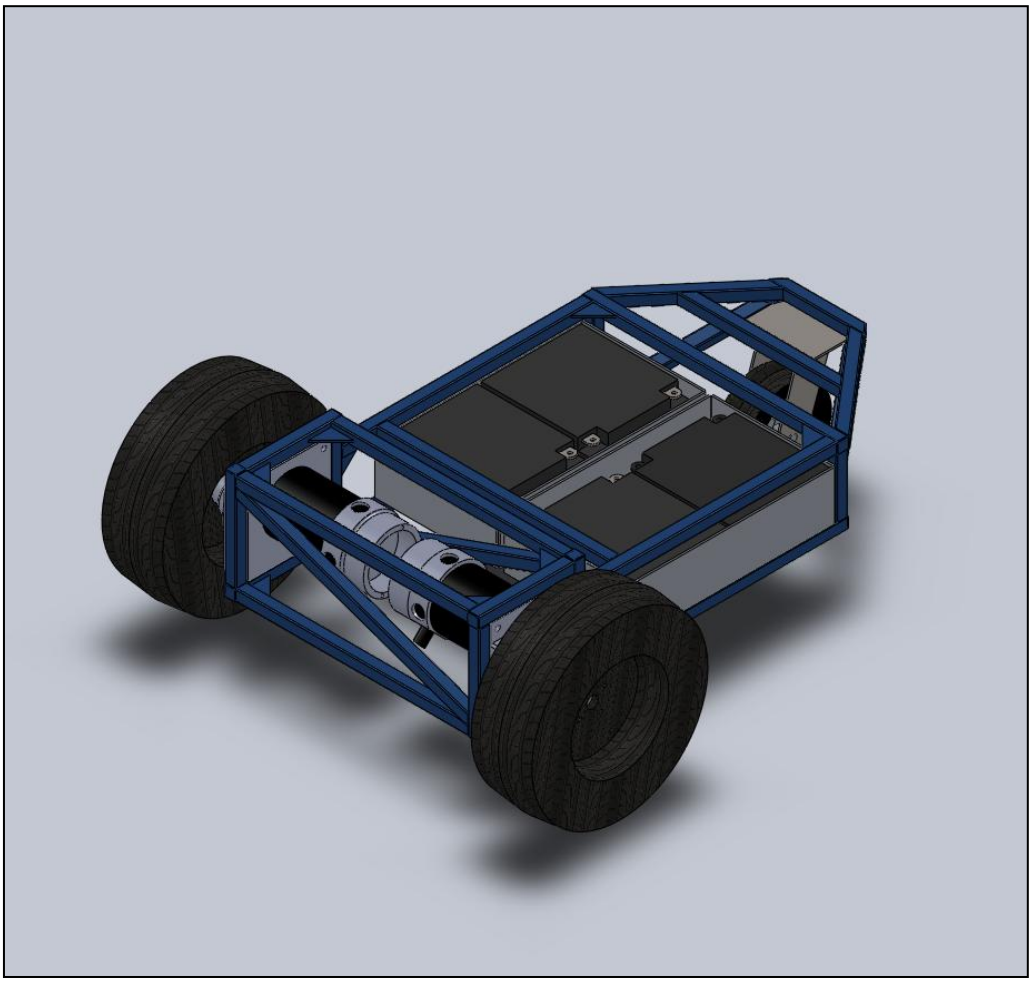
Al

Penn State Robotics Club



Faculty Advisor Statement

I, Sean N. Brennan, certify that the design and development of Al has been significant, and that each student performing this work is a registered student. This work as part of a graduate class project and as an extracurricular project represents a participation level equivalent to what would be awarded credit as a senior design project.

_____

Sean N. Brennan, Department of Mechanical Engineering, The Pennsylvania State University

Introduction

The Penn State Robotics Club's 2010 entry into the IGVC is an autonomous robot affectionately named “Al.” During the first two years of the competition which the team had competed, the team utilized pre-built platforms, either acquired commercially or secondhand from previous club competitions. This year the platform was designed and built by an interdisciplinary team of engineering students specifically for the IGVC competition. The challenges faced during the past two years helped the team formulate new strategies for avoiding, minimizing and quickly addressing problems which affect the robots performance, often occurring in basic and previously reliable systems.

This year Al is smaller, lighter, and more compact. The software team decided to re-utilize and significantly improve the Player interface algorithms developed to integrate communication with the sensor network and control systems.

This year's robot also contains a number of significant innovations. One notable feature is the student designed single-voltage power bus, which includes modular sensor nodes that regulate power as needed by the node. Sensor communication architecture is now standardized, utilizing an Ethernet bus for sensor communication. Fault tolerance and detection has also been designed to ensure more reliable sensor communications. These are only a few notable improvements representative of numerous others.

Platform Design Process

The design process for 2010 was much more involved and directed than in previous years. To ensure an efficient development cycle, scientific methods were used to define the characteristics of the robot and these equations created parameters that were used to acquire and operate the physical components of the robotic platform. Prodigious amounts of time were spent analyzing previous failures and developing new and innovative solutions.

The IGVC team is split into three logical sections: Platform, Sensing, and AI respectively held responsible for each of the different aspects of the robot's design. Table 1, below, outlines the team members which contributed to the 2010 entry, which aspects of development they were most involved in, and how many hours each spent on the project.

Name:	Team/Role:	Hours Spent:
Rich Mattes	Team Leader/Sensing Lead/AI	100
Anthony Cascone	Platform Lead/AI	100
Robin Pritz	Platform	40
Tony Jones	Platform	40
Xibei Ding	Platform/Sensing	80
Miles Frain	Platform	40
Jeremy Bridon	AI	40
Shawn Moffit	Sensing/Documentation	20

Table 1: Team time chart.

Design Parameters

The 2009 IGVC entry was a Segway RMP400, borrowed from Penn State's Networked Robotics and Systems Laboratory. The decision to borrow this robot was based on time constraints. With a whole software stack to write, it was not feasible to, given the team's experience and manpower to compete that year and create a new robot frame from scratch. Building on some criticisms of the 2009 entry (far too large, unable to be commercialized in its current state,) and leveraging two years of experience of best practices for the IGVC competition, the initial process began with laying out a set of quantitative performance goals and requirements which are listed below.

- 150lb maximum weight
- 8-10mph overall top speed
- Two motors on drive wheels, trailing caster
- Climb 40 degree inclines

- 3+ hour battery runtime
- Small size, as close to 3' x 2' as possible
- Support sustained sensor draw of 20W
- Zero radius turn ability
- 12" diameter wheels
- 1m stopping distance
- Hot-swappable batteries

From these requirements, design parameters were identified which would meet the desired performance characteristics as indicated in the list below:

- Motor Size (Torque, Power, Angular Velocity)
- Minimum Battery Capacity (Ah)
- Battery Chemistry (Lead Acid, Lithium-Polymer, etc)
- Ground Clearance
- CG Location
- Track width (distance between drive wheels)
- Wheelbase/Caster location
- Tire type/material

This list of design parameters was utilized to generate mathematical relationships between the performance goals and parameters. The performance requirement of "3 hours of battery run time" is dependent on battery chemistry, type of battery, and average power draw (of sensors and motors), all interdependent relationships. Maintaining a low center of gravity, for example, requires locating the weight as low to the ground as possible. Choosing four lead-acid batteries for the "3 hours of battery run time" influences frame design, in addition to the wheel and motor constraints needing to be close to the ground, this conflicts with the dimension related goal of a 3’x2’ frame size requirement. Choosing four heavy lead acid

batteries pushes the 150lb maximum weight performance goal. These dependencies require careful planning and design.

Motor selection

With the above interdependence problems in mind, motor selection was the team's initial focus. Torque and power parameters, run time, incline climbing goals, and maximum angular velocity for top speed goal achievement is calculated as follows.

For straight-line travel power requirements, the power requirements are

$$P_{req} = \left(\frac{C_{rr} \cdot F_n}{2}\right) \cdot v_{travel}$$

This formula derives from the instantaneous power formula, where P = F x v. This is a force balance under constant velocity, which means the force the motors must exert has to be the same as the force pulling the robot back due to rolling friction (C_{rr} * F_n). Each motor only needs to provide half of this force, hence the division by two. The C_{rr} for grass was determined to be approximately 0.055 to 0.065.

For maximum incline angle, building on the above force balance, in addition to the force of weight of the robot opposing motion up an incline, the following formula gives how much power needed to climb up an incline.

$$P_{req} = \left(\frac{C_{rr} \cdot F_n}{2} + \frac{F_w \cdot \sin\theta}{2}\right) \cdot v_{travel}$$

Torque requirements for the motor in both situations can be derived from the T=r x F formula, where r is the outer radius of the tires we've chosen, and F is the force the motors each need to overcome to move the robot. From a dead stop, the force is the static rolling resistance, and in the incline case, the force is the rolling resistance plus the weight of the robot opposing motion due to gravity.

Finally, the biggest constraint on vehicle top speed is the motor's maximum RPM. Finding the speed of the robot is fairly trivial using the motor RPM and tire diameter. For a 12“

tire diameter, a maximum RPM of at least 140 was needed for 5mph travel. This value gives the final requirement for the motor. A list of motors consisting of their rated torque, power, and maximum RPM based on the manufacturer spec sheets was created in order to find which motors were able to satisfy all of the requirements and to compare costs. Given budget constraints and availability of the NPC T74 motors from inventory – the decision was clear. These motors are much more powerful than required, so the budget constraint forced us to design around this core component.

Battery selection

In order to meet runtime requirements, the selected batteries must be able to provide enough power to drive the motors and sensors for the amount of time specified. Sensor datasheets provided power draw, which was modeled as a constant during robot operation. A power profile was generated for the motors using the above power requirements. It was estimated that the robot would spend 50% of its time in straight line travel, 50% of its time climbing inclines at an average of 10 degrees, and 10% of its time sitting idle while being programmed, troubleshot, and for debugging problems. With this power profile, the average estimated power draw can be applied to battery capacity as indicated by the following formula:

$$t = \frac{C}{I} \cdot 0.7$$

which is used to determine the runtime as a function of current draw and energy available from the batteries. Rearranging the formula, to solve for battery capacity in amp-hours (C) as a function of runtime (t)., the average current, (I), is determined by using P = IV on the power profile's power draw, at a voltage of 24V.

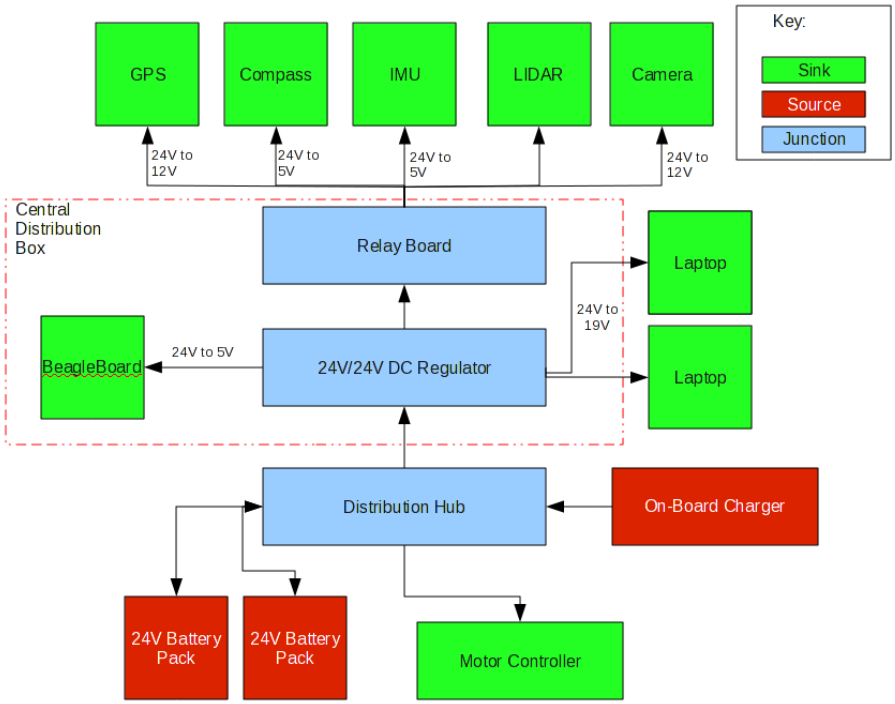


Figure 2: Power Distribution Layout

Using the above power profile percentages, an average power draw of 227.38 watts was calculated. Conclusion: in order to run for 3 hours, the robot has to have a battery pack capacity of 42 amp-hours.

Power Distribution System Design

The power distribution system should be simple, modular, and robust. Batteries should be hot swappable to minimize downtime, and one power source should supply the components. Last year's robot had separate power supplies for the drive systems and the sensors, which created requirements for two different kinds of chargers. The robot should also be able to charge

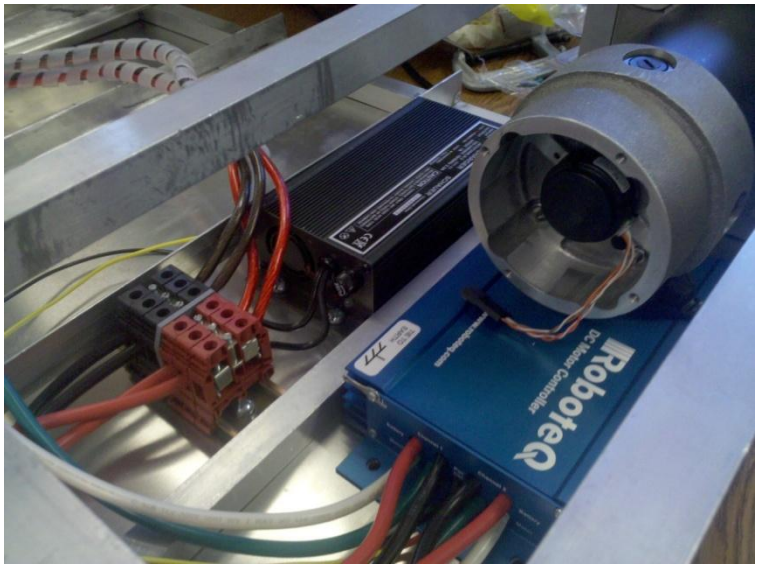


Figure 3: Distribution hub, on-board charger, motor controller, and motor with encoder

without having to disconnect power to the sensors.

Given the run time requirements listed above, budget and inventory, it was decided that a full 24V DC power system using two packs of two 12V lead acid batteries would be created. Each battery pack is outfitted with a high-current quick-disconnect connector, and both packs are connected in parallel to a distribution hub. This layout allows each battery pack to be removed and replaced individually, while the other battery pack remains connected to power the robot. The central distribution hub allows all major components to tap into the raw 24V DC power bus at a single central point. An on-board 24V DC charger provides current to this central distribution hub, allowing the vehicle to charge without turning off any subsystems.

Each battery is rated at 18 amp-hours, which means that four batteries have a combined capacity of 36 amp-hours in their current configuration. Based on power draw calculations, the pack should last 2.5 hours before it needs to be charged.

Low-level Control

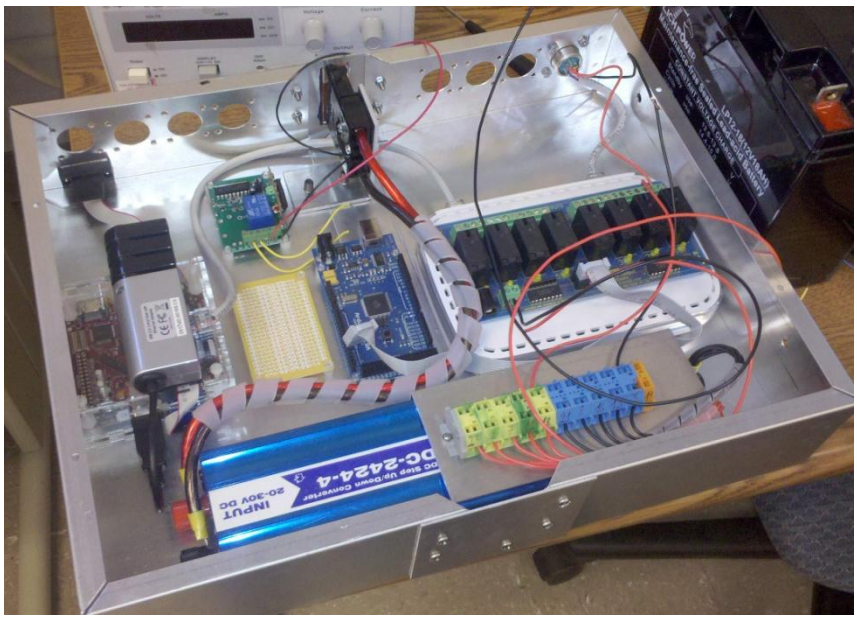


Figure 4: Testing the Central Distribution Box

Past years have shown that relying solely on laptops for control of the robot is a bad design paradigm, as laptops take significant time to boot up and configure. This means that the robot often not ready to move, and sometimes will take up to 5 minutes to enter that state. In the past the work for this was R/C car equipment that operated independently of the software controls, however this approach often required

manual intervention to switch between operating states. Given the goals of fast cold-start time, and minimal manual intervention, an embedded ARM based computer, the BeagleBoard was chosen. This board runs a full-fledged Linux-based operating system, which enables the software team to compile and run software on it with minimal effort. The BeagleBoard also features a fast boot-up time. The BeagleBoard was used to control the Motor Controller directly, which means that the Motor Controller is available on the network as soon as the BeagleBoard has completed its boot-up sequence. This significantly reduces time constraints that made previous platforms non-competitive and was a significant innovation.

Another innovation for this year is finely tuned sensor power control. Since a dedicated computer comes online at power-up, this could be used to control the entire robot platform startup procedure. The BeagleBoard is tied to a relay board, which can individually switch power off and on to each sensor node. This allows the BeagleBoard to power on and off all of the sensors on the robot through software. This functionality saves power, automates robot startup, and is also used to quickly recover from communication faults.

Emergency Stop Architecture

Emergency stop is a crucial part of the robot architecture. This functionality has three modes of actuation. First and foremost, a large pushbutton mounted on the chassis of the robot instantly turns off the motor controller, removing all power from the motors. The second line of defense is remotely-controlled relay, satisfying the remote-stop requirement of the rules. Finally, a software controlled relay is present, to allow the motor controller to be disabled by the controlling algorithm for safety (for instance, when the vehicle is charging.) The motor controller's stopped state is also monitored externally by the BeagleBoard, to assist in fault tolerance. If the BeagleBoard loses communication with the motor controller, but finds that the motor controller has been disabled externally (by a remote or physical stop switch,) the BeagleBoard will not attempt to re-establish communication until the motor controller is able to power on again.

Sensing Design process

Design Parameters

The team had much of its sensing equipment available from the past two years of competition. While software was utilized to communicate with the sensors, bringing sensors online and offline often took several configuration steps. Further, it seemed that each sensor had a specific connector to support, be it RS/232 DB-9, 3.3v or 5v TTL serial, or Ethernet. Each sensor also had its own power requirements, which meant facilities were needed to regulate and distribute power at 3.3V, 5V, 12V, and 24V DC. It was also difficult to track down communication faults, as the team rarely knew if it a bug was power issue, loose solder joint in a DB9 connector, or an outright sensor failure.

Software Architecture

Building on these deficiencies, a modular sensor node architecture was decided upon. It was mandated that, wherever possible, each sensor node must communicate over Ethernet, and that all sensor nodes shall accept a 24V DC power source, and regulate down to the sensor's required voltage as necessary. This 24V DC and Ethernet interface will be contained in a 10 pin locking aerospace connector, such that the same style of cable may be used to plug in to any sensor installed onboard. This architecture allows any computer on the network to initiate and host communications to a sensor, providing us with the ability to perform load-balancing between

Figure 5: Sensor node with example Ethernet/Power cable

each of the computers at will. It also allows the team to centrally regulate and distribute only one voltage, simplifying routing of power and signal wires to all of the different sensors.

Each sensor has an accompanying driver to manage connection and communication. To date, drivers would fail outright and crash if communication to any particular sensor was unavailable or interrupted. This wreaked havoc with software algorithms, which had no way of knowing when a sensor dropout took place. More often than not, once one of the sensor threads stopped working, it would cause a chain reaction that would crash the rest of the system. To alleviate this, fault tolerance was added to all of the software drivers. A driver model was established which dictated that drivers must periodically check to verify the connection to the sensor is active. If the connection is found to be inactive, the driver will go into a fault mode, where it will continuously check and try to re-establish a connection with the sensor. While in fault mode, the driver will publish a flag to all of its subscribers, notifying them that this sensor is down and readings from the sensor are not to be trusted. Drivers will also continue to broadcast the last known good information. Once connection to the sensor is re-established, the driver will clear the fault flag and begin publishing current information.

Part	**Maunfacturer/Model#**
IMU	Sparkfun 6DOFv4
LIDAR	SICK LMS291
GPS	Hemisphere A100 Smart Antenna
Camera	Point Grey Dragonfly
Compass	OceanServer OS5000
Motor	NPC T74
Motor Controller	Roboteq AX2850
Wheel Encoders	USDigital E6

Table 2: Discrete component list

All of the sensor Ethernet/power cables are plugged in to the central distribution box. This box provides a regulated 24V DC source from the batteries, which can be activated and deactivated by a relay. The relays are controlled by an Arduino microcontroller, which responds to commands from a BeagleBoard over USB. Since the power to each sensor can be

modulated in software as mentioned in the Low-Level Control section, a driver that detects a fault in sensor communication has the option to power-cycle the sensor while attempting to re-establish communication.

Software and Controls design process

The software and control framework once again uses the open-source Player framework, running on top of the Fedora and Angstrom Linux distributions. The open nature of these development tools and frameworks allows for great customization and control during the development process. It also allows the team to track down and fix bugs within the software frameworks quickly, and communicate with the upstream developers for fast resolution of problems that are outside team expertise. In the spirit of free and open software, this year's IGVC team played a large role in packaging and submitting the Player robot server, the Stage robot simulator, and the Gearbox serial/Ethernet communication library for inclusion in Fedora's package database. Now, all users of Fedora can install any of these software packages with a few mouse clicks, no complex compilation steps necessary.

Design Parameters

For this year's IGVC robot, the team wanted to build on the failings of last year's software design to create a system that's more tolerant to intermittent faults, and more user-friendly. To that end, a separate interface was created in each device driver that notifies any subscribing algorithms of a fault.

Map Generator

Map generation code collects the robot's current pose and all laser scans made by the robot. It updates an occupancy grid (represented with an image file) with laser range measurements, drawing obstacles where laser collisions are detected. When updating an area that already exists within the map, obstacles that already exist in the map are made to fade

away slowly. This accounts for drift errors, where the current position estimate is different from the position estimate when the map was first updated. The end result is that, if nothing is detected where an obstacle previously was, the recorded obstacle will eventually disappear. However, if new scans do agree with a previously found obstacle, the obstacle is treated normally.

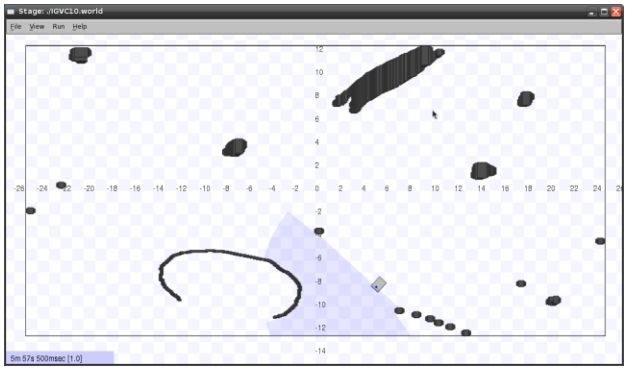


Figure 6: Simulated environment

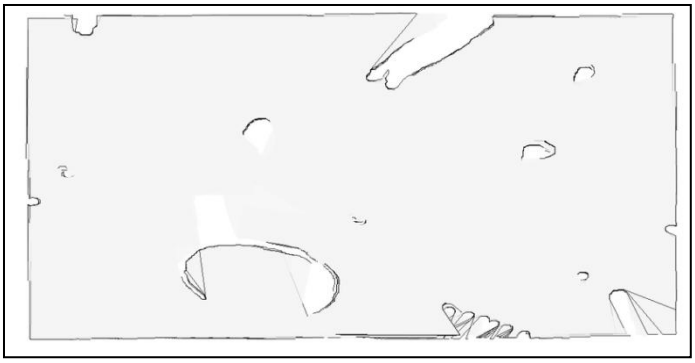


Figure 7: Map Generator output from simulated environment

Path Planner

Planning a path over mostly unknown terrain is not a very easy task. The path planner needed to work well with partially unknown environments, and it had to be fast to re-plan if the known obstacles suddenly change. For these reasons, the D* planning algorithm was chosen. Periodically, the path planner will request an area of map from the Map Generator, based on the robot's current position and goal pose. Once the Map Generator satisfies this request, a new array of goal points is calculated and made available to any clients that request them.

The Path Planner will try to come as close to any obstacles as possible thus shortening the distance to travel; however, this sometimes creates issues as it tries to plan through the pixels immediately surrounding known obstacles from the Map Generator. In order to avoid this, obstacles are dilated by the width of the robot, thereby adding a buffer, when they are transferred from the Map Generator to the Path Planner.

Image Processing

Image processing is traditionally a very difficult problem. The solution chosen, however, is quite simple. A color image is acquired from the camera, and each channel is instantly down sampled to 1 bit per pixel. The resulting image allows for only 8 pixel states. The image is then converted to grayscale, making an intensity image where the highlights consist of bright objects from the original image. At this stage, the desired obstacles stand out, but there is also a lot of small random noise. A median filter is run across the image, with a sampling area of 5x5 pixels, to eliminate outliers throughout the image, and then the image is thresholded so all pixels below white are turned black. All that remains at this point is a binary image with line markers in white, and background in black. The image is subjected to a perspective transformation, which converts the image such that it looks like a bird's eye view of the area in front of the robot. Lines are traced from the bottom center of the image, radially to the edges of the image, creating a virtual laser scan of the image. This laser scan is then treated by the algorithms as any other laser scan would, with laser hits representing real obstacles.

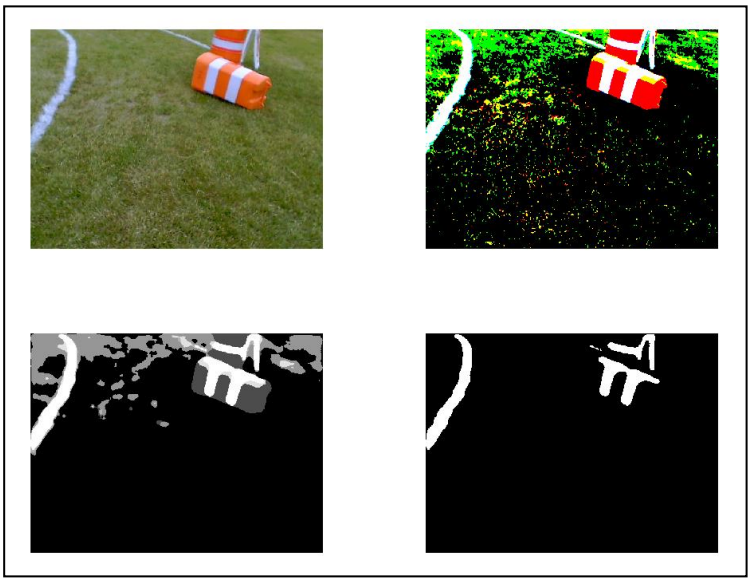


Figure 6: Image processing algorithm sequence

Localization

Localization of the robot takes place with several different sensors. Inputs from a 6 degree of freedom accelerometer and gyro, WAAS-enabled GPS receiver, magnetic compass, and wheel encoders mounted in the motors are fed into a Kalman Filter to keep an accurate estimate of the robot's position. The localization routine is fault tolerant: when a sensor raises a fault flag, the Kalman Filter will assign zero confidence to the incoming measurements, effectively blocking them out. The state will continue to update with the rest of the available

measurements, and when a faulty sensor comes back online, its measurements will start to be considered again.

Operational States

Navigation Challenge

The Navigation Challenge relies heavily on most of the algorithms. The provided set of waypoint GPS coordinates is loaded from a text file by the controller algorithm, and converted to the robot's internal coordinate system. The Map Generator starts and begins recording the map, and the Path Planner starts and awaits a command. Once the controller algorithm is ready, it will send the point of the first waypoint to the Path Planner. The Path Planner will request the necessary map tiles from the Map Generator, plan its path, and publish that data back to the main controller algorithm. The controller algorithm will then compare the points along the path with the vehicle's current estimated position (from the Localization algorithm described above), and using a proportional controller, attempt to navigate to the intermediate points in sequence until the final goal point is reached. Along the way, new waypoints are periodically requested, and if the planner decides to alter its route due to new obstacles, the controller algorithm will adjust accordingly.

Autonomous Challenge

The strategy for the Autonomous challenge is to take a reactive approach, which means that accurate sensor data is of great importance. The robot is set to try to maintain a nominal straight-forward velocity, until an obstacle is detected. Obstacle detection takes place using two lasers: the front-mounted SICK scanning laser rangefinder and the virtual laser provided by the Image Processing algorithm developed. The physical laser handles detections of barrels, trash cans, and fences, while the virtual laser detects white lines and represents them as obstacles. As the robot encounters obstacles on either side, it alters its path in an effort to avoid collisions.