Proteus

Oakland University

2005 IGVC Design Report

Design Methodology

The design of *Proteus* began almost a year ago by asking a simple question - "What should our robot do?" While this may seem like a simplistic question, it paves the way for the entire design and implementation process for our robot. After discussing the official rules, possible ideas, and experiences from previous competitions, our team came up with the following requirements for our robotic system:

- Simple to manufacture, maintain and operate
- Easily expandable to add new hardware and software
- Able to operate in adverse environments such as snow and rain
- Powerful enough to maneuver thru bumpy terrain and climb steep inclines
- Implement simple solutions instead of complex ones when possible
- Over two hours of continuous runtime
- Detailed displayed information on running software processes
- Fool-proof safety systems which ensure control during worst-case scenarios

After creating these requirements, our team began designing the hardware and software systems which would later be implemented to create a functioning system. During all phases of development, the requirements listed above were continuously referred to in order to ensure that our design goals would be best met.

Team Organization

After determining the design requirements for our robot, our team was broken into smaller task groups based on knowledge and interest to further investigate, design, and implement individual components within each task group. While the entire team was always made aware of the current research and status of the other task groups, individual groups were given the authority to implement each component after discussing the feasibility of the underlying reasons and principals of components. The team was divided into the following task groups:

Task Group	Team Member	Standing	Major
Software Systems	Brian Clark	Senior	Comp. Sci.
	Kevin Hayes	Senior	Business
Motor Control	Jason Rutherford	Senior	Comp. Sci.
Vision Processing	William Clements	Junior	Comp. Sci.
Electrical & Structural	Jason Finch	Senior	Elec. Eng.

The task group *Software Systems* was primarily responsible for creating the core software controlling the robot, including path planning, sensor integration and obstacle avoidance. The *Motor Control* task group was responsible for implementing a motor control system which provides easy control over the motors and wheels, resulting in proper movement of the robot. The *Vision Processing* task group researched and developed software algorithms and capture techniques to identify relevant objects of interest from cameras and provide locations of identified objects to the core system for analysis. Finally, the *Electrical & Structural* task group was responsible for the design and construction of the frame, cover, and electrical systems per agreed physical requirements. The entire team collaborated and created a timeline for critical events for each task group during the development and integration of the robot, with the culmination of the timeline ending in a complete system abiding to our design requirements ready for field testing.

Hardware Overview

Frame Construction & Design

The structural, electrical and sensory equipment on *Proteus* was designed in such a way to give the most flexibility in component placement, with a heavy emphasis on future expandability for experimentation and research of other technologies. Keeping this in mind, the following is an overview of the physical construction and implementation of all physical items on the robot.

The frame of the robot is constructed out of 1-inch extruded aluminum. By fastening the aluminum together using end screws and simple aluminum brackets, we are able to quickly build an extremely stable frame in a matter of a few hours. Additionally, by using our method of fastening aluminum, we are able to quickly make structural changes to our frame in a matter of minutes. For example, we can untighten some screws, move a support beam a few inches, and tighten the screws back down again to accommodate a larger payload.

After reviewing our design requirements, we constructed the frame to be compartmentalized in nature; this is to say that there are several different sections in the frame design; each with their own purpose. The frame is rectangularly shaped and divided into several layers in which to hold equipment. The four corners underneath the

frame were designated as motor mount positions. Directly above them inside the main frame is an area to accommodate any motor control or electronics for the drive system of that individual wheel. The center of the robot was lowered for battery storage and accommodations, and the left side was reserved for mounting sensory equipment. The right side of the robot was designed to hold the controlling electrical systems, and the top layer was slated to hold the master laptop and external sensory equipment. Once again, it was easy to change the dimensions of each section due to the nature of our frame construction. Additionally, the nature of the extruded aluminum allowed us to place plexiglass within each section if so desired to allow for the secure mounting of robotic components.

Sensory Equipment

The modularity of the frame design allowed for the efficient selection of sensory equipment which would be guiding the robot. The sensory equipment can be classified under two categories; equipment which is used to determine the robot's position (Internal Sensors) and equipment which is used to detect objects in the outside world (External Sensors). Outlined below are the specifics of the sensory equipment in both of these categories.

There are several components which consist of the "Internal Sensors" system. The first of these is a Novatel OEM4 GPS unit. This device uses the Global Positioning System of satellites to determine where on the planet the robot is physically located. Readings are received from the GPS unit once a second, which is timely enough for the robot to make any path planning decisions. We have achieved an accuracy of 1.5 meters with this system when utilizing WAAS correctional factors. With this accuracy, we are extremely confident in the robot's ability to accurately identify it's location with respect to other entities in the world around it.

In addition to the GPS, there is also an Inertial Momentum Unit (IMU) manufactured my Crossbow Technologies on board the robot. It is mounted close to the center of gravity, allowing for accurate readings on the accelerations for all six axis of motion (X, Y, Z, Roll, Pitch & Yaw). Accelerations are measured out to over ten decimal places, giving us extremely precise readings. These readings are taken from the IMU every twenty

milliseconds, making us able to accurately calculate direction and velocity based on acceleration history.

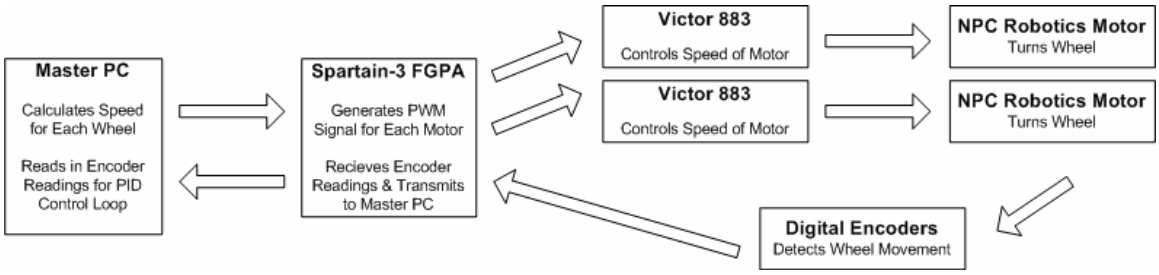
External Sensors provide the means for our robot to determine any and all entities and obstacles which exist around it. One of these sensors is a SICK Laser Rangefinder. This unit, which is mounted in the front of the robot approximately 12 inches from the ground, is able to determine if any physical objects exist in front of the robot. The SICK scans a 180-degree arc in front of the robot, with individual readings every half-degree. Each reading gives us a distance to an object up to ten meters away, with readings accurate to the centimeter. Using this information, we plot a picture of the world directly in front of the robot according to the closest detected objects. This "picture" is then used to identify obstacles (such as barrels) in front of the robot and avoid them, as will be explained below.

While the SICK Laser Rangefinder can identify physical objects, we also need a system in place to identify objects visually, such as a line or pothole. To this end we use two Logitech Quickcam 4000 cameras on either side of the robot to construct a visual picture of what is in front of it. The cameras are angled downwards and to the sides of the robot to give the best possible angle of any lines or potholes which are in the area. These cameras are connected to the main system using high-speed USB connections, and produce a 640x480 pixel picture each which is then analyzed and manipulated within vision processing software.

Drive Systems

The robot drives itself via a simple, skid steering method. The rear two wheels are unpowered castors, and the front two wheels are powered tires locked in a forward position. The front motors are heavy-duty 24-volt motors from NPC Robotics, with a maximum speed of 235 RPM. During our field testing these motors have proved themselves as extremely reliable and weather-resistant, and have been able to handle whatever speeds and forces were demanded of them. Each of these two motors is independently regulated by a Victor 883 Motor Controller. The Victor motor controllers control the speed my which the NPC Robotics motors rotate, from a minimum of 3 percent to a maximum of 100 percent. These Victor motor controllers are controlled by a

PWM signal which is generated by a custom processing program on a Spartain-3 FGPA board. This FGPA board is running custom software written by our team to take command packets send by the master PC via RS232 and generate the appropriate PWM signal to the Victor motor controllers. The FGPA board also takes reading from digital encoders mounted on the wheels and transmits them back to the master PC for use in a PID loop to ensure that the wheels are operating at their desired speeds.



This skid-steering drive system is a result of one of our design philosophies, specifically to implement simple systems instead of complex ones where appropriate. While in the past we have experimented with advanced drive systems and holonomic steering, we felt that for this competition we should focus on the essentials of obstacle avoidance instead of struggling with perfecting an advanced drive system. Controlling only two motors is much simpler than coordinating eight motors as we were in the past, and has proven to be more reliable and nearly as robust as other options which we have researched.

Electrical Systems

One of the main requirements in our robotic system was to provide a long runtime at full-load capacity – at least two hours. Additionally, downtime between runs had to be minimized in order to maximize the time available for testing and competition. While a combustion system was briefly considered, we determined that the upside of having continuous runtime was out shadowed by the limitations of expensive fuel and the inability to run indoors. To that end, we decided to power our robot using two Optima 12-volt deep-cycle marine batteries. These batteries are rated at 75aH each, and have proven themselves by giving us over three hours in the field at full system loads with recharge times of less than two hours. With these runtime statistics, we are able to

perform extended tests on the robot in the field, and use the short downtime to correct problems which are noted.

In order to power both 12-volt and 24-volt components, the two batteries were connected in series and wired to a main electrical box. Inside the main electrical box are terminals for both the 24 and 12 volt power sources, which are appropriately wired to toggle switches on the front of the electrical panel. Each switch is connected to a pair of wires which control a single device on the robot; using this design methodology we can turn components on and off as we please without causing harm to any other components. There are individual circuits for each of the two drive motors, the GPS & IMU, the SICK Laser Rangefinder, the Spartain-3 FGPA Motor Controller, and the master PC. Each of these lines have their voltages regulated and have breakers installed to provide safe, customized power requirements to any power-drawing devices. The design of our electrical system allows for the quick and easy addition of new power lines by simply running two wires and hooking them up to a switch.

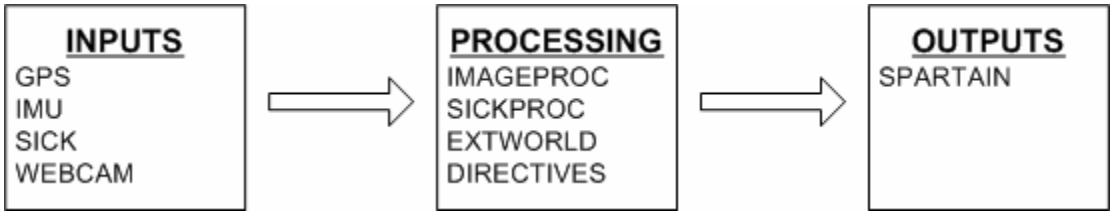
The Emergency-Stop (ESTOP) system is crucial in order to maintain proper control over the robot. With our current ESTOP system, there is a custom relay board sitting between the Spartain-3 FGPA and the Victor motor controllers. This system works in a default-off scenario, meaning that unless a proper signal is received, the PWM signal will not be passed to the motors and they will remain stationary. A “GO” signal is transmitted from a hand-held wireless control box to a receiver on the robot. From there, the signal passes thru a large physical push-button switch located on the frame for emergency use. The signal can then finally pass from there to a relay on the ESTOP board, which can then allow the PWM signal to traverse the relay and continue on for motor control. If at any time the GO signal is lost or the circuit is broken via the manual pushbutton, the relay snaps open and the signal going to the victor motor controllers is lost, resulting in the victors coming to a controlled stop.

Software Overview

The design of any software system requires the precise definition of how individual components, starting with the basics of any software system: Inputs, Processing, and Outputs. In doing this, we identified components and tasks which would fall under each

of these three categories. Sensory equipment (GPS, IMU, SICK & Webcams) provide the inputs to our computer system, and the main output of our computer system is a speed signal for each wheel. Everything else in the system lies in the center under the umbrella term “Processing” which turns the raw data generated from the sensory equipment into meaningful movement of the robot itself. The following will document how we take our inputs, process them, and turn them into valid outputs for use in the competition.

All programming done for the master PC was coded using C++. The use of this language allows us to create software “objects” pertaining to individual components of the robot, with each object having properties and methods pertaining to that specific object. For example, the GPS software object is responsible for acquiring and organizing data from our Novatel GPS unit. This object works independently of any other software objects in the system, eliminating the chance of accidental data corruption from any other objects. The programming methodology for each object is identical to that of the entire system as a whole; this is to say that each object expects inputs, performs processing, and delivers outputs which then can be used by other objects. This leads back to the main feature of object-orientated programming, specifically that drastic changes can be made from within the “processing” portion of each object and there is no need to change any other code elsewhere in the system because the expected outputs still exist in a documented format. For implementation, a strategy was developed for developing software objects from within our software system. The software task group brainstormed and created a list of objects which could be represented as different entities from within our code. These entities included objects simulating physical devices, such as the GPS unit and the motor controller. They also included more logical constructs such as data buffers, communication ports and the external world. Once this list of objects was created, they were linked together by deciding which objects would need to access the data from other relevant objects.



Once our module list was created, the task of implementing it commenced. Our team focused on an "outwards-in" approach to coding the modules. Using this approach, priority was initially given to objects on the "outside" of the object roadmap, and as objects were coded and completed, priority was then given to objects closer to the center of our diagram. This was done for a few reasons; one of them is simplicity. Reading data from a device is simpler then constructing a complex driving algorithm, and the modules near the outside of the roadmap are simpler in design and context than those in the center. A second reason why this coding approach was taken was so we could focus exclusively on path planning and processing as the final task. Reliable inputs and documented outputs were needed in our system before any credibility could be given to any core processing and planning which would take place; otherwise we would be uncertain if any occurring problems would actually due to processing error.

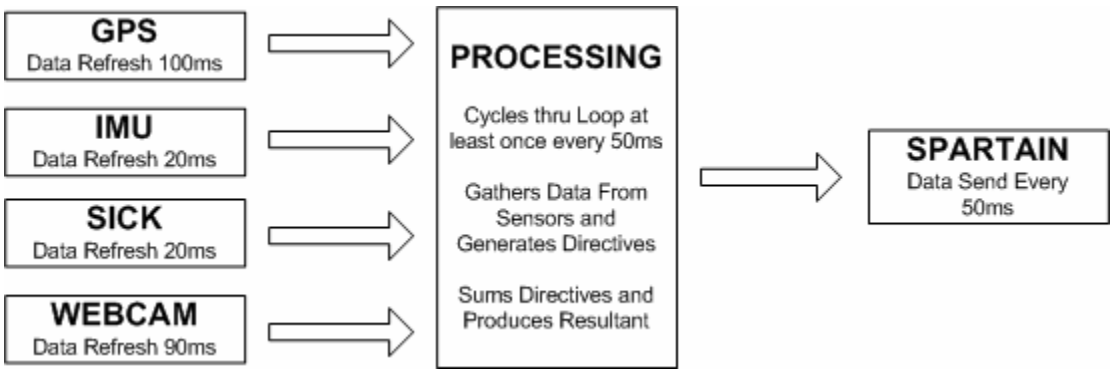
The main display of our software system was designed so that the maximum amount of information possible could be displayed for debugging purposes. This allows programmers to see all relevant information on the inner workings of the robot in real time during field testing. This dashboard also allows us to turn individual modules on and off, and change the operational state of the robot to either off, manual control, navigational challenge, or autonomous challenge. The operational state of the robot controls which modules should be turned on at appropriate times, and assigns "directives" specific to that particular operational mode, which are explained more in detail below. Additionally, software support for "preferences" was enabled for quicker debugging purposes. With preferences, we are able to change critical variables inside the software system such as thresholds, speeds and configuration values. This can be done while the robot is running, eliminating the need to stop the program, recompile, and start back up again. This has reduced debugging time in the field from several minutes per issue to several seconds.

Systems Integration

All processing on our robot is accomplished by a Compaq Laptop PC which is mounted on top of the robot. This laptop is responsible for coordinating all relevant hardware using the software system described above. The PC communicates with sensory

equipment and the motor controller using both RS232 and USB communication protocols. Once power is applied to the entire robot, all robot inputs (such as the IMU) start sending data, and robot outputs(such as the Spartain-3 FGPA) start expecting data, so it is up to this master PC to coordinate all incoming data, process the data into meaningful results, and output it to the motor controlling system in a timely fashion.

As previously stated, software objects were used to represent physical objects such as sensory equipment on the robot. Once again, we were able to use this fact to our advantage when designing how all systems would integrate. The independent nature of the objects allowed us to independently "thread" data acquisition and sending routines so we could take advantage of multithreading performance in our laptop PC. With this, we are able to independently refresh sensor information, perform path processing, and send data to the motor controller system separately at appropriate refresh rates.

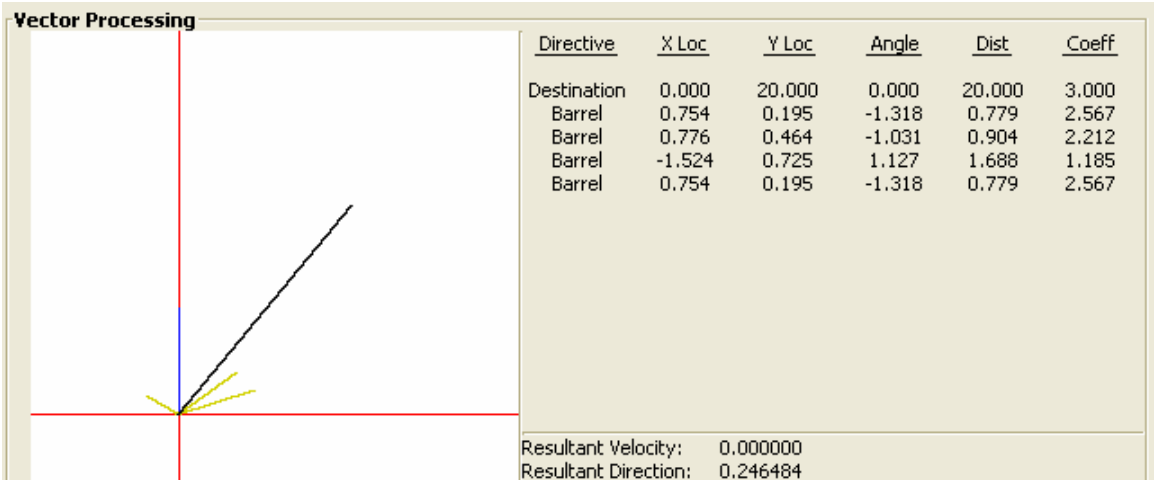


By performing these independent threading techniques, we can reliably achieve an efficient processing loop time. The main process loop is guaranteed to cycle at least once every 20 milliseconds, meaning that the robot is calculating what it should do over 50 times a second. We have confirmed that processing power is plentiful enough to allow for such calculations, giving us a reliable indication that the robot is constantly made aware on it's surroundings and what it should do about them.

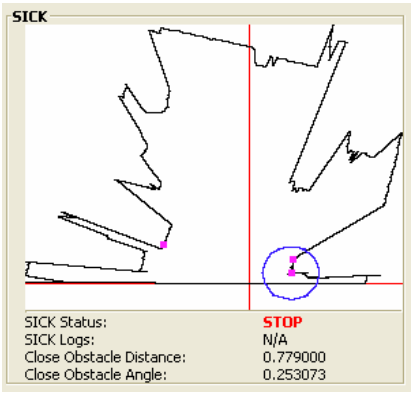
Path Planning & Obstacle Avoidance

The navigation system of *Proteus* is based on the concept of "directives". A directive is generated by sensory equipment from within the robot, and represents an object in the external world that the robot either wants to advance towards (an objective) or retreat

from (an obstacle). Each directive consists of an angle the robot should attempt to achieve, and an urgency or importance of said directive. These two components then combine to create a vector for each directive. Since directives are generated automatically based on information from the sensory equipment, simply adding up all of the active directive vectors results in a single resultant directive for the robot to move in. This resultant vector is then sent out to the motor controller FGPA for translation to PWM signals to drive the two wheel motors.

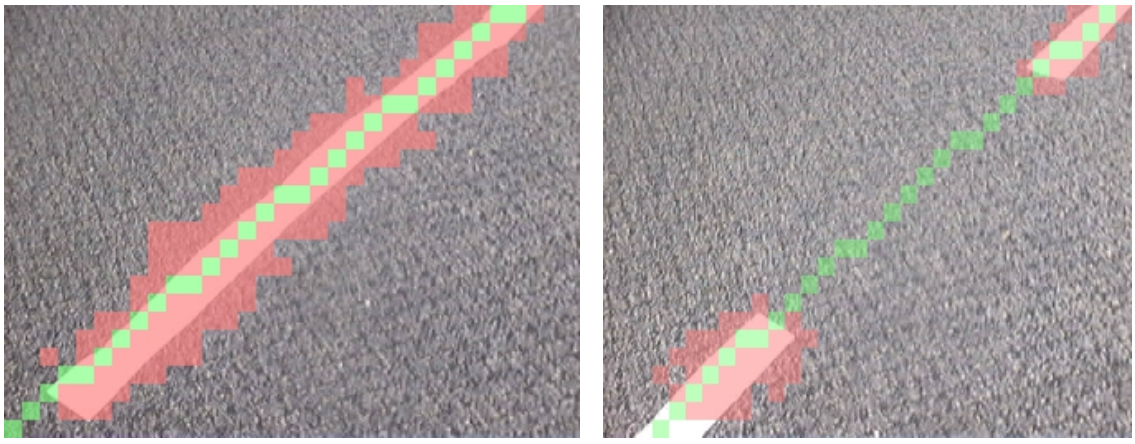


As mentioned, sensory equipment generates the directives which determine robot movement. Depending on the operational mode of the robot, different equipment will generate different directives. The SICK Laser Rangefinder detects physical obstacles in front of the robot. By looking at the data and finding large changes in distance between adjacent points, we can identify individual components such as buckets and barrels and assign a single directive on each obstacle based on it's proximity to the robot and it's angle from the front of the robot. As the obstacle gets closer and closer to the robot, the intensity of the directive increases, forcing a direction change which results in the robot avoiding the obstacle.



While physical obstacle avoidance is crucial, detecting while lines is a critical portion of the Autonomous challenge. To this end, we have

created software algorithms which capture images from the mounted cameras, identify white portions of the images based on configurable thresholds, and calculate a straight line based on the picture information. Each large 640x480 image is broken into segments, each segment being part of a 64x64 grid. Each segment is then analyzed for whiteness based on thresholds which are easily configurable from the field. After determining which portions of the image are white, another algorithm is run to calculate a tangent line connecting the white segments of the picture. Once this line is calculated, it is simple to generate a directive which steers the robot away from the line as it gets closer and closer to the robot. As shown below, this method works extremely well for both solid and dashed lines. The red areas are sections where the software "sees" white, and the green line is an indication on where the line should then be. The nature of creating a tangent line means that the robot will "see" a line even when one does not exist in it's entirety; so long as a small bit of information is available, a line is generated producing a obstacle directive for the robot to move away from.



As stated before, the robot takes the sum of all generated directives to determine its resultant direction and speed. We do not actively plan out where the robot will drive in the future; instead we use a passive method of always positioning yourself to go in the best direction based on available information. For the navigation challenge, the only directives which are generated are by the SICK Laser Rangefinder for obstacle detection, and by the GPS for determining which location to drive towards. In the case of the autonomous challenge, the SICK and the two cameras provide directives for obstacle avoidance, while we add a third directive for constant forward movement to facilitate the

actual running of the obstacle course. Once again, the modularity of the code allows us to change the inclusion of these directives at our will, so we can create additional modes of operation using whatever generated directives we see best fit for the robot to use in determining what path it should take.

Performance Benchmarks

Proper performance is engrained in virtually all of our design requirements as listed at the beginning of this report. Reported below are the performance figures achievable by our robot system, which have been mathematically estimated as well as empirically confirmed during field trials:

Speed

The motors we chose for implementation on our robot were selected for their heavy-duty torque and performance. Because of this, the motors themselves have a high top speed of over seven miles per hour. Due to safety considerations, however, the master PC only sends signals of less than five miles per hour. As an extra safety feature, the Spartain-3 FGPA motor controller will not send out PWM signals to the Victor motor controllers which would exceed these limits.

Ramp Climbing Ability

We have tested the maneuvering capabilities of *Proteus* throughout the various landscapes on the Oakland University campus. One of our proudest points of these abilities is that the robot can successfully traverse a 35-degree hill outside one of our buildings in both directions. During it's ascent and descent, the robot maintains control of itself and still functions to the maximum capabilities as expected during either the navigation or autonomous challenges.

Reaction Times & Battery Life

Due to the independent modularity of the code, data acquisition and processing is handled independently, resulting in extremely quick reaction times. The main process loop cycles over 50 times a second and has priority over all other processes so that performance does not degrade. CPU Utilization has not exceeded 30% in any of our trials, so we are not worried about a lack of computational resources under full system loads. In terms of

battery life, we have experienced over three hours of continuous runtime out in the field, with only two hours needed for a batter recharge. This culminates in an extremely fast debugging cycle allowing for the most time available for correcting any problems noticed in the software system.

Obstacle Detection

The main sensors used to detect obstacles in the external world are the SICK Laser Rangefinder and the two cameras. The SICK can detect and classify obstacles over 30 feet away, and the cameras can detect white lines and potholes up to15 feet away based on the angle of the camera in relation to the robot. This means that the robot has a very good idea on what obstacles are around it at all times. Because of this, it can easily identify areas which may be considered traps or dead-ends. While path planning does not actively go out and find the best route for the robot to take in the future, the area by which the robot can detect obstacles is large enough to determine that going in the direction of a trap or a dead-end is not the best idea. As for pothole detection, potholes are detected in the same fashion that white lines are; any potholes in the course are detected as extension of the forbidden white lines and maneuvered around accordingly.

Navigational Accuracy

When configured with an accurate WAAS signal, our GPS unit can achieve an accuracy of approximately 1.8 meters. Combined with correctional measures obtained from the Inertial Momentum Unit (IMU), we can achieve an accuracy of approximately one meter in an open field. Given that the GPS antenna is installed in the center of our robot, we expect the robot to consistently come within 50 centimeters of any point on the course.

Cost Estimate

The following is a breakdown of the costs involved in building this robot, including materials and equipment.

Extruded Aluminum	$300
Plexiglass	$60
Brackets & Screws	$65
Castors & Tires	$120
Electrical Wires & Enclosures	$100
Batteries	$185 each
NPC Robotics Motors	$285 each
Victor Motor Controllers	$149 each
Crossbow IMU300CC	$300
NovAtel OEM4 GPS	$3,000
SICK LMS200-6	$6,000
Logitech Quickcam 4000	$89 each
Spartain-3 FGPA	$100
Compaq Presario R3000 Laptop	$1,300
TOTAL	**$12,761**