PROMETHEUS: WPI's 2011 ENTRY TO IGVC

Authors:

Mehmet Ali Akmanalp
COMPUTER SCIENCE

Ryan Doherty
ROBOTICS ENGINEERING

Jeffrey Gorges
MECHANICAL ENGINEERING

Peter Kalauskas
COMPUTER SCIENCE
& ROBOTICS ENGINEERING

Ellen Peterson
ROBOTICS ENGINEERING

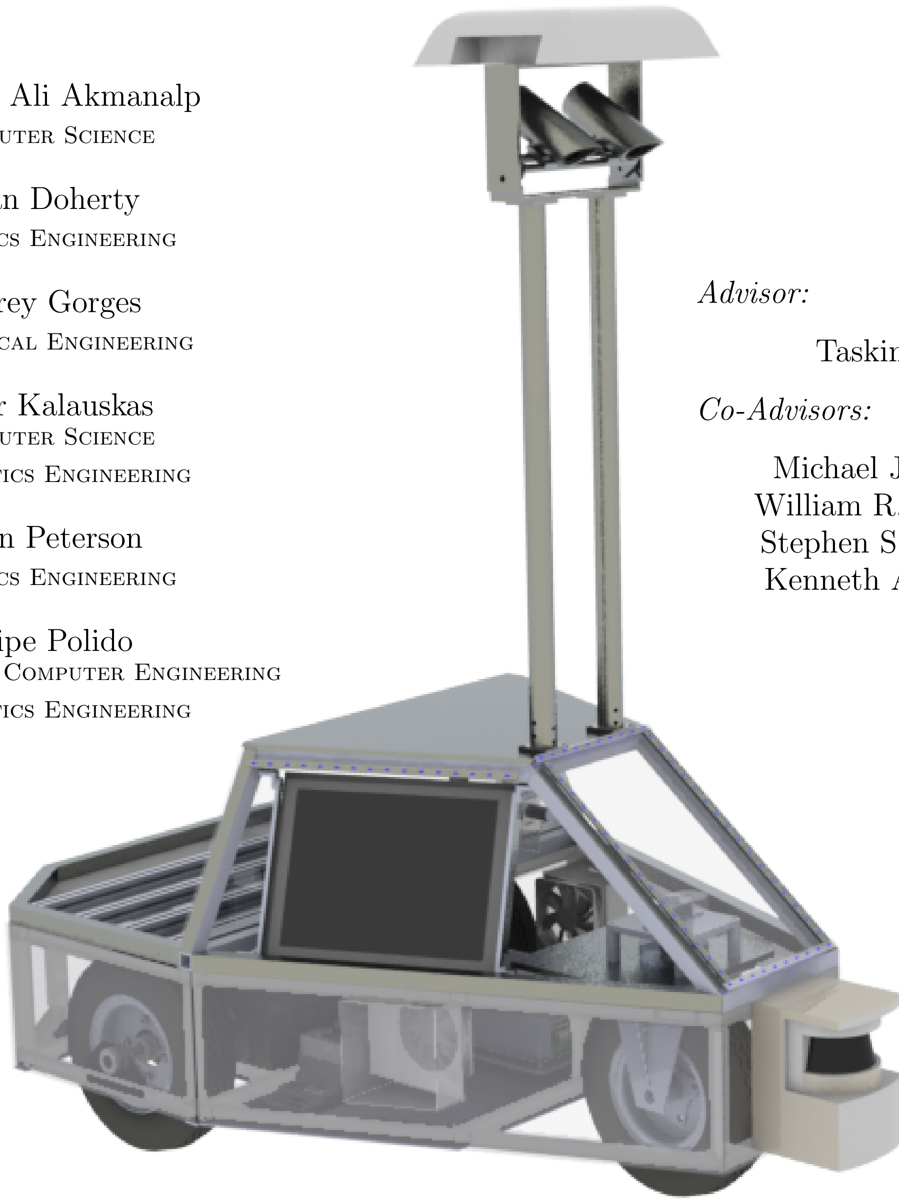
Felipe Polido
ELECTRICAL AND COMPUTER ENGINEERING
& ROBOTICS ENGINEERING

Advisor:

Taskin Padir

Co-Advisors:

Michael J. Ciaraldi
William R. Michalson
Stephen S. Nestinger
Kenneth A. Stafford



Faculty Advisor Statement:
I, professor Taskin Padir of the Robotics Engineering Program and Electrical Computer Engineering Department at Worcester Polytechnic Institute, Worcester do certify that the design and implementation of this vehicle has been credited to each team member for their work.

May 9, 2011

1 Introduction

2011 marks the second year of Worcester Polytechnic Institute's (WPI) Prometheus at the IGVC. The 2010 team was honored with the Rookie of the Year award for their great accomplishments as a first year entry. Although the team did well, there was still plenty of room for improvement. With a new team taking over for 2011, these areas were researched and improved to make Prometheus an even more competitive entry to the IGVC.

Table 1 illustrates the capabilities of Prometheus 2011 versus those of Prometheus 2010. Efforts were in large part focused on giving Prometheus the intelligence capabilities necessary to be competitive in the autonomous and navigation challenges. These can be summarized as line detection, localization, path planning, and waypoint navigation.

Other efforts were focused on improving the overall usability of the robot. Easy interaction with the robot greatly reduces the time required for developing, debugging, and testing. This is accomplished through the external interface, touchscreen, and visual cue.

Finally, improvements were made to the robot's chassis. Reconfigurable camera mounts were designed and fabricated in a way that allows camera height and angle to be adjusted based on weather conditions or visibility range. A cover redesign has given Prometheus a modular payload area, meaning various devices can be attached. It has also allowed for easier access to, and separation of, batteries and processing components.

Capabilities	**2010**	**2011**
Accurate Compass	✓	✓
Accurate DGPS		✓
External Interface		✓
Innovation	✓	✓
JAUS	✓	✓
Effective Line Detection		✓
Localization		✓
Modular Payload		✓
Obstacle Avoidance	✓	✓
Path Planning		✓
Reconfigurable Camera Mounts		✓
Touchscreen		✓
Visual Cue		✓
Waypoint Navigation		✓

Figure 1: Comparison of the capabilities of Prometheus 2010 and Prometheus 2011.

1.1 Team Organization and Design Process

The Worcester Polytechnic Institute team is comprised of six members with a variety of skills and specialties. To most effectively complete the tasks necessary to produce a qualifying and competitive IGVC robot, we split up the team as shown in the table below.

	Task						
Team member	Chassis Upgrades	Control System	Line Detection	Localization	Obstacle Detection & Avoidance	Sensors	Waypoint Navigation
Mali Akmanalp			✓				
Ryan Doherty				✓		✓	
Jeffrey Gorges	✓						
Peter Kalauskas					✓		✓
Ellen Peterson				✓		✓	
Felipe Polido	✓	✓				✓	

The team used an iterative design process in all aspects of the project, including software and mechanical design. This process is detailed in the figure below. Each design decision was made by using this process, as performed by a team member who is experienced in that area.

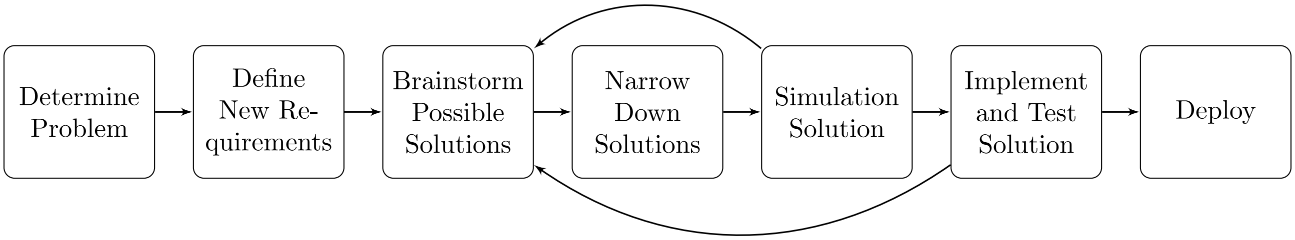
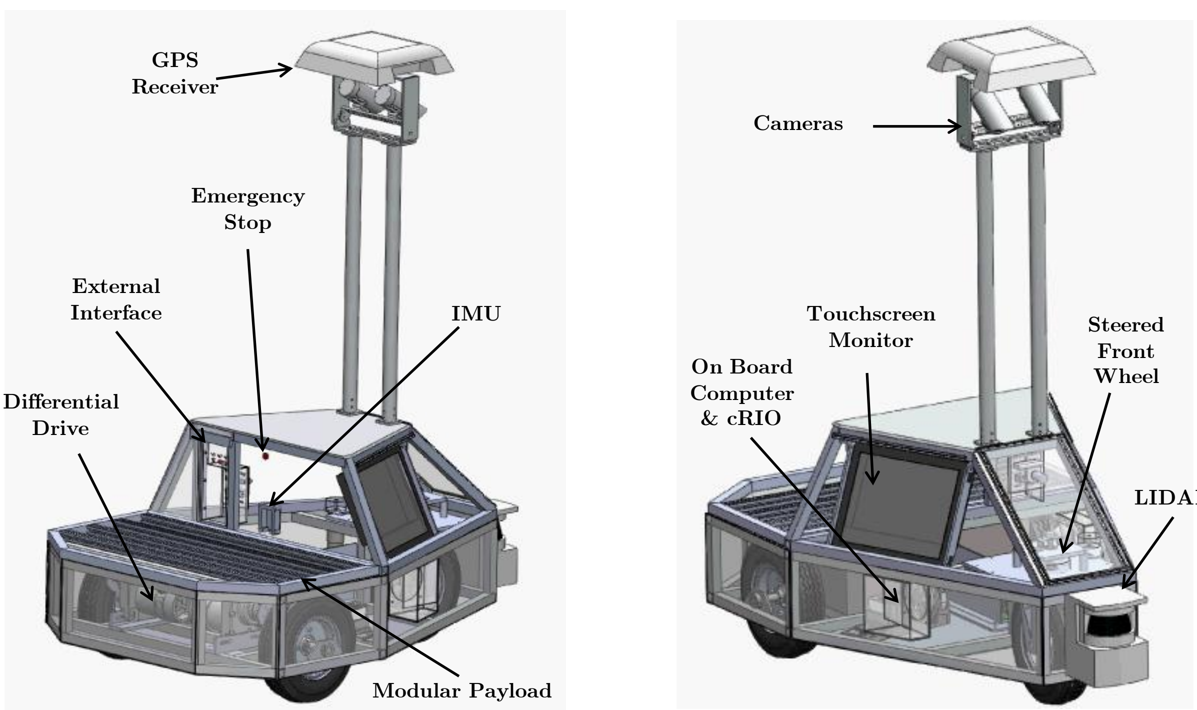


Figure 2: Design process for Prometheus 2011.

2 Prometheus Overview

Prometheus has a custom aluminum chassis with a reconfigurable payload area. Its differential drive system with a steered front wheel allows for high maneuverability and zero-point turning radius. Additionally, the vehicle's power comes from two interchangeable 12V 55Ah sealed lead acid batteries connected in series.



(a) (b)

Figure 3: Overview of the components of Prometheus 2011.

In order to accomplish the challenges presented by the IGVC, Prometheus has an array of sensors on board. These include a PNI Fieldforce TCM inertial measurement unit, a Trimble AG DGPS receiver with OmniStar HP subscription, 2x Point Grey Flea2 Firewire cameras, a SICK LMS-291 LIDAR rangefinder, and US Digital optical encoders. For the vast amount of data processing and low level interfacing Prometheus relies on a National Instruments cRIO unit and a custom built on board computer (Intel Core i7 Quad, 6GB DDR3 RAM). Moreover, the development team is able to interface with Prometheus through the included router's Wi-Fi, the built-in touch screen, a 6 channel remote control, and a centralized external interface panel.

3 Robot Structure

3.1 Usability Improvements

Prometheus main purpose is to be a functional autonomous ground vehicle. However, for the vast majority of time it operates under testing conditions. Most of the research and prototyping on Prometheus is done on one specific subsystems at a time, such as vision detection, mapping, or sensor fusion. Our team believes having an emphasis on the human-robot interaction portion of Prometheus ultimately makes the developing, debugging and testing phases happen significantly faster and smoother. In order to accommodate such a requirement we came up with a multifaceted solution. The improvements to Prometheus' chassis include:

- Creation of a robot external interface panel.
- Top cover redesign for environment isolation and sensor reposition.
- Addition of a touch screen monitor to the chassis for information transmission and control.
- Addition of a visual cue informing the current status of the robot.
- Smoother control under manual mode.

First, the need for an external interface occurs from the recurring interaction with the many different hardware components susceptible to weather damage. During debugging there is a need to directly interact with Prometheus' on-board computer, cRIO and power systems. Last year's design required the top cover to be openend any time a component had to be resetted or a wire connected, gratuitously exposing the many electronic components to a non-friendly environment. A central weatherproof external interface is a straighforward solution to this problem.

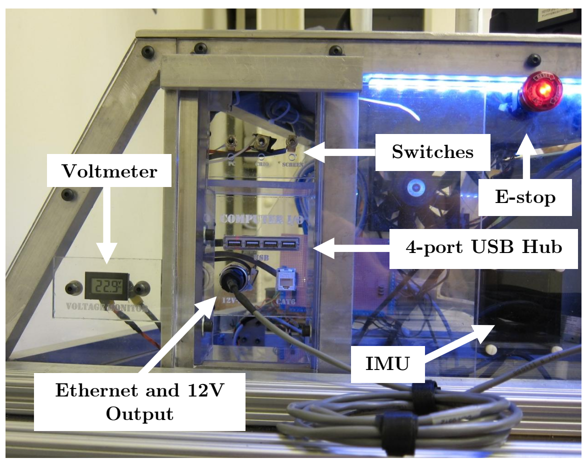


Figure 4: The external interface components, the IMU and the emergency stop button.

Second, prior to the IGVC rule's update our team was already brainstorming a form to quickly visualize Prometheus' state during development. The final design is a highly visible lighting system that changes color reflecting the state of the robot. Our team opted for adding strips of 12V weatherproof RGB LED; the strips are powered through N-Channel MOSFETS controlled by the NI cRIO. The use of an independet Pulse Width Modulation(PWM) signal for each channel allows for a practically infinite amount of color conbinantions.

The visual cue has become an essential component of Prometheus. The different colors are clearly associated with Prometheus' respective states,such as purple for manual mode,and blinking red for autonmous. The user can quickly glance at the vehicle and know what to expect from its behavior, as well be reminded of a low battery condition. Examples of the visual cue can be seen in Figure 5.

Third, Prometheus is made to drive autonomously, but having a manual remote control is essential during transportation, debugging, and developing phases. Last year setup involved using a joystick through a laptop connected wirelessly; this configuration suffered in range and had a significantly slow starting procedure. The solution was to switch to a RC

Figure 5: Different visual cue colors representing different states.

hobby system controller, the Futaba 6-channel 2.4GHz Transmitter and Receiver. The extra communication channels on the remote control allow us to bind a switch to the wireless emergency stop. The remote control works flawlessly at over 100m range. Another channel was binded to an autonomous/manual switch, featured that proved to be very practical during testing.

Lastly, the addition of a monitor to Prometheus' chassis greatly improved usability. A monitor allows real-time feedback of the many systems of the robot in a concise and informative way. After extensive research and comparison we decided on a 3M MicroTouch Display C1500SS (15") Serial. This solution is visible in direct sunlight, rugged, relatively small, DC powered and the touchscreen capabilities allow for a simple user interface. The monitor greatly improved field testing. The ability to quickly analyze the line detection, local map, and waypoint navigation proved to be crucial. This can be seen in Figure 5(a).

The usability improvements made on Prometheus proved to be very effective during field test. The enhancements allow quick testing, swapping, and comparing of different algorithms. It also permits us to identify software bugs and hardware malfunctions much faster than before, ultimately decreasing development time.

3.2 Safety

In order to comply with the IGVC rules Prometheus is equipped with a hardware emergency stop (E-stop) as well as a wireless E-stop. The hardware E-stop is connected directly to a set of relays that shut power off to all systems, the e-stop red button is conveniently located on the top back part of Prometheus.

The wireless E-stop is connected through the cRIO to a set of relays that shut off power to the three motor controllers, disabling any movement. The old design relied on a car alarm system as their wireless E-stop; as a spectrum frequency analysis demonstrated the old system operated within interference frequencies generate from the vehicle's motors and on-board computer, which substantially decrease the operating range. This year the wireless E-stop circuitry is connected to the 6-channel 2.4GHz Futaba controller, which operates far beyond the interference frequency levels and has naturally a much higher range and reliablity.

3.3 Electrical Design

Prometheus 24V DC power source comes from two 12V 55Ah Sealed Lead Acid batteries connected in series. Empirical testing has confirmed the theoretical values and shown that the vehicle can operate for one hour and a half in a fully charged set of batteries. Additionally, the vehicle can idle for approximately six hours. A second pair of batteries in conjuction with 40 A chargers allows us to run Prometheus continuosly for unlimited periods of time.

Component	Max Power (W)	Nominal Power (W)	Idle Power (W)
Computer	367.8	344	344
GPS Reviever+ Antenna	4.2	4.2	1
Drive motor 1	1200	600	0
Drive motor 2	1200	600	0
Steering Motor	34.7	15	0
cRIO-Chassis	20	20	20
cRIO- Modules	28	20	20
LIDAR	30	20	20
Camera	7.6	7.6	7.6
Visual Cue LED Strip	12	12	12
Monitor Touchscreen	25	25	20
Total	2929.3	1667.8	444.6

Table 1: The power consumption of Prometheus' various components. The battery power available is 2640 Watts. The expected running times for max power, nominal power, and idle power are 0.90 hours, 1.58 hours, and 5.94 hours repesctively.

3.4 Mechanical Design

A number of improvements have been implemented for the design and functionality of the mechanics of the robot. . One problem that last year's team encountered was difficulties discerning the painted lines on the grass, often due to low angle light or not being able to see lines close to the front of the robot. To minimize glare from low angle light, polarized filters were added to the lenses and the cameras were relocated higher and further forward on the robot. The viewing angle was improved and the minimum distance of the field of view was also improved as shown in Figure 6.

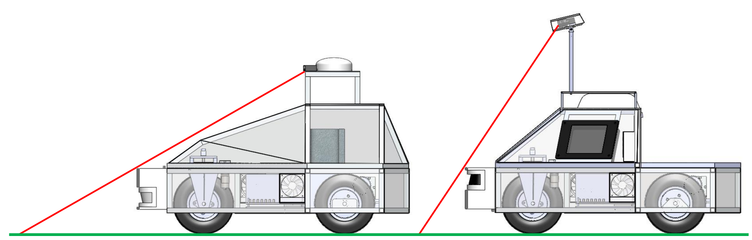
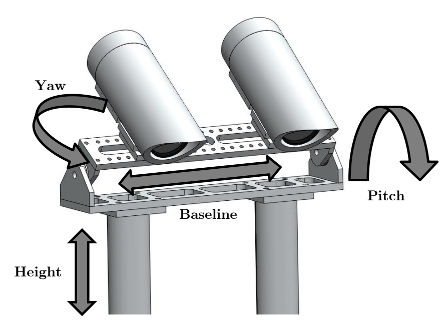


Figure 6: Comparison of the location of the camera in the old and new Prometheus designs.

In order to improve the use of the robot for different applications, the camera mounts are designed to be adjustable. With the previous design, only the pitch of the cameras could be adjusted. To allow for more configurations, the new design allows for the adjustment of the height, pitch, individual yaw and baseline for the two cameras as shown in Figure 7(b).

(a) The old camera mounts with adjustable pitch



(b) New design adjustable for yaw, pitch, height, and baseline

Figure 7: Comparison of the old and new camera mount designs.

Maintaining position stability and repeatability were two important design criteria for the new camera mounts. A planar knurling design, as shown in Figure 8, is used between the two faces for both the pitch and the yaw adjustment.

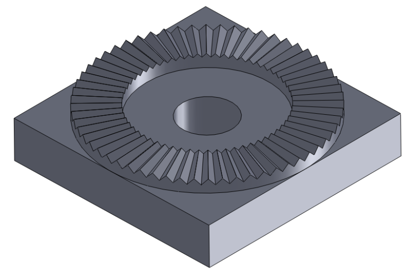
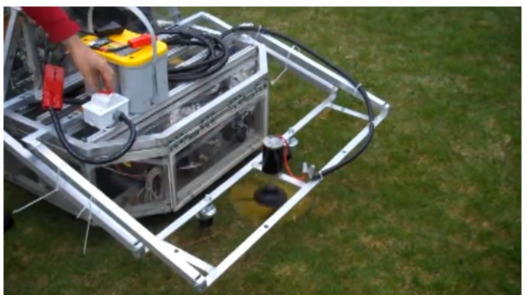


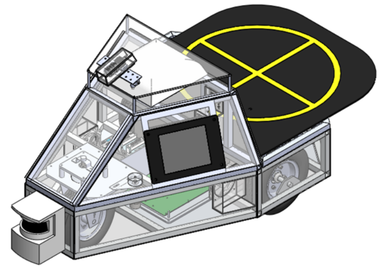
Figure 8: Planar knurling used to keep rotational joints from slipping.

The top cover was redesigned to include a modular payload area. The old design of the cover included an internal payload area for objects the size and weight of a cinderblock. This limits the type of payload considerably. Having a modular, external payload platform allows for a much wider range of uses.

The final design for the cover includes the fixed front cover with side doors to the computer compartment and a back cover that opens to access the batteries and drive train. The back cover has four extruded aluminum t-slot bars to which a wide variety of components can easily be attached. One device that was built for the robot was a lawnmower that attaches to the back cover, as shown in Figure 9(a), to be used in the ION Robotic Lawnmower competition. Other attachment possibilities for future robotics research include a quad rotor landing pad or a robotic arm as shown in Figures 9(b) and 9(c).



(a) Lawn mower attachment



(b) Unmanned Ariel Vehicle (UAV) landing pad attachment

(c) Robotic arm attachment

Figure 9: Prometheus with various attachments for its modular payload area.

3.5 Sensors

Prometheus 2010 used an array of sensors that gave the robot information about its surrounding. The robot got velocity with optical wheel encoders on the back driven wheels, position with a differential global positioning system (DGPS) receiver, heading with a compass, and information about obstacles with a light detection and ranging (LIDAR) sensor. Unfortunately, the team was unable to qualify at IGVC 2010 due to several problems with the sensors.

The 2011 team decided that several new design innovations were needed in order to succeed at the IGVC. These included a new DGPS receiver and the addition of an inertial measuremet unit (IMU). The team also fixed problems gathering reliable data from the encoders and updated the LIDAR data collection.

3.5.1 Wheel Encoders

Due to excessive sensor error, the 2010 team could not rely heavily on the encoder data, making Prometheus unable to navigate successfully. This year, the team determined that the encoder data was unreliable because the cRIO has a sampling rate of 110 KHz and the encoders have a 15 KHz signal, which was overloading the cRIO data acquisition system.

The 2011 team fixed this overloading problem by decreasing the encoder signal to a lower frequency, passing it through a logic counter chip. Once this was done, the team saw much more reliable data from the encoders.

3.5.2 Differential GPS Receiver

The 2010 team experienced very high position error with the Sokkia Axis1 DGPS at the IGVC. Because of this, the 2011 team decided to use the Trimble AG252, which has an OmniSTAR subscription and 10 centimeter accuracy.

3.5.3 Laser Measurement Sensor

The 2010 team implemented successful obstacle detection and avoidance the IGVC in 2010. The 2011 team, however, discovered that data was only being gathered in a 90 degree window in front of the robot. This was adjusted and the lIDAR now reads 180 degrees, giving the robot much more information about it's surroundings.

3.5.4 Inertial Measurement Unit

Prometheus 2010 was outfitted with a basic 2-axis compass module. The 2011 team decided that more capabilities were needed for further development. The team added a PNI 6 degree-of-freedom IMU, which gives Prometheus much more information about its location.

4 Intelligence

4.1 Software Architecture

Two main design considerations for the software architecture were extensibility and modularity. The architecture must be modular because it is likely that future teams will want to easily swap out current components of the system with different ones. The architecture must be extensible because it is likely that we have not foreseen all the possible use cases for the research platform, and the architecture might need to be adapted to those use cases.

4.1.1 Data Flow

There are two separate computers on Prometheus. The first one is the Onboard Computer (OBC) which is a standard x86_64 computer that runs Ubuntu Linux. The OBC is used for high level tasks such as path planning and waypoint navigation, line detection, and obstacle detection. The second one is a National Instruments cRIO that runs VxWorks. The cRIO contains an FPGA and low level I/O connections, which makes it suitable for low-level data collection tasks such as gathering data from the encoders and the IMU, and performing motor controls and the front wheel differential drive. Additionally, the cRIO houses our Kalman Filter which fuses the different available sources of odometry data.

Since both of these systems require data from each other, as well as the controls laptop which displays realtime data, there is a constant flow of information across the network. Figure 10 represents the end to end data flow inside the system.

4.1.2 Framework

We have chosen to use Willow Garage's Robot Operating System (ROS) as the underlying framework for our code. ROS is essentially a set of pre-packaged opensource libraries supplemented with additional low level tools such as a serialization and communications system, a build system, a data collector and visualizer. ROS lends itself to the specified architectural requirements: Because software is arranged into "nodes" that that performs a single task, modularity is idiomatically enforced. Data is shared through a publisher / subscriber method and services which means that newly added nodes can also listen to these data sources and use these services, which aids extensibility.

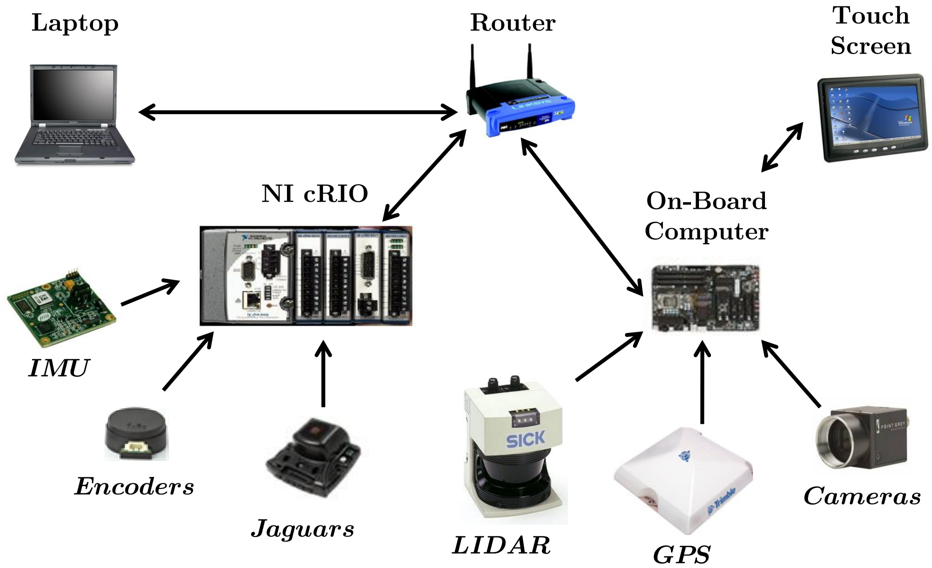


Figure 10: The major sensors and computing devices of Prometheus 2011.

In Prometheus 2010, the software was implemented as a single monolithic process with multiple threads that run individually. This has several disadvantages:

- When two components need to share data, shared memory with locking is used which introduces difficult to debug problems such as race conditions and deadlock. The new architecture shares data through message passing, which eliminates such problems.
- It is difficult to swap out components while the system is running. However with the new architecture, any component can be started, stopped and replaced dynamically.
- If a thread crashes, it is likely to cause a catastrophic failure that affects the whole process. In comparison, in the new architecture a crash is limited to a single node and there is a watchdog that automatically restarts any node that dies.

4.1.3 Simulation

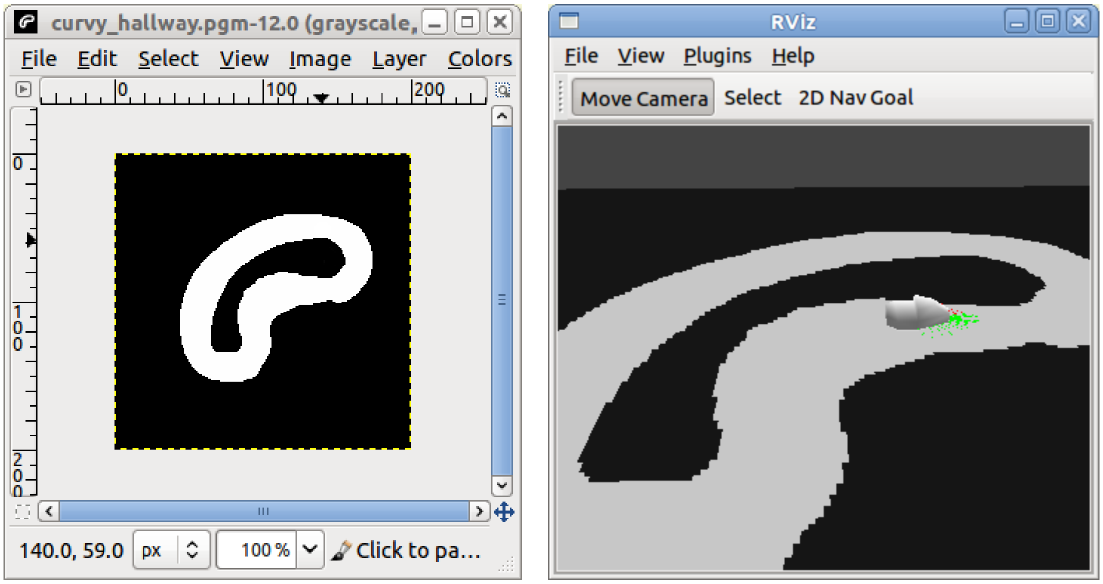
We have made use of logged data to extensively test Prometheus' various software components. In addition, we have written a simulator for the ROS framework that allows us to load maps from image files and test Prometheus' path planning ability on them. Figure 11 depicts the software used to both edit and visualize maps in simulation.

4.2 Localization

Localization is an important aspect of navigation. In order to get to different waypoints, we must know the robot's position. Current position is determined by adding the change in position to the previous position. This means that any error in sensor measurements will be present in the position calculation, and continually accumulate over time. A good solution to this problem is the implementation of an extended Kalman filter (EKF).

Prometheus 2010 had no form of sensor error reduction for localization. Determination of the robot's state was solely dependent on the calculation of position based on wheel encoders, a compass, and the previously mentioned unreliable DGPS receiver. This year's team was able to successfully implement an EKF. The filter is applied to the wheel encoders and IMU, and initialized with the position obtained from the DGPS receiver.

The overall goal of our implementation of an EKF is to determine our robot's state in terms of its position and



(a) A map being edited as an image file in GIMP

(b) Prometheus navigating on an occupancy grid loaded from an image file.

Figure 11: A simulation of the robot performing navigation on a map loaded from an image file.

heading. Our state vector is described in Equation 1, where x and y are the robot's Cartesian coordinates in meters in a local frame, and θ is the robot's heading in degrees.

$$\vec{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T \tag{1}$$

The filter is updated with measurements from the encoders and IMU. The form of our measurement matrix is shown in Equation 2.

$$z = \begin{bmatrix} left\_velocity \\ right\_velocity \\ \theta \end{bmatrix} \tag{2}$$

4.2.1 Testing and Results

The EKF is run in LabVIEW where incoming sensor data can be input directly. Testing was performed by doing laps of a rectangular building on campus. During this time, the raw and filtered sensor positions were being logged. Figure 12 shows how the raw sensor position (red) accumulated error and caused drift during the second lap of the building. In the filtered sensor position (blue) it can be seen that the two laps were much more overlaid, indicating that the error accumulation was being reduced.

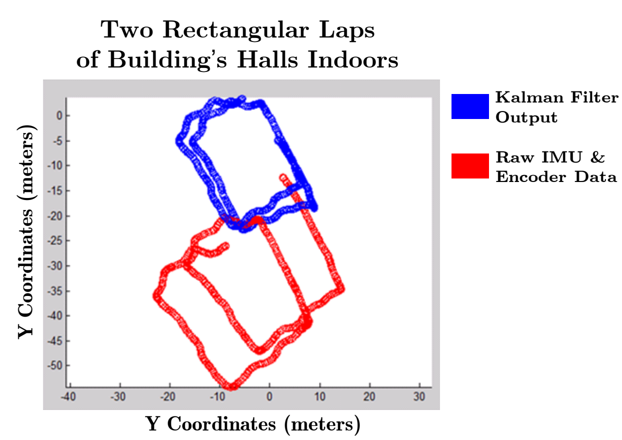


Figure 12: Raw sensor position and filtered sensor position while driving two laps around a campus building.

4.3 Line Detection

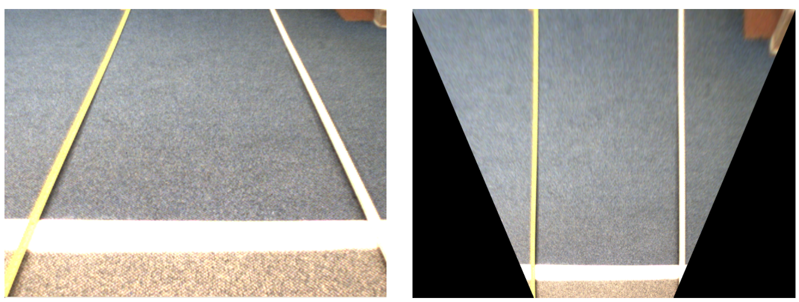
One requirement that Prometheus must fulfill is to not cross the white or yellow solid and dashed lines that indicate the boundaries the robot must drive in. Since the only easily distinguishable feature of these lines is their visual appearance, image processing must be used to detect the lines. Two cameras cover the front area of the robot to an approximate range of 4 meters with the current camera alignment. The processing for the data from these two cameras is done using the Willow Garage OpenCV library.

4.3.1 Image Capture

The image is captured over ieee1394 at 1024x768 at 15 fps per camera. This data is constantly published for further processing down the pipeline. This has the advantage of decoupling the image capture from the image processing, which means that we can also record live image data to use in simulations, or easily switch to using different image sources.

4.3.2 Inverse Perspective Mapping

Since the cameras are angled forwards but the data we wish to obtain is from the top-down perspective, the image data must be transformed to the top-down point of view. To do this, we make use of the "Flat Ground Assumption" which states that the ground is completely planar and always at a specific angle to the robot. This assumption allows the placement of lines in 3D space without depth perception. While this might be an unacceptable for applications where the terrain heavily varies, the competition grounds are mostly flat, and the small errors remain negligible.



(a) Unfiltered control image. (b) Inverse Perspective Mapped image.

Figure 13: The effects of Inverse Perspective Mapping on an image of two rods placed in front of the robot.

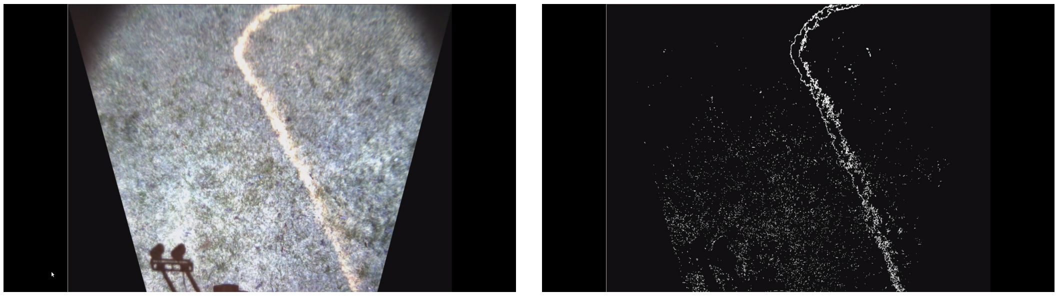
4.3.3 Color Segmentation

The color content of a random sample of white line pixels through the camera were analyzed, along with a random sample of different sorts of non-white areas like grass and soil. Then, the differences were identified. It was noted that in the HSV (Hue, Saturation, Value) color space, the hues of the line pixels were largely distinct from most of the other kinds of pixels. This left the detector with only a few false positives, mainly in exposed soil. Through the previous analysis, it was determined that the value channel had different ranges of values for the two types of terrain, so additionally limiting the range of value channel removed the false positives.

4.3.4 Pre-Processing and Post-Processing

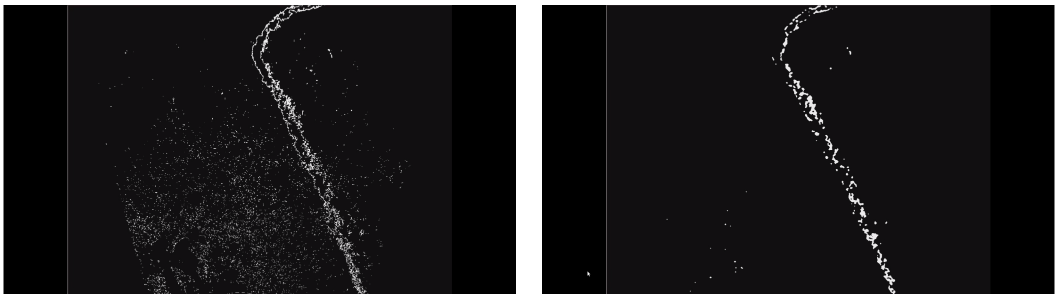
The detection target, painted lines on grass, can be challenging to detect accurately on the raw image because of the relative lack of contrast between the lines and the grass, the surface of the grass not being uniformly white and containing shadows from the grass, and the paint wearing off (which from now on will be called "noise"). To elminate the effects of noise, a series of image processing operators are applied to the image.

Smoothing brings outliers in the data set closer to the prevailing mean or median. The resulting image appears blurred, but also less variance of different colored pixels within a given area. It was found empirically that median



(a) Inverse Perspective Mapped image. (b) Color segmented image.

Figure 14: Color segmentation only leaves relevant parts of the image for line detection.



(a) Before filtering. (b) After filtering.

Figure 15: The results of post processing using morphological opening and median filtering. As can be seen, the results in 14(b) have much less noise.

filtering worked best to create the desired effect of contiguous smooth areas of similar colors.

Morphological opening and morphological closing are two morphological operations that have the effect of eliminating contiguous foreground (bright) or background (dark) color regions in the image. This allows us to remove "salt and pepper noise" which is a type of noise that consists of lots of unwanted and randomly distributed small specks.

4.3.5 Line Detection

A popular method for line detection in computer vision is a process called the Hough Transform, as described in (Hough, 1962). The basic process of the Hough Transform involves iterating through each non-empty pixel and for each pixel "voting" on what the point thinks are valid lines. Each pixel will vote for a specific number of lines going through that pixel. More lines allow for more granularity, but add to processing complexity since this process must be repeated for every pixel. Certain lines from collinear pixels will get a much higher amount of votes, which will be the "lines"" detected.

The Probabilistic Hough Transform (PHT) is a refinement of the standard Hough Transform, in which the pixels are randomly sampled instead of processing every single pixel. Because of the way that the random normal distribution works, this usually generates a sufficient amount of votes from the edges to detect lines, while saving time by not counting a good amount of redundant votes.

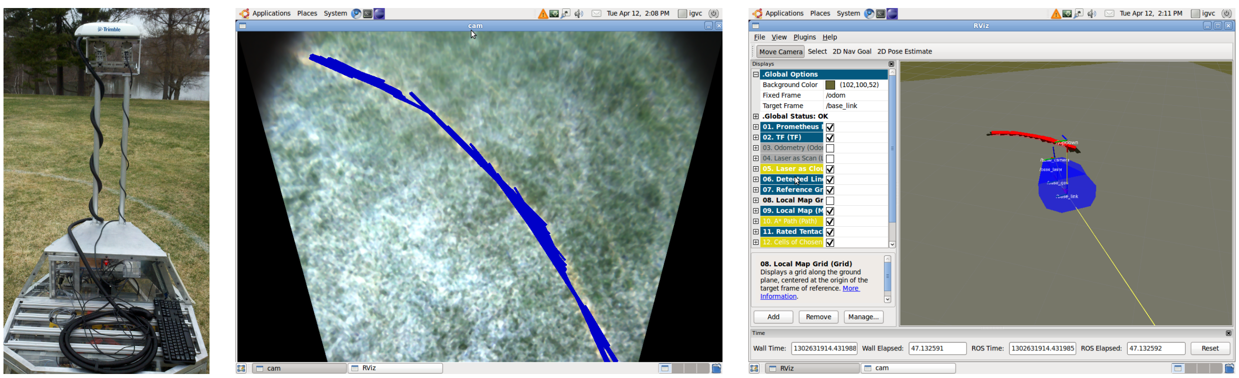
4.3.6 Results

As can be seen from Figure 17, we have effectively developed a line detection system that functions robustly under a wide variety of lighting conditions. While formal testing has not been conducted, during the routine testing of the robot the line detection system has functioned well in many different times of day and different amounts of cloud cover,

(a) Source image. (b) Hough transform results.

Figure 16: The results of the Hough Transform overlaid on the source image.

as well as under indoor lighting. The performance limitation of the system running in a single thread on an Intel Core i7 CPU is approximately 8 fps.



(a) Outside view. (b) Camera view with detected lines. (c) View of detected lines in rviz.

Figure 17: Demonstrating the three phases of detection.

4.4 World Representation

Prometheus needs to relate its position to the positions of nearby obstacles in order to perform obstacle avoidance. We decided to do this by creating a probability map that estimates the positions of obstacles in the world. As the robot moves, it updates the probability that a given position on the map is occupied by an obstacle.

The technique for mapping currently used for Prometheus is similar to the one used by last year's team, in that a probability map is computed which is fixed relative to the robot's local frame. However, some major improvements have been made. First, laser measurements and detected lines are now stored in the program as a list of continuous points rather than being immediately converted to and stored as discrete grid cells. The points are then later used to build a grid map that is broadcast using a ROS message that can be visualized in rviz. This preserves more accuracy from each measurement, because the map is fixed to the robot so there must for determining when a measurement has moved from one grid cell to another is it moves. Calculating the cell that a measurement occupies after the robot has moved is easiest when using continuous points because we can transform the measurement from its old location to its new location and then convert the point to a grid cell. Another improvement we made was to use Bresenham's line algorithm to fill in the

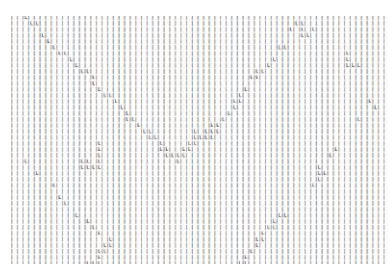
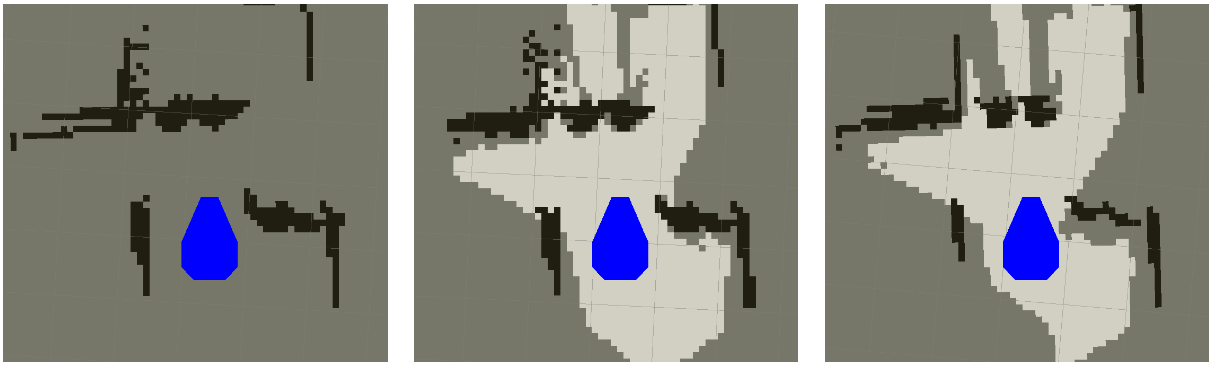


Figure 18: Map being visualized using an ASCII image by last year's team. A major improvement for Prometheus this year has been use of the robotics visualization software, rviz.

free spaces between the robot and obstacles. This allows us to distinguish between unoccupied regions and regions that have not yet been observed.



(a) Local map without line of sight drawn to the obstacles. (b) Local map with line of sight drawn to the obstacles. (c) Local map using a 2.5 second timeout for measurements that are in front of the LIDAR.

Figure 19: The progression of the techniques used for creating the local map from LIDAR data. All techniques used a maximum age of 10 seconds for the LIDAR measurements.

We also now use a more flexible design for the mapping program. Because our robot takes measurements from different sensors, laser and cameras, we make use of polymorphism for the different measurements. The responsibility of determining how each measurement should be drawn on the probability map is then delegated to the respective sub-class. For example, camera measurements know to draw a line of occupied cells between each endpoint, and laser measurements know to draw a single occupied cell for the obstacle as well as a line of unoccupied cells between the robot's position when the measurement was taken and the obstacle. The process of determining when a point expires is also delegated to sub-classes. Given the maximum age and range, the measurement determines if it is too old or too far from the robot. Because of our use of rviz, we were able to visualize the improvements made to the map during each step of our implementation (see Figure 19). This is a major improvement over the ASCII map visualization (pictured in Figure 18) used by last year's team.

Lastly, the parameters for building the map are now much easier to change. The width, height, and resolution of the map are all configurable by passing parameters to the mapping program. Additionally, parameters for configuring how measurements are stored and drawn on the map are also available. These include the maximum age of a measurement and the maximum distance a measurement can be from the robot. All parameters used for creating the map are outlined in Table 2.

Parameter	**Description**
Width	The total number of cells on the map to the robot's left and right.
Height	The total number of cells on the map to the robot's front and back.
Resolution	The length of the side of each grid cell. Grid cells are always square.
Maximum age	The maximum length of time for each measurement to persist in memory.
Maximum range	The maximum distance a measurement can be from the robot to persist in memory. By default this is the distance to the corner of the map.
Maximum size	The maximum number of measurements to persist in memory at any time. If this number is exceeded, the oldest measurements are deleted to make room for new measurements.
Margin size	The size of the margin between the last cell in the line of sight to an obstacle and the obstacle itself.

Table 2: Listing of parameters used for building the probability map.

As expected, accurate odometry data is necessary to build a coherent map using the technique outlined above. In testing, it was found that poor odometry data would cause the map to smear as the same static obstacle would continuously be measured at a different location as the robot moved. To mitigate the smearing effect, the laser measurements expire at different times depending on their location. Laser measurements that are in front of the LIDAR sensor expire very quickly because they are likely to be observed during the next scan. However, once a laser measurement moves behind the LIDAR sensor, it takes much longer to expire. This is done because laser measurements behind the robot will likely not be observed again, but are still necessary to help the robot avoid brushing the sides of obstacles.

The local map program also performed considerably well on maps of varying resolutions. The main factor that causes the program to slow down is the number of measurements stored in memory. It was determined in testing that the number of measurements we can comfortably store in memory at a time is about 10,000 measurements. For this reason, the number of measurements is currently limited to 10,000.

4.5 Path Planning

Last year, the team was able to effectively avoid obstacles by generating and evaluating a set of arced paths originating from the robot's turning center. This technique, sometimes referred to as driving with tentacles, is generally used for large, outdoor vehicles and has been effectively used by a number of competitors in the DARPA grand challenge (von Hundelshausen et al., 2008). One advantage of this technique is that the paths are extremely easy to follow because each arced path can be summarized by two motion commands: a linear velocity and an angular velocity. While last year's team was successful in implementing tentacles and avoiding obstacles, they were unable to plan long term paths to the goal. We were able to overcome this problem by combining the tentacles algorithm with the A* path finding algorithm discussed in Section 4.5.2. In addition, to keep up with the dynamic nature of the robot environment, each of the algorithms used in the path planning process is repeated as fast as possible.

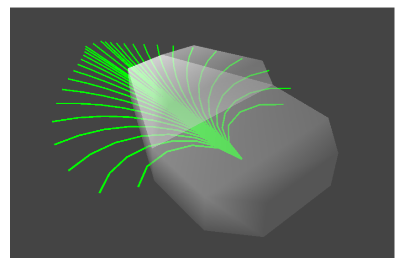


Figure 20: The arced paths used for driving with tentacles on Prometheus being visualized in rviz.

4.5.1 Driving with Tentacles

We decided to continue using tentacles because of its known effectiveness in avoiding obstacles. The code from last year was rewritten for compatibility with ROS and our local map implementation. After generating a set of arced paths (see Figure 20), each path must be evaluated for drivability. There are several methods of doing this, but in general most methods should take into account 1) the obstacles that the robot will encounter when driving on the given path, and 2) how much following the path will help the robot reach its destination.

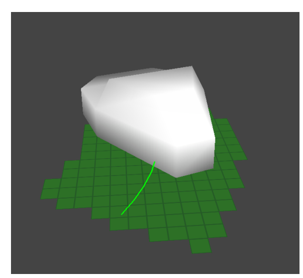


Figure 21: The inflated grid cells of an arced path that account for the width of the robot and are used to evaluate the drivability of a path.

Because the local map is represented as an occupancy grid, the evaluation method needs to convert tentacles to grid cells. In doing this, each tentacle is inflated to account for the width of the robot. This extra padding around each tentacle is known as the classification area (von Hundelshausen et al., 2008).

To inflate each arced path, a list of points for the inner and outer radius is generated using the algorithm described in Figure 22. After the points for the inner and outer radii are generated, each of the points is converted to a grid cell, and then Bresenham's line algorithm is used to connect each of the cells to create an outline that traces the inner radius and the outer radius and connects both of their end points together. Following this, a point from the center of the arced path is selected, and a flood-fill algorithm is run to fill in each of the grid cells of the boundary. An image of the grid cells for an arced path is shown in Figure 21.

Algorithm *GenerateClassificationArea*
Input: A list, L_{path}, of vectors on the path, and the width of the robot, W
Output: A list, $L_{classification-pairs}$, of pairs of vectors that are the borders for the classification area of path

```
L_{classification-pairs} ← ∅
for v_k in L_path
    (* For each v_k, calculate v, the vector from the last point on the path to the v_k *)
    do v ← p_k − p_{k−1}
        v_displacement ← v rotated π/2 radians about the z-axis
        v_displacement ← v_displacement normalized
        v_displacement ← v_displacement * W/2
        insert (v_k + v_displacement, v_k − v_displacement) into L_{classification-pairs}
return L_{classification-pairs}
```

Figure 22: Algorithm for determining the points on the edges of the classification area of a path. One advantage of this algorithm is that the technique can be applied to other types of arced path as well. For example, if a tentacle algorithm were used where the arced paths were replaced by Euler spirals, then the same algorithm would still be able to determine the classification areas of the paths.

Once the grid cells have been generated, they can be overlaid on the map to evaluate each arced path. Both choosing a tentacle that does not collide with an obstacle and choose one that will eventually lead to the goal are equally important. However, mixing these two operations into one evaluation function could be dangerous, because if tuned incorrectly, a path that hits obstacles to reach the goal may be rated higher than a path that takes a safer but longer route around the obstacles. For this reason, it was decided to separate the tentacle evaluation functions, which determined the scores of tentacles based on the local map, from the tentacle choosing function, which choose one of the rated arced path to drive on.

4.5.2 Waypoint Navigation with A*

To plan a long term path to the goal, the robot needs to know the general direction to drive in. This is more complicated than simply finding the heading to the waypoint since obstacles may be obstructing the straight-line path between the robot and the goal. To guide the robot in a direction that will avoid obstacles and eventually lead to the waypoint, we first use the A* algorithm to calculate the best path from the robot to the waypoint on the occupancy grid. Once we know the best path, we choose to drive on the tentacle that most closely follows this path.

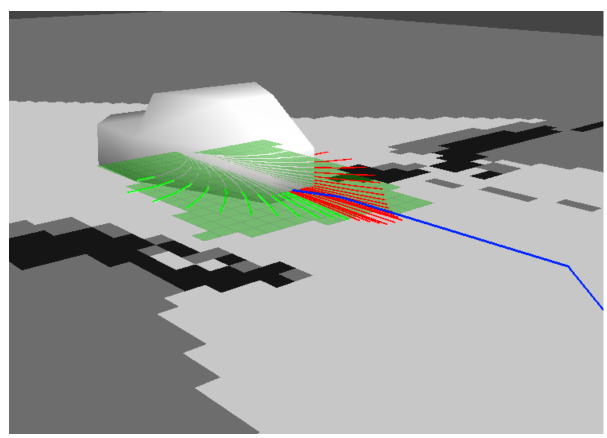


Figure 23: Tentacle planning and A* path planning for waypoint navigation. Drivable paths are shown in green, undrivable ones in red, and the best path in blue. The green grid cells show the cells that the robot will occupy as it follows a path.

There were a couple of issues we needed to overcome with A* in order to make it effective for Prometheus. First, A* sometimes calculates a best path through a small opening between obstacles that the robot cannot fit through. To overcome this, we assigned a distance value to each grid cell proportional to the number of nearby obstacles. The second issue we ran into was A*'s slow running time. To mitigate this problem, a timeout was added to the A* calculation. If the algorithm takes longer than a specified time, then the resolution of the grid is reduced, and A* is recalculated. This process is repeated until A* finds the best path to the goal.

Once a path to the goal is found, the A* tentacle chooser function looks at each arced path that is rated above a certain threshold. It uses the algorithm described in Figure 24 to find the arced path that is closest to the A* path. An image of Prometheus driving with the A* tentacle chooser is seen in Figure 23.

Algorithm *FindTentacleClosestToAStarPath*
Input: A list, $L_{tentacles}$, of tentacles to be evaluated, and the best path to the waypoint, *path*, as calculated by A*.
Output: A tentacle in $L_{tentacles}$ that is judged to follow the given path most closely.

for *tentacle* in $L_{tentacles}$
    do $L_{distances} \leftarrow \emptyset$
        for each point v in *tentacle*'s path
            do $d \leftarrow$shortest distance from v to a point in $L_{tentacles}$
                append d to $L_{distance}$
        score $\leftarrow$geometric mean of values in $L_{distances}$
        Associate *score* with *tentacle*
return tentacle in $L_{tentacles}$ with lowest *score*

Figure 24: Algorithm for determining the tentacle closest to the path planned by A*.

4.5.3 Late Obstacle Avoidance

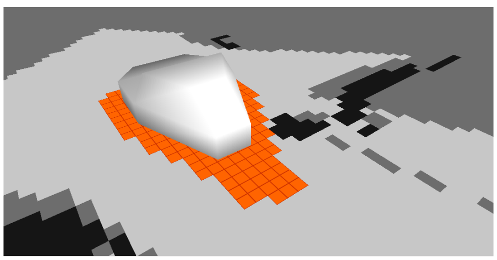


Figure 25: The inflated footprint being used to prevent Prometheus from continuing to follow a path that will cause it to strike an obstacle.

Once a tentacle is chosen, the robot sends the motor commands to the control system. Unfortunately, it is sometimes possible for the robot to mistakenly choose path that will eventually collied with an obstacle. To prevent this, a method of late obstacle avoidance is used in tandem with the path planning algorithms. An inflated footprint of the robot's shape, seen in Figure 25, is generated and then overlaid on the occupancy grid to search for obstacles. If an obstacle enters this footprint, the robot stops to avoid a collision and then replans a path to the goal.

5 Conclusion

Prometheus 2011 brings major improvements to Prometheus 2010 in many areas to better fit the requirements of the robot. Mechanically, the chassis cover and camera mounts were completely redesigned, and the usability was revamped accordingly. Concerning intelligence, a new software architecture was built from the ground up and new line detection and and path planning algorithms were implemented, as well as a new sensor fusion system.

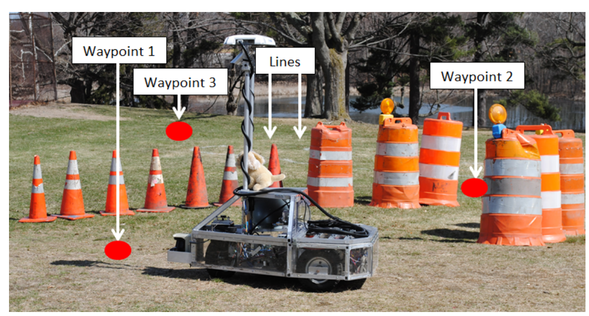


Figure 26: Prometheus in action during the April 8, 2011 demonstration in Institute Park.

6 Budget

Item	**Cost (USD)**
Touchscreen 3M™ MicroTouch™ Display C1700SS (15”) Serial	467.86
OCZ Vertex 2 60GB SATA II MLC Internal Solid State Drive (SSD)	129.99
External Interface	79.33
Futaba 6EX 2.4GHz 6-channel Remote Control	214.00
Chassis Improvements	925.43
cRIO Serial Module	579.00
Camera Lens	233.00
12V Schumacher SE-4022 Battery Charger and Tester	246.00
Subtotal	2874.61

Item	**Sponsorship**	**Cost (USD)**
OmniStar GPS Subscription	OmniStar	850.00
TCM-XB Evaluation Kit (Inertia Momentum Unit)	PNI	1,096.00
ThinkPad T410 Laptop	Dyn	1,200.00
	Subtotal	3146.00
	Total Cost	6020.61

References

Hough, P. V. (1962). Method and means for recognizing complex patterns. US Patent 3,069,654.

von Hundelshausen, F., Himmelsbach, M., Hecker, F., Mueller, A., & Wuensche, H.-J. (2008). Driving with tentacles: Integral structures for sensing and motion. *J. Field Robot.*, *25*, 640–673. Retrieved from http://portal.acm.org/citation.cfm?id=1405647.1405650.