



PROMETHEUS: WPI'S 2011 ENTRY TO THE ION ROBOTIC LAWN MOWER COMPETITION

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.

1 Introduction

2011 marks the first entry of Worcester Polytechnic Institute into the ION Robotic Lawn Mower Competition. While Prometheus was conceived of in 2010 for the Intelligent Ground Vehicle Competition (IGVC) and many advancements were made in terms of constructing the robot, this former version required many improvements to be able to perform as a competitive entry in the competition.

Efforts this year were in large part focused on giving Prometheus the necessary intelligence capabilities to be able to perform complicated navigation tasks. This requires line detection, localization, path planning, waypoint navigation, and field coverage techniques. Other efforts were also 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 new 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.

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 robot, we split up the team as shown in the table below.

Team member	Task							
	Chassis	Control System	Line Detection	Localization	Obstacle Detection & Avoidance	Sensors	Waypoint Navigation	
Mali Akmanalp			✓					
Ryan Doherty				✓		✓		
Jeffrey Gorges	✓							
Peter Kalaukas					✓		✓	
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.

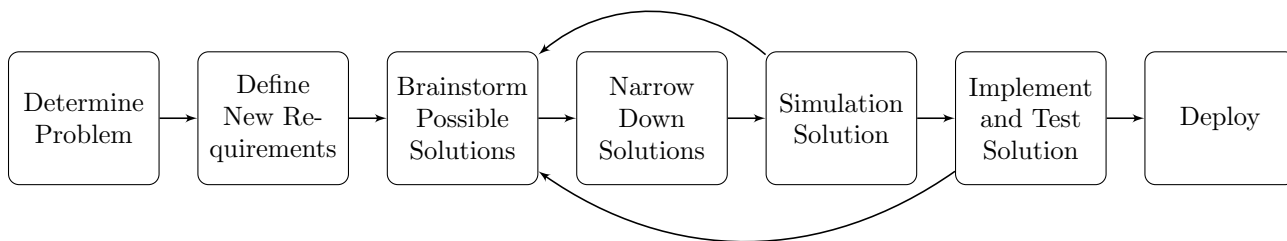


Figure 1: Design process for Prometheus 2011.

2 Prometheus Overview

The Prometheus mower consists of two main components, the vehicle and the detachable mower. 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.

Physical	Dimensions	0.89 m × 1.42 m × 1.65 m
	Weight	112 kg
Mechanical	Maximum Speed	8.9 km/h
	Minimum turning radius	0 m
	Maximum climbing angle	35 deg.
Power	Nominal run time	1.6 hours

Table 1: Overview of the physical attributes of the Prometheus vehicle

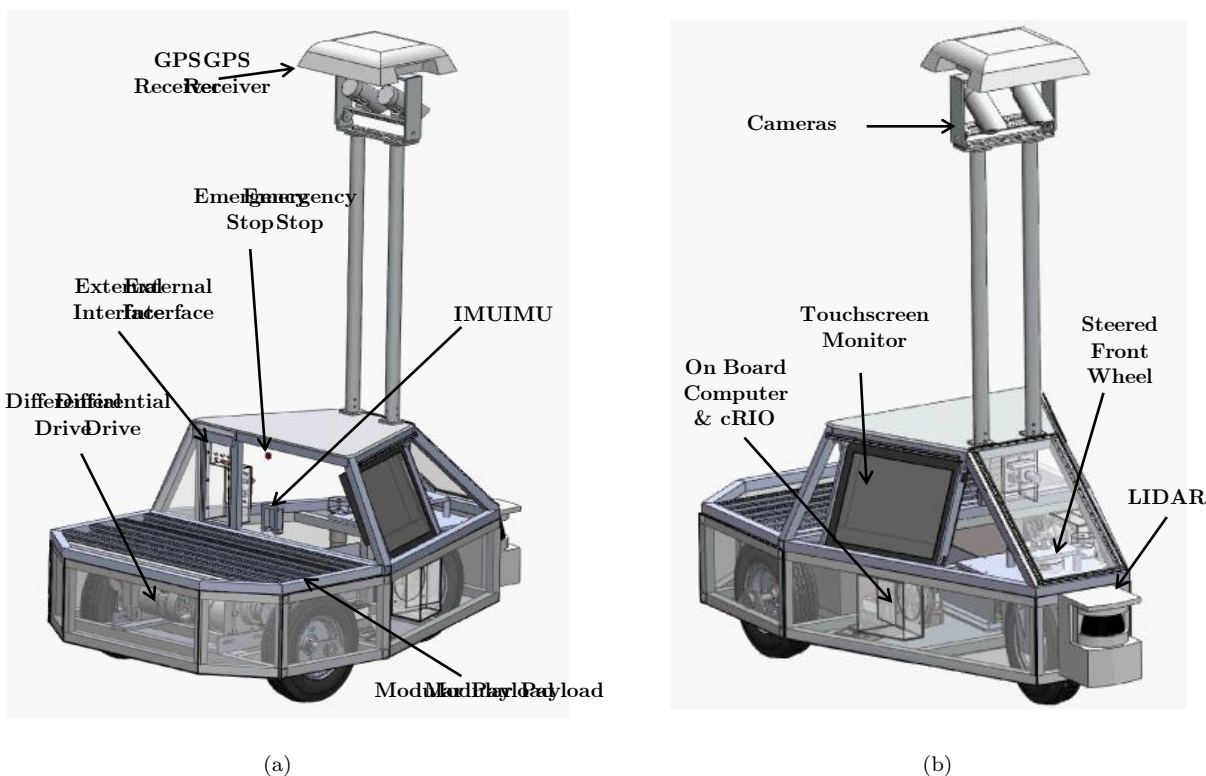


Figure 2: Overview of the components of Prometheus 2011.

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

Physical	Cutting width	1.14 m
	Addition length	0.45 m
	Weight	48 kg
Mechanical	Maximum cutter speed	6000 rpm
Power	Nominal run time	1.1 hours

Figure 3: Overview of the physical attributes of the Prometheus lawn mower attachment

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.

For the ION competition Prometheus has a custom aluminum lawn mowing solution attached to its modular payload area. The lawnmower attachment consist of an array of five (5) weed whacker blades connected directly to 12V 6000 RPM motors. This configuration was chosen because of the competition robot size limitations, motor availability, and 12V power source. The four bar linkage setup allows the cutters to always be parallel to the robot. The wheels on the cutter attachment can have their height regulated to control the height of the cut.

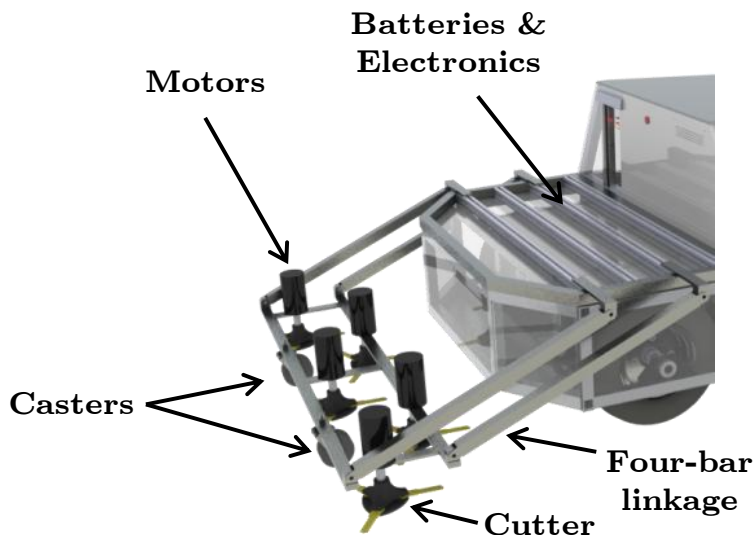


Figure 4: Components of lawn mower attachment.

3 Robot Structure

3.1 Usability Features

Prometheus' main purpose is to be a functional autonomous ground vehicle. However, for a vast majority of the time it operates under testing conditions. Most of the research and prototyping on Prometheus is done on one specific subsystem at a time, such as vision detection, mapping, or sensor fusion. Our team believes an emphasis on the human-robot interaction for Prometheus ultimately makes the development, debugging, and testing of each subsystem happen significantly faster and smoother. In order to accommodate such a requirement we came up with a multifaceted approach. 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, an external interface is needed because of 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. A central weatherproof external interface is a straightforward solution to this problem.

Second, our development team felt the need for a way to quickly visualize Prometheus' state during development. After extended brainstorming, the final design is a highly visible lighting system that changes color reflecting the state of

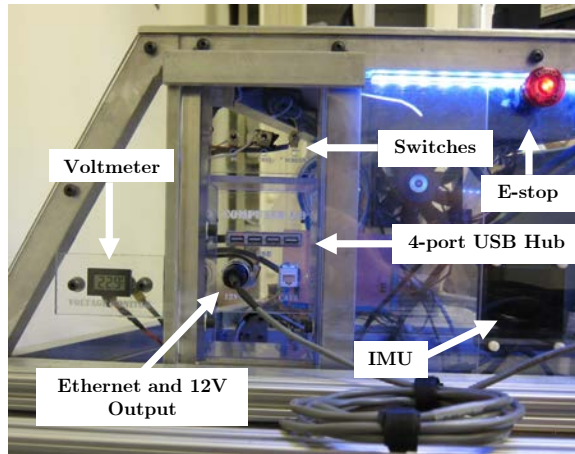


Figure 5: The external interface components, the IMU and the emergency stop button.

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 independent Pulse Width Modulation (PWM) signal for each channel allows for a practically infinite amount of color combinations.



Figure 6: Different visual cue colors representing different states.

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 autonomous. 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 6.

Third, Prometheus is made to drive autonomously, but having a manual remote control is essential during transportation, debugging, and developing phases. The solution is a RC 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 without error 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 6(a).

The usability features in Prometheus are very effective during field test. The features allow quick testing, swapping, and comparing of different algorithms. It also permits us to identify software bugs and hardware malfunctions much faster than without them, ultimately decreasing development time.

3.2 Safety

In order to comply with the ION competition 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 reliability.

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.

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
Subtotal	2929.3	1667.8	444.6
Lawnmower Attachment	3900	1200	0
Total	6829.3	2867.8	444.6

Table 2: 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 respectively.

The lawnmower attachment is powered by two external (2) 12 V 55Ah batteries connected in parallel to offer up to 110Ah. The electrical circuitry goes from the batteries, through resettable fuses, followed by a relay connected to a switch and Prometheus e-stop, to the motors. This circuitry is contained on an outdoor weatherproof box connected to the lawn mower attachment. There is no control over the motor speeds, they run freely unless the switch or the emergency stop is pressed.

The power consumption of the lawnmower attachment assumes a 65A maximum draw and a 20A nominal draw for each of the five(5) motors. Because the lawnmower attachments is connected to an external set of batteries, they discharge independently from Prometheus main system. The total nominal consumption for the lawnmower circuitry is 100A at 12V, which translate to a run time of 1.1 hours. Under nominal conditions the total power expenditure for all the vehicle’s systems combined is 2867.8W. Assuming a 2 m/s average velocity along with a 1.066m cutting edge, Prometheus consumes a total of 1528.5 W per square meter cut.

The vehicle's total running time is limited by the lawnmowing attachment at 1.1 hours. During this time, at an average speed of 2 m/s, Prometheus is ideally able to lawn an area of 8442.7 m^2 or 2.086 acres.



Figure 7: Weatherproof lawn mower electrical connection box

3.4 Mechanical Design

An important consideration in the overall design of the cover was camera placement and orientation. A balance had to be made between fitting within size limitations, maximizing the field of view, minimizing the closest point of the field of view and minimizing the adverse effects of low angle light glare. To minimize glare from low angle light, polarized filters were added to the lenses and the cameras were relocated as seen in Figure 8.

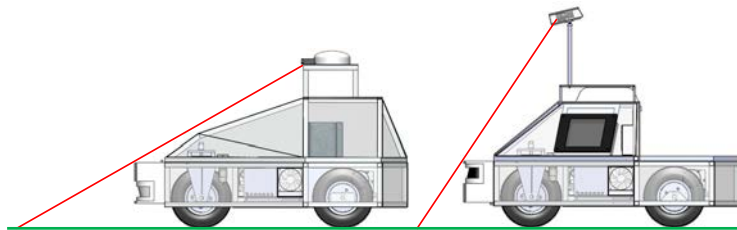


Figure 8: Comparison of a low angled view to a high angled view on Prometheus.

To allow for easy configuration, adjustment and stable positioning an innovative camera mount was designed. This design allows 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 9.

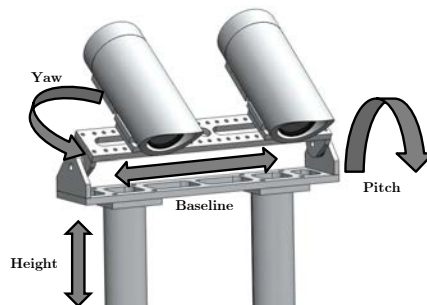


Figure 9: Camera mount design with adjustable for yaw, pitch, height, and baseline.

Maintaining position stability and repeatability were two important design criteria for the new camera mounts. A planar knurling design, as shown in Figure 10, is used between the two faces for both the pitch and the yaw adjustment.

The top cover was designed to include a modular payload area. Having a modular payload area allows for a wide 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. The cover is designed to allow for various components to be attached, namely the mower for the ION Robotic Lawn Mower competition. Other attachment possibilities for future robotics research include a quad rotor landing pad or a robotic arm as shown in Figures 11(b) and 11(c).

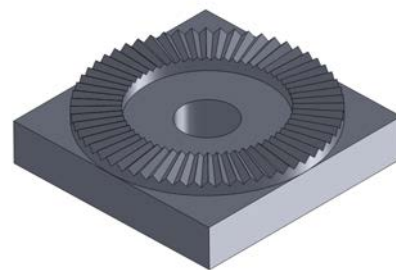
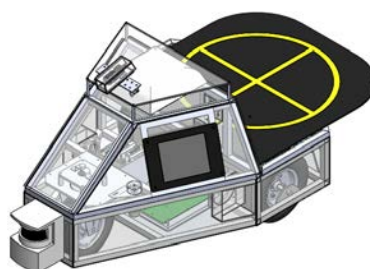


Figure 10: Planar knurling used to keep rotational joints from slipping.



(a) Lawn mower attachment



(b) Unmanned Ariel Vehicle (UAV) land-
ing pad attachment



(c) Robotic arm attach-
ment

Figure 11: Prometheus with various attachments for its modular payload area.

The mower attachment was designed to fit on the modular payload platform of the robot. A number of designs were considered including cutters on the side, bottom or back of the robot; gas or electric power; rotary or reel and a number of other design problems that had to be addressed. The main priorities in the design were simplicity, manufacturability, fitting within the size limitations, maintaining load balance and having a wide cutting path. Based on these general priorities, a list of specific design criteria was established. Seven different mower designs were compared to these criteria and the one that best fit was chosen.

The final design includes five 12 VDC motors with 10 inch diameter flail cutting heads attached to each motor shaft, offset with a one inch cutting path overlap on a motor mount platform. The platform rests on two caster wheels and is attached to the payload platform with a parallel four bar linkage on either side. The height of the castor wheels can be adjusted to adjust the height of the cut and the four bar linkage allows the platform to move independently of the main robot structure while maintaining an orientation parallel to the back platform of the robot.

3.5 Sensors

Prometheus has an extensive sensor suite, which includes wheels encoders, a differential DGPS receiver, a laser range finding (LIDAR) sensor, and an inertial measurement (IMU).

Prometheus uses two US Digital E8P Optical Quadrature Encoders with 512 counts per revolution. These sensors provide Prometheus with the current wheel velocity, which is then used to determine the robot's position. This helps the robot locate itself and navigate around the lawn which it is mowing.

Prometheus uses a Trimble AG252 DGPS receiver with an OmniSTAR subscription that ideally gives position information as accurately as 10 centimeters. The Trimble receiver was tested extensively, both on and off Prometheus,

and was found to be accurate consistently to about half a meter, often times even less. This information is also used in determining the robot's position and successfully covering the entire lawn.

Prometheus uses a SICK LMS291-S05 light detection and ranging (LIDAR) sensor to detect and avoid obstacles. This sensor is used to detect obstacles in the robot's path. With this device, we will ideally be able to avoid obstacles such as a flower bed, fence, or a pet that has wandered onto the lawn. Prometheus uses FieldForce TCM XB IMU, which was generously donated to the team by PNI Sensor Corporation. It is a high precision magnetic compass which gives Prometheus information about its heading and accelerations in different directions. The team has primarily used the IMU to collect heading information, but due to the extensive capabilities of the unit, its use is much more expandable.

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 12 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.

The software could be implemented as a single monolithic process with multiple threads that run individually, or multiple processes that communicate with eachother, each performing a single well-specified task. It was determined that the former 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 process based 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 process based architecture, any component can be started, stopped and replaced dynamically.

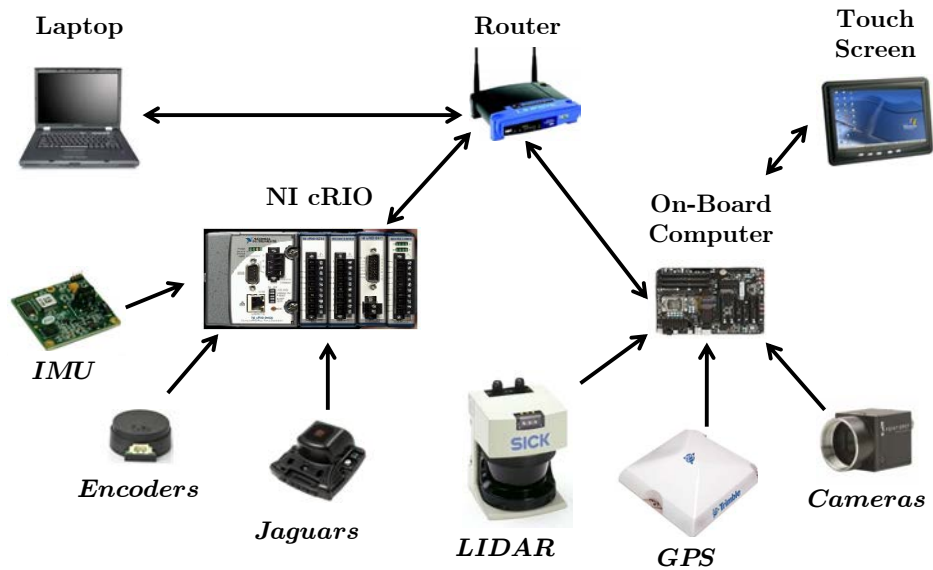
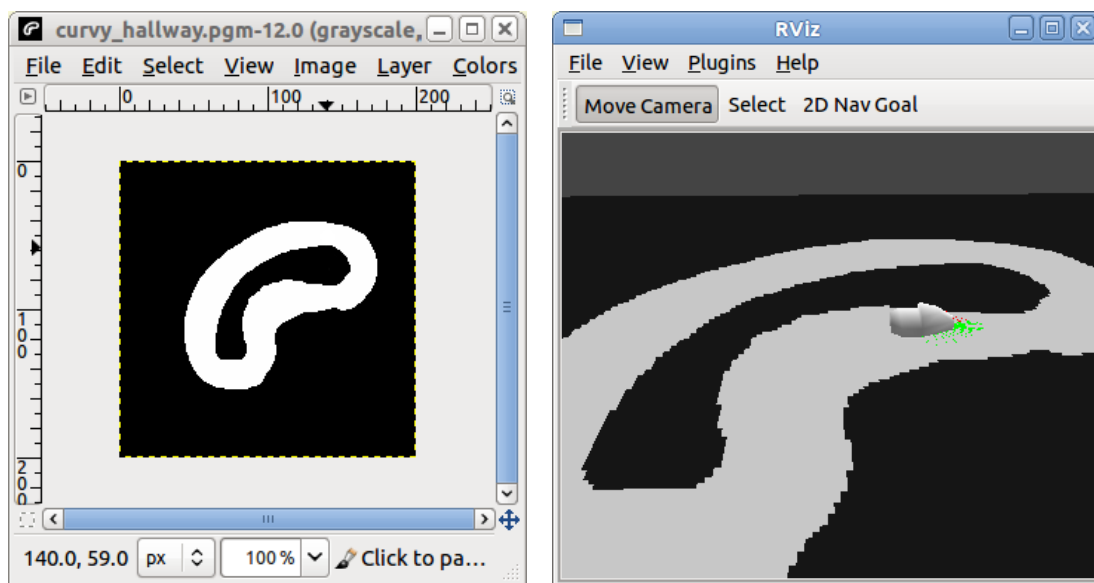


Figure 12: The major sensors and computing devices of Prometheus.

- If a thread crashes, it is likely to cause a catastrophic failure that affects the whole process. In comparison, in the process based 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 13 depicts the software used to both edit and visualize maps in simulation.



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

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

Figure 13: A simulation of the robot performing navigation on a map loaded from an image file.

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). Our 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 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.

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

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

$$z = \begin{bmatrix} left_velocity \\ right_velocity \\ \theta \end{bmatrix} \quad (2)$$

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 14 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.

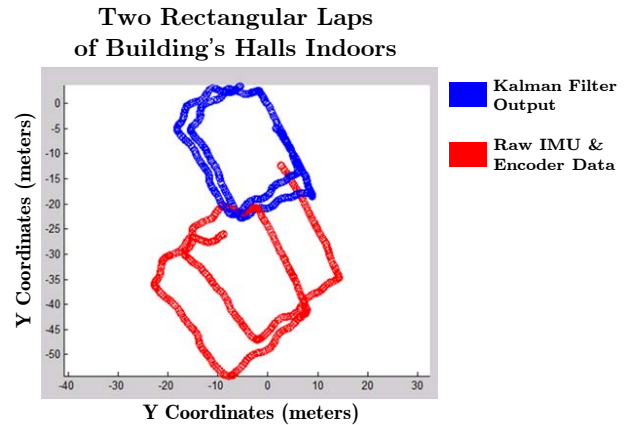


Figure 14: Raw sensor position and filtered sensor position while driving two laps around a campus building.

4.3 Line Detection

One requirement that Prometheus must fulfill is to not cross the white or yellow 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 unacceptable for applications where the terrain heavily varies, the competition grounds are mostly flat, and the small errors remain negligible.

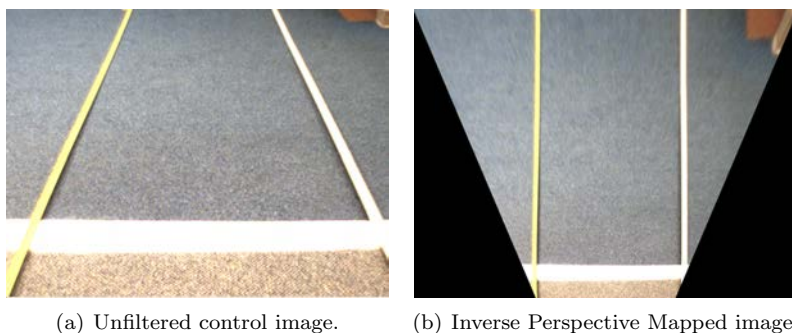


Figure 15: The effects of Inverse Perspective Mapping on an image of two rods placed in front of the robot.

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.

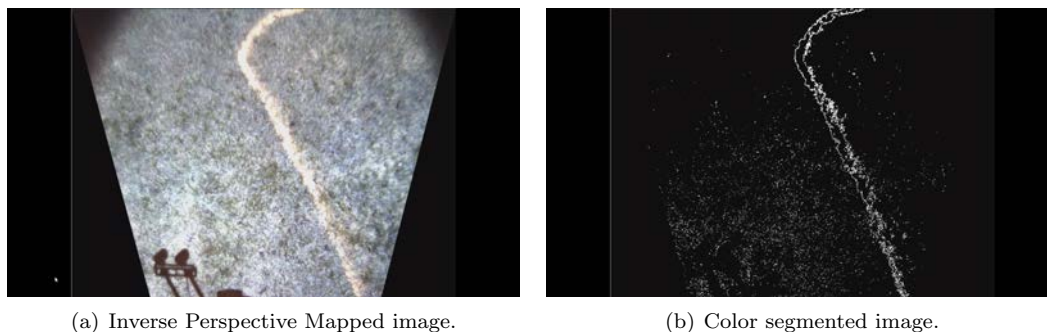


Figure 16: Color segmentation only leaves relevant parts of the image for line detection.

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 eliminate 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



Figure 17: The results of post processing using morphological opening and median filtering. As can be seen, the results in 16(b) have much less noise.

blurred, but also less variance of different colored pixels within a given area. It was found empirically that median 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 (?). 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.

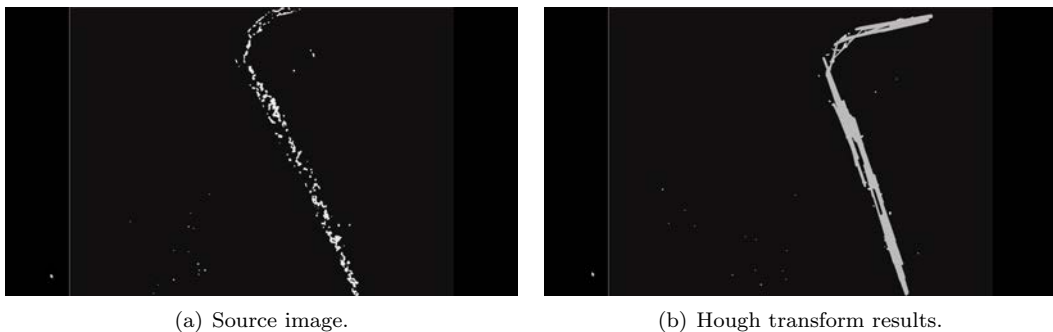


Figure 18: The results of the Hough Transform overlaid on the source image.

4.3.6 Results

As can be seen from Figure 19, 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,

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.

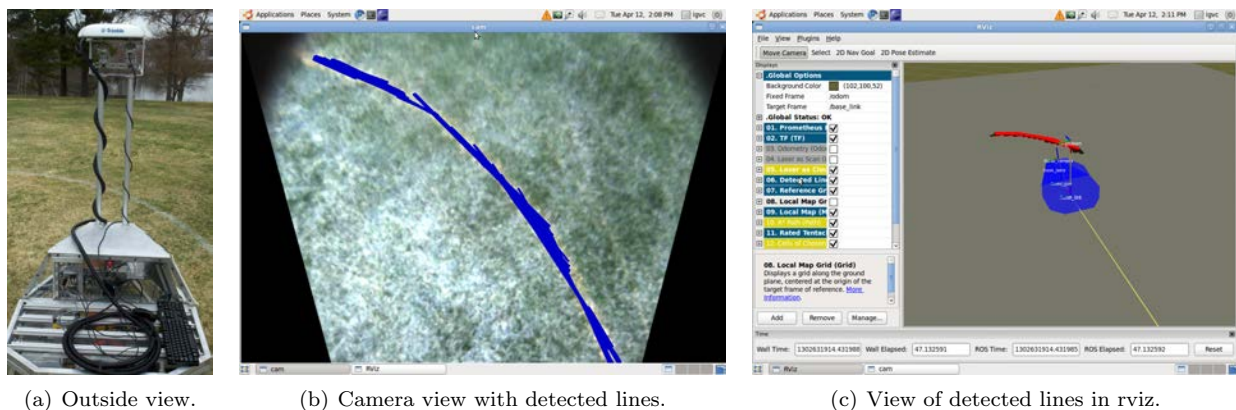


Figure 19: Demonstrating the three phases of detection.

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 used for mapping on Prometheus employs a probability map fixed to the robot's local frame. Laser measurements and detected lines are 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. Using a list of measurements preserves accuracy from each measurement because the map is fixed to the robot. The reason for this is that there must be a technique of determining when a measurement has moved from one grid cell to another as 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 free spaces between the robot and obstacles. This allows us to distinguish between unoccupied regions and regions that have not yet been observed.

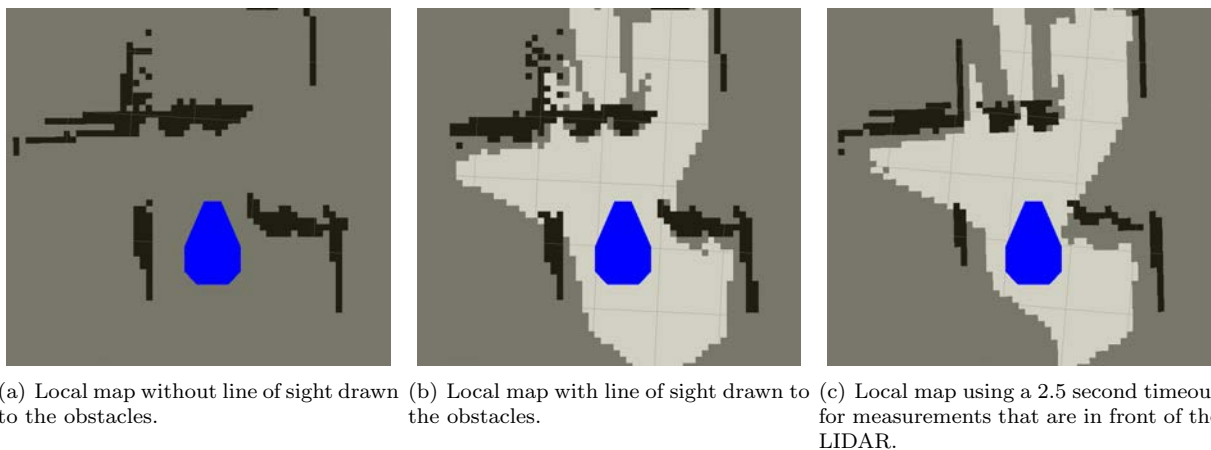


Figure 20: The progression of the techniques used for creating the local map from LIDAR data. All techniques used a maximum age of 10 seconds for the LIDAR measurements.

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 20). This is a major improvement over the ASCII map visualization (pictured in Figure ??) 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 3.

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 3: 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

We are 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 (?). 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 tentacles is effective in avoiding obstacles, it is 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.

4.5.1 Driving with Tentacles

We decided to use tentacles because of its known effectiveness in avoiding obstacles. After generating a set of arced paths (see Figure 21), 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.

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 (?).

To inflate each arced path, a list of points for the inner and outer radius is generated using the algorithm described in Figure 23. 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 22.

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.

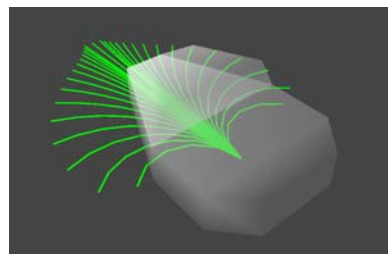


Figure 21: The arced paths used for driving with tentacles on Prometheus being visualized in rviz.

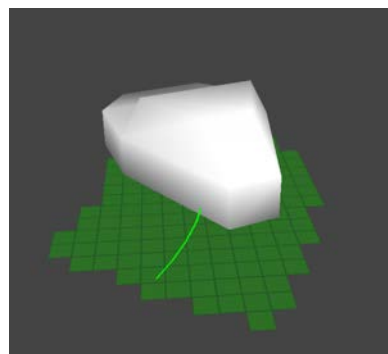


Figure 22: The inflated grid cells of an arced path that account for the width of the robot and are used to evaluate the drivability of a path.

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

1. $L_{classification-pairs} \leftarrow \emptyset$
2. **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 *)
3. $v \leftarrow p_k - p_{k-1}$
4. $v_{displacement} \leftarrow v$ rotated $\frac{\pi}{2}$ radians about the z-axis
5. $v_{displacement} \leftarrow v_{displacement}$ normalized
6. $v_{displacement} \leftarrow v_{displacement} * \frac{W}{2}$
7. insert $(v_k + v_{displacement}, v_k - v_{displacement})$ into $L_{classification-pairs}$
8. **return** $L_{classification-pairs}$

Figure 23: 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.

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.

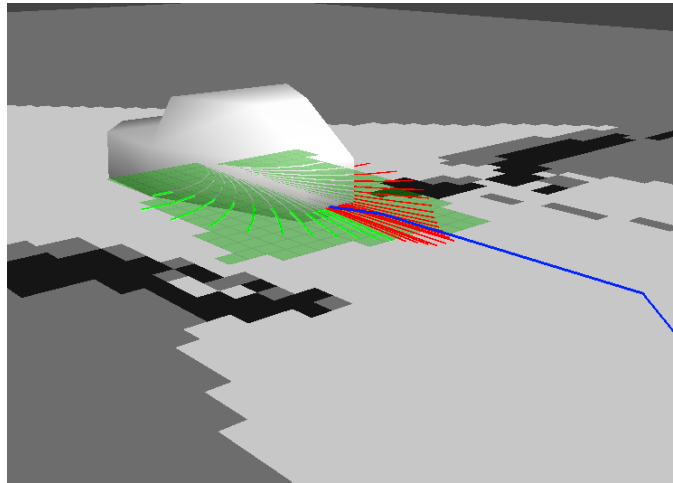


Figure 24: Tentacle planning and A* path planning for waypoint navigation. Drivable paths are shown in green, undrivable ones in red, and the best path in blue. The green grid cells show the cells that the robot will occupy as it follows a path.

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 27 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 24.

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.

1. **for** $tentacle$ in $L_{tentacles}$
2. $L_{distances} \leftarrow \emptyset$
3. **for** each point v in $tentacle$'s path
4. $d \leftarrow$ shortest distance from v to a point in $L_{tentacles}$
5. append d to $L_{distance}$
6. $score \leftarrow$ geometric mean of values in $L_{distances}$
7. Associate $score$ with $tentacle$
8. **return** tentacle in $L_{tentacles}$ with lowest $score$

Figure 25: Algorithm for determining the tentacle closest to the path planned by A*.

4.5.3 Late Obstacle Avoidance

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 collided 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 26, 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.

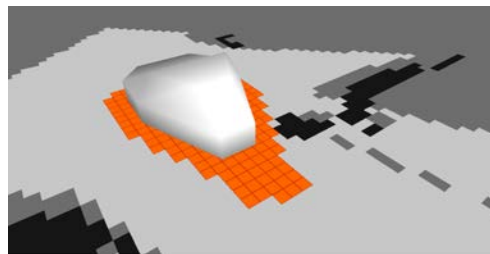


Figure 26: The inflated footprint being used to prevent Prometheus from continuing to follow a path that will cause it to strike an obstacle.

4.6 Lawn Mowing Strategy

The ION competition requires Prometheus to mow as much of the lawn it is placed in as it can within a period of 20 minutes. To accomplish this task effectively, a high level intelligence strategy must be determined to guide the robot. This strategy must determine the area to be covered and plan a path through the area. Since Prometheus already has lower-level path planning functionalities such as waypoint navigation and driving with tentacles, all that is required is the robot to calculate a set of navigation waypoints that follow an accurate and efficient coverage pattern.

Algorithm *TraverseMap*

Input: A list, L_{cells} , of grid cells to be covered

1. **while** $\text{len}(L_{cells}) \neq 0$
2. $targetcell \leftarrow L_{cells}$
3. $targetwaypoint \leftarrow \text{ConvertMowingGridCellToWaypoint}(targetcell)$
4. $path \leftarrow \text{GetAStarPath}(targetwaypoint)$
5. **if** $\text{Exists}(path) = True$
6. **then** $\text{SendNavigationCommand}(path)$
7. **while** $CurrentLocation \neq targetwaypoint$
8. $MowerFootprintCells \leftarrow \text{GetFootPrintCells}()$
9. Remove $MowerFootprintCells$ from L_{cells}
10. Remove $targetcell$ from L_{cells}
- 11.

Figure 27: Algorithm for completely covering the field to be mowed.

Prometheus uses a mowing grid to keep track of areas that need to be mowed. The mowing grid, which is separate

from the occupancy grid used for obstacle detection, is placed relative to the starting position of the robot. This is so that cells do not lose accuracy as the robot rotates. When the mower is on, the robot determines which cells of the mowing grid should be marked as cut. To do this, the robot uses the same technique that is used for calculating the footprint discussed in Section 4.5.3. Cells that are in the mower's footprint are then marked as cut in the mowing grid. The mowing grid is also used to determine waypoints for the robot that will cause it to plan and drive on a path through uncut grass. This is done using the algorithm outlined in Figure 27

5 Conclusion

Prometheus is a robust research platform that will be competitive entry on the ION Lawn Mower Competition; The vehicle's custom aluminum chassis in conjunction with the differential drive train and steered front wheel proved to be an effective and reliable solution. Additionally, the modular payload area allows us to quickly connect the lawnmower attachment. Prometheus can reliably detect lines on grass, detect obstacles, generate drivable paths between GPS waypoints, and traverse through these same paths. The combination of these capabilities along with our lawn mowing algorithm allows Prometheus to efficiently cover a delimited area while cutting the lawn.

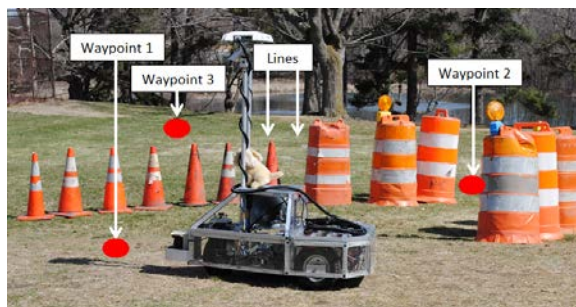


Figure 28: Prometheus in action during the April 8, 2011 demonstration in Institute Park.

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
--	-------------------	---------