

// Inputs:
// x, P - SLAM state and covariance
// z, R - range-bearing measurements and covariances
// idf - feature index for each z
// batch - switch to specify whether to process measurements together or sequentially

// Outputs:
// x, P - updated state and covariance

StateMat update(StateMat S, Matrix z, Matrix R, idf, int batch);
StateMat batch_update(StateMat S, Matrix z, Matrix R, idf);
StateMat single_update(StateMat S, Matrix z, Matrix R, idf);

#endif
```
update.cpp

#include "stdafx.h"

// Inputs:
// x, P - SLAM state and covariance
// z, R - range-bearing measurements and covariances
// batch - switch to specify whether to process measurements together or sequentially

// Outputs:
// x, P - updated state and covariance

StateMat batch_update(StateMat S, Matrix z, Matrix R);
StateMat single_update(StateMat S, Matrix z, Matrix R);

StateMat update(StateMat S, Matrix z, Matrix R, int batch){
    if (batch == 1){
        S = batch_update(S, z, R);
    } else{
        S = single_update(S, z, R);
    }

    return S;
}

StateMat batch_update(StateMat S, Matrix z, Matrix R){
    int lenz = size(z,2);
    int lenx = length(x);
    Matrix H(2*lenz, lenx);
    H.Null();
    Matrix v(2*lenz, 1);
    v.Null();
    Matrix RR(2*lenz);
    RR.Null();

    for (int i=0; i<lenz; i++){
        ii = 2*i + (-1:0);
        Matrix[zp,H(ii,:)] = observe_model(x, idf(i));
        v(ii) = [z(1,i)-zp(1);
                pi_to_pi(z(2,i)-zp(2))];
        RR(ii,ii) = R;
    }

    S = KF_simple_update(S, v, RR, H);
    return S;
}
StateMat single_update(StateMat S, Matrix z, Matrix R, idf) {
    int lenz = size(z, 2);
    for (int i = 0; i < lenz; i++) {
        [zp, H] = observe_model(x, idf(i));
        Matrix v = [z(1, i) - zp(1);
                    pi_to_pi(z(2, i) - zp(2))];
        S = KF_simple_update(S, v, R, H);
    }
    return S;
}
ekflasmsim.h

#ifndef _EKFLASMSIM_H_
#define _EKFLASMSIM_H_

#include "stdafx.h"

// function data= ekflasmsim(lm, wp)

// INPUTS:
// lm - set of landmarks
// wp - set of waypoints

// OUTPUTS:
// data - a data structure containing:
// data.true: the vehicle 'true'-path (ie, where the vehicle *actually* went)
// data.path: the vehicle path estimate (ie, where SLAM estimates the vehicle went)
// data.state(k).x: the SLAM state vector at time k
// data.state(k).P: the diagonals of the SLAM covariance matrix at time k

// NOTES:
// This program is a SLAM simulator. To use, create a set of landmarks and
// vehicle waypoints (ie, waypoints for the desired vehicle path). The program
// 'frontend.m' may be used to create this simulated environment - type
// 'help frontend' for more information.
// The configuration of the simulator is managed by the script file
// 'configfile.m'. To alter the parameters of the vehicle, sensors, etc
// adjust this file. There are also several switches that control certain
// filter options.

DataMat ekflasmsim(Matrix lm, Matrix wp, UGV curUGV);

#endif
ekfslamsim.cpp

#include "stdafx.h"

// function data= ekfslam_sim(lm, wp)

// INPUTS:
// lm - set of landmarks
// wp - set of waypoints
// OUTPUTS:
// data - a data structure containing:
// data.truepath: the vehicle 'true'-path (ie, where the vehicle *actually* went)
// data.SLAMpath: the vehicle path estimate (ie, where SLAM estimates the vehicle went)
// data.state(k).x: the SLAM state vector at time k
// data.state(k).P: the diagonals of the SLAM covariance matrix at time k

// NOTES:
// This program is a SLAM simulator. To use, create a set of landmarks and
// vehicle waypoints (ie, waypoints for the desired vehicle path). The program
// 'frontend.m' may be used to create this simulated environment - type
// 'help frontend' for more information.
// The configuration of the simulator is managed by the script file
// 'configfile.m'. To alter the parameters of the vehicle, sensors, etc
// adjust this file. There are also several switches that control certain
// filter options.

DataMat ekfslam_sim(Matrix lm, Matrix wp, UGV curUGV){

    // initialise states
    Matrix xtrue(5,1);
    xtrue.Null();
    Matrix x(5,1);
    x.Null();
    Matrix P(5,5);
    P.Null();

    // define Matrices
    Matrix Q(2,2);
    Q(0,0) = pow(sigmaV,2);
    Q(0,1) = 0;
    Q(1,0) = 0;
    Q(1,1) = pow(sigmaG,2);
    Matrix R(2,2);
    R(0,0) = pow(sigmaR,2);
    R(0,1) = 0;
    R(1,0) = 0;
    R(1,1) = pow(sigmaB,2);
// initialise other variables and constants
float dt = DT_CONTROLS;  // change in time between predicts
float dtsum = 0;         // change in time since last observation
Matrix ftag = 1:size(lm,2);  // identifier for each landmark
Matrix da_table(1,size(lm,2));  // data association table
da_table.Null();
int iwp = 1;              // index to first waypoint
float G = 0;              // initial steer angle
DataMat data(x, P, x);   // stored data for off-line
Matrix QE(Q);
Matrix RE(R);

// main loop
while (iwp != 0){

    // compute true data
    G = curUGV.getHeading();

    xtrue = vehicle_model(xtrue, V, G, WHEELBASE, dt);

    // EKF predict step
    S = predict(S, V, G, QE, WHEELBASE, dt);

    // if heading known, observe heading
    S = observe_heading(S, xtrue(3), SWITCH_HEADING_KNOWN);

    // EKF update step
    dtsum = dtsum + dt;
    if (dtsum >= DT_OBSERVE){
        dtsum= 0;
        [z,ftag_visible]= get_observations(xtrue, lm, ftag, MAX_RANGE);
        RANSAC(UGV[0], lmDB);
        S = KF_simple_update(S, zf, RE, idf, SWITCH_BATCH_UPDATE);
        S = augment(S, zn, RE);
    }

    // offline data store
    data = UpdateData(x, P, xtrue);

}
vehiclemodel.h

#ifndef _VEHICLEMODEL_H_
#define _VEHICLEMODEL_H_

#include "stdafx.h"

// INPUTS:
// xv - vehicle pose [x;y;phi]
// V - velocity
// G - steer angle (gamma)
// WB - wheelbase
// dt - change in time

// OUTPUTS:
// xv - new vehicle pose

Matrix vehicle_model(Matrix xv, float V, float G, float WB, float dt);

#endif
vehiclemodel.cpp

```cpp
#include "stdafx.h"

// INPUTS:
// xv - vehicle pose [x; y; phi]
// V - velocity
// G - steer angle (gamma)
// WB - wheelbase
// dt - change in time

// OUTPUTS:
// xv - new vehicle pose

Matrix vehicle_model(Matrix xv, float V, float G, float WB, float dt){
    Matrix newxv(5, 1); // temporary vehicle pose
    newxv(0, 0) = xv(0, 0) + V*dt*cos(G + xv(2,0));
    newxv(1, 0) = xv(1, 0) + V*dt*sin(G + xv(2,0));
    newxv(2, 0) = pi_to_pi(xv(2,0) + V*dt*sin(G)/WB);
    newxv(3, 0) = V;
    newxv(4, 0) = xv(4, 0);

    xv = newxv;

    return xv;
}
```
# Control Parameters

- \( V = 3 \) m/s
- \( MAXG = 30 \times \pi/180 \) radians, maximum steering angle \((-MAXG < g < MAXG)\)
- \( RATEG = 20 \times \pi/180 \) rad/s, maximum rate of change in steer angle
- \( WHEELBASE = 4 \) metres, vehicle wheelbase
- \( DT_{\text{CONTROLS}} = 0.025 \) seconds, time interval between control signals

## Control Noises

- \( \sigmaV = 0.3 \) m/s
- \( \sigmaG = (3.0 \times \pi/180) \) radians

## Observation Parameters

- \( MAX\_\text{RANGE} = 30.0 \) metres
- \( DT_{\text{OBSERVE}} = 8 \times DT_{\text{CONTROLS}} \) seconds, time interval between observations

## Observation Noises

- \( \sigmaR = 0.1 \) metres
- \( \sigmaB = (1.0 \times \pi/180) \) radians

## Data Association Innovation Gates (Mahalanobis Distances)

- \( GATE_{\text{REJECT}} = 4.0 \) maximum distance for association
- \( GATE_{\text{AUGMENT}} = 25.0 \) minimum distance for creation of new feature

## For 2-D Observation:

- Common gates are: 1-sigma (1.0), 2-sigma (4.0), 3-sigma (9.0), 4-sigma (16.0)
- Percent probability mass is: 1-sigma bounds 40%, 2-sigma 86%, 3-sigma 99%, 4-sigma 99.9%.
// waypoint proximity
static const float AT_WAYPOINT = 1.0;  // metres, distance from current waypoint at
    which to switch to next waypoint
static const int NUMBER_LOOPS = 2;    // number of loops through the waypoint list

// switches
static const int SWITCH_CONTROL_NOISE = 1;  // if 0, velocity and gamma are perfect
static const int SWITCH_SENSOR_NOISE = 1;   // if 0, measurements are perfect
static const int SWITCH_INFLATE_NOISE = 0;  // if 1, the estimated Q and R are inflated
    (ie, add stabilising noise)
static const int SWITCH_HEADING_KNOWN = 0;  // if 1, the vehicle heading is observed
    directly at each iteration
static const int SWITCH_ASSOCIATION_KNOWN = 0;  // if 1, associations are given, if 0, they
    are estimated using gates
static const int SWITCH_BATCH_UPDATE = 1;    // if 1, process scan in batch, if 0,
    process sequentially
static const int SWITCH_SEED_RANDOM = 0;    // if not 0, seed the randn() with its value
    at beginning of simulation (for repeatability)
static const int SWITCH_USE_IEKF = 0;       // if 1, use iterated EKF for updates, if 0,
    use normal EKF

// returns angles in the range -pi to pi
float pi_to_pi(float angle);
double pi_to_pi(double angle);

// C++ equivalent of MATLAB diag() function
Matrix diag(Matrix k);

// State - pose (x) and covariance of pose (P)
struct StateMat{ Matrix x; Matrix P; };

// State Data - for all states
class DataMat{
    private:
        vector<Matrix> truepath;  // true path
        vector<Matrix> SLAMpath;  // path of SLAM
        vector<StateMat> state;   // vector of states
        int i;                     // keeps a track of which state we’re up to

    public:

        // initialisation/constructor
        DataMat(Matrix x, Matrix P, Matrix xtrue){
            i=1;
            truepath.push_back(xtrue);
            SLAMpath.push_back(x);
            StateMat s;
            s.x = x;
s.P = diag(P);
state.push_back(s);
}

// destructor
~DataMat() {};

// get current state data
StateMat getCurrentState() {
    return state[i];
};

void UpdateData(Matrix x, Matrix P, Matrix xtrue);  
void AddOneState(StateMat s);  
void UpdateTruepath(Matrix x);  
void UpdateSLAMpath(Matrix xtrue);  
};  // data matrix
```cpp
#include "stdafx.h"
#include "SLAMUtils.h"

// Update method for data
void DataMat::UpdateData(Matrix x, Matrix P, Matrix xtrue){
    DataMat::i++;
    DataMat::UpdateSLAMpath(x);
    DataMat::UpdateTruepath(xtrue);
    StateMat s;
    s.x=x;
    s.P=diag(P);
    DataMat::AddOneState(s);
}

// Update method for the Data.state
void DataMat::AddOneState(StateMat s){
    DataMat::state.push_back(s);
}

// Update method for the Data.truepath
void DataMat::UpdateTruepath(Matrix xtrue){
    DataMat::truepath.push_back(xtrue);
}

// Update method for the Data.SLAMpath
void DataMat::UpdateSLAMpath(Matrix x){
    DataMat::SLAMpath.push_back(x);
}

///////////////////////////////////////////////////////////////////////

// returns angles in the range -pi to pi
float pi_to_pi(float angle){
    float tempangle = fmodf(angle, PI);
    return tempangle;
}

// returns angles in the range -pi to pi
double pi_to_pi(double angle){
    double tempangle = fmod(angle, PI);
    return tempangle;
}

// C++ equivalent of the MATLAB daig() function
```
Matrix diag(Matrix k){
    int row = k.RowNo();
    int col = k.ColNo();
    Matrix diagonal;

    if (col>1){
        int num = ((col < row) ? col : row);
        diagonal.SetSize(num,1);
        for(int i=0; i<num; i++){
            diagonal(i,0) = k(i,i);
        }
    }

    if (col==1){
        diagonal.SetSize(row, row);
        for (int i=0; i<row; i++){
            for(int j=0; j<row; j++){
                if (i==j){
                    diagonal(i,i) = k(i,0);
                }else{
                    diagonal(i,j) = 0;
                }
            }
        }
    }

    return diagonal;
}
predict.h

#ifndef _PREDICT_H_
#define _PREDICT_H_

#include "stdafx.h"

StateMat predict (StateMat S, float v, float g, Matrix Q, float WB, float dt);

#endif
#include "stdafx.h"

// Inputs:
// x, P - SLAM state and covariance
// v, g - control inputs: velocity and gamma (steer angle)
// Q - covariance matrix for velocity and gamma
// WB - vehicle wheelbase
// dt - timestep
// Outputs:
// xn, Pn - predicted state and covariance

StateMat predict (StateMat S, float v, float g, Matrix Q, float WB, float dt){
    float s = sin(g + S.x(2,0));
    float c = cos(g + S.x(2,0));
    float vts = v*dt*s;
    float vtc = v*dt*c;

    // jacobians
    Matrix A(5,5);
    A(0,0)=1; A(0,1)=0; A(0,2)=-vts;
    A(1,0)=0; A(1,1)=1; A(1,2)=vtc;
    A(2,0)=0; A(2,1)=0; A(2,2)=1;

    Matrix Az(3,2);
    Gu(0,0)=dt*c; Gu(0,1)=-vts;
    Gu(1,0)=dt*s; Gu(1,1)=vtc;
    Gu(2,0)=dt*sin(g)/WB; Gu(2,1)=v*dt*cos(g)/WB;

    // predict covariance
    Matrix P_15_15(5,5); // the first 5x5 matrix of P
    for(int i=0;i<5;i++)
        for(int j=0; j<5; j++){
            P_15_15(i,j)=S.P(i,j);
        }

    Matrix new_15_15(A*P_15_15*~A + Az*Q*~Az); // P(1:5,1:5)= A*P(1:5,1:5)*A^T + Az*Q*Az^T
    for(int i=0;i<5;i++)
        for(int j=0; j<5; j++){
            S.P(i,j)=new_15_15(i,j);
        }

    int row = S.P.RowNo();
    if (row>5){
Matrix P_15_6row(5, row-5); // P(1:5,6:end)
for(int i=0; i<5; i++){
    for(int j=0; j<row-5; j++){
        P_15_6row(i,j)=S.P(i,5+j);
    }
}

Matrix new_15_6row(A*P_15_6row); // P(1:5,6:end)= A*P(1:5,6:end);
for(int i=0; i<5; i++){
    for(int j=0; j<row-5; j++){
        S.P(i,5+j)=new_15_6row(i,j);
    }
}

Matrix new_6row_15(~new_15_6row); // P(6:end,1:5)= P(1:5,6:end)'
for(int i=0; i<row-5; i++){
    for(int j=0; j<5; j++){
        S.P(5+i,j)=new_4row_15(i,j);
    }
}

// predict state
S.x(0,0)=S.x(0,0)+vtc;
S.x(1,0)=S.x(1,0)+vts;
S.x(2,0)=pi_to_pi(S.x(2,0)+ v*dt*sin(g)/WB);
return S;
H.2 GCS CODE

H.2.1 MAPPING

H.2.1.1 PHYSICAL MAPS

occupancy_map.h

```cpp
#ifndef MGC_OCCUPANCY_MAP_HPP
#define MGC_OCCUPANCY_MAP_HPP

#include "mgc/map/map.hpp"
#include "mgc/exception.hpp"
#include "mgc/datatypes.hpp"
#include "mgc/map/feature_map.hpp"

#define MGC_OMAP_RAY_STEP_LENGTH 0.01f

#define MGC_OMAP_UNOCCUPIED 115

#define MGC_OMAP_OCCUPIED 143

#define MGC_OMAP_MAX_HEIGHT 20

#define MGC_OMAP_MAX_VALUE 255
#define MGC_OMAP_MIDDLE_VALUE 128
#define MGC_OMAP_MIN_VALUE 0

namespace mgc {
    namespace map {
        class occupancy_map : public map<uint8> {
            public:
                /**
                 * Define the maximum middle and minimum values of the occupancy map
                 */
                #define MGC_OMAP_MAX_VALUE 255
                #define MGC_OMAP_MIDDLE_VALUE 128
                #define MGC_OMAP_MIN_VALUE 0

                namespace mgc {
                    namespace map {
                        class occupancy_map : public map<uint8> {
                            public:
                                /**
                                 * This defines the increments in which the ray length should be increased. This can be
                                 * thought of as a % of the
                                 * unit vector length which is 1. Thus 0.05f is a step of 0.05 units, or 20 steps
                                 */
                                #define MGC_OMAP_RAY_STEP_LENGTH 0.01f

                                /**
                                 * This defines the maximum value for a cell in the occupancy map before it's considered
                                 * unoccupied
                                 */
                                #define MGC_OMAP_UNOCCUPIED 115

                                /**
                                 * This defines the minimum value for a cell in the occupancy map before it's considered
                                 * unoccupied
                                 */
                                #define MGC_OMAP_OCCUPIED 143

                                #define MGC_OMAP_MAX_HEIGHT 20

                                /**
                                 * Define the maximum middle and minimum values of the occupancy map
                                 */
                                #define MGC_OMAP_MAX_VALUE 255
                                #define MGC_OMAP_MIDDLE_VALUE 128
                                #define MGC_OMAP_MIN_VALUE 0
                            }
                        }
                    }
                }
            }
        }
    }
}
```
* This is the default constructor. It used to create a blank occupancy map of the
  specified size and
* initialise all the values to 128.
* @param[in] width The width of the occupancy map
* @param[in] height The height of the occupancy map
* @param[in] depth The depth of the occupancy map
*/
occupancy_map(uint32 width = 0, uint32 height = 0, uint32 depth = 0);

/**
* This is the default destructor. Its not currently used in this class, but is
defined virtual to ensure
that all the destructors are invoked in the inheritance tree.
*/
virtual ~occupancy_map(void);

/**
* Updates the occupancy value of given map cube
* @param[in] (i,j,k) The map coordinates of the desired occupancy cube
* @param[in] occupied True if the cube is occupied, else false if its not occupied
*/
virtual void update(int32 i, int32 j, int32 k, bool occupied);

/**
* Updates the current Occupancy map based upon a ray trace of the current LIDAR
scans
* @param scan The lidar scan points as real coordinates (X,Y,Z)
* @param origin The position of the LIDAR at the time the scan was taken.
*/
virtual void update_occupancy_ray(real_loc scan[LIDAR_SCAN_POINTS], real_loc origin);

/**
* This loads a true occupancy map from a specified feature map. This is used
primarily for testing purposes
* and is used to verify the correct behavior of the algorithms.
* @param[in] fmap The feature map from which to generate the occupancy map
*/
virtual void load(const feature_map & fmap);

/**
* This function is used to get the elevation at the point (i, j)
*/
virtual uint8 get_elevation(uint32 i, uint32 j);

/**
* This function is used to get the feature at the point (i, j)
*/
virtual uint8 get_feature(uint32 i, uint32 j);
# endif // MGC_OCCUPANCY_GRID
occupancy_map.cpp

```cpp
#include "mgc/map/occupancy_map.hpp"
#include <stdlib.h>
#include "string"
#include "iostream"
#include <memory.h>
#include "mgc/map/map_functions.hpp"

using namespace mgc;
using namespace mgc::map;

occupancy_map::occupancy_map(uint32 width, uint32 height, uint32 depth) : map<uint8>(width, height, depth)
{
    try
    {
        // Clear the entire occupancy map to 128
        clear(128);
    }
    catch(exception & ex)
    {
        // Pass any exceptions caught up to generate a proper stack trace
        MGC_THROW_EXCEPTION_UP(ex);
    }
}

occupancy_map::~occupancy_map(void)
{
}

void occupancy_map::update(int32 i, int32 j, int32 k, bool occupied)
{
    try
    {
        // Check if the current square is occupied or not
        if(occupied == true)
        {
            uint8 cur_val = get_value(i, j, k);
            // Check to make sure we get an overflow error (Note we consider evidence of a
            // wall to be twice that of open spaces
            uint8 new_val = get_value(i, j, k) < 253 ? get_value(i, j, k) + 2 : get_value(i, j, k);
            // If it is, then increment it's value in the map
            set_value(new_val, i, j, k);
        }
        else
        {
            // Check to make sure we get an overflow error
            uint8 new_val = get_value(i, j, k) > 0 ? get_value(i, j, k) - 1 : get_value(i, j, k);
        }
    }
```
void occupancy_map::update_occupancy(real_loc scan[LIDAR_SCAN_POINTS], real_loc origin)
{
    try
    {
        // Update the occupancy map with the given LIDAR scans
        for (uint32 i = 0; i < LIDAR_SCAN_POINTS; i++)
        {
            // Calculate the length of the ray
            float dist = (scan[i] - origin).length();

            // Calculate the step vector
            real_loc step = (origin - scan[i]) * MGC_OMAP_RAY_STEP_LENGTH / dist;

            // Calculate the first point on the ray
            real_loc cur_loc = origin;

            // The current grid location
            grid_loc cur_grid_loc;

            // The current distance traveled along the ray
            float cur_dist = 0.0f;

            // Transverse the ray until the intersection is reached
            while (cur_dist < dist)
            {
                // Get the map coordinate of the
                cur_grid_loc = cur_loc.to_grid_loc();

                // Update the current position. Since the lidar didn’t stop here we know that
                // it is not occupied
                update(cur_grid_loc.i, cur_grid_loc.j, cur_grid_loc.k, false);

                // Add the step to the ray origin
                cur_loc += step;

                // Update the current distance
                cur_dist = (cur_loc - origin).length();
            }

            // Update the final map location where the lidar stopped
            update(cur_grid_loc.i, cur_grid_loc.j, cur_grid_loc.k, true);
        }
    }
    catch(exception & ex)
    {
        // Pass any exceptions caught up to generate a proper stack trace
        MGC_THROW_EXCEPTION_UP(ex);
    }
}
catch(exception & ex)
{
    // Pass any exceptions caught up to generate a proper stack trace
    MGC_THROW_EXCEPTION_UP(ex);
}
};

void occupancy_map::load(const feature_map & fmap)
{
    try
    {
    // Check if the feature map and occupancy dimensions match
        if(get_width() != fmap.get_width() || get_height() != fmap.get_height())
        {
            // If the dimensions differ, then resize it
            resize(fmap.get_width(), fmap.get_height(), get_depth());
        }
    }
    // Iterate through every row
    for(uint32 j = 0; j < fmap.get_height(); j++)
    {
    // Iterate through every column
        for(uint32 i = 0; i < fmap.get_width(); i++)
        {
        // Get the current feature
            uint8 cur_feature = fmap.get_value(i, j);
            switch (cur_feature)
            {
            case (MGC_FEATURE_OPEN): // Set the occupancy of each occupancy cube to represent open space
                for (uint32 k = 0; k < get_depth(); k++)
                {
                    if (k <= MGC_GROUND_LEVEL){
                        set_value(MGC_OMAP_MAX_VALUE, i, j, k); // Below ground
                    } else {
                        set_value(MGC_OMAP_MIN_VALUE, i, j, k); // Open above ground
                    }
                }
                break;
            case (MGC_FEATURE_BLOCKED): // Set the occupancy of each occupancy cube to represent occupied space
                for (uint32 k = 0; k < get_depth(); k++)
                {
                    set_value(MGC_OMAP_MAX_VALUE, i, j, k); // All Occupied Space
                }
                break;
            case (MGC_FEATURE_UNKNOWN): // Set the occupancy of each occupancy cube to represent occupied space
                for (uint32 k = 0; k < get_depth(); k++)
                {
                }
```c++
set_value(MGC_OMAP_MIDDLE_VALUE, i, j, k); // Occupied Space Below ground
break;

case (MGC_FEATURE_SEMIBLOCKED): // Set the occupancy of each occupancy cube to represent open space
for (uint32 k = 0; k < get_depth(); k++)
{
    if (k <= MGC_SEMI_BLOCKED_LEVEL)
    {
        set_value(MGC_OMAP_MAX_VALUE, i, j, k); // Below ground
    } else {
        set_value(MGC_OMAP_MIN_VALUE, i, j, k); // Open above ground
    }
}
break;

case (MGC_FEATURE_DITCH): // Set the occupancy of each occupancy cube to represent open space
for (uint32 k = 0; k < get_depth(); k++)
{
    if (k <= MGC_DITCH_LEVEL)
    {
        set_value(MGC_OMAP_MAX_VALUE, i, j, k); // Below ground
    } else {
        set_value(MGC_OMAP_MIN_VALUE, i, j, k); // Open above ground
    }
}
break;
// Note other levels may need to be set here

uint8 occupancy_map::get_elevation(uint32 i, uint32 j)
{
    int max_occupied = 0;
    int min_unoccupied = 20;
    // Yet to implement how to get the elevation
    for (int k=0; k<MGC_MAX_LEVEL; k++)
    {
        if (get_value(i,j,k) < MGC_OMAP_UNOCCUPIED)
        {
            if (k<min_unoccupied)
            {
                min_unoccupied = k;
            }
        }
    }
    return max_occupied;
}
```
```c
if (get_value(i,j,k) > MGC_OMAP_OCCUPIED)
{
    if (k>maxoccupied)
        maxoccupied = k;
}

if (minunoccupied < MGC_SEMI_BLOCKED_LEVEL)
{
    return min.unoccupied >= 1 ? min.unoccupied - 1 : min.unoccupied;
}

if (maxoccupied > MGC_GROUND_LEVEL)
    return maxoccupied;

return MGC_GROUND_LEVEL;
```

```c
uint8 occupancy_map::get_feature(uint32 i, uint32 j)
{
    int maxoccupied = 0;
    int minunoccupied = 20;

    for (int k=0; k<MGC_MAX_LEVEL; k++)
    {
        if (get_value(i,j,k) < MGC_OMAP_UNOCCUPIED)
            if (k<minunoccupied)
                minunoccupied = k;
        if (get_value(i,j,k) > MGC_OMAP_OCCUPIED)
            if (k>maxoccupied)
                maxoccupied = k;
    }

    if (minunoccupied <= MGC_GROUND_LEVEL ){
        if (minunoccupied < MGC_GROUND_LEVEL ) {
            return MGC_FEATURE_DITCH;
        } else {
            return MGC_FEATURE_OPEN;
        }
    }

    if (maxoccupied > MGC_GROUND_LEVEL ){
        if (maxoccupied > MGC_SEMI_BLOCKED_LEVEL ){
            return MGC_FEATURE_BLOCKED;
        } else {
            return MGC_FEATURE_SEMIBLOCKED;
        }
    }

    return MGC_FEATURE_UNKNOWN;
}
```
elevation_map.h

#ifndef MGC_ELEVATION_MAP_HPP
#define MGC_ELEVATION_MAP_HPP

#include "mgc/map/map.hpp"
#include "mgc/exception.hpp"
#include "mgc/datatypes.hpp"
#include "mgc/map/occupancy_map.hpp"

namespace mgc
{
  namespace map
  {
    class elevation_map : public map<uint8>
    {
      public:
        /**
         * This constructor is used to create a map of the specified sizes.
         * @param[in] width The width of the elevation map
         * @param[in] height The height of the elevation map
         */
        elevation_map(uint32 width = 0, uint32 height = 0);

        /**
         * This is the destructor, it is not currently in use, but is added for
         * completion. This is mostly
         * required to be defined as virtual so it can be invoked by any inheriting
         * classes upon destruction.
         */
        virtual ~elevation_map(void);

        /**
         * This function is used to load the entire elevation map from a occupancy map
         * @param[in] omap The occupancy map from which to generate the elevation
         * map
         */
        virtual void load(const occupancy_map & omap);
    }
  }
}
#endif // MGC_ELEVATION_MAP_HPP
```cpp
#include "mgc/map/elevation_map.hpp"
#include <memory.h>

using namespace mgc;
using namespace mgc::map;

elevation_map::elevation_map(uint32 width, uint32 height): map<uint8>(width, height)
{
    try
    {
        // Clear the entire map to ground level
        clear((uint8)MGC_GROUND_LEVEL);
    }
    catch(exception & ex)
    {
        // Throw any exceptions generated up
        MGC_THROW_EXCEPTION_UP(ex);
    }
}

elevation_map::~elevation_map(void)
{
}

void elevation_map::load(const occupancy_map & omap)
{
    try
    {
        // Check if the map dimensions match
        if(get_width() != omap.get_width() || get_height() != omap.get_height())
        {
            // Resize the map
            resize(omap.get_width(), omap.get_height());
        }

        // Iterate through all the rows
        for(uint32 j = 0; j < get_height(); j++)
        {
            // Iterate through all the columns
            for(uint32 i = 0; i < get_width(); i++)
            {
                // Declare the depth variable k here so it can be used outside the loop
                uint32 k = 0;
                bool unknown = 0;

                // Iterate through all the depth values
                for(; k < omap.get_depth(); k++)
                {
                    // Get the current cell occupancy
                    uint8 occupancy = omap.get_value(i, j, k);
                    // Check if it is below the minimum unoccupied threshold value
                }
            }
        }
    }
    catch(exception & ex)
    {
        // Throw any exceptions generated up
        MGC_THROW_EXCEPTION_UP(ex);
    }
}
```

if (occupancy < MGC_OMAP_OCCUPIED)
{
    if (occupancy < MGC_OMAP_UNOCCUPIED)
    {
        unknown = 0;
        break; // This is the first open square, so defines elevation
    } else {
        unknown = 1;
    }
}

// Set the elevation to be k - 1 for the elevation value i, j
//set_value(k > 0 ? k - 1 : 0, i, j); // This would just be occupancy accuracy
if (unknown)
{
    set_value((uint8) MGC_GROUND_LEVEL, i, j);
    //set_value((uint8) round(MGC_GROUND_LEVEL*12.8f), i, j);
} else {
    set_value(k > 0 ? (uint8) k - 1 : 0, i, j);
    //set_value(k > 0 ? (uint8) round((k-1)*12.8f): 0, i, j); // These are
    value for 20 levels over the 255 spread
}

} catch (exception & ex)
{
    // Pass any exceptions up where it can be dealt with
    MGC_THROW_EXCEPTION_UP(ex);
}
visibility_map.h

```c
#ifndef MGC_VISIBILITY_MAP_HPP
#define MGC_VISIBILITY_MAP_HPP

#include "mgc/exception.hpp"
#include "mgc/datatypes.hpp"
#include "mgc/ugv.hpp"
#include "vector"

namespace mgc {

class visibility_map {

public:

  visibility_map(int32 width, int32 height);
  ~visibility_map(void);

  uint8  get_value(int32 i, int32 j);
  uint8  get_uav_value(int32 i, int32 j);
  void   generate(std::vector<ugv> ugvs, uint8* feature_map);
  int32  get_width(void) { return _width; };
  int32  get_height(void) { return _height; };

protected:

  uint8*  _visibility;
  uint8*  _uav_visibility;
  int32   _width;
  int32   _height;

};

#endif // MGC_VISIBILITY_MAP_HPP
```
visibility_map.cpp

#include "mgc/map/visibility_map.hpp"

using namespace mgc;

visibility_map::visibility_map(int32 width, int32 height)
{
    _width = width;
    _height = height;
    _visibility = new uint8[ _width * _height];
    memset(_visibility, 0, _width * _height);
};

visibility_map::~visibility_map(void)
{
    MGC_SAFE_DELETE_ARRAY(_visibility);
};

uint8 visibility_map::get_value(int32 i, int32 j)
{
    return _visibility[j*_width+i];
};

uint8 visibility_map::get_uav_value(int32 i, int32 j)
{
    return _uav_visibility[j*_width+i];
};

void visibility_map::generate(std::vector<ugv> ugvs, uint8* feature_map)
{
### H.2.1.2 CONCEPTUAL MAPS

**exploration_map.h**

```c++
#ifndef MGC_EXPLORATION_MAP_HPP
#define MGC_EXPLORATION_MAP_HPP

#include "mgc/datatypes.hpp"
#include "mgc/map/map.hpp"

namespace mgc {
    namespace map {
        class exploration_map : public map<uint8> {
            public:
                exploration_map(void);
                exploration_map(uint32 width, uint32 height);
                ~exploration_map(void);
                bool get_explored(uint32 i, uint32 j);
                uint8 get_raw_value(uint32 i, uint32 j);
                void update(uint32 i, uint32 j);
                void reset(uint32 i, uint32 j);
                uint8 get_threshold(void) { return _exploration_threshold; }

            protected:
                uint8 _exploration_threshold;
                uint32 * _argb;
            }
        }
    }
}
#endif //MGC_EXPLORATION_MAP_HPP
```
exploration_map.cpp

```cpp
#include "mgc/map/exploration_map.hpp"
#include <string.h>
#include <stdlib.h>

using namespace mgc;
using namespace mgc::map;

exploration_map::exploration_map(void): map<uint8>({0, 0}), _exploration_threshold(MGC_EXPLORE_THRESH), _argb(NULL)
{
    
};
exploration_map::exploration_map(uint32 width, uint32 height) : map<uint8>({width, height}), _exploration_threshold(MGC_EXPLORE_THRESH), _argb(NULL)
{
    try
    {
        // Clear the entire exploration map to 0
        clear(0);
    }
    catch(exception & ex)
    {
        // Pass any exceptions caught up to generate a proper stack trace
        MGC_THROW_EXCEPTION_UP(ex);
    }
    /*_argb = new uint32[width * height];
    memset(_argb, 128, width * height * sizeof(uint32));*/
}
exploration_map::~exploration_map(void)
{
}

/** Returns whether a location has been viewed enough to be considered explored.
 * @param i The i coordinate
 * @param j The j coordinate
 * @return The exploration state.
 */
bool exploration_map::get_explored(uint32 i, uint32 j)
{
    if (get_value(i, j) > _exploration_threshold)
        return true;
    return false;
}
/** Return the actual value in the exploration map
 * 
*/
```
```cpp
uint8 exploration_map::get_raw_value(uint32 i, uint32 j) {
    return get_value(i, j);
}

void exploration_map::update(uint32 i, uint32 j) {
    uint8 val = get_value(i, j);
    if (val < 250) set_value(++val, i, j);
}

void exploration_map::reset(uint32 i, uint32 j) {
    set_value(0, i, j);
}
```
#ifndef MGC_DATATYPES_HPP
#define MGC_DATATYPES_HPP

#include "mgc/config.hpp"
#include <math.h>

#define MGC_SAFE_DELETE_PTR(x) if(x != NULL) delete x; x = NULL;
#define MGC_SAFE_DELETE_ARRAY(x) if(x != NULL) delete [] x; x = NULL;

//namespace mgc
{
    typedef unsigned char uint8;
    typedef unsigned short uint16;
    typedef unsigned int uint32;
    typedef unsigned long long uint64;
    typedef unsigned int fuint;

    typedef char int8;
    typedef short int16;
    typedef int int32;
    typedef long long int64;
    typedef int fint;
}

typedef void * (*mgc_pthread_func_ptr)(void * params);

/** This macro is used to align the memory to specific byte boundaries */
#define MGC_ALIGN_MEMORY(ptr, align) (void*)((int)((unsigned char *)ptr + align) & ~(align -1))

/** Fastest signed / unsigned integer that is at least 32 bits long */
typedef int fint32;
typedef unsigned int fuint32;

/** Fastest signed / unsigned integers that is at least 16 bits long */
typedef int fint16;
typedef unsigned int fuint16;

/** Faster signed / unsigned integers that is at least 8 bits long */
typedef char fint8;
typedef unsigned char fuint8;

#ifndef NULL
#define NULL 0x00
#endif // NULL

#define MGC_ASCII_00 x30

// Define pi and 2*Pi
// Define Expected Map ratios
#define GRID_X 10 // Grid squares per meter
#define GRID_Y 10 // Grid squares per meter
#define GRID_Z 10 // Grid squares per meter
#define XOFFSET 0
#define YOFFSET 0
#define MAX_ELEVATION 1.5f // Highest allowed elevation
#define MIN_ELEVATION -0.5f // Lowest allowed elevation

namespace mgc
{
  float inline round(float r)
  {
    return (r > 0.0f) ? floor(r + 0.5f) : ceil(r - 0.5f);
  }
}

// Define Magic Data structures
struct grid_loc
{
  uint32 i, j, k; // Location relative to the top left corner
  grid_loc(uint32 i = 0, uint32 j = 0, uint32 k = 0) : i(i), j(j), k(k) {};
};

/**
 * Location relative to the base station (X East, Y North, Z elevation)
 */
struct real_loc
{
  float x, y, z;
  real_loc(float x = 0.0f, float y = 0.0f, float z = 0.0f) : x(x), y(y), z(z) {};
  real_loc operator - (const real_loc & rl) { return real_loc(x - rl.x, y - rl.y, z - rl.z); }
  real_loc operator + (const real_loc & rl) { return real_loc(x + rl.x, y + rl.y, z + rl.z); }
  real_loc operator * (float val) { return real_loc(x * val, y * val, z * val); }
  real_loc operator / (float val) { return real_loc(x / val, y / val, z / val); }
  float length(void) { return sqrt(x * x + y * y + z * z); }
  void operator += (const real_loc & rl) { x += rl.x; y += rl.y; z += rl.z; }
  grid_loc to_grid_loc(void)
  {
    return grid_loc((uint32)(x * GRID_X+XOFFSET), (uint32)(y * GRID_Y+YOFFSET),
                     (uint32)mgc::round((z - MIN_ELEVATION)*GRID_Z));
  }
};

struct angles
{
  float roll, pitch, yaw;
  angles(float r = 0, float p = 0, float y = 0) : roll(r), pitch(p), yaw(y) {};
}
struct ptangles{
    float pan, tilt;
    ptangles(float p = 0, float t = 0) : pan(p), tilt(t) {};
};  // Angles relative to Y North, X East, Z elevation

struct ooi_info {  // Information about potential Objects Of Interest
    int32 id;
    real_loc location;
    uint8 type;
    float detection_range;
    float confidence;
    //uint64 timestamp;
    ooi_info(void) : id(0), location(), type(0), detection_range(0), confidence(0) {};
};

struct ugv_info {  // Information about the UGV
    int32 id;
    real_loc ugv_location;
    angles ugv_angles;
    ptangles ugv_ptangles;
    //uint64 timestamp;
    ugv_info(void) : id(0), ugv_location(), ugv_angles(), ugv_ptangles() {};
};

// Define OOI types
#define DETONATION_RANGE 10.0f
#define STATIC_DETECTION 2.5f
#define MOBILE_DETECTION 10.0f
#define OOI_STATIC_THREAT 0
#define OOI_MOBILE_THREAT 1
#define OOI_MOBILE_SAFE 2
#define OOI_MOBILE_JUDGE 3

// Define Lidar information
#define MGC_LIDAR_SCAN_POINTS 529
#define LIDAR_SCAN_POINTS 529
#define LIDAR_ANGLE 0.36f
#define LIDAR_START_ANGLE -95.04f
#define LIDAR_LIMIT 60.0f

struct raw_lidar {  // Structure to contain raw Lidar Data
    uint16 data[LIDAR_SCAN_POINTS];
};

struct lidar_locations {  // Structure to contain raw Lidar Data
    real_loc data[LIDAR_SCAN_POINTS];
};

#define MGC_EXPLORE_THRESH 10  // Level at which a map square is considered explored

// Define the map levels for certain features (assumes GRID.Z is 10 and elevation max 1.5m min
-0.5m.
#define MGC_MAX_LEVEL 19
#define MGC_GROUND_LEVEL 5
#define MGC_SEMI_BLOCKED_LEVEL 11
#define MGC_DITCH_LEVEL 2
#define MGC_GROUND_LEVEL_MM 500 // Used for elevation map in mm

#define MGC_CONFIGURATION 4 // The grid squares around a blocked region that a robot should not enter

// Define the feature types
#define MGC_FEATURE_BLOCKED 0 // A region the UGV can’t see over
#define MGC_FEATURE_SEMIBLOCKED 1 // A region that is more than 60cm high
#define MGC_FEATURE_DITCH 2 // A region the UGV can see over, but not drive over
#define MGC_FEATURE_UNNAVIGABLE 3 // A region the UGV can’t drive over
#define MGC_FEATURE_UNKNOWN 6 // An unknown region of space
#define MGC_FEATURE_CURB 7 // A map region where the ground changes suddenly by 10-20cm
#define MGC_FEATURE_RAMP 8 // A ramp region. Consult the elevation map to determine ends
#define MGC_FEATURE_OPEN 9 // A region of open space with unknown ground

#endif // MGC_DATATYPES_HPP
feature_map.h

```cpp
#ifndef MGC_FEATURE_MAP_HPP
#define MGC_FEATURE_MAP_HPP

#include "mgc/map/map.hpp"
#include "mgc/datatypes.hpp"
#include "mgc/log_abstract.hpp"
#include "mgc/progress_abstract.hpp"

namespace mgc {
    namespace map {
        class feature_map : public map<uint8>
        {
            public:
                /**
                 * This constructor will create an unexplored feature map of the specified size
                 * @param [in] width The width of the feature map
                 * @param [in] height The height of the feature map
                 */
                feature_map(int32 width = 0, int32 height = 0);

                /**
                 * This function is currently unused, but is included for completeness
                 */
                virtual ~feature_map(void);

                /**
                 * This function is used to load a template feature map from file. This is
                 * predominantly for testing of the
                 * algorithms which use the feature map.
                 * @param file_name The file name and path of where the map is located
                 * @param log A pointer to the log object used
                 * @param progress A pointer to the progress object used
                 */
                virtual void load(const char * file_name, log_abstract * log = NULL,
                                   progress_abstract * progress = NULL);

                /**
                 * This function is used to update the feature map. Currently it only have to
                 * update the feature at a point
                 * @param i The map coordinate i
                 * @param j The map coordinate j
                 * @param feature Set the feature at coordinate (i, j)
                 */
                virtual void update(int32 i, int32 j, uint8 feature);
        }
    }
}
#endif // MGC_FEATURE_MAP_HPP
```
```cpp
#include <fstream>
#include <stdlib.h>
#include <string.h>
#include <iostream>

#include "mgc/map/feature_map.hpp"
#include "mgc/exception.hpp"
#include <memory.h>

using namespace mgc;
using namespace mgc::map;

feature_map::feature_map(int32 width, int32 height) : map<uint8>(width, height)
{
    // Clear the map
    clear((uint8)MGC_FEATURE_UNKNOWN);
}

feature_map::~feature_map(void)
{
}

void feature_map::load(const char * file_name, log_abstract * log, progress_abstract * progress)
{
    try
    {
        // Tell the user we are about to start loading the map file
        MGC_PROGRESS_PRINT(progress, "Attempting to load feature map — " << file_name);
        MGC_LOG_APPEND(log, "Attempting to load feature map — " << file_name);

        // Open the file to read from
        std::ifstream map_file(file_name);

        // Check if the file was opened. Since the map file should confidently exist by the
        // time this function is
        // called, we'll just throw an exception.
        if(map_file.is_open() == false)
        {
            MGC_THROW_EXCEPTION("Failed to open map file: " << file_name);
        }

        // A temporary string to store the information read
        std::string cur_line;

        // Get the height of the map
        std::getline(map_file, cur_line, '\n');
        uint32 height = (uint32)atoi(cur_line.c_str());

        // Get the width of the map
```
std::getline(map_file, cur_line, '\n');
uint32 width = (uint32)atoi(cur_line.c_str());

// Resize the pixel map
resize(width, height);
//MGC_LOG_APPEND(log, "Height: " << height);
//MGC_LOG_APPEND(log, "Width: " << width);
MGC_LOG_APPEND(log, "Height: " << get_height());
MGC_LOG_APPEND(log, "Width: " << get_width());

// Print a message to indicate that we are starting to load the map data
MGC_LOG_APPEND(log, "Starting to process map data");

// Set the size of the progress bar
MGC_PROGRESS_SET_MAX(progress, get_height());

// Track the current row count
uint32 row_count = 0;

// Read the rest of the file, line by line
while(map_file.eof() == false)
{
    // Make sure the row count does not exceed the map height
    if(row_count > get_height())
    {
        //MGC_LOG_APPEND(log, "Map height in header: " << _height << ", does not
        // match actual data height" <<
        // (int)row_count);
        MGC_THROW_EXCEPTION("Map height in header: " << get_height() << ", does not
        match actual data height" <<
        (int)row_count);
    }

    // Read the current line
    std::getline(map_file, cur_line, '\n');

    // If there is no data to process, then jump out of string
    if(cur_line.length() == 0)
        break;

    // Check if the current line width match the file width, if it doesn't then the
    // headers is wrong
    if(cur_line.length() != (uint32)get_width())
    {
        //MGC_LOG_APPEND(log, "Map width in header: " << _width << ", does not match
        // actual data width: " <<
        // (int)cur_line.length());
        MGC_THROW_EXCEPTION("Map width in header: " << get_width() << ", does not
        match actual data width: " <<
        (int)cur_line.size());
```cpp
uint32 i = 0;
std::string::iterator iter = cur_line.begin();
while(iter != cur_line.end() && i < get_width() && j < get_height())
{
    // Set the map value
    set_value((*iter++) - MGC_ASCII_0, i, j);
    // Increment the column count
    ++i;

    // Increment the row count
    ++j;

    // Increment the progress
    MGC_PROGRESS_INC(progress);
    // Increment the row count
    ++row_count;

    // Check if the user decided to cancel
    if(MGC_PROGRESS_CANCELLED(progress))
    {
        map_file.close();
        MGC_LOG_APPEND(log, "Map loading was cancelled by the user");
        return;
    }

    // Print a message to indicate we finished loading it
    MGC_LOG_APPEND(log, "Finished loading map data");
    // Close the file
    map_file.close();
}
```

```cpp
void feature_map::update(int i, int j, uint8 feature)
{
    try
    {
        // Set the value to the map
    }
```
    set_value(feature, i, j);
    }
    catch(exception & ex)
    {
        // Throw the exception back up
        MGC_THROW_EXCEPTION_UP(ex);
    }
}
#ifndef MGC_VANTAGE_MAP_HPP
#define MGC_VANTAGE_MAP_HPP

#include "mgc/datatypes.hpp"
#include "mgc/map/feature_map.hpp"
#include <memory.h>

/**
 * Due to the nature of the vantage calculations, and their relatively high processing
 * requirements, it was decided that
 * the code should not be written in C++ but instead in C. Although this is not pretty, this
 * is definitely the faster
 * method since it avoid a lot of the C++ concepts and practices that make it flexible. These
 * functions are however
 * encapsulate by the vantage_map class and interface is provided where necessary.
 */

extern "C"
{
/**
 * This structure stores the vantage mask, which is the "wheel" like template that is
 * applied for each pixel in
 * order to calculate vantage.
 */
typedef struct _mgc_vantage_mask
{
    int32  ray_count;
    int32  ray_length;
    int32 * coords;
} mgc_vantage_mask;

/**
 * This structure stores the "vantage" of every pixel in the feature map.
 */
typedef struct _mgc_vantage_map
{
    int32 width;
    int32 height;
    uint32 * vantage;
} mgc_vantage_map;

/**
 * This structure stores the content of the feature map. This is initialised using the
 * fields of the feature map
 * when calculating the vantage.
 */
typedef struct _mgc_feature_map
{
    const int32 width;
    const int32 height;
    const uint8 * features;
}
/**
 * This function is used to calculate the vantage mask used.
 */
void mgc.calc_vantage_mask(mgc_vantage_mask * mask);

/**
 * This function is used to calculate the vantage of a point x, y. Note that the
 * coordinates x, y are specified in
 * map coordinates which are assumed to be bounded by [0,0] and [0 + width, 0 + height]
 */
fuint16 mgc.calc_vantage(fuint16 x, fuint16 y, mgc_vantage_mask * mask, mgc_feature_map * fmap);

/**
 * This function is used to calculate vantage for a region
 */
void mgc.calc_vantage_region(fuint16 x0, fuint16 y0, fuint16 x1, fuint16 y1, mgc_vantage_mask * mask,
                           mgc_feature_map * fmap, mgc_vantage_map * vmap);

namespace mgc
{
    namespace map
    {
        class vantage_map : public map<uint32>
        {
            public:
            /**
             * This is the default constructor, it creates a vantage map of the specified
             * width and height.
             * @param[in] width The width of the map
             * @param[in] height The height of the map
             */
            vantage_map(uint32 width = 0, uint32 height = 0, uint32 ray_count = 32, uint32 ray_length = 100);

            /**
             * This is the destructor. It is not currently in use but is added for
             * completedness.
             */
            virtual ~vantage_map(void);

            /**
             * This function is used to update the entire vantage map from the specified
             * feature map.
             * @param[in] fmap The featuremap to be used for the map update
             */
            virtual void update(const feature_map & fmap);

            /**
             * This function is used to update only a specific region of the vantage map
             */
        }
    }
}
* @param[in] fmap The feature map to use for the update
* @param[in] i The i coordinate where the update region start
* @param[in] j The j coordinate where the update region start
* @param[in] width The width of the update region
* @param[in] height The height of the update region
*/

virtual void update(const feature_map & fmap, uint32 i, uint32 j, uint32 width,
                     uint32 height);

protected:
  mgc_vantage_mask _vmask;
  uint32    _max_vantage;
};

#endif // MGC_VANTAGE_GRID_HPP
vantage_map.cpp

```cpp
#include <math.h>

#include "mgc/map/vantage_map.hpp"
#include "mgc/exception.hpp"
#include "mgc/map/map_functions.hpp"
#include <math.h>

#include <stdio.h>
#include <stdlib.h>

using namespace mgc;
using namespace mgc::map;

void mgc_calc_vantage_mask(mgc_vantage_mask * mask)
{
    /* Get a pointer to the first ray's coordinate position */
    fint16 * cur = mask->coords;

    /* A counter to keep track of the spokes */
    fint16 s = 0;

    while(s < mask->ray_count)
    {
        /* A counter to keep track of the points in the spoke */
        fint16 p = 0;

        // Calculate the angle
        float angle = MGC_TWO_PI * (float)s / (float)mask->ray_count;

        // Calculate the unit vector in those directions
        float dx = cos(angle);
        float dy = sin(angle);

        // Calculate all the points on the ray
        while(p < mask->ray_length)
        {
            /* Pre-increment is faster */
            ++p;
            (*cur) = (fint16)mgc::round(p * dx);
            ++cur;
            (*cur) = (fint16)mgc::round(p * dy);
            ++cur;
        }
    }
    /* Pre-increment is faster than post increment */
    ++s;
}
```

inline fuint16 mgc_calc_vantage(fuint16 x, fuint16 y, mgc_vantage_mask * mask, mgc_feature_map * fmap)
{
    /* Initialize the vantage for the current point [x, y] */
    fuint16 vant = 0;

    /* Get a pointer to the first value in the mask */
    fint16 * cur = mask->coords;

    /* Get a pointer to the last value in the mask */
    fint16 * end = (mask->coords) + (((mask->ray_count) * (mask->ray_length)) << 1);

    fint16 ray_offset = ((mask->ray_length) << 1);

    /* Get a pointer to the star of the next ray */
    fint16 * next_ray = cur + ray_offset;

    while(cur < end)
    {
        fint16 tx;
        fint16 ty;
        while(cur < next_ray)
        {
            /* Calculate the current x, y coordinate of the point */
            tx = x + (*cur++);
            ty = y + (*cur++);

            /* Check if the points are in bounds, then check if its transparent */
            if(tx > -1 && ty > -1 && tx < (fint16)(fmap->width) && ty < (fint16)(fmap->height)
                && fmap->features[ty * fmap->width + tx] >= MGC_FEATURE_UNKNOWN)
            {
                /* Increment the vantage of the point */
                ++vant;
                continue;
            }
            else
            {
                /* Set the pointer to start at the next ray */
                cur = next_ray;
            }
        }

        /* Set the next ray end */
        next_ray = next_ray + ray_offset;
    }

    /* Return the vantage of the point */
    return vant;
}

void mgc_calc_vantage_region(fuint16 x0, fuint16 y0, fuint16 x1, fuint16 y1, mgc_vantage_mask * mask,
                              mgc_feature_map * fmap, mgc_vantage_map * vmap)
```cpp
uint16 y = y0;
while(y < y1)
{
    uint16 x = x0;
    while(x < x1)
    {
        /* Calculate the vantage for the point */
        vmap->vantage[y * vmap->width + x] = mgc_calc_vantage(x, y, mask, fmap);
        ++x;
    }
    ++y;
}
```

```cpp
vantage_map::vantage_map(uint32 width, uint32 height, uint32 ray_count, uint32 ray_length) :
    map<uint32>(width, height)
{
    try
    {
        // Create the vantage mask
        _vmask.ray_count = ray_count;
        _vmask.ray_length = ray_length;
        _vmask.coords = new fint16[(_vmask.ray_count * _vmask.ray_length + sizeof(fint16)) * 2];
        mgc_calc_vantage_mask(&_vmask);
        _max_vantage = _vmask.ray_count * _vmask.ray_length;
    }
    catch(std::exception & ex)
    {
        MGC_THROW_EXCEPTION(ex.what());
    }
}
```

```cpp
void vantage_map::update(const feature_map & fmap)
{
    try
    {
        // Update the entire vantage map
        update(fmap, 0, 0, get_width(), get_height());
    }
    catch(exception & ex)
    {
        // Throw the exception up
        MGC_THROW_EXCEPTION_UP(ex);
    }
}
```
void vantage_map::update(const feature_map & fmap, uint32 i, uint32 j, uint32 width, uint32 height)
{
    // Check if there is anything to be done at all for the vantage map
    if(get_width() == 0 || get_height() == 0 || get_depth() == 0)
    {
        return;
    }

    // Make sure the i index is in range
    if(i >= get_width())
    {
        MGC_THROW_EXCEPTION("Index out of bounds − i = = " << i << " >= " << get_width());
    }

    // Make sure the j index is in range
    if(j >= get_height())
    {
        MGC_THROW_EXCEPTION("Index out of bounds − j = = " << j << " >= " << get_height());
    }

    // Make sure the region is in the map bounds
    width = i + width < get_width() ? width : get_width() - i;

    // Make sure the height is in the map bounds
    height = j + height < get_height() ? height : get_height() - j;

    // Check that the width of the feature map matches the width of the vantage map
    if(get_width() != fmap.get_width())
    {
        MGC_THROW_EXCEPTION("Feature map and Vantage map widths does not match − vmap: " << get_width() << ", fmap: " << fmap.get_width());
    }

    // Check if the height of the feature map matches the height of the vantage map
    if(get_height() != fmap.get_height())
    {
        MGC_THROW_EXCEPTION("Feature map and Vantage map heights does not match − vmap: " << get_height() << ", fmap: " << fmap.get_height());
    }

    // Check if the depths of the feature map matches the depth of the vantage map
    if(get_depth() != fmap.get_depth())
    {
        MGC_THROW_EXCEPTION("Feature map and Vantage map depths does not match − vmap: " << get_depth() << ", fmap: " << fmap.get_depth());
    }

    // Create a temp feature map struct
    mgc_feature_map fmap_struct = { fmap.get_width(), fmap.get_height(), fmap.get_data()};
// Create a temp vantage map struct
mgc.vantage.map vmap_struct = {get_width(), get_height(), (uint32*)get_data()};

// Calculate the vantage for the map
mgc.calc.vantage.region(i, j, width, height, &vmask, &fmap_struct, &vmap_struct);
}

/*int32 vantage.map::get_width(void)
 { //return _width;
 }

int32 vantage.map::get_height(void)
 { //return _height;
 }*/

/*void vantage.map::set_value(int32 i, int32 j, uint32 val)
 { //_vantage[i * _width + j] = val;
 }

/*uint32 vantage.map::get_value(int32 i, int32 j)
 { return _vantage[i * _width + j];
 }

/*void vantage.map::resize(int32 width, int32 height)
 { // Make sure the specified width is valid
   if(width < 0)
     { MGC_THROW_EXCEPTION("Vantage map width must be greater than: 0px, requested width: 
       " << width << "px!" );
   }
   // Make sure the specified height is valid
   if(height < 0)
     { MGC_THROW_EXCEPTION("Vantage map height must be greater than: 0px, requested height: 
       " << height << "px!" );
   }
   // Delete any of the previous memory if any were allocated
   MGC_SAFE_DELETE_ARRAY(_vantage);
   MGC_SAFE_DELETE_ARRAY(_argb);
   // Allocate new memory
   _vantage = new uint32[width * height];
   _argb = new uint32[width * height];
   // Clear the memory 0 for the vantage, and to white for the argb
   memset(_vantage, 0, width * height * sizeof(uint32));
memset(_argb, 255, width * height * sizeof(uint32));

// Record the width and height of the map
_width = width;
_height = height;
}*/

/*void vantage_map::to_argb(uint32 * argb_buffer)
{
  // Get a pointer to the first value in the vantage list
  uint32 * cur = _vantage;
  uint32 * end = _vantage + (_width * _height);

  while(cur < end)
  {
    if(*cur == 0)
    {
      (*argb_buffer++) = 0xFF000000;
    }
    else
    {
      uint32 alpha = 255 - (uint32)(((float)(*cur) / (float)(_max_vantage))*(255));
      (*argb_buffer++) = (alpha << 24) | 0x000000CC;
    }
    ++cur;
  }*/

/*const uint32 * vantage_map::get_cached_argb(void) const
{
  return (const uint32*)_argb;
}*/
1 /*
2 Vision.cpp
3 Author: Chris Madden and Mark Baulis
4 Purpose: Object Detecting, Identifying and Locating
5 (see Vision.cpp for more detailed explanation)
6 */
7
8 #ifndef _INC_VISIONINIT
9 #define _INC_VISIONINIT
10 using namespace std;
11 #include <cv.h>
12 #include <cxcore.h>
13 #include <highgui.h>
14
15 #define numLidarBeams 529 // Total LIDAR beams per scan
16
17 // Arguments
18 const char * use="<image> <red threshold> <blue and green threshold> <blue green max diff>\n\n";
19
20 struct UGV_info {
21     int UGV_ID; // UGV vehicle number
22     float UGV_x; // UGV map position x coord (metres) from lower-left hand map corner
23     float UGV_y; // UGV map position y coord (metres) from lower-left hand map corner
24     float UGV_heading; // UGV heading relative to North (degrees) (this is also yaw)
25     float UGV_roll; // UGV roll angle (degrees) or slope UGV is climbing/descending
26     float UGV_pitch; // UGV pitch angle (degrees) or sideways slope UGV is on
27     float UGV_pan; // pan-tilt vertical angle (degrees), zero is horizontal
28     float UGV_tilt; // pan-tilt horizontal angle (degrees), zero is directly ahead
29     int timestamp; // format to be decided
30     int framenumber; // camera image number
31     }
32
33 struct ExtraOOIDatabaseInfo {
34     int consecDetectns; // Number of consecutive frames the object has been detected
35     bool confirmedObject; // 1 if the object is a confirmed OOI, 0 if not.
36     int lastImgNumSpotted; // The last image number when the OOI was spotted
37     }
38
39 struct object_info {
40     int ID;
41     int colour; // blue = 0, red = 1
42     CvRect objectRect; // Rectangle framing object
43     int x; // Actual map x coord of object detected
44     int y; // Actual map y coord of object detected
45     int z;
float panAngle; // +ve means left of straight ahead of camera, -ve means right
float tiltAngle; // +ve means up, -ve means down
int type; // static = 0, mobile = 1
float certainty;
bool active;
ExtraOOIDatabaseInfo OOIdata;

vector<object_info> position_ooi; // Database for OOI spotted

//IplImage* Threshold(IplImage* img); // Threshold function to find the useful pixels
char input_file[100]; // space given to name of created video stream
int img_num[3]; // Initialise integer array of UGV numbers

// Colour Thresholds
int x,y,red,green,blue,RGBtotal;
int minRedMean = 100;
int minBlueMean = 50;
float minMobileOoiDimRatio = 1.6f;
float maxMobileOoiDimRatio = 4.6f;
float minStaticOoiDimRatio = 0.8f;
float maxStaticOoiDimRatio = 1.5f;

// Colours assigned to screen additions
CvScalar contourColour = CV_RGB(0,255,255); // Cyan - Blob/Contour border colour
CvScalar boundRectColour = CV_RGB(0,0,255); // Red - Bounding box perimeter colour
CvScalar sampleRectColour = CV_RGB(255,255,255); // White - Sample bound box perimeter colour

int lidarCentreBeamNum = (numLidarBeams+1)/2; // Number of centre LIDAR beam
float lidarAngleIncrement = 0.36f; // The angle between each LIDAR beam
//int *lidarDistanceScans = 0;

// Note that this code assumes that we are getting back a portion of full resolution image.
// Therefore pixels per degree is assumed static, even if the width can vary.

// Camera Image Specs and parameters. These will be adjusted based on expected image size.
//int ActualImageWidth = 1624;
int pixels_per_degree = 20;
int MIN_CONTOUR = 100; // The outline of the objec much be a reasonable size
int MIN_HEIGHT = 20; // Objects must be at least 15 pixels high
int MIN_WIDTH = 15; // Objects must be at least 10 pixels wide

// Video Writer Parameters
int fps = 4; // framerate of created video stream
int is_colour = 1; // video writer creates colour stream
int bit_depth = 8; // 256 colour bit depth for processed images
int channel = 3; // colour images processed

// Images used in this code
IplImage* Threshold(IplImage* img, int thresholdOpColour); // Declare the function prototype, otherwise errors are produced because the compiler assumes a function will return an int.

// Continuity-checking variables
int latestDatabaseSize = 0; // The number of entries in the database before checking matches
int ooiRadius = 2; // Radius (m) between frames that OOI is expected to remain within
int maxFrameContinuity = 20; // The number of frames after which if an object is not spotted again, new objects are created

#endif // Use VisionInit.h
Vision.cpp

1 /*
2 Vision.cpp
3 Author: Chris Madden and Mark Baulis
4 Purpose: Object Detecting, Identifying and Locating
5 - get latest image from UGV USB over network
6 1. Object Detection
7 - colour thresholding
8 - morphology
9 2. Object Identification
10 - bounding rectangles
11 - second colour examination
12 - continuity with object database
13 3. Object Location
14 - find corresponding LIDAR beam distance
15 - convert location relative to UGV to map coordinates
16 */
17
18 #include <stdio.h>
19 #include <math.h>
20 #include <vector>
21 #include <iostream>
22 #include <iomanip>
23 #include <VisionInit.h>
24 #include <LoadLogData.h>
25 #include <fstream>
26 using namespace std;
27 #include <cv.h>
28 #include <cxcore.h>
29 #include <highgui.h>
30 #define PI 3.1415
31
32 // This is all the MPI stuff required
33 //#include <MessageOOIShared.h>
34 #include <UGV.h>
35
36 /* images are saved into this directory */
37 #ifdef _WIN32
38 #define IMAGE_SAVE_PATH "C:/Data/MAGIC/"
39 #else
40 #define IMAGE_SAVE_PATH "/home/mecheng/MAGIC_images/
41 #endif
42
43 #define IMG_FORWARD 10 // Number of frames to look forward to find possible image jumps
44
45 // Function to check if the spotted ooi exists in the ooi database yet
46 bool is_found(int x, int y, vector<object_info> pos_ooi_list, int size)
47 {
48     cout << "size(pos_ooi_list) == " << size << endl;
49     // Check if there are any OOIs in the list
50     if(size == 0)
51         return false;
// Check if the midpoint of the rectangle is in small region around the current ooi_positions
for(int i = 0; i < size; i++)
{
    // Calculate the differences between the two points
    float dx = x - pos.ooi_list[i].x;
    float dy = y - pos.ooi_list[i].y;

    // Calculate the radius
    float r = sqrt(dx*dx + dy*dy);

    // Print the calculated values
    // std::cout << "dx: " << dx << "dy: " << dy << "r: " << r << std::endl;

    // Check if it smaller than the threshold
    if(r < 0.5f)
        return true;
}

// return false to indicate nothing was found
return false;

int main(int argc, char* argv[])
{
    cout << "Initialising the ugv_list" << endl;
    /*vector<mpi::UGV> ugv_list;
    ugv_list.push_back(new mpi::UGV("172.16.0.11"));
    ugv_list.push_back(new mpi::UGV("172.16.0.21"));

    cout << "Finished initialising ugv_list" << endl;
    // Initialise an OOI Shared Bus
    mpi::UGV ugv("172.16.0.61");
    mpi::MessageOOIShared & msg_ooi_shared = ugv.getOOI();
    mpi::MessageOOI & msg_ooi = msg_ooi_shared;
    mpi::MessageLidar & msg_lidar = ugv.getLidar();
    mpi::MessageUGVPose & msg_pose = ugv.getPose();
    mpi::MessageOOIShared ooi_bus;
    ooi_bus.write(msg_ooi);
    */

    int ActualImageWidth;
    char window[100];

    double pixels_per_rad = pixels_per_degree*180.0/PI;
    float ooi_distance;

    // Initialise image numbers to 0
    for (int i=0; i<3; i++)
    {
        img_num[i] = 0;
        cout << "img_num[" << img_num[i] << "] == 0\n    }
// Initialization of Image processing components
img.roi = cvCreateImage(cvSize(320, 240), bit_depth, channel);  // Region of interest
image setup
IplConvKernel* element = cvCreateStructuringElementEx( 5,5,3,3, CV_SHAPE.ELLIPSE, 0); // set up kernel element (ellipse shape)
CvScalar colour = CV_RGB(0,255,255);  // assign a colour to the border outline of the blob

// Object Position & Angle
int frame_x; // x pixel of object centroid in frame
int frame_y; // y pixel of object base in frame
double object.pan;
double object.tilt;
int lidar.beam.number;

// Used for video capture
CvCapture* capture = cvCreateFileCapture( "MAGICtestFrames.avi" );
CvCapture* capture = cvCaptureFromCAM( 0 );

// Metadata file Creation
ofstream metadata;
metadata.open("cameradata.txt");

// Output Video Initialisation
//sprintf(input_file, "outvideo.avi");  // create a file and specify name
//CvVideoWriter* vidWriter = cvCreateVideoWriter( input_file, CV_FOURCC('D','I','V','X'),
//fps, cvSize(320, 240), isColour); // Setup created video stream parameters

// Set up windows to display output for all given UGVs
//cvNamedWindow( "UGV 1", 1 ); // create a window with specified name
cvNamedWindow("UGV 2",1); // create a window with specified name

//for(int ugv = 0; ugv < ugv_list.size(); ugv++)
{
  mpi::MessageOII & msg_ooi = ugv_list[ugv]->getOOI();
  msg_ooi.pos_ooi = vector<object_info>();
}

int num_iterations = 0;
while(num_iterations < 10)  // start the loop
{
  for ( int ugv=2; ugv<3; ugv++)
  {
    // Selection of image to load
    sprintf(input_file, IMAGE_SAVE_PATH "UGV%ld/image_%05d.jpg", ugv, img_num[ugv]);
    // %05 means 5 possible digit slots with 0s fill.
    stringstream next_image;
    next.image << IMAGE_SAVE_PATH << "UGV_" << ugv << "/image_" << std::setw(5) << std::setfill('0') << (img_num[ugv] + 1) << ".jpg";
    cout << "Next Image: " << next_image.str() << std::endl;
    ifstream next_img_file(next_image.str().c_str());
    if(next_img_file.is_open() == false)
```cpp
{ usleep(100000); continue; }

next_img_file.close();

img = cvLoadImage(input_file, 1); // 1 is for colour

// Ensure that we have a true image, and check that it has not skipped forward
bool valid_im = false;
int tmp = img_num[ugv];
if (img) {
    while ((tmp < img_num[ugv] + IMG_FORWARD) & (!img)) {
        tmp++;
        cout << "tmp: " << tmp << endl;
        sprintf(input_file, IMAGE_SAVE_PATH "UGV_%i/image_%05d.jpg", ugv, tmp);
        img = cvLoadImage(input_file, 1); // 1 is for colour
        if (img) {
            valid_im = true;
            img_num[ugv] = tmp;
        }
    }
    if (tmp >= (img_num[ugv] + IMG_FORWARD)) {
        cout << "UGV_" << ugv << " Image " << img_num[ugv] << " not found\n";
    }
} else {
    valid_im = true;
    usleep(500000);
    cout << "UGV_" << ugv << " Image " << img_num[ugv] << " processing\n";
    cout << img_num[ugv] << img_num[ugv] << endl;
    cout << "Loaded Image: " << input_file << endl;
}

if (valid_im) { // Process true images
    for (int thresholdOpColour = 1; thresholdOpColour<2; thresholdOpColour++) { // The colour the threshold operation is highlighting (blue = 0, red = 1)
        ActualImageWidth = img->width;
        //mpi::MessageLidar msg_lidar = ugv_list[ugv - 1]->getLidar();
        //mpi::MessageUGV Pose msg_ugv = ugv_list[ugv - 1]->getPose();

        // Create image result for threshold
        img_morph= cvCreateImage(cvSize(img->width, img->height), IPL_DEPTH_8U, 1);
        img_temp= cvCreateImage(cvSize(img->width, img->height), IPL_DEPTH_8U, 1);
        img_result_threshold = Threshold(img, thresholdOpColour);
        cvNamedWindow("Threshold", 1 ); // create a window
        with specified name
        cvShowImage("Threshold", img_result_threshold ); // place an image in
        the name-specified window
```

// Morphology section - joining close white blobs on black background
cvMorphologyEx(img_result_threshold, img_morph, img_temp, element,
    CV_MOP_CLOSE, 2);
cvNamedWindow( "Threshold after Morphology", 1 );       // create a window
    with specified name
cvShowImage( "Threshold after Morphology", img_morph );    // place an image
    in the name-specified window

// Storage and Arrays
CvMemStorage* storage = cvCreateMemStorage(0);    // a dynamic array on the
    computer for calculations
CvSeq* contour = 0;                                 // make a contour of
    initial size zero

// Contours and Bounding Boxes section
// Contours and Bounding Boxes section
cvFindContours( img_morph, storage, &contour, sizeof(CvContour),
    CV.RETR_CCOMP, CV.CHAIN_APPROX_SIMPLE, cvPoint(0,0) ) ;

vector<CvRect> boxes;
bool object = false;
CvRect largestbox = cvRect(0,0,0,0);
// Contour Identification
for( ; contour != 0; contour = contour->h_next )
{
    CvRect boundbox = cvBoundingRect(contour);      // Make each contour
    of structure? CvRect
    // cout << "boundbox (" << boundbox.x << "," << boundbox.y << "," <<
        boundbox.width << "," << boundbox.height << ") "
            << contour->total
        << " " << (boundbox.y+boundbox.height) << "\n";
    if ((contour->total>MIN_CONTOUR) && (boundbox.y+boundbox.height)>
    (img->height/3)) && (boundbox.y<(img->height/1.35)) && (boundbox.
        height>MIN_HEIGHT) && (boundbox.width>MIN_WIDTH))
    {
        // Only add a bounding box for the larger red objects, the rest is
        noise
        if (boundbox.height*boundbox.width > largestbox.height*largestbox.
            width){
            // Store the largest object in largestbox
            largestbox = boundbox;
            object = true;
        }
    object_info cur_object;

    cur.object.colour = -1;
    cur.object.x = -1;
    cur.object.y = -1;
    cur.object.panAngle = -1;
    cur.object.tiltAngle = -1;
    cur.object.type = -1;
    cur.object.certainty = -1;
    cur.DrawContours( img, contour, colour, colour, -1, 2, 8, cvPoint
        (0,0) );   // Draw contour
    boxes.push_back(boundbox);    // Add the contour
    rectangle to a vector
cvRectangle(img, cvPoint(boundbox.x, boundbox.y), cvPoint(boundbox.x + boundbox.width, boundbox.y + boundbox.height), cvScalar(0, 0, 255), 2, 8, 0);  // Place a rectangle around the contour referencing the top left hand corner
frame_x = boundbox.x + boundbox.width/2;  // image x coordinate for centre of object's rectangle base
frame_y = boundbox.y + boundbox.height/2;  // image y coordinate for centre of object's rectangle base
object_pan = -(img->width/2-frame_x)*ActualImageWidth/(img->width*pixels_per_rad);  // objects to the right give positive angles
// Note that the objectPan formula assumes that the image has not been scaled.
object_tilt = (img->height/2-frame_y)*ActualImageWidth/(img->width*pixels_per_rad);

// Calculate Object Location
lidar_beam_number = (int)(lidarCentreBeamNum - object.pan/50.0);  // The 0.5 and int casting round the beam float correctly up or down
out << "Lidar beam number: " << lidar_beam_number << endl;
out << "object_pan: " << object_pan << " lidarAngleIncrement:" << lidarAngleIncrement << endl;

ooi_distance = 10;
/*ooi_distance = msg_lidar.lidar[lidar_beam_number]/1000.0f;  // Gets the distance value from the latest LIDAR array (will be in shared memory)
out << "ooi_distance: " << ooi_distance << endl;
cur.object.x = msg.lidar.or_lidar.pos_ugv.easting + ooi_distance*cos(msg_lidar.or_lidar.yaw + object_pan) + 0.3f*cos(msg_ugv.or_ugv.yaw);
cur.object.y = msg.lidar.or_lidar.pos_ugv.northing + ooi_distance*sin(msg_lidar.or_lidar.yaw + object_pan) + 0.3f*sin(msg_ugv.or_ugv.yaw);
*/
out << "Object (" << frame_x << "," << frame_y << ") ";
out << "width=" << img->width << " Pan=" << object_pan << " Beam=" << lidar_beam_number;
out << "Dist=" << ooi_distance << "\n";

// Object Identification by colour
CvRect samplingRect;
samplingRect.x = boundbox.x + (int)(0.4*boundbox.width);
samplingRect.y = boundbox.y + (int)(0.4*boundbox.height);
samplingRect.width = (int)(0.2*boundbox.width);
samplingRect.height = (int)(0.2*boundbox.height);
//printf("x,y,width,height: %d, %d, %d, %d,",samplingRect.x,
samplingRect.y,samplingRect.width,samplingRect.height);

// Show the Sampling Region
//cvRectangle(img, cvPoint(samplingRect.x,samplingRect.y), cvPoint(samplingRect.x+samplingRect.width, samplingRect.y+samplingRect.height), cvScalar(255, 255, 255));
// Extract the sample section for analysis
cvSetImageROI(img, samplingRect);

// Find the mean and standard deviation of the sample section
CvScalar sampleMean;
CvScalar sampleStdDev;
cvAvgSdv(img, &sampleMean, &sampleStdDev); // Means and Std Devs are outputted as Scalar vectors
cvResetImageROI(img);

// Second Pass Colour Classification
pleMean.val[0], sampleMean.val[1])){
    cur_object.colour = 1; // Red object
} else{
    if (sampleMean.val[0] > minBlueMean && sampleMean.val[0] > max(sampleMean.val[1], sampleMean.val[2])){
        cur_object.colour = 0; // Blue object
    } else{ // Inform the programmer the second pass of the region failed the requirements.
        cur_object.colour = -1;
        cvRectangle(img, cvPoint(boundbox.x, boundbox.y), cvPoint(boundbox.x + boundbox.width, boundbox.y + boundbox.height), CV_RGB(255,255,0),2,8,0); //Yellow
        cout << "The region at " << object_pan << " deg did not fulfil the colour requirements of the second pass.\n";
        cout << "The means are: R = " << sampleMean.val[2] << " G = " << sampleMean.val[1] << " B = " << sampleMean.val[0] << "\n";
    }
}

// Object Identification by Dimensions
float dim_ratio = (float)boundbox.width/boundbox.height;
//int ID;
if ((0 <= dim_ratio) || (dim_ratio > 0.8)){
    cur_object.certainty = 0.0;
} else{
    if(0.8 <= dim_ratio){
        cur_object.ID = 0;
        cur_object.certainty = 0.9;
    }
    // Continue this down.
}
//cout << "Object.ID: " << cur_object.ID << "\n";

metadata << cur_object.ID << " ";
metadata << frame_x << " ";
metadata << frame_y << " ";
metadata << object.pan << " ";
metadata << object.tilt << endl;

// Region of Interest
/*object_info tmp_ooi;*/
H.2 gcs code | 437

tmp.ogg.x = cur.object.x;
tmp.ogg.y = cur.object.y;
tmp.ogg.z = 0;
tmp.ogg.active = true;
tmp.ogg.type = 0;
tmp.ogg.certainty = cur.object.certainty;
mpi::MessageOOI & msg.ogg = ugv.list[ugv-1]->getOOI();

//msg.ogg.pos.ogg.push_back(tmp.ogg);
int MAX_OOIS = 10;

// Check if the current ooi is already in the list.
if (is_found(cur.object.x, cur.object.y, msg.ogg.pos.ogg, msg.ogg.

msg.ogg.ooi_count) == false)
{
    cout << "The object found at (" << cur.object.x << ", " <<
cur.object.y << ") was added" << endl;
    //msg.ogg.pos.ogg.push_back(tmp.ogg);
    if(msg.ogg.ooi_count < MAX_OOIS)
    {
        msg.ogg.pos.ogg[msg.ogg.ooi_count++] = tmp.ogg;
    }
    else
    {
        // Shift all the objects up by one, dropping the oldest
        ones
        memcpy(msg.ogg.pos.ogg, msg.ogg.pos.ogg + 1, sizeof(
        position.ogg) * (msg.ogg.ooi_count - 1));
        msg.ogg.pos.ogg[msg.ogg.ooi_count - 1] = tmp.ogg;
    }
}

//cvSetImageROI(img, boundbox); // Make the video the size of the
bounding rectangle
//cvResize(img, img_roi, CV.INTER.LINEAR); // Scale the image up
or down to fit the 320x240 size
//cvWriteFrame(vidWriter, img_roi);
//cvRectangle(img_roi, cvPoint(0,0), cvPoint(boundbox.width,
boundbox.height),
//cvScalar(0, 0, 255),2,8,0); // added boundary box to fit the
unscaled ROI

}

sprintf(window, "UGV %d", ugv);
if (img->width > 900) { // if it is a big image
    IplImage *img_scaled = cvCreateImage(cvSize((img->width/2),(img->
          height/2)), bit_depth, channel); // Region of interest image
    cvResize(img, img_scaled, CV.INTER.LINEAR);
//cvNamedWindow( "Objects Of Interest", 1 );// create a window with specified name
//cvShowImage( "Objects Of Interest", img_scaled); // place an image in the name-specified window - use just "img" for unscaled
cvShowImage( window, img_scaled); // place an image in the name-specified window - use just "img" for unscaled
cvReleaseImage(&img_scaled);
} else {
    // Show full sized Objects
    //cvNamedWindow( "Objects Of Interest", 1 );// create a window with specified name
    //cvShowImage( "Objects Of Interest", img); // place an image in the name-specified window - use just "img" for unscaled
    cvShowImage( window, img); // place an image in the name-specified window - use just "img" for unscaled
}
    cvWaitKey(30);
    usleep(10000000);
    // Delete temporary images
    cvReleaseImage(&img_temp);
    cvReleaseImage(&img_morph);
    cvReleaseImage(&img_result_threshold);
}
    cvReleaseImage(&img);
    char c = cvWaitKey(1); // Exit if the escape key is pressed
    if(c == 27) break;
    img_num[ugv-1]++; // Increment the UGV image number
}
} // End loop for this UGV
} // End loop for all UGVs
// Used for video capture
//cvReleaseCapture( &capture );
//ugv_list.clear();
} // Threshold Function
IplImage* Threshold(IplImage* img, int thresholdOpColour){
    IplImage* img_result_threshold;
    img_result_threshold= cvCreateImage(cvSize(img->width, img->height), IPL_DEPTH_8U, 1); // ACTUAL: IplImage* cvCreateImage(CvSize size,int depth,int channels );
    for(x=0; x<img->width; x++) //for each row in the image
    {
        for(y=0; y<img->height; y++) // for each pixel in each row
        {
            red=((uchar*)(img->imageData + img->widthStep*y))[x*3+2]; // assign the value of every third pixel from and inc. the first pixel to
green=((uchar*)(img->imageData + img->widthStep*y))[(x*3)+1]; //imageData - A pointer to the aligned image data
blue=((uchar*)(img->imageData + img->widthStep*y))[x*3]; //widthStep - The size of an aligned image row, in bytes
uchar* temp_ptr=&((uchar*)(img_result_threshold->imageData + img_result_threshold->widthStep*y))[x]; //uchar = unsigned char

if (thresholdOpColour == 0){
    if (( (float)blue/green > 1.5) && ((float)blue/red > 1.15) ){
        temp_ptr[0]=255; //White for greater than threshold
    }else{
        temp_ptr[0]=0; //Black other
    }
}else{
    if (thresholdOpColour == 1){
        if (( (float)red/green > 2) && ((float)red/blue > 1.5) && ((float)blue/green > 1.2)){
            temp_ptr[0]=255; //White for greater than threshold
        }else{
            temp_ptr[0]=0; //Black other
        }
    }else{
        cout << "thresholdOpColour has not been assigned blue (0) or red (1)\n";
    }
}
return(img_result_threshold);
LoadLogData.cpp

/*
Vision.cpp
Author: Mark Baulis
Purpose: Gather the corresponding LIDAR data for the image into an array
*/

#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
using namespace std;
#define numLidarBeams 529 // Total LIDAR beams per scan

int* loadLogData(int desiredImageNumber)
{
    //Initialisation
    struct IMUobject{ // IMU data
        float yaw;
        float pitch;
        float roll;
    };

    struct GPSobject{ // GPS data
        int easting;
        float northing;
        float altitude;
        float speed;
    };

    struct LIDARobject{ // LIDAR
        int *Readings [numLidarBeams];
    };

    struct Dataobject{ // Combine the structures into one for organisation
        IMUobject IMU;
        GPSobject GPS;
        LIDARobject LIDAR;
    };

    Dataobject curData;

    float GPSeasting;
    float GPSnorthing;
    float GPSaltitude;
    float GPSspeed;
    float IMUyaw;
    float IMUpitch;
    float IMUroll;

    int *Readings = new int[numLidarBeams];
int imgNum = -1; // Initialise the integer for the found image number value
int bufferSize = 500;
string searchString("image");
long int currentPtrPosn; // Where the pointer is up to searching
int eofPtrPosn;
char * buffer; // to store search search block of code
buffer = new char [bufferSize]; // allocate memory

ifstream inFile; // Initiate the input file stream
inFile.open("C:\datafile.txt", ifstream::in);

inFile.seekg(0, ios::end); // Go to the end of the file
eofPtrPosn = inFile.tellg(); // Store the end of file address to an integer
currentPtrPosn = eofPtrPosn;
while (desiredImageNumber != imgNum){
    // go back one buffer size in the text file (the 11 accounts for the possible overlap
    // of "image ####" across the buffer ends except for the first read at the end of
    // file
    if(eofPtrPosn == currentPtrPosn){
        currentPtrPosn = eofPtrPosn - bufferSize;
    }else{
        currentPtrPosn = currentPtrPosn - bufferSize + 11;
    }
inFile.seekg(currentPtrPosn);
currentPtrPosn = inFile.tellg();
    // read data into buffer
inFile.read(buffer,bufferSize);

    //display buffer read
cout << "The contents of the buffer are: " << endl;
cout << "________________________________________" << endl;
cout << buffer.write(buffer,bufferSize);
cout << endl << "________________________________________" << endl;

    // Put contents of buffer into a string
string bufferString(buffer);

    // size_type imageStringAddr is the location of the first string matching searchString
    //IN the string bufferStringfound by .find
string::size_type imageStringAddr = bufferString.find(searchString, 0);
    //cout << "imageStringAddr: " << imageStringAddr << endl;
    //cout << "imageNumInTextPosn: " << imageNumInTextPosn << endl;
    //cout << "(currentPtrPosn + imageStringAddr): " << (currentPtrPosn + imageStringAddr) << endl;
    if (imageStringAddr != string::npos){ // If the location of the first string is not
    // at the end of the string
        // This makes a string out of the characters following the space after "image"
        string numberString = bufferString.substr(imageStringAddr + searchString.size() + 1); // stringstream provides an interface to manipulate strings as if they were input/ output streams.
        stringstream numberStream(numberString);
```cpp
if((numberStream >> imgNum).fail()){ // Convert stream to integer
    cout << "ERROR: Failed string to integer conversion";
}
cout << "imgNum integer: " << imgNum << endl;

// Find the number of digits in imgNum
int imgNumDigits = 0;
int step = 1;
while (step <= imgNum) {
    imgNumDigits++;
    step *= 10;
}
// The "+2" accounts for buffer overlap, not sure why - "\n"? perhaps?
inFile.seekg (currentPtrPosn + imageStringAddr + searchString.size() + imgNumDigits + 2);
}
delete[] buffer; // Do this outside the loop to avoid unnecessary thrashing of the heap
inFile >> GPSeasting;
cout << "GPSeasting: " << GPSeasting << endl;
inFile >> GPSnorthing;
cout << "GPSnorthing: " << GPSnorthing << endl;
inFile >> GPSaltitude;
cout << "GPSaltitude: " << GPSaltitude << endl;
inFile >> GPSspeed;
cout << "GPSspeed: " << GPSspeed << endl;
inFile >> IMUyaw;
cout << "IMUyaw: " << IMUyaw << endl;
inFile >> IMUpitch;
cout << "IMUpitch: " << IMUpitch << endl;
inFile >> IMUroll;
cout << "IMUroll: " << IMUroll << endl;
for(int i=0; i<numLidarBeams; i++){
    inFile >> Readings[i];
}
cout << "curData.curLidar.Readings[0]: " << Readings[0] << endl;
inFile.close();
return Readings;
```
CopyOverNetwork.cpp

1 /*
2 CopyOverNetwork.cpp
3 Author: Mark Baulis
4 Purpose: get latest image from UGV USB over network
5 */
6
7 #include <sys/types.h>
8 #include <dirent.h> // contains constructs that facilitate directory traversing
9 #include <errno.h>
10 #include <vector>
11 #include <string>
12 #include <iostream>
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <algorithm>
16
17 using namespace std;
18
19 int getdir (string strSource, vector<string> &files)
20 {
21     DIR *dp; // Directory stream type
22     struct dirent *dirp;
23
24     // Check the directory can be opened
25     if((dp = opendir(strSource.c_str())) == NULL) {
26         cout << "Error( " << errno << " ) opening " << strSource << endl;
27         return errno;
28     }
29
30     char cmdBuff[200];
31     sprintf(cmdBuff,"ls -t %s",strSource.c_str()); // Order by timestamp
32     system(cmdBuff);
33
34     // Read in the file/folder names to a vector
35     while ((dirp = readdir(dp)) != NULL) {
36         files.push_back(string(dirp->d_name));
37     }
38     closedir(dp);
39     return 0;
40 }
41
42 int main()
43 {
44     // Copy images over the network
45     int ugv = 2;
46     char strSource[200];
47     sprintf(strSource, "vision@10.42.43.21:/MAGIC/UGV%/d", ugv);
48     //string strSource = "vision@10.42.43.21:/MAGIC"; // Directory of images on the UGV
49     //string strSource = "/home/mecheng/TestDirectory/SourceDir";
50     string strDestn = "/home/mecheng/TestDirectory/DestnDir";
vector<string> files = vector<string>(); // Vector for storing file/folder names
getdir(strSource,files); // Function call to get the contents of the directory
for (unsigned int i = 0; i < files.size(); i++) { // Print the contents to console
    cout << files[i] << endl;
}

sort(files.begin(), files.end());
for (int i = 0; i < files.size(); i++)
    cout << files[i] << endl;

int latestImgNo = -1;
scanf(files[files.size()-1].c_str(), "%c%d", &latestImgNo);
cout << "int latestImgNo is: " << latestImgNo << endl;
string imageNo = "image_00041.jpg";

char cmdBuff[200];
//sprintf(cmdBuff,"scp %s/%s %s/%s", strSource, files[files.size()-1].c_str(), strDestn.c_str());
sprintf(cmdBuff,"scp %s/%s %s/%s", strSource, imageNo.c_str(), strDestn.c_str(), imageNo.c_str());
cout << "cmdBuff is: " << cmdBuff << endl;
system(cmdBuff);
return 0;
% CUBIC TRAJECTORY GENERATION

% Author: Sundar Komandurelaiyavalli
% Student ID: 1154422
% REVISION HISTORY

% * Original Release: 24th July 2010
% * Revision 1 - Using Nagy and Kelly's algorithm
% * Revision 2: 6th Sep 2010 - Generating a simple cubic path
% * Assumptions: * Initial x,y coordinates are (0,0).
% * Theta orientation is with respect to base station.
% * Revision 3: 28th Sep 2010 - Generalise path generation algorithm
% * Assumptions: * Initial x,y coordinates not (0,0).
% * Theta orientation is positive n anti-clockwise direction from positive axis.

Given 2 points, determine valid cubic trajectory

Get user to input start and end pose
clc;
clear;
X0 = input('Enter initial x coordinate')
Y0 = input('Enter initial y coordinate')
Tho = input('Enter initial orientation (In Degrees)') %In degrees
Xf = input('Enter final desired x coordinate')
Yf = input('Enter final desired y coordinate')
Thf = input('Enter final desired orientation (In degrees)')

% REVISION 3 - GENERALISED SUNDAR'S ALGORITHM

% Assume path is of the general form y = ax^3 + b^x2 + cx + d, where a,b,c
% and are some unknown constants.
% This algorithm creates four equations with four unknowns (a,b,c, and d),
% and then solves these equations simultaneously in order to determine the
% four unknowns. Once these unknowns are determined, the cubic polynomial
% between the two poses may be formed, which is the path that the UGV must
% follow.

Convert degrees to radians
Th0_rad = (Th0*pi)/180;
Thf_rad = (Thf*pi)/180;

% First equation
disp('The first equation is:')
disp('(a*(X^3))+(b*(X^2))+ (c*X) +d = Y0')

%Second equation
disp('Second equation is: ')
disp('(a*(Xf^3))+(b *(Xf^2))+(c *Xf) +d = Yf')

%Third equation
disp('The third equation is: ')
disp('(3*a*(X0^2))+(2*b*X0)+ c = Th0')

%Fourth equation
disp('The fourth equation is: ')
disp('(3*a*(Xf^2))+(2*b*Xf)+ c = Thf')

disp('SOLVING ALL FOUR EQUATIONS SIMULTANEOUSLY....')

[a,b,c,d] = solve('(a*(X0^3))+(b *(X0^2))+(c *X0)+d = Y0' , '(a*(Xf^3))+(b *(Xf^2))+(c *Xf) +d = Yf' , '(3*a*(X0^2))+(2*b*X0)+ c = Th0' , '(3*a*(Xf^2))+(2*b*Xf)+ c = Thf')
a = eval(a);
b = eval(b);
c = eval(c);
d = eval(d);

x = [X0:1:Xf]
F = (a.*(x.^3)) + (b.*(x.^2)) + c.*x + d;
plot(x,F)

%*****************THIS VERSION IS NOW OBSOLETE************

%% REVISION 2 - SUNDAR's ALGORITHM
% Assume cubic path is of the following form;
% f ( x )=a x ^3 +b x ^2 +c x+d
%sym a;
%sym b;
%First equation
%sym x;
%Substitue initial pose into equation:
% Yo = o = a(o) + b(o) + c(o) +d
% => d = o

%Second equation
% disp('Second equation is: ')
% disp('(Yf = (a*Xf^3)+(b*Xf^2)+(c*Xf)')
% %Third equation
% disp('tan(Th0) = 3*a*(X0^2) + 2*b*X0 + c')
% % Convert degrees to radians since MATLAB works with radians
% %Tho_rad = (Tho*pi)/180;
% %Thf_rad = (Thf*pi)/180;
% %c = tan(Tho_rad); %ONLY IF X0 = 0!!
% %[a,b] = solve('a*(X0^3)+(b*X0^2)+25-(c*X0)', '2*b*X0 + 3*a*(X0^2)=tan(Tho_rad)-c')
% %[a,b] = solve('a*(Xf^3)+b*(Xf^2)=Yf-(c*Xf)', '3*a*(Xf^2) + 2*b*Xf = tan(Thf_rad)-c', 'a', 'b')
% % disp('FOR SOME REASON MATLAB LEAVES a AND b AS SYMBOLS!!! I HAVE NOW SOLVED THIS ISSUE USING EVAL')
Hence, the polynomial is given by:

```
x = [0:1:Xf];
F = eval(a).*x.^3 + eval(b).*x.^2 + c.*x;
figure
plot(x,F)
```

% Convert omega from degrees to radians
% \( \text{Thf} = \frac{\text{thf} \cdot \pi}{180} \)
%\( \text{d} = \sqrt{Xf^2 + Yf^2} \)
%\( \text{DelTh} = \text{abs(Thf)} \)
\( s = d^*((\text{DelTh}^2/5)+1+(2/5)*\text{DelTh}) \)
%\( c = 0 \)
%\( K0 = 0 \)
%\( Kf = 0.5; \) % Assume Kf is somehow mysteriously known!
% Solve Theta and Kappa equations simultaneously to find a and b
% Write equations in matrix form \( Ma + Nb = C \)
\( M = [s s^2; (s^2)/2 (s^3)/3]; \)
\( N = [Kf; \text{Thf}]; \)
\( \text{SOL} = MN; \)
\( a = \text{SOL}(1,1); \)
\( b = \text{SOL}(2,1); \)
% Now find c
PathGenGUI.m

function varargout = PathGenGUI(varargin)

% PATHGENGUI M-file for PathGenGUI.fig
% PATHGENGUI, by itself, creates a new PATHGENGUI or raises the existing
% singleton*.

H = PATHGENGUI returns the handle to a new PATHGENGUI or the handle to
the existing singleton*.

PATHGENGUI(‘CALLBACK’, hObject, eventdata, handles,...) calls the local
function named CALLBACK in PATHGENGUI.M with the given input arguments.

PATHGENGUI(‘Property’,’Value’,...) creates a new PATHGENGUI or raises the
existing singleton*. Starting from the left, property value pairs are
applied to the GUI before PathGenGUI_OpeningFunction gets called. An
unrecognized property name or invalid value makes property application
stop. All inputs are passed to PathGenGUI_OpeningFcn via varargin.

*See GUI Options on GUIDE’s Tools menu. Choose "GUI allows only one
instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIDATA

% Copyright 2002–2003 The MathWorks, Inc.

% Edit the above text to modify the response to help PathGenGUI

% Last Modified by GUIDE v2.5 28–Sep–2010 17:02:06

% Begin initialization code -- DO NOT EDIT

% gui_Singleton = 1;

% gui_State = struct(‘gui_Name’, mfilename, ...
% ‘gui_Singleton’, gui_Singleton, ...
% ‘gui_OpeningFcn’, @PathGenGUI_OpeningFcn, ...
% ‘gui_OutputFcn’, @PathGenGUI_OutputFcn, ...
% ‘gui_LayoutFcn’, [], ...
% ‘gui_Callback’, []);

if nargin && ischar(varargin{1})

gui_State.gui_Callback = str2func(varargin{1});
end

if nargout

[varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else

gui_mainfcn(gui_State, varargin{:});
end

% End initialization code -- DO NOT EDIT

% ——— Executes just before PathGenGUI is made visible.

function PathGenGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved -- to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to PathGenGUI (see VARARGIN)

% Choose default command line output for PathGenGUI
handles.output = hObject;
set(handles.Xo, 'String', 0);
set(handles.Xf, 'String', 8);
set(handles.Yo, 'String', 0);
set(handles.Yf, 'String', 10);
set(handles.Tho, 'String', 10);
set(handles.Thf, 'String', 30);

% Update handles structure
guidata(hObject, handles);

Jx = [3:1:10];
Jy = (0.02144.*x.^3)−(0.30716.*x.^2)+(1.44077.*x)+2.86316;
plot(Jx, Jy)

% UIWAIT makes PathGenGUI wait for user response (see UIRUN)
% uisave(handles.figure1);
msgbox('Welcome to MAGIC2010 path generator!!','JOSH SAYS WELCOME!!')

% Outputs from this function are returned to the command line.
function varargout = PathGenGUI_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

% Get default command line output from handles structure
varargout{1} = handles.output;

function Yo_Callback(hObject, eventdata, handles)
% hObject    handle to Yo (see GCBO)
% eventdata  reserved -- to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Yo as text
% str2double(get(hObject,'String')) returns contents of Yo as a double
Yo = str2num(get(hObject,'String'));

if isnan(Yo)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

handles.Yo = Yo;
guidata(hObject, handles)

% Executes during object creation, after setting all properties.
function Yo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Yo (see GCBO)
450 | APPENDIX H : SOFTWARE CODE

function Th0_Callback(hObject, eventdata, handles)
% hObject handle to Th0 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

handles.Th0 = Th0;
guidata(hObject,handles)
%—— Executes during object creation, after setting all properties.
function Th0_CreateFcn(hObject, eventdata, handles)
% hObject handle to Th0 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function X0_Callback(hObject, eventdata, handles)
% hObject handle to X0 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Xo as text
% str2double(get(hObject,'String')) returns contents of Xo as a double
X0 = str2num(get(hObject,'String'));

if isnan(X0)
set(hObject, 'String', 0);
errordlg('Input must be a number', 'Error');
end

% Save X data
handles.Xo = Xo;
guidata(hObject, handles)

%—— Executes during object creation, after setting all properties.
function Xo_CreateFcn(hObject, eventdata, handles)
% hObject handle to Xo (see GCBO)
% eventdata reserved – to be defined in a future version of MATLAB
% handles empty – handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function Yf_Callback(hObject, eventdata, handles)
% hObject handle to Yf (see GCBO)
% eventdata reserved – to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of Yf as text
% str = str2double(get(hObject, 'String')) returns contents of Yf as a double
Yf = str2num(get(hObject, 'String'));

if isnan(Yf)
    set(hObject, 'String', 0);
    errordlg('Input must be a number', 'Error');
end
handles.Yf = Yf;
guidata(hObject, handles)
%—— Executes during object creation, after setting all properties.
function Yf_CreateFcn(hObject, eventdata, handles)
% hObject handle to Yf (see GCBO)
% eventdata reserved – to be defined in a future version of MATLAB
% handles empty – handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function Thf_Callback(hObject, eventdata, handles)
% hObject handle to Thf (see GCBO)
% eventdata reserved – to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of edThf as text
% str2double(get(hObject, 'String')) returns contents of edThf as a double

Thf = str2num(get(hObject, 'String'));

if isnan(Thf)
    set(hObject, 'String', 0);
    errordlg('Input must be a number', 'Error');
end

handles.Thf = Thf;
guidata(hObject, handles)

%—— Executes during object creation, after setting all properties.
function Thf_CreateFcn(hObject, eventdata, handles)

% hObject handle to edThf (see GCBO)
% eventdata reserved — to be defined in a future version of MATLAB
% handles empty — handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc & isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function Xf_Callback(hObject, eventdata, handles)

% hObject handle to Xf (see GCBO)
% eventdata reserved — to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of Xf as text
% str2double(get(hObject, 'String')) returns contents of Xf as a double

Xf = str2num(get(hObject, 'String'));

if isnan(Xf)
    set(hObject, 'String', 0);
    errordlg('Input must be a number', 'Error');
end

handles.Xf = Xf;
guidata(hObject, handles)

%—— Executes during object creation, after setting all properties.
function Xf_CreateFcn(hObject, eventdata, handles)

% hObject handle to Xf (see GCBO)
% eventdata reserved — to be defined in a future version of MATLAB
% handles empty — handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc & isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end
%—— Executes on button press in pbCreate.
function pbCreate_Callback(hObject, eventdata, handles)
% hObject handle to pbCreate (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
handles.Yo = Yo;
handles.Tho = Tho;
handles.edThf = Thf;
handles.Xo = Xo;
handles.Xf = Xf;
handles.Yf = Yf;
Th0_rad = ((handles.Th0)*pi)/180;
Thf_rad = ((handles.Thf)*pi)/180;
[a,b,c,d] = solve('a*(x0^3)+b*(x0^2)+c*x0+d = (handles.Y0)', 'a*(xf^3)+b*(xf^2)+c*xf+d = (handles.Yf)', '(3*a*(x0^2)+2*b*x0)+c = Th0_rad', '(3*a*(xf^2)+2*b*xf)+c = Thf_rad', 'a', 'b', 'c', 'd');
A = eval(a);
B = eval(b);
C = eval(c);
D = eval(d);
x = [(handles.Xo):1:(handles.Xf)];
F = (A.*x.^3) + (B.*x.^2) + C.*x + D;
plot(x,F)
%CreatePath(handles.Xo,handles.Xf,handles.Yo,handles.Yf,handles.Tho,handles.Thf)

%—— Executes on button press in pbClear.
function pbClear_Callback(hObject, eventdata, handles)
% hObject handle to pbClear (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
H.2.3.2 PATH TRACKING

MAGIC_Path_Tracker.fis

1 [System]
2 Name='MAGIC_Path_Tracker'
3 Type='mamdani'
4 Version=2.0
5 NumInputs=2
6 NumOutputs=2
7 NumRules=25
8 AndMethod='min'
9 OrMethod='max'
10 ImpMethod='min'
11 AggMethod='max'
12 DefuzzMethod='centroid'

13 [Input1]
14 Name='Delta_X'
15 Range=[-200 200]
16 NumMFs=5
17 MF1='Neglarge': 'trimf', [-100000 -120 -30]
18 MF2='Negsmall': 'trimf', [-90 -60 0]
19 MF3='Zero': 'trimf', [-30 0 30]
20 MF4='Possmall': 'trimf', [0 60 90]
21 MF5='Poslarge': 'trimf', [30 120 100000]

22 [Input2]
23 Name='Delta_Y'
24 Range=[-200 200]
25 NumMFs=5
26 MF1='Neglarge': 'trimf', [-100000 -120 -30]
27 MF2='Negsmall': 'trimf', [-90 -60 0]
28 MF3='Zero': 'trimf', [-30 0 30]
29 MF4='Possmall': 'trimf', [0 60 90]
30 MF5='Poslarge': 'trimf', [30 120 100000]

31 [Output1]
32 Name='VLR'
33 Range=[-2.1 2.1]
34 NumMFs=5
35 MF1='Leftlarge': 'trimf', [-2.1 -2 -0.6]
36 MF2='Leftsmall': 'trimf', [-0.6 -0.4 -0.2]
37 MF3='Zero': 'trimf', [-0.2 0 0.2]
38 MF4='Rightsmall': 'trimf', [0.2 0.4 0.6]
39 MF5='Rightlarge': 'trimf', [0.6 2 2.1]

40 [Output2]
41 Name='VFB'
42 Range=[-2.1 2.1]
43 NumMFs=5
44 MF1='Baclarge': 'trimf', [-2.1 -2 -0.6]
45 MF2='Bacsmall': 'trimf', [-0.6 -0.4 -0.2]
46 MF3='Zero': 'trimf', [-0.2 0 0.2]
51 MF4='For small': 'trimf', [0.2 0.4 0.6]
52 MF5='For large': 'trimf', [0.6 2 2.1]

54 [Rules]
55 1 1, 5 5 (1) : 1
56 1 2, 5 4 (1) : 1
57 1 3, 5 3 (1) : 1
58 1 4, 5 2 (1) : 1
59 1 5, 5 1 (1) : 1
60 2 1, 4 5 (1) : 1
61 2 2, 4 4 (1) : 1
62 2 3, 4 3 (1) : 1
63 2 4, 4 2 (1) : 1
64 2 5, 4 1 (1) : 1
65 3 1, 3 5 (1) : 1
66 3 2, 3 4 (1) : 1
67 3 3, 3 3 (1) : 1
68 3 4, 3 2 (1) : 1
69 3 5, 3 1 (1) : 1
70 4 1, 2 5 (1) : 1
71 4 2, 2 4 (1) : 1
72 4 3, 2 3 (1) : 1
73 4 4, 2 2 (1) : 1
74 4 5, 2 1 (1) : 1
75 5 1, 1 5 (1) : 1
76 5 2, 1 4 (1) : 1
77 5 3, 1 3 (1) : 1
78 5 4, 1 2 (1) : 1
79 5 5, 1 1 (1) : 1
This section of the Appendices contains the Firmware settings document.
Device Firmware Settings

Summary
This document outlines the Device Firmware and Settings to ensure standardisation across the system and across sites. This document outlines the latest configuration settings and should be consulted when configuring or adding any devices in the system. The system is broken into devices and device related software component information to outline any drivers or other software that needs to be installed to use each device.

Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>28/06/2010</td>
<td>0.1</td>
<td>Chris Madden</td>
<td>Document created</td>
</tr>
<tr>
<td>29/06/2010</td>
<td>0.2</td>
<td>Ben Cazzolato</td>
<td>Formatted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added EPOS2 configuration</td>
</tr>
<tr>
<td>02/06/2010</td>
<td>0.3</td>
<td>Peter Hardy, Anton Steketee</td>
<td>Added LIDAR, GPS and IMU configurations</td>
</tr>
<tr>
<td>02/06/2010</td>
<td>0.4</td>
<td>Mark Baulis</td>
<td>Added Lex Box and some GCS config.</td>
</tr>
<tr>
<td>18/07/2010</td>
<td>0.5</td>
<td>Anton Steketee</td>
<td>Improved GPS config</td>
</tr>
<tr>
<td>30/09/2010</td>
<td>0.6</td>
<td>Mark Baulis, Anton Steketee</td>
<td>Improved Lex Bos and GCS config</td>
</tr>
<tr>
<td>18/10/2010</td>
<td></td>
<td>Anton Steketee</td>
<td>Tidy up and add a few minor details. No major changes.</td>
</tr>
<tr>
<td>18/10/2010</td>
<td>0.7</td>
<td>Mark Baulis</td>
<td>Added guide on accessing the Ubuntu machine via Xming and Putty from a Windows machine (e.g. laptop)</td>
</tr>
</tbody>
</table>
**Camera system firmware and settings:**

Note that the Baumer camera SDK must be installed to obtain images as it ensures the network driver is GigE compliant. This can be performed on Ubuntu using the sh ./autoinstall.sh. On QNX the autoinstall is more problematic, and although the autoinstall is still run, the files need to be copied by hand. This should allow the Baumer Viewer to be run to find and adjust the cameras. Note also that the camera software also requires the installation of OpenCV 2.0 to be installed upon the target system to perform image compression and saving.

**Camera firmware and settings:**

The Baumer cameras are currently running with the firmware on the cameras. The camera driver software will initialise the camera capture system, and try to set the following parameters to ensure that images are acquired properly. If there are image acquisition problems, then the Baumer Viewer should be used to set these parameters.

- Camera IP Address - 10.42.43.x3 - where x is the UGV number
- Camera Exposure - Set low to minimise blur
- Camera Gain - 4.0 - Hopefully automatic control.
- Camera type - 1 - Set to full frame non-HQ mode
- PixelType - BGR () - set to structure that OpenCV expects
- CameraIOLine - 1 - possibility to control laser pointer.
LiDAR firmware and settings

LiDAR IP Address – 10.42.43.x2 - where x is the UGV number
Number of readings per scan - 529 (default, also max value)
Angular resolution - 0.36 deg (default)
Range – 65m (default, and max value)
Scanning Frequency – 25Hz (not changeable).
Data transfer protocol – Binary (can also use ASCII but binary is the default and the LIDAR will automatically revert back to binary on reset or power off)
Dust suppression – Active (can also be deactivated).
Object Size Detection - Objects of size below 70mm within a distance of 7m are ignored as dust. Can set this to automatic or down to 35mm at 6m, but making it more sensitive will increase measurement noise.
Object detection velocity – 1600mm/s (default value). I personally don’t see how this can be a value that we can change but it’s in the firmware, this may be for the RodSoft applications with the field detection stuff, will get more clarity with testing.

To change firmware settings need to install RODSoft and RodSoftPlus (will probably also need a german-english translation program, contact Peter Hardy for one or google). To run RodSoft you need an RS232 cable with a Y3 connection (i.e. connects to the LIDAR on the Y3 plug). Open RodSoftPlus, set interface Y2 for process data (Ethernet) and interface Y3 for parameterisation (RS232), check connectivity for both Y2 (Ethernet) and Y3 (serial), click on the configuration drop down menu to open RodSoft, Authorisation is AU, Password: ROD4LE. To change configurations click on the green configuration tab so that it goes yellow, click on the configuration drop down menu at the top of the screen, click on wizard, all parameters to change are in here.
GPS firmware and settings

The GPS units chosen are Novatel OEMV-2 mounted in Flexpack G2 enclosures and with a GPS-701-GG antenna. The antenna is connected using 50 ohm coaxial cables with male TNC connectors. Data communication is using RS232, but a crossover cable (as supplied by Novatel) must be used. The GPS units should all be upgraded to the 3.80 firmware and activated with the relevant auth code:

<table>
<thead>
<tr>
<th>Serial Number</th>
<th>PSN</th>
<th>Auth Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NHH10140006</td>
<td>BZZ10120079</td>
<td></td>
</tr>
<tr>
<td>NHH10140004</td>
<td>BZZ10120193</td>
<td></td>
</tr>
<tr>
<td>NHH10140008</td>
<td>BZZ10120207</td>
<td></td>
</tr>
<tr>
<td>NHH10140003</td>
<td>BZZ10120223</td>
<td></td>
</tr>
</tbody>
</table>

Unmatched SN/PSN

To enter these codes, flash the firmware and select 'Skip' when prompted for an auth code. When firmware flash is complete, connect to the unit (either via a serial terminal or Novatel CDU) and enter the command LOG VERSION. This output will include the PSN. Then you will need to execute the AUTH command, eg: "AUTH,"

Serial Numbers

NHH10140012
NHH10140009

PSN    Auth Code

BZZ10120200
BZZ10120089

These auth codes should allow operation at up to 50Hz, although this has not been verified.

Data communication with the GPS is through the RS232 COM1 port. Baud rate is 115200, with 8 data bits, 1 stop bit and no flow control. This Baud rate is required to be able to transmit enough data when operating at high update frequencies. Differential corrections are sent through COM2. This could be reduced to only require one serial port but has not been implemented for the sake of simplicity.

For differential GPS a base station at a fixed location is required to calculate corrections that are sent to the rover GPS units. These corrections significantly improve the accuracy of the final position determined by assuming that the errors at the base station position and the rovers’ position are similar. The output from the base station is designed so that it can be directly sent to the rover unit, without any processing. This is implemented with a high degree of robustness on board the GPS, such that dropped packets and loss of differential corrections do not pose a large problem. The RTCA (Radio Technical Commission for Aeronautics) system of sending corrections is used.

Rover Unit Set – up

The rover unit is set up to send its position, velocity and heading estimates to the Lex box using
Novatel’s proprietary ASCII logs. It receives corrections of the rtca form.

Assuming that the main data from the GPS will be received on COM1 and corrections sent on COM2 on the GPS unit, the following commands are needed to set up the GPS firmware:

ANTENNAMODEL 702GG [serial] 3 USER
BASEANTENNAMODEL 702GG NAE10130048 3 USER
INTERFACEMODE COM2 RTCA NONE OFF
LOG COM1 BESTUTMA ONTIME 0.1
LOG COM1 BESTVELA ONTIME 0.1
SAVECONFIG

Position information is returned from the GPS using the BESTUTM log.
- The position is in WGS84 datum and UTM position format.
- The information that is required from this log is Easting, Northing and height.
- These are returned in metres from a point that is arbitrary for the purposes of the project.
- The estimated accuracy of the Easting, Northing and height are also returned in the form of a standard deviation. This information is useful for the Kalman filter.

Velocity and heading information are returned using the BESTVEL log.
- The velocity is in metres per second.
- heading is measured in degrees clockwise from North (0-360).
Both are relatively inaccurate because they rely on differentiating the position of the vehicle (although I think the velocity user Doppler also – position may also, not sure) and so only update reliable when the vehicle is in motion. For this reason the velocity is never meaningfully negative and both forward and reverse motion of the vehicle return a positive velocity. Error in these measurements is approximated using (Novatel 2010, OEMV Family Firmware Version 3.700 Reference Manual Rev 7):

Velocity vector (speed and direction) calculations involve a difference operation between successive satellite measurement epochs and the error in comparison to the position calculation is reduced. As a result you can expect velocity accuracy approaching plus or minus 0.03 m/s, 0.07 m.p.h., or 0.06 knots assuming phase measurement capability and a relatively high measurement rate (that is, 1 Hz or better) by the GPS receiver.

Direction accuracy is derived as a function of the vehicle speed. A simple approach would be to assume a worst case 0.03 m/s cross-track velocity that would yield a direction error function something like:

\[ d \text{ (speed)} = \tan^{-1}(0.03/\text{speed}) \]

Both logs chosen are referred to as “BEST…” because they will return the most accurate estimate of position or velocity that the unit can calculate. For example, if the unit is receiving differential corrections then position calculated using these corrections is returned but if the network drops out then the position calculated using normal GPS methods is returned. This helps to simplify the GPS driver.

**Base station unit set up**
The base station is set up to send differentials corrections in the RTCA form
The best method to set-up the GPS base station is to collect data for single point averaging over a 12-24 hour period. Graeme Hooper (pers. comms. 2010) suggest the following method: (The commands are sent to the GPS using a terminal tool like HyperTerminal or Minicom).

If you have decided on a specific Ref location, then collecting data for 12 hrs..24hrs and submitting to Geoscience Australia (free via the internet), AusPOS will give you about 2cm accuracy tied into Aust Geodic network (used to monitor continental drift..)

LOG RANGEB ONTIME 5.0
LOG RAWEPHEMB ONNEW

Collect the data for 12..18 hrs.

Use NovAtel CONVERT tools (free on Web) to convert to RINEX format Submit RINEX file to AusPOS Your get an email report some hours later with exact coordinates..

This approach is not feasible for the MAGIC project, as the base station location is not fixed for long periods of time. Reasonable precision can be achieved by setting the base station position approximately. This means that any fixed error in the base station position will be transferred to the position of any UGV but the overall precision should be excellent. Novatel claims 2cm accuracy is achievable (Novatel 2010) under this situation but reliable accuracy was never better than 20cm, as measured by the standard deviation estimate attached to position estimates from the GPS. The accuracy of single point averaging over time is shown below:

The final set-up used for the base station is: (Novatel 2010b, p56)
POSAVE 1 (wait for one hour, then:)
INTERFACEMODE COM2 NONE RTCA OFF
LOG COM2 RTCAOBS ONTIME 1
LOG COM2 RTCAREF ONTIME 10
LOG COM2 RTC1 ONTIME 5
LOG COM2 RTCAEPHEM ONTIME 10 1

Corrections are then sent, in RTCA format, from COM2. All data returned from the base station GPS unit can be sent directly to all rover GPS units.
IMU firmware and settings

Each UGV has a Microstrain 3DM-GX3 IMU. Current firmware version is 1.1.27. Operational mode is Acceleration, Angular Rate and Orientation Matrix (0xC8). This mode was chosen as it returns the most relevant information while only requiring the use of one mode. The device is set up in continuous mode (0xC4), so that it returns data at a constant frequency of 100Hz. If required, it is still responsive to commands in continuous mode. When the IMU driver is started, the continuous command requesting this operational mode is set. It is not preset or saved in non-volatile memory on the IMU. Communication is via RS232 with a baud rate of 115200, 8 data bits, 1 stop bit, flow control disabled.

All devices require hard iron calibration after they have been installed in the UGV. See http://www.microstrain.com/updates/3DM-GX3_Software_CD_1.7.zip.

All communication with the IMU are in a hexadecimal form. The data received from contains a string of 67 bytes with 8 bits each. These are used in combination to form standard 16 bit floating point numbers. Form of the response is:

<table>
<thead>
<tr>
<th>Acceleration, Angular Rate &amp; Orientation Matrix (0xC8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function: The 3DM-GX3™ will output a data record containing the Acceleration and Angular Rate Vectors and the Orientation Matrix.</td>
</tr>
<tr>
<td>Command: Byte 1 0xC8</td>
</tr>
<tr>
<td>Response: Byte 1 0xC8</td>
</tr>
<tr>
<td>Bytes 2-5 Accelx (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 6-9 Accely (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 10-13 Accelz (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 14-17 AngRateX (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 18-21 AngRateY (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 22-25 AngRateZ (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 26-29 M1,1 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 30-33 M1,2 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 34-37 M1,3 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 38-41 M2,1 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 42-45 M2,2 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 46-49 M2,3 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 50-53 M3,1 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 54-57 M3,2 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 58-61 M3,3 (IEEE-754 Floating Point)</td>
</tr>
<tr>
<td>Bytes 62-65 Timer</td>
</tr>
<tr>
<td>Bytes 66-67 Checksum</td>
</tr>
</tbody>
</table>
The rotation matrix corresponding to a given set of Euler angles can be calculated using:

\[
M = \begin{bmatrix}
\cos(\psi)\cos(\theta) & \sin(\psi)\cos(\theta) & -\sin(\theta) \\
\cos(\psi)\sin(\theta)\sin(\phi) - \sin(\psi)\cos(\phi) & \sin(\psi)\sin(\theta)\sin(\phi) + \cos(\psi)\cos(\phi) & \cos(\theta)\sin(\phi) \\
\cos(\psi)\sin(\theta)\cos(\phi) + \sin(\psi)\sin(\phi) & \sin(\psi)\sin(\theta)\cos(\phi) - \cos(\psi)\sin(\phi) & \cos(\theta)\cos(\phi)
\end{bmatrix}
\]

where \( Pitch = \theta, Roll = \phi, Yaw = \psi \)

Further details of the implementation can be found in 3DM-GX3 Data Communications Protocol.pdf (http://www.microstrain.com/pdf/3DM-GX3%20Data%20Communications%20Protocol.pdf)

The IMU is actually mounted upside down in the UGV. This means that the forward acceleration measured is actually the negative of the desired value and the heading is actually \((-\theta + \pi)\).
EPOS firmware and settings

Each vehicle contains 8 MAXONMOTOR EPOS2 24/5 position controllers. The current EPOS2 firmware is Software Version: 0x2111 and Hardware Version: 0x6220.

The firmware parameters are given the table below. For more information regarding the parameters, datatypes and terminology please refer to the Maxon EPOS2 Firmware Specification document.

Parameter definitions have been classed according to use for the MAGIC project as follows:

<table>
<thead>
<tr>
<th>Employed</th>
<th>Should be employed</th>
<th>Unnecessary</th>
<th>Unknown</th>
</tr>
</thead>
</table>

The following notation has been used to denote static variables and dynamic variables.

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Data Type</th>
<th>Access Type</th>
<th>Subindex of… persistent parameters</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>Device Type</td>
<td>UNSIGNED32</td>
<td>RO</td>
<td></td>
<td></td>
<td>Value identifies standard and device (servo).</td>
</tr>
<tr>
<td>0x1001</td>
<td>Error Register</td>
<td>UNSIGNED8</td>
<td>RO</td>
<td></td>
<td>0x00</td>
<td>Needed for fault detection and handling</td>
</tr>
<tr>
<td>0x1003</td>
<td>Error History</td>
<td>ARRAY</td>
<td>RO</td>
<td></td>
<td></td>
<td>Needed for fault detection and handling</td>
</tr>
<tr>
<td>0x1005</td>
<td>COB-ID SYNC</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td>(0x00)</td>
<td>0x00000080</td>
<td></td>
</tr>
<tr>
<td>0x1008</td>
<td>Manufacturer Device Name</td>
<td>VISIBLE_STRING</td>
<td>CONST</td>
<td></td>
<td>EPOS2</td>
<td></td>
</tr>
<tr>
<td>0x100C</td>
<td>Guard Time</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td></td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>0x100D</td>
<td>Lifetime Factor</td>
<td>UNSIGNED8</td>
<td>RW</td>
<td></td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>0x1010</td>
<td>Store</td>
<td>ARRAY</td>
<td>RW</td>
<td></td>
<td></td>
<td>Should be used to save all device parameters in non-volatile memory. “SAVE” needs to be written to this object. One way of updating firmware settings across the net.</td>
</tr>
<tr>
<td>0x1011</td>
<td>Restore Default Parameters</td>
<td>ARRAY</td>
<td>RW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1012</td>
<td>COB-ID Time Stamp Object</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td>(0x00)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1013</td>
<td>High Resolution Time Stamp</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td></td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>0x1014</td>
<td>COB-ID EMCY</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td>(0x00)</td>
<td></td>
<td>Used to broadcast errors across the CAN.</td>
</tr>
<tr>
<td>0x1016</td>
<td>Consumer Heartbeat Time</td>
<td>ARRAY</td>
<td>RW</td>
<td></td>
<td>0x01, 0x02</td>
<td>Should be set to ensure that all EPOS units are active.</td>
</tr>
<tr>
<td>0x1017</td>
<td>Producer Heartbeat Time</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td></td>
<td>0x00</td>
<td>Should be set to ensure that all EPOS units are active.</td>
</tr>
<tr>
<td>0x1018</td>
<td>Identity Object</td>
<td>RECORD</td>
<td>RO</td>
<td></td>
<td></td>
<td>Can be used to check firmware and hardware up to date.</td>
</tr>
<tr>
<td>0x1020</td>
<td>Verify</td>
<td>ARRAY</td>
<td>RW</td>
<td></td>
<td>(0x01, 0x02)</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td>Type</td>
<td>Access</td>
<td>Pointer</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
<td>-------</td>
<td>--------</td>
<td>---------</td>
<td>----------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>0x1200</td>
<td>Server SDO Parameter</td>
<td>RECORD</td>
<td>RO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1400</td>
<td>Receive PDO 1 Parameter</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01, 0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1401</td>
<td>Receive PDO 2 Parameter</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01, 0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1402</td>
<td>Receive PDO 3 Parameter</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01, 0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1403</td>
<td>Receive PDO 4 Parameter</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01, 0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1600</td>
<td>Receive PDO 1 Mapping</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1601</td>
<td>Receive PDO 2 Mapping</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1602</td>
<td>Receive PDO 3 Mapping</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1603</td>
<td>Receive PDO 4 Mapping</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1800</td>
<td>Transmit PDO 1 Parameter</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1801</td>
<td>Transmit PDO 2 Parameter</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1802</td>
<td>Transmit PDO 3 Parameter</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1803</td>
<td>Transmit PDO 4 Parameter</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1A00</td>
<td>Transmit PDO 1 Mapping</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1A01</td>
<td>Transmit PDO 2 Mapping</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1A02</td>
<td>Transmit PDO 3 Mapping</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1A03</td>
<td>Transmit PDO 4 Mapping</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2000</td>
<td>Node ID</td>
<td>UNSIGNED8</td>
<td>RW</td>
<td>(0x00)</td>
<td>See table below for settings. Each EPOS unit has a unique CAN jumper setting.</td>
<td></td>
</tr>
<tr>
<td>0x2001</td>
<td>CAN Bitrate</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>(0x00)</td>
<td>0 Needed to allow comms between units. 0 (default) = 1Mbit/s</td>
<td></td>
</tr>
<tr>
<td>0x2002</td>
<td>RS232 Baudrate</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>(0x00)</td>
<td>5 Needed to allow comms between LEX box and master EPOS2. 5 (default) = 115.2 kbit/s</td>
<td></td>
</tr>
<tr>
<td>0x2003</td>
<td>Version</td>
<td>RECORD</td>
<td>RO</td>
<td></td>
<td>Current software version is 0x2111.</td>
<td></td>
</tr>
<tr>
<td>0x2004</td>
<td>Serial Number</td>
<td>UNSIGNED64</td>
<td>CONST</td>
<td></td>
<td>Could be used to track life cycle of EPOS unit.</td>
<td></td>
</tr>
<tr>
<td>0x2005</td>
<td>RS232 Frame Timeout</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>(0x00)</td>
<td>500 Timeout for RS232. Default is 500ms.</td>
<td></td>
</tr>
<tr>
<td>0x2006</td>
<td>USB Frame Timeout</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>(0x00)</td>
<td>500 Timeout for CAN. Default is 500ms.</td>
<td></td>
</tr>
<tr>
<td>0x2007</td>
<td>Miscellaneous Configuration</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>0x00</td>
<td>0x0000 Probably unnecessary. Most options can be implemented in code that calls EPOS. Default is 0x0000.</td>
<td></td>
</tr>
<tr>
<td>0x200A</td>
<td>CAN Bitrate Display</td>
<td>UNSIGNED16</td>
<td>RO</td>
<td></td>
<td>Unnecessary when hard coding bitrate.</td>
<td></td>
</tr>
<tr>
<td>0x200C</td>
<td>Custom Persistent Memory</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x04</td>
<td>4 unit32 for storage of user data. Could be used for ID.</td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td>Description</td>
<td>Type</td>
<td>Access</td>
<td>Value</td>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>0x2020</td>
<td>Incremental Encoder 1 Counter</td>
<td>UNSIGNED32</td>
<td>RO</td>
<td>0x00</td>
<td>It shows the actual encoder position in quadcounts. Needed for PTU to know orientation.</td>
<td></td>
</tr>
<tr>
<td>0x2021</td>
<td>Incremental Encoder Counter 1 at Index Pulse</td>
<td>UNSIGNED32</td>
<td>RO</td>
<td>0x00</td>
<td>Not needed with limit switch homing.</td>
<td></td>
</tr>
<tr>
<td>0x2022</td>
<td>Hall Sensor Pattern</td>
<td>UNSIGNED32</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2027</td>
<td>Current Actual Value Averaged</td>
<td>INTEGER16</td>
<td>RO</td>
<td>0x00</td>
<td>Could be used to detect faulty drive unit. Low current relative to other units indicates lack of drive (flat tyre, slipping in coupling), whereas the opposite indicates high friction (eg fouling around axle). Can be integrated to estimate power consumption to estimate battery life.</td>
<td></td>
</tr>
<tr>
<td>0x2028</td>
<td>Velocity Actual Value Averaged</td>
<td>INTEGER32</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2030</td>
<td>Current Mode Setting Value</td>
<td>INTEGER16</td>
<td>RW</td>
<td>0x00</td>
<td>Need not be employed explicitly. However defined in the EPOS code. Don’t know why.</td>
<td></td>
</tr>
<tr>
<td>0x2031</td>
<td>Current Demand Value</td>
<td>INTEGER16</td>
<td>RO</td>
<td>0x00</td>
<td>Not employed explicitly.</td>
<td></td>
</tr>
<tr>
<td>0x2062</td>
<td>Position Mode Setting Value</td>
<td>INTEGER32</td>
<td>RW</td>
<td>0x00</td>
<td>Only needed if direct position control for PTU. Not needed if interpolation mode.</td>
<td></td>
</tr>
<tr>
<td>0x2069</td>
<td>Auxiliary Velocity Sensor Actual Value</td>
<td>INTEGER32</td>
<td>RO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x206B</td>
<td>Velocity Mode Setting Value</td>
<td>INTEGER32</td>
<td>RW</td>
<td>0x00</td>
<td>Needed to command velocity.</td>
<td></td>
</tr>
<tr>
<td>0x206C</td>
<td>Auxiliary Velocity Actual Value</td>
<td>INTEGER32</td>
<td>RO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2070</td>
<td>Configuration of Digital Inputs</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x10</td>
<td>Needs definition for PTU. Should be used on all units for quickstop and drive enable.</td>
<td></td>
</tr>
<tr>
<td>0x2071</td>
<td>Digital Input Functionalities</td>
<td>RECORD</td>
<td>RW</td>
<td>0x02…0x04 0x01</td>
<td>Needs definition for PTU. Should be used on all units for quickstop and drive enable.</td>
<td></td>
</tr>
<tr>
<td>0x2074</td>
<td>Position Marker</td>
<td>RECORD</td>
<td>RO</td>
<td>0x02, 0x03 0x01, 0x04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2078</td>
<td>Digital Output Functionalities</td>
<td>RECORD</td>
<td>RW</td>
<td>0x02, 0x03 0x01</td>
<td>Currently no digital outputs are employed.</td>
<td></td>
</tr>
<tr>
<td>0x2079</td>
<td>Configuration of Digital Outputs</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x05</td>
<td>Needs to be configured if ESTOP wired rather than RS232.</td>
<td></td>
</tr>
<tr>
<td>0x207A</td>
<td>Position Compare</td>
<td>RECORD</td>
<td>RW</td>
<td>0x03…0x05 0x01, 0x02</td>
<td>Not really needed except for commissioning.</td>
<td></td>
</tr>
<tr>
<td>0x207B</td>
<td>Configuration of Analog Inputs</td>
<td>ARRAY</td>
<td>RW</td>
<td>0x01, 0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x207C</td>
<td>Analog Inputs</td>
<td>RECORD</td>
<td>RO</td>
<td>0x01, 0x02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x207D</td>
<td>Analog Input</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td>Type</td>
<td>Mask</td>
<td>Value</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------</td>
<td>--------------</td>
<td>------</td>
<td>--------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>0x207E</td>
<td>Analog Output 1</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>0x00</td>
<td>Can be used when no limit switches are present. Drive motor to stall to a register.</td>
<td></td>
</tr>
<tr>
<td>0x2080</td>
<td>Current Threshold for Homing Mode</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>0x00 0x00</td>
<td>Needs definition for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x2081</td>
<td>Home Position</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td>(0x00) 0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2082</td>
<td>Home Position Displacement</td>
<td>INTEGER32</td>
<td>RO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x20C1</td>
<td>Interpolation Data Record</td>
<td>STRUCT</td>
<td>WO</td>
<td>0x00</td>
<td>Needed for interpolation position mode. Needed to implement PTU Lissajous figure.</td>
<td></td>
</tr>
<tr>
<td>0x20C4</td>
<td>Interpolation Buffer</td>
<td>RECORD</td>
<td>RW</td>
<td>0x02, 0x03 0x01</td>
<td>Should be read to ensure interpolation data not inducing errors.</td>
<td></td>
</tr>
<tr>
<td>0x20F4</td>
<td>Following Error Actual Value</td>
<td>INTEGER16</td>
<td>RO</td>
<td>0x00</td>
<td>Not critical but could be used to locate poor tracking of PTU.</td>
<td></td>
</tr>
<tr>
<td>0x2100</td>
<td>Holding Brake Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td>No external brake (yet).</td>
<td></td>
</tr>
<tr>
<td>0x2101</td>
<td>Standstill Window Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03 30, 50, 1000</td>
<td>Is used to configure the way the motors are halted. Default settings probably ok. Critical parameter is 1000ms. If not at standstill (30RPM) then will fault.</td>
<td></td>
</tr>
<tr>
<td>0x2210</td>
<td>Sensor Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x04 500, 1, 0, 0</td>
<td>Used to define sensor type; encoder pulses/rev = 500, 1 indicates Incremental Encoder 1 with index (3-channel), Internal Absolute Position Offset = 0, Position Sensor Polarity = 0 (same direction as motor).</td>
<td></td>
</tr>
<tr>
<td>0x2211</td>
<td>SSI Encoder Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01, 0x02, 0x04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2212</td>
<td>Incremental Encoder 2 Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01 0x02, 0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2213</td>
<td>Sinus Incremental Encoder 2 Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2220</td>
<td>Controller Structure</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>0x00</td>
<td>No dual loop control available with EPOS2 24/5.</td>
<td></td>
</tr>
<tr>
<td>0x2230</td>
<td>Gear Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td>See little value in this except to add another constraint on speed.</td>
<td></td>
</tr>
<tr>
<td>0x2300</td>
<td>Digital Position Input</td>
<td>RECORD</td>
<td>RW</td>
<td>0x02…0x05</td>
<td>Don’t think needed. Only for external digital input commands.</td>
<td></td>
</tr>
<tr>
<td>0x2301</td>
<td>Analog Current Setpoint Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td>Analog inputs not employed in this config.</td>
<td></td>
</tr>
<tr>
<td>0x2302</td>
<td>Analog Velocity Setpoint Configuration</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2303</td>
<td>Analog Position</td>
<td>RECORD</td>
<td>RW</td>
<td>0x01…0x03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td>Type</td>
<td>RW/RO</td>
<td>Value</td>
<td>Remarks</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------</td>
<td>---------</td>
<td>-------</td>
<td>-------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>0x6007</td>
<td>Abort Connection Option Code</td>
<td>INTEGER16</td>
<td>RW</td>
<td>0x00</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defines what happens in the event of a comp failure. Currently set to quickstop (3). Other options are fault signal (1) or disable voltage (2).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6040</td>
<td>Controlword</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Must be employed to affect state changes like enable/disable, fault reset, etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6041</td>
<td>Statusword</td>
<td>UNSIGNED16</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Must be used to detect status of unit – like an acknowledgement. Can be used to check faults and other modes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x605B</td>
<td>Shutdown Option Code</td>
<td>INTEGER16</td>
<td>RW</td>
<td>0x00</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defines how motors shut down when shut down. 0 = shut off power drive stage.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x605C</td>
<td>Disable Operation Option Code</td>
<td>INTEGER16</td>
<td>RW</td>
<td>0x00</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>As above.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x605E</td>
<td>Fault Reaction Option Code</td>
<td>INTEGER16</td>
<td>RW</td>
<td>0x00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Currently set to quickstop in the event of a fault. Other options are fault signal only (-1), disable (0) or slow down ramp (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6060</td>
<td>Modes of Operation</td>
<td>INTEGER8</td>
<td>RW</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defines type of control mode, eg homing, position, velocity, etc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6061</td>
<td>Modes of Operation Display</td>
<td>INTEGER8</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Confirms state above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6062</td>
<td>Position Demand Value</td>
<td>INTEGER32</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Not explicitly used.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6064</td>
<td>Position Actual Value</td>
<td>INTEGER32</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Used to read the actual encoder position count. Referenced against home.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6065</td>
<td>Maximal Following Error</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td>0x00 0x00</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defines largest error allowed between actual and demand position. Default to 2000 counts.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6067</td>
<td>Position Window</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Probably not needed unless flags need returning to identify when a position bound is reached.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6068</td>
<td>Position Window Time</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>As above.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6069</td>
<td>Velocity Sensor Actual Value</td>
<td>INTEGER32</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Could be used to get instantaneous velocity. Better to use average velocity 0x2028</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x606B</td>
<td>Velocity Demand Value</td>
<td>INTEGER32</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Not used explicitly</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x606C</td>
<td>Velocity Actual Value</td>
<td>INTEGER32</td>
<td>RO</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Could be used to get instantaneous velocity. Better to use average velocity 0x2028</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x606D</td>
<td>Velocity Window</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Probably not needed unless flags need</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td>Type</td>
<td>Default Value</td>
<td>Value Range</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
<td>---------------</td>
<td>---------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>0x606E</td>
<td>Velocity Window Time</td>
<td>UNSIGNED16</td>
<td>0x00</td>
<td></td>
<td>As above.</td>
<td></td>
</tr>
<tr>
<td>0x6078</td>
<td>Current Actual Value</td>
<td>INTEGER16</td>
<td>0x00</td>
<td></td>
<td>Used for control loop. If interested in mean current better to use 0x2027.</td>
<td></td>
</tr>
<tr>
<td>0x607A</td>
<td>Target Position</td>
<td>INTEGER32</td>
<td>0x00</td>
<td></td>
<td>Needed to command position. Only for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x607C</td>
<td>Home Offset</td>
<td>INTEGER32</td>
<td>0x00 0x00</td>
<td></td>
<td>Needs definition for PTU. Needs to be defined for each vehicle so that after homing the pan and tilt move to [0.0] degrees.</td>
<td></td>
</tr>
<tr>
<td>0x607D</td>
<td>Software Position Limit</td>
<td>ARRAY</td>
<td>0x01, 0x02</td>
<td></td>
<td>Needed to define acceptable bounds orientation for pan and tilt units. Only for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x607F</td>
<td>Maximal Profile Velocity</td>
<td>UNSIGNED32</td>
<td>0x00</td>
<td></td>
<td>Needed limit max velocity during profiling. Only for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x6081</td>
<td>Profile Velocity</td>
<td>UNSIGNED32</td>
<td>(0x00) 0x00</td>
<td></td>
<td>Needed to limit to set velocity target while positioning. Only for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x6083</td>
<td>Profile Acceleration</td>
<td>UNSIGNED32</td>
<td>(0x00) 0x00</td>
<td></td>
<td>Needed to limit to set accel target while positioning. Only for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x6084</td>
<td>Profile Deceleration</td>
<td>UNSIGNED32</td>
<td>(0x00) 0x00</td>
<td></td>
<td>Needed to limit to set decel target while positioning. Only for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x6085</td>
<td>Quickstop Deceleration</td>
<td>UNSIGNED32</td>
<td>(0x00) 0x00</td>
<td></td>
<td>Needs to be defined to limit deceleration during “quickstop”. Only for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x6086</td>
<td>Motion Profile Type</td>
<td>INTEGER16</td>
<td>0x00 0x00</td>
<td></td>
<td>Needed to define profile type. Only for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x6089</td>
<td>Position Notation Index</td>
<td>INTEGER8</td>
<td>(0x00) 0x00</td>
<td>0x00</td>
<td>0x00 – no choice.</td>
<td></td>
</tr>
<tr>
<td>0x608A</td>
<td>Position Dimension Index</td>
<td>UNSIGNED8</td>
<td>(0x00) 0xAC</td>
<td></td>
<td>Steps are the dimension of position (no choice)</td>
<td></td>
</tr>
<tr>
<td>0x608B</td>
<td>Velocity Notation Index</td>
<td>INTEGER8</td>
<td>(0x00) 0x00</td>
<td>0x00</td>
<td>Used for units/sec</td>
<td></td>
</tr>
<tr>
<td>0x608C</td>
<td>Velocity Dimension Index</td>
<td>UNSIGNED8</td>
<td>(0x00) 0xAD</td>
<td></td>
<td>Steps/rev</td>
<td></td>
</tr>
<tr>
<td>0x608D</td>
<td>Acceleration Notation Index</td>
<td>INTEGER8</td>
<td>(0x00) 0x00</td>
<td>0x00</td>
<td>Rev/min</td>
<td></td>
</tr>
<tr>
<td>0x608E</td>
<td>Acceleration Dimension Index</td>
<td>UNSIGNED8</td>
<td>(0x00) 0xA4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6098</td>
<td>Homing Method</td>
<td>INTEGER8</td>
<td>0x00 0x00</td>
<td>17, 18, 23 or 27 depending on PTU config.</td>
<td>Needs definition for PTU motors.</td>
<td></td>
</tr>
<tr>
<td>0x6099</td>
<td>Homing Speeds</td>
<td>ARRAY</td>
<td>0x01, 0x02</td>
<td></td>
<td>Needs definition for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x609A</td>
<td>Homing Acceleration</td>
<td>UNSIGNED32</td>
<td>0x00 0x00</td>
<td></td>
<td>Needs definition for PTU.</td>
<td></td>
</tr>
<tr>
<td>0x60C0</td>
<td>Interpolation Sub Mode</td>
<td>INTEGER16</td>
<td></td>
<td></td>
<td>Needed for interpolation position mode. Needed</td>
<td></td>
</tr>
<tr>
<td>Selection</td>
<td>Value</td>
<td>Type</td>
<td>Access</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------</td>
<td>----------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x60C2</td>
<td>Interpolation Time Period</td>
<td>RECORD</td>
<td>RW</td>
<td>Needed for interpolation position mode. Needed to implement PTU Lissajous figure.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x60C4</td>
<td>Interpolation Data</td>
<td>RECORD</td>
<td>RO</td>
<td>Needed for interpolation position mode. Needed to implement PTU Lissajous figure.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configuration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x60C5</td>
<td>Max Acceleration</td>
<td>UNSIGNED32</td>
<td>RW</td>
<td>Needed to restrict acceleration to avoid slipping of wheels or couplings.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x60F6</td>
<td>Current Control Parameter Set</td>
<td>RECORD</td>
<td>RW</td>
<td>Used indirectly by position and velocity controllers. Defines PI gain for current loop.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x60F9</td>
<td>Velocity Control Parameter Set</td>
<td>RECORD</td>
<td>RW</td>
<td>Needed for velocity commanding. Defines PI controller gains, v and a feedforward gains.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x60FB</td>
<td>Position Control Parameter Set</td>
<td>RECORD</td>
<td>RW</td>
<td>Needed for position commanding. Defines PID controller gains, v and a feedforward gains.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x60FF</td>
<td>Target Velocity</td>
<td>INTEGER32</td>
<td>RW</td>
<td>Commanded velocity. Needed for drive wheels.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6402</td>
<td>Motor Type</td>
<td>UNSIGNED16</td>
<td>RW</td>
<td>10 Needed for all motors. Default (10) is EC motor sinus commutated with Hall sensors &amp; Incremental Encoder 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6410</td>
<td>Motor Data</td>
<td>RECORD</td>
<td>RW</td>
<td>Needed for all motors. Max average and short term current limits, num of pole pairs, max motor speed, thermal time constant.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6502</td>
<td>Supported Drive Modes</td>
<td>UNSIGNED32</td>
<td>CONST</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Type, Description, Size [Bits], Range

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Size [Bits]</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER8</td>
<td>Signed Integer</td>
<td>8</td>
<td>-128…127</td>
</tr>
<tr>
<td>INTEGER16</td>
<td>Signed Integer</td>
<td>16</td>
<td>-32 768…32 767</td>
</tr>
<tr>
<td>INTEGER32</td>
<td>Signed Integer</td>
<td>32</td>
<td>-2 147 483 648…2 147 483 647</td>
</tr>
<tr>
<td>UNSIGNED8</td>
<td>Unsigned Integer</td>
<td>8</td>
<td>0…255</td>
</tr>
<tr>
<td>UNSIGNED16</td>
<td>Unsigned Integer</td>
<td>16</td>
<td>0…65 535</td>
</tr>
<tr>
<td>UNSIGNED32</td>
<td>Unsigned Integer</td>
<td>32</td>
<td>0…4 294 967 265</td>
</tr>
<tr>
<td>UNSIGNED64</td>
<td>Unsigned Integer</td>
<td>64</td>
<td>0…18 446 744 073 709 551 615</td>
</tr>
<tr>
<td>VISIBLE_STRING</td>
<td>Array of (8-Bit)</td>
<td>n * 8</td>
<td>-</td>
</tr>
<tr>
<td>RECORD</td>
<td>Structure of other Types</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Attribute, Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>read and write access</td>
</tr>
<tr>
<td>RO</td>
<td>read only access</td>
</tr>
<tr>
<td>CONST</td>
<td>read only access, value is constant</td>
</tr>
</tbody>
</table>

### CAN Node Identification

The CAN jumper settings (CAN-ID node address) for each vehicle are given in the table below. Please note that LSB is on the left, MSB on the right. Jumper settings copied from EPOS Board Communication Schematic (04/10).

<table>
<thead>
<tr>
<th>EPOS2 Unit</th>
<th>CAN Jumper Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Front (M1)</td>
<td>01000000 = 2</td>
</tr>
<tr>
<td>Left Middle (M2)</td>
<td>00100000 = 4</td>
</tr>
<tr>
<td>Left Rear (M3)</td>
<td>11000000 = 5</td>
</tr>
<tr>
<td>Right Front (M4)</td>
<td>10100000 = 6</td>
</tr>
<tr>
<td>Right Middle (M5)</td>
<td>01100000 = 7</td>
</tr>
<tr>
<td>Right Rear (M6)</td>
<td>11100000 = 8</td>
</tr>
<tr>
<td>Tilt Motor</td>
<td></td>
</tr>
<tr>
<td>Pan Motor</td>
<td></td>
</tr>
</tbody>
</table>
Wireless Hardware configuration settings

The wireless unit is a Ubiquiti RouterStation Pro with SR-71A wireless modem. It is attached to 3 9dBi whip antennas on the UGVs and 3 7dBi antennas at the base station. Bigger wireless antennas may improve performance but the cost is very high and they will consume considerable space. Attempts to identify the best product for our needs in this area turned out to be very difficult and the products easily available in Australia seem quite limited.

The best wireless configurations to use would be 5 GHz 802.11n but this has been problematic with the dd-wrt firmware. The dd-wrt firmware was chosen as it has more features than the open-wrt firmware that is standard on the Router Station Pro. It was also hoped that dd-wrt would be more stable but it has turned out this is not the case. Updating to the latest version of dd-wrt might help. The disadvantage or 802.11n is that it is not supported by the wireless hardware on most computers.

Using 802.11b has shown to be very stable but data rates are low. 802.11a seems to be a good compromise.

Further investigation is required in this area to finalise the configuration.

Interesting information about Flashing the firmware on the RS Pro is here: http://www.dd-wrt.com/phpBB2/viewtopic.php?t=73349

IP Address’:

UGV1: 10.42.43.11
   Wireless Card 1: 10.42.43.10
   LiDAR: 10.42.43.12
   Camera: 10.42.43.13

UGV2: 10.42.43.21
   Wireless Card 2: 10.42.43.20
   LiDAR: 10.42.43.22
   Camera: 10.42.43.23

GCS: 10.42.43.1
GCS Ubiquiti: 10.42.43.100
GCS typically shares its standard wireless internet connection (to UofA), this requires the gateway on the UGVs to be set to 10.42.43.1.

Subnet Mask: 255.255.255.0
Lex Box configuration settings
OS: Ubuntu 10.04 (Server Edition)

A hack was done to get Ubuntu to see all 6 serial ports, instead of the standard four by http://www.linux.org/docs/ldp/howto/Serial-HOWTO-16.html#ss16.3

CONFIG_SERIAL_8250_RUNTIME_UARTS=6
CONFIG_SERIAL_8250_NR_UARTS=6

Are added to the /etc/default/grub file. Then run update-grub when the lines have been added.

Port Plan
The port labelled as COM1 on the box comes up as /dev/ttyS0 on the lex box. The same for all indices of the COM ports. (eg COM6 is /dev/ttyS5).

NB: COM3 is an RS485 port and could only be used with a RS485 to RS232 converter. COM1 on UGV 2 is unreliable. It doesn’t seem possible to use COM5 and 6 simultaneously, although this may be a programming bug. Hence the final port plan is up the air.

The following has been known to work reliably on UGV2:

COM2: IMU
COM4: GPS COM1
COM5: EPOS
USB-Serial Converter: GPS COM2
**Ground Control Station configurations:**
OS: Ubuntu 10.04 LTS (Adelaide),
Software installed: OpenCV 2.0, Qt (version 4),

Configuration of the touchscreen can be found in `/usr/lib/X11/xorg.conf.d`
This uses the evdev driver, which seems quite stable but a little short on features for touchscreens. Evtouch does not work in our case and causes X11 to be unstable and prone to preventing the computer starting up correctly. Don’t do it.

The system can be recovered by booting from the Ubuntu Live CD (may take up to 10 mins) and then mounting the hard drive and fixing whatever you did to break it. For example, if playing around with X11 settings then the following would let you edit the file:

Open a terminal (Applications -> Accessories -> Terminal)

```bash
$ mkdir /mnt/hd
$ sudo mount /dev/sdb1 /mnt/hd
$ cd /mnt/hd/etc/X11
$ sudo pico xorg.conf
```

**Port Plan**

COM1: GPS COM1/COM2
Ethernet: To Ubiquiti board
Screen connections are still be finalised.

**Using Xming to work on the GCS from a Windows computer**

A modified guide from
http://qiu.bioweb.hunter.cuny.edu/index.php?option=com_content&view=article&id=110
Since it is uncertain how long this useful website will be up, the modified contents will be displayed below.

These instructions are for Windows® users only!!

In the screenshots, I use Windows 7 64-bit as an example in order to cover all special cases, but the instructions should work on any version of Windows.

This guide will demonstrate how to connect to eniac.geo.hunter.cuny.edu (or in this case the MAGIC desktop of 10.42.43.25) using **PuTTY** for remote access and **Xming** for displaying Linux GUI programs on your local computer screen.

*Any programs run from Linux in this way are still running on the server. So if you save files, they will be saved to the server. This is the same as if you were using a computer in the lab room.*
1. Download PuTTY

Go to http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
Scroll down to Binaries and download PuTTY. This is the program you will use to connect remotely to the server at Hunter.

**Binaries**

_The latest release version (beta 0.60). This will get reporting it to me._

**For Windows on Intel x86**

PuTTY: **putty.exe**
PuTTYtel: puttytel.exe
PSCP: pscp.exe
PSFTP: psftp.exe
Plink: plink.exe
Pageant: pageant.exe
PuTTYgen: puttygen.exe

Save the file putty.exe to a convenient location. **Don't run it yet.**

2. Download Xming

Now go to http://www.straightrunning.com/XmingNotes/

Scroll down to Releases and download the two files shown in the screenshot below:

**Releases**

<table>
<thead>
<tr>
<th>Website Releases</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xming</td>
<td>7.5.0.23</td>
</tr>
<tr>
<td>Xming-portablePuTTY</td>
<td>7.5.0.24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Public Domain Releases</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xming-fonts</td>
<td>7.5.0.22</td>
</tr>
<tr>
<td>Xming</td>
<td>6.9.0.31</td>
</tr>
<tr>
<td>Xming-mesa</td>
<td></td>
</tr>
</tbody>
</table>

**First**, run the Xming-6-9-0-31-setup.exe file. Click "Run" if it asks you if you are sure you want to run the software.
Click "Next" through the installer to install with all defaults. However, uncheck the "Launch Xming" checkbox at the end.

![Completing the Xming Setup Wizard]

Now run the Xming-fonts-7-5-0-22-setup.exe file. Again, click "Run" if it asks if you are sure.

Click "Next" until it asks you to pick what to install. This part is optional, but it is highly recommended you install the Dejavu Fonts, as shown in the screenshot below:
Continue to click "Next" until the install is finished, then "Finish" to exit setup.

Now find Xming on your start menu, and run the program called XLaunch.

Click through the defaults and Finish to run Xming at the end. *Use XLaunch each time to run Xming.*

***Windows 7 or Vista users doing this for the first time may get a firewall warning here. If you are using Xming on a laptop which you carry around with you, you might want to check both checkboxes off. If you are using from a home computer, make sure at least the first checkbox is checked.***
Windows 7/Vista users may get a firewall warning

Xming will now appear as an icon in your system tray beside your clock, along with any other programs. On some systems, it may be hidden behind the system tray arrow.

Xming's icon in the system tray. You can right-click on it to exit when you are finished.

3. Configure PuTTY to use Xming

OK, now run putty.exe. Again, click "Run" if it asks if you are sure (you might also want to uncheck the box which says to ask you each time).

Before you fill in any information about the server, first click on the '+' sign next to "SSH" on the left-hand side, then on "X11" and change it to look like this:
PuTTY X configuration

Make sure the "Enable X11 Forwarding" checkbox is checked, and that the "X display location" box reads localhost:0

4. Set up connection, save settings, and connect.
Now click on "Session" at the very top of the left-hand side, and fill in eniac.geo.hunter.cuny.edu (in the case of the MAGIC project, 10.42.43.25) for the Host Name box.

Save these settings by filling in a name for them and clicking the "Save" button. I named my session "eniac" or "connect to MAGIC desktop". In the future, you can just pick your session from the choices and load it to connect.
PuTTY configuration for connecting to enaic. Save this configuration before connecting so that you won't have to do it again.

After saving, click "Open" to connect.

The first time you try to connect, you will get this message. Simply hit "Yes" to continue.

Auth key warning. This will only happen the first time you connect.
You will then be asked to "login as:" Here, put in your username. Hit "enter" and you will be asked for your password.

Login with YOUR username and password. The password prompt will appear after entering your username.

Hit enter again and you will be logged in.

5. *Test X forwarding is working. Rejoice.*
You can test that Xming is working by running this command:

```
  gedit &
```

You should get a new window for the Text Editor which you can use like a regular window on your computer.
The gedit window behaves just like a local window. Sometimes, you will get sluggish/stuttering performance due to a slow network. (for example, too many people doing this!)

This didn’t always work well. The most reliable method found to work was

**sudo nautilus &** - this starts up the Ubuntu file manager (or window in which you can see folders/directories). The “sudo” command gives you administrator privileges after typing in “cs1234”, which means you can delete files and folders (you won’t get a permission denied error). The ampersand (&) tells Ubuntu to run the process in the background (so the command prompt is not tied up with the process, and you can use the command prompt for other things). Sometimes “sudo nautilus &” starts a process (you see a [1] 5495 in the command line window or something similar – this denotes the process number on the computer), but the nautilus file manager does not open. It would usually start after typing in “sudo nautilus” again, without the ampersand. Sometimes the “gksudo nautilus” command would work better, although “gksudo” is the equivalent of “sudo” for starting graphical software, so is more appropriate for other programs.

**sudo gedit &** (sometimes followed by a Ctrl+C, the quit command, followed by just “gedit” again) brings up the gnome editor which will open any C++ file.

**firefox &** brings up the firefox web browser almost without fail

Unfortunately, a way to start up multiple terminals using Xming was never discovered. This does make working on the Ubuntu machine MUCH more difficult, and you always have to remember to add “&” on the end of commands that start up software to run them in the background. In general, the Xming and Putty combination was very useful, because it means you don’t have to tag team the Ubuntu machine (if your project software is Unix based), and you can both use the same Ubuntu computer (and not have to worry about setting up another one). However, network problems that caused dropouts could be a nightmare at times, and so made work quite inefficient sometimes. On occasions, Xming just wouldn’t connect, and so half hour sessions would be wasted trying to connect before you just give up and find something else to do until the Ubuntu machine is free.

### 6. Exit when finished. Exit, alright? Remember that. EXIT.

When you are finished with your session, enter the command
exit
to disconnect. Putty will close completely after this, but Xming will continue running in the background.

After following these instructions, all you have to do in the future is:
1. Launch XLaunch and click through until you launch Xming.
2. Run putty.exe, select your saved session, and click "Load" (or double click the session name to connect with those settings).
3. Enter your user name and password.

And you will be connected!

Advanced usage of Xming and putty

It is possible to save Xming's configuration so that you won't have to use XLaunch each time, but I won't cover it here.

You can learn more about both applications from their respective websites if you are so inclined. They both have many options and it may be useful to learn it for the future if you ever need to connect to other systems using UNIX while working from a Windows machine.

Setting up ssh keys

Reason for using this in the project: Normally when connecting to the UGV onboard computer from the MAGIC desktop, after typing in “ssh vision@10.42.43.21” into the command prompt, you may have to wait some time (up to 10 seconds), and after entering the password, have to wait another period (up to 15 seconds) before any command may be entered on the UGV command line. To avoid having the password confirmation, ssh keys may be set up, where two identical keys are stored on both computers and compared upon attempting to log in. This will not reduce the time before a password is asked for, but there will be no wait once the keys have been compared.

Modified from Paul Keck, 2001 < http://pkeck.myweb.uga.edu/ssh/>

1. First, install OpenSSH on two UNIX machines, GCS and UGV (It is installed on the GCS (Ubuntu 10.04) ). This works best using DSA keys and SSH2 by default as far as I can tell. All the other HOWTOs I've seen seem to deal with RSA keys and SSH1, and the instructions not surprisingly fail to work with SSH2.
2. On each machine, ssh into the other: i.e. type

   ssh vision@10.42.43.2 from the GCS, and enter the password: cs1234

   Then from the UGV (while you are connected via ssh) type:

   ssh mecheng@10.42.43.1 and enter the password: cs1234 (or use magicmark@... And the password: cs1234). This will create a .ssh dir in both of your home directories with the proper permissions (the directory is /home/username/.ssh where “username” is mecheng, magic, etc).
3. On your primary machine where you want your secret keys to live (the GCS) type

```
ssh-keygen -t dsa
```

This will prompt you for a secret passphrase (as usual: cs1234). If this is your primary identity key, make sure to use a good passphrase. If this works right you will get two files called `id_dsa` and `id_dsa.pub` in your `.ssh` dir.

4. Copy the `id_dsa.pub` file to the UGV's `.ssh` dir with the name `authorized_keys2`.

```
scp /home/mecheng/.ssh/id_dsa.pub vision@10.42.43.2:.ssh/authorized_keys2
```

Note that you don’t have to specify the `/home/vision` directory on the UGV, because when logged onto the UGV as user “vision”, the current directory is `/home/vision`.

5. Now the UGV is ready to accept your ssh key. How to tell it which keys to use? The `ssh-add` command will do it. From your “double ssh’ed” GCS, type

```
ssh-agent sh -c 'ssh-add < /dev/null && bash'
```

This will start the `ssh-agent`, add your default identity (prompting you for your passphrase), and spawn a bash shell. From this new shell you should be able to:

6. `ssh vision@10.42.43.21`

This should let you in without typing a password or passphrase. Hooray! You can ssh and scp all you want from this bash shell and not have to type any password or passphrase.