PROMETHEUS: WPI's 2012 ENTRY TO IGVC

Authors:

Craig DeMello
Robotics Engineering

Eric Fitting
Robotics Engineering
Computer Science

Samson King
Robotics Engineering

Greg McConnell
Robotics Engineering

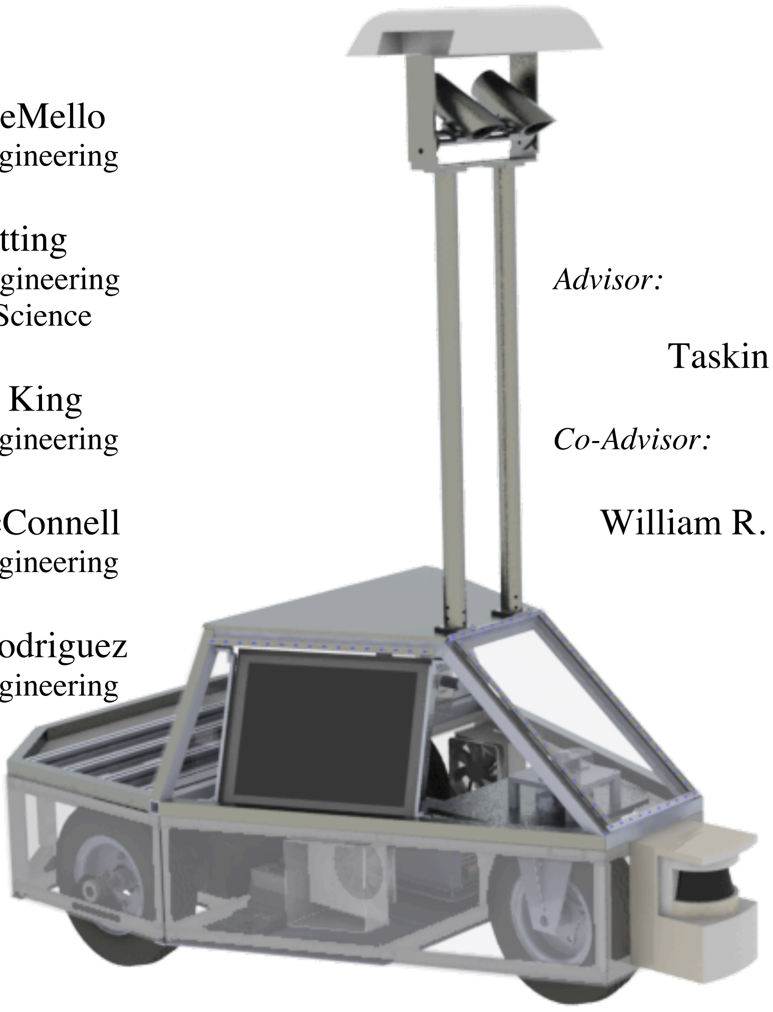
Michael Rodriguez
Robotics Engineering

Advisor:

Taskin Padir

Co-Advisor:

William R. Michalson



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, 2012

1 Introduction

2012 is the third year that Worcester Polytechnic Institute's (WPI) Prometheus will participate in the IGVC. Last year, the 2011 team placed 13th overall and the year before, the 2010 team received Rookie of the Year award for their accomplishments as a first year entry. Although last year's team performed well and improved from the year before, there was still room for more improvement. The 2012 team observed the previous year's performance and determined which areas needed improvement to make Prometheus a more competitive entry for the IGVC.

Prometheus was mechanically and electrically in shape when the 2012 team took over. While last year's team gave Prometheus substantial intelligence capabilities, efforts were heavily focused in this area this year as this seemed to be the aspect that needed the most improvement to be competitive at the competition. The added and improved intelligence include, line detection, localization through use of an Extended Kalman Filter (EKF), path planning, and waypoint navigation.

Efforts were also focused on developing a system that was easily adaptable by newcomers to Prometheus. A system that could be picked up and learned without too much effort so that further improvements could be implemented faster. This was accomplished through removing certain components that separated the sensors from the main computer and went through a different process before it could be fully processed by the main program. Most of the sensors eventually were implemented on the main computer through ROS.

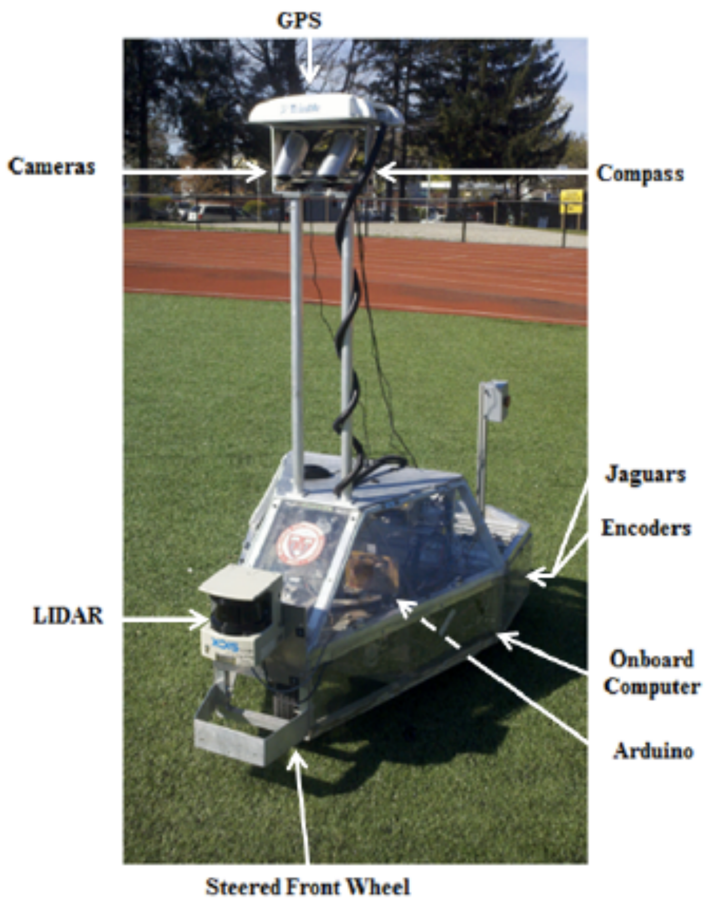
1.1Team Organization

The WPI team is comprised of five members that all have a background in Robotics Engineering. In order to accomplish our goals in a timely manner we first determined what would be considered a project and listed them all. We then split up these projects. For the most part, configuring and implementing sensors into ROS were their own projects. Then line detection, obstacle avoidance, navigation, localization, and stereovision were considered projects and split up among the team.

The approach was very modular. We focused on each project and completed them before moving on to the next. Finally we put all of the projects together to begin testing Prometheus.

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. The vehicle is powered by two interchangeable 12V 55Ah sealed lead acid batteries connected in series.



In order to be competitive in the IGVC, Prometheus has many sensors on board. These include a PNI Fieldforce TCM compass, a Trimble AG DGPS receiver with OmniStar HP subscription, 2x Point Grey Flea2 Firewire cameras, a Microsoft Lifecam, a SICK LMS-291 LIDAR rangefinder, and US Digital optical encoders. Most of these sensors are processed on the onboard computer and an Arduino MEGA handles robot status lights, encoders, and remote control.

3 Robot Structure

3.1 Usability Improvements

Prometheus's main purpose is to be a functional autonomous ground vehicle. However, for the majority of the time Prometheus is being used in testing conditions, where one specific subsystem such as the LIDAR or line detection is being worked on at a time. Due to this, our team decided to place heavy emphasis on several improvements to Prometheus that made development, debugging, and testing of each subsystem easier. The primary usability improvement implemented this year was the removal of the National Instruments cRIO.

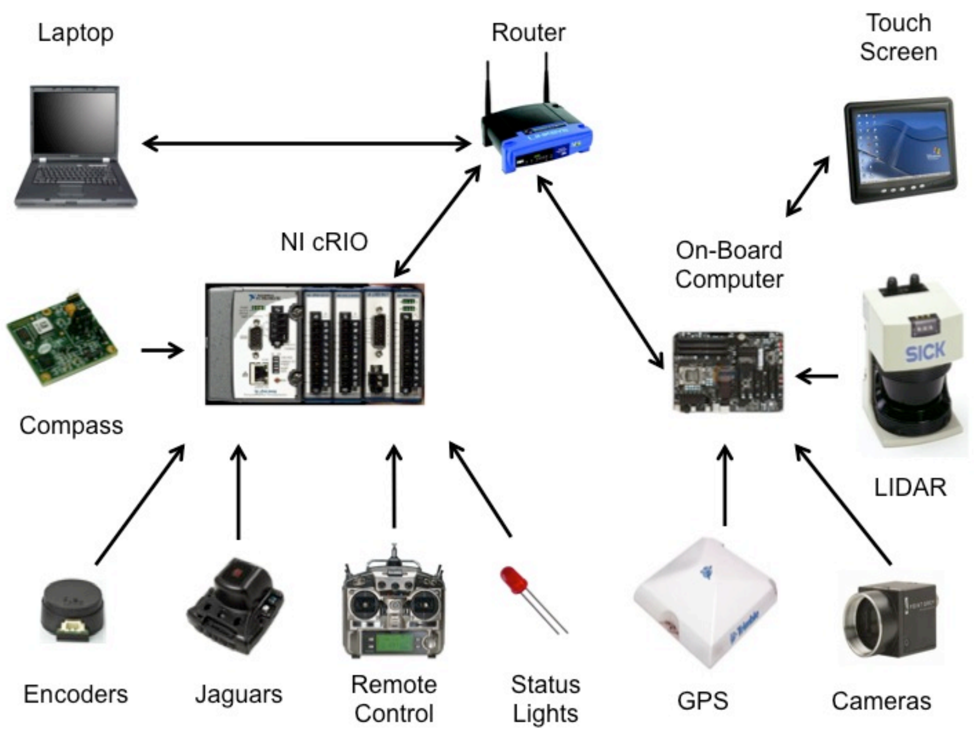


Figure 1: 2011 Prometheus System Architecture

The cRIO played an integral role in Prometheus for the 2011 IGVC interfacing with various low level devices such as the motor controllers, wheel encoders, and compass. In order for the cRIO to communicate with the main computer, which controlled the remainder of the sensors, a special communications protocol was implemented. This communications protocol worked for the majority of the time however when problems did occur there was no debug information and the best option to resolve the error was resetting the system. When combined with the fact that programming the cRIO takes approximately 30 minutes, testing components related to the cRIO was a time consuming and frustrating endeavor.

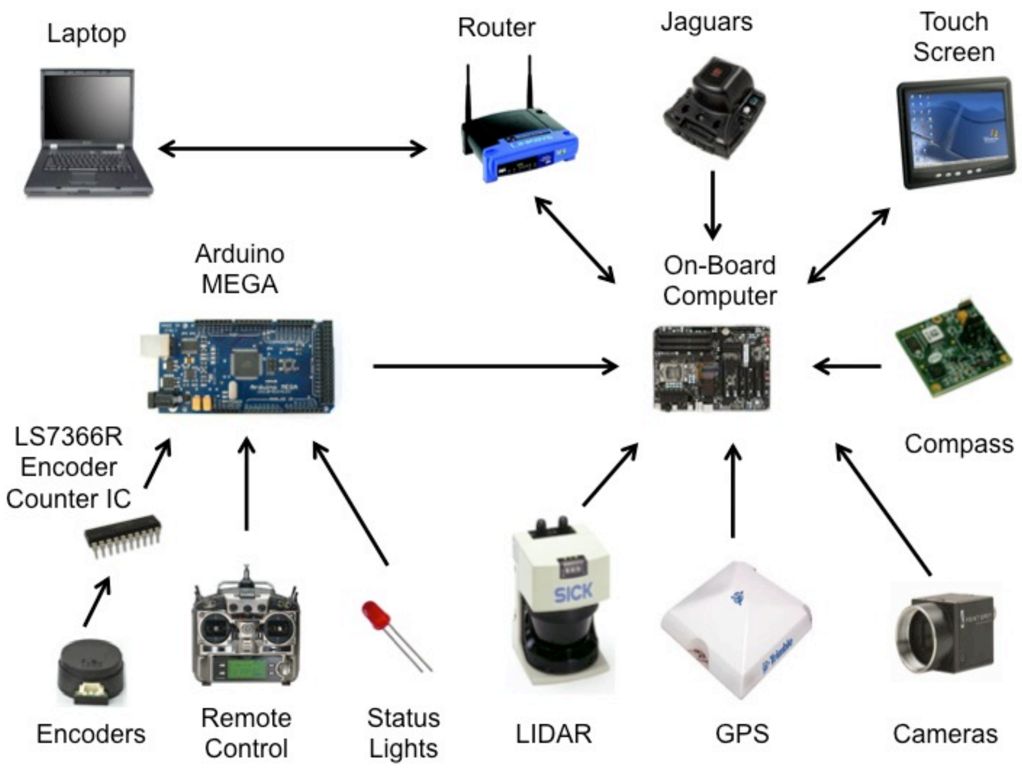


Figure 2: 2012 Prometheus System Architecture

To alleviate the issues with the cRIO the team decided to replace it with a relatively inexpensive Arduino MEGA. The Arduino would be responsible for interfacing with only the lowest level components, mainly the wheel encoders, remote control, and status lights. Even then there is minimal processing occurring on the Arduino. Instead most of the data is passed through it to the main computer. The removal of the cRIO also had the added benefit of locating the majority of the sensors on the main computer, which allowed the team to implement a proper sensor fusion algorithm.

The removal of the cRIO resulted in a new version of Prometheus, which enabled us to test subsystems at a rate unattainable before. Thus making our time more effective and enabling us to accomplish more in the same time frame as before.

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 Arduino to a set of relays that shut off power to the three motor controllers, disabling any movement. The hardware E-stop was implemented in past years and required no improvement. The wireless E-stop however was connected to the cRIO and had to be implemented again when the cRIO was replaced with the Arduino.

3.3 Electrical Design

Prometheus's 24V DC power source comes from two 12V 55Ah Sealed Lead Acid batteries connected in series. Empirical testing performed by previous Prometheus teams has confirmed the theoretical values and shown that the vehicle can operate for one hour and a half with a fully charged set of batteries. Additionally, the vehicle can idle for approximately six hours. When combined with a second set of batteries and two 40A chargers Prometheus can theoretically run for an unlimited period of time.

Component	Max Power (W)	Nominal Power (W)	Idle Power (W)
Computer	400	300	300
GPS Receiver + Antenna	4.2	4.2	1
Drive Motor 1	1200	600	0
Drive Motor 2	1200	600	0
Steering Motor	34.7	15	0
LIDAR	30	20	20
Cameras	20	20	20
Visual Cue LED Strip	12	12	12
Touchscreen Monitor	25	25	20
Total	2925.9	1596.2	373

Table 1: The power consumption of Prometheus's various components.

3.4 Mechanical Design

This year the team felt that the mechanical design having been improved by two previous Prometheus teams was satisfactory. As such no improvements to the mechanics of the robot were made and the majority of our time was spent focusing on the intelligence aspects.

3.5 Sensors

Prometheus 2011 used an array of sensors that gave the robot information about its surroundings. The robot received velocity information from optical encoders on the rear drive wheels, position information from the differential global position system (DGPS) receiver, heading from the compass, and information about any obstacles from a light detection and ranging (LIDAR) sensor. The majority of these sensors worked, however a few still had problems. To resolve these issues some of the sensors were redesigned and others were added.

3.5.1 Wheel Encoders

Due to excessive error, the 2011 team could not rely heavily on the encoder data, making Prometheus unable to navigate to its full potential. This year, the team determined that the encoder data was unreliable because the previous team used a "divide-by" IC before sampling the data. This resulted in a loss of the ability to determine which direction the wheels are turning.

The 2012 team fixed this problem by interfacing each wheel encoder with an LS7366R encoder counter. The LS7366R then communicated the encoder counts to the Arduino, which then passed that information along to the main computer. Once this was done, the team saw much more reliable data from the encoders.

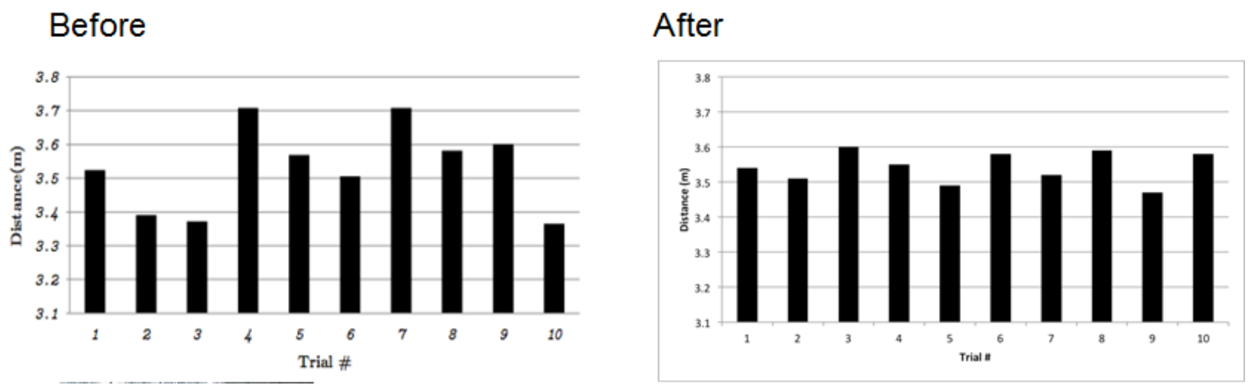


Figure 3: Before and After comparison of the wheel encoders

3.5.2 Stereo Vision

The team decided that to add to Prometheus's reliable obstacle detection stereovision would be implemented. Past years had implemented stereovision several times. However each time various aspects prevented it from reaching production and being used during competition. This year the team believes that their implementation of stereovision is both reliable and robust enough to be used during the competition.

4 Software Architecture

With a focus on modularity and ease of use the development of a robust and user-friendly low-level architecture was an imperative improvement for Prometheus in the year 2012. Having a system that is easily understood and modular allows for future sensors to be integrated with minimal effort. In order to accomplish these tasks it was determined that the Robot Operating System (ROS) would be the best way to organize all of the information that the robot would be processing. It was decided that a ROS mapping package, known as the navigation stack, would be a good starting point for obstacle mapping and path planning. Figure 4 shows the overall software architecture of the system.

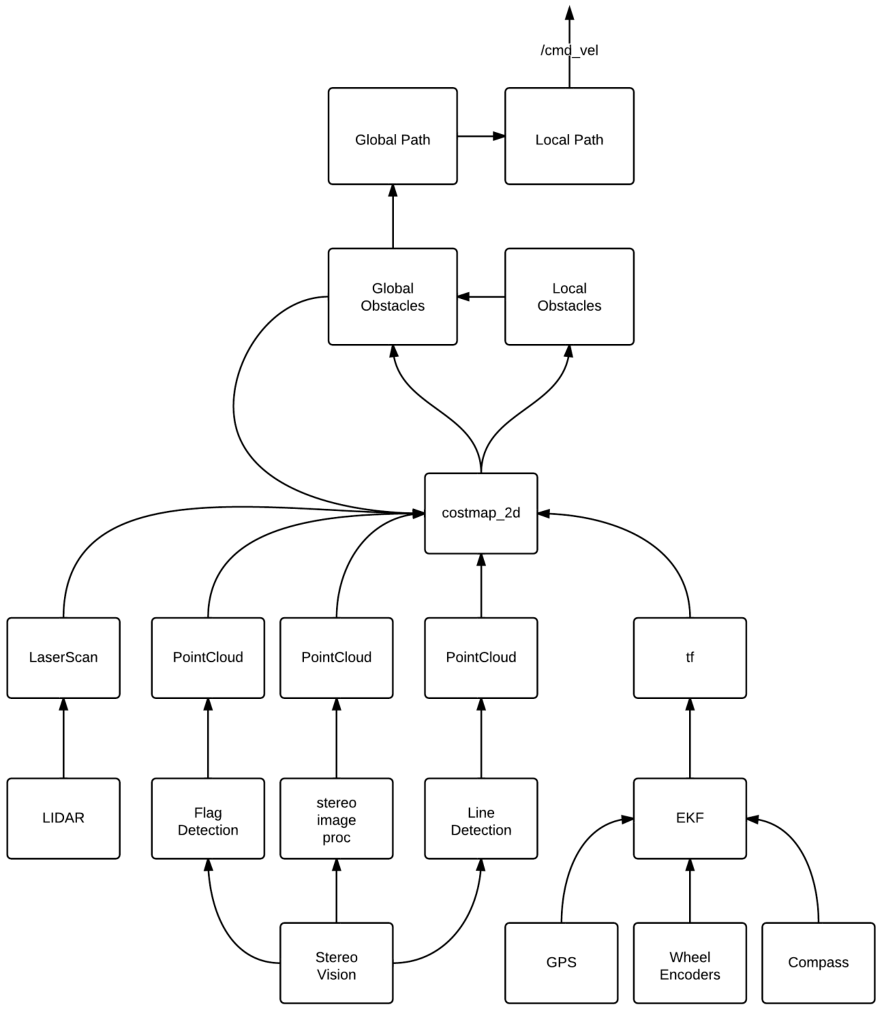


Figure 4: Overall Software Architecture

4.1 Localization

The EKF takes information from the encoders, GPS, and compass to localize Prometheus. Since the GPS and compass are in the units that we needed, the Jacobian matrices for them were identity matrices. Testing the EKF this year consisted of taking the robot outside and hand driving it in a circle. The circle was spray painted on the ground using a tape measure that was planted in the middle and spun to make consistent markings. Information was gathered from the encoders, GPS, and EKF by logging their output into Comma Separated Value (CSV) files. This data was then imported and plotted in MATLAB. The results of a run are shown in Figures 5.

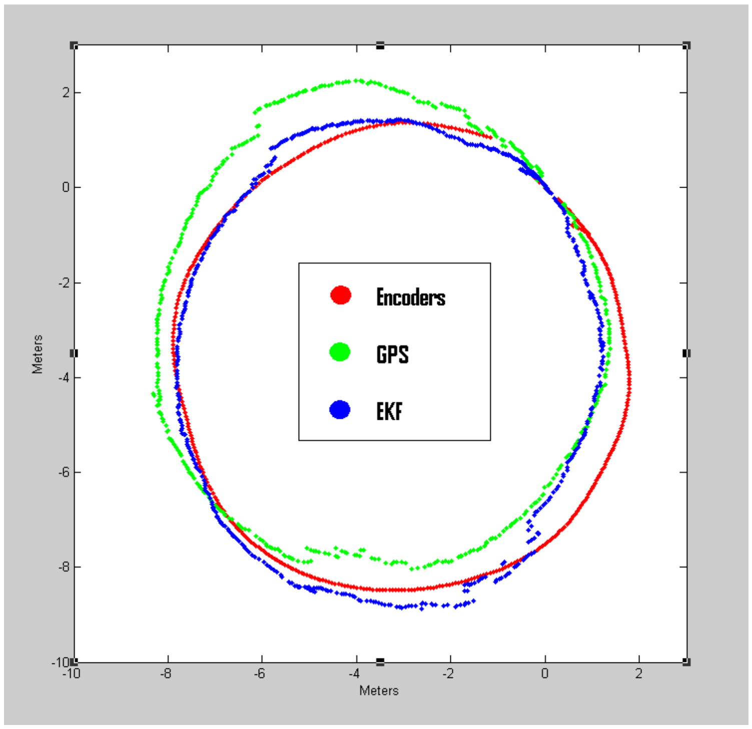


Figure 5: EKF Results showing minimized error

4.2 Line Detection

The 2011 implementation of line detection worked quite well, and was used successfully in the 2011 competition. Line detection is based primarily on the Hough Transform, which determines where lines most likely are in the image by first calculating possible lines that could intersect with each point in the image. Lines that intersect with multiple points are considered to be a line in the image. (Eldar and Bruckstein) The 2012 line detection implementation reuses much of the work completed by the 2011 team. Changes include adapting for integration with our new navigation system, editing the program for easier tuning, and further increasing the reliability by adjusting algorithm parameters.

4.2.1 Methodology

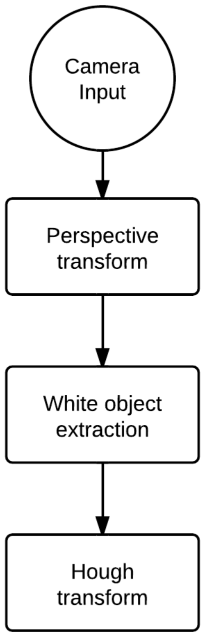


Figure 6: Line Detection Flow Chart

Line detection was implemented using the OpenCV image-processing library. The flow chart above in Figure 6 shows the line detection process. First, the line detection node subscribes to camera feed from the Microsoft LifeCam used for line detection. Next, a perspective transform is performed to remove any distortion from the source image, because the camera is mounted at an angle with respect to the ground. After the perspective transform is complete, the white objects in the image are extracted, and the rest of the image is set to black. Because white lines are the only objects desired, this helps to eliminate noise or false detections in the remaining steps of the line detection process. The desired range of white values to be extracted is supplied as a command line argument when the line detection node is launched. The exact color of the lines can vary depending on lighting conditions, so it is important that this value can be easily changed.

Finally, a Hough Transform is applied to the black and white image. The Hough Transform determines where lines most likely are in the image by first calculating possible lines that could intersect with each point in the image. Lines that intersect with multiple points are considered to be a line in the image. The Hough Transform then returns the end points of found lines.

The line end points are returned in pixels, so they must be converted to distance values in meters, using the measured frame size of the camera. Next, a coordinate system transform must be performed, because the OpenCV uses a different coordinate system than CostMap. After the conversions are performed, a series of points are fitted along each line, to allow them to be represented in the occupancy grid. This is accomplished by using the end points to calculate the slope and y-intercept of the line. The equation y=mx+b is then used to calculate x and y values for each point that lies on the line between the end points. These points are then stored in the ROS Pointcloud format – a 3 dimensional array – for input into the Costmap.

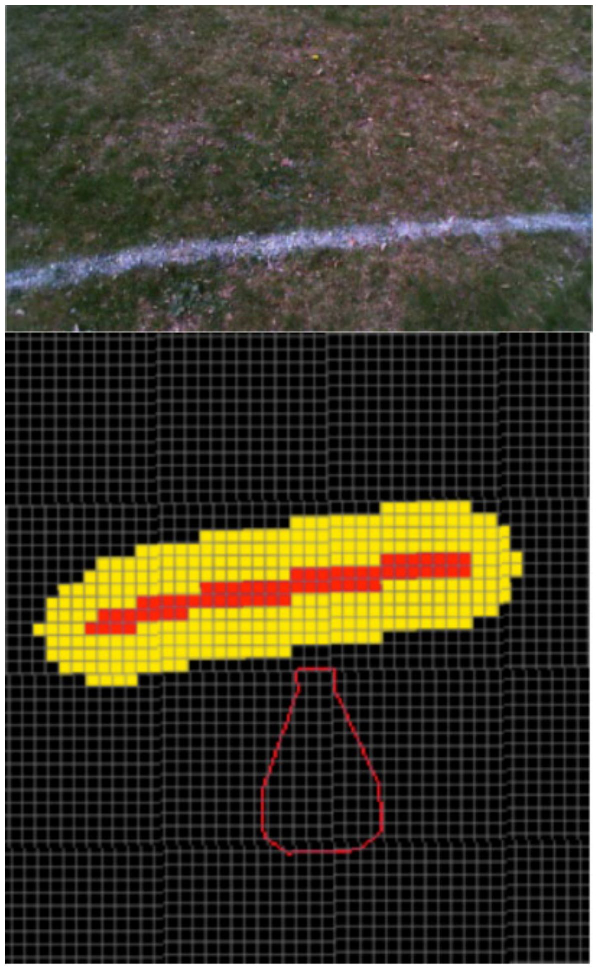


Figure 7: Line Detection Results

Figure 7 above shows an example of the result of the line detection process. The top portion of the image shows the line painted on the ground, as the robot sees it. The image below shows the line after it has completed line detection and been imported into CostMap. As previously mentioned, the red represents the actual line, and the yellow is a margin of safety. It should also be noted that the slope and shape of the line closely resemble the source image.

Outdoor testing has shown that line detection is very reliable once the color selection portion of the algorithm is properly tuned for the current lighting conditions. That is, the shade of white seen by the camera will change depending on the current lighting, and the line detection must be updated to detect only the shade of white the lines are currently. If the range of whites is set too broadly, undesired objects can be detected as lines, such as glare reflecting off the grass, or dead, tan patches of grass. The current approach is to set the range of acceptable white values wide enough to allow compensation for minor changes, while still preventing undesired interference. This method has not presented a significant problem during testing, and is anticipated to work well and be easy to adjust during the competition.

4.3 Navigation System

The 2011 implementation utilized a custom made mapping and path planning system based upon the A* path planning algorithm and tentacle approach for driving the robot along paths. Tentacles function by projecting a series of arcs for the robot to potentially drive on and a series of factors are used to weigh which arc is the most ideal way for the robot to traverse along a planned path. There were several inherent problems with the navigation implementation used by the 2011 IGVC: an inability to properly avoid obstacles that often resulted in often hitting obstacles before avoiding them, and a lack of adjustability that made tuning the navigation system difficult for specific environments. To overcome these problems, the 2012 navigation was based upon the open source Navigation Stack provided with ROS. The navigation stack provides a robust path planning system utilizing Dijkstra's Algorithm, and a tentacle based approach for guiding the robot along planned paths. While this is a similar approach to the 2011 implementation, the navigation stack includes many more features including the ability to inflate a margin of safety around obstacles, and precisely define and tune numerous parameters. Additionally, the navigation stack performs much faster, allowing the robot to detect and react to obstacles before hitting them. These improvements have greatly improved performance and reliability.

4.4 World Representation

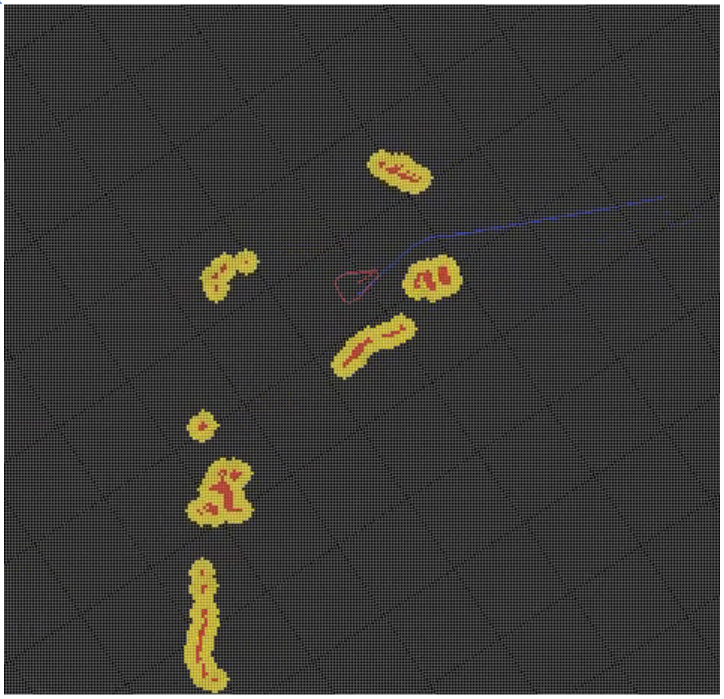


Figure 8: Occupancy Grid

Figure 8 shows an example occupancy grid generated by the CostMap system. Sensor information is used to overlay the position of obstacles and the robot over an occupancy grid. Here, the red grid cells indicate obstacles and the yellow border indicates an inflated margin of safety. The robot's footprint is shown in red, and the planned path to a goal is shown in blue.

When configuring CostMap, two maps are created: the Local CostMap and the Global CostMap. The Local CostMap is used for real time sensor information, and is used to plan the robot's immediate path. The Global CostMap shares the Local CostMap's sensor information, and is used to plan the entire path the robot will take from start position, to end goal. (ROS)

First, the footprint of the robot must be defined. This is a series of points, which allows the navigation stack to model the shape of the robot. Next, sensor information must be configured. The LaserScan is a two dimensional, polar coordinate representation of obstacles, and is the default format for the incoming data from the LIDAR. A point cloud, a three-dimensional array, is used to import data from the line detection and stereovision. This information tells CostMap what topic the sensor information is published on, and provides additional configuration information, such as the obstacle marking and clearing. For each sensor source CostMap also subscribes to a coordinate system transform (tf), so CostMap is aware of the sensor's position, with respect to the center of the robot. Another tf from the EKF provides the robot's current location with respect to the map origin.

Finally, the obstacle inflation is configured. Obstacle inflation provides a margin of safety around obstacles, to ensure the robot doesn't navigate too close to them. The red cells indicate the definitive position of the obstacles, and the yellow cells indicate a margin of inflation for safety, as seen in Figure 9. This is also necessary because the path the robot will follow is calculated from the robot's center. Obstacle inflation allows the navigations stack to account for the width of the robot, and ensure the chosen path won't allow any portion of the robot to intersect with any obstacles.

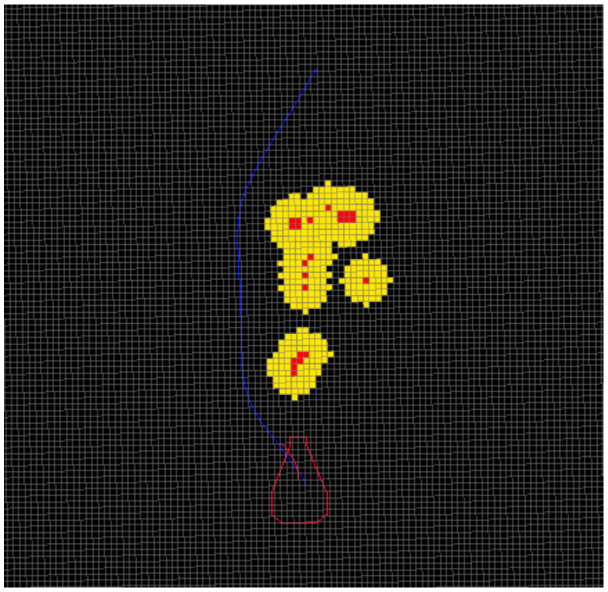


Figure 9: Obstacle Inflation

Once the CostMap is successfully populated with sensor data, the path-planning portion of the navigation stack is configured. As stated above, the navigation stack is aware of the robot's current position by subscribing to a tf published by the EKF. Next, a goal must be sent to the navigation stack, so a path can be planned from the start to end position. (ROS) At the IGVC competition, goals are supplied in the form of a series of GPS waypoints the robot must navigate to throughout the obstacle course. Software was written that allows the these GPS way points to be fed into the navigation stack as a series of comma separated values, and executed sequentially.

4.5 Path Planning

The navigation stack utilizes two components to navigate around obstacles: the global plan and the local plan. The global plan is the complete path and is the ideal route for the robot to follow from its start to end position, calculated using Dijkstra's Algorithm. The local path then tells the robot how to actually drive along this path using tentacles. Tentacles function by projecting a series of arcs from the robot center along the global path. The best arc for the robot to drive along is determined using a weighing system, comparing the arcs proximity to the global path, distance from obstacles, and the estimated time that it will take the robot to drive along the arc. (Burgard and Thrun)

The robots velocity, acceleration capabilities, and acceptable deviation from the global path are specified. Tuning of these parameters is used to achieve a balance between accuracy and smooth driving. That is, setting very wide tolerances results in a robot that drives very smoothly and aesthetically, but may result in the robot deviating too much from the global path and hitting obstacles. Setting tolerances that are too tight can result in very short tentacles that cause the robot to oscillate, jitter, and drive too slowly. Thorough tuning and testing were performed to arrive at the correct values for our specific chassis and outdoor application.

After a tentacle is chosen, the CostMap publishes information that tells the motor controllers how to drive along this tentacle. This information is published over the topic "cmd_vel" and specifies the linear and rotational velocities necessary for the robot to move along the desired path of motion.

After tuning, the navigation stack has proven to be very reliable. Sensor information is taken in from multiple sources, and fused into one map. The following video - https://vimeo.com/40559488 - shows a qualifying run completed by the robot, where it successfully avoids lines and obstacles while navigating sequentially to three GPS waypoints. This was a significant accomplishment – ensuring that all implemented systems were working together, and the robot met the base requirements for the IGVC. The robust, easy tuning provided by the navigation stack should be a great advantage that will allow the team to quickly tune the robot for specific conditions encountered at the competition.

5 Conclusion

Prometheus 2012 shows many improvements over Prometheus 2011. The main focus was intelligence and it has definitely performed better. The software architecture was overhauled and redone to be far more efficient and easily adaptable. New path planning algorithms, localization filters, and obstacle and line detection methods were implemented. Prometheus is shown during a practice run in Figure 10.

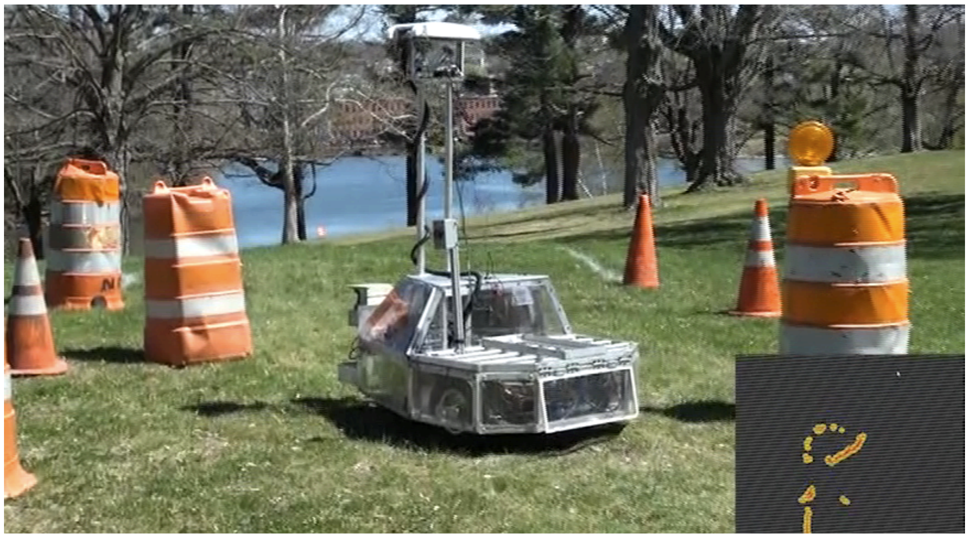


Figure 10: Prometheus during a practice run at Institute Park

References

ROS(2010)

Retrieved from http://ww.ros.org/ (Accessed on 2010-12-17).

Burgard and Thurn

D. Fox, W.B., & Thrun, S. (2004). The Dynamic Window Approach to Collision Avoidance. Retreived from http://portal.acm.org/citation.cfm?id=1405647.1405650