WatBot: Design Report

University of Waterloo
Stuart Alldritt, Tim Vieregge, AJ Rosewarne
Steven Waslander (stevenw@uwaterloo.ca)

ABSTRACT

The Intelligent Ground Vehicle Competition is a competition designed to challenge the capabilities of autonomous vehicles. This report details the design and implementation of an autonomous navigation platform designed by the University of Waterloo Robotics Team to compete at IGVC.

The robot, WatBot, is an omni-directional drive robot designed for outdoor use. By utilizing wheels which can swivel 360 degrees, motion in any direction can be achieved. WatBot was constructed in 2013 and has been refined and improved since then.

WatBot is powered by a custom battery management system, designed to improve the performance and lifespan of its onboard lithium ion battery. The onboard battery provides power directly to high power components as well as power to several onboard DC-to-DC power supplies for a wider range of available voltages.

The sensor platform onboard WatBot allows it to detect objects at great distance and with high precision. Overall, WatBot carries two laser scanners, three cameras, an inertial measurement unit and a GPS receiver. The data from these sensors is combined to provide a highly accurate picture of the world the robot is in.

The sensor data is processed by two powerful computers. One computer is dedicated to mapping and navigation tasks, while the other is dedicated to vision tasks. The vision computer is specially equipped to perform video processing operations more effectively.

A mixture of custom and 3rd party software is used as the navigation framework on WatBot. A global map, written by the UW Robotics Team, manages data over the entire IGVC course. A local map, provided by the open-source Robot Operating System, manages precise obstacle tracking close to the robot. Different path planning algorithms with varying levels of precision plat routes through each of the two maps. Custom vision processing software, written by the UW Robotics Team, collects images from the cameras on WatBot and produces obstacles for the navigation software to avoid (such as flags and white lines).

Overall, the WatBot system represents an autonomous navigation platform that is capable of traversing unknown terrain whilst avoiding physical objects and visual cues including flags and lines painted on grass.

INTRODUCTION

The 2014 Intelligent Ground Vehicle Competition provides the robotics community at the University of Waterloo with a unique opportunity to learn, practice and explore the design and implementation of advanced unmanned systems. As the proliferation of unmanned systems becomes a requirement in several different industries, competitions such as IGVC provide the testing grounds required for the team to prepare for participation in the unmanned systems industry. With this in mind the University of Waterloo team set out to design and build a versatile, dependable and novel solution to the autonomous systems problem posed by IGVC. The result, WatBot, is presented in this report.

Design Innovations

The team working on WatBot has created several distinct and innovative solutions to the problems presented by the IGVC challenge. These include:

- An integrated battery management system
- Omni-directional movement (holonomic drive)
- Combination of 360 degree point-cloud LIDAR and linear scanning LIDAR
- Integration of LIDAR and camera data
- Saliency and colorspace video analysis

These design innovations will be discussed later in the report.

Design Team

Table 1: Breakdown Of Team Activities By Member

Team Member	Department and Class	Design Task
Stuart Alldritt	Computer Engineering, Class 2016	System Integration, GPS Navigation, Mapping Software
Tim Vieregge	Computer Science, Class 2016	Vision Systems
James Servos	Masters of Mechatronics Engineering, Class 2014	Global Mapping
AJ Rosewarne	Electrical Engineering, Class 2014	Electrical systems, Battery Management system, Motion Controller Tuning
Sarah Elliot	Systems Engineering, Class 2014	Mapping, Navigation, Planning Systems
Andy Lee	Systems Engineering, Class 2014	Mapping, Navigation, Planning Systems
Ayman Nadeem	Systems Engineering, Class 2014	Vision Systems
Cat Mercer	Systems Engineering, Class 2014	Vision Systems
Daniel Giles	Mechatronics Engineering, Class 2013	Wheel Module Design
Steven Lao	Mechatronics Engineering, Class 2013	Wheel Module Design, Control Systems
Kent Stoltz	Mechatronics Engineering, Class 2013	Chassis Design

Sean Walsh	Mechatronics Engineering, Class 2013	Chassis Design
Ryan Turner	Accounting and Financial Management, Class 2014	Mechanical Design

The team that worked on WatBot was comprised of 13 undergraduate and graduate team members. These people were responsible for creating the hardware and software specifically for the IGVC challenge. Since the WatBot platform was designed for use in multiple robotics challenges, this section lists the team members who contributed in any way to portions of the hardware or software that will be used at IGVC. The breakdown of the team members is shown in Table 1.

The total man hours spent on WatBot is difficult to estimate due to the nature of the team organization. A rough estimate would place the number of man-hours around 2000.

Design Process

WatBot was designed by several teams over multiple years, with each year adding new software and hardware to the robot. The first team that worked on the robot was the Chassis Design team in 2013. The Chassis Design team created the design specification, CAD model, testing plan and executed the construction of the WatBot chassis. This initial version of WatBot was an excellent platform for software development, but had issues. Specifically, the motor drivers on WatBot were not sufficiently advanced to prevent motor burnout. A small team lead by AJ Rosewarne began upgrading the control software and hardware on the robot to support a new, more sophisticated motor controller which would protect the motors from overcurrent-induced burnout. Whilst this was occurring, a software team lead by James Servos designed the global mapping software that is used on the robot. Finally, a team lead by Stuart Alldritt and Tim Vieregge worked to implement software that was specific to IGVC, such as the white line detection and GPS navigation software.

In this fashion, the design of WatBot was incremental and extremely compartmentalized. Standard software messages and interfaces were drawn from the Robot Operating System and used to communicate between software components. Team and code management was provided through Assembla, where we use the Wiki and Git repository services extensively. Additionally, general team announcements are made through a Google Groups mailing list.

PHYSICAL DESIGN

This section of the report deals with the physical design of WatBot. This includes the design of the mechanical systems, construction process and electrical system. First, a breakdown of the cost of components is given under “Cost Breakdown”. A complete mechanical overview of WatBot is then given in the section titled “Mechanical Design”. An overview of the electrical systems on WatBot are given under “Electrical Design”.

Cost Breakdown

The overall cost of components that were used on WatBot is provided in Table 2.

Table 2: List of Costs for Building WatBot. All zero-cost items were donated.

Item	Actual Cost	Team Cost
NovaTel OEMV-3 GPS	$10,100.00	$0.00
SICK LMS111 LIDAR	$3,000.00	$1,500.00
Velodyne HDL-32E	$50,000.00	$0.00
3x PointGrey Firefly MV Camera	$1,200.00	$0.00
MicroStrain IMU	$2,000.00	$0.00
Vision Computer	$1500.00	$1500.00
Mapping Computer	$1200.00	$1200.00
Chassis	$3,000.00	$3,000.00
Motors and Motor Controllers	$2,320.00	$2,320.00
Misc. Mechanical Components	$1,000.00	$1,000.00
Batteries and Battery Chargers	$1,650.00	$1,650.00
Ethernet Router	$75.00	$75.00
TOTAL	$77,045.00	$12,245.00

Mechanical Design

WatBot was designed to be able to drive in any direction: it's wheels can pivot 360 degrees to allow it to move in any direction while keeping its front sensors pointed at a target. The final design of the wheel module is shown in Figure 1, which integrates several key features. All parts of the module are connected through a piece of structural tubing (red) to facilitate maintenance and hot swapping of modules. The rotation of the module is achieved by the rotation motor and custom gearbox (teal) which also allows for angular feedback. An integrated suspension system (pink) in the module allows for high speed traversal of rough terrain. In addition, the sheet metal design (green) increases the rigidity of the module in key directions while reducing the weight. The drive motor is integrated into the module with a custom gearbox (purple) which directly drives a pneumatic wheel.

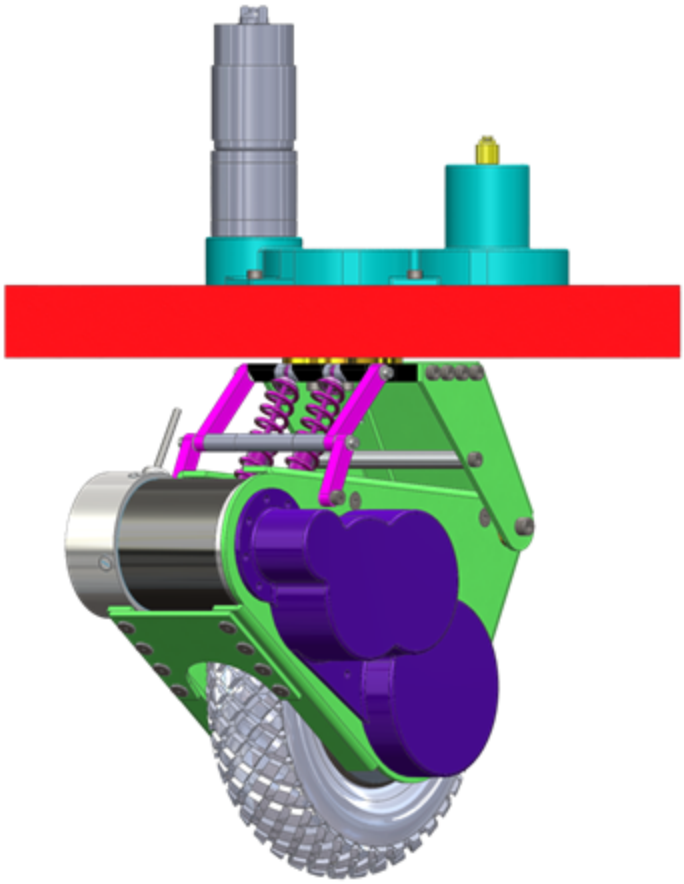

Figure 1: wheel module CAD design

To steer the module it was decided that a gear train should be used from the steering motor. The power is transferred to a shaft which rotates the module and to an additional gear attached to an absolute encoder for position feedback. In addition, the steering shaft is hollow to allow the drive motor wires to pass through the module, allowing for nearly continuous rotation of the module. To house the gears and allow for the mounting of the motor and absolute encoder, a nylon 3D-printed part is used to reduce weight and the number of parts required. A bronze sleeve bushing and thrust bushing are used to absorb the thrust and radial loads from the module.

Suspension has been integrated into the module allowing the chassis to quickly traverse rough terrain while reducing vibration. This is functionally critical due to the chassis being designed for a speed of approximately 2.2 meters per second. A suspension system typically used for remote control cars was selected and can be seen integrated into the module in Figure 1 above. Based on calculations and testing, the spring constant for each has been selected as 40 lbs per inch.

A 200mm diameter pneumatic wheel was chosen to drive the chassis. Pneumatic wheels were chosen due to being lightweight for their size, having inherent damping in their design, and having better traction than an equivalent foam wheel. The motor has been placed in the module in order to reduce the complexity and gears are used for power transmission due to the close proximity of the motor to the driveshaft. Aluminum gears are used to transfer motor power to the wheels to reduce weight: due to the

low stress and service life of the application, aluminium gears are sufficiently robust. To achieve the desired gear ratio of 11.4:1, a stage of 20:50 and 14:64 was used based on available parts:

$$N = \frac{50 * 64}{20 * 14} = 11.4 \quad (1)$$

An incremental encoder is attached to the back of the motor to allow for feedback for speed control. To house the gearbox, a 3D-printed nylon part is used to reduce the weight and number of parts, and shield the gears from contaminants, which can be seen above in Figure 1 (dark purple).

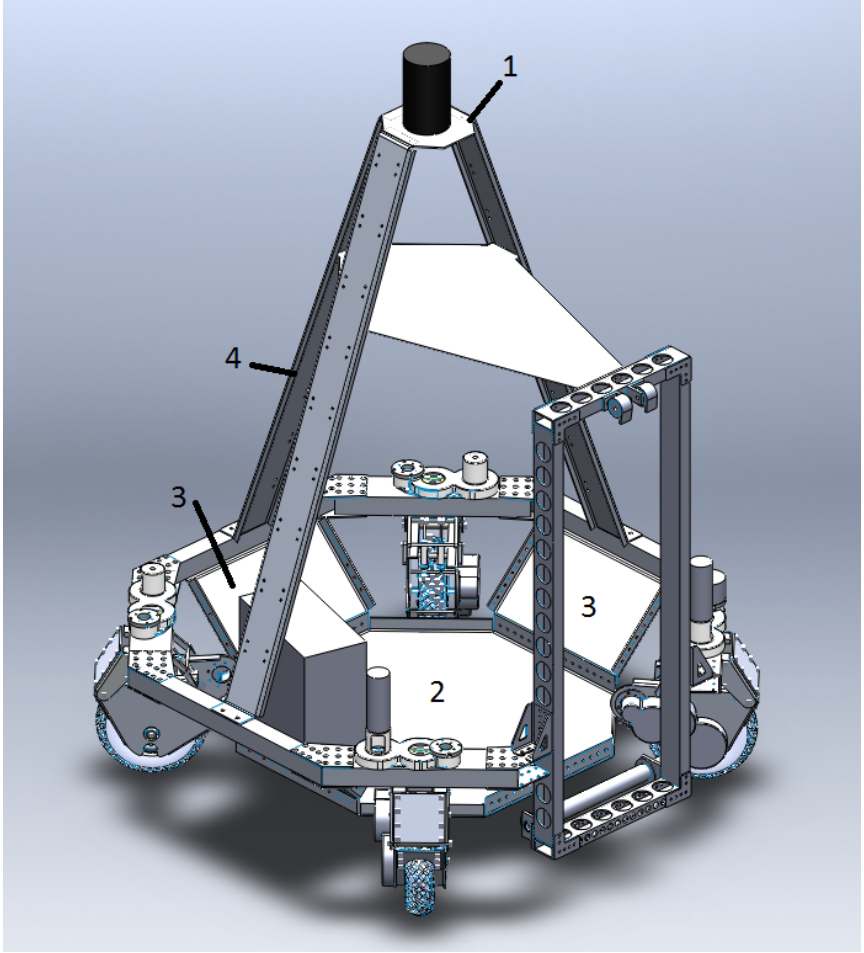

Figure 2: Chassis CAD design

The lower frame is made of bent 5052 0.125" sheet Aluminum. The lower plate (shown as 2 in Figure 2) allows space for the battery and other electronics and keeps it low for a low centre of gravity of the robot. The side panels (shown as 3 in Figure 2) provide space for motor drivers and relay switches. The frame is riveted together to provide rigidity and ease of assembly. The flanges on the side panels and bottom panel give the sheet metal parts rigidity as well as space for holes on the side panels for future part mounting.

The upper sensor tower (shown as 4 in Figure 2) raises the top plate (shown as 1 in Figure 2) to adequate height for the Velodyne LIDAR sensor. It also provides space for camera and other sensor mounting. It follows the same principle as the side panels with the flanges to add rigidity. The sensor tower bolts to ¼" tapped holes in the upper frame tubing for easy removal for transport and maintenance.

Power System Design

The power onboard WatBot is supplied by a Lithium Ion battery (specifically, a 12 volt, 100 Amp/hour LiFeMnPo battery). Lithium ion batteries have very specific requirements for charging and discharging safety. To that end, a custom Battery Management System (BMS) was designed to manage the charging, discharging and monitoring of the battery onboard WatBot.

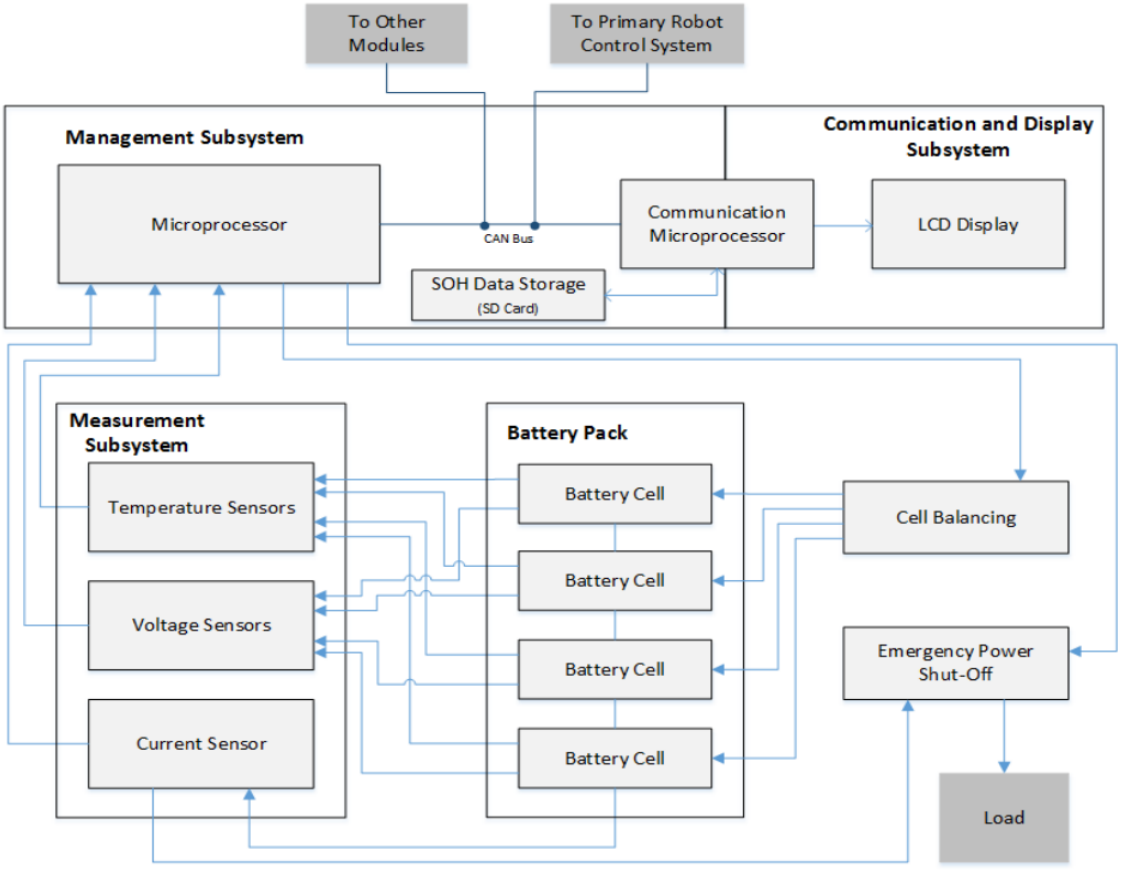

Figure 3: Custom BMS on WatBot

The system-level overview of this BMS is shown in Figure 3. The BMS provides this information to the primary robot control system. The BMS also monitors the battery over the lifetime of the pack and provides the needed information to determine the state of health of the battery pack. The BMS stores usage and charging data in order to predict the runtime remaining for the primary robot control system. As opposed to other battery management systems, this management system is integrated with the battery. The management system is designed to remain with the battery for the entire life of the battery. By integrating the battery with the management system, the management system can achieve more accurate statistics and store life time information for the most accurate system statuses.

The battery on WatBot was originally designed to allow for a 3 hour operational lifespan. Actual trial runs of the robot reveal that this lifespan is highly related to the level of activity of the robot: it can vary from between 2 hours to 4 hours depending on the amount of driving done and the weight of the payload carried by the robot.

Power from the battery is first distributed through 80 Amp fuses. These protect the battery against short-circuits which could cause damage to the battery or other robot systems. Once through the fuses, the power splits into two subsystems: motor power and control systems power.

The motor power system is connected to the wireless e-stop, then to the high-power motor control boards. The wireless e-stop triggers a high-power solid-state relay. This high-power relay will completely isolate the motors from the main power system: if the wireless e-stop is triggered, the motors cannot receive power from the battery. The high-power solid-state relay is also controlled by the physical e-stop: should the physical e-stop be pressed, the motor power is immediately cut.

The control system power is routed through the physical e-stop. If the e-stop button on the robot chassis is pressed, it triggers another high-power solid-state relay. This relay will immediately shut down all electronic systems on the robot, including all sensors, embedded microprocessors and main computers. This shutdown is not graceful and can cause data corruption in the main computers, which is the reason for the separation between motor and control system power. Control system power can be used directly from the battery on devices which contain an internal voltage regulator, however several devices on the robot require constant power that is of a different voltage than provided by the battery pack. To allow devices with various power requirements, WatBot is fitted with several programmable high-power DC-to-DC power supplies. These power supplies convert the battery voltage into the voltages required by the subsystems on WatBot.

SOFTWARE DESIGN

This section of the report deals with the design of the software on WatBot. First, the section “Control System Design” details the motor control system design. Next, the section “Software Platform” discusses the software and hardware platform on which the main control software runs.

Control System Design

The control system for WatBot is split into two halves down the centre of the robot. Each half of the robot is controlled by an Arduino Mega. These Arduinos communicate over a USB connection with the main control software running on the central computers.

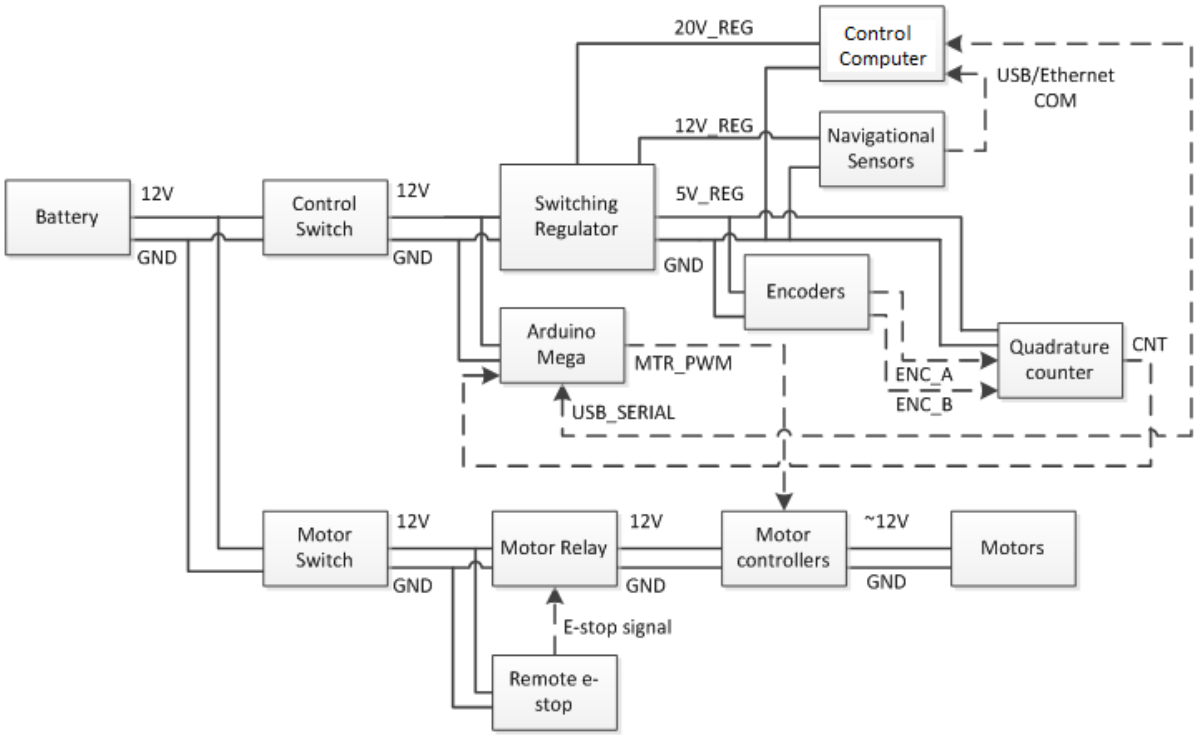

Figure 4: Overview of one of the Arduino motion control systems on WatBot

The Arduinos communicate with the main computers and listen for commands that indicate the desired driving direction. The computers send a velocity vector, which is converted by the Arduino firmware into an angle and speed for each wheel. The target angle and speed are fed into separate PID control loops: these loops read data from rotary encoders attached to each moving joint on the wheels as their source of feedback. This control system is illustrated in Figure 4.

The use of PID controllers ensures that none of the wheels of the robot will deviate from the assigned speed, even if they encounter obstacles or resistance. Similarly, the wheels will maintain their assigned angle regardless of the torque applied to the robot. This property allows the robot to climb hills and drive over small obstacles without special commands from the high-level navigation software, which would be difficult and time consuming to implement.

Once the PID controllers have calculated the appropriate motor power percentages, commands are sent to four RoboClaw motor controllers. These motor controllers communicate with the Arduinos over an I2C interface and allow precise control over a high power PWM signal that is sent to each motor.

Feedback about the status of each wheel is sent to the main computers by the Arduinos, as well as the status of the radio link to the remote control. This allows for quick diagnostics of the firmware and high-level electrical system from the main computers.

Software Platform

WatBot is powered by the Robot Operating System (ROS). ROS provides a platform on which to build robotic software and provides standardized interfaces for sensor drivers, a robust inter-process

communications framework, as well as a host of existing software that can be leveraged to build autonomous robots.

The software that runs WatBot is broken down into several individual modules which each run as a separate process. ROS provides highly robust publish-and-subscribe message passing between the modules which allows us to reduce the complexity of each module. Various teams worked on different software modules: for example, one team worked on designing the vision processing module while another team designed the global mapping module. Since the interfaces between the software components were both easy to define and program, the creation of these individual modules was simplified.

There are two computers on WatBot, both housed in a custom-built 3D printed enclosure mounted in the centre of the robot. This enclosure contains a gigabit ethernet switch which allows for high-speed communications between the two computers as well as other ethernet enabled sensors on the robot, such as the SICK LIDAR. Both computer run Ubuntu Linux with ROS installed.

The first computer, called the "mapping computer", handles all tasks related to mapping. It is fitted with an Intel Core i7 processor with 16GB of RAM and runs off a 128GB SSD. It features 2 gigabit ethernet ports: one is connected to the gigabit switch, the other is connected directly to the Velodyne HDL-32E LIDAR. This allows the mapping computer direct access to the high-bandwidth sensor and prevents the flood of packets from the Velodyne from degrading the connection quality over the gigabit switch. The mapping computer is responsible for all navigation decisions and as such is connected to a wide range of sensors and control components. It is connected directly to the Velodyne and SICK LIDAR, Microstrain IMU, NovaTel GPS and both of the control Arduinos.

The second computer, called the "vision computer", handles all tasks associated with processing the images from the three PointGrey Firefly cameras. This computer is outfitted similarly to the mapping computer but has the addition of an NVidia Gefore GTX 660 which is used for image processing tasks. The vision computer is connected directly to its three cameras sources via USB. Once the images from the cameras have been processed by the vision computer, they are converted into a virtual physical object and represented as a point cloud message which is sent over the ethernet connection to the mapping computer for integration into the map generation process.

NAVIGATION

This section of the report deals with the navigation software onboard WatBot. It includes details on the obstacle detection, path planning and localization software that was developed for IGVC.

Obstacle Detection and Mapping

The mapping system on WatBot is dependent on the open source software provided with ROS. The software was chosen because it is highly reliable and provides an existing implementation of an autonomous mapping solution. It reads data from the laser scanning sensors and uses it to generate a dynamic map of the environment. The map updates in real time, 5 times per second. There are two maps on the robot which are used for separate navigation tasks: the global map and the local map.

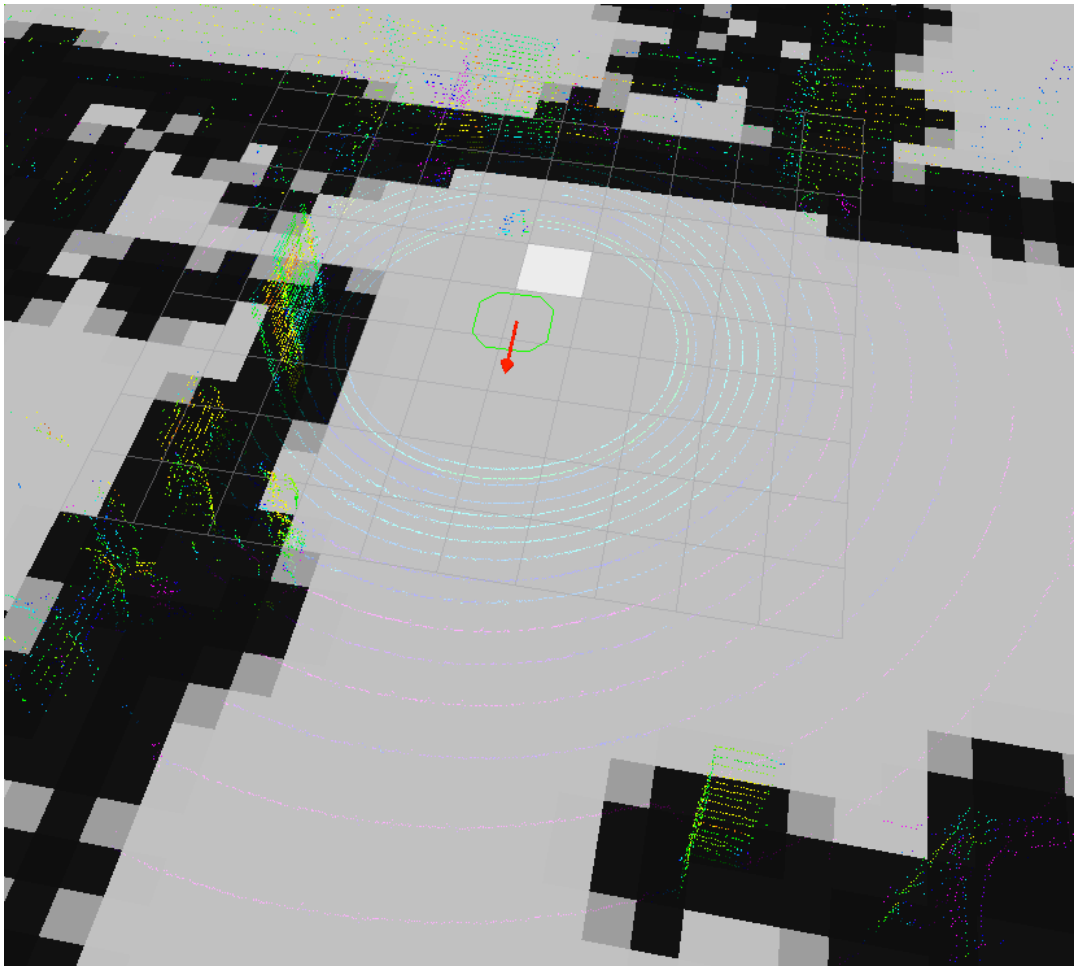

Figure 5: Global map of an indoor testing environment

The global map is a custom piece of software written by the UW Robotics Team. It takes data from the Velodyne LIDAR to produce a long-range map of the environment the robot is in. The map covers the entire field of the course with 0.5 meters^2 grid cells. Data from the Velodyne is split into groups that correspond to each cell in the grid. When enough points are grouped into one cell, the cell is marked “occupied” and the mapping software will not attempt to plan paths through it.

The global mapping software does not integrate each scan from the Velodyne into the map: instead, it computes an uncertainty value which describes how different the scan is from the current map. The global map will only begin integrating new data when the scan diverges sufficiently from the current map.

The global map is a multi-level surface map. This essentially means that the map is capable of supporting various heights and inclines without considering them obstacles. Whilst not essential for IGVC, this feature improves the robustness of the mapping software and allows for better mapping overall. The map is capable of detecting the plane of the ground it is currently over and will specifically exclude it from the obstacles inserted into the map.

Figure 5 illustrates the output of the global mapping algorithm, which produces a grid of cells that are deemed driveable. Overlayed on this grid is the output from the Velodyne laser scanner, which assists in debugging the mapping algorithm.

The global map will update over time to remove objects that are no longer present. The global map uses a simple ray tracing algorithm to clear cells when it receives data from a point farther along a linear path from the sensor. Because the sensor will be blocked by any physical object, receiving data from behind an existing object in the map must indicate that the object in the map is no longer present. To prevent false positives, the global map will not remove objects from the map immediately: it will wait for several scans that all provide the same information before attempting to raytrace an object out of the map.

White lines in the environment are converted into point clouds that are interpreted by the mapping code as physical obstacles. These white lines will not be cleared from the global map, which allows the robot to plan a route which avoids previously detected white lines that are no longer within its field of view.

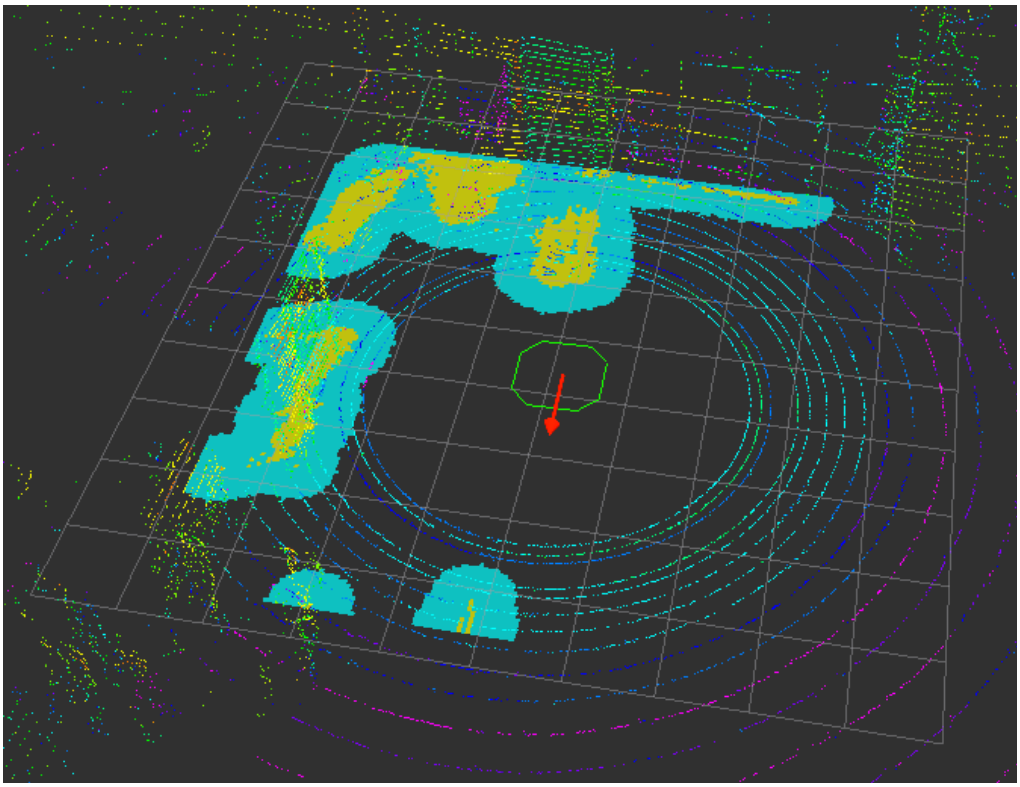

Figure 6: Local map of an indoor testing environment

The local map comes from the ROS navigation package. It is provided by the costmap_2d package, which implements a 3D voxel-based occupancy grid. The local map has significantly higher resolution than the global map (2 cm grid resolution) but is also significantly smaller (8 meters wide). Data taken from the sensors is inflated by the radius of WatBot to generate a map of areas the mapping algorithm cannot safely enter. This is due to the fact that the mapping algorithm simulates the movement of a single point through space, whereas the robot has a frame with a radius that cannot come into contact with obstacles.

Figure 6 shows the output from the local mapping algorithm. In it, detected obstacles are marked in yellow. The inflated exclusion zone around the obstacles is marked in blue. Overlayed on top of this map is the input data from the Velodyne and SICK LIDAR. This overlay has again been created to assist the debugging of the local mapping software.

The costmap_2d package draws information from 3 sensors: the Velodyne 3D LIDAR, the SICK 1D LIDAR and the virtual sensor created to represent white line data. This virtual sensor behaves like a 3D LIDAR. The sensors feed information into the local map differently: the Velodyne, SICK and virtual sensor are allowed to add information to the map, but only the SICK 1D LIDAR can remove data from the map. For the virtual sensor, this choice is inherent to its design: the virtual sensor only sees white lines and cannot provide information about the presence or absence of real physical objects. The Velodyne has a dead-zone around the sensor that extends to a range of approximately 2m from the robot. Whilst this dead-zone does not present a problem for global mapping, the local mapping software cannot tolerate objects being cleared from the map when they approach the robot. The SICK sensor was added to the design to accommodate the dead-zone of the Velodyne.

Localization

The mapping software is augmented by the robot localization software. This consists of two parts: the Extended Kalman Filter (EKF) localization package and the GPS localization package.

The EKF localizer uses the readings from the Microstrain IMU to calculate the position of the robot relative to it's start location. The software reads the IMU packets and extracts velocity, acceleration and orientation data. It uses an extended kalman filter to predict WatBot's motion and updates the robot velocity and position data that is visible to the navigation software. Additionally, the EKF localizer augments its predictions and corrects for IMU drift by applying a Point Cloud SLAM algorithm. Using this algorithm, the robot reads data from the Velodyne LIDAR and compares it to the existing global map. By comparing the global map data and current sensor reading, the robot can determine if it has strayed from the map coordinate frame, and can being to correct its predicted position on the map.

The GPS localization software allows WatBot to convert between GPS co-ordinates and co-ordinates in its local reference frame. When WatBot's mapping software turns on, the robot is at the (0, 0) coordinate facing into the negative X axis. The map orients around this coordinate frame, regardless of what direction the robot is facing relative to magnetic north. By taking a reading from the GPS and the Microstrain IMU's magnetometer at power on, the GPS localization software can determine the magnetic heading of the robot when the map was initialized, as well as the GPS coordinates that represent (0, 0). This software can then apply this data to convert between GPS waypoints and local map targets, which are sent to the navigation software as goals.

Overall, this system provides high accuracy positioning of the robot. Given a sufficiently precise target point and home location, the GPS software can position the robot to within an error of approximately 0.5 meters. The EKF localization software is even more precise: trials of the planner show an average full-loop error (error accrued over a path that starts and ends at the same location) to be approximately 22 cm.

Path Planning

WatBot has two navigation routines: the global and local planner. They both perform different functions, and are implemented using different algorithms.

The local planner is based on a Dynamic Window Approach planner. This planner looks for a goal position on the global plan within the bounds of the local map. It then attempts to plan a trajectory to that point using a simple polynomial curve. The local planner makes many attempts to plan a path with various acceleration and turning values, then ranks each plan based on a simulation of the robot's movement given that set of trajectory parameters. Each possible path is scored based on how close the path gets to obstacles, how close the robot gets to its goal, and maximum required speed[2].

The local planner also corrects the global plan by avoiding obstacles with higher precision than the global plan provides. If the global planner attempts to order the robot through an obstacle, the local plan will attempt to navigate around it, though with reduced efficacy. Once a plan has been selected, the required acceleration and direction commands are sent to the motion control subsystem.

The global planner is based on an implementation of Dijkstra's Algorithm[3]. When a new goal is created, the global planner uses the existing map information to plan a vague route to the goal, avoiding obstacles in its map. The global planner will abandon its plan and create a new plan if the robot becomes stuck in an area where it can't get out. The plan will also regenerate if new information about the environment becomes available (such as the detection of a white line on the previous global plan).

VISION

The following sections discuss the vision algorithms used on WatBot. This includes a description of the algorithm used to detect and categorize white lines, as well as a description of the software that processes the detected lines into obstacles for the navigation software.

Training

The first step in the vision algorithm is precomputation of the color spaces and thresholds for detecting the object of interest, namely the white lines and colored flags. To do this, representative pictures of the location are taken and each object of interest segmented out into a separate file. The separated objects are then each analysed for mean, min and max values in each component of the color space (ie red green and blue for RGB). These values are then compared across the different classifications of objects to minimise overlap, providing robustness to changes in conditions and optimal thresholding.

Image Processing

The video processing is separated in three pipelines, one for white line detection and two for red/blue flag detection. The images for each classification are then summed to produce a heat map, assigning each pixel a likelihood as part of the lane marking. For flag detection, the heat maps are then directly thresholded based on the likelihood of a pixel being part of that classification.

For white line detection the image of likely pixels goes through heuristic processing, removing objects which are unlikely to be one of the lines based on location or proximity to other colors (eg orange). After heuristic processing, the image goes through a Random Sample Consensus (RANSAC) algorithm to reject outlying white pixels, then attempts to fit parabolic models on continuous segments of likely pixels in the heat map[1]. The ability of RANSAC to reject outlying white pixels offers robustness over a straightforward regression fit.

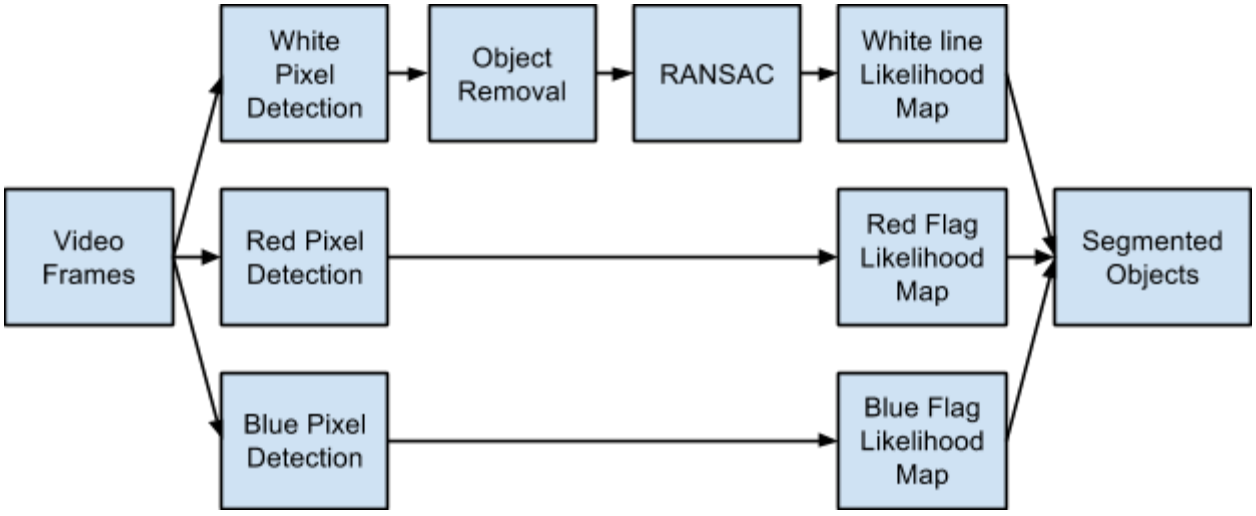

Figure 7: Illustration of video processing pipeline

Figure 7 illustrates the overall video processing pipeline. These stages run in parallel in separate processes, with each process reading the video data from the cameras. To improve processing time, some of the image transformation functions are performed on the GPU using the native OpenCL GPU Computation calls.

Data Conditioning

Data from the vision processing pipeline is converted into a grid of pixels that surround the robot. The pixels can have two values which represent the presence or absence of a white line on that position. This grid of pixels represents the location of all identified solid or dashed lines around the robot. The conversion of the images into this grid format is quite simple: the image processing pipeline outputs images which have been orthographically projected to produce a flat, birds-eye image. Since the position of the cameras is known, and the optical properties of the fish-eye lens on the cameras is also know, the physical dimensions of this orthographic projection can easily be calculated and stored as camera calibration information. The images that are outputted from the pipeline are black and white, where white represents the detected white lines. It is important to note that these output images are not simply any white object that has been detected. Instead, the images are the result of the image processing algorithms which specifically look for linear objects.

Both dashed and solid lines are detected in this manner. Dashed lines will not have sufficient space between dashes to allow the robot to pass, so due to the obstacle inflation routines in the local map the navigation software will not interpret the empty space between the dashed lines as a driveable.

Once the images have been converted to grid form, the data is stored in a global map of white lines. This map is continuously updated based on the data from the vision grid, and once the robot moves out of visual range of a certain area the last known line position is stored. This persistent storage assists in the global planning process: by applying the global map of white lines to the global map of physical objects, the global planning software will be able to generate long-range plans that avoid areas which are known to be inaccessible due to lines.

To assist in local planning, the data from the white line detection software is also fed into the local mapping software. This data takes the form of a virtual laser scan sensor, which uses the current white line detection grid to produce a laser scan message that indicates the presence of physical objects in the place that the white lines occupy. This allows the 3rd party local mapping software to account for the non-physical obstacles without significant modifications to the map generation code. Once the map considers the white lines obstacles, the normal planning code is able to avoid them.

CONCLUSION

The UW Robotics team has created a platform for mobile navigation over a period of 2 years. Through incremental design changes and refinements the design has been improved to the platform we use today. Over that time period, software has been developed to allow the robot to navigate in unknown environments and plan paths that allow it to avoid obstacles and navigate in tight spaces.

REFERENCES

(1) "RANSAC." *Wikipedia*. Wikimedia Foundation, 05 Nov. 2014. Web. 12 May 2014. <http://en.wikipedia.org/wiki/RANSAC>.

(2) “Navfn.” *Wiki.* Web. 12 May 2014. <http://wiki.ros.org/navfn?distro=hydro>.

(3) “Dwa_local_planner.” *Wiki.* Web. 12 May 2014. <http://wiki.ros.org/dwa_local_planner?distro=hydro>.