

# The Design of a Rudimentary Autonomous Ground Vehicle for Participation in the 2014 Autonomous Ground Vehicle Competition

Authored by Garrick Alexander Paskos, 2014

## Table of Contents

1. Letter to the Dean.....	4
2. Abstract.....	5
3. Acknowledgements.....	5
4. Introduction .....	6
4.1. Motivation.....	6
4.1.1. The 2014 “Autonomous Ground Vehicle Competition” .....	6
4.2.2. Project Management and Engineering Practice .....	8
4.3. Literature Review.....	8
4.3.1. Comparison of Robotics Middlewares.....	8
4.3.2. Comparison of Available SLAM Solutions .....	9
4.4. Aims and Objectives.....	10
5. Project Planning and Structure .....	10
5.1. Available Resources .....	10
5.2. Early Design Decisions.....	12
5.3. Task Assignment .....	13
5.3.1. Lane Obstacle Detection .....	14
5.3.2. Autonomous Navigation in a 2D Environment Featuring Obstacles .....	14
5.3.3 Goal Planning .....	15
6. Hardware Architecture .....	16
6.1. Vehicle Configuration.....	16
6.2. RTK GPS Base Station .....	20
7. Software Architecture.....	21
7.1. The “Robot Operating System” (ROS).....	21
7.1.2. Employed Existing Packages and Nodes .....	23
7.2. Abstract Software Subsystems .....	23
7.3. Development Environment and Workflow.....	24
7.3.1. Installation of Existing Packages .....	24
7.3.2. Configuring a Workspace .....	25
7.3.3. Constant Device Names .....	25
7.3.4. Computer Networking .....	25
7.3.5. Visualization Software .....	26
7.3.6. Simulators .....	26
8. Navigation Stack Configuration.....	26
8.1. Rudimental Configuration.....	26
8.1.1. Costmap Parameters.....	27
8.1.2. Path Planner Parameters .....	28
8.2. Visual “Lane” Obstacle Integration.....	28

8.3. Map Loading and Saving Functionality .....	30
8.4. Improved Odometry Through Sensor Fusion.....	30
9. Rudimental “Goal Planning” Subsystem .....	31
10. Results .....	32
10.1 Navigation Performance .....	32
10.1.2. Against Landmark Sparsity .....	32
10.1.3. With Lane Obstacle Layer .....	32
10.2. Piksi RTK GPS Performance .....	32
10.3. GPS Integration Performance .....	33
11. Conclusions .....	33
12. References .....	34
13. Appendixes.....	35

## 1. Letter to the Dean

Thursday, October 23, 2014

W/Prof John Dell

Faculty of Engineering Computing and Mathematics, The University of Western Australia, 35 Stirling Highway, CRAWLEY WA 6009

Dear Sir

I submit to you this final report entitled "The Design of a Rudimental Autonomous Ground Vehicle for Participation in the 2014 Autonomous Ground Vehicle Competition" in partial fulfilment of the requirement of the award of Bachelor of Engineering.

Yours Faithfully, Garrick Alexander Paskos.

## 2. Abstract

The “Autonomous Ground Vehicle Competition” (AGVC) is an annual Australian interuniversity competition intended to “stimulate robotics related research in Australian tertiary institutions”, facilitated by the Defence Science and Technology Organisation (DSTO). The 2014 AGVC poses the problem of autonomous navigation of a wheeled robot (holonomic or differential drive) within a 2-dimensional plane, constrained by visual boundaries, discrete obstacles and time.

Presented is a partial solution to the problem posed by the 2014 AGVC – employing the required sensors and robot base, accessed by a notebook computer running Ubuntu Linux and the Robot Operating System (ROS) robotics middleware.

The use of ROS provides a concrete interface to hardware and functional software components, facilitating the division of the problem into smaller, less dependent “project tasks”. Additionally, the collaborative nature of the ROS project provides the employment of existing software solutions to common robotics problems that would otherwise be outside the technical scope of the project.

This report will be mainly concerned with the navigation components of the project and secondarily, their integration with other components (such as visual obstacle detection and filtered odometry sources).

It has been found that the employment of existing Rao-Blackwellized particle filter based simultaneous localization and mapping (SLAM) software is sufficient to localize the robot within an environment consisting of sparse landmarks and slippery terrain. Additionally, the use of existing “costmap” based navigation software provides reliable and optimal navigation to arbitrary poses within the frame of reference provided by the SLAM components.

Finally, it has been found that the combination of an accurate magnetic compass and RTK GPS solution is sufficient to navigate the AGV to arbitrary GPS points, as required by the competition.

## 3. Acknowledgements

A significant part of the author’s work was the integration of other project team member’s tasks into the overarching “navigation stack” of the AGV. Because of this, extra care is given in delineating the author’s original contributions and that of other project team members (section 5.2, “Task Assignment” is devoted to briefly specifying other team member’s contributions).

Additionally, since this is the University’s 2<sup>nd</sup> entry in to the AGVC, at the commencement of the project, the entirety of the 2013 entry’s source code and configuration files were made available to all project team members. Despite not employing any of these in our current configuration, their perusal helped familiarity with the ROS system and configuration of hardware drivers.

Stuart Speidel, our mentor, who participated in the 2013 competition, was an invaluable source of information, who we met with on a weekly basis and was available for consultation frequently.

In the integration of the Piksi RTK GPS (section 9.3), the author collaborated with Ruvan Muthu-Krishna of the REV project in understanding its operation and limitations.

Finally, the author’s project supervisors, Professor Thomas Bräunl and Doctor Adrian Boeing must be acknowledged for providing technical assistance and motivation.

## 4. Introduction

This section provides motivation for the project and a brief literature review which helped finalize initial design decisions.

It should also be said here that all source code and configuration files developed throughout the project are available through git at <https://github.com/gappyp/UWAAGVC> with the username “guest” and password “sgj65oejaf”.

### 4.1. Motivation

#### 4.1.1. The 2014 “Autonomous Ground Vehicle Competition”

The “Autonomous Ground Vehicle Competition” (AGVC) is an annual Australian interuniversity competition intended to “stimulate robotics related research in Australian tertiary institutions” [1]. The competition is facilitated by the Defence Science and Technology Organisation (DSTO).

The 2<sup>nd</sup> annual AGVC is to take place during the 4 December 2014 to 8 December 2014 [2] and will feature the same competition rules as the 2013 competition in consequence of the DSTO deeming all entries into the 2013 competition as unsatisfactory. This will be UWA’s second entry into the competition.

The 2013 and 2014 AGVC rules (appx. 1, p. 1) can be briefly summarized as follows:

1. Pass an initial qualification test that ensures an entry adheres to a set technical specification, including:
  - Dimensions.
  - Propulsion (electric powered only).
  - Emergency shutdown routines (the existence of wired and wireless “stop” buttons).
  - Autonomous mode indication (via flashing lights).
2. Pass a further qualification test that requires the vehicle autonomously navigate a simplistic, known course (fig. 4.1.1.1) whilst demonstrating that it is able to:
  - Only travel within a speed range of 1.6km/h and 16km/h.
  - Detect and follow lanes (shown as the white lines in fig. 4.1.1.1).
  - Detect and avoid obstacles (barrels shown as dots in fig. 4.1.1.1).
  - Navigate to a given GPS waypoint whilst staying within lanes and avoiding obstacles.
3. If successful in qualifying, the vehicle will be required to navigate a more complicated course (fig. 4.1.1.2). This is called the “AutoNav” challenge.
  - This course also features blue and red coloured flags, where tolerable points will be deducted for the AGV not staying to the right or left of them, respectively.
4. Submit 15 page report and deliver a 20 minute oral presentation based on design decisions made. This has some weighting in the final ranking of competitors.

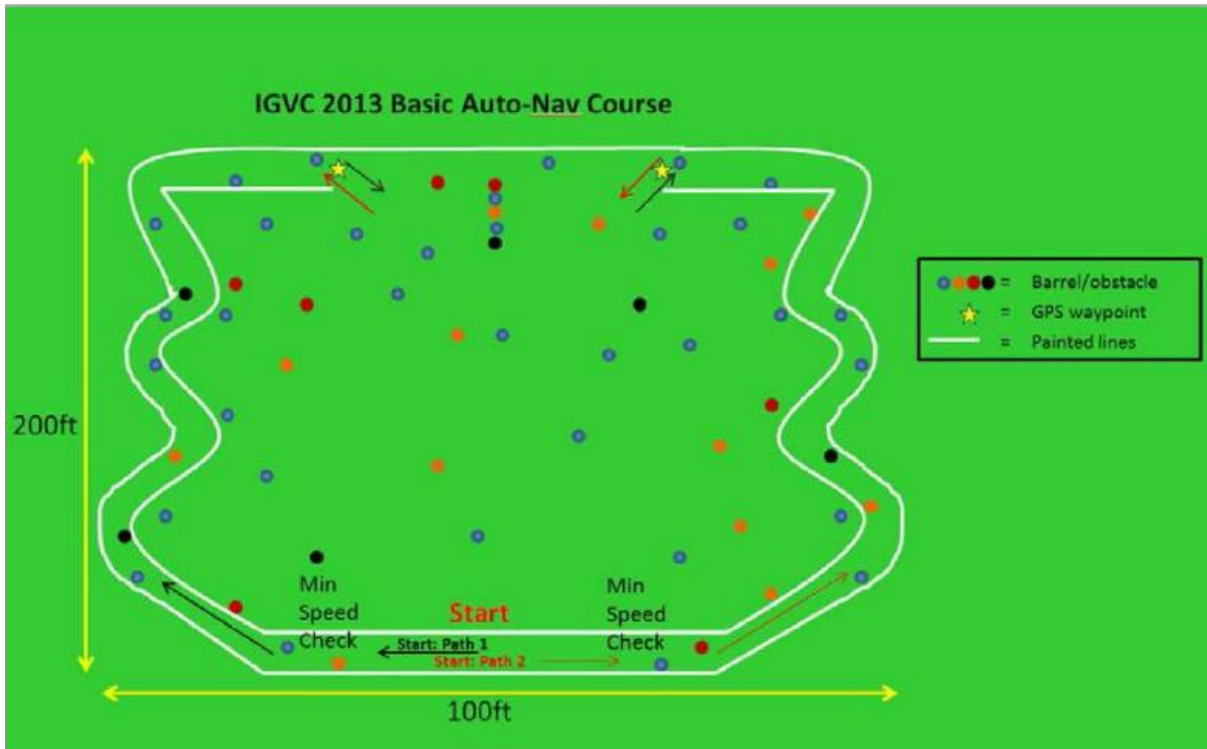


Fig. 4.1.1.1: Qualifying navigation course (from appx. 1).



Fig. 4.1.1.2: Sample AutoNav navigation course (from appx. 1).

#### 4.2.2. Project Management and Engineering Practice

In consistency with the faculty's new emphasis on "team-based" final year projects, particular attention has been paid to the employment of standard project management practices, such as time management through the use of Gantt charts and parallel task assignment.

Tasks have been assigned in the intention of producing a functioning and competitive entry into the AGVC, rather than 3 incompatible final year projects using the same hardware. Realistically, though, for the sake of the team members' final year reports some aspects of the AGVC were ignored, such as the "flag" feature, or greatly simplified (see "goal planning" in section 5.3.3).

Despite no formal agreement of individual roles other than responsibility for assigned tasks, the author has taken on the general role of project manager – responsible for the integration of other team member's efforts in to the final entry in the AGVC. The author is also the point of contact with the competition facilitators and produced and submitted the required "intent to compete" form (appx. 1, p. 21) and entry form (appx. 1, p. 22). Further from that, the "technical report" is required to be submitted on the 7 November 2014 [2] and the presentation also needs to be prepared.

#### 4.3. Literature Review

At the onset of the project, it was thought appropriate to review literature concerned with the evaluation and comparison of different robotics software environments available to utilize in the AGV. The reason for this is because the choice of framework could severely limit or complicate the development and functionality of the AGV. This is done in section 4.3.1.

Section 4.3.2 reviews literature related to existing SLAM algorithms. It is said in section 5.2 that it was necessary to employ some stable and accurate frame of reference in the AGV that GPS points would then be transformed in to. Wheel odometry or a fusion of it with an IMU was not sufficient as confirmed by Qing Zhou in his tasks. The use of a laser scanner, performing SLAM, with or without a source of traditional odometry was needed. It was even felt by the author that this was "hinted at" by the competition facilitators by the inclusion of the numerous and sporadic barrel obstacles throughout the course that would aid in the SLAM process. Hence a review of available SLAM techniques is warranted. Further, the complexity of implementing a SLAM algorithm restricts the review to only those already implemented in ROS.

##### 4.3.1. Comparison of Robotics Middlewares

[3] compares mainly several commercial and free robotics platforms but doesn't mention ROS whatsoever. Of particular interest is ARIA, which is developed by MobileRobots, Inc., the producers of the P3-AT base we're employing. The ROS driver we're using for the P3-AT base, p2os\_driver (see section 7.1.2) was developed "from scratch" by consulting the P3-AT hardware manual. There is in fact another ROS driver for the P3-AT base that is simply a "wrapper" of the ARIA software [4]. Several of the middlewares provide distributed computing capabilities, although most require the opening of socket interfaces (Player/Stage, Pyro, CARMEN). ADE only allows distribution through Java's Remote procedure calls. This contrasts to ROS's higher level system of simply entering network addresses of distributed computers and ROS taking care of all the opening of socket connections "behind the scenes" [5].

[6] analyzes most of the middleware in [3] with the addition of several others.

[7] specifically analyzes the MIRA middleware and compares it to ROS. Important aspects of MIRA outlined include its decentralized communication structure (compared to ROS's name and parameter server), its publisher/subscriber and RPC method of data transport (same method as ROS)



and its greater number of supported platforms (Linux and Windows as opposed to just Linux for ROS).

[8] only compares open source middleware, including Orca, Miro, OpenRDK, Marie, Smarisoft and Orocos.

This literature encourages the further use of ROS in the AGV, mainly because of its support for the majority of the hardware employed in the AGV (section 7.1.2). The only other contender would be ARIA because of its support for the P3-AT base, but other than that it provides little of the additional functionality that ROS does.

#### 4.3.2. Comparison of Available SLAM Solutions

The two most used and most supported ROS SLAM packages are hector\_slam [9] and gmapping [10]. Important attributes of both SLAM packages are shown in table 4.3.2.1.

ROS SLAM Package	Attributes
hector_slam	<ul style="list-style-type: none"> <li>• Can be used without odometry as well as on platforms that exhibit roll/pitch motion [9]</li> <li>• Does not provide explicit loop closing ability but is sufficiently accurate for many real world scenarios [9]</li> <li>• Combines a 2D SLAM system based on the integration of laser scans and an integrated 3D navigation system based on IMU [11]</li> </ul>
gmapping	<ul style="list-style-type: none"> <li>• A wrapper for OpenSlam's Gmapping [10]</li> <li>• Uses a highly efficient Rao-Blackwellized particle filter to learn grid maps from laser range data [12]</li> <li>• Rao-Blackwellized particle filters have only been recently introduced as effective means to solve the simultaneous localization and mapping (SLAM) problem [12].</li> <li>• Adaptive techniques are employed to reduce the number of particles in a Rao- Blackwellized particle filter for learning grid maps [12].</li> <li>• Takes raw laser range data and odometry [12]</li> <li>• Optimized for long-range laser scanners like SICK LMS or PLS scanner [12]</li> </ul>

**Table 4.3.2.1:** Important attributes of the hector\_slam and gmapping ROS SLAM packages.

[11] says “Rao-Blackwellized particle filters like Gmapping... work best in planar environments, rely on available, sufficiently accurate odometry and do not leverage the high update rate provided by modern LIDAR systems”.

[13] lists extended Kalman filters (EKFs), maximum likelihood techniques, sparse extended information filters (SEIFs), and Rao-Blackwellized particle filters (RBPFs) as the most popular estimation techniques for the SLAM problem. It concludes Rao-Blackwellized particle filters are the most optimizable in terms of quality and computational efficiency.

This literature review confirms the use of gmapping over hector\_slam due to its optimization for longrange laser scanners and planar environments (the situation present in the 2014 AGVC) and computational efficiency. The use of gmapping was also recommended by project supervisor Doctor Adrian Boeing.

#### 4.4. Aims and Objectives

The final aim or objective of the project, was to “produce the best possible entry into the 2014 AGVC with the resources available, whilst satisfying the requirement of the engineering final year project unit”.

In terms of the AGV design, communication with the previous year’s participants revealed that a robust and well tested AGV would be desirable.

In achieving this objective, the author was originally to be responsible for configuring the autonomous navigation components of the AGV including GPS integration. The author was also, later required to consider the goal visitation components of the AGV due to task reassignment. Specific tasks descriptions and how the team arrived at them are detailed in section 5.3.

### 5. Project Planning and Structure

#### 5.1. Available Resources

Three final year electrical engineering students were assigned to the AGVC final year project at the commencement of semester 1, 2014. These were Ross Green, Qing Zhou and Garrick Paskos (the author). The project was supervised by Doctor Adrian Boeing during the first semester, then later Professor Thomas Bräunl during the second semester.

The team met with project mentor, Stuart Speidel, weekly, who provided the team with an overview of the 2013 AGV and its various hardware and software components. Later, in semester 2, he also assisted with individual tasks. Development and team meetings primarily took place in the CIIPS “mobile robots” lab on the 2<sup>nd</sup> floor of the EE building.

The EE department provided most of the necessary hardware required for the AGVC. A list of relevant hardware, made available by the faculty is shown in table 5.1.1.

Category	Specifics
Laser Scanners	<ul style="list-style-type: none"><li>• 4 × SICK LMS111-10100<sup>1</sup><ul style="list-style-type: none"><li>○ 100 meter range, full 360 degree capability</li><li>○ Ethernet interface</li><li>○ Configurable via SOPAS Engineering Tool 3.0.1 software</li></ul></li><li>• 4 × Hokuyo URG-04LX-UG01<sup>2</sup><ul style="list-style-type: none"><li>○ Used to mimic detected lane point cloud in testing the navigation stack</li></ul></li></ul>
Robot Bases	<ul style="list-style-type: none"><li>• 4 × MobileRobots Pioneer 3-AT (P3-AT)<sup>3</sup><ul style="list-style-type: none"><li>○ Differential drive</li><li>○ Mounts for SICK LMS111-10100</li><li>○ Only bases 3 and 4 are operating perfectly</li><li>○ Configurable by ARCOS software</li></ul></li></ul>
IMUs	<ul style="list-style-type: none"><li>• 6 × Xsens Mti-28A53G35<ul style="list-style-type: none"><li>○ Outputs:<ul style="list-style-type: none"><li>▪ Linear acceleration</li><li>▪ Angular velocity</li><li>▪ Orientation (Euler or Quaternion)</li></ul></li></ul></li></ul>

<sup>1</sup> Specifications available from [http://www.sick.com/us/en-us/home/products/product\\_news/laser\\_measurement\\_systems/Pages/lms100.aspx](http://www.sick.com/us/en-us/home/products/product_news/laser_measurement_systems/Pages/lms100.aspx)

<sup>2</sup> Specifications available from [https://www.hokuyo-aut.jp/02sensor/07scanner/urg\\_04lx\\_ug01.html](https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html)

<sup>3</sup> Operations manual available from [http://www.inf.ufrgs.br/~prestes/Courses/Robotics/manual\\_pioneer.pdf](http://www.inf.ufrgs.br/~prestes/Courses/Robotics/manual_pioneer.pdf)

	<ul style="list-style-type: none"> <li>○ Configurable and assessable via MT Manager software <ul style="list-style-type: none"> <li>▪ Several of the IMUs were producing erroneous results even after calibration</li> <li>▪ This was mainly orientation drift</li> </ul> </li> <li>○ Calibrated via MagField Mapper software</li> </ul>
GPSs	<ul style="list-style-type: none"> <li>● 4 × QStarz GPS 818X modules <ul style="list-style-type: none"> <li>○ Single point positioning (SPP) solution</li> <li>○ Suffers from “GPS Drift”</li> <li>○ Approximate 3m radius accuracy before drift</li> </ul> </li> <li>● 2 × Piksi RTK GPS modules <ul style="list-style-type: none"> <li>○ Purported “centimeter level relative positioning accuracy” [14]</li> <li>○ Development product</li> <li>○ Acquired 2<sup>nd</sup> semester</li> <li>○ Each use a 915MHz 3DR wireless digital modem<sup>4</sup> to communicate information between the two modules <ul style="list-style-type: none"> <li>▪ Configurable through the Piksi console software or the official software (both through sending AT-like commands)</li> </ul> </li> <li>○ Only usable since firmware 0.11 released Oct 7 2014</li> </ul> </li> </ul>
Batteries	<ul style="list-style-type: none"> <li>● 12 × new Henda 12V 7.2AH/20HR compatible lead acid batteries <ul style="list-style-type: none"> <li>○ Made available 2<sup>nd</sup> semester</li> <li>○ Prior to that it was required to use faulty, old batteries <ul style="list-style-type: none"> <li>▪ These only held charge for approximately 30 minutes</li> </ul> </li> </ul> </li> </ul>
USB Cameras	<ul style="list-style-type: none"> <li>● 3 × Logitech Quickcam Orbit AF</li> <li>● 4 × LifeCam HD-3000<sup>5</sup> <ul style="list-style-type: none"> <li>○ Will replace the Orbit AF once suitable mounts are acquired</li> </ul> </li> </ul>
Networking	<ul style="list-style-type: none"> <li>● Dlink DIR-636L modem router <ul style="list-style-type: none"> <li>○ Used in place of Unifi in lab to provide unfiltered communication between AGV PC and control PC</li> </ul> </li> <li>● 4 × PicoStation2HP<sup>6</sup> <ul style="list-style-type: none"> <li>○ Outdated wireless b and g</li> <li>○ Max transmit power of 1W which may be illegal [15]</li> </ul> </li> </ul>
Other	<ul style="list-style-type: none"> <li>● 4 × Mo-Co-So MiniITX automotive PC <ul style="list-style-type: none"> <li>○ x86 Intel Core2Duo 8700, 4GB of RAM</li> <li>○ 254mm x 235mm x 80mm (L x W x H)</li> </ul> </li> <li>● Logitech LS11 stereo speakers</li> <li>● Logitech F710 wireless gamepad<sup>7</sup></li> <li>● Crucial 64GB SATA II 2.5” SSD <ul style="list-style-type: none"> <li>○ Replaced mechanical disk in AGV PC</li> </ul> </li> <li>● Electrical wiring and other electrical components</li> <li>● Various sized aluminum sheet steel and mounting tape</li> <li>● Trolley borrowed from mechanical engineering department for transporting robots to James Oval for testing</li> </ul>

**Table 5.1.1:** Relevant hardware made available by the faculty.

<sup>4</sup> Specifications available from <https://store.3drobotics.com/products/3dr-radio>

<sup>5</sup> Specifications available from <http://www.microsoft.com/hardware/en-au/p/lifecam-hd-3000>

<sup>6</sup> Data sheet available at [http://dl.ubnt.com/pico2hp\\_ds.pdf](http://dl.ubnt.com/pico2hp_ds.pdf)

<sup>7</sup> Specifications available at <http://gaming.logitech.com/en-au/product/f710-wireless-gamepad>

Hardware necessary and acquired by author were the following:

- 2 × Notebook computers:
  - Dell Inspiron N301Z on the AGV running its software
  - Dell Inspiron N4110 used for development and the control PC (over wifi or LTE)
- Components of the Piksi base-station (see section 6.2)
  - 275cm telescoping upright pole
  - CPT C120503 12V DC to 5V/3A DC USB power transformer
- Netgear Aircard 782s<sup>8</sup>
  - Replaced the PicoStation2HP as the AGV PC to control PC wireless communications link
  - 300n wifi
  - 4G LTE band 900 (B8) / 1800 (B3) / 2100 (B1) / 2600 (B7) w/2x2 MIMO
  - Separate battery power source
  - Immediately compatible with Ubuntu 14.04

## 5.2. Early Design Decisions

The most important overall project design decision was the selection of the underlying robotics middleware or environment, if decided to use any.

In line with the project goal, it was unfeasible to not use any middleware or environment that didn't at least provide a set interface to hardware and common robotics data structures, if not more. Otherwise unrealistic collaboration between all 3 project team members would have been necessary. Additionally, development ("from scratch") of several required software components, such as SLAM, route planning and hardware drivers would have been beyond the scope of the project.

Ultimately, after a review of other available robotics middlewares (section 4.3.1) and a group consensus, the use of ROS, running on 64-bit Ubuntu Linux was agreed on and this allowed us to rapidly produce an entry into the AGVC that would otherwise be outside the technical scope of the project. A basic introduction to ROS and its use in the AGV project is provided in section 7.1.

As mentioned in section 4.4, a robust system was desired. In terms of the author's tasks, this required the consideration of hardware and software failure, particularly GPS failure. There are two implications of GPS failure:

1. The GPS should not be used to provide a frame of reference and localize the AGV
2. True north should be obtained from a more reliable source other than the GPS

Problem 1 was addressed by employing a reliable frame of reference that is provided by a SLAM method that takes raw laser range data and traditional odometry [12]. GPS points are then transformed into this frame of reference and visited by the AGV.

The only available way of countering problem 2 was the employment of a magnetic compass to obtain the direction of true north. Fortunately the Xsens Mti-28A53G35 provided this capability through its magnetic compass and ability to compensate for magnetic declination. The effectiveness of this technique is evaluated in section 10.3.

---

<sup>8</sup> Specifications available from <http://www.netgear.com.au/service-providers/products/mobile/mobile-hotspots/782s.aspx#tab-techspecs>

### 5.3. Task Assignment

At the end of semester 1, specific, independent tasks were identified and assigned to individual team members. This was also required by the faculty in assessing the final year project unit.

The initial Gantt chart, agreed on by all team members and included in the project proposal is available in appx. 1, p. 26. The “tasks” section of this chart is reproduced here, in figure 5.3.1 for readability and to give an indication of the various tasks involved in the design and construction of the AGV.

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1	★	<b>Initialization</b>	<b>6 days</b>	<b>Mon 2/24/14</b>	<b>Mon 3/3/14</b>	
2	★	Familiarization with AGVC rules	5 days	Mon 2/24/14	Fri 2/28/14	
3	★	Familiarization with FYP	5 days	Mon 2/24/14	Fri 2/28/14	
4	★	Installation of Ubuntu & ROS on personal	5 days	Mon 2/24/14	Fri 2/28/14	
5	★	Familiarization with Ubuntu and Linux	5 days	Mon 2/24/14	Fri 2/28/14	
6	★	Acquisition of previous year's resources	5 days	Mon 2/24/14	Fri 2/28/14	
7	★	<b>Familiarization with ROS</b>	<b>10 days</b>	<b>Mon 3/3/14</b>	<b>Fri 3/14/14</b>	
8	★	ROS wiki basic tutorials	5 days	Mon 3/3/14	Fri 3/7/14	
9	★	ROS wiki intermediate to advanced tutorials	5 days	Mon 3/10/14	Fri 3/14/14	
10	★	Literature review for Adrian	0 days	Wed 3/5/14	Wed 3/5/14	
11	★	<b>ROS drivers</b>	<b>5 days</b>	<b>Mon 3/17/14</b>	<b>Fri 3/21/14</b>	
12	★	Installing and testing precompiled drivers	5 days	Mon 3/17/14	Fri 3/21/14	
13	★	Compiling and testing laser scanner drivers	5 days	Mon 3/17/14	Fri 3/21/14	
14	★	Initialization of ROS workspace and practice	5 days	Mon 3/24/14	Fri 3/28/14	
15	★	<b>Navigation stack</b>	<b>15 days</b>	<b>Mon 3/31/14</b>	<b>Fri 4/18/14</b>	
16	★	Installation and testing of required packages	5 days	Mon 3/31/14	Fri 4/4/14	
17	★	Installation, configuration and testing of	15 days	Mon 4/7/14	Fri 4/25/14	
18	★	Tutorials with stuart concerning line detection	10 days	Mon 4/28/14	Fri 5/9/14	
19	★	Establishment of individual tasks	5 days	Mon 5/12/14	Fri 5/16/14	
20	★	Exam break	15 days	Mon 6/2/14	Fri 6/20/14	
21	★	Rudimental competition ready entry to be	0 days	Mon 7/28/14	Mon 7/28/14	
22	★	Semester 2 starts	0 days	Mon 7/28/14	Mon 7/28/14	
23	★	<b>Garrick's individual tasks</b>	<b>65 days</b>	<b>Mon 4/7/14</b>	<b>Fri 7/4/14</b>	
24	★	Testing of ROS's distributed computing	20 days	Mon 4/7/14	Fri 5/2/14	
25	★	Over wifi	10 days	Mon 4/7/14	Fri 4/18/14	
26	★	Over 4g	10 days	Mon 4/21/14	Fri 5/2/14	
27	★	Tweaking of navigation stack	20 days	Mon 5/5/14	Fri 5/30/14	
28	★	Sending base to GPS point functionality	10 days	Mon 6/23/14	Fri 7/4/14	
29	★	Integration of new GPS module (may involve	10 days	Mon 6/23/14	Fri 7/4/14	
30	→	<b>Ross's individual tasks</b>	<b>55 days</b>	<b>Mon 5/12/14</b>	<b>Fri 7/25/14</b>	
31	★	Study of computer vision theory	15 days	Mon 5/12/14	Fri 5/30/14	
32	★	Practice with C++ and openCV library	15 days	Mon 5/12/14	Fri 5/30/14	
33	★	Experiment with line detection algorithms	10 days	Mon 5/19/14	Fri 5/30/14	
34	★	Lines-to-map functionality	25 days	Mon 6/23/14	Fri 7/25/14	
35	→	<b>Qing's individual tasks</b>	<b>55 days</b>	<b>Mon 5/12/14</b>	<b>Fri 7/25/14</b>	
36	★	Study of goal planning algorithms	15 days	Mon 5/12/14	Fri 5/30/14	
37	★	Practice with C++	15 days	Mon 5/12/14	Fri 5/30/14	
38	★	Rudimental goal planning functionality	10 days	Mon 6/23/14	Fri 7/4/14	
39	★	More advanced goal planning functionality	20 days	Mon 6/30/14	Fri 7/25/14	
40	→	<b>Combined tasks</b>	<b>15 days</b>	<b>Mon 7/7/14</b>	<b>Fri 7/25/14</b>	
41	★	Navigation <--> line detection (Garrick &	15 days	Mon 7/7/14	Fri 7/25/14	
42	★	Navigation <--> goal planning (Garrick &	15 days	Mon 7/7/14	Fri 7/25/14	
43	★	Emergency stop and flashing light functionality	5 days	Mon 7/28/14	Fri 8/1/14	
44	★	Final testing and improment and extra work if	80 days	Mon 8/4/14	Fri 11/21/14	
45	★	Project seminar	5 days	Mon 9/29/14	Fri 10/3/14	
46	★	Final Report Submission	0 days	Mon 10/27/14	Mon 10/27/14	
47	★	Preperation for competition report	14 days	Mon 11/10/14	Thu 11/27/14	
48	★	Expected competition date	2 days	Fri 11/28/14	Mon 12/1/14	

Figure 5.3.1: Task list in Gantt chart from appx. 1, p. 26.

Due to inexperience, the initial task assignment was most likely to prove difficult and unachievable; mainly due to the dependence of Qing Zhou's "goal planning" tasks (explained in section 5.3.3) on that of the remaining team members'. Additionally, the developmental Piksi GPS modules that were acquired by the school at the commencement of 2<sup>nd</sup> semester were proving difficult to use reliably and were holding back the author's progress in that respect. Consequently, a revised task assignment was agreed on which abandoned the complex goal planning algorithms and transferred the investigation of the use of an extended Kalman filter (EKF) to output a more accurate odometry, from the author to Qing Zhou.

In the next sub-sections, the most important tasks or components constituting the overall AGV project are elaborated on and given context in relation to the author's individual contributions. Mentioned also are the original and now simplified goal planning tasks.

### 5.3.1. Lane Obstacle Detection

As briefly summarized in section 4.2.1, a main requirement of the competition is that the AGV not cross over any white lanes that are spray painted on the course.

After acquiring an understanding of the autonomous navigation framework that ROS provides (section 8), it was evident that software which would output a point cloud representing solely the white lanes and their distance relative to the AGV would be required.

This task was assigned to Ross Green, and the author invites the reader to pursue his final year project report for a more detailed explanation of the techniques used to produce this point cloud.

In essence, though, images captured from a USB camera located relatively high on the AGV and angled down are processed in C++ with the openCV image processing libraries to extract the white lanes on the mainly green background. This processed image is then warped into a "bird's eye" perspective according to a warp matrix obtained from camera calibration software was also written by Ross Green. This would then provide an accurate x and y distance of each pixel (that constitutes the lane) from the camera. This was then output as the required PointCloud2 message type that was then further transformed into data that was compatible with the navigation stack (see figure 7.2.1).

### 5.3.2. Autonomous Navigation in a 2D Environment Featuring Obstacles

This task involved the production of software which would allow the AGV to reliably navigate to arbitrary GPS coordinates within the AGVC course whilst avoiding lane and barrel obstacles. The interface to this component was to simply supply the desired GPS coordinate (in the standard decimal NE latitude and longitude) and the component would output information relating to the status of this goal – such as it being achieved or in the process of being a achieved or if it is unattainable (only if it is "physically impossible" to achieve the goal, due to obstacles prohibiting it).

With this functional model of the system, it is obvious that there are several more subsystems of this component involved, having functions such as:

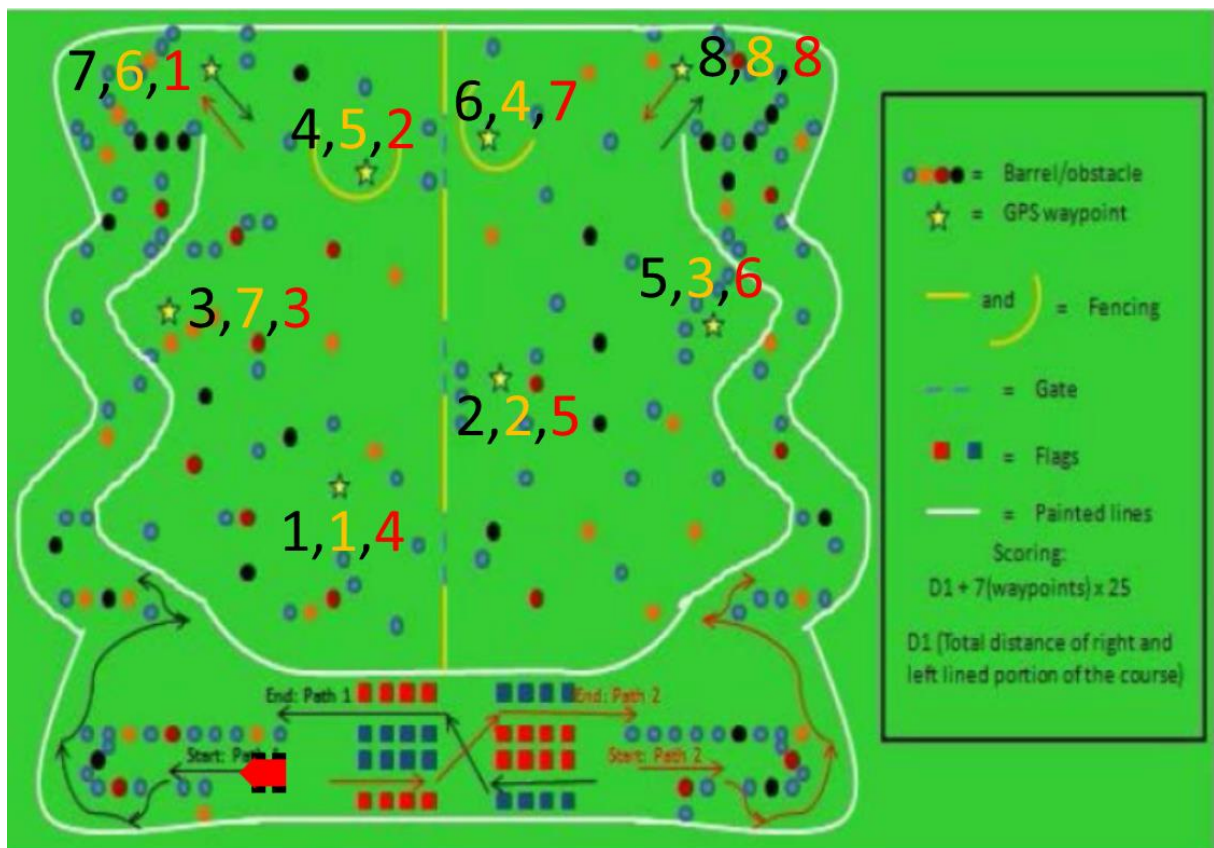
- The localization of the AGV to a "reliable" frame of reference (it was mentioned in section 5.2 that this was to be provided by SLAM software)
- The construction of an "obstacle map" (more correctly a "costmap" as explained in section 8)
- The planning of routes between points within this obstacle map
- The sending of velocity commands that constitute the above routes to the robot base
- The translation of absolute GPS points into the SLAM frame of reference

The number of tasks listed above may seem daunting and outside the scope of a final year project, but, through the use of ROS as the project’s robotics framework, implementation of much of the above functionality only required configuration of ROS’s pre-existing “navigation stack”.

### 5.3.3 Goal Planning

By “goal planning” it is meant that once the lane detection and navigation components have been implemented and the AGV is able to reliably navigate to any specified GPS waypoints within the course whilst avoiding barrel and lane obstacles, it is necessary that AGV destinations are updated according to what is most optimal in terms of distance travel and hence time. The idea is best explained by figure 5.3.3.1 below. In the figure, 3 visitation orderings are given for each waypoint, represented by the black, yellow and red numbers.

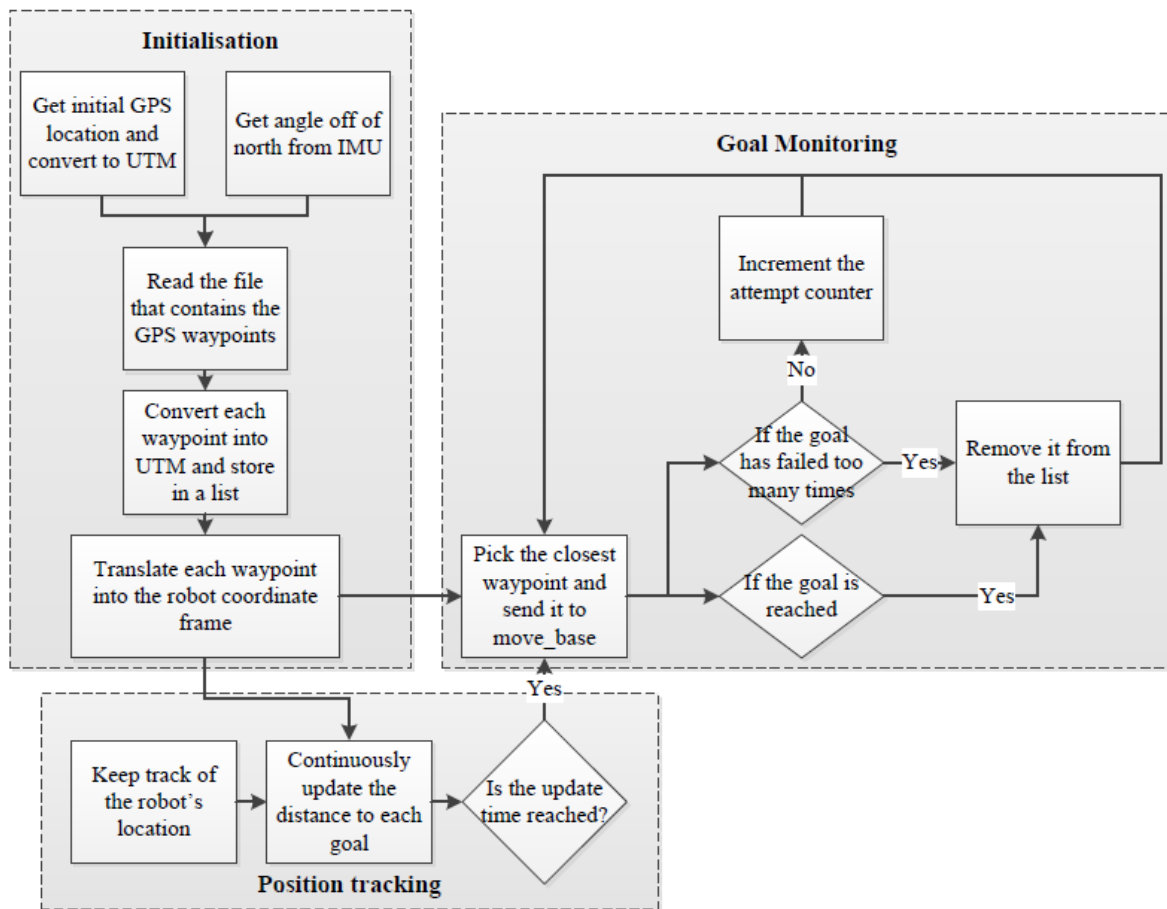
The black ordering is simply determined by the shortest Euclidean distance from the AGV at its starting position. The yellow ordering is determined by the shortest distance from the current waypoint to the next. The red ordering is the optimal one.



**Figure 5.3.3.1:** Example of different goal visitation algorithms.

As the AGV progresses through the course some waypoints may become more optimal to visit than others, and the goal planning software should recognize this and request the navigation stack replace its current goal with the new, optimal one. Optimal determination of the order of GPS waypoint visitation thus involves analysis of the current “obstacle map” produced by the navigation stack. The task is further complicated by the fact that this obstacle map, during the first run of the course will be incomplete and updating periodically with new information. Hence some form of “guess work” will be required and leads to the application of such algorithms that predict unknown lanes of a probable structure (the general form of the AGVC course is said to be like that in figure 5.3.3.1).

The 2013 AGVC goal planning algorithm is shown below in figure 5.3.3.2 to further aid in the understanding of the goal planning components.



**Figure 5.3.3.2:** The UWA 2013 AGVC entry goal planning algorithm. Taken from [16].

## 6. Hardware Architecture

A list of relevant, available hardware was given in section 5.2. In this section the current AGV hardware configuration is given and compared to the 2013 entry.

Additionally, the construction of a “GPS base-station” was necessary in order to elevate the base-station’s radio modem communicating observations to the rover’s module.

### 6.1. Vehicle Configuration

The current vehicle configuration is shown in fig. 6.1.1. The AGV has been substantially simplified from the 2013 entry shown in fig. 6.1.2.







**Fig. 6.1.1:** Current AGV hardware configuration.



**Fig. 6.1.2:** 2013 hardware configuration.

Probably the most important change from the 2013 configuration was the use of a notebook computer in place of the Mo-Co-So MiniITX automotive PC. This had the advantages of:

- Decoupling the control computer's power source from the base, that has the possibility of fluctuating or failing under high load conditions on the brushed motors.
- Greater energy efficiency
- Providing a terminal to simplify development and debugging

A further improvement from the previous year's configuration was the simplification and increased reliability of power distribution. For unknown reasons, the P3-AT's board auxiliary power terminals (fig. 6.1.3) were not utilized in the 2013 AGV, two of these are even controlled by the auxiliary switches on the base user control board, which we found useful to connect to the LMS111 to disable it when it was not required.

The PoE powered wireless access point of the 2013 AGV was replaced a Netgear AC782S LTE modem router that was also able to create its own wireless 300n network. This was also partly in anticipation of future requirements of teleportation of the AGV outside of standard wifi range. The discarding of the 2013 wireless access point also helped with the power routing simplification through discarding its PoE transformer.

Since the 2014 AGV is using SLAM to create its reliable frame of reference, as opposed the 2013's use of a fusion of wheel odometry and IMU [16], it was ideal that the angular distance scanned by the SICK LMS111-10100 was as large as possible. The LMS111 can be configured via its official software to ignore scanning a range of angular positions, and this was necessary to allow for the cables from the USB camera and rover Piksi to come down from their location above the LMS111. Since the cables are relatively thin, this ignored angle can be reduced to approximately 5 degrees, but it has been set higher because of requirements of the GENG5508 unit, which also uses the hardware.

It was necessary to isolate the IMU just as on the 2013 AGV due to inability of the MagField Mapper software to calibrate it otherwise.

In fig. 6.1.1 3 Logitech Quickcam Orbit AFs are mounted above the laser scanner and below the Piksi. There is an additional Logitech C270 webcam located at the front of the base pointing directly downward. This is an experimental configuration still being evaluated by Ross Green. The C270 may be required to alleviate potential "blind spot" problems when the AGV drives too close to a lane.

Finally, the QStarz GPS 818X was replaced with the Piksi RTK GPS solution, which requires two modules – one on the AGV (the rover) and the other ultimately to be placed at a known and accurate surveyed GPS point (the base-station).

For both the rover and the base-station, it is ideal to elevate both the GPS antennae and the radio modems. Embarrassingly, much time was wasted by testing with the base-station's modem essentially at ground level, resulting in system functioning inconsistently if not at all. An ad hoc elevating base station, which has resulted in reliable operation of the Piksi system (on firmware 0.11) over a 300m range is shown in the next section.

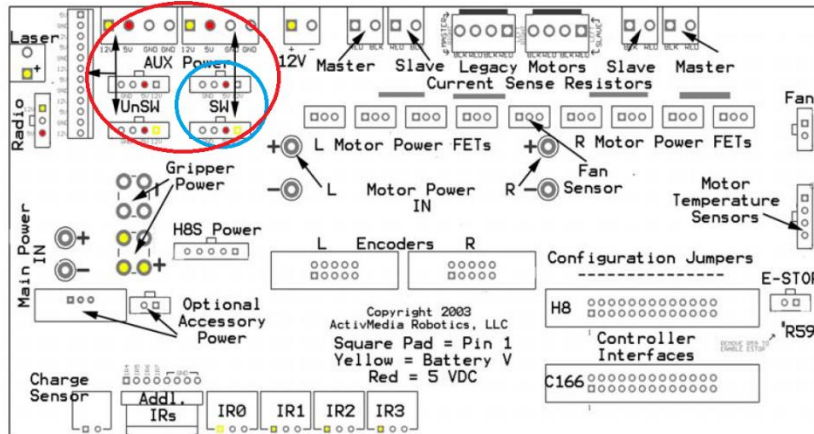


Fig. 6.1.3: Base auxiliary power terminals – which supply more reliable power.

## 6.2. RTK GPS Base Station

The Piksi base-station is shown in figure 6.2.1. Both the levels of the GPS receiver and radio transmitter can be adjusted. The conductive ground plane was highly recommended by the designers of the Piksi whereby the “ground plane boosts the sensitivity of the antenna towards the sky and helps block out unwanted signals that have reflected off the ground and cause interference (known as multipath)” [17].

It was also necessary to use a voltmeter to ensure that the voltage of the lead acid battery powering the base station did not drop to unsafe levels that may damage the battery.



Figure 6.2.1: Piksi base-station.

## 7. Software Architecture

With a general description of the required tasks and hardware given in previous sections, it is now appropriate to discuss, still at a relatively abstract level, the software that will implement the required functionality of the AGV.

Firstly a brief introduction to ROS is given in section 7.1, with section 7.2 listing all the pre-existing ROS software or “nodes” employed. Section 7.3 then identifies the relations between these nodes and original software written by project team members. This gives an overview of the entire AGV system. Finally, some information is given as to tools and methods the author and other project team members have found useful in developing and configuring the software for the AGV.

### 7.1. The “Robot Operating System” (ROS)

ROS is best described as a set of software libraries and tools that help the development of robot applications. This includes drivers, state-of-the-art algorithms and powerful developer tools. Additionally, the software is free, open source and free to profit on [18].

At present ROS is only supported on Ubuntu Linux. Additionally, each version of ROS is usually only supported on a few different versions of Ubuntu. At the commencement of the project, ROS Hydro Medusa was the latest version and was being used on Ubuntu Linux 12.04 LTS.

On the 22 July 2014 ROS Indigo Igloo was released for Ubuntu 14.04 LTS. It was decided upgrade the ROS system to this version, despite potential problems with version changes. Fortunately there were no considerable problems in upgrading to Igloo and for the most part source code and configuration files required little change.

The most important concepts of ROS are as follows:

- Data, wherever it comes from is “published” on “topics” and “nodes” can “subscribe” to these topics and do work on the data. Additionally nodes are generally the publishers of data.
- Several set structures for common types of data are specified by ROS as “messages”. For example, there exists messages for laser scan, IMU, velocity and pose data.
- Nodes are configured by “parameters” that can be set when the node is run
- Nodes can also provide “services” which are akin to remote procedure calls where the calling of a service triggers some desired function of the node.
- Several nodes can be configured and run by including them in a “launch” file and then “launching” that launch file.
- All nodes belong to “packages” and this provides a namespace when running individual nodes or calling launch files.

The ROS Wiki [19] provides several tutorials which elaborate further on the above concepts.

ROS is primarily accessed and initialized through the command line, though certain nodes, once run can represent the information “flowing” through ROS in graphical form. Probably the most important of these is “rviz” which visualizes topic data in an appropriate way, depending on the data type or message. Rviz has been an invaluable tool in solving numerous problems encountered in the development of the AGV, so an example of its usage is given in section 7.3.5.

Figure 7.1.1 is an example launch file that is useful for further explaining the important ROS concepts. In the figure, “p2os\_driver” is the node that provides the function of sending the velocity commands to the P3-AT base. It is part of the p2os\_driver package. It isn’t shown in the launch file, but the p2os\_driver node subscribes to the “cmd\_vel” topic. The programmed function of the p2os\_driver node is to send the appropriate velocity commands to the P3-AT in the form that it expects and cause the base to go in to the desired motion. For this to happen, the base has to be physically connected to the AGV PC, and this is through a USB to serial converter. Under Linux, devices are access by their “/dev/” filename. We see in the launch file that the p2os\_driver has the port parameter set to “/dev/gandobot/p3at1”, so this node knows to look for the P3-AT base on that port.

```

1 <launch>
2   <node name="base" pkg="p2os_driver" type="p2os_driver" args="pose:=odom">
3     <param name="port" value="/dev/gandobot/p3at1"/>
4     <param name="pulse" type="double" value="1.0"/>
5   </node>
6   <node name="rostopic" pkg="rostopic" type="rostopic" args="pub -1 /cmd_motor_state p2os_msgs/MotorState 1"/>
7
8   <include file="$(find p2os_urdf)/launch/upload_pioneer3at.xml"/>
9   <node pkg="robot_state_publisher" type="state_publisher" name="robot_state_publisher">
10    <param name="publish_frequency" type="double" value="30.0"/>
11    <param name="tf_prefix" type="string" value="" />
12  </node>
13  <node pkg="tf" type="static_transform_publisher" name="sick1_tf_broadcaster" args="0.135 0 0.25 0 0 0 top_plate sick1 100"/>
14  <node pkg="tf" type="static_transform_publisher" name="hok1_tf_broadcaster" args="0.223 0 0.060 0 0 0 top_plate hok1 100"/>
15  <node pkg="tf" type="static_transform_publisher" name="cam_tf_broadcaster" args="0.135 0 0 0 0 0 base_link cam_frame 100"/>
16
17  <node pkg="lms1xx" name="sick1" type="LMS1xx_node" args="scan:=scan_sick1">
18    <param name="host" value="192.168.7.3"/>
19    <param name="frame_id" value="sick1"/>
20  </node>
21
22  <node name="cam" pkg="usb_cam" type="usb_cam_node">
23    <param name="video_device" value="/dev/video1"/>
24  </node>
25
26  <node name="mapper_sick1" pkg="gmapping" type="slam_gmapping" args="scan:=scan_sick1 map:=map_sick1 map_metadata:=map_sick1_meta">
27    <param name="map_frame" value="map_sick1"/>
28    <param name="xmin" value="-10"/>
29    <param name="ymin" value="-10"/>
30    <param name="xmax" value="10"/>
31    <param name="ymax" value="10"/>
32  </node>
33 </launch>

```

**Figure 7.1.1:** Example launch file for aid in explaining important ROS concepts.

Additionally the p2os\_driver node publishes on the topic “odom”. The “pose:=odom” part of the launch file specifies that the default topic, “pose”, which p2os\_driver publishes its wheel encoder information is to be replaced or “remapped” to a topic called “odom”. This feature of remapping is useful for several reason such as when multiple instances of the same node are required and the data needs to be identified by different topic names.

It can be seen in the remainder of figure 7.1.1 that the node that provides the SLAM is called slam\_gmapping and is part of the gmapping package. The default topic this node subscribes to is “scan”, but we have remapped it to “scan\_sick1” to indicate that it is subscribing to the laser scan data provided by the LMS111 laser scanner. It can also be seen that the node that provides the LMS111 scan data is the LMS1xx\_node node that is part of the lms1xx package. Determining the function or context of each topic requires inspection of package or node documentation. Fortunately, most nodes have an associated documentation page on the ROS wiki with a standard structure. For instance, [10] is the wiki page for the gmapping package which indicates that “scan” is the topic which the slam\_gmapping node subscribes to, to obtain “sensor\_msgs/LaserScan” messages to create the map from.

Finally, there is an important topic called “tf” to which appropriate nodes publish to specify translations and rotations between different frames of reference. In figure 7.1.1 the “static\_transform\_publisher” node of the “tf” package publishes 3 different static transforms. For example, one of them is the relation between the top\_plate of the P3-AT base and the LMS111 laser scanner (the frame “sick1”). The node publishes a transform on the tf topic that specifies the relation between the top\_plate and sick1 frames of reference is such that the sick1 frame is 0.135m translated forward in the x direction, 0m in the y, and 0.25m in the z (using the right hand axis). The final 3 numbers represent the rotation in terms of Euler angles, which is zero. This is in fact the transformation used in figure 6.6.1 (note the x and y position of the top\_plate frame is in the center of the robot).

### 7.1.2. Employed Existing Packages and Nodes

Table 7.1.2.1 lists all the nodes currently employed in the AGV with a description of their function.

Package	Node	Functional Description
p2os_driver	p2os_driver	Hardware driver for the P3-AT
robot_state_publisher	state_publisher	Loads the URDF file of the P3-AT and publishes static transforms of useful frames of reference
tf	static_transform_publisher	Publishes static transforms between two frames specified by translation and rotation
lms1xx	LMS1xx_node	ROS driver for the LMS111
gmapping	slam_gmapping	Provides SLAM
move_base	move_base	Provides costmap and path planning functionality
sound_play	soundplay_node	Vocalizes text
xsens_driver	mtnode.py	ROS driver for the Xsens Mti-28A53G35
robot_localization	ekf_localization_node	Implements an extended Kalman filter for fusing odometry sources
map_server	map_server	Loads and latches a map on a topic
	map_saver	Saves a map from a topic
pointcloud_to_laserscan	pointcloud_to_laserscan	Converts a PointCloud2 message to the equivalent LaserScan message
hokuyo_node	hokuyo_node	ROS driver for the Hokuyo URG-04LX-UG01
rviz	rviz	Node for visualizing data

**Table 7.1.2.1:** List of all existing nodes currently employed in the AGV.

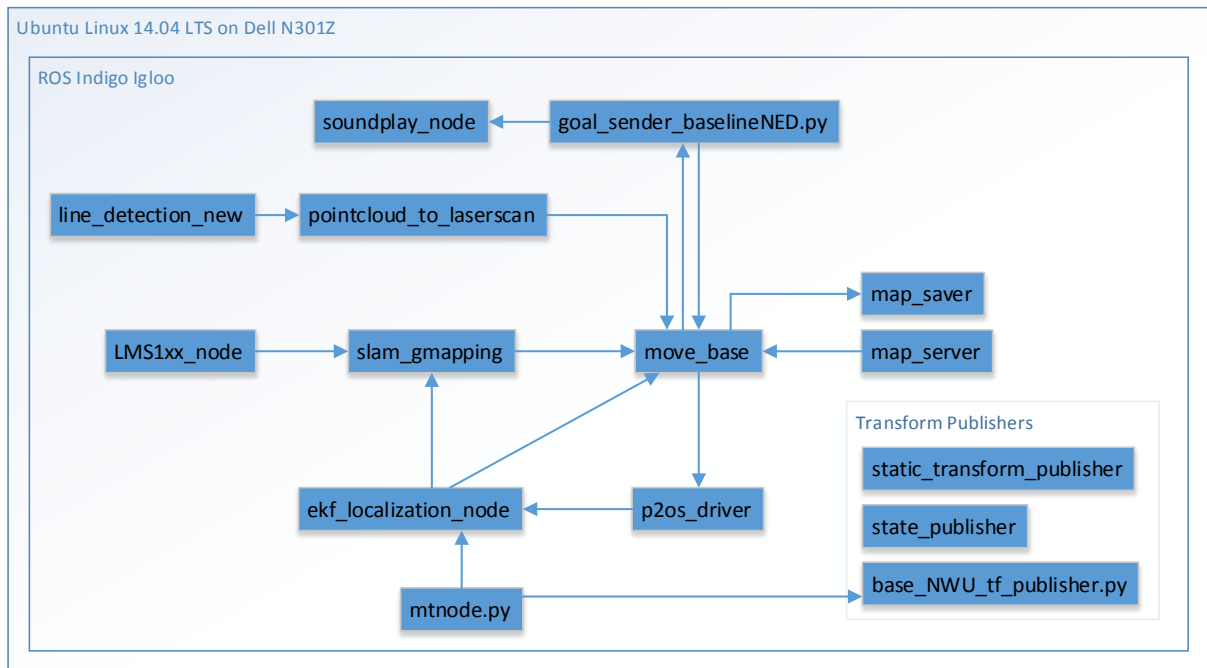
## 7.2. Abstract Software Subsystems

This section specifies the entire AGV software system in terms of its component subsystems (both existing ROS nodes and ones developed by the project team). Figure 7.2.1 shows these subsystems and their interrelations.

In terms of other team member’s tasks, Ross green was responsible for the development of the line\_detection\_new node (included in appx. 2 for completeness), in which the author made a contribution in converting the detected lines to the PointCloud2 message type.

Qing Zhou, after task reassignment (see section 5.3) was to investigate the ekf\_localization\_node node and endeavor to find a configuration which would produce the most accurate and precise odometry.

The remainder of the nodes in fig. 7.2.1 were left to the author to configure and implement. Out of these remaining nodes, only `goal_sender_baselineNED.py` and `base_NWU_tf_publisher.py` were to be implemented “from scratch” (these nodes’ functions are specified in section 9). The other nodes were pre-existing and required configuration to suite our specific situation.



**Figure 7.2.1:** Relation of existing and new software components.

### 7.3. Development Environment and Workflow

This section discusses some of the nuances in creating a ROS packages and nodes as well as some operating system considerations. Also introduced is “rviz”, ROS’s node responsible for visualizing data being published on topics and relations between frames of reference.

#### 7.3.1. Installation of Existing Packages

The installation of existing packages is usually as simple as adding the appropriate ROS software repository and obtaining the desired package from apt-get. The reason this is possible is that ROS packages use the “catkin” build system, which is based on CMAKE, for building nodes programmed in C++. The “CMakeLists.txt” file, which is present in every package containing C++ programmed nodes specifies build and run dependencies on other nodes. The ROS software repository uses this information to build the packages for the appropriate system (e.g. Ubuntu 14.04 LTS) and also for specifying apt-get run dependencies. It should also be noted that C++ and Python are the only programming languages at present supported by ROS.

In the 2014 AGV all existing packages are available on the repository apart from the `pointcloud_to_laserscan` package, which is expected to make its way there soon. For the AGV it was necessary to download the source of this package and compile it using the method detailed in [20].



### 7.3.2. Configuring a Workspace

It was necessary to create our own ROS package to give a namespace to our nodes and launch files. The package name we settled on for our code and launch files was “pandobot”. Nodes could then be run at the command line by entering the command “roslaunch pandobot line\_detection\_new” for example. Before this, however, the appropriate environment variables need to be set to configure the ROS system. This was achieved by sourcing our config.sh shell script (appx. 3, p. 1) in the user “.bashrc” startup script. Hence whenever a new terminal is opened, the appropriate environment variables are set. [21] details the process of creating a workspace whose path is included in “config.sh”. Environment variables that configure ROS’s networking features are also set in this file, but explanation of this is withheld until section 7.3.4.

### 7.3.3. Constant Device Names

The default behavior of plugging a USB device into a Linux system is that a device file such as “/dev/ttyUSB0” is created which is read or written, to access the device. The number of the device file depends of the number of devices already connected.

Constant device names were necessary so nodes whose parameters included the location of a device file did not need constant changing. This was achieved by using the Linux program “udev” which reads hardware data and assigns a device filename based on it. The program works by reading through a rules file whenever a device is plugged and checking for a match. Appendix 3, p. 2 is the rules file used by the AGV.

### 7.3.4. Computer Networking

A useful feature of ROS is its distributed computing capabilities. Essentially, nodes do not need to be run on the same PC, even if they are publishing or subscribing to topics. The configuration of this feature is detailed in [5], though figure 7.3.4.1 demonstrates how it was used in the 2014 AGV. In the figure one host is set as the “master”. Other hosts then set their “ROS\_MASTER\_URI” environment variable to the address of the master. It is also necessary the “/etc/hosts” file is configured correctly to translate hostnames into IP addresses.

This feature was useful whereby the AGV PC was set as the master and the control PC connected to it to visualize its data in rviz. The entirety of opening socket connections was taken care of by ROS behind the scenes.

This feature was mainly used over wifi in the lab, but it was tested and verified working over 4g/LTE. In that case, though, it was necessary to configure a VPN because each ROS host requires open ports.

```
1 #!/bin/bash
2
3 echo "sourcing ros files and setting up env. var"
4
5 source /opt/ros/indigo/setup.bash
6 source ~/AGVC/catkin_ws/devel/setup.bash
7
8 #export ROS_HOSTNAME=localhost
9 #export ROS_MASTER_URI=http://localhost:11311
10
11 export ROS_HOSTNAME=N3_wam
12 export ROS_MASTER_URI=http://N3_wam:11311
13
14 #export ROS_HOSTNAME=N3_lan
15 #export ROS_MASTER_URI=http://N3_lan:11311
16
17 #export ROS_HOSTNAME=N3_vpn
18 #export ROS_MASTER_URI=http://N3_vpn:11311
```

**Fig. 7.3.4.1:** Section of the cfg.sh file concerned with using ROS’s networking features.

### 7.3.5. Visualization Software

Rviz visualizes data broadcast over topics. It is shown in figure 7.3.5.1 below. In the figure the current state of the AGV's frames of reference are shown, along with laser scan data and the generated obstacle map. What is to be visualized is selected in the upper left pane. Video in ROS is considered to be a topic of images and can also be visualized in rviz.

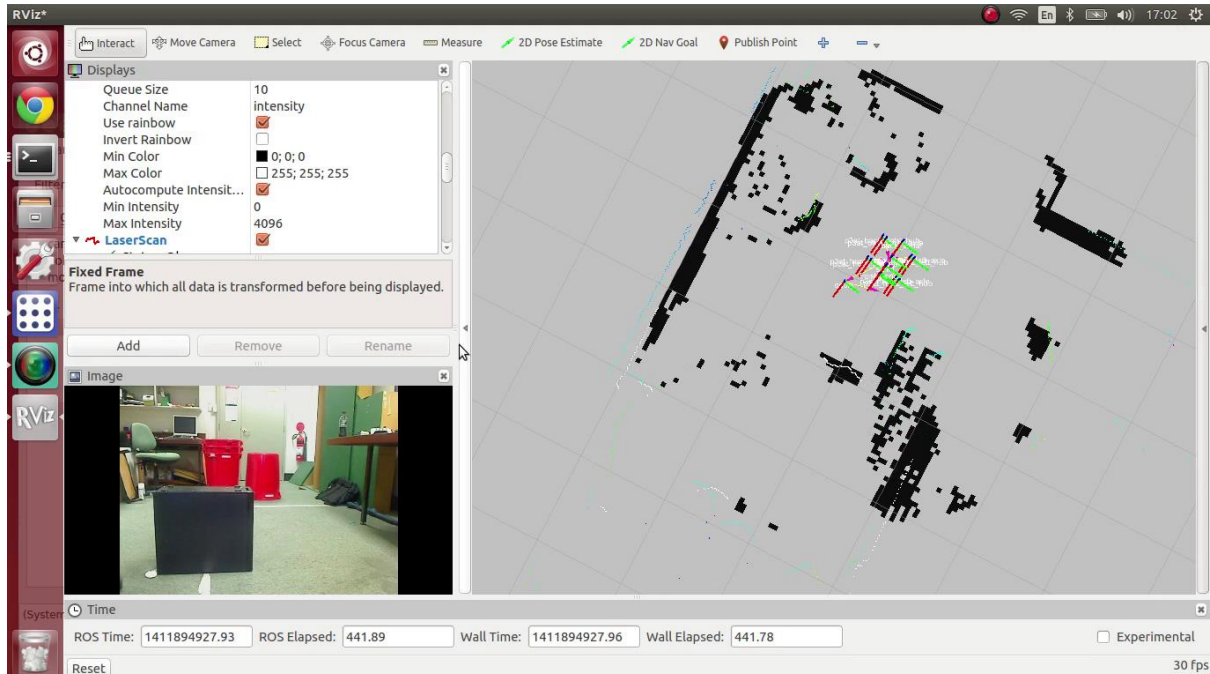


Figure 7.3.5.1: Rviz visualizing various “topics” of data.

### 7.3.6. Simulators

Simulators were not used this year in the expectation that they would distract from the production of a “real”, functioning AGV. Despite this, ROS supports Gazebo, Player and Stage and should possibly be used next year now that a functioning AGV has been developed.

## 8. Navigation Stack Configuration

This section details the author's main contribution the project – the configuration ROS's existing navigation software. The section starts with its basic configuration, then details other features required for the competition, such as point cloud obstacle input and map loading.

### 8.1. Rudimental Configuration

Autonomous navigation is primarily achieved through the `move_base` node, as shown in figure 7.2.1. It takes in localization data as well as obstacle sensor sources and constructs a “costmap” which is used to plan routes to specified destinations and sends the required velocity commands to the robot base.

Appendix 3, p. 3 (`all_basic.launch`) is a launch file that starts and configures all the nodes required for basic autonomous navigation within a 2-D environment (without lane obstacle input). The `move_base` node loads its parameters from 3 separate files – `global.yaml` (appx. 3, p. 5), `local.yaml` (appx. 3, p. 6) and `planner.yaml` (appx. 3, p. 7), configuring the global and local costmaps and path planner, respectively.

### 8.1.1. Costmap Parameters

The costmap parameters are shown in figure 8.1.1.1 for readability.

```

1 global_costmap:
2   plugins:
3     - {name: static,          type:
4       "costmap_2d::StaticLayer"}
5     - {name: obstacles1,     type:
6       "costmap_2d::VoxelLayer"}
7     - {name: obstacles2,     type:
8       "costmap_2d::VoxelLayer"}
9     - {name: inflater,       type:
10      "costmap_2d::InflationLayer"}
11
12  obstacles1:
13    observation_sources: scanner1
14    scanner1: {data_type: LaserScan, sensor_frame:
15              sick1, clearing: true, marking: true, topic: /
16              scan_sick1}
17
18  obstacles2:
19    observation_sources: scanner2
20    scanner2: {data_type: LaserScan, sensor_frame:
21              hok1, clearing: true, marking: true, topic: /scan_hok1}
22
23  footprint: [[-0.2485,-0.254],[-0.2485,0.254],
24             [0.2485,0.254],[0.2485,-0.254]]
25
26  global_frame: map_sick1
27  robot_base_frame: base_link
28  update_frequency: 5.0
29  publish_frequency: 2.0
30  width: 20.0
31  height: 20.0
32  origin_x: -10.0
33  origin_y: -10.0
34  resolution: 0.05
  
```

```

1 local_costmap:
2   plugins:
3     - {name: obstacles1,     type:
4       "costmap_2d::VoxelLayer"}
5     - {name: obstacles2,     type:
6       "costmap_2d::VoxelLayer"}
7     - {name: inflater,       type:
8       "costmap_2d::InflationLayer"}
9
10  obstacles1:
11    observation_sources: scanner1
12    scanner1: {data_type: LaserScan, sensor_frame:
13              sick1, clearing: true, marking: true, topic: /
14              scan_sick1}
15
16  obstacles2:
17    observation_sources: scanner2
18    scanner2: {data_type: LaserScan, sensor_frame:
19              hok1, clearing: true, marking: true, topic: /scan_hok1}
20
21  footprint: [[-0.2485,-0.254],[-0.2485,0.254],
22             [0.2485,0.254],[0.2485,-0.254]]
23
24  global_frame: odom
25  robot_base_frame: base_link
26  update_frequency: 5.0
27  publish_frequency: 2.0
28  rolling_window: true
29  width: 2.0
30  height: 2.0
31  resolution: 0.05
  
```

Figure 8.1.1.1: Global and local costmap parameters.

The effect of each parameter is briefly described in table 8.1.1.1.

Parameter	Description
plugins	<ul style="list-style-type: none"> <li>Specifies the “layers” to use in the costmap</li> <li>Obstacle layers add obstacles to the costmap</li> <li>A static layer seeds a costmap based on the map provided by the map_server node</li> <li>The inflation layer superimposes the previous layers and adds “padding” based on the footprint or size of the robot.</li> </ul>
obstacles1, obstacles2,	<ul style="list-style-type: none"> <li>Parameters of the specific layers</li> <li>sensor_frame is the frame of reference of the sensor</li> <li>topic is the topic which obstacle data is received on</li> <li>Clearing decides whether the sensor clears obstacles (i.e. if they move)</li> </ul>
global_frame	<ul style="list-style-type: none"> <li>The frame of reference in which to run the costmap</li> </ul>
robot_base_frame	<ul style="list-style-type: none"> <li>Where to treat the center of the robot</li> <li>The robot footprint is specified as each edge point’s distance from this point</li> </ul>
update_frequency	<ul style="list-style-type: none"> <li>How often to update the costmap</li> </ul>
publish_frequency	<ul style="list-style-type: none"> <li>Used for controlling how often to publish the costmap (i.e. the rate it is visualized in rviz)</li> </ul>
rolling_window	<ul style="list-style-type: none"> <li>Whether the costmap should move with the robot_base_frame or be tied to the global_frame</li> </ul>

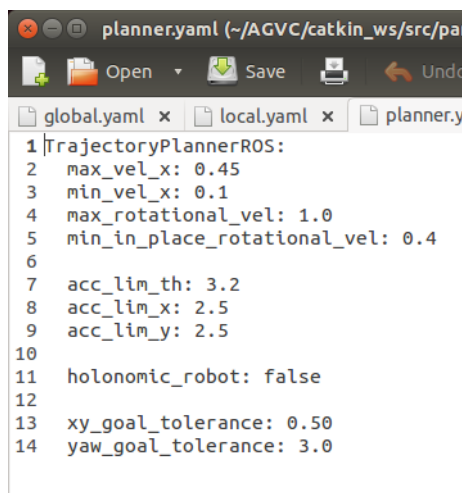
Table 8.1.1.1: Description of important costmap parameters.

The reason the navigation stack employs 2 separate costmaps is that it is required by the path planning subcomponent of move\_base (move\_base actually “wraps” several other nodes [22]). It is common for the frame of reference provided by the SLAM node to “jump” or jitter if it is having trouble localizing from landmarks. It would therefore be unideal to use this to determine the immediate velocity commands to be sent to the base in achieving a goal. The odometry frame of reference, however, does not suffer from this jittering phenomenon and so it is used to avoid obstacles over a short area as the local costmap. Odometry generally loses accuracy over large distances so a global costmap must also be used which maps obstacles in the SLAM frame of reference.

The costmaps were extremely difficult to understand and configure due to insufficient documentation. Perusal of the move\_base.cpp source file [23] was required in addition to the main documentation page [22].

### 8.1.2. Path Planner Parameters

Figure 8.1.2.1 shows the path planner parameters. These parameters are set as to satisfy the speed requirements of the AGVC specified in section 4.2.1. There are also parameters for tolerance in achieving a goal. Goal yaw tolerance is set high because it is unimportant (i.e. it doesn’t matter which way the AGV is facing when it reaches a goal) and setting it low can result in the AGV not achieving a goal or taking too long to achieve it while the AGV rotates back and forth trying to acquire the desired orientation.



```
1 |TrajectoryPlannerROS:
2   max_vel_x: 0.45
3   min_vel_x: 0.1
4   max_rotational_vel: 1.0
5   min_in_place_rotational_vel: 0.4
6
7   acc_lim_th: 3.2
8   acc_lim_x: 2.5
9   acc_lim_y: 2.5
10
11  holonomic_robot: false
12
13  xy_goal_tolerance: 0.50
14  yaw_goal_tolerance: 3.0
```

**Fig. 8.1.2.1:** Path planner parameters.

## 8.2. Visual “Lane” Obstacle Integration

Ross Green’s line detection is fed into the pointcloud\_to\_laserscan node for reasons specified in [24]. A layer is then created in both the global and local costmaps that uses the laser scan topic output by the pointcloud\_to\_laserscan node as a sensor source.

Appendix 3, p. 8 is the launch file that launches the line\_detection\_new node and the pointcloud\_to\_laserscan node with the appropriate parameters.

The transformation of the camera’s frame of reference to the common top\_plate is defined in the all\_basic.launch launch file.

The detected lanes then come up as obstacles in the costmaps as shown in figure 8.2.1 overleaf.

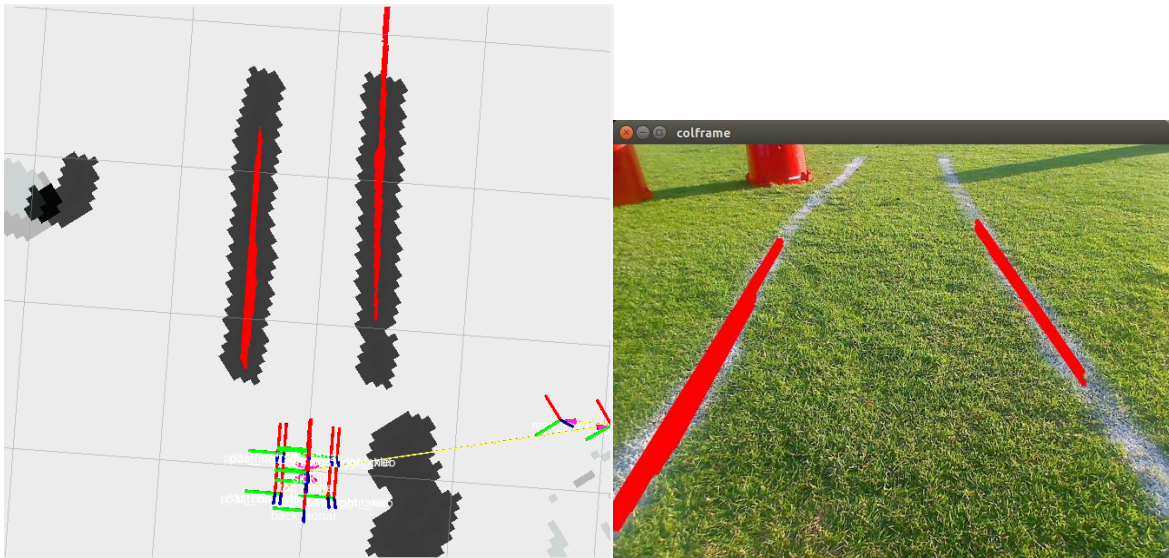
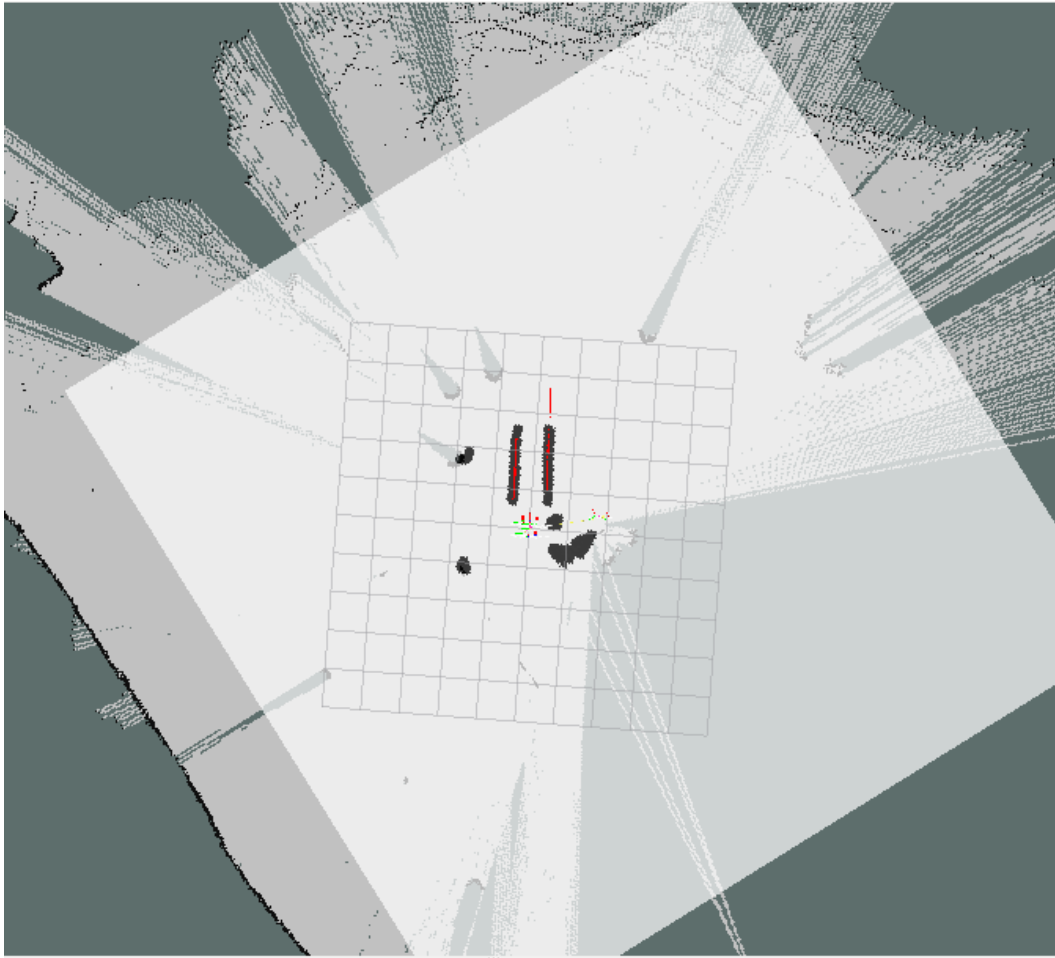


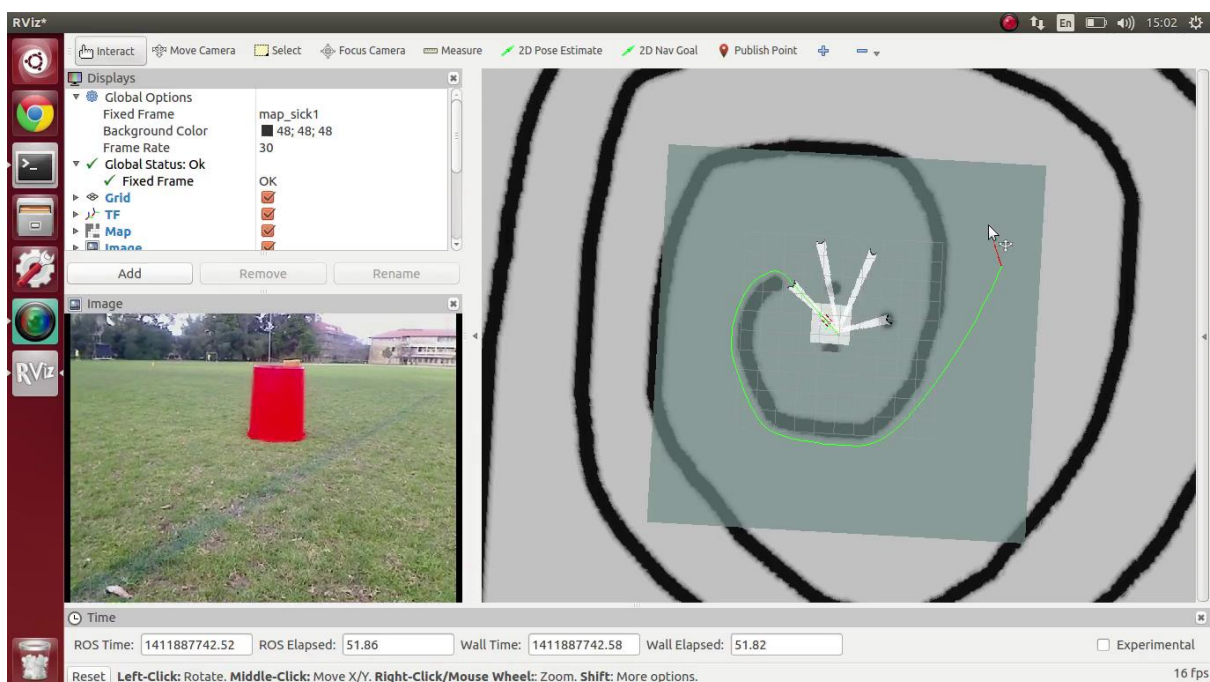
Figure 8.2.1: Detected lanes populating the costmaps.

### 8.3. Map Loading and Saving Functionality

Optional, but desired map loading functionality is implemented through a static layer, which seeds the costmap based on a map obtained from the map\_server node. The following code should be run in a launch file to start the map\_server node connected to a specified map file:

```
<node name="map_server" pkg="map_server" type="map_server"
args="$ (find pandobot) /launch/map.yaml">
  <param name="frame_id" value="map_sick1"/>
</node>
```

Figure 8.3.1 shows a spiral map being loaded and seeding the costmap. When given navigation goals, the AGV treats the spiral as if it were an actual obstacle. The path generated from the path planner in attempting to reach the given goal is shown as the green line.



**Figure 8.3.1:** Loaded map being treated as an obstacle.

The map saving functionality whilst the robot progresses through the course has yet to be implemented, but is planned to be as follows:

- A separate costmap node must be run which subscribes solely to the lane laser scan topic output by the pointcloud\_to\_laserscan node (the reasons for this are detail in [25]).
- The terminal command “roslaunch map\_server map\_saver” will be run periodically by cron<sup>9</sup> to save the lines map to file.

### 8.4. Improved Odometry Through Sensor Fusion

This was originally to be investigated by the author, though it was later assigned to Qing Zhou for reasons explained in section 5.3. Despite this, substantial contributions by the author were made.

The main result was the great improvement in odometry by using the ekf\_localization\_node to create a new odometry topic from the orientation velocity of the IMU and the forward velocity of the P3-AT.

<sup>9</sup> Cron is a time-based job scheduler in Unix-like computer operating systems.

The filter supports the fusion of multiple velocity and orientation topics. Another input to the filter could be the base velocity obtained from the Piksi GPS, though time constraints resulting from operational problems of the Piksi have forbid this.

An important thing to note in using this node is that by default, it provides an odom  $\rightarrow$  base\_link transformation – representing the improved odometry – published on the tf topic. The p2os\_driver node also publishes the same transform on this topic, so it must be remapped to another topic (as in all\_basic.launch) or disabled in the source code and recompiled.

## 9. Rudimental “Goal Planning” Subsystem

This section details the GPS  $\rightarrow$  SLAM map coordinate conversion and the sending of goals to the navigation stack. It was felt appropriate to include these two functions in one node – goal\_sender\_baselineNED.py (appx. 3, p. 9). This node, at present, only sends goals in coordinates relative to the base-station (north and east of it). This is because there is a bug in the Piksi firmware which causes the modules to fail when the base station is set to surveyed GPS mode [26]. This bug has since been fixed in the 0.12 firmware which was released on the 24<sup>th</sup> of November 2014 [27].

There is no goal planning algorithm, as described in section 5.3.3, per say. The goal\_sender\_baselineNED.py node, at present creates a fixed array of “GoalAndStatus” objects and their relative NED coordinates are specified in the script. There are then a series of helper functions that send goals to the navigation stack and update goals as they are achieved.

“cb\_baseline\_NED(data)” reads in the current GPS solution and assigns it to a global variable. This function should be called periodically if the GPS stays in fixed RTK mode (the highly accurate mode).

The “cb\_done(status, res)” callback function mainly controls the flow of the script. When it is called the navigation stack has “finished” with a goal – whether it was achieved or not is determined by the status variable passed to the function. Depending on the status, the function sets the appropriate variables in the goal object and determines the next goal to send to the navigation stack.

Other than that, the “send\_goal” method of the client object (the interface to the navigation stack) requires explanation. When a goal is sent to the navigation stack it expects a frame of reference. It was previously mentioned that the map\_sick1 frame of reference is the reliable frame reference; so we want to send goals relative to that. The problem is, however, when navigating to GPS coordinates, even simple “north east down” (NED) coordinates, the direction of true north must be known. This is where the elegance of ROS is shown – by the use of a node which defines a dynamic transformation between the “north west up” coordinate system and base\_link (the rotational center of the robot). The “base\_NWU\_tf\_publisher.py” node (appx. 3, p. 12) is exactly this node. It achieves its functionality by subscribing to the true north vector output by the IMU. With this node running, goals are specified in the absolute “north west up” coordinate system and ROS takes care of the transformation to the map\_sick1 frame of reference required by the navigation stack.

Latitude and longitude coordinates, which are the standard and expected coordinates to be used in the AGVC are planned to be implemented in the goal planning subsystem by converting them to the UTM coordinate system<sup>10</sup> and calculating the north and east distance between two points. This can then be sent to the navigation stack using the same method outlined in the previous paragraph. This feature is partially implemented in appx. 3, p. 14. The latitude and longitude  $\rightarrow$  UTM transformation is provided by a python function in the ROS geodesy package.

---

<sup>10</sup> The Universal Transverse Mercator (UTM) conformal projection uses a 2-dimensional Cartesian coordinate system to give locations on the surface of the Earth.

## 10. Results

### 10.1 Navigation Performance

#### 10.1.2. Against Landmark Sparsity

The navigation stack (without lane obstacles) was tested on James Oval with varying clusters of barrel obstacles acting as landmarks for the SLAM component of the navigation stack. The primary metric of the test was the accuracy of the localization as the AGV moved to arbitrary goals in the vicinity of the landmarks. This metric is hard to quantify numerically, so a worded description of the results is given in table 10.1.2.1. The barrels used in the test had a 30cm radius at the plane scanned by the laser. The laser scanner range had to be reduced to 30m by the official configuration software to stop the periphery of James Oval from being detected (this can be seen in fig. 8.2.1, where the laser scanner has a range of 80m).

Cluster Size	Results
1 barrel	<ul style="list-style-type: none"><li>• Instantaneous “jitters” of the base_link frame</li><li>• This configuration has the most jitters of the base_link frame, but other than that, measured distances reported by rviz and specific landmarks coincide with the actual distance with an error no more than 10cm</li></ul>
2 barrels 3 meters apart	<ul style="list-style-type: none"><li>• Similar results as above, less frequency of jitters</li></ul>
6 barrels scattered with an average distance of 4 meters between them	<ul style="list-style-type: none"><li>• No jitters</li><li>• Accuracy again within 10cm</li></ul>
2 lots of the previous configuration with a distance of 20 meters between them	<ul style="list-style-type: none"><li>• No jitters when navigating the AGV between the two clusters</li><li>• Same accuracy</li></ul>

**Table 10.1.2.1:** Results of testing the navigation stack on James Oval, without lane obstacle layer.

These general results indicate the AGV is localizing very accurately and base\_link jitters are infrequent if none. The tested conditions superset those expected in the AGVC, so the use of slam\_gmapping to localize the AGV is considered successful.

#### 10.1.3. With Lane Obstacle Layer

As in section 10.1.2, numerical data is difficult to obtain. Also, due to time constraints this function has not been tested as much as others – such as localization, map loading and GPS integration. What has been tested, though, looks promising.

We tested this functionality by spray painting two short white lines on James Oval as shown in figure 8.2.1. Five barrels were scattered around the lines (as can be seen in the costmap) to help with localization. The navigation stack was then sent several goals in the map\_sick1 frame of reference which intentionally transected the white lanes and it avoided them. The AGV was even able to navigate between the white lanes when given the appropriate goals.

### 10.2. Piksi RTK GPS Performance

With suitable elevation, the Piksi RTK GPS system has been verified working up to ranges of 300m by Ruvan Muthu-Krishna of the REV project.

Independent testing by the author has revealed an accuracy of approximately 60cm. This was arrived at by viewing the “distance from base-station” output of the Piksi console software at various measured positions from the base station (up to 35m using a measuring wheel, tape and spikes).



### 10.3. GPS Integration Performance

The `goal_sender_baselineNED.py` (appx. 3, p. 16) has been tested working by giving it a goal from the negative of its current GPS position. This should theoretically send the AGV to the base-station from whatever location the script is called.

It is absolutely necessary to ensure that the correct magnetic declination has been flashed to the IMU before calling the `goal_sender_baselineNED.py` script. The author used the magnetic declination provided by the official IMU software based on latitude and longitude.

It was found that the AGV was able to navigate to less than 1.5m from the base-station from points up to 100m away from it (similar to expected competition conditions). The AGV navigated closer to the base-station from closer initial positions. This indicates that the Piksi should be sufficient in providing the GPS data that will be used to send the AGV to the 2m radius waypoints featured in the AGVC.

## 11. Conclusions

All the subsystems of the AGV are implemented, but some of them are only in rudimentary form. More work will be required after semester 2 2014 in preparing the AGV for the competition. This is mainly further lane obstacle testing and work on the goal planning subsystem.

The goal planning subsystem, from the results in 10.3 should be able to satisfy the requirements of navigating to the 2m radius waypoints. This, however needs to be verified after it is fully complete, because the latitude to longitude  $\rightarrow$  UTM transformation could introduce some error. If that is the case, however, it should be rectifiable by comparing the AGV's actual latitude longitude position with the goal position when it "supposedly" achieves its goal; and if it's not within a certain tolerance, send the goal again with a recomputed UTM transformation.

Obviously, the actual goal planning algorithm still needs a substantial amount of work. If it is not ready for the competition, the author will request of the competition facilitators that the optimal waypoint visitation order is programmed into the goal sender just to demonstrate that the remaining functionality is has been implemented.

Apart from these problems, and the initial task assignment problems, the project has been a great success. The author has learnt a substantial amount about common robotics methods, frameworks and problems. The author is also looking forward to attending the 2014 AGVC and using ROS in his own personal robotics projects.

Appendix 4 consists of documents obtained from a Curtin University lecturer providing surveying tutorials to UWA civil engineering students on James Oval who the author and Ruvan Muthu-Krishna conversed with on a couple of occasions whilst testing the operation of the Piksi RTK GPS.

Unfortunately we did not get to know his name, otherwise he would be mentioned in section 3, acknowledgments. He, however provided us with a wealth of information concerning the operation of the entire GPS system and methods such as fixed RTK. Appendix 4 consists of his surveyed GPS points of two landmarks on James Oval – one near the civil and mechanical building and one near the business school building. The ultimate test of the AGV project will be setting up the Piksi base-station at one of these points and autonomously navigating the AGV to the other with lane and barrel obstacles in the way.

## 12. References

- [1] (10/24/2014). AGVC » *About*. Available: <http://www.agvc.net.au/about/>
- [2] (10/24/2014). AGVC » *Key Dates*. Available: <http://www.agvc.net.au/about/key-dates/>
- [3] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, pp. 101-132, 2007.
- [4] R. Hedges. (2014, 5/23/2014). *ROSARIA - ROS Wiki*. Available: <http://wiki.ros.org/ROSARIA>
- [5] M. Purvis. (2013, 5/23/2014). *ROS/NetworkSetup - ROS Wiki*. Available: <http://wiki.ros.org/ROS/NetworkSetup>
- [6] A. Elkady and T. Sobh, "Robotics middleware: A comprehensive literature survey and attribute-based bibliography," *Journal of Robotics*, vol. 2012, 2012.
- [7] E. Einhorn, T. Langner, R. Stricker, C. Martin, and H. Gross, "Mira-middleware for robotic applications," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, pp. 2591-2598.
- [8] M. Namoshe, N. Tlale, C. Kumile, and G. Bright, "Open middleware for robotics," in *Mechatronics and Machine Vision in Practice, 2008. M2VIP 2008. 15th International Conference on*, 2008, pp. 189-194.
- [9] S. Kohlbrecher. (2014, 5/23/2014). *hector\_slam - ROS Wiki*. Available: [http://wiki.ros.org/hector\\_slam?distro=hydro](http://wiki.ros.org/hector_slam?distro=hydro)
- [10] A. Hacinecipoglu. (2013, 5/23/2014). *gmapping - ROS Wiki*. Available: <http://wiki.ros.org/gmapping>
- [11] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, 2011, pp. 155-160.
- [12] C. S. Giorgio Grisetti, Wolfram Burgard. (10/25/2014). *OpenSLAM.org*. Available: <https://www.openslam.org/gmapping.html>
- [13] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and accurate SLAM with Rao-Blackwellized particle filters," *Robotics and Autonomous Systems*, vol. 55, pp. 30-38, 2007.
- [14] (10/25/2014). *Piksi - Swift Navigation*. Available: <http://swiftnav.com/piksi.html>
- [15] (10/26/2014). *Wireless LANs in the 2.4 GHz band FAQ*. Available: <http://www.acma.gov.au/Citizen/Consumer-info/My-connected-home/Wireless-local-area-networks/wireless-lans-in-the-24-ghz-band-faqs>
- [16] S. Spiedel, "AGVC Technical Document."
- [17] (10/26/2014). *Piksi User Getting Started Guide - Swift Navigation Wiki*. Available: [http://docs.swiftnav.com/wiki/Piksi\\_User\\_Getting\\_Started\\_Guide](http://docs.swiftnav.com/wiki/Piksi_User_Getting_Started_Guide)
- [18] (10/26/2014). *ROS.org | Powering the world's robots*. Available: <http://www.ros.org/>
- [19] P. Rey. (2013, 5/23/2014). *Documentation - ROS Wiki*. Available: <http://wiki.ros.org/>
- [20] (10/26/2014). *catkin/Tutorials/using\_a\_workspace - ROS Wiki*. Available: [http://wiki.ros.org/catkin/Tutorials/using\\_a\\_workspace](http://wiki.ros.org/catkin/Tutorials/using_a_workspace)
- [21] (10/26/2014). *catkin/Tutorials/create\_a\_workspace - ROS Wiki*. Available: [http://wiki.ros.org/catkin/Tutorials/create\\_a\\_workspace](http://wiki.ros.org/catkin/Tutorials/create_a_workspace)
- [22] (10/26/2014). *move\_base - ROS Wiki*. Available: [http://wiki.ros.org/move\\_base](http://wiki.ros.org/move_base)
- [23] (10/26/2014). *navigation/move\_base.cpp at hydro-devel · ros-planning/navigation*. Available: [https://github.com/ros-planning/navigation/blob/hydro-devel/move\\_base/src/move\\_base.cpp](https://github.com/ros-planning/navigation/blob/hydro-devel/move_base/src/move_base.cpp)
- [24] (10/26/2014). *publishing 0 points in a point cloud - ROS Answers: Open Source Q&A Forum*. Available: <http://answers.ros.org/question/194212/publishing-0-points-in-a-point-cloud/>
- [25] (10/26/2014). *attaching Costmap2DROS object to existing costmaps (i.e. from movebase) - ROS Answers: Open Source Q&A Forum*. Available: <http://answers.ros.org/question/191620/attaching-costmap2dros-object-to-existing-costmaps-ie-from-movebase/>

- [26] (10/26/2014). *Bug report - Google Groups*. Available: <https://groups.google.com/forum/#!topic/swiftnav-discuss/r3CvFt-pemc>
- [27] (10/26/2014). *downloads.swiftnav.com/piksi\_v2.3.1/stm\_fw/*. Available: [http://downloads.swiftnav.com/piksi\\_v2.3.1/stm\\_fw/](http://downloads.swiftnav.com/piksi_v2.3.1/stm_fw/)

### 13. Appendixes

The appendixes are located in the appropriate pdf files. Table 13.1 below is a brief description of them.

Appendix File	Appendix Number	Page	Description
appx1.pdf	1	01/34	2014 AGVC rules
		21/34	2014 AGVC intent to compete form
		22/34	2014 AGVC entry form
		26/34	Initial Gantt chart
appx2.pdf	2	01/20	Ross Green's line detection code
appx3.pdf	3	01/16	config.sh
		02/16	udev rules file
		03/16	all_basic.launch
		05/16	global.yaml
		06/16	local.yaml
		07/16	planner.yaml
		08/16	Launch file for starting up line detection
		09/16	Basic goal sending Python script
		12/16	NWU publisher tf publisher
		14/16	Experimental latitude longitude goal sender
appx4.pdf	4	01/02	GPS coordinates of James Oval point 1
		02/02	GPS coordinates of James Oval point 1

**Table 13.1:** Listing of appendixes

# AGVC 2014

## Introduction

Advances in robotic technology offers land forces new operational capabilities resulting in greater survivability, flexibility and sustainability. Although military robots will continue to be operated under some human control it is desirable that in future there will be less need for human intervention. Robots will operate with a higher level of autonomy and independent behaviour and the operator will move from 'in-the-loop' to 'on-the loop'. Currently the principle limiting factors on this higher degree of autonomy are the reliability of the system and task complexity. The major requirement to fill these gaps is the development of more sophisticated sensing and decision making algorithms.

The Autonomous Ground Vehicle Competition (AGVC) initiative aims at sourcing proposals for the next generation of autonomous ground robotic vehicles from Australian Universities' by increasing their exposure to this highly complex technical area and promoting robotic research for real word applications. AGVC will be part of university teaching program where graduate/post-graduate students learn theoretical knowledge-based intelligence skills needed to develop their skill level to meet future robotic requirements for Defence applications.

AGVC-13 was held on the 28 Nov. - 1 Dec. 2013 at Deakin University grounds at Waurm Ponds Campus, Geelong (as shown in Figure 1). The ten teams who competed and obtained rankings are listed in Table 1.



*Figure 1: Snapshot of the outdoor obstacle course held at Deakin University, Geelong.*

For AGVC-2013 Table 1 reports the winners and compares the 10 team rankings for the nominal awards, as none of the teams passed the money barrel (located at the latter end of the course) for the major prizes. The Design Test Score is a combination of written report, oral presentation and examination of the vehicle. The 3 Winners for Nominal awards in order of ranking for AGVC-13 were as follows:

- (1) Order 66 (Deakin University)
- (2) dUNSWiftly (UNSW)
- (3) Aperire Incognitum (Deakin University)

The Design Test Award Innovation prize was given to Team Trial & Error for their design innovative in their hardware strategy work. Figure 2 shows Team Order 66 and Team Zelos preparing, practicing and fine tuning their robots during AGVC-13.

Team Name & Affiliation		Ranking for Nominal Awards
Trial & Error	ANU	6
Order 66	Deakin Uni	1
Dynamic	RMIT (Team 1)	5
Team Redback	Flinders Uni	4
Aperire Incognitum	Deakin Uni	3
Team UQ (Elderly Gentleman)	Uni of Qld	0
dUNSWiftly	UNSW	2
UWA Robotics	Uni of WA	0
Team Zelos	Uni of Adelaide	0
Team Tesla	RMIT (Team 2)	6

*Table 1: Reports the Winners and Compares the AGVC-2013 Team Ranking for Nominal Awards*



*Figure 2: Shows Team Order 66 practicing out in the field and Team Zelos fine tuning their Robots during AGVC-13.*

Chapter 2 of the report outlines the rules and technical qualifications for AGVC-14.

## **Autonomous Ground Vehicle Competition 2014**

### **The Objective of the AGVC-14**

AGVC-14 is organised by DSTO in collaboration with Deakin University. It describes the configuration of two obstacle courses in the Auto Navigation part of the competition. The overall judging also includes a written report, oral presentation and examination of the vehicle as the UGV design strategy is an important part of systems engineering and learning for students. The winners are selected based on the sum of both the Design and Auto Nav. competitions.

The Objective of AGVC-14 is to develop the theoretical and practical knowledge of robotics of students across a number of disciplines and to build personal and institutional experience in developing the technologies and skills needed to produce an Unmanned Ground Vehicle (UGV) for real world applications, including military operations. Under the guidance of DSTO and Deakin University this will not only provide the foundations for students to compete in future international competitions but also stimulate the development of Australia's research, science, innovation and technology base. Examples of some the technology and skill areas needed are:-

- sensor data fusion,
- image and sensor processing,
- target recognition,
- artificial intelligence,
- knowledge based systems
- open system architectures,
- machine vision,
- autonomous navigation and mapping,
- modelling and simulation,
- human machine interfaces and integration,
- computer hardware and software designs,
- mechanical, electronic architectures and systems

- communication networks
- developing fast search algorithms
- multi vehicle coordination teaming algorithms
- Hardware sensor systems
- Real-time computer hardware and software systems.

## Competition Overview

The competition comprises three stages: technical qualification, an autonomous navigation test and an evaluation of the AGV design.

- Technical qualification tests the vehicle and team compliance with the competition rules. Technical Qualification is described at Appendix A. It involves a technical inspection and performance on an outdoor qualifying course. Vehicle's must pass technical qualification before proceeding to the later stages of the competition.
- The Auto Nav test (described in Appendix B), requires autonomous navigation of an advanced outdoor course. The advanced course will be set up with obstacles and navigation waypoints and needs to be completed within a specified time.
- Design innovation is one of the key objectives and will be given special attention by the judges. Innovation is defined as a technology (hardware or software) that has not been used by this or any other vehicle in this competition. The design portion of the competition will examine the strategy and process that teams followed to produce their vehicle and will evaluate the innovation and ingenuity applied. The judging of AGV design will rest on evaluation of a written report, oral presentation and examination of the vehicle. This is described in detail at Appendix C.

Overall placings will be based on the combined scores achieved on the Auto-Nav test and the Design competition. The top three teams will be ranked.

Teams may represent any Australian and \* New Zealand tertiary institution. Teams may comprise only tertiary students, and must be supervised by at least one faculty/academic advisor. Interdisciplinary teams are encouraged. Only the student component of each team will be eligible for the cash awards however, winning organisations will be awarded trophies.

## Summary and Conclusion

AGVC-14 is organised by DSTO in collaboration with Deakin University. It describes the configuration of two obstacle courses in the Auto Navigation part of the competition. The overall judging also includes a written report, oral presentation and examination of the vehicle as the UGV design strategy is an important part of systems engineering and learning for students. The winners are selected based on the sum of both the Design and Auto Nav. competitions.

In conclusion the AGVC Competition:

- Progresses key technology areas in robotic automated systems that can improve the military's ability to operate in high threat environments without unacceptably increasing the risks faced by soldiers.
- Helps develop these highly specialised technologies within our local Universities and industry.

- Prepares successful Australian competitors, to be able to compete in International Robotic Competitions by TARDEC and DARPA in the USA.

## References

[1] The 20<sup>th</sup> Annual Intelligent Ground Vehicle Competition (IGVC), June 8-11<sup>th</sup>, 2012 Oakland University, Rochester, Michigan, [www.igvc.org](http://www.igvc.org).



# Appendix A: Competition Rules and Technical Qualifications [1]

## Team Entry

- Team member/s will be currently enrolled tertiary student/s supervised by at least one academic/faculty member.
- No institution shall enter more than 3 teams.
- Only student/s will be allowed to compete at the event. Each team will have a team name and distinctive uniform.

(Note \* signifies alteration from AGVC-13)

## Technical Qualifications [1]

All vehicles must pass the technical inspection and negotiate the Qualifying Navigation Course to be eligible to proceed to the remainder of the competition. The technical inspection will confirm that vehicles conform to the specification laid down for the IGVC organised by Oakland University/ TARDEC [1]. The Qualifying Navigation Course tests the vehicle's ability to maintain requisite speeds, its navigation ability, lane keeping and obstacle avoidance.

## Technical Inspection

In the Technical inspection the following vehicle characteristics will be confirmed:

- **Source.** The vehicle chassis may be fabricated or commercially bought.
- **Design:** Must be a ground vehicle (propelled by direct mechanical contact to the ground such as wheels, tracks, pods, etc. or hovercraft).
- **Length:** Minimum length 0.9 meters, maximum length 2.1 meters.
- **Width:** Minimum width 0.6 meters, maximum width 1.2 meters.
- **Height:** Not to exceed 1.8 meters (excluding emergency stop antenna).
- **Propulsion:** Vehicle must be battery powered. Teams are responsible for charging their own batteries. However standard 15A single-phase power will be available in team maintenance areas.
- **Mechanical E-stop:** Each vehicle must have a mechanical e-stop. The E-stop button must be a push to stop, red in colour and a minimum of 2 cm in diameter. It must be easy to identify and activate safely, even if the vehicle is moving. It must be located in the centre rear of vehicle at least 0.6 meter from ground, not to exceed 1.2 meter above ground. Vehicle E-stops must be hardware based and not controlled through software. Activating the E-Stop must bring the vehicle to a quick and complete stop.
- **Wireless E-Stop:** Each vehicle must have wireless E-stop which can be hand carried by the judges. The wireless E-Stop must be effective for a minimum of 30 meters.

Vehicle E-stop must be hardware based and not controlled through software. Activating the E-Stop must bring the vehicle to a quick and complete stop.

- **Safety Light:** The vehicle must have an easily viewed solid indicator light which is turned on whenever the vehicle power is turned on. The light must go from solid to flashing whenever the vehicle is in autonomous mode. As soon as the vehicle comes out of autonomous mode the light must go back to solid.
- **Payload:** Each vehicle will be required to carry a 9 kilograms payload with approximate configuration of 46 cm x 20 cm x 20 cm. The payload must be securely mounted on the vehicle. If the payload falls off the vehicle during a run, the run will be terminated.

### **Qualifying Navigation Course [1]**

The Qualifying Navigation Course: The general layout of the Qualifying Navigation Course is shown at Figure 3. It will be approximately 30 meters wide and 60 meters deep and consist primarily of sinusoidal curves with series of repetitive barrel style obstacles held on a possibly grassy landscape. A waypoint pair will be on the course prior to completion. Maximum time for the basic course will be based on 1.6 kph over the computed distance. Other aspects of the Qualifying Course, such as marking and obstacles etc, are the same as the Advanced Course described in Appendix B. Figure 4 shows portions of the Qualifying Navigation Course used in AGVC-13. The top part of the left image shows the qualifying course with lanes and obstacle bins and the right image displays a close up of the field.

*On the Qualifying Navigation Course vehicles will be tested for the following:*

- **Speed:** The vehicle will have to drive over a prescribed distance where its minimum and maximum speeds will be determined. The vehicle must not drop below the minimum of 1.6 km per hour nor exceed the maximum speed of 16 km per hour. Minimum speed of 1.6 kph will be assessed in the fully autonomous mode and verified over a 15 meters distance between the lanes and avoiding obstacles. No change to maximum speed control hardware is allowed after qualification. If the vehicle completes a performance event at a speed faster than the one it passed Qualification at, that run will not be counted.

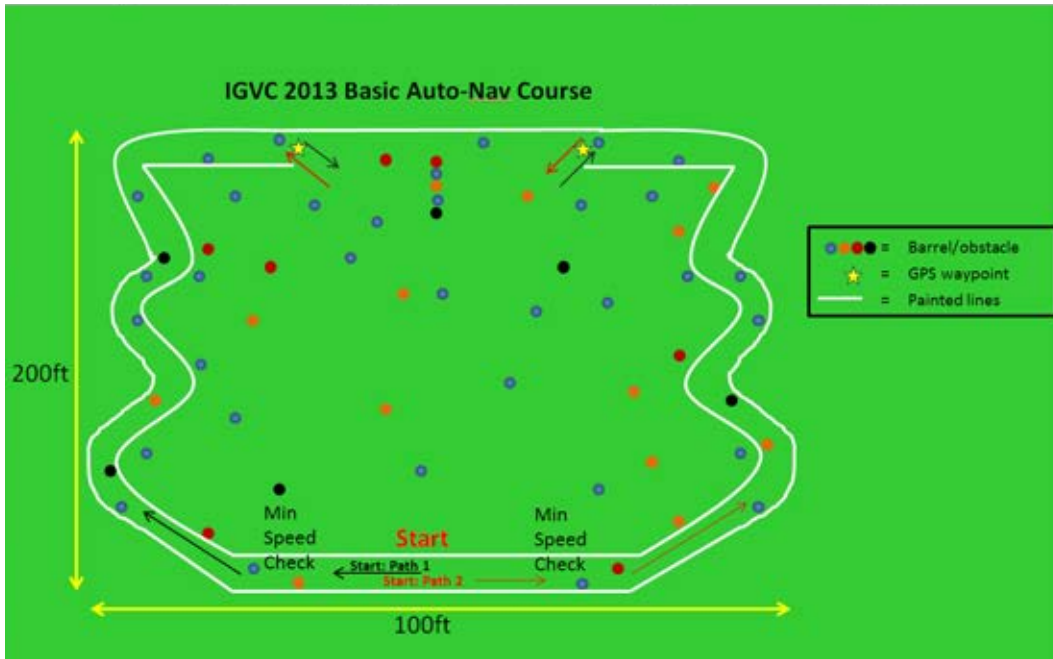


Figure 3: Qualifying Navigation Course (Example Adapted from IGVC) [1]



Figure 4: Shows portions of the Qualifying Navigation Course used in AGVC-13. The top part of the left image shows the start of the qualifying course with lanes and obstacle bins and the right image displays a close up of this field.

- **Lane Following:** The vehicle must demonstrate that it can detect and follow lanes.
- **Obstacle Avoidance:** The vehicle must demonstrate that it can detect and avoid obstacles.
- **Waypoint Navigation:** Vehicle must prove it can find a path to a single two meter navigation waypoint by navigating around an obstacle.

During the Qualification the vehicle must be put in autonomous mode to verify the mechanical and wireless E-stops and to verify minimum speed, lane following, obstacle avoidance and waypoint navigation. The vehicle software cannot be reconfigured for waypoint navigation qualification. It must be integrated into the original autonomous software. For the max speed run the vehicle may be in autonomous mode or joystick/remote controlled. Judges will not

qualify vehicles that fail to meet these requirements. Teams may fine tune their vehicles and resubmit for Qualification. There is no penalty for not qualifying the first time. Vehicles that are judged to be unsafe will not be allowed to compete. In the event of any conflict, the judges' decision will be final.

# Appendix B: Autonomous Navigation Test [1]

## General

- A fully autonomous unmanned ground robotic vehicle must negotiate around an outdoor obstacle course under a prescribed time while maintaining a minimum of speed of 1.6 kph over a section and a maximum speed limit of 16 kph, remaining within the lane, negotiating flags and avoiding the obstacles on the course.
- Judges will rank the entries that complete the course based on shortest adjusted time taken. In the event that a vehicle does not finish the course, the judges will rank the entry based on longest adjusted distance travelled. Adjusted time and distance are the net scores given by judges after taking penalties, incurred from obstacle collisions and boundary crossings, into consideration.
- Each qualified team will have up to two runs in each of three heats.
- \* **Each run per heat will be conducted in two different directions. The two best runs in the two different directions will be averaged to obtain a ranking for the Auto. Nav. test.**

(Figures 5, 6 and 7 displays examples of the Autonomous Navigation course obstacles, waypoints and flags used in AGVC-13)

## Vehicle Control

- Vehicles must be unmanned and autonomous. They must compete based on their ability to perceive the course environment and avoid obstacles.
- Vehicles cannot be remotely controlled by a human operator during competition.
- All computational power, sensing and control equipment must be carried on board the vehicle.
- No base stations contributing to positioning accuracy are allowed.
- Teams are encouraged to map the course and use that information to improve their performance on the course.

## Obstacle Course

- The Auto Nav Test will take place on the Advanced Course (layout similar to that shown in Figure 8 & 9).
- Outer boundaries will be designated by continuous or dashed white lines approximately 7.5 wide, painted on the ground (grass). Track width will vary from 3 meters to 6 meters wide with a turning radius not less than 1.5 meters
- Alternating side-to-side dashes will be 4.5-6.0 meters long, with 3.0-4.5 meters separation.
- The vehicle will then need to average over 1.6 kph for the entire run.
- Competitors should expect natural or artificial inclines with gradients not to exceed 15% and randomly placed obstacles along the course. The course will become more difficult to navigate autonomously as the vehicle progresses.
- Obstacles on the course will consist of various colours (white, orange, brown, green, black, etc.) of construction barrels/drums that are used on roadways and highways. Natural obstacles such as trees or shrubs and manmade obstacles such as light posts

or street signs could also appear on the course. The placement of the obstacles may be randomized from left, right, and centre placements prior to every run.

- There will be a minimum of 1.5 meters clearance, minimum passage width, between the line and the obstacles; i.e., if the obstacle is in the middle of the course then on either side of the obstacle will be 1.8 meters of driving space or if the obstacle is closer to one side of the lane then the other side of the obstacle must have at least 1.8 meters of driving space for the vehicles.
- There will be complex obstacle (barrel) arrangements with switchbacks and centre islands. These will be adjusted for location between runs. Direction of the obstacle course will not change between heats. The course set up for the proposed AGVC will be similar to the 2012 IGVC AutoNav Course shown in (Figure 8). Note Figure 5 shows lane following and Figure 6 waypoint navigation.



Figure 5: Example of Obstacle course (AGVC-13) Figure 6: Example of Waypoint Navigation (circle shown in black)(AGVC-13)

- Alternating red (right) flags and blue (left) flags will be placed on the later part of the course. Flags will have a minimum passage width between them of 1.8 meters; i.e., if the flag is near the edge of the course then between the flag and the line will be 1.8 meters of driving space. Flags are not obstacles and vehicles can touch flags to increase speed and optimized route, vehicles are not allowed to go over flags. The objective is for the vehicle to stay to the left of the red flags and to the right of the blue flags. Flags can be staggered or the vehicle could be driving through a set of flags.
- Auto-Nav Challenge Advanced course (Figure 8 & 9) will contain eight Global Positioning System (GPS) waypoints, one at each entry and exit and three on each side of the navigation no-man's land separated by a fence with three alternating gates. Distance achieved will be totalled by adding straight-line distances between waypoints and added to total distance driving on lined portion of the course. The open space between the navigation waypoints will contain a mix of obstacles which must be avoided while staying with-in the course.

- The exact waypoint locations will be marked on the ground (grass) for use by the judges, but there will be no stand up markers to indicate those positions. Construction barrels, barricades, fences, and certain other obstacles will be located on the course in such positions that they must be circumvented to reach the waypoints. These may be randomly moved between runs.
- The course will be divided into two areas by a fence with a two meter wide opening located somewhere along it (no coordinates are provided). The opening will be randomly relocated along the fence at the start of each run. Waypoints will have two meter circles around them.
- Auto-Nav Course direction on both courses will change for each run of a Heat.
- \* Final placings for the Auto Nav. test will be calculated from the average distance or time taken from the two best runs from the two different course directions.



*Figure 7: Red & Blue flags used in Auto Nav Course*

### **Competition Rules and Procedures**

- Each qualified team will have up to two runs in each of three heats.
- Starting order will be based on order of qualification. Teams will setup on-deck in that order.

Failure to be on-deck will place you at the end of the order for the run and may forfeit your final (second) run in a heat based on heat time completion.

- No team participant is allowed on the course before the team's first run, and only one student team member is allowed on the course during a run. This shall in no case be the faculty / academic advisor.



Figure 8: Advanced Course – Indicative Layout [1]



Figure 9: An Aerial view of the Advanced Course used in AGVC-13

- At the designated on-deck time, the competing team will be asked to prepare their vehicle for an attempt. On-deck teams start in the order they arrive in the starting area unless they give way to another team.
- A Starting Official will call teams to the starting line. The Starting Official's direction is final. The Starting Officials may alter the order to enhance the competition flow of entries; e.g. slower



vehicles may be grouped together to allow successive running of two vehicles on the course simultaneously.

- A team will have one minute in the starting point to prep the vehicle at the starting line and point out to the Competition Judges the buttons to start and stop the vehicle.
- The Competition Judge will start the vehicle by a one touch motion; i.e. pushing a remote control button, hitting the enter key of a keyboard, a left mouse click, lifting the e-stop up, flipping a toggle switch, etc. The Competition Judge will also carry the E-Stop.
- An attempt will be declared valid when the Competition Judge initiates the start signal at the designated competing time. An attempt will continue until one of the following occurs:
  - + The vehicle finishes the course.
  - + The vehicle was E-Stopped by a judge's call.
  - + The team E-Stops the vehicle.
  - + Ten minutes have passed after the vehicle run has started for the Advanced Course. The vehicle has not started after one minute after moving to the start line or at the judges' discretion.
- Time for each heat will be strictly observed.
- Tactile sensors will not be allowed.
- Based on the above allowable run times, if the vehicle has not completed the course in the 10 minute time period, the attempt will be ended by a judge's choice E-stop, with no additional penalty for that run.
- Each vehicle must navigate the course by remaining inside the course boundaries and navigating around course obstacles. Crossing internal lines is not allowed and will be judged an E-Stop end of run with penalty. For the following Traffic Violations, the appropriate ticket will be issued and deducted from the overall distance or time score. Refer to Table 2: Traffic Violation Laws.

	<b>Traffic Violations</b>	<b>Ticket Value</b>	<b>E-Stop</b>	<b>Measurement</b>
<b>1</b>	Hold-up Traffic	End of Run	Yes	>60 secs. to 26.7 meters
<b>2</b>	Leave the Course/Scene	- 3 meters	Yes	Yes
<b>3</b>	Crash/Obstacle Displacement	- 3 meters	Yes	Yes
<b>4</b>	Careless Driving	- 1.5 meters	No	No
<b>5</b>	Sideswipe/Obstacle Touch	- 1.5 meters	No	No
<b>6</b>	Student's Choice E-Stop	- 3 meters	Yes	Yes
<b>7</b>	Judge's Choice E-Stop	0 m	Yes	Yes
<b>8</b>	Blocking Traffic	- 1.5 meters	Yes	Yes
<b>9</b>	Loss of Payload	0 m	Yes	Yes
<b>10</b>	Wrong Side of Flag	-1.5 m	No	No
<b>11</b>	Run over Flag	-3 meters	Yes	Yes
<b>12</b>	Too slow, did not average 1.6	Disqualified	No	No

*Table 2: Traffic Violation Laws [1]*

- **Hold-up traffic:** Must maintain 1.6 kph, there will be a speed check at 26.7 meter mark of the course, will result in end of run with time recorded.
- **Leave the scene\course:** All portions of the vehicle cross the boundary. The overall distance will be measured from the starting line to the furthest point where the final part of the vehicle crossed the boundary outside edge.
- **Crash:** The overall distance will be measured from the starting line to the collision point with the obstacle.
- **Careless Driving:** Crossing the boundary while at least some part of the vehicle remains in bounds.
- **Student E-Stop:** Student e-stop is used if the team feels that there may be damaged caused to their vehicle or they know that it is stuck and want to end their time.
- **Judge E-Stop:** The overall distance will be measured from the starting line to the front of the vehicle or where the final/furthest remaining part of vehicle if stopped, crossed the boundary outside edge.
- **Obstacle Displacement:** Defined as displacing permanently the obstacle from its original position. Slightly rocking/Tilting an obstacle with no permanent displacement is not considered obstacle displacement. An obstacle that rocks or tilts significantly but with no displacement will still be considered an end of run. Judges calls are final.
- **Blocking Traffic:** Vehicles stopping on course for over one minute will be E-Stopped and measured.
- **Loss of Payload:** If the payload falls of the vehicle the run will be ended.
- **Wrong Side of Flag:** Vehicles must remain on the left side of red flags and the right side of blue flags.
- **Run over Flag:** Vehicles drive over the top of a red or blue flag will results in End of Run.
- **Too Slow:** If the vehicle does not maintain 1.6 kph minimum average speed limit throughout the course this run is disqualified.

## Judging

- Designated competition judges will determine the official times, distances and ticket deductions of each entry. At the end of the competition, those vehicles crossing the finish line will be scored on the time taken to complete the course minus any ticket deductions. Ticket values will be assessed in seconds (0.3 meters = one second) if the vehicle completes the course within the run time.
- The team with the adjusted shortest time will be declared the winner.
- In the event that no vehicle completes the course, the score will be based on the distance travelled by the vehicle minus the ticket deductions. The team with the adjusted longest distance will be declared the winner.
- For standard award money consideration, entry must exhibit sufficient degree of autonomous mobility by passing the money barrel. The money barrel location is determined by the judges during the final/actual course layout. If a tie is declared between entries, the award money will be split between them.
- If your vehicle is overtaken by a faster vehicle you will be commanded to stop and your time will be recorded and allowed to be restarted with remaining time after the faster vehicle passes. Total distance will be assessed at the 10 minute mark.

### **Grounds for Disqualifications**

- Judges will disqualify any vehicle which appears to be a safety hazard or violate the safety requirements during the competition.
- Intentional interference with another competitor's vehicle and/or data link will result in disqualification of the offending contestant's entry.
- Damaging the course or deliberate movement of the obstacles or running over the obstacles may result in disqualification.
- Actions designed to damage or destroy an opponent's vehicle are not in the spirit of the competition and will result in disqualification of the offending contestant's entry.

# Appendix C: Design Competition

## Objective

Although the ability of the vehicles to negotiate the competition courses is the ultimate measure of product quality, the officials are also interested in the design strategy and process that engineering teams follow to produce their vehicles. Design judging will be by a panel of expert judges and will be conducted separate from and without regard to vehicle performance on the test course. Judging will be based on a written report, an oral presentation and examination of the vehicle.

Design innovation will be given special attention by the judges. Innovation is defined as a technology (hardware or software) that has not ever been used by this or any other vehicle in this competition. The innovation needs to be documented, as an innovation, clearly in the written report and emphasized in the oral presentation.

## Written Report

- The report should not exceed 15 A4-sized pages, including graphic material and all appendices, but not including the title page. Reports will lose 5 points in scoring for each page over 15. Font size is to be 11 point and lines spacing 1.5.
- Each vehicle must have a distinct and complete report of its own (a report cannot cover more than one vehicle). Participants are required to submit an electronic copy in PDF format on a CD; failure to submit will result in disqualification.
- All reports must include a statement signed by the faculty advisor certifying that the design and engineering of the vehicle (original or changes) by the current student team has been significant and equivalent to what might be awarded credit in a senior design course.
- Written reports arriving after the specified due date will lose 10 points in scoring for each business day late.
- Teams are encouraged to submit reports even several weeks early to avoid the last minute rush of preparing vehicles for the competition, and there will be no penalty for last minute changes in the vehicle from the design reported.
- The paper should present the conceptual design of the vehicle and its components. Especially important to highlight are any unique innovative aspects of the design and the intelligence aspects of the vehicle, including descriptions of some of the following as shown in Table 3:

electronics	design planning process
electrical system	signal processing
actuators	plan for path following
software strategy	(both solid & dashed lines)
sensors	plan for control decisions
computer	system integration plan
mappin	high speed operations

*Table 3: Aspects of Design in the report*

- Design of the lane following and obstacle detection/avoidance systems must be specifically described along with how the vehicle uses mapping techniques to perceive and navigate through its environment. Describe how the system uses GPS for waypoint navigation and localization.
- Components acquired ready-made must be identified, but their internal components need not be described in detail. The steps followed during the design process should be described along with any use of Computer-Aided Design (CAD).
- How considerations of safety, reliability, and durability were addressed in the design process should be specifically described, as well as problems encountered in the design process and how they were overcome. The analysis leading to the predicted performance of the vehicle should be documented, specifically:
  - Speed
  - Ramp climbing ability
  - Reaction times
  - Battery life
  - Distance at which obstacles are detected
  - How the vehicle deals with complex obstacles including switchbacks and centre islands dead ends, traps, and potholes
  - Accuracy of arrival at navigation waypoints
  - Comparison of these predictions with actual trial data is desirable.
- Although cost itself is not a factor in judging (these are considered research vehicles), the report should include a cost estimate (not counting student labour) for the final product if it were to be duplicated. A breakdown of the cost by component is helpful.
- The team organization and the names of all members of the design team, with academic department and class, should be included along with an estimate of the project's total number of person-hours expended. Scoring for written report shown in Table 4.
- Judges will score the written reports as follows:

Criteria	Maximum Points
1. Conduct of the design process and team organization	50
2. Completeness of the documentation	50
3. Quality of documentation (English, grammar, and style)	50
4. Effective innovation represented in the design (as described)	150
5. Description of mapping technique	100
6. Description of electronic design	100
7. Description of software strategy	150
8. Description of systems integration Descriptions to include: lane following, obstacle detection/avoidance, and waypoint navigation (GPS or other)	150
9. Efficient use of power and materials	50
10. Attention given to safety, reliability, and durability	50
<b>Total</b>	<b>900</b>

*Table 4: Judges score for written report*

## Oral Presentation

- The technical presentation should relate the highlights of the written report described above and include any updates of the design since the written report.
- Audio or video tape presentations of the text are not allowed, but graphic aids may be presented by video, slide projection, computer projection, overhead transparencies, or easel charts.
- The presentation must be made by one or more student members of the team to the judges and other interested members of the audience and should last not more than 20 minutes. A penalty of 5 points will be assessed for each minute or fraction thereof over 21 minutes.
- After the presentation, judges only may ask questions for up to 5 minutes. The audience should be considered as a senior management group of generally knowledgeable engineers upon whom the project is dependent for funding and the team is dependent for their employment. Scoring for oral presentation will be as shown in Table 5:
- **Judges will score the oral presentations as follows:**

Criteria	Maximum Points
1. Clear and understandable explanation of the innovations	50
2. Logical organization of the talk	25
3. Effective use of graphic aids	25
4. Articulation	20
5. Demonstrated simulation of vehicle control in performance	10
6. Response to questions	10
7. Salesmanship	10
<b>Total</b>	<b>150</b>

*Table 5: Judges score for oral presentation*

- Effective use of graphic aids includes not blocking the view of the screen by the presenter and simple enough graphics that are large enough to read (block diagrams rather than detailed circuit diagrams).
- Articulation refers to the clarity and loudness of speaking.
- Response to questions means short answers that address only the question.
- Salesmanship refers to the enthusiasm and pride exhibited (why this vehicle is the best).
- During the oral presentation, following the question period and the examination of the vehicle, team members sitting in the audience may participate to assist the oral presenters, but the faculty advisor will not be allowed to participate in this part of the design competition.

## Examination of Vehicle

- The vehicle will be examined by the judges preferably immediately after the oral presentation or at another convenient time during the competition. Software is not included in this part of judging.
- Judges will score the vehicle examinations as follows (Table 6):

Criteria	Maximum Points
----------	----------------

1. Packaging neatness, efficient use of space	20
2. Serviceability	20
3. Ruggedness	20
4. Safety	20
5. Degree of original content in the vehicle (as opposed to ready-	50
6. Style (overall appearance)	20
<b>Total</b>	<b>150</b>

*Table 6: Vehicle Examination score*

### **Final Design Scoring**

The number of points awarded by the individual judges will be averaged for each of the 23 judging areas above, and these results will be offered to each participating team for their edification. The total of the average scores over all 23 areas (max 1200) will be used to determine the ranking.

### **Awards**

The final awards (trophies + cash awards) will be awarded to top 3 teams that performed the best overall in two competitions. Awards certificates will be presented to teams ranked in fourth and fifth place. For each competition points will be awarded to each team according to Table 7 below:

Ranking	Auto-Nav Test		Design Test
	Passed Money Barrel	Short of Money Barrel	
First Place	50	25	25
Second Place	40	20	20
Third Place	30	15	15
Fourth Place	20	10	10
Fifth Place	10	5	5

*Table 7: Vehicle Ranking score*

# Autonomic Ground Vehicle Competition

## A DSTO/ Deakin University Initiative

4<sup>th</sup> – 8<sup>th</sup> December 2014

Deakin University, Waurm Ponds Campus, Geelong, VIC

### - INTENT TO COMPETE FORM -

The Autonomous Ground Robotic Competition is jointly organised by the Defence Science and Technology Organisation (DSTO) and CISR, Deakin University, Geelong. The event will be held at Waurm Ponds Campus, Deakin University, Geelong and is open to Tertiary Students from Australia. AGVC is an annual event intended to stimulate robotics related research in Australian Tertiary institutions.

Please forward your completed INTENT TO COMPETE FORM to [agvc@deakin.edu.au](mailto:agvc@deakin.edu.au) prior to the closing date of **31 July 2014**.

DETAILS					
Team Name	Team UWA				
Name of Institution	The University of Western Australia				
TEAM CAPTAIN CONTACT DETAILS					
First Name	Garrick	Last Name	Paskos		
TEAM MEMBERS (first name, last name)					
#1	Qing Zhou	#2	Ross Green		
#3	Stuart Speidel	#4			
#5		#6			
PREFERRED MAILING ADDRESS (for printed correspondence)					
Street Address	2/141 Seventh Ave, Inglewood		Suburb/City	Perth	
State	WA	Postcode	6052	Phone	
Mobile	0420540750		Fax		
Email	10432185@student.uwa.edu.au				
ALTERNATE TEAM CONTACT					
Name	Stuart Speidel				
Phone	0432055258				
Email	sjmspeidel@gmail.com				

For detailed information please visit our website at : [www.AGVC.net.au](http://www.AGVC.net.au)

POC for Technical Enquiries:

**Dr Arthur Filippidis**  
**Land Operations Division**  
**DSTO, Edinburgh, SA 5111**

**Ph: (08)7389 4849**  
**Mob: 0410873211**  
**arthur.filippidis@dsto.defence.gov.au**

**Dr Mick Fielding**  
**CISR, Deakin University**  
**Waurm Ponds, Geelong, Vic 316**

**Ph: 03 5227 2807**  
**mick.fielding@deakin.edu.au**



Australian Government  
 Department of Defence  
 Defence Science and  
 Technology Organisation



# Team UWA 2014 AGVC Entry Form

## Table of Contents

1. Team organization & contact details .....	2
2. Abstract .....	2
3. Introduction .....	2
3.1. Statement of the problem as specified in the competition rules .....	2
3.2. Conceptual solution proposed .....	3
3.2.1. Localization .....	3
3.2.2. Navigation and obstacle avoidance .....	3
3.2.3. Goal planning .....	3
3.3. Overall systems architecture .....	3
3.4. Work breakdown and milestones .....	3
4. Ground vehicle component & systems .....	4
4.1. Hardware .....	4
4.2. Software .....	4
5. AGV system autonomy & coordination strategy .....	4
6. Operational approach/missions operations strategy .....	5
7. Risk reduction strategy .....	5
7.1. EMI/RFI & electrical .....	5
7.2. Vibration & physical .....	5
7.3. Modelling & Simulation .....	5
7.4. Safety, E-Stop, Freeze & lost-link .....	5
7.5. Communications architecture .....	5
7.6. Test Plan .....	5
8. Summary .....	6
9. Signature of Team Leader & Date .....	6

## 1. Team organization & contact details

Name	Role	Email	Phone Number
Professor Thomas Bräunl	Supervisor	thomas.braunl@uwa.edu.au	0864881763
Garrick Paskos	Team Leader	10432185@student.uwa.edu.au	0420540750
Qing Zhou	Participant	21078355@student.uwa.edu.au	0451129066
Ross Green	Participant	20247459@student.uwa.edu.au	0424035787

All postal mail has been set to be addressed to Team Leader, Garrick Paskos at 2/141 Seventh Ave Inglewood WA 6052.

## 2. Abstract

In this document we discuss the UWA Robotics entry to the 2014 AGVC competition. The problem presented in this competition involves creating a robot that autonomously navigates an outdoor obstacle course marked out by painted white lines. This is further complicated by the inclusion of relatively large distances compared to the size of the robot, coloured flags and GPS waypoints. The robot hardware includes a rigid commercially available robot chassis, a retail Dell laptop PC, a high end laser scanners, a high end IMU, multiple cameras and a GPS receiver. The open source ROS system was adopted as it provides a fully functional robot operating system and comes with a large number of well-maintained existing packages. Three new packages were also created to handle the GPS navigation, lane detection and overall system coordination. The final system is tested in a mockup of the expected course and from the multiple successful tests we hope to successfully compete in the 2014 competition.

## 3. Introduction

### 3.1. Statement of the problem as specified in the competition rules

1. Pass an initial qualification test that ensures an entry adheres to a set technical specification, including:

- Dimensions.
- Propulsion (electric powered only).
- Emergency shutdown routines (the existence of wired and wireless “stop” buttons).
- Autonomous mode indication (via flashing lights).

2. Pass a further qualification test that requires the vehicle autonomously navigate a simplistic, known course whilst demonstrating that it is able to:

- a. Only travel within a speed range of 1.6km/h and 16km/h.
- b. Detect and follow lanes (shown as the white lines in Fig. 1.1).
- c. Detect and avoid obstacles (barrels shown as dots in Fig. 1.1).
- d. Navigate to a given GPS waypoint whilst staying within lanes and avoiding obstacles.

3. If successful in qualifying, the vehicle will be required to navigate a more complicated course and score points by completing several tasks and this will mainly decide the winner of the competition. This is called the “AutoNav” challenge.

4. Submit a report and deliver an oral presentation based on design decisions made.

## 3.2. Conceptual solution proposed

### 3.2.1. Localization

We are planning to localize either using a fusion of wheel odometry, IMU data and GPS data or this with a long range laser scanner as the input to an implementation of OpenSlam Gmapping.

### 3.2.2. Navigation and obstacle avoidance

We plan to use an existing and free software system that can reliably navigate to points within a frame of reference provided that localization in that frame of reference is sufficiently accurate.

This software adds in obstacles detected and outputted as laser scan data. This is directly the case for “barrel” obstacles. For “painted line” obstacles we are writing software to output detected lines as laser scan data.

### 3.2.3. Goal planning

Ideally, especially in the later parts of the Autonav course, it would be good to have an algorithm which could look at the current or saved map and determine the optimal visitation to GPS waypoints as to minimize completion time.

## 3.3. Overall systems architecture

Figure 1.2.1 overleaf abstractly describes the overall systems architecture.

## 3.4. Work breakdown and milestones

A Gantt chart is shown on page 5 indicating division of labor and milestones.

Competition deadline reflect estimates prior to availability of newer information and have since been updated.

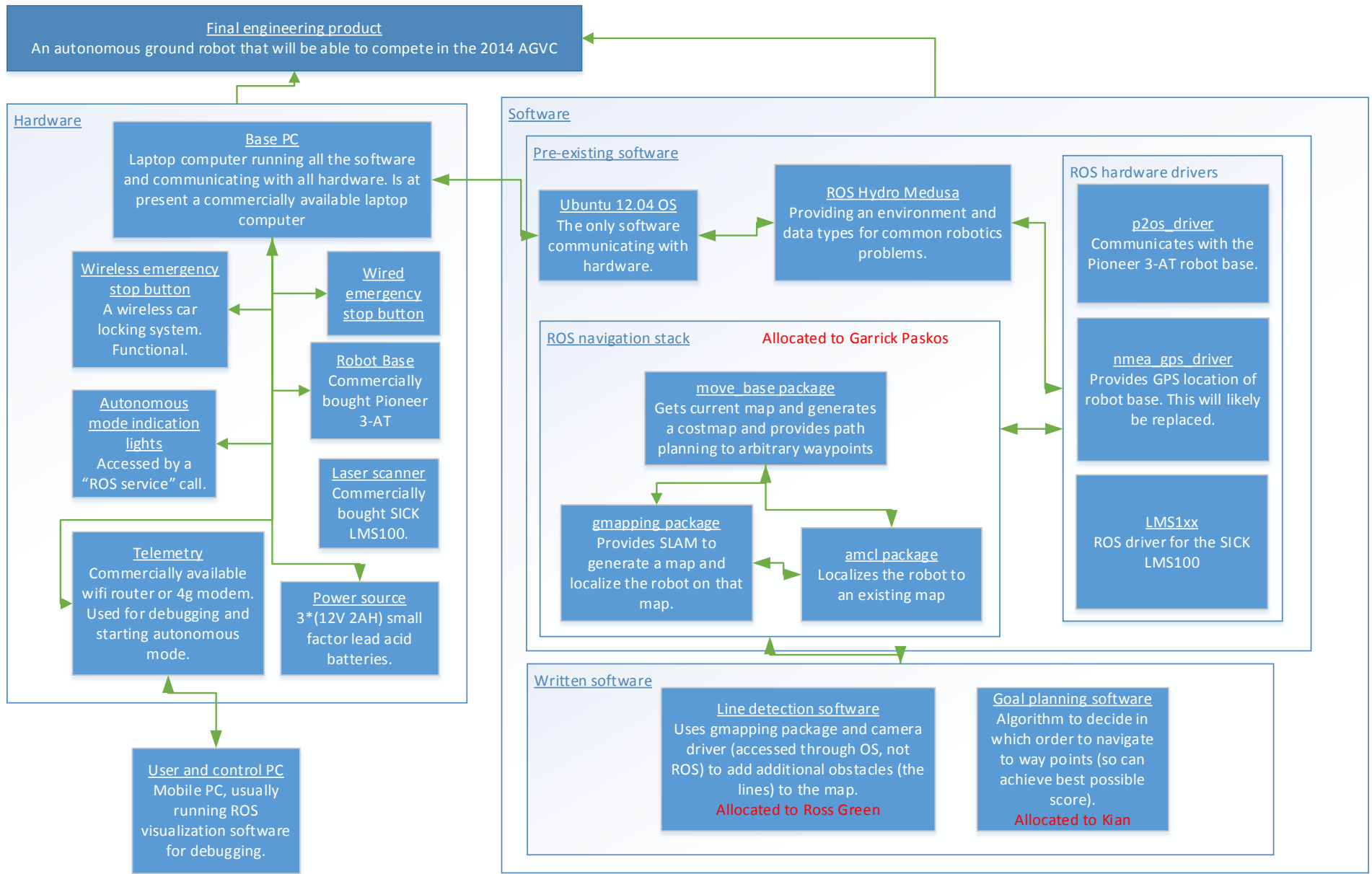
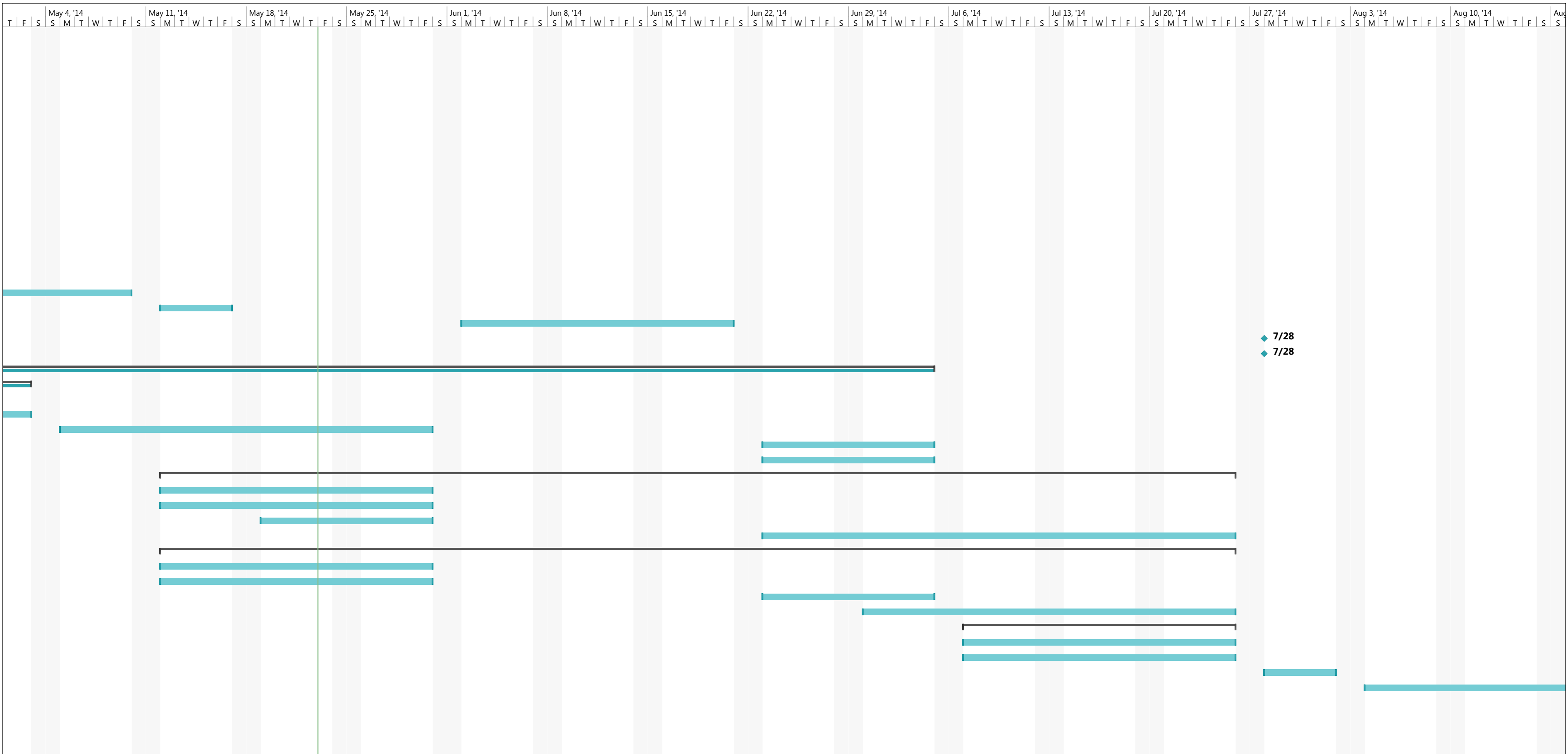


Fig. 1.2.1: Various subsystems employed in the AGV.

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Feb 23, '14	Mar 2, '14	Mar 9, '14	Mar 16, '14	Mar 23, '14	Mar 30, '14	Apr 6, '14	Apr 13, '14	Apr 20, '14	Apr 27, '14
							S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S
1	★	<b>Initialization</b>	<b>6 days</b>	<b>Mon 2/24/14</b>	<b>Mon 3/3/14</b>		[Task bar]									
2	★	Familiarization with AGVC rules	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
3	★	Familiarization with FYP	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
4	★	Installation of Ubuntu & ROS on personal	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
5	★	Familiarization with Ubuntu and Linux	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
6	★	Acquisition of previous year's resources	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
7	★	<b>Familiarization with ROS</b>	<b>10 days</b>	<b>Mon 3/3/14</b>	<b>Fri 3/14/14</b>		[Task bar]									
8	★	ROS wiki basic tutorials	5 days	Mon 3/3/14	Fri 3/7/14		[Task bar]									
9	★	ROS wiki intermediate to advanced tutorials	5 days	Mon 3/10/14	Fri 3/14/14		[Task bar]									
10	★	Literature review for Adrian	0 days	Wed 3/5/14	Wed 3/5/14		[Task bar]									
11	★	<b>ROS drivers</b>	<b>5 days</b>	<b>Mon 3/17/14</b>	<b>Fri 3/21/14</b>		[Task bar]									
12	★	Installing and testing precompiled drivers	5 days	Mon 3/17/14	Fri 3/21/14		[Task bar]									
13	★	Compiling and testing laser scanner drivers	5 days	Mon 3/17/14	Fri 3/21/14		[Task bar]									
14	★	Initialization of ROS workspace and practice	5 days	Mon 3/24/14	Fri 3/28/14		[Task bar]									
15	★	<b>Navigation stack</b>	<b>15 days</b>	<b>Mon 3/31/14</b>	<b>Fri 4/18/14</b>		[Task bar]									
16	★	Installation and testing of required packages	5 days	Mon 3/31/14	Fri 4/4/14		[Task bar]									
17	★	Installation, configuration and testing of	15 days	Mon 4/7/14	Fri 4/25/14		[Task bar]									
18	★	Tutorials with stuart concerning line detection	10 days	Mon 4/28/14	Fri 5/9/14		[Task bar]									
19	★	Establishment of individual tasks	5 days	Mon 5/12/14	Fri 5/16/14		[Task bar]									
20	★	Exam break	15 days	Mon 6/2/14	Fri 6/20/14		[Task bar]									
21	★	Rudimental competition ready entry to be	0 days	Mon 7/28/14	Mon 7/28/14		[Task bar]									
22	★	Semester 2 starts	0 days	Mon 7/28/14	Mon 7/28/14		[Task bar]									
23	★	<b>Garrick's individual tasks</b>	<b>65 days</b>	<b>Mon 4/7/14</b>	<b>Fri 7/4/14</b>		[Task bar]									
24	★	<b>Testing of ROS's distributed computing</b>	<b>20 days</b>	<b>Mon 4/7/14</b>	<b>Fri 5/2/14</b>		[Task bar]									
25	★	Over wifi	10 days	Mon 4/7/14	Fri 4/18/14		[Task bar]									
26	★	Over 4g	10 days	Mon 4/21/14	Fri 5/2/14		[Task bar]									
27	★	Tweaking of navigation stack	20 days	Mon 5/5/14	Fri 5/30/14		[Task bar]									
28	★	Sending base to GPS point functionality	10 days	Mon 6/23/14	Fri 7/4/14		[Task bar]									
29	★	Integration of new GPS module (may involve	10 days	Mon 6/23/14	Fri 7/4/14		[Task bar]									
30	☛	<b>Ross's individual tasks</b>	<b>55 days</b>	<b>Mon 5/12/14</b>	<b>Fri 7/25/14</b>		[Task bar]									
31	★	Study of computer vision theory	15 days	Mon 5/12/14	Fri 5/30/14		[Task bar]									
32	★	Practice with C++ and openCV library	15 days	Mon 5/12/14	Fri 5/30/14		[Task bar]									
33	★	Experiment with line detection algorithms	10 days	Mon 5/19/14	Fri 5/30/14		[Task bar]									
34	★	Lines-to-map functionality	25 days	Mon 6/23/14	Fri 7/25/14		[Task bar]									
35	☛	<b>Qing's individual tasks</b>	<b>55 days</b>	<b>Mon 5/12/14</b>	<b>Fri 7/25/14</b>		[Task bar]									
36	★	Study of goal planning algorithms	15 days	Mon 5/12/14	Fri 5/30/14		[Task bar]									
37	★	Practice with C++	15 days	Mon 5/12/14	Fri 5/30/14		[Task bar]									
38	★	Rudimental goal planning functionality	10 days	Mon 6/23/14	Fri 7/4/14		[Task bar]									
39	★	More advanced goal planning functionality	20 days	Mon 6/30/14	Fri 7/25/14		[Task bar]									
40	☛	<b>Combined tasks</b>	<b>15 days</b>	<b>Mon 7/7/14</b>	<b>Fri 7/25/14</b>		[Task bar]									
41	★	Navigation <-> line detection (Garrick &	15 days	Mon 7/7/14	Fri 7/25/14		[Task bar]									
42	★	Navigation <-> goal planning (Garrick &	15 days	Mon 7/7/14	Fri 7/25/14		[Task bar]									
43	★	Emergency stop and flashing light functionality	5 days	Mon 7/28/14	Fri 8/1/14		[Task bar]									
44	★	Final testing and improment and extra work if	80 days	Mon 8/4/14	Fri 11/21/14		[Task bar]									
45	★	Project seminar	5 days	Mon 9/29/14	Fri 10/3/14		[Task bar]									
46	★	Final Report Submission	0 days	Mon 10/27/14	Mon 10/27/14		[Task bar]									
47	★	Preperation for competition report	14 days	Mon 11/10/14	Thu 11/27/14		[Task bar]									
48	★	Expected competition date	2 days	Fri 11/28/14	Mon 12/1/14		[Task bar]									

◆ 3/5

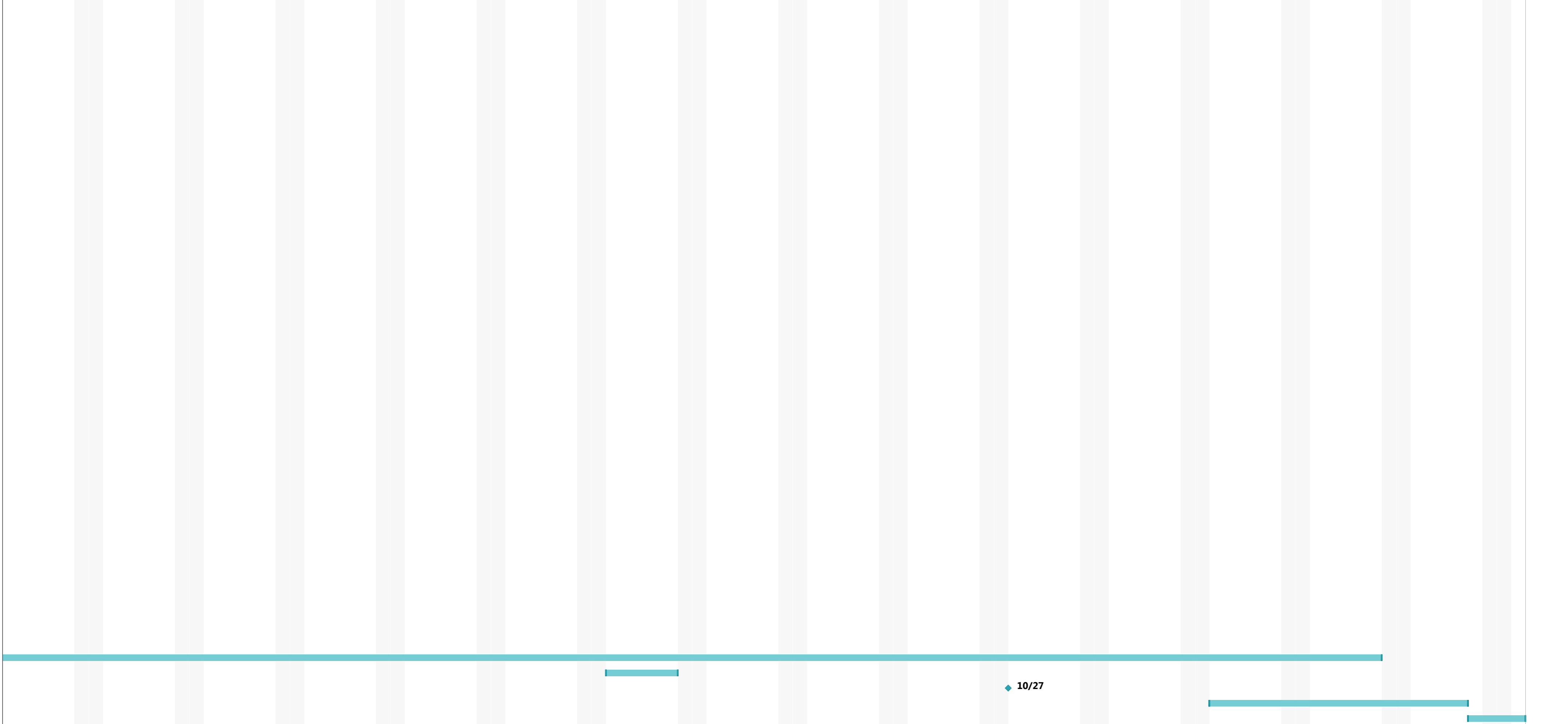
Project: baseline_sched Date: Fri 5/23/14	Task		Summary		Inactive Milestone		Duration-only		Start-only		External Milestone		Manual Progress	
	Split		Project Summary		Inactive Summary		Manual Summary Rollup		Finish-only		Deadline			
	Milestone		Inactive Task		Manual Task		Manual Summary		External Tasks		Progress			



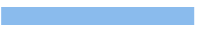

















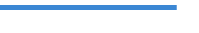
Project: baseline\_sched  
Date: Fri 5/23/14

Task		Summary		Inactive Milestone		Duration-only		Start-only		External Milestone		Manual Progress	
Split		Project Summary		Inactive Summary		Manual Summary Rollup		Finish-only		Deadline			
Milestone		Inactive Task		Manual Task		Manual Summary		External Tasks		Progress			

Jul 17, '14 | Aug 24, '14 | Aug 31, '14 | Sep 7, '14 | Sep 14, '14 | Sep 21, '14 | Sep 28, '14 | Oct 5, '14 | Oct 12, '14 | Oct 19, '14 | Oct 26, '14 | Nov 2, '14 | Nov 9, '14 | Nov 16, '14 | Nov 23, '14 | Nov 30, '14



Project: baseline\_sched  
Date: Fri 5/23/14

Task		Summary		Inactive Milestone		Duration-only		Start-only		External Milestone		Manual Progress	
Split		Project Summary		Inactive Summary		Manual Summary Rollup		Finish-only		Deadline			
Milestone		Inactive Task		Manual Task		Manual Summary		External Tasks		Progress			

## 4. Ground vehicle component & systems

### 4.1. Hardware

- Pioneer 3-AT chassis used as the robot chassis. This includes three hot swappable 12V lead acid batteries and a rigid two motor, four wheel differential drive system
- Dell N301Z laptop computer (this may be upgraded nearing competition time)
- SICK LMS 111. A laser scanner used for mapping the environment
- 1 Microsoft LifeCam HD-3000 Webcam's
- QStarz BT-Q818X. NMEA compliant GPS receiver with DGPS capability
- XSenS MTI IMU. Inertial Measurement unit containing a three axis rate gyroscope, a three axis accelerometer and a magnetometer
- Net gear aircard 4g wifi modem router with 2\*2.4 GHz antennas
- Logitech rumblepad and associated USB Bluetooth receiver
- Arduino Duemilanove to control the LED warning lights

When all three batteries are loaded into the robot and without the 10kg payload it weighs 16.9kg. We have since included 3 additional batteries for extended running time, increasing the weight by 7.65kg to a total 24.55kg.

### 4.2. Software

As with UWA's 2013 AGVC entry, we are to continue using the ROS robotics middleware.

A list of currently employed software subsystems are shown below:

- `gmapping_slam`: This packages uses the LMS111 to produce a map of the environment and localises the robot within the map
- The navigation stack: The navigation stack contains a number of packages whose combined purpose is to allow the robot to receive and navigate to goals. This stack is described in detail in section 3.3
- `xsens_driver`, `joy`, `lms100` and `nmea_gps_driver`: Driver for the Xsens IMU, Logitech joystick, SICK LMS111 and GPS respectively
- new joystick driver
- `sound_play`: Plays sounds through the speakers and supports text to speech
- `robot_pose_ekf`: Uses an EKF to fuse the Hector SLAM based position, the IMU orientation and the GPS fix to produce a more accurate estimation of the robots position
- `agvc_coordinator`: Starts up the system and provides high level control
- `goal_provider`: Sends the GPS waypoints to `move_base` as movement goals and monitors the status of the current goal

## 5. AGV system autonomy & coordination strategy

The software subsystems selected above provide the functionality for an "autonomous robot".

ROS, by providing set interfaces to hardware and data streams, facilitates the combination of these software subsystems with relative ease.



## 6. Operational approach/missions operations strategy

Our focus is first to create a robust entry with only a rudimentary GPS visitation order algorithm.

We will then implement the goal planning algorithm used in last year's completion and improve on it. This is shown below.

If time permits we may completely rewrite this goal planning algorithm.

## 7. Risk reduction strategy

### 7.1. EMI/RFI & electrical

We are using legal wireless digital communications hardware (Netgear 782S), whether through wifi or 4g.

We are using 12V lead acid batteries incapable of providing a lethal level of electrical current.

### 7.2. Vibration & physical

All team members have been versed in the appropriate OHS routines. We have access to a flat top trolley for transportation of the robot base and batteries. Two people are always required for lifting the robot base on and off of the trolley, and in the correct procedure.

### 7.3. Modelling & Simulation

Significantly changed code is tested in the Gazebo Simulator before being tested on the UGV.

### 7.4. Safety, E-Stop, Freeze & lost-link

Hardware E-Stop measures have been unchanged from last year and tested periodically.

In the case of freezes and lost-links, the procedure is the shutdown the robot via its E-Stop button.

### 7.5. Communications architecture

Meetings are held every Wednesday where general concerns of the direction of the project are discussed, including safety.

### 7.6. Test Plan

Changed code is usually tested in the simulator before on the vehicle.

We regularly test in a designated area of the university off limits to other students.

## 8. Summary

The “Autonomous Ground Vehicle Competition” (AGVC) is an annual Australian interuniversity competition intended to “stimulate robotics related research in Australian tertiary institutions”. The competition is facilitated by the Defence Science and Technology Organisation (DSTO).

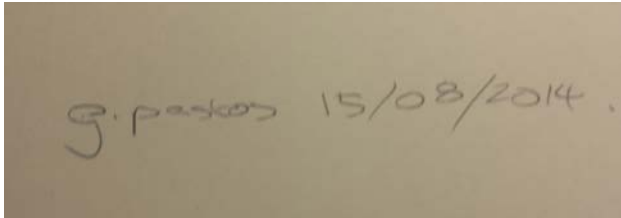
The 2nd annual AGVC is to take place during the 4-8 December 2014 and will feature the similar competition rules as the 2013 competition in consequence of the DSTO deeming all entries into the 2013 competition as unsatisfactory. This will be UWA’s second entry into the competition.

We hope to improve on last year’s entry by producing a more “robust” AGV – one that is more thoroughly tested and verified in similar, reproduced contest conditions.

Throughout our extension of the AGV, we will adhere to a strict and sufficient safety plan, aiming to avoid any injuries to ourselves or spectators.

On behalf of the University of Western Australia’s robotics department, we would like to thank Deakin University and the DSTO for the opportunity of participating in the 2014 AGVC.

## 9. Signature of Team Leader & Date

A photograph of a piece of paper with a handwritten signature and date. The signature is "G. Paskos" and the date is "15/08/2014".

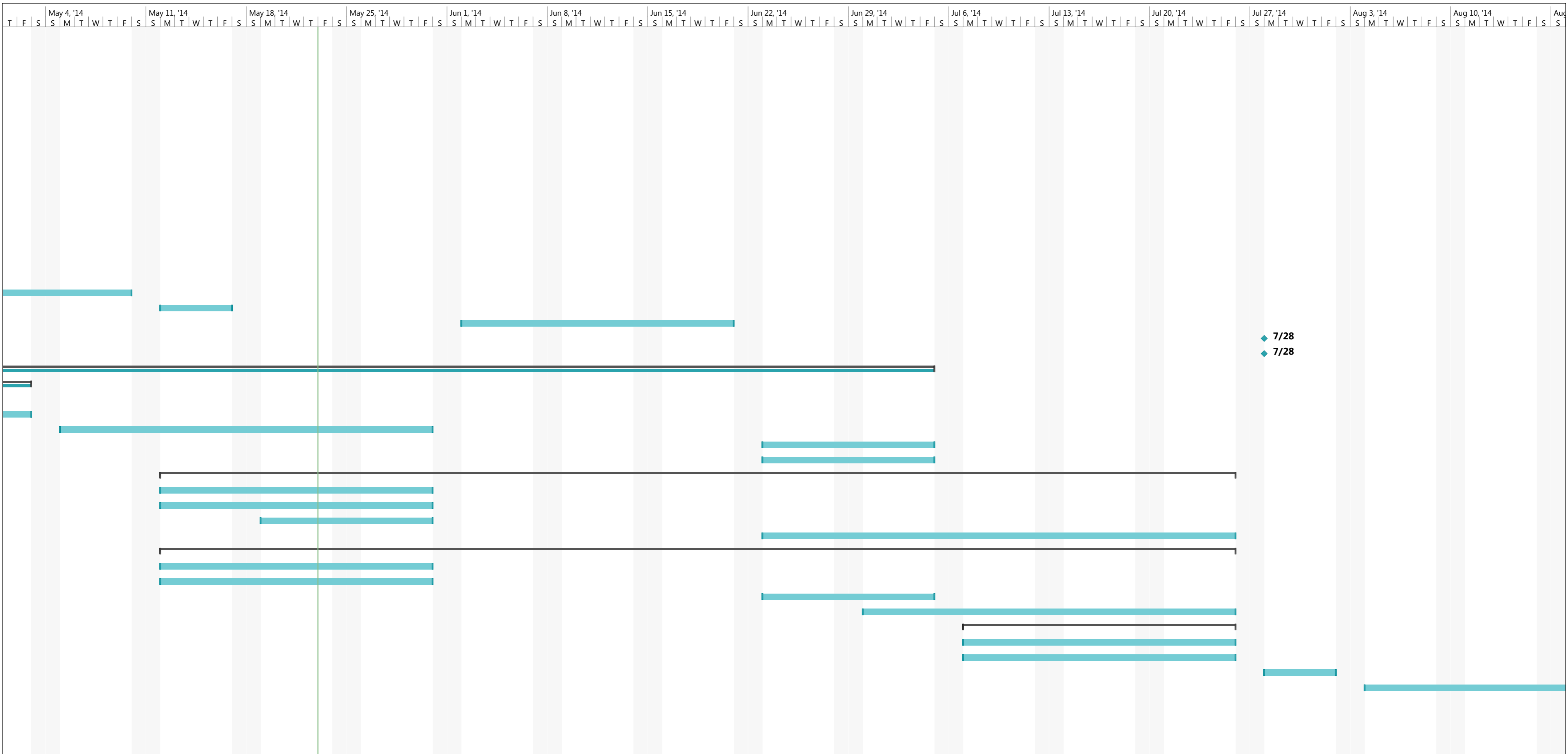
Garrick Paskos 15/08/2014

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Feb 23, '14	Mar 2, '14	Mar 9, '14	Mar 16, '14	Mar 23, '14	Mar 30, '14	Apr 6, '14	Apr 13, '14	Apr 20, '14	Apr 27, '14
							S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S	S M T W T F S
1	★	<b>Initialization</b>	<b>6 days</b>	<b>Mon 2/24/14</b>	<b>Mon 3/3/14</b>		[Task bar]									
2	★	Familiarization with AGVC rules	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
3	★	Familiarization with FYP	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
4	★	Installation of Ubuntu & ROS on personal	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
5	★	Familiarization with Ubuntu and Linux	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
6	★	Acquisition of previous year's resources	5 days	Mon 2/24/14	Fri 2/28/14		[Task bar]									
7	★	<b>Familiarization with ROS</b>	<b>10 days</b>	<b>Mon 3/3/14</b>	<b>Fri 3/14/14</b>		[Task bar]									
8	★	ROS wiki basic tutorials	5 days	Mon 3/3/14	Fri 3/7/14		[Task bar]									
9	★	ROS wiki intermediate to advanced tutorials	5 days	Mon 3/10/14	Fri 3/14/14		[Task bar]									
10	★	Literature review for Adrian	0 days	Wed 3/5/14	Wed 3/5/14		[Task bar]									
11	★	<b>ROS drivers</b>	<b>5 days</b>	<b>Mon 3/17/14</b>	<b>Fri 3/21/14</b>		[Task bar]									
12	★	Installing and testing precompiled drivers	5 days	Mon 3/17/14	Fri 3/21/14		[Task bar]									
13	★	Compiling and testing laser scanner drivers	5 days	Mon 3/17/14	Fri 3/21/14		[Task bar]									
14	★	Initialization of ROS workspace and practice	5 days	Mon 3/24/14	Fri 3/28/14		[Task bar]									
15	★	<b>Navigation stack</b>	<b>15 days</b>	<b>Mon 3/31/14</b>	<b>Fri 4/18/14</b>		[Task bar]									
16	★	Installation and testing of required packages	5 days	Mon 3/31/14	Fri 4/4/14		[Task bar]									
17	★	Installation, configuration and testing of	15 days	Mon 4/7/14	Fri 4/25/14		[Task bar]									
18	★	Tutorials with stuart concerning line detection	10 days	Mon 4/28/14	Fri 5/9/14		[Task bar]									
19	★	Establishment of individual tasks	5 days	Mon 5/12/14	Fri 5/16/14		[Task bar]									
20	★	Exam break	15 days	Mon 6/2/14	Fri 6/20/14		[Task bar]									
21	★	Rudimental competition ready entry to be	0 days	Mon 7/28/14	Mon 7/28/14		[Task bar]									
22	★	Semester 2 starts	0 days	Mon 7/28/14	Mon 7/28/14		[Task bar]									
23	★	<b>Garrick's individual tasks</b>	<b>65 days</b>	<b>Mon 4/7/14</b>	<b>Fri 7/4/14</b>		[Task bar]									
24	★	<b>Testing of ROS's distributed computing</b>	<b>20 days</b>	<b>Mon 4/7/14</b>	<b>Fri 5/2/14</b>		[Task bar]									
25	★	Over wifi	10 days	Mon 4/7/14	Fri 4/18/14		[Task bar]									
26	★	Over 4g	10 days	Mon 4/21/14	Fri 5/2/14		[Task bar]									
27	★	Tweaking of navigation stack	20 days	Mon 5/5/14	Fri 5/30/14		[Task bar]									
28	★	Sending base to GPS point functionality	10 days	Mon 6/23/14	Fri 7/4/14		[Task bar]									
29	★	Integration of new GPS module (may involve	10 days	Mon 6/23/14	Fri 7/4/14		[Task bar]									
30	☛	<b>Ross's individual tasks</b>	<b>55 days</b>	<b>Mon 5/12/14</b>	<b>Fri 7/25/14</b>		[Task bar]									
31	★	Study of computer vision theory	15 days	Mon 5/12/14	Fri 5/30/14		[Task bar]									
32	★	Practice with C++ and openCV library	15 days	Mon 5/12/14	Fri 5/30/14		[Task bar]									
33	★	Experiment with line detection algorithms	10 days	Mon 5/19/14	Fri 5/30/14		[Task bar]									
34	★	Lines-to-map functionality	25 days	Mon 6/23/14	Fri 7/25/14		[Task bar]									
35	☛	<b>Qing's individual tasks</b>	<b>55 days</b>	<b>Mon 5/12/14</b>	<b>Fri 7/25/14</b>		[Task bar]									
36	★	Study of goal planning algorithms	15 days	Mon 5/12/14	Fri 5/30/14		[Task bar]									
37	★	Practice with C++	15 days	Mon 5/12/14	Fri 5/30/14		[Task bar]									
38	★	Rudimental goal planning functionality	10 days	Mon 6/23/14	Fri 7/4/14		[Task bar]									
39	★	More advanced goal planning functionality	20 days	Mon 6/30/14	Fri 7/25/14		[Task bar]									
40	☛	<b>Combined tasks</b>	<b>15 days</b>	<b>Mon 7/7/14</b>	<b>Fri 7/25/14</b>		[Task bar]									
41	★	Navigation <-> line detection (Garrick &	15 days	Mon 7/7/14	Fri 7/25/14		[Task bar]									
42	★	Navigation <-> goal planning (Garrick &	15 days	Mon 7/7/14	Fri 7/25/14		[Task bar]									
43	★	Emergency stop and flashing light functionality	5 days	Mon 7/28/14	Fri 8/1/14		[Task bar]									
44	★	Final testing and improment and extra work if	80 days	Mon 8/4/14	Fri 11/21/14		[Task bar]									
45	★	Project seminar	5 days	Mon 9/29/14	Fri 10/3/14		[Task bar]									
46	★	Final Report Submission	0 days	Mon 10/27/14	Mon 10/27/14		[Task bar]									
47	★	Preperation for competition report	14 days	Mon 11/10/14	Thu 11/27/14		[Task bar]									
48	★	Expected competition date	2 days	Fri 11/28/14	Mon 12/1/14		[Task bar]									

◆ 3/5

Project: baseline\_sched  
Date: Fri 5/23/14

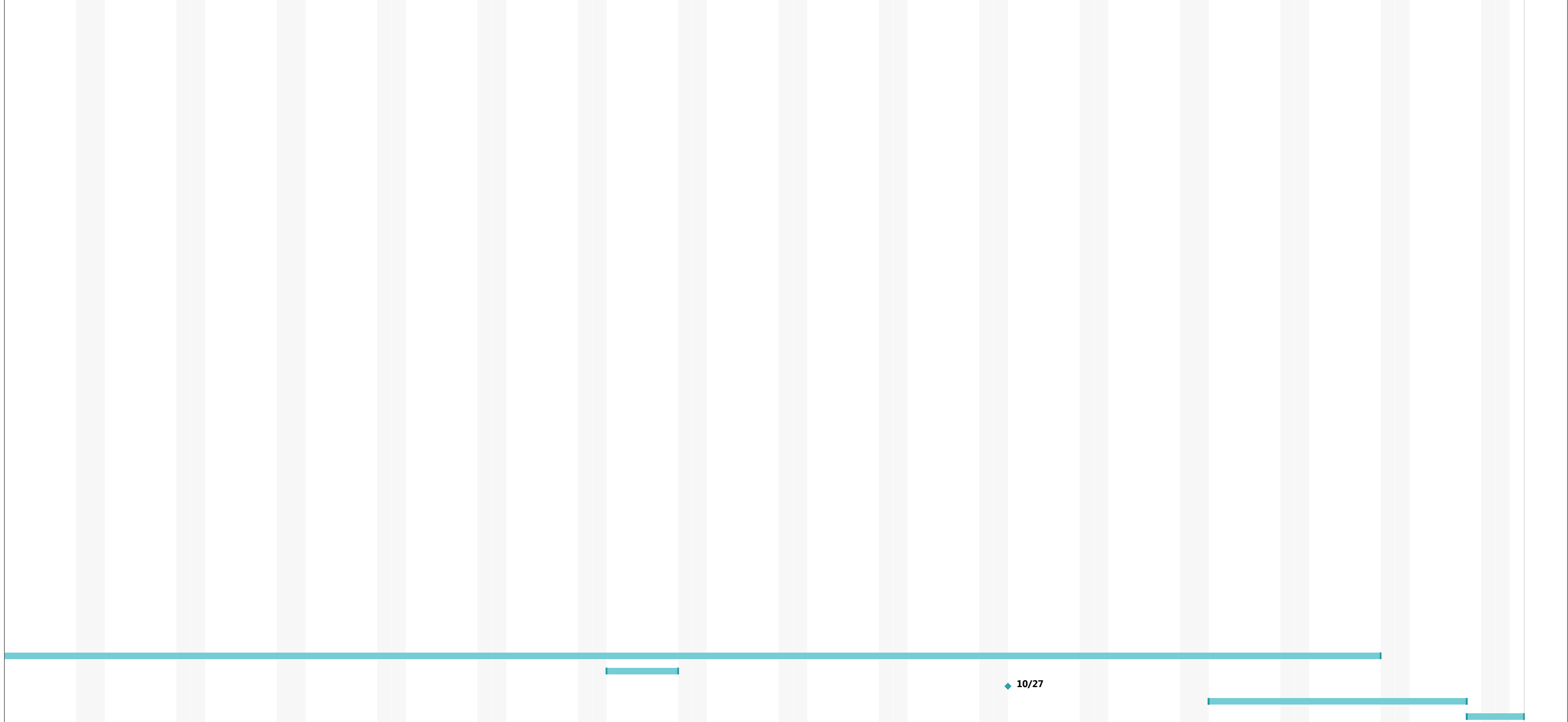
Task		Summary		Inactive Milestone		Duration-only		Start-only		External Milestone		Manual Progress	
Split		Project Summary		Inactive Summary		Manual Summary Rollup		Finish-only		Deadline			
Milestone		Inactive Task		Manual Task		Manual Summary		External Tasks		Progress			

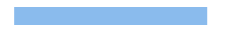

















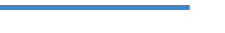


Project: baseline\_sched  
Date: Fri 5/23/14

Task		Summary		Inactive Milestone		Duration-only		Start-only		External Milestone		Manual Progress	
Split		Project Summary		Inactive Summary		Manual Summary Rollup		Finish-only		Deadline			
Milestone		Inactive Task		Manual Task		Manual Summary		External Tasks		Progress			

Jul 17, '14 | Aug 24, '14 | Aug 31, '14 | Sep 7, '14 | Sep 14, '14 | Sep 21, '14 | Sep 28, '14 | Oct 5, '14 | Oct 12, '14 | Oct 19, '14 | Oct 26, '14 | Nov 2, '14 | Nov 9, '14 | Nov 16, '14 | Nov 23, '14 | Nov 30, '14  
 M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S



Task		Summary		Inactive Milestone		Duration-only		Start-only		External Milestone		Manual Progress	
Split		Project Summary		Inactive Summary		Manual Summary Rollup		Finish-only		Deadline			
Milestone		Inactive Task		Manual Task		Manual Summary		External Tasks		Progress			

```
// Brief: This node detects white lines (continuous or dashed) in a 2D image
// converts these lines into 3D world coordinates. These are published so as
// to make a barrier in the robots world map.
//
// Dependencies:
//             - sensor_msgs
//             - cv_bridge
//             - roscpp
//             - std_msgs
//             - image_transport

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <vector>
#include <fstream>
#include <signal.h>
#include <boost/thread.hpp>
#include <string.h>
#include <math.h>
#include <sstream>
#include <map>
#include <list>

#include <ros/ros.h>
#include <ros/callback_queue.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>

#include <tf/transform_datatypes.h>
#include <tf/LinearMath/Matrix3x3.h>

#include <sensor_msgs/image_encodings.h>
#include <sensor_msgs/CameraInfo.h>
#include <sensor_msgs/Image.h>
#include <sensor_msgs/PointCloud.h>
#include <sensor_msgs/PointCloud2.h>
#include <sensor_msgs/point_cloud_conversion.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>
#include <geometry_msgs/PoseStamped.h>
#include <sensor_msgs/LaserScan.h>
#include <nav_msgs/OccupancyGrid.h>

#include "opencv2/opencv.hpp"
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/calib3d/calib3d.hpp>

#include "rapidxml/rapidxml.hpp"
#include "rapidxml/rapidxml_utils.hpp"

#include <errno.h>
#include <sys/types.h>
#include <sys/inotify.h>

#define PI 3.14159265
#define d2r PI/180
#define r2d 180/PI
#define EVENT_SIZE ( sizeof (struct inotify_event) )
#define BUF_LEN ( 1024 * ( EVENT_SIZE + 16 ) )

const char * file_path = "~/AGVC/vision_config";

using namespace std;
using namespace cv;
using namespace sensor_msgs;
using namespace rapidxml;

//prototypes:
void line_marking_detection(cv::Mat &src, cv::Mat &dest, int tau);
void update_gui(Mat & image, int position, string tag);
```

```

void draw_gui();
void img_to_img_data(Mat & img);

cv::Mat grid_traversable;
cv::Mat grid_traversable_soft;

Mat scaled_img;

//to synchronise the pose and camera
bool pose_updated = false;
ros::CallbackQueue ros_queue;

//filepath for the image to be written
string img_gui_path="/home/garrick/AGVC/images/test.jpg";

//file path where vision.xml is stored
string vision_config_path = "/home/garrick/AGVC/vision2.xml";

//main gui image
Mat gui = Mat::zeros( 720, 1280 , CV_8UC3 );

//fields to hold position data
//x=0, y=1, rotation=2
double live_position[3];
double position[3];

//LASER
sensor_msgs::LaserScan lsr_data;
ros::Publisher scanPublisher;

int img_data[1500*1500];
nav_msgs::OccupancyGrid combined_grid;

//structure containing all the things for image analysis
struct image_analysis {
    VideoCapture cam;                //camera image
    Mat warp;                        //warp matrix to give birds eye view(from
calibration_cam or read from conf. file)
    float angle;                    //offset angle of camera
    Point corners[4];               //corners(x,y) of image
    int cam_num;                    //camera number used(0,1,2)
    Mat include;                    //the seen area in the warped image
};

//structure containing all things necessary to read an xml file containing function name and setup
struct function {
    string name;                    //function name
    int perform;
    int screen;                    //perform if 1
    map<string, int> param;        //list of function parameters with name as key(string) and int
value
};

//field to store a list of functions
list<function> func_list;

//field to store the
Mat img_grid;

//field to store the grid data from laser scan
//0 = unoccupied, 100 = occupied
nav_msgs::OccupancyGrid grid_data;

//field to store laser scan data
sensor_msgs::LaserScan out_lsr;

/*
NAME: rotateImage
FUNCTION:
    Rotates a given image around the centre point(64,48) by a given angle.
VARIABLES:
    Mat source: image to be rotated

```

```

    double angle: the angle the image will be rotated by
*/
Mat rotateImage(const Mat& source, double angle)
{
    Point2f src_center(64, 48);
    Mat rot_mat = getRotationMatrix2D(src_center, angle, 1.0);
    Mat dst;
    warpAffine(source, dst, rot_mat, source.size());
    return dst;
}

/*
NAME: rotateImage2
FUNCTION:
    Rotates a given image around the centre point given, by a given angle.
VARIABLES:
    Mat source: image to be rotated
    double angle: the angle the image will be rotated by
    int x,y: the centre point of the image
*/
Mat rotateImage2(const Mat& source, double angle, int x, int y)
{
    Point2f src_center(x, y);
    Mat rot_mat = getRotationMatrix2D(src_center, angle, 1.0);
    Mat dst;
    warpAffine(source, dst, rot_mat, source.size());
    return dst;
}

//updates the main gui image by copying an image roi onto it
void update_gui(Mat & image, int position, string tag) {
    cv::Rect roi;
    if(position < 3) {
        Mat im = image.clone();
        if (im.channels() == 1)
            cvtColor(im, im, CV_GRAY2RGB);
        resize(im,im,Size(640,480),0,0,INTER_CUBIC);
        if(position == 1)
            roi = cv::Rect(0, 0,640 ,480);
        else
            roi = cv::Rect(640, 0, 640 ,480);

        im.copyTo(gui(roi));

        if (position == 1) {
            putText(gui, tag, Point2f(0,25), FONT_HERSHEY_PLAIN, 2, Scalar(255,255,255), 2);
        } else {
            putText(gui, tag, Point2f(640,25), FONT_HERSHEY_PLAIN, 2, Scalar(255,255,255), 2);
        }
    }
    else if (position <7) {
        Mat im = image.clone();
        if (im.channels() == 1)
            cvtColor(im, im, CV_GRAY2RGB);
        resize(im,im,Size(320,240),0,0,INTER_CUBIC);
        switch(position) {
            case 3:
                roi = cv::Rect(0, 480,320 ,240);
                break;
            case 4:
                roi = cv::Rect(320, 480,320 ,240);
                break;
            case 5:
                roi = cv::Rect(640, 480,320 ,240);
                break;
            case 6:
                roi = cv::Rect(960, 480,320 ,240);
                break;
        }
        im.copyTo(gui(roi));
        switch(position) {
            case 3:

```



```

        putText(gui, tag, Point2f(0,505), FONT_HERSHEY_PLAIN, 2, Scalar(255,255,255), 2);
        break;
    case 4:
        putText(gui, tag, Point2f(320,505), FONT_HERSHEY_PLAIN, 2, Scalar(255,255,255), 2);
        break;
    case 5:
        putText(gui, tag, Point2f(640,505), FONT_HERSHEY_PLAIN, 2, Scalar(255,255,255), 2);
        break;
    case 6:
        putText(gui, tag, Point2f(960,505), FONT_HERSHEY_PLAIN, 2, Scalar(255,255,255), 2);
        break;
    }
}
else {
    cout << "Function <update_gui>: invalid position" <<endl;
}
}

//writes the gui image to the path specified
void draw_gui() {
    imwrite(img_gui_path, gui);
}

//Scales down image to 5cm pixel raster
Mat& scale_down_linear(Mat& I, Mat& map_img)
{
    // accept only char type matrices
    CV_Assert(I.depth() != sizeof(uchar));
    int channels = I.channels(); //1 for gray image, 3 for RBG image
    int nRows = I.rows;
    int nCols = I.cols * channels;

    if (I.isContinuous())
    {
        nCols *= nRows;
        nRows = 1;
    }

    int i,j;
    uchar* p;
    float average = 0;

    for( i = 0; i < 480; i+=5) //move down rows in intervals of 5 pixels
    {
        for ( j = 0; j < 640; j+=5) //move across columns in intervals of 5 pixels
        {
            for (int p = i; p < i + 5; p++) //move down 5x5 interval
                for (int q = j; q < j + 5; q++) //move across 5x5 interval
                    average += I.ptr<uchar>(p)[q] * 1.0 / 255; //get average pixel intensity of
5x5 interval
            p = map_img.ptr<uchar>(i/5);
            if (average > 0) { //if average pixel intensity of image
is greater than 0 //than pixel intensity of 5x5 space on
map_img = 255 (white) = line detected
                p[j/5] = 255;
            }
            average = 0; //reset average to zero for next block
        }
    }

    return map_img;
}

/*
add map_img to 1500x1500 image
if line is detected in raster than the raster the line exists in is increased by a set amount
if no line detected than raster is decreased a set amount
*/
void heuristic_insert_image_map(Mat& I, Mat& test)
{
    int total_sub = 0;

```

```

int hur_add = 25;
int hur_subtract = 25;
int hur_keep = 25;

int i,j;
int start_x = (750+position[0]*20-128);
int start_y = (750+position[1]*20-128);
int im_val;

uchar* p;
uchar* q;

for( i = start_x; i < start_x + 256; ++i)
{
    for ( j = start_y; j < (start_y+256); ++j)
    {
        p = I.ptr<uchar>(i);
        q = test.ptr<uchar>(i-start_x);
        im_val = q[j - start_y];
        if(im_val > 75)
            p[j] = min(p[j] + hur_add,255);
        if(im_val == 1) {
            if(p[j]<255-hur_keep)
                p[j] = max(p[j] - hur_subtract,0);
            else
                p[j] = max(p[j] - 1,0);
            total_sub += 1;
        }
    }
}
//cout << total_sub <<endl;
}

void blur(Mat src, int max_kernel_length) {
    for ( int i = 1; i < max_kernel_length; i = i + 2 ) {
        blur( src, src, Size( i, i ), Point(-1,-1) );
    }
}

void gaussian_blur(Mat src, int max_kernel_length) {
    for ( int i = 1; i < max_kernel_length; i = i + 2 ) {
        GaussianBlur( src, src, Size( i, i ), 0, 0 );
    }
}

void median_blur(Mat src, int max_kernel_length) {
    for ( int i = 1; i < max_kernel_length; i = i + 2 ) {
        medianBlur ( src, src, i );
    }
}

//Erosion
void Erosion( Mat & src, int erosion_elem, int erosion_size)
{
    int erosion_type;
    if( erosion_elem == 0 ){ erosion_type = MORPH_RECT; }
    else if( erosion_elem == 1 ){ erosion_type = MORPH_CROSS; }
    else if( erosion_elem == 2 ) { erosion_type = MORPH_ELLIPSE; }

    Mat element = getStructuringElement( erosion_type,
                                        Size( 2*erosion_size + 1, 2*erosion_size+1 ),
                                        Point( erosion_size, erosion_size ) );

    /// Apply the erosion operation
    erode( src, src, element );
}

//Dilation

```

```

void Dilation( Mat & src, int dilation_elem, int dilation_size )
{
    int dilation_type;
    if( dilation_elem == 0 ){ dilation_type = MORPH_RECT; }
    else if( dilation_elem == 1 ){ dilation_type = MORPH_CROSS; }
    else if( dilation_elem == 2 ) { dilation_type = MORPH_ELLIPSE; }

    Mat element = getStructuringElement( dilation_type,
                                        Size( 2*dilation_size + 1, 2*dilation_size+1 ),
                                        Point( dilation_size, dilation_size ) );

    /// Apply the dilation operation
    dilate( src, src, element );
}

//Binary Threshold
void binary_thresh( Mat src, int value)
{
    threshold( src, src, value, 255, 0);
}

//Laplacian
Mat Laplacian( Mat src, int kernel_size)
{
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;

    Mat src_gray, dst;

    /// Remove noise by blurring with a Gaussian filter
    GaussianBlur( src, src, Size(3,3), 0, 0, BORDER_DEFAULT );

    /// Convert the image to grayscale
    ///cvtColor( src, src_gray, CV_RGB2GRAY );

    /// Apply Laplace function
    Mat abs_dst;

    Laplacian( src, dst, ddepth, kernel_size, scale, delta, BORDER_DEFAULT );
    convertScaleAbs( dst, abs_dst );
    return abs_dst;
}

//Canny
void Canny(Mat src, int lowThreshold, int ratio)
{
    Canny( src, src, lowThreshold, ratio*lowThreshold, 3);
}

double distance(double dx0, double dy0, double dx1, double dy1)
{
    return sqrt((dx1-dx0)*(dx1-dx0) + (dy1-dy0)*(dy1-dy0));
}

void open(Mat src) {
    int morph_elem = 0;
    int morph_size = 0;
    int const max_elem = 2;
    int const max_kernel_size = 21;
    Mat element = getStructuringElement( morph_elem, Size( 2*morph_size + 1, 2*morph_size+1 ),
    Point( morph_size, morph_size ) );
    morphologyEx( src, src, 2, element );
}

void close(Mat src) {
    int morph_elem = 0;
    int morph_size = 0;
    int const max_elem = 2;
    int const max_kernel_size = 21;
    Mat element = getStructuringElement( morph_elem, Size( 2*morph_size + 1, 2*morph_size+1 ),
    Point( morph_size, morph_size ) );
    morphologyEx( src, src, 3, element );
}

```

```
}

vector<Vec4i> linesP;
void HoughLinesP(Mat src, int hough_min, int threshold, int minLinLength, int maxLineGap) {
    //vector<Vec4i> lines;
    HoughLinesP(src, linesP, 1, CV_PI/180, threshold, minLinLength, maxLineGap );
    for( size_t i = 0; i < linesP.Size(); i++ )
    {
        Vec4i l = linesP[i];
        if(distance(l[0],l[1],l[2],l[3]) > hough_min) {
            line( src, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(255), 3, CV_AA);
        }
    }
}

void col_draw_linesP(Mat src, int hough_min) {
    for( size_t i = 0; i < linesP.size(); i++ )
    {
        Vec4i l = linesP[i];
        if(distance(l[0],l[1],l[2],l[3]) > hough_min) {
            line( src, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, CV_AA);
        }
    }
}

void white_draw_linesP(Mat src, int hough_min) {
    for( size_t i = 0; i < linesP.size(); i++ )
    {
        Vec4i l = linesP[i];
        if(distance(l[0],l[1],l[2],l[3]) > hough_min) {
            line( src, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(255), 3, CV_AA);
        }
    }
}

void open(Mat src, int morph_size) {
    int morph_elem = 0;
    Mat element = getStructuringElement( morph_elem, Size( 2*morph_size + 1, 2*morph_size+1 ),
    Point( morph_size, morph_size ) );
    morphologyEx( src, src, 2, element );
}

void close(Mat src, int morph_size) {
    int morph_elem = 0;
    Mat element = getStructuringElement( morph_elem, Size( 2*morph_size + 1, 2*morph_size+1 ),
    Point( morph_size, morph_size ) );
    morphologyEx( src, src, 3, element );
}

void split_image(Mat src) {
    Mat bgrchannel[3];
    split (src, bgrchannel);

    imshow("blue", bgrchannel[0]);
    imshow("green", bgrchannel[1]);
    imshow("red", bgrchannel[2]);
}

Mat mixChannel(Mat src) {
    Mat gray(src.size(),CV_8UC1,Scalar(0));
    int nrows = src.rows;
    int ncols = src.cols;

    for (int y = 0; y < nrows; y++) {
        for (int x = 0; x < ncols; x++) {
            Vec3b bgrPixel=src.at<cv::Vec3b>(y,x);

            gray.at<uchar>(y,x) = bgrPixel[0];
        }
    }
    return gray;
}
```

```
}

void histogram(Mat src)
{
    Mat image = src;

    // allcoate memory for no of pixels for each intensity value
    int histogram[256];

    // initialize all intensity values to 0
    for(int i = 0; i < 255; i++)
    {
        histogram[i] = 0;
    }

    // calculate the no of pixels for each intensity values
    for(int y = 0; y < image.rows; y++)
        for(int x = 0; x < image.cols; x++)
            histogram[(int)image.at<uchar>(y,x)]++;

    // draw the histograms
    int hist_w = 512; int hist_h = 400;
    int bin_w = cvRound((double) hist_w/256);

    Mat histImage(hist_h, hist_w, CV_8UC1, Scalar(255, 255, 255));

    // find the maximum intensity element from histogram
    int max = histogram[0];
    for(int i = 1; i < 256; i++){
        if(max < histogram[i]){
            max = histogram[i];
        }
    }

    // normalize the histogram between 0 and histImage.rows
    for(int i = 0; i < 255; i++){
        histogram[i] = ((double)histogram[i]/max)*histImage.rows;
    }

    // draw the intensity line for histogram
    for(int i = 0; i < 255; i++)
    {
        line(histImage, Point(bin_w*(i), hist_h),
            Point(bin_w*(i), hist_h - histogram[i]),
            Scalar(0,0,0), 1, 8, 0);
    }

    // display histogram
    namedWindow("Intensity Histogram", CV_WINDOW_AUTOSIZE);
    imshow("Intensity Histogram", histImage);
}

void adaptive_thresh(Mat src) {
    threshold(src, src, 0, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
}

//Convert image to gray image
Mat BW(Mat src) {
    Mat dst;
    cvtColor(src, dst, CV_RGB2GRAY);
    return dst;
}

vector<Vec2f> hough_lines;
void hough(Mat src)
{
    HoughLines(src, hough_lines, 1, 5*PI/180, 80);
}
```

```

        for( size_t i = 0; i < hough_lines.size(); i++ )
    {
        float rho = hough_lines[i][0], theta = hough_lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        pt1.x = cvRound(x0 + 1000*(-b));
        pt1.y = cvRound(y0 + 1000*(a));
        pt2.x = cvRound(x0 - 1000*(-b));
        pt2.y = cvRound(y0 - 1000*(a));
        line( src, pt1, pt2, Scalar(255), 3, CV_AA);
    }
}

void draw_hough_lines(Mat src)
{
    for( size_t i = 0; i < hough_lines.size(); i++ )
    {
        float rho = hough_lines[i][0], theta = hough_lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        pt1.x = cvRound(x0 + 1000*(-b));
        pt1.y = cvRound(y0 + 1000*(a));
        pt2.x = cvRound(x0 - 1000*(-b));
        pt2.y = cvRound(y0 - 1000*(a));
        line( src, pt1, pt2, Scalar(0,0,255), 3, CV_AA);
    }
}

//function to draw the occupancy grid to an image
Mat draw_grid(const nav_msgs::OccupancyGrid &grid) {
    Mat map_img = Mat::zeros( 1500, 1500 , CV_8UC1 );
    for (int i =0; i < 1500; i++) {
        uchar* rowi = map_img.ptr<uchar>(i);
        for (int j = 0; j < 1500; j++) {
            rowi[j] = grid.data[i+j*1500];
            //map_img.at<double>(i,j) = 0;
        }
        //imwrite(img_gui_path, map_img);
    }
    return map_img;
}

//grid callback function
void grid_callback(const nav_msgs::OccupancyGrid &grid) {
    grid_data = grid;
    //cout<<grid<<endl;
    //for (int i =0; i < sizeof(grid.data); i++)
    //    cout << grid.data[i] << endl;
    //draw_grid(grid);
}

void grid_callback2(const nav_msgs::OccupancyGrid &grid) {
    combined_grid = grid;
}

// Image Processing Functions

//This function takes in the image and applies it to both the grid and GUI
void vision_into_grid(cv::Mat img, Point corners[], float offset_angle, Mat & include, int cam_num) {
    static int num = 0;
    const int gui_update = 2;

```

```

    //vision scaled to 5cm pixel raster
    //Fills the area the camera "sees" with 1's instead of 0's
    //Allows for the logic to remove unwanted parts.
    Mat scaled_img = include.clone();
    imshow("1", scaled_img);

    scaled_img = scale_down_linear(img, scaled_img);
    imshow("2", scaled_img);

    //Flips the image, because the view from the camera is the opposite of the
    //hector slam and image map
    scaled_img = rotateImage(scaled_img,180);
    imshow("3", scaled_img);

    //Creates an image four times the size, adds the image in and the rotates it to get the correct
    //angle in the real world.
    Mat rotatable_img= Mat::zeros( 256, 256 , CV_8UC1 );
    cv::Rect roi( cv::Point( 64, 128 ), scaled_img.size() );
    scaled_img.copyTo( rotatable_img( roi ) );
    imshow("4", rotatable_img);

    rotatable_img= rotateImage2(rotatable_img,position[2]*180/M_PI-offset_angle,128,128);
    imshow("5", rotatable_img);

    heuristic_insert_image_map(img_grid, rotatable_img);
    imshow("6", img_grid);

    //Mat grid = draw_grid(grid_data);
}

//this is for getting the corners of the warp matrix from the warp matrix. To remove edges detected by
the warping process
void get_world_points(Mat & warp, Point points[], Mat & include) {

    vector<Point2f> camera_points;
    camera_points.push_back(Point2f(0,0));
    camera_points.push_back(Point2f(0,96));
    camera_points.push_back(Point2f(128,96));
    camera_points.push_back(Point2f(128,0));

    //insert points from your camera image here

    vector<Point2f> world_points;
    perspectiveTransform(camera_points, world_points, warp);

    for(int i = 0; i < world_points.size(); i++) {
        cout << world_points.at(i);
        points[i] = Point(world_points.at(i).x, world_points.at(i).y);
    }
    Mat img = Mat::zeros( 480, 640 , CV_8UC1 );
    img.setTo(1);
    cv::warpPerspective(img,img,warp,img.size());
    resize(img,img,Size(128,96),0,0,INTER_CUBIC);
    include = img.clone();
}

//generates an image from laser scanner data
Mat generate_lsr_img(sensor_msgs::LaserScan data, float offset) {
    int laser_scale = 100;
    Mat lsr_img = Mat::zeros( 480, 640 , CV_8UC1 );
    int i = 0;
    if(data.angle_min == 0) {
        printf("generate_lsr_img: 1, LASER SCANNER IS RETURNING NO DATA\n");
    } else {
        float scaler = laser_scale/100.0;
        float off_angle = offset/180*M_PI;
        float angle = data.angle_min;

        while (angle < data.angle_max) {
            if(data.ranges[i] == 0)
                data.ranges[i] = 9999;
            int x = -1 * data.ranges[i]*200/scaler * sin(angle + off_angle ) +320;

```

```

        int y = -1 * data.ranges[i]*200/scaler * cos(angle + off_angle ) +480;
        cv::Point start(320, 480);
        cv::Point end(x, y);
        cv::line(lsr_img,start,end, Scalar::all(255), 8);
        //cout << data.ranges[i] << " ";
        angle+=data.angle_increment;
        i++;
    }
}
if(i==0)
    printf("generate_lsr_img: 2, LASER SCANNER IS RETURNING NO DATA\n");

return lsr_img;
}

//remove laser from image
void remove(Mat& image, Mat& lsr) {
    for(int y = 0; y < image.rows-1; y++) {
        for(int x = 0; x < image.cols; x++) {
            if(lsr.at<uchar>(y,x) == 0) {
                image.at<uchar>(y,x) = 0;
            }
        }
    }
}

void find_corners(Mat src)
{
    // output vectors of image points
    std::vector<cv::Point2f> imageCorners;
    // number of corners on the chessboard
    cv::Size boardSize(9,9);
    // Get the chessboard corners
    bool found = cv::findChessboardCorners(src,
    boardSize, imageCorners);
    //Draw the corners
    cv::drawChessboardCorners(src,
    boardSize, imageCorners,
    found); // corners have been found
    for(int i = 0; i < imageCorners.size(); i++)
    {
        cout << imageCorners[i].x << "," << imageCorners[i].y << endl;
    }
}

//function to perform image processing
void img_process(image_analysis details){
    const clock_t begin_time = clock();
    VideoCapture cap = details.cam;
    cv::Mat warp = details.warp;
    float angle = details.angle;
    Point corners[4] = details.corners;
    //old begin time place
    static int i =0;
    static float total_time = 0;
    Mat colframe, frame, gray_frame, frame_cap;
    pose_updated = false;

    //ros_queue.callAvailable(ros::WallDuration(1));
    position[0] = live_position[0]; //save the position when the picture was taken.
    position[1] = live_position[1]; //save the position when the picture was taken.
    position[2] = live_position[2]; //save the position when the picture was taken.

    //here is where the picture is taken from the camera
    cap.grab();
    cap.retrieve(frame);
    //warpPerspective(frame, frame, warp, frame.size());
    //find_corners(frame);
    imwrite("/home/garrick/AGVC/testpic/image3.png", frame);
}

```



```

imshow("frame", frame);
waitKey(10);

Mat line_frame(frame.size(), CV_8U, cv::Scalar(0));
colframe = frame.clone();

//updating the GUI
//update_gui(frame,3,"raw");
//update_gui(frame,4,"raw");
//update_gui(frame,5,"raw");
//update_gui(frame,6,"raw");

//frame =BW(frame);
for(list<function>::iterator it = func_list.begin();
it != func_list.end(); it++)
{
    if ((strcmp(it->name.c_str(), "warp1") == 0) && (it->perform == 1))
    {
        warpPerspective(frame, frame, warp, frame.size());
        if (it->screen == 1){
            imshow("warp", frame);
        }
    }
    if ((strcmp(it->name.c_str(), "gray") == 0) && (it->perform == 1)){
        gray_frame = BW(frame);
        if (it->screen == 1){
            imshow("gray frame", gray_frame);
        }
    }
    if ((strcmp(it->name.c_str(), "mixChannel") == 0) && (it->perform == 1)){
        frame = mixChannel(frame);
        if (it->screen == 1){
            imshow("mixChannel", frame);
        }
    }
    if ((strcmp(it->name.c_str(), "blur") == 0) && (it->perform == 1)){
        blur(frame, it->param["max_kernel_length"]);
        if (it->screen == 1){
            imshow("blur", frame);
        }
    }
    if ((strcmp(it->name.c_str(), "gaussian_blur") == 0) && (it->perform == 1)){
        gaussian_blur(frame, it->param["max_kernel_length"]);
        if (it->screen == 1){
            imshow("blur", frame);
        }
    }
    if ((strcmp(it->name.c_str(), "median_blur") == 0) && (it->perform == 1)){
        median_blur(frame, it->param["max_kernel_length"]);
        if (it->screen == 1){
            imshow("blur", frame);
        }
    }
    if ((strcmp(it->name.c_str(), "histogram") == 0) && (it->perform == 1))
    {
        histogram(frame);
    }
    if ((strcmp(it->name.c_str(), "binary_thresh") == 0) && (it->perform == 1)){
        binary_thresh(frame, it->param["thesh_value"]);
        if (it->screen == 1){
            imshow("thresh_1", frame);
        }
    }
    if ((strcmp(it->name.c_str(), "adaptive_thresh") == 0) && (it->perform == 1)){
        adaptive_thresh(frame);
        if (it->screen == 1){
            imshow("thresh_1", frame);
        }
    }
    if ((strcmp(it->name.c_str(), "close") == 0) && (it->perform == 1)){
        close(frame, it->param["morph_size"]);
        if (it->screen == 1){

```

```

        imshow("open",frame);
    }
    if ((strcmp(it->name.c_str(), "open") == 0) && (it->perform == 1)){
        open(frame, it->param["morph_size"]);
        if (it->screen == 1){
            imshow("open",frame);
        }
    }
    if ((strcmp(it->name.c_str(), "houghP") == 0) && (it->perform == 1))
    {
        HoughLinesP(frame, it->param["hough_min"], it->param["thresholdP"], it->param
["minLinLength"], it->param["maxLineGap"]);
        if (it->screen == 1){
            imshow("houghP",frame);
        }
        col_draw_linesP(colframe, it->param["hough_min"]);
        white_draw_linesP(line_frame, it->param["hough_min"]);
        imshow("colframe", colframe);
        imshow("lines frame", line_frame);
    }
    if ((strcmp(it->name.c_str(), "hough") == 0) && (it->perform == 1))
    {
        hough(frame);
        draw_hough_lines(colframe);
        imshow("hough col", colframe);
        if (it->screen == 1){
            imshow("hough",frame);
        }
    }
    if ((strcmp(it->name.c_str(), "warp2") == 0) && (it->perform == 1))
    {
        warpPerspective(line_frame,line_frame,warp,line_frame.size());
        if (it->screen == 1){
            imshow("warp",line_frame);
        }
    }
}

imshow("warped", line_frame);

//scaled_img = details.include.clone();
//scaled_img = scale_down_linear(line_frame, scaled_img);
//scaled_img = rotateImage(line_frame,180);
scaled_img = line_frame;
imshow("output_frame", scaled_img);

//vision_into_grid(frame, corners, angle, details.include, details.cam_num);

i+=1;

//draw_gui();
float duration = float( clock () - begin_time ) / CLOCKS_PER_SEC;
total_time += duration;
//cout <<"cam: " << details.cam_num << " fps: " << i/ total_time << ", total time: " <<duration<< " ,
frames: " <<i << endl;
//cout << " x:" <<position[0]<<" y:" <<position[1]<<" o:" <<position[2]<< "cam_angle:
"<<details.angle << endl;
}

//pose callback function
void pose_callback(const geometry_msgs::PoseWithCovarianceStamped &pose) {
    geometry_msgs::Quaternion o = pose.pose.pose.orientation;
    double x = live_position[0] = pose.pose.pose.position.x ;
    double y = live_position[1] = pose.pose.pose.position.y;
    tf::Quaternion q;
    tf::quaternionMsgToTF(o,q);
    double roll, pitch, yaw;
    tf::Matrix3x3(q).getRPY(roll, pitch, yaw);
    //tf::Matrix3x3 mat;

```

```

    //cout << pose << endl;
    double yaw_deg = yaw *180 / M_PI;
    live_position[2] = yaw;
    pose_updated = true;
    cout << "x, y, rotation " << x << " " << y << " " << yaw_deg << endl;
}

/*NAME: deserialise_conf
FUNCTION:
    deserialises the calibration file containing a warp matrix for a camera
VARIABLES:
    cam_num: The opencv camera number
    angle: the cameras offset angle
*/
Mat deserialise_conf(int cam_num, float & angle) {

    std::ostringstream s;
    s << "/home/garrick/AGVC/0_cam_calibration.conf";
    std::ifstream file(s.str().c_str());
    int vals[16];
    std::string line;
    int counter = 0;
    for(int i = 0; i < 2 && std::getline( file, line ); i++ )
    {
        std::istringstream iss( line );

        std::string result;

        while( std::getline( iss, result, ',' ) )
        {
            vals[counter] = atoi(result.c_str());
            counter ++;
        }

    }

    std::cout << "read in calibration file" << std::endl;
    cv::Point2f src[4],dst[4];
    for(int i = 0; i < 4; i++) {
        src[i].x = vals[i*2];
        src[i].y = vals[i*2+1];
        dst[i].x = vals[i*2+8];
        dst[i].y = vals[i*2+9];
    }
    for(int i = 0; i < 4; i++) {
        cout << src[i].x << " " << src[i].y << endl;
    }
    for(int i = 0; i < 4; i++) {
        cout << dst[i].x << " " << dst[i].y << endl;
    }
    Mat warp_matrix = getPerspectiveTransform(src,dst);

    //GET THE ANGLE
    std::getline( file, line );
    angle = atof(line.c_str());
    return warp_matrix;
}

//Function to read an xml file containing order and parameters of image processing functions
void read_vision_config() // _callback(const ros::TimerEvent&)
{
    cout << "reading config file" << endl;
    file<> xmlFile("/home/garrick/AGVC/vision_config/vision2.xml");
    xml_document<> doc;
    doc.parse<0>(xmlFile.data());
    xml_node<> *node = doc.first_node();
    function func;
    func_list.clear();
    while(node != 0) {
        //cout << node->name() << endl;
        xml_node<> *node2 = node->first_node();
        while(node2 != 0)
        {

```

```

        //cout << node2->name() << endl;
        xml_node<> *child = node2->first_node();
        while(child != 0)
            {
                //cout << child->name() << " " << child->value() <<
endl;

                if (strcmp(child->name(), "name") == 0)
                {
                    //function name
                    func.name = child->value();
                }
                if (strcmp(child->name(), "perform") == 0) { //
perform function?
                    func.perform = atoi(child->value());
                }
                if (strcmp(child->name(), "screen") == 0) { //
perform function?
                    func.screen = atoi(child->value());
                }
                if ((strcmp(child->name(), "name") != 0) && (strcmp(child->name(),
"perform") != 0) && (strcmp(child->name(), "screen") != 0)) {
                    func.param[child->name()] = atoi(child->value()); //function
parameters
                }
                child = child->next_sibling();
            }
        func_list.push_back(func);
        node2 = node2->next_sibling();
    }
    node = node->next_sibling();
}

//Laser Callback function
void lsr_callback(const sensor_msgs::LaserScan &data) {
    lsr_data = data;
}

//variables used in calibrate cam
int calibrate_clicks[4][2] ;
int click_count = 0;

//calibrate clicks callback function
void calibrate_callback(int event, int x, int y, int flags, void* userdata) {
    if((flags==33 || flags == 1) and click_count < 4) {
        calibrate_clicks[click_count][0] = x;
        calibrate_clicks[click_count][1] = y;
        cout << "click at (x,y) ("<<x<<","<<y<<")"<<endl;
        click_count++;
    }
}

cv::Point objects[2];

/*NAME: cal_lsr_dist
//FUNCTION: finds an object in the laser scanner data between the start and end angles and returns a
max distance and angle of the center of the object
//VARIABLES:
    data: laser scanner data

//NOTE, angles are in degrees
*/

float cal_lsr_dist(sensor_msgs::LaserScan data, float s_angle, float e_angle, float max_dist, float
&ret_angle) {
    //converting to radians
    s_angle *= M_PI/180;
    e_angle *= M_PI/180;

    float angle = data.angle_min;
    int i = 0;
    cout << s_angle << " " << e_angle << " " <<endl;

```

```

//TESTING
Mat lsr_img = Mat::zeros( 960, 1280 , CV_8UC3 );

bool is_next = false;
int first_p_x = 0;
int first_p_y = 0;
float first_dist = 0;
float first_ang = 0;

int last_p_x = 0;
int last_p_y = 0;
float last_dist = 0;
float last_ang = 0;

int num_points = 0;

//|| angle < e_angle
while (angle < data.angle_max) {
    if(angle > s_angle && angle < e_angle) {
        //TESTING
        int x = -1 * data.ranges[i]*100 * sin(angle) +640;
        int y = -1 * data.ranges[i]*100 * cos(angle) +480;
        cv::Point start(640, 480);
        cv::Point end(x, y);
        cv::line(lsr_img,start,end, Scalar::all(255), 3);
        //cout << angle << " " <<endl;
        if(data.ranges[i]<max_dist && data.ranges[i]!=0) {
            float d_angle = angle *180 / M_PI;
            cout<<"range: " << data.ranges[i]<< " " << d_angle <<endl;
            if(is_next == false) {
                first_p_x= x;
                first_p_y= y;
                first_dist = data.ranges[i];
                first_ang = d_angle;
            } else {
                last_p_x= x;
                last_p_y= y;
                last_dist = data.ranges[i];
                last_ang = d_angle;
            }

            is_next = true;
        } else {
            if(is_next == true) {
                int center_x = (first_p_x+ last_p_x )/2 -640;
                int center_y = (first_p_y+ last_p_y )/2 -480;
                ret_angle = atan2(center_y,center_x)*180/M_PI;
                if (ret_angle > 0) {
                    ret_angle -= 90;
                } else
                    ret_angle += 90;
                printf("start (%d,%d) end (%d,%d)\n",first_p_x, first_p_y, last_p_x, last_p_y);
                printf("center (%d, %d) angle %f)\n",center_x, center_y, ret_angle);

                objects[0] = cv::Point(first_p_x, first_p_y);
                objects[1] = cv::Point(last_p_x, last_p_y);
                num_points ++;
            }
            is_next = false;
        }
    }

    angle+=data.angle_increment;
    i++;
}

//for(int a = 0; a < 100; a++)
imshow("lsr test", lsr_img);
cv::waitKey(5000);

```

```

cout << endl;

if(num_points ==0) {
    cout << "no objects found.. :(" << endl;
} else if (num_points > 1) {
    cout << num_points << " objects found... cannot compute." << endl;
} else {
    cout << "1 object found! " << first_dist << " " << last_dist << endl;
    float ob_dist = (first_dist + last_dist) /2;
    cout << "distance away... " << ob_dist << endl;

    return ob_dist;
}

return 0;
}

void serialise_conf(int cam_num, cv::Point2f src[4], cv::Point2f dst[4], float angle) {
    ofstream myfile;
    std::ostringstream s;
    s << cam_num << "_cam_calibration.conf";
    myfile.open (s.str().c_str());
    myfile << src[0].x << "," << src[0].y << "," << src[1].x << "," << src[1].y << ",";
    myfile << src[2].x << "," << src[2].y << "," << src[3].x << "," << src[3].y << "\n";
    myfile << dst[0].x << "," << dst[0].y << "," << dst[1].x << "," << dst[1].y << ",";
    myfile << dst[2].x << "," << dst[2].y << "," << dst[3].x << "," << dst[3].y << "\n";
    myfile << angle << "\n";
    myfile.close();
}

/*NAME: calibrate_cam

//FUNCTION:
    Calibrates a cameras warp matrix using a chessboard and a pole.

//VARIABLES:
    cam_num: the camera number for opencv. generally speaking its from lowest to highest usb value
    nh: The node handle so the function can access the laser scanner
    s_angle: the starting angle to search through the laser scanner data to find the object representing
the chessboard distance
    e_angle: ending angle (see above)

*/
void calibrate_cam(int cam_num, ros::NodeHandle nh, int s_angle, int e_angle) {
    cout << "starting calibration of camera number: " << cam_num << endl;
    cout << "click four corners in the following order: top left, bottom left, bottom right, top
right" << endl;
    VideoCapture cap = VideoCapture(cam_num);
    int cam_width = cap.get(CV_CAP_PROP_FRAME_WIDTH);
    int cam_height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);

    Mat frame;
    cap.grab();
    cap.retrieve(frame);

    float angle = 0;
    float distance = cal_lsr_dist(lsr_data,s_angle,e_angle, 1.8, angle);
    if(distance != 0) {
        resize(frame,frame,Size(640,480),0,0,INTER_CUBIC);

        cout <<"get clicks.." << endl;
        imshow("calibrate camera",frame);
        cv::setMouseCallback("calibrate camera",calibrate_callback);
        while(click_count < 4){
            cv::waitKey(1);
        }

        int height = 480;
        int width = 640;
        float board_size = 40; //board size in cm
        cv::Point2f src[4],dst[4];

```

```

    src[0].x = calibrate_clicks[0][0];
    src[0].y = calibrate_clicks[0][1];
    src[1].x = calibrate_clicks[1][0];
    src[1].y = calibrate_clicks[1][1];
    src[2].x = calibrate_clicks[2][0];
    src[2].y = calibrate_clicks[2][1];
    src[3].x = calibrate_clicks[3][0];
    src[3].y = calibrate_clicks[3][1];

    int y_top = height - distance*100;
    int y_bottom = height - distance*100 + board_size;

    dst[0].x = width/2 - board_size /2; //dst Top left
    dst[0].y = y_top;
    dst[1].x = width/2 - board_size /2; //dst Top right
    dst[1].y = y_bottom;
    dst[2].x = width/2 + board_size /2; //dst Bottom left
    dst[2].y = y_bottom;
    dst[3].x = width/2 + board_size /2; //dst Bot right
    dst[3].y = y_top;

    cv::Mat warpMatrix =cv::getPerspectiveTransform(src,dst);

    serialise_conf(cam_num, src, dst, angle);

    cv::warpPerspective(frame,frame,warpMatrix,frame.size());
    imshow("calibrate camera",frame);
    cv::waitKey(5000);
    cout << "calibration complete for camera " << cam_num <<endl;
} else {
    cout << "calibration failed" <<endl;
}
}

//access point for function
int main(int argc, char** argv) {
    cout<<"LINE DETECTION STARTING MAIN VERSION 1.0"<<endl;

    //calibration for cameras
    ros::init(argc, argv, "line_detection");
    ros::NodeHandle nh;

    //ros::Timer read_vision = nh.createTimer(ros::Duration(1.0), read_vision_config);

    cout<<"getting laser data (sometimes this takes 30 seconds or so...)"<<endl;

    ros::Subscriber lsr_sub = nh.subscribe("/scan_sick1", 1, &lsr_callback);
    while (lsr_data.scan_time==0){
        //apparently it can return nothing, keep spinning till it returns data.
        ros::spinOnce();
        //cout<<"got laser data"<<lsr_data<<endl;
        //cout << +lsr_data <<endl;
    }
    cout<<"got laser data"<<endl;

    //if arguments calibrate, cam_num, start_angle and end_angle provided in command line
    //than calibrate camera (cam_num)
    if(argc>4 && strcmp(argv[1],"calibrate")== 0) {
        calibrate_cam(atoi(argv[2]),nh,atoi(argv[3]),atoi(argv[4]));
    }

    //otherwise begin image processing
    else {

        img_grid = Mat::zeros( 1500, 1500 , CV_8UC1 );

        //nh.setCallbackQueue(&ros_queue);

        scanPublisher = nh.advertise<sensor_msgs::LaserScan>("/scan_updated",10);

```

```

    ros::Subscriber pose_sub = nh.subscribe("/poseupdate", 1, &pose_callback);
    ros::Subscriber lsr_sub = nh.subscribe("/scan", 1, &lsr_callback);
    ros::Subscriber grid_sub = nh.subscribe("/map", 1, &grid_callback);

//ros_queue.callAvailable(ros::WallDuration(1));

//SETUP CAMERAS
cout << "starting setup" <<endl;
int num_cams =1;
image_analysis cam_run[num_cams];

for (int i = 0; i < num_cams; i ++) {
    cam_run[i].warp = deserialise_conf(i,cam_run[i].angle);
    cam_run[i].cam = VideoCapture(i); // open the default camera
    cam_run[i].cam_num = i;
    get_world_points(cam_run[i].warp, cam_run[i].corners, cam_run[i].include);
}

ros::Publisher gridPublisher = nh.advertise<nav_msgs::OccupancyGrid>("/map_updated",1);
ros::Publisher cloud_pub = nh.advertise<sensor_msgs::PointCloud2>("cloud",50);

int i = 0;

cout << "beginning image analysis" <<endl;
grid_traversable = Mat::zeros( 1500, 1500 , CV_8UC3 );

while(ros::ok()) {
    //run camera

        //read configuration file at 1Hz

        //ros::Timer read_vision = nh.createTimer(ros::Duration(1.0),
read_vision_config);
        read_vision_config();

        //perform image processing and insert image into img_grid
        img_process(cam_run[0]);

        i +=1;
        /*if (i >= num_cams) {
            i = 0;
        }
        */

        //this code is for creating the point cloud that can be inserted into the
global costmap
        sensor_msgs::PointCloud cloud;
        cloud.header.frame_id = "cam_frame";
        int cloud_num_points = 0;

        for(int y = 0; y < scaled_img.rows-1; y++) {
            for(int x = 0; x < scaled_img.cols; x++) {
                if (scaled_img.at<uchar>(y,x) > 200) {
                    cloud_num_points++;
                }
            }
        }

        //cloud_num_points = scaled_img.rows * scaled_img.cols;

        cout << "num points " << cloud_num_points << endl;
        int cpi = 0;

        cloud.channels.resize(1);

```



```
cloud.channels[0].name = "intensities";
cloud.channels[0].values.resize(cloud_num_points);
cloud.points.resize(cloud_num_points);
for(int y = 0; y < scaled_img.rows-1; y++) {
    for(int x = 0; x < scaled_img.cols; x++) {
        if (scaled_img.at<uchar>(y,x) > 200) {
            if(cpi < cloud_num_points) { //quick seg fault fix
                cloud.points[cpi].x = -(y-480.0)*0.01;
                cloud.points[cpi].y = -(x-320.0)*0.01;
                cloud.points[cpi].z = 0;
                cloud.channels[0].values[cpi] = 100;
                cpi++;
            }
        }
    }
}
cloud.header.stamp = ros::Time::now();

sensor_msgs::PointCloud2 cloud2;
/*
sensor_msgs::PointCloud2 cloud2 = sensor_msgs::PointCloud2();
sensor_msgs::PointCloud2* cloud2 = new sensor_msgs::PointCloud2();
*/

sensor_msgs::convertPointCloudToPointCloud2(cloud, cloud2);

cloud_pub.publish(cloud2);

}
//Spin ros once, run again
ros::spinOnce();
}

return 0;
}
```

```
#!/bin/bash
```

```
echo "sourcing ros files and setting up env. variables."
```

```
source /opt/ros/indigo/setup.bash
```

```
source ~/AGVC/catkin_ws/devel/setup.bash
```

```
#export ROS_HOSTNAME=localhost
```

```
#export ROS_MASTER_URI=http://localhost:11311
```

```
export ROS_HOSTNAME=N3_wam
```

```
export ROS_MASTER_URI=http://N3_wam:11311
```

```
#export ROS_HOSTNAME=N3_lan
```

```
#export ROS_MASTER_URI=http://N3_lan:11311
```

```
#export ROS_HOSTNAME=N3_vpn
```

```
#export ROS_MASTER_URI=http://N3_vpn:11311
```

```
SUBSYSTEMS=="usb", KERNEL=="ttyUSB[0-9]*", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="d38b", SYMLINK
+="gandobot/xsens1", MODE="0666", GROUP="dialout"
SUBSYSTEMS=="usb", KERNEL=="ttyACM[0-9]*", ATTRS{idVendor}=="15d1", ATTRS{idProduct}=="0000", SYMLINK
+="gandobot/hokuyo1", MODE="0666", GROUP="dialout"
SUBSYSTEMS=="usb", KERNEL=="ttyACM[0-9]*", ATTRS{idVendor}=="2341", ATTRS{idProduct}=="0042", SYMLINK
+="gandobot/arduino1", MODE="0666", GROUP="dialout"
SUBSYSTEMS=="usb", KERNEL=="ttyUSB[0-9]*", ATTRS{idVendor}=="067b", ATTRS{idProduct}=="2303", SYMLINK
+="gandobot/p3at1", MODE="0666", GROUP="dialout"
SUBSYSTEMS=="usb", KERNEL=="ttyACM[0-9]*", ATTRS{idVendor}=="0e8d", ATTRS{idProduct}=="3329", SYMLINK
+="gandobot/qstarz1", MODE="0666", GROUP="dialout"
SUBSYSTEMS=="usb", KERNEL=="ttyACM[0-9]*", ATTRS{idVendor}=="26ba", SYMLINK+="gandobot/uno1",
MODE="0666", GROUP="dialout"
```

```

<launch>
  <node name="base" pkg="p2os_driver" type="p2os_driver" args="pose:=odom tf:=tf_p2os">
    <param name="port" value="/dev/gandobot/p3at1"/>
    <param name="pulse" type="double" value="1.0"/>
  </node>
  <node name="rostopic" pkg="rostopic" type="rostopic" args="pub -1 /cmd_motor_state p2os_msgs/
MotorState 1"/>

  <include file="$(find p2os_urdf)/launch/upload_pioneer3at.xml"/>
  <node pkg="robot_state_publisher" type="state_publisher" name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="30.0"/>
    <param name="tf_prefix" type="string" value=""/>
  </node>
  <node pkg="tf" type="static_transform_publisher" name="sick1_tf_broadcaster" args="0.135 0
0.25 0 0 0 top_plate sick1 100"/>
  <node pkg="tf" type="static_transform_publisher" name="hok1_tf_broadcaster" args="0.223 0
0.060 0 0 0 top_plate hok1 100"/>
  <node pkg="tf" type="static_transform_publisher" name="cam_tf_broadcaster" args="0.135 0 0 0 0
0 base_link cam_frame 100"/>
  <node pkg="tf" type="static_transform_publisher" name="xsens1_tf_broadcaster" args="-0.330 0.0
0.170 0 0 0 top_plate xsens1 100"/>
  <node pkg="pandobot" type="base_NWU_tf_publisher.py" name="base_NWU_tf_publisher"/>      <!--
mag field ref -->

  <node pkg="lms1xx" name="sick1" type="LMS1xx_node" args="scan:=scan_sick1">
    <param name="host" value="192.168.4.3"/>
    <param name="frame_id" value="sick1"/>
  </node>

  <node name="mapper_sick1" pkg="gmapping" type="slam_gmapping" args="scan:=scan_sick1
map:=map_sick1 map_metadata:=map_sick1_meta">
    <param name="map_frame" value="map_sick1"/>
    <param name="xmin" value="-10"/>
    <param name="ymin" value="-10"/>
    <param name="xmax" value="10"/>
    <param name="ymax" value="10"/>

    <param name="map_update_interval" value="2.5"/>
  </node>

  <node pkg="move_base" type="move_base" name="move_base" output="screen">
    <rosparam file="$(find pandobot)/launch/move_basic/local.yaml" command="load" />
    <rosparam file="$(find pandobot)/launch/move_basic/global.yaml" command="load" />
    <rosparam file="$(find pandobot)/launch/move_basic/planner.yaml" command="load" />
  </node>

  <node name="soundplay_node" pkg="sound_play" type="soundplay_node.py"/>

  <node name="xsens1" pkg="xsens_driver" type="mtnode.py">
    <param name="device" value="/dev/gandobot/xsens1"/>
    <param name="frame_id" value="xsens1"/>
  </node>

  <node pkg="robot_localization" type="ekf_localization_node" name="ekf_localization">
    <param name="frequency" value="30"/>
    <param name="sensor_timeout" value="0.1"/>

    <param name="odom0" value="/odom"/>
    <param name="imu0" value="/imu/data"/>

    <rosparam param="odom0_config">[false, false, false, false, false, false, true, false,
false, false, false, false]</rosparam>
    <rosparam param="imu0_config">[false, false, false, false, false, false, false, false,
false, false, false, true]</rosparam>

    <param name="odom0_differential" value="false"/>
    <param name="imu0_differential" value="false"/>

    <param name="odom_frame" value="odom"/>
    <param name="base_link_frame" value="base_link"/>
    <param name="world_frame" value="odom"/>

```

```
    <roscparam param="process_noise_covariance">[0.03, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
    0.0, 0.03, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0,
    0.0, 0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0,
    0.0, 0.0, 0.0, 0.03, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.03, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.00, 0.06, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.00, 0.0, 0.025, 0.0, 0.0, 0.0,
0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.00, 0.0, 0.0, 0.025, 0.0, 0.0,
0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.00, 0.0, 0.0, 0.0, 0.05, 0.0,
0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.00, 0.0, 0.0, 0.0, 0.0, 0.002,
0.0, 0.0,
    0.002, 0.0,
0.0, 0.004]</roscparam>
  </node>
</launch>
```

```
global_costmap:
  plugins:
    - {name: obstacles1,          type: "costmap_2d::ObstacleLayer"}
    - {name: inflater,           type: "costmap_2d::InflationLayer"}

  obstacles1:
    observation_sources: scanner1
    scanner1: {data_type: LaserScan, sensor_frame: sick1, clearing: true, marking: true, topic: /
scan_sick1}

  footprint: [[-0.2485,-0.254],[-0.2485,0.254],[0.2485,0.254],[0.2485,-0.254]]

global_frame: map_sick1
robot_base_frame: base_link
update_frequency: 5.0
publish_frequency: 2.0
width: 20.0
height: 20.0
origin_x: -10.0
origin_y: -10.0
resolution: 0.05
```

```
local_costmap:
  plugins:
    - {name: obstacles1,          type: "costmap_2d::ObstacleLayer"}
    - {name: inflater,           type: "costmap_2d::InflationLayer"}

  obstacles1:
    observation_sources: scanner1
    scanner1: {data_type: LaserScan, sensor_frame: sick1, clearing: true, marking: true, topic: /
scan_sick1}

  footprint: [[-0.2485,-0.254],[-0.2485,0.254],[0.2485,0.254],[0.2485,-0.254]]

global_frame: odom
robot_base_frame: base_link
update_frequency: 5.0
publish_frequency: 2.0
rolling_window: true
width: 2.0
height: 2.0
resolution: 0.05
```

TrajectoryPlannerROS:

max\_vel\_x: 0.45  
min\_vel\_x: 0.1  
max\_rotational\_vel: 1.0  
min\_in\_place\_rotational\_vel: 0.4

acc\_lim\_th: 3.2  
acc\_lim\_x: 2.5  
acc\_lim\_y: 2.5

holonomic\_robot: false

xy\_goal\_tolerance: 0.20  
yaw\_goal\_tolerance: 6.2



```
<launch>
```

```
  <node pkg="pandobot" type="line_detection_new" name="green_lines"/>
  <!--<node pkg="pandobot" type="line_speed" name="speed_lines"/>-->
```

```
  <node pkg="pointcloud_to_laserscan" type="pointcloud_to_laserscan"
name="pointcloud_to_laserscan">
    <param name="use_inf" value="true"/>
    <param name="use_concurrency" value="true"/>

    <param name="target_frame" value="cam_frame"/>
    <remap from="cloud_in" to="/cloud"/>
    <remap from="scan" to="/scan_lines"/>
  </node>
```

```
</launch>
```

```

#! /usr/bin/env python

import roslib; roslib.load_manifest('pandobot')          # manifest('my_pkg_name')
import rospy
import actionlib

#from chores.msg import *
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, PoseWithCovarianceStamped, PointStamped, Point, Quaternion, Twist,
Vector3Stamped
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from random import sample
from math import pow, sqrt

from std_msgs.msg import String, Header
from sound_play.msg import SoundRequest
from sound_play.libsoundplay import SoundClient

import tf

from collections import namedtuple          # http://stackoverflow.com/questions/35988/c-like-structures-in-python

import geodesy.utm          # check this
from geodesy.utm import UTMPoint

import sys
sys.path.insert(0, '/home/garrick/piksi_firmware/scripts')
import sbp_piksi as sbp
import serial_link
import time

from sensor_msgs.msg import Imu

import math
# -----
# -----
class GoalAndStatus:
    def __init__(self):
        self.isComplete = False          # if goal is no longer to be considered in the goal
        queue (i.e. done for this run)

        self.goal = MoveBaseGoal()
        self.goal.target_pose.pose.orientation = Quaternion
        (*tf.transformations.quaternion_from_euler(0.0, 0.0, 0.0))          # goal orientation is irrelevant
        self.goal.target_pose.header.frame_id = "base_NWU"          # this is default. if visited
        == true, then will change to map frame

        self.visited = False
        self.sickl_map_visited_coord_x = 0
        self.sickl_map_visited_coord_y = 0
# -----
# -----
# relevant data structures available in sbp.pdf in piksi_firmware/libswiftnav/docs/sbp.pdf
def cb_pos_LLH(data):
    global posLLH
    posLLH = sbp.PosLLH(data)

    return
# -----
def cb_baseline_NED(data):
    global baselineNED
    baselineNED = sbp.BaselineNED(data)

    return
# -----
def cb_done(status, res):
    global goalIt
    str_goalIt = str(goalIt)

    if (status == 3):
        fb_str = "achieved goal " + str_goalIt

```

```

        print fb_str
        soundhandle.say(fb_str, "voice_kal_diphone")

    else:
        fb_str = "goal failed so moving to next goal"
        print fb_str
        soundhandle.say(fb_str, "voice_kal_diphone")
        # don't say is complete... will come back to it.

# check if finished all goals
if (check_finished() == False):          # if not finished...
    inc_goalIt()                          # move on from this goal (i.e. if it failed

last)

    while (goallist[goalIt].isComplete):
        inc_goalIt()

    send_goal(goalIt)

    return
# -----
def cb_active():
    print "goal sent to the nav stack..."
    return
# -----
# -----
def check_finished():                    # helper function -- checks through goal array and determine if all
goals are complete
    isFin = True
    for x in range(0, numGoals):
        if (goallist[x].isComplete == False):
            isFin = False
            break

    return isFin                          # return true if all goals are finished
# -----
def inc_goalIt():                        # helper function to increase global goal array iterator and loop back
to the start
    global goalIt
    if (goalIt == (numGoals - 1)):
        goalIt = 0
    else:
        goalIt = goalIt + 1

    return goalIt
# -----
def send_goal(goalNum):                  # helper function to send goal @ position in goal array
    goallist[goalNum].goal.target_pose.header.stamp = rospy.Time.now()
    client.send_goal(goallist[goalNum].goal, done_cb = cb_done, active_cb = cb_active)
# -----
# -----"GLOBAL VARIABLES"-----
# http://stackoverflow.com/questions/9612484/locking-global-variables-under-the-threading-module-of-
python
posLLH = 0          # lat, long, height coords...
baselineNED = 0    # relative coords from base station (for debugging)

goalIt = 0          # global iterator for goal array
# -----
# -----
if __name__ == '__main__':
    rospy.init_node('goal_planner_py')

    # connect to gps and enable relevant callbacks
    link = serial_link.SerialLink(port="/dev/ttyUSB0")          # eventually set up udev so constant
dev name. @tm must plug in piksi before anything else to steal ttyUSB0
    #link.add_callback(sbp.SBP_POS_LLH, cb_pos_LLH)
    link.add_callback(sbp.SBP_BASELINE_NED, cb_baseline_NED)

    rospy.sleep(3.0)          # sleep so make sure we have something in the position variables

    soundhandle = SoundClient()

```

```

tf_listener = tf.TransformListener()

client = actionlib.SimpleActionClient("move_base", MoveBaseAction)
client.wait_for_server(rospy.Duration(60))
rospy.loginfo("Connected to move base server...")

# start inputting goal information
numGoals = 1
goalList = list()

# initializing a little. should use an x instead of a goalIt
while goalIt < numGoals:
    goalList.append(GoalAndStatus())           # our object to store goal information
    goalIt = goalIt + 1

goalIt = 0

'''
surveyed points are:
J01 lat  = -31.980692
J01 long = 115.817406

J02 lat  = -31.980302
J02 long = 115.818665

smaller long (J01) should be near ECM
J02 is near law and commerce

they are both on the concrete plate corners and marked with a cross

general method...
* publish transform base_NWU <-> base_link or top_plate or even sick1 (since on top of it...)
* calculate UTM(dest) - UTM(current) and send with base_NWU as frame
    * +ve northing goes to +ve north (x), +ve easting goes to negative west (y)
    * thinking this gets transfered all the way up to map_sick1 frame
* for now assume GPS always on and as accurate as ever
'''

J01_lat    = -31.980692
J01_long   = 115.817406

crick_lat  = -31.980624           # from google maps. haven't verified yet.
crick_long = 115.818214

J02_lat    = -31.980302
J02_long   = 115.818665

# test by sending to base-station, then back to original position
current_NED = baselineNED

'''
pt1_NWU = PointStamped()
pt1_NWU.point.x =                # because in [mm]
pt1_NWU.point.y = -(-current_NED.e/1000)   # double negative because east to west...
pt1_NWU.point.z = 0
pt1_NWU.header.frame_id = 'base_NWU'
#pt1_NWU.header.stamp = rospy.Time.now()

pt1_map_sick1 = tf_listener.transformPoint('map_sick1', pt1_NWU)
'''

goalList[0].goal.target_pose.header.frame_id = "base_NWU"
goalList[0].goal.target_pose.pose.position.x = -current_NED.n/1000
goalList[0].goal.target_pose.pose.position.y = -(-current_NED.e/1000)

send_goal(goalIt)

rospy.spin()    # Blocks until ROS node is shutdown.

```

```

#! /usr/bin/env python

import roslib; roslib.load_manifest('pandobot')          # manifest('my_pkg_name')
import rospy
import actionlib

#from chores.msg import *
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, PoseWithCovarianceStamped, Point, Quaternion, Twist, Vector3Stamped
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from random import sample
from math import pow, sqrt

from std_msgs.msg import String
from sound_play.msg import SoundRequest
from sound_play.libsoundplay import SoundClient

import tf

from collections import namedtuple          # http://stackoverflow.com/questions/35988/c-like-structures-in-python

import geodesy.utm          # check this
from geodesy.utm import UTMPoint

import sys
sys.path.insert(0, '/home/garrick/piksi_firmware/scripts')
import sbp_piksi as sbp
import serial_link
import time

from sensor_msgs.msg import Imu

import math
# -----
# -----
def cb_magnetic(data):
    #print data.orientation
    #global posYaw
    #(roll,pitch,posYaw) = tf.transformations.euler_from_quaternion([data.orientation.x,
data.orientation.y, data.orientation.z, data.orientation.w])
    #print posYaw # works. outputs -pi to 0 (hopefully north) to pi
    #print data.vector.x
    #print data.vector.y
    #print data.vector.z

    invTan = math.degrees(math.atan(abs(data.vector.y/data.vector.x)))

    if (data.vector.x > 0 and data.vector.y > 0):
        out = invTan

    if (data.vector.x > 0 and data.vector.y < 0):
        out = -invTan

    if (data.vector.x < 0 and data.vector.y > 0):
        out = 180.0 - invTan

    if (data.vector.x < 0 and data.vector.y < 0):
        out = -180.0 + invTan

    #print out

    br.sendTransform((0.135, 0.0, 0.25),          # where ever the gps is mounted...
                    tf.transformations.quaternion_from_euler(0, 0, out*math.pi/180),
                    rospy.Time.now(),
                    "base_NWU",          # child
                    "top_plate"          # parent
                    )

    return
# -----
# -----

```

```
if __name__ == '__main__':
    rospy.init_node('base_NWU_tf_publisher')
    rospy.Subscriber("/magnetic", Vector3Stamped, cb_magnetic)
    br = tf.TransformBroadcaster()

    # pub tf everytime get a reading... so just block here...
    rospy.spin()    # Blocks until ROS node is shutdown.
```

```

#! /usr/bin/env python

import roslib; roslib.load_manifest('pandobot')          # manifest('my_pkg_name')
import rospy
import actionlib

#from chores.msg import *
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, PoseWithCovarianceStamped, Point, Quaternion, Twist, Vector3Stamped
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from random import sample
from math import pow, sqrt

from std_msgs.msg import String
from sound_play.msg import SoundRequest
from sound_play.libsoundplay import SoundClient

import tf

from collections import namedtuple          # http://stackoverflow.com/questions/35988/c-like-structures-in-python

import geodesy.utm          # check this
from geodesy.utm import UTMPoint

import sys
sys.path.insert(0, '/home/garrick/piksi_firmware/scripts')
import sbp_piksi as sbp
import serial_link
import time

from sensor_msgs.msg import Imu

import math
# -----
# -----
class GoalAndStatus:
    def __init__(self):
        self.isComplete = False
        self.goal = MoveBaseGoal()
        self.goal.target_pose.pose.orientation = Quaternion
(*tf.transformations.quaternion_from_euler(0.0, 0.0, 0.0))
        self.goal.target_pose.header.frame_id = "base_NWU"          # this will depend...
        self.lat = 0
        self.long = 0
        self.UTM = UTMPoint()
        self.finalMapPoint = 0          # once 100% got to goal. set this. for next run... in
the map frame of ref...
# -----
# -----
def cb_pos_LLH(data):
    global posLLH
    posLLH = sbp.PosLLH(data)

    return
# -----
def cb_done(status, res):
    global goalIt
    str_goalIt = str(goalIt)

    if (status == 3):
        fb_str = "achieved goal " + str_goalIt
        print fb_str
        soundhandle.say(fb_str, "voice_kal_diphone")

    else:
        goalList[goalIt].isComplete = True

        fb_str = "fuck goal " + str_goalIt
        print fb_str
        soundhandle.say(fb_str, "voice_kal_diphone")
        # don't say is complete... will come back to it.

```

```

# check if finished all goals
if (check_finished() == False):           # if not finished...
    inc_goalIt()                           # move on from this goal (i.e. if it failed
last)
    while (goalList[goalIt].isComplete):
        inc_goalIt()

    send_goal(goalIt)

return
# -----
def cb_active():
    print "goal sent to the nav stack..."
    return
# -----
def cb_timeout(event):
    global goalIt
    goalIt = goalIt + 1
    # will look through goal array and choose a new goal...
    soundhandle.say("timed out", "voice_kal_diphone")
    return
# -----
# -----
def check_finished():
    isFin = True
    for x in range(0, numGoals):
        if (goalList[x].isComplete == False):
            isFin = False
            break # break out of the for loop

    return isFin
# -----
def inc_goalIt():
    global goalIt
    if (goalIt == (numGoals - 1)):
        goalIt = 0
    else:
        goalIt = goalIt + 1

    return goalIt
# -----
def send_goal(goalNum):
    goalList[goalNum].goal.target_pose.header.stamp = rospy.Time.now()
    client.send_goal(goalList[goalNum].goal, done_cb = cb_done, active_cb = cb_active)
# -----
# -----"GLOBAL VARIABLES"-----
# http://stackoverflow.com/questions/9612484/locking-global-variables-under-the-threading-module-of-python
posLLH = 0      # will hold the correct object. don't know how to init it to default.
bearing = 0     # direction of true north...
goalIt = 0
# -----
# -----

if __name__ == '__main__':
    rospy.init_node('goal_planner_py')

    # connect to gps and enable callback
    #link = serial_link.SerialLink(port="/dev/ttyUSB0")
    #link.add_callback(sbp.SBP_POS_LLH, cb_pos_LLH)

    rospy.sleep(3.0)      # sleep so make sure outputting position solutions...

    soundhandle = SoundClient()      # why doesn't this need to be global?

    client = actionlib.SimpleActionClient("move_base", MoveBaseAction)      # or this? think only
if modifying. invoking methods must be fine
    client.wait_for_server(rospy.Duration(60))
    rospy.loginfo("Connected to move base server...")

    # goal related shit
    numGoals = 1

```



```
goalList = list()

# initializing a little. should use an x instead of a goalIt i think...
while goalIt < numGoals:
    goalList.append(GoalAndStatus())
    goalIt = goalIt + 1

goalIt = 0

...
surveyed points are:
J01 lat = -31.980692
J01 long = 115.817406

J02 lat = -31.980302
J02 long = 115.818665

smaller long (J01) should be near ECM
J02 is near law and commerce

it's marked with a cross...

general method...
* publish transform base_NWU <-> base_link or top_plate or even sick1 (since on top of it...)
* calculate UTM(dest) - UTM(current) and send with base_NWU as frame
    * +ve northing goes to +ve north (x), +ve easting goes to negative west (y)
    * thinking this gets transfered all the way up to map_sick1 frame
* for now assume GPS always on and as accurate as ever
    * ultimately
...

J01_lat = -31.980692
J01_long = 115.817406

crick_lat = -31.980624
crick_long = 115.818214

J02_lat = -31.980302
J02_long = 115.818665

#origin_posLLH = posLLH # gets set when not moving (@ the origin)
#origin_posUTM = geodesy.utm.fromLatLong(origin_posLLH.lat, origin_posLLH.lon)

goalList[0].lat = J01_lat
goalList[0].long = J01_long
goalList[0].UTM = (geodesy.utm.fromLatLong(goalList[0].lat, goalList[0].long))
# below gets set relative to base_NWU frame...
thisPosLLH = posLLH # just incase it changes...
goalList[0].goal.target_pose.header.frame_id = "base_NWU" # this is default...
goalList[0].goal.target_pose.pose.position.x = goalList[0].UTM.northing -
(geodesy.utm.fromLatLong(thisPosLLH.lat, thisPosLLH.lon)).northing
goalList[0].goal.target_pose.pose.position.y = -(goalList[0].UTM.easting -
origin_UTM.easting) # east to west

send_goal(goalIt)

rospy.spin() # Blocks until ROS node is shutdown.
```



## Post-Processing Service Based on RTX Technology

TrimbleRTX.com

Contributor: walkerj@crystal.com.au  
Reference Name: RTX\_44992580.14o  
Upload Date: 09/15/2014 15:20:22 UTC

Report Time Frame:  
Start Time: 09/15/2014 03:17:55 UTC  
End Time: 09/15/2014 05:30:00 UTC  
Observation File Type(s): RINEX  
Observation File(s): RTX\_44992580.14o  
Antenna:  
Name: TRMR10 NONE  
Height: 1.482 m  
Reference: Bottom of antenna mount  
Receiver Name: R10  
Coordinate Systems: GDA94 & ITRF2008  
Tectonic Plate: Australia  
Tectonic Plate Model: MORVEL56  
Processing Interval: 10 s

### Statistics

# Total Obs	# Usable Obs	# Used Obs	Percent
1586	793	782	98

### Used Satellites

# Total Satellites:	20
GPS:	G02 G04 G05 G06 G07 G08 G10 G13 G16 G20 G23 G28 G30
GLONASS:	R03 R04 R05 R13 R14 R15 R16

### Processing Results

GDA94 at Epoch 1994.0		
Coordinate	Value	$\sigma$
X	-2358330.127 m	0.016 m
Y	4874662.154 m	0.014 m
Z	-3358602.073 m	0.014 m
Latitude	31° 58' 50.52808" S	0.007 m
Longitude	115° 49' 2.63223" E	0.012 m
El. Height	-27.308 m	0.022 m

ITRF2008 at Epoch 2014.70		
Coordinate	Value	$\sigma$
X	-2358331.100 m	0.016 m
Y	4874662.302 m	0.014 m
Z	-3358601.031 m	0.014 m
Latitude	31° 58' 50.48983" S	0.007 m
Longitude	115° 49' 2.66315" E	0.012 m
El. Height	-27.388 m	0.022 m

### Report Information

Trimble RTX Solution ID: 4181575  
Solution Type: Static  
Software Version: 3.3.0.14133  
Creation Date: 09/15/2014 15:20:45 UTC

#### Disclaimer

Trimble Navigation Limited does not guarantee availability, reliability, and performance of the current RTX Post-Processing service and accepts no legal liability arising from, or connected to, the use of information on this document or use of this service.



## Post-Processing Service Based on RTX Technology

TrimbleRTX.com

Contributor: walkerj@crystal.com.au  
Reference Name: RTX\_44992582.14o  
Upload Date: 09/15/2014 15:07:36 UTC

Report Time Frame:  
Start Time: 09/15/2014 06:10:15 UTC  
End Time: 09/15/2014 08:18:00 UTC  
Observation File Type(s): RINEX  
Observation File(s): RTX\_44992582.14o  
Antenna:  
Name: TRMR10 NONE  
Height: 1.591 m  
Reference: Bottom of antenna mount  
Receiver Name: R10  
Coordinate Systems: GDA94 & ITRF2008  
Tectonic Plate: Australia  
Tectonic Plate Model: MORVEL56  
Processing Interval: 15 s

### Statistics

# Total Obs	# Usable Obs	# Used Obs	Percent
512	512	502	98

### Used Satellites

# Total Satellites:	16
GPS:	G05 G07 G08 G10 G15 G17 G26 G28 G30
GLONASS:	R04 R05 R06 R07 R09 R15 R16

### Processing Results

GDA94 at Epoch 1994.0		
Coordinate	Value	$\sigma$
X	-2358447.005 m	0.008 m
Y	4874630.499 m	0.015 m
Z	-3358565.135 m	0.011 m
Latitude	31° 58' 49.12558" S	0.005 m
Longitude	115° 49' 7.16488" E	0.006 m
El. Height	-27.865 m	0.019 m

ITRF2008 at Epoch 2014.70		
Coordinate	Value	$\sigma$
X	-2358447.979 m	0.008 m
Y	4874630.647 m	0.015 m
Z	-3358564.093 m	0.011 m
Latitude	31° 58' 49.08733" S	0.005 m
Longitude	115° 49' 7.19580" E	0.006 m
El. Height	-27.945 m	0.019 m

### Report Information

Trimble RTX Solution ID: 4181557  
Solution Type: Static  
Software Version: 3.3.0.14133  
Creation Date: 09/15/2014 15:07:56 UTC

#### Disclaimer

Trimble Navigation Limited does not guarantee availability, reliability, and performance of the current RTX Post-Processing service and accepts no legal liability arising from, or connected to, the use of information on this document or use of this service.