

# Old Dominion University

## AUVSI RoboBoat 2012 Competition

### FINAL REPORT

April 26, 2012

MAE 435: Project Management and Design II

Dr. S. Bawab



### ME Advisor

Dr. Gene Hou

### ECE Advisor

Dr. Chung-Hao Chen

### Graduate Student Advisor

Stanton Coffey

### ME Senior Project Students

Kenley Fong-Sam

Eun-Sil Heo

Rachel Mittelstaedt

Kyle Rogachenko

Stevin Rossman

Steven Schesventer

Jason Toutkoushian

### ECE Senior Project Students

Robert Tolentino

Justin Maynard

Haole Guo

### Volunteers

Kevin Mcleod

Daniel Becker

Jeff Roper



## **ABSTRACT**

The Old Dominion University (ODU) Autonomous Surface Vehicle (ASV) Team (ODU ASV Team) has designed the 2012 model of Little Blue, an autonomous surface vehicle, to participate in the 5th annual RoboBoat Competition hosted by the Association for Unmanned Vehicle Systems International (AUVSI) Foundation held June 21-24, 2012. For the past three years students from the mechanical aerospace engineering and the electrical and computer engineering departments have participated in preparing the team for the competition.

Unmanned vehicles are required in many applications. The competition allows students to participate in the drive for autonomous robot technology. Throughout the War on Terror engineers have been developing robots that can take the place of troops called Unmanned Ground Vehicles (UGV). Much of the technology developed for UGV can be applied to the ASV and vice versa. The team at ODU has integrated multiple sensors that will detect a course made up of green, red, yellow, and blue buoys, a heat source, and an underwater object. The input from the sensors will be continuously streamed through an onboard computer. The computer will control the power and auxiliary outputs in order to navigate through the course, complete various secondary tasks, and return to the starting point.

## **BACKGROUND**

The ODU ASV Team experienced a major transition between the fall 2011 and spring 2012 semesters. A large amount of new students joined the team and many students left. During the fall 2011 semester a custom computer was built to handle the large processing requirements of the computer vision (Section 6), and additional sensors (Section 7). A gimbal mount was also designed during the fall semester, and was constructed in the beginning of the spring semester. The fall team members decided to purchase a LIDAR sensor and began interpreting the raw data into usable object locations.

The spring team members completed a working model to be competitive in the 2012 competition. This required a reassessment of goals including: a new hull design (Section 2), a power source upgrade (Section 3.4), and fully operational navigation logic (Section 8) relying on sensory input and motor output through serial communication.

## **HULL DESIGN**

The team was able to successfully design and construct a vehicle that was lighter and more maneuverable than the 2011 model.

After considerable research, sailboat stabilizers were chosen to be used as pontoons (shown in Figure 1 in Appendix E). This option provided a pre-constructed pontoon made of high-impact

plastic that was much more durable than was feasible to create with an equivalent size and weight.

A simple buoyancy calculation estimated that each stabilizer should support approximately 35 pounds; therefore set of three were purchased to ensure enough buoyancy. Each stabilizer was six inches wide (which easily allowed for the triple hull design) and 42 inches long (also within size requirements).

The framework used on the 2011 model was a heavy fiberglass unistrut. Various replacement frame designs and materials were proposed to reduce weight while retaining or even adding stability. An adjustable modular frame was constructed to experimentally determine an optimal design. The modular design consisted of a frame mostly made from perforated aluminum angle (Figures 2 & 3, Appendix F) and included a deck to seat the computer and other requirements. A 3/32" thick polypropylene perforated plate with 1/2" center-to-center spaced 1/4" holes arranged in a straight pattern was chosen for the decking (Figure 4 & 5, Appendix E).

#### *Finalized Structure*

The pontoons were able to move significantly forward and back along the angle to form a total possible length of over six feet. The final placement of the pontoons was determined first by static testing as shown in Figure 6, Appendix E (using weights in the place of the electronics box.) After the vehicle was statically balanced dynamic tests were performed using a remote control. The vehicle was subjected to the most extreme possible maneuvers and adjusted until no water went over top any part of the pontoons. (See Figures 7 & 8 in Appendix E)

The total fully assembled weight is approximately 57 pounds. Which is a 50% weight reduction from 2011.

#### **MOTORS AND POWER**

A dual motor system was adopted for previous models as opposed to a rudder design, only using forward and reverse to steer. This system will remain in place for multiple reasons. A major reason is that the motor commands will have the same type of output allowing verification with previous software written for the vehicle.

The Motors and Power Team has selected replacements for the motors. The new motors, Watersnake ASP T18S, are considerably lighter, reducing the overall weight by more than ten pounds. Three of the new motors were purchased to have a spare and for testing purposes. The new motors have a propeller that is much more forward-thrust bias than the previous motors. The propeller may need replacing due to this fact. Also, counter rotating propellers would be

ideal, but the prop walk can be taken into account in the motor output code. A new shroud is also being constructed for the new smaller motors.

The motors-and-power sub-team has now moved on to researching battery options. The total power that will be required must be determined through testing, and could possibly vary with hardware adjustments.

Currently nickel-metal hydride cells are being considered. A comparison of several options is shown in Appendix B-1. The most favorable qualities of the nickel-metal hydride cell are its small volume, life cycle, voltage discharge curve, and Amp Hours per weight (about twice than a lead acid battery). The nickel-metal cell cost about twice that of the currently used lead acid option, but the extra cost is well worth it for the intended use.

### **Motor Selection**

The ASV has been equipped with Watersnake A-18 trolling motors. (Compare to the 2011 selection of the Minn Kota Endura 30.) The new motors have a weight of 5.25 pounds and a thrust of 18 pounds per motor with the original propellers. The previous motors had a weight of 15 pounds and a thrust of 30 pounds per motor. The total weight savings from 2011 is 19.5 pounds and thrust was reduced by 24 pounds. The current motor runs on a 12-volt system so no modification needed to be made to the power configuration.

### **Propellers and Thrust**

The stock propellers are designed to provide maximum thrust in order to propel a somewhat larger vessel than the ASV. One of the main design features is a cupped blade which provides greater thrust in the forward direction. In reverse, the props provide variable results. It is important that motors have consistent thrust in reverse to maneuver the ASV.

A second issue, commonly referred to as “prop walk,” which is a term used to describe the tendency veer off course when moving forwards or backwards. A right-handed propeller (which rotates clockwise) will tend to push the stern of the boat to starboard. When in reverse (rotating counter-clockwise), the stern of the boat will drift to port. A way that many boats with two or more props address this issue is by using a counter-rotating propeller. The counter rotating prop is identical to the standard prop, except for an opposite pitch of each blade. This means that the motors are spinning in opposite directions. The starboard prop spins clockwise while the port prop spins counter-clockwise. For example, if the boat is attempting to turn to port, the operator will apply significantly greater thrust to the starboard motor. Since the starboard motor is spinning clockwise, the stern of the boat will drift to the right to aid turning.

The ASV is using two custom designed props. The original props have been replaced by a symmetrical design to provide similar results when operating in forward and reverse. An important part of prop selection is matching the size and pitch of the prop blades with the weight of the boat and power of the motors. The new props were created in four stages: Research, Prototyping, Design, and Manufacturing. The ASV will most benefit from a high thrust prop. Some existing applications of props designed to generate high thrust and maintain consistent speed include bow thrusters, tug boats, and sport ski boats. (See Figure 9 in Appendix E)

The lateral thruster is a ducted propeller similar to the ASV prop with protective shroud. Tug boat props are solely designed to generate maximum thrust to push larger vessels. Ski boats must operate with high thrust at precise speeds to provide the best watersport experience.

The prototyping stage involved modeling and simulation using OpenProp software as well as stress analysis using SolidWorks. OpenProp is free software for designing optimized marine propellers or axial-flow turbines. This software is coded as a suite of m-files running in MATLAB. By integrating the lift and drag forces across each of the prop blades, OpenProp can make a theoretical calculation for the performance of any prop. The program seemed to struggle when designing a small prop for our ASV so larger scale models of proposed designs were compared and tested. (See Figure 10 in Appendix E)

After a design was verified, it was created in SolidWorks. The hub was drawn to the exact dimensions as the original. This would ensure that it would be a direct fit and could be used interchangeably with a variety of trolling motors. The complex shapes were able to be analyzed using the Simulation Xpress Analysis Wizard included with SolidWorks. (See Figure 11 in Appendix E)

The final step in creation of the ASV props is manufacturing. Creating a high quality custom prop is usually an extremely difficult task. This propeller was able to be made using the Dimension Elite 3D Printer. The Dimension Elite is ideal for printing intricate 3D product mockups as well as functional models. A .STL file was able to be saved directly from SolidWorks and loaded into Old Dominion University's 3-D printer.

### **Propeller Shrouds**

The 2012 ODU ASV Team has chosen to use propeller propulsion. The competition rules explain that moving parts associated with the propulsion must have a shroud. Shrouds also increase the directional thrust. The shrouds used for the 2011 ODU ASV were made from PVC tubing. They had a diameter of 12 inches and a depth of 10 inches and were connected by utilizing the motor shaft and skeg. It was apparent that the original shrouds were oversized which caused several

problems. The first problem was the additional weight. The oversized dimensions and material used increased each shroud's weight to 3 lbs. The second problem was the hydrodynamic forces that were generating from the large size. This caused there to be a dramatic reduction in speed. The third problem was that the shrouds were not increasing thrust. The diameter was too large with respect to the propeller to give any noticeable thrust effects. The final problem associated with the original shrouds was that they are permanently connected to the motor shaft.

The new shrouds design had to be smaller, lightweight, and removable. The current shrouds were first designed using Autodesk Inventor. The shroud diameter was reduced to 6.125 inches. The depth of the shroud was reduced to 2 inches. The shroud was then designed to connect to the motor shaft and skeg non-permanently. This was achieved by using clamping force around the shaft and skeg. The shroud was then manufactured by using a 3-D printer. The printers use lightweight but strong plastic material. These improvements reduced the shrouds weight to much less than one pound each. (See Figure 12 in Appendix E)

### **Battery Selection**

The ASV battery is a custom pack of ten Nickel-Metal Hydride (NiMH) cells. The pack has an operating range from 10-14 Volts and is rated for up to 9.5 amp hours. There are many considerations when selecting a battery. A comparison chart of battery types can be found in Appendix C. Load current, energy density, cost, and reliability were the major factors. Through testing it was determined the ASV system has a peak current draw of 30 amps. The energy density of each battery material determines how heavy a battery will need to be for the desired energy capacity. As the energy density increases, so does the cost. A sealed lead acid battery has the lowest cost, but greatest mass. A Lithium Polymer has the highest cost and highest energy density, but can become dangerous when misused affecting safety and reliability.

The voltage discharge curve of different batteries will vary as well. This factor is determined by the chemical properties of the battery. Certain batteries such as lead acid exhibit a sloped curve. In this type of battery, the voltage gradually degrades as the battery is used. Batteries exhibiting flat voltage curves such as NiMH and Lithium Polymer maintain their voltage at a somewhat constant level until the end of their life, where the voltage suddenly drops off. Sealed Lead Acid and NiMH were the two cell types compared during testing. Tests were conducted using both battery types under similar conditions and monitored by a voltage and current sensor. Test results can be found in Appendix D.

### **MECHATRONICS**

The ASV project requires a combination of mechanical, electrical, computer, software, control, and systems design engineering. Mechatronics encourages the development of simple,

economical, and reliable systems. Each of the individual components used are chosen for more than individual performance. An analysis is done in order to select the component that will fulfill a requirement a work with the system. Major considerations include: compatibility, cost, weight, power consumption, ease of use, and durability.

A single system has been developed. The ASV will channel sensory information to the motors through a central computer.

The ASV relies on a combination of vision processing and LiDAR (Light Detection and Ranging) mapping to sense the surrounding environment. Two cameras cover a 140 degree field of view. The LiDAR serves as a method to accurately measuring distance by creating a 3-D view of its surroundings. The ArduPilot Mega (APM) will bring in additional information such as heading, GPS location, speed, and acceleration. The APM is a microcontroller board used to link multiple sensors. It will also control the gimbal used to hold the cameras and LiDAR.

Sensory information is collected on the central computer. The computer operates using a standard motherboard and components. The Windows 7 operating system is installed in order to keep the system user friendly and compatible with other components. The central programming for the ASV is written in C++ coding language and executed using Visual Studio 2010 Express software. An open source coding library, OpenCV is extensively used for the image processing. The ASV program includes the steps needed to communicate with sensory input, process data, and output appropriate motor commands.

The Arduino Mega is the base system of the Ardupilot Mega. This microcontroller will serve as the link between autonomous and manual operation. The Arduino Mega receives information from the computer and the remote control receiver. A second task handled by the Arduino Mega is generation of a Pulse Width Modulation (PWM) signal. A PWM signal is required by the speed controllers which are the last step before reaching the electric motors.

## **SYSTEM COMPONENTS**

### **Microcontrollers**

Microcontrollers are a necessary component to the ASV's operation. These miniature computers can easily handle a variety of inputs and outputs while running on a continuous programming loop. Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments. This project uses three Arduino Mega boards to handle sensory input, communication between the user and ASV, and power management. These boards will be referred to as State Arduino, Drive Arduino, and Power Arduino. The Arduino allows the user to read/write analog, digital, and PWM signals.

#### *State Arduino:*

The State Arduino collects data from a 9 DOF sensor stick. This stick is a small sensor board with 9 degrees of freedom. It includes the ADXL345 accelerometer, the HMC5883L magnetometer, and the ITG-3200 gyro. This information is used to drive the gimbal used to level our cameras and LiDAR. The State Arduino also collects GPS information. The data is also passed to the central computer for use in the navigation code's PID controller.

#### *Drive Arduino:*

The Drive Arduino is the final step before motor commands are sent to the speed controllers. It is at this controller that the decision between user or computer control is made. The PWM signal from the RC Receiver is read and scaled for use. The one channel of the six channel receiver is used as an on/off switch to set control mode. Two other channels are used to control the left and right motor. All of these values must be carefully scaled to work with variable declarations in coding and system components. An added feature of the Drive Arduino is the ability to limit motor output when running low on power. This enables all of the system components to continue operating until the ASV can be safely returned to shore. (A section of the Drive Arduino program can be found in Appendix F-1.)

#### *Power Arduino:*

The Power Arduino is used to monitor the system voltage and current usage. This controller will monitor vital information for safe and reliable operation of the system. The Power Arduino first reads the outputted motor commands and user mode from the Drive Arduino. The next step is to read the analog input from a voltage and current sensor to verify safe operating conditions. If necessary, the Power Arduino will send a signal to the Drive Arduino to limit power. The user is also given various warnings by way of a loud piezo buzzer. (A section of the Power Arduino program can be found in Appendix F-2.)

### **Onboard Computer**

In order for the ASV to successfully navigate a course without human interaction all of the inputs from the sensors and cameras must be collected and interpreted into useable data. This data must then be processed to make the required navigational decision. These calculations require far more processing than can be provided by the Arduino controllers alone. The vision and navigation codes require the use of a powerful computer to be able to process the data and make navigation decisions in real time.

In the past this processing has been handled by a laptop. However, this solution presented several problems of its own. The primary problem with the laptop was a lack of power. The



processor was not fast enough for the code to keep up with the movements of the boat and this caused the boat to stop responding to its surroundings, become stuck in a navigational loop or become lost on the course. The second most prominent problem the laptop presented was that when sealed in the waterproof electronics box on the boat, it would overheat after prolonged testing. This forced it to shut down completely and become useless until it had been recovered and allowed to cool. An additional problem related to the laptop was that the screen, keyboard, and other integrated peripherals added to the overall weight of the boat which is a critical judgment point in the competition.

In order to solve these problems the laptop has been replaced by an entirely new, custom built computer made of more powerful and reliable desktop and automotive computer components. The new computer is much faster in single threaded applications and has four available threads verses the laptop's two threads. The new computer also has vast improvements to its cooling systems; with the addition of air snorkels to the electronics cabinet and the option to water cool the processor for longer testing runs. The new computer also can be able to be remotely controlled from land, reducing the weight from the monitor, mouse, and keyboard.

### **Speed Controllers**

The ASV motors are powered by two Victor 883 speed controllers specifically engineered for robotic applications. The high current capacity, low voltage drop, and peak surge capacity make it ideal for drive systems. This controller safely handles the high continuous current draws and extreme current surges produced by the ASV trolling motors. The low voltage drop and high switching speed ensures the motor receives maximum power.

## **VISION PROCESSING**

### **Object Detection**

A function was written into the program that captures images from the camera. The images are first combined before being searched for objects. Combining the images first allows for fewer errors when an object appears in both cameras' view.

Color segmentation occurs after the images are combined where every pixel is given an RGB (red, green, blue) value. The RGB values are then converted to hue saturation values (HSV) using the `cvtColor` function in the OpenCV library. Color ranges are matched to buoy colors either using manual or self-adjusting calibration (to be discussed in section 6.3). The ranges are multidimensional and are defined using the `threshold` function in OpenCV. The pixels that fall into the ranges set for red, green, yellow, and blue are then enhanced using the `dilation` and `erosion` functions in the OpenCV library. The `contour` function (in the OpenCV library) is then used for each threshold color image. The `contour` function will give the size and position of all

objects found in the color ranges. The heading is calculated from the position and the distance is calculated from the size. The size of the buoys must be known for this calculation.

## **Color Calibration**

### *Manual Calibration:*

Color thresholds can be manually input upon startup of the program using a series of sliders generated by the `cvCreateTrackbar` function in OpenCV. Each slider is linked to one of the twelve threshold values. The colors detected can be monitored and compared to the image captured using multiple windows shown in Figure 14 in Appendix E.

### *Auto-Adjusting Calibration:*

Auto-adjusting calibration is needed to compensate for various lighting conditions. Lighting can change during a single run simply due to the orientation of the ASV in reference to the sun. A function has been written to allow the threshold values to self-adjust based on the brightness.

After the images are captured and combined the single image is evaluated to determine a brightness value using the `YCbCr` function in the OpenCV for nine equal regions. Nine regions were chosen as an optimal balance between speed and accuracy. The brightness value is then used to update the threshold values.

## **SENSOR DATA**

### **Cameras**

Cameras are the ASV's primary sensor. The cameras currently used are two identical 1.3 Megapixel USB webcams (see Figure 15 in Appendix E). Key features of this camera include 30 frames per second refresh rate, automatic white balancing, and automatic exposure control. These cameras deliver an acceptable image quality in a variety of lighting conditions.

To optimize image processing performance, the ASV needs to capture the greatest amount of information in the fewest number of pixels. As processing speed increases, higher resolution images can be processed.

### **LiDAR**

The LIDAR in use is a URG-04LX-UG01, made by HOKUYO (see Figure 16 in Appendix E). This scanning laser rangefinder is a small, affordable, and accurate laser scanner that is perfect for robotic applications. It is able to report ranges from 20mm to 5600mm (1mm resolution) in a 240° arc (0.36° angular resolution). Its power consumption, 5V 500ma, allows it to be used on a battery operated platform. The laser and sensor in the LIDAR are mounted to the top of a

single-axis gimbal that keeps the sweep parallel with the surface. When the command to read in data is received the LIDAR waits for the laser to get to the starting point it then reads in data until it reaches the end point. The data is then sent to the computer where it is stored and processed by the main program.

The data is initially read in as an array, with the index being a representation of the angle and the value stored; the distance returned in millimeters. The first step performed is a conversion of the index to an equivalent radian value. The data index ranges from 0 to 726 and the sweep is  $-120^{\circ}$  to  $120^{\circ}$ , a simple conversion equation is used. The next step is filtering the data to locate the buoys. To do this the data is read in one point at a time, if the point is within a certain distance of the previous point, which is dependent on the overall distance from the LIDAR, it is considered to be part of the same object. This step is capable of picking out as many buoys as can be seen by the LIDAR, and creates a new vector that contains the distance of the starting point, the starting point angle, the distance of the ending point, and the ending point angle. The current process requires several passes of the data set to filter the buoys, but by implementing more efficient algorithms it is hoped to be accomplished in one pass.

### **State Information**

State information gives the data about where the sensor is and what the sensor is doing. The position information is returned as latitude and longitude. The orientation information is returned as the yaw, pitch and roll. The rotational information returned is the yaw rate, pitch rate and roll rate. This information allows more precise control of the ASV through the course.

Two sensor units were required to obtain all of the state information; a global position sensor (GPS) and a nine degree of freedoms sensor unit (9DOF).

The “66 Channel LS20031 GPS 5Hz Receiver” was used. This unit costs \$59.95 and is very small and lightweight. Two GPS units were used to build the Arduino Mega State Sensor. This was done to increase accuracy and precision by averaging the latitudes and longitudes. This unit has an accuracy of two meters which was not sufficient enough to navigate but can be used for estimation. (See Figure 17 in Appendix E)

The 9DOF sensor unit has three sensors on board: a three-axis accelerometer, a three-axis gyroscope, and a three-axis magnetometer. These three sensors used together give yaw, pitch, roll, yaw rate, roll rate and pitch rate previously stated. The unit costs \$99.95 and is also small and lightweight. (See Figure 18 in Appendix E)

A protoboard was used to attach the sensors to the top of the raw Arduino Mega board giving an elevated platform to place sensors. The board costs \$17.95. (See Appendix E, Figures 19, 20)

Programming of the sensor unit was more challenging and time consuming than the physical construction. 707 lines of code run the Arduino Mega State Sensor unit, which includes initialization, calibration procedures, and various other outputs to show the accuracy of the sensors information.

### **Power Monitoring and Management**

A voltage and current sensor is needed to measure power usage. The ASV uses a compact DC voltage and current sense PCB developed and assembled by AttoPilot International. DC current is determined by measuring voltage drop across shunt resistors then converted to analog voltage output by the Texas Instruments INA-169. Voltage sense is accomplished by scaling to 3.3V ADC range by a precision resistor divider. The current limit coincides with maximum power rating of the shunt resistors. This particular board is designed to operate up to 13.6V and 45A.

### **NAVIGATION LOGIC**

After sensory input is collected and processed, the navigation logic decides how the ASV reacts. For best results, this direct response system must run at a high refresh rate. This is because the ASV is essentially operating on past information. A major advantage to this system is the ability to move with great accuracy. For the purpose of this competition the boat is required to drive between channel buoys and avoid obstacles. Each object detected by the sensory input will be assigned three characteristics, color, position, and distance. Based on these characteristics, the navigation logic can determine an appropriate course of action.

### **Course Strategy**

There are an infinite number of strategies or techniques for navigating through the channel. Each of these strategies requires the development of a logic loop which is the section of code that continuously analyzes the list of objects in view. The logic loop should be as simple as possible while achieving the highest level of success. The loop starts by finding any objects in view which can be used to navigate. If no objects exist, a program runs to search for navigation objects. When objects are found the loop continues with the navigation logic. Once a single navigation object has been detected, the ASV sets a target relative to that object and takes the appropriate action. The same procedure is used when multiple navigation objects are detected.

### **Setting Target Point**

The ASV will set a new target point with each code cycle. This is the step where the navigation code analyzes the list of known buoys and decides where to go. The first thing the navigation code looks for is if there are any buoys in the object list. If there are not any objects in view, the

ASV will run a predefined search command. If any navigational objects are in the object list, the navigation program will continue. See Appendix D for the Channel Navigation Flow Chart.

#### Part 1 – List Analysis

The list of objects in view may be short or very long depending on the amount of buoys as well as “noise” that the vision program thinks are buoys. The job of this part of the program is the sort through the list and select the nearest appropriate red, green, or other navigational object. (For code see Appendix F-3)

#### Part 2 – Buoy Scenarios

After the object list has been analyzed and a red and/or green buoy have been selected, the ASV will either target a single red, single green, or pair of red and green buoys. A small adjustment is made for each of the three scenarios so the buoys will pass on the correct side of the boat. (For code see Appendix F-4)

#### Part 3 – Special Features

Some additional functions have been added for channel navigation. If there are multiple same color buoys in view, the program was written to target the nearest buoy. A special variable has been added so that the ASV can pass from one set of buoys to the next. Once the buoy is within a set range, the program will ignore that object and head for the next. Another variable is used to adjust the ASV’s desire to drive at a perpendicular angle to a set of gates. When the ASV has decided to target the center of a set of buoys, the target point will be adjusted to aim closer to the buoy at greater distance. This adjustment will make it easier for the ASV to remain on course.

#### **PID Controller (proportional only)**

Each time a target point is set, the “drive” portion of the navigation program will decide how much thrust to add to each motor. The target point consists of a single target theta (angle from straight ahead) and target distance. The motor output is set in two steps. In step one, the differential between left and right thrust is set. As the target moves away from  $\theta = 0$ , the difference in thrust between the left and right motors is increased. In step two, the left and right thrust is simultaneously adjusted according to the target distance. (See Appendix F-5 for code and graphs.)

**APPENDIX A: Cost Summary**

|                              |         |
|------------------------------|---------|
| <b>Beginning of Semester</b> | \$4,000 |
| <b>Remaining</b>             | \$1,820 |

*Purchases:*

|                          |                |
|--------------------------|----------------|
| <b>Pontoons</b>          | \$400          |
| <b>T-Shirts</b>          | \$250          |
| <b>Competition Fee</b>   | \$500          |
| <b>Hardware</b>          | \$30           |
| <b>Motors</b>            | \$250          |
| <b>Computer Parts</b>    | \$500          |
| <b>Testing Equipment</b> | \$250          |
| <b>TOTAL</b>             | <b>\$2,180</b> |

*\*\*Approximations*

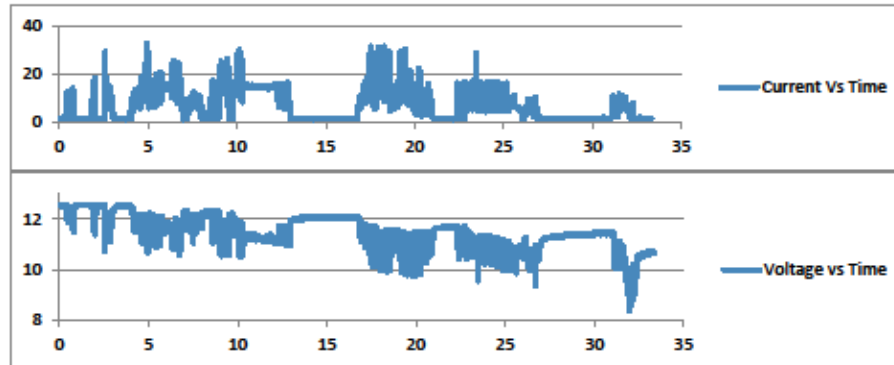
APPENDIX B: Battery Option Comparison

|  | NiCd                               | NiMH                               | Lead Acid                     | Li-ion                               | Li-ion polymer                       | Reusable Alkaline                   |
|--|------------------------------------|------------------------------------|-------------------------------|--------------------------------------|--------------------------------------|-------------------------------------|
| <b>Gravimetric Energy Density</b> (Wh/kg)                          | 45-80                              | 60-120                             | 30-50                         | 110-160                              | 100-130                              | 80 (initial)                        |
| <b>Internal Resistance</b><br>(includes peripheral circuits) in mW | 100 to 200 <sup>1</sup><br>6V pack | 200 to 300 <sup>1</sup><br>6V pack | <100 <sup>1</sup><br>12V pack | 150 to 250 <sup>1</sup><br>7.2V pack | 200 to 300 <sup>1</sup><br>7.2V pack | 200 to 2000 <sup>1</sup><br>6V pack |
| <b>Cycle Life</b> (to 80% of initial capacity)                     | 1500 <sup>2</sup>                  | 300 to 500 <sup>2,3</sup>          | 200 to 300 <sup>2</sup>       | 500 to 1000 <sup>3</sup>             | 300 to 500                           | 50 <sup>3</sup><br>(to 50%)         |
| <b>Fast Charge Time</b>  | 1h typical                         | 2-4h                               | 8-16h                         | 2-4h                                 | 2-4h                                 | 2-3h                                |
| <b>Overcharge Tolerance</b>  | moderate                           | low                                | high                          | very low                             | low                                  | moderate                            |
| <b>Self-discharge / Month</b><br>(room temperature)                | 20% <sup>4</sup>                   | 30% <sup>4</sup>                   | 5%                            | 10% <sup>5</sup>                     | ~10% <sup>5</sup>                    | 0.3%                                |
| <b>Cell Voltage</b> (nominal)                                      | 1.25V <sup>6</sup>                 | 1.25V <sup>6</sup>                 | 2V                            | 3.6V                                 | 3.6V                                 | 1.5V                                |
| <b>Load Current</b><br>- peak<br>- best result                     | 20C<br>1C                          | 5C<br>0.5C or lower                | 5C <sup>7</sup><br>0.2C       | >2C<br>1C or lower                   | >2C<br>1C or lower                   | 0.5C<br>0.2C or lower               |
| <b>Operating Temperature</b><br>(discharge only)                   | -40 to 60°C                        | -20 to 60°C                        | -20 to 60°C                   | -20 to 60°C                          | 0 to 60°C                            | 0 to 65°C                           |
| <b>Maintenance Requirement</b>                                     | 30 to 60 days                      | 60 to 90 days                      | 3 to 6 months <sup>9</sup>    | not req.                             | not req.                             | not req.                            |
| <b>Typical Battery Cost</b><br>(US\$, reference only)              | \$50<br>(7.2V)                     | \$60<br>(7.2V)                     | \$25<br>(6V)                  | \$100<br>(7.2V)                      | \$100<br>(7.2V)                      | \$5<br>(9V)                         |
| <b>Cost per Cycle</b> (US\$) <sup>11</sup>                         | \$0.04                             | \$0.12                             | \$0.10                        | \$0.14                               | \$0.29                               | \$0.10-0.50                         |
| <b>Commercial use since</b>  | 1950                               | 1990                               | 1970                          | 1991                                 | 1999                                 | 1992                                |

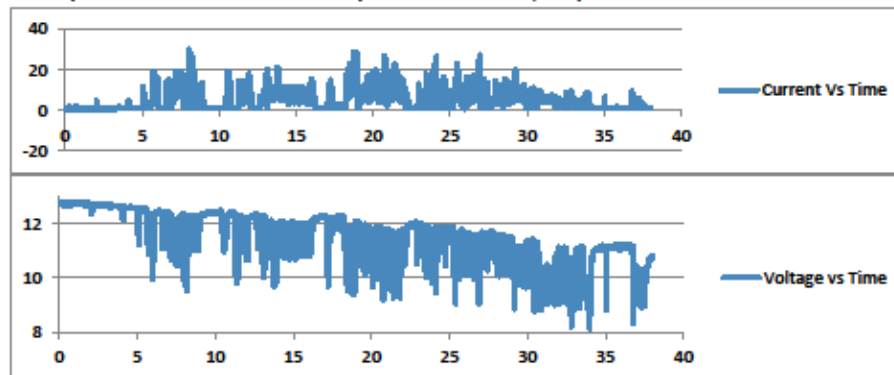
APPENDIX C: Battery Comparison Data

April 7, 2012 Battery Test

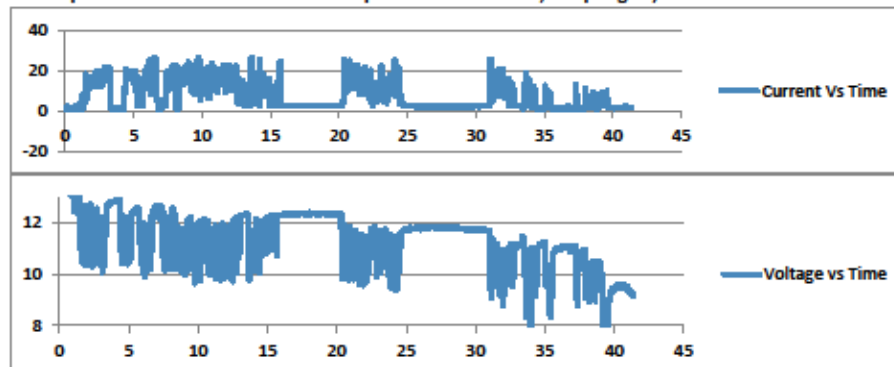
7.5 Amp Hour – Lead Acid – Used 3.8 Amp Hours - End: 12.1V, temp ambient



9.5 Amp Hour – NiMH – 1 – Used 3.3 Amp Hours - End: 12.1V, temp ambient

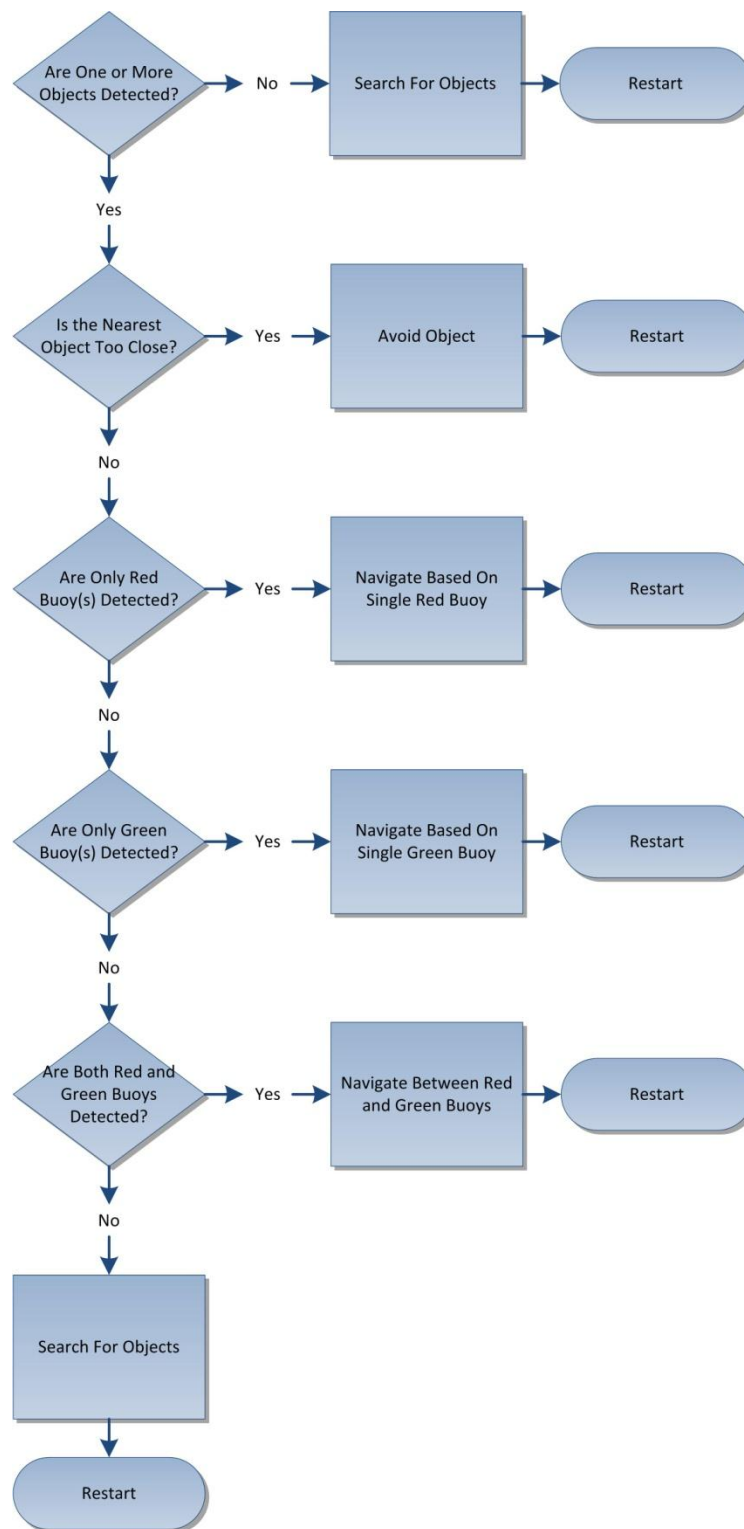


9.5 Amp Hour - NiMH – 2 – Used 5.18 Amp Hours - End: 11.7V, temp slightly above ambient





APPENDIX D: Channel Navigation Flowchart



**APPENDIX E: Reference Figures**



**Figure 1: Stabilizer float intended use**



**Figure 2**



**Figure 3**

APPENDIX E: Reference Figures



Figure 4: Perforated Decking (1)

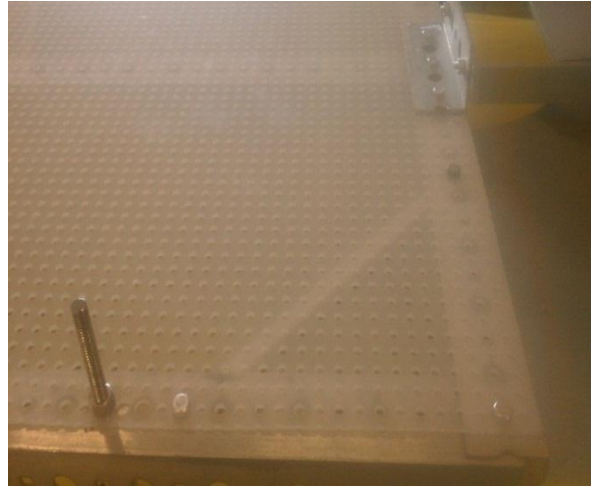


Figure 5: Decking (2)



Figure 6: Static testing with dummy weights and temporary decking



Figures 7 and 8: Autonomous testing.

APPENDIX E: Reference Figures



Figure 9

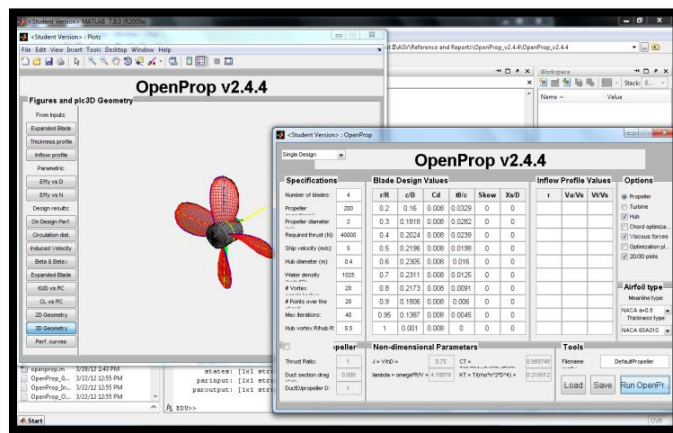


Figure 10: OpenProp user interface

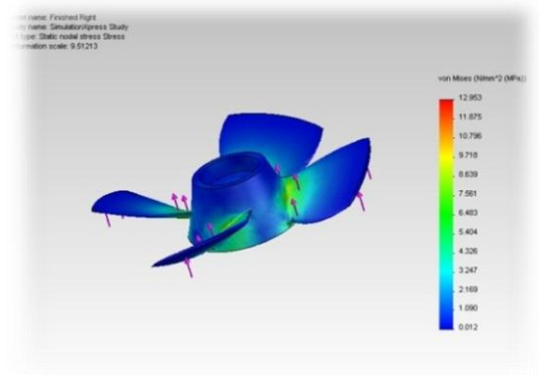


Figure 11: SolidWorks design used by 3-D printer

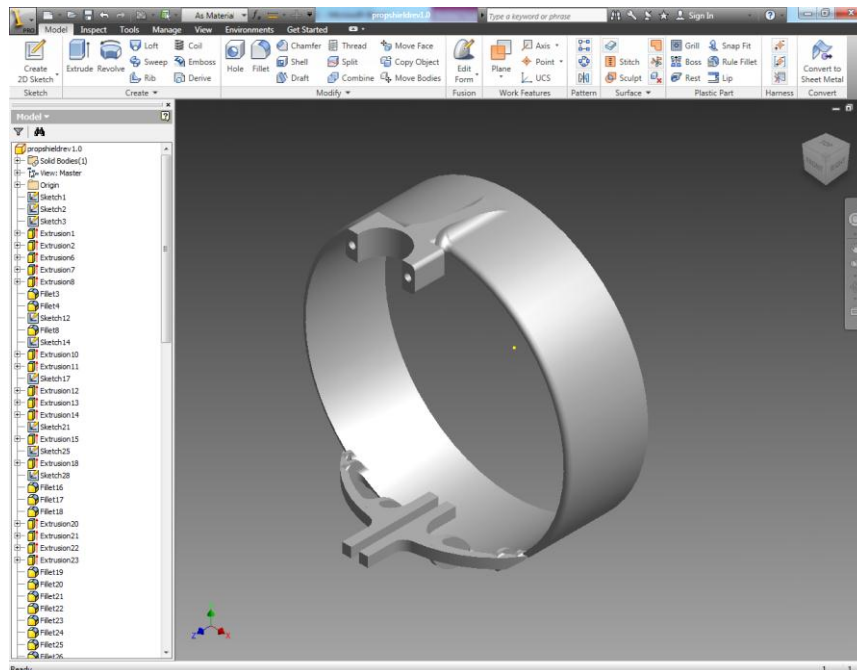


Figure 12



# APPENDIX E: Reference Figures

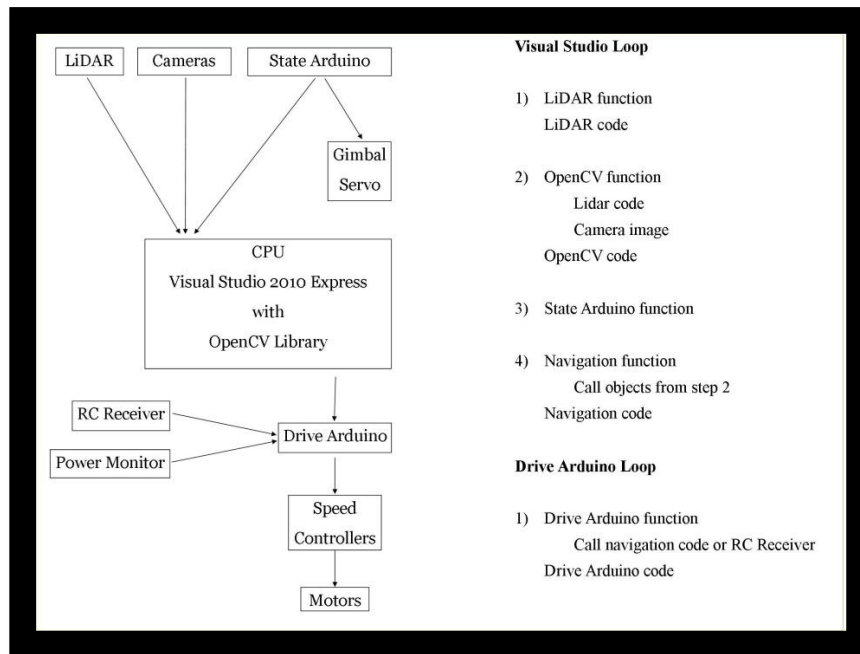


Figure 13: System Architecture and Hierarchy of Functions

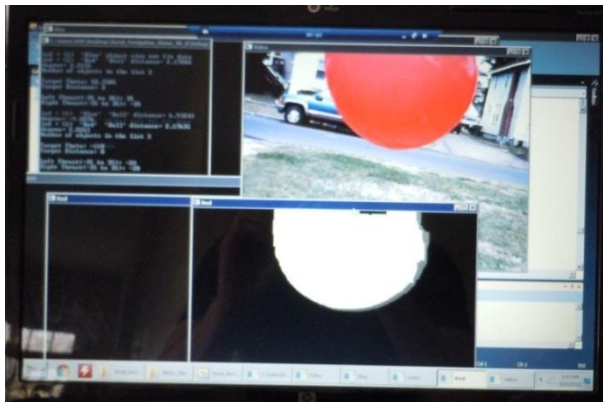


Figure 14: Red Threshold being Monitored



Figure 15: 1.3 Megapixel USB Webcams

## APPENDIX E: Reference Figures



Figure 16: URG-04LX-UG01  
LIDAR made by HOKUYO

Figure 17: 66  
Channel LS20031  
GPS 5Hz Receiver

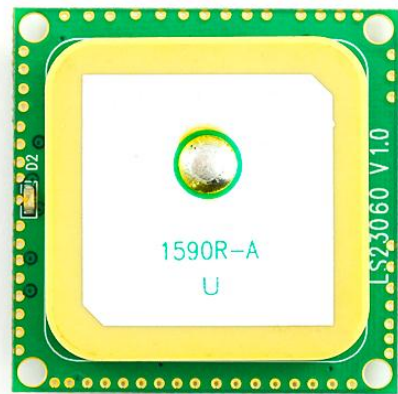


Figure 18: 9 DOF Sensor

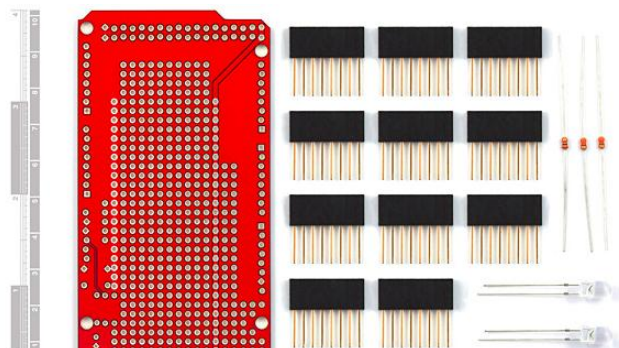
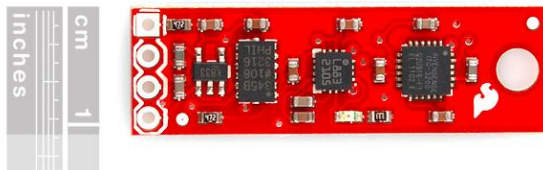
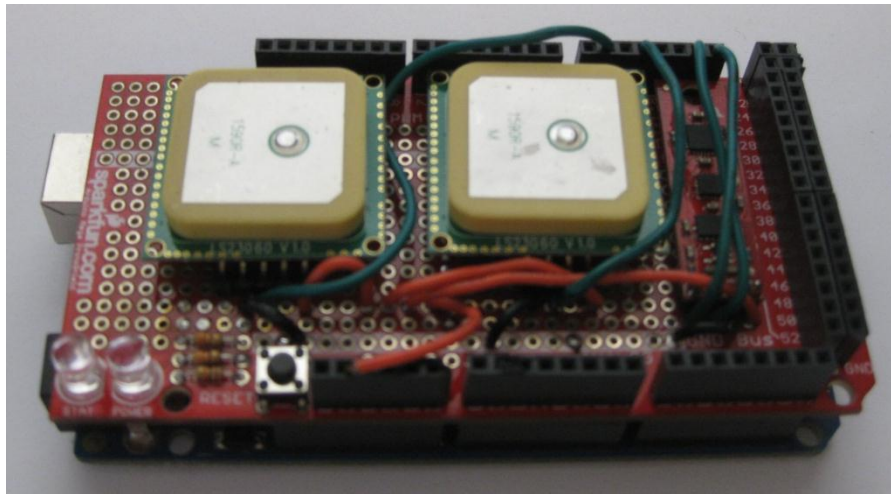


Figure 19: Protoboard

**APPENDIX E: Reference Figures**



**Figure 20: Fully Assembled Arduino State Sensor**

## APPENDIX F-1: Drive Arduino Code

```
void loop() {
  rawUser = pulseIn(userControlPin, HIGH);    //Reads PWM signal
  user = map(rawUser,1000,2000,10,-10);        //converts User Switch PWM data to single integer
  rawLeft = pulseIn(leftJoyStickPin, HIGH);    //Reads PWM signal
  rawRight = pulseIn(rightJoyStickPin, HIGH);  //Reads PWM signal

  //select source for motor commands
  if ((user > 0) && (user < 20)){                //manual mode
    leftMotorSpeed = map(rawLeft,1000,2000,140,50); //converts Left Stick PWM data to single integer
    rightMotorSpeed = map(rawRight,1000,2000,140,50); //converts Right Stick PWM data to single integer
  }
  else if ((user > -20) && (user < 0)){           //autonomous mode
    while(Serial.available()>0){
      temp1=Serial.read();                      //read left motor command from computer
      temp2=Serial.read();                      //read right motor command from computer
      leftMotorSpeed = (temp1 - 35);             //returns temp to navigation integer
      rightMotorSpeed = (temp2 - 35);            //returns temp to navigation integer
      leftMotorSpeed = (leftMotorSpeed + 95);    //convert navigation output to speed controller range
      rightMotorSpeed = (rightMotorSpeed + 95);
    }
  }
  else {
    leftMotorSpeed = 95;                        //neutral
    rightMotorSpeed = 95;                      //neutral
  }

  //Power Management - from power arduino
  if (digitalRead(reducePowerPin) == HIGH){
    leftMotorSpeed = (leftMotorSpeed - 95) * 0.5 + 95; //reduce to half power
    rightMotorSpeed = (rightMotorSpeed - 95) * 0.5 + 95; //reduce to half power
  }

  //Pass drive info to "power arduino" for monitoring
  userOut.write(rawUser);                      //uses servo command to generate pwm output
  leftOut.write(leftMotorSpeed);
  rightOut.write(rightMotorSpeed);

  //Output motor values to speed controller (50 - 140)
  leftMotor.write(leftMotorSpeed);             //uses servo command to generate pwm output for speed controllers
  rightMotor.write(rightMotorSpeed);
}
```



## APPENDIX F-2: Power Arduino Code

```
void loop() {

  //Drive Arduino Information
  int rawUser = pulseIn(userPin, HIGH);           //Reads PWM signal
  int rawLeft = pulseIn(leftMotorSpeedPin, HIGH); //Reads PWM signal
  int rawRight = pulseIn(rightMotorSpeedPin, HIGH); //Reads PWM signal
  int User = map(rawUser,1000,2000,10,-10);       //converts User Switch PWM data to single integer
  int Left = map(rawLeft,1000,2000,-100,100);      //converts User Switch PWM data to single integer
  int Right = map(rawRight,1000,2000,-100,100);    //converts User Switch PWM data to single integer

  //Power Monitoring
  VRaw = analogRead(voltagePin);                   //read raw voltage
  IRaw = analogRead(currentPin);                   //read raw current
  V = VRaw/49.44;                                   //45 Amp board calibration
  I = IRaw/14.9;                                    //45 Amp board calibration
  time = millis();                                  //duration program has been running
  I_total = I_total + I;                           //sum all current readings
  AmpAverage = I_total / cycles;                   //find average current
  AmpHours = AmpAverage * time / 3600000;          //divide average current by run time
  cycles = cycles + 1;                             //number of code cycles

  //Power Management
  if (V >= 11){                                     //11V for 6 cell lead acid, 10V for 10 cell NiMH
    digitalWrite(AlarmPin, LOW);                   //alarm off
    digitalWrite(reducePowerPin, LOW);             //full power
  }
  else{
    digitalWrite(AlarmPin, HIGH);                  //alarm on
    digitalWrite(reducePowerPin, HIGH);            //reduced power
  }
}
```

### APPENDIX F-3: Setting Target Point – List Analysis Code

```
void Navigation::Navigate(list<Objects> stuff)           //Navigation if a list of buoys is passed
{
    ////////////////////////////////////////////////////
    //specialty//////////////////////////////////////////
    too_close = 90;           //back up if object is within this range (cm)
    next_buoy = 150;         //target a farther buoy if previous buoy is within this range (cm)
    square_rating = 0;       //drive straight through gate (0-Least Aggressive to 90-Most Aggressive)
    //specialty//////////////////////////////////////////
    ////////////////////////////////////////////////////

    list<Objects>::iterator bitr;           //Creates an iterator for referencing data in the list
    bitr = stuff.begin();                 //Points the iterator to the beginning of the list
    int size = stuff.size();              //Stores the total size of the list
    Objects *buoys;                      //Creates the pointer buoys
    buoys = new Objects[size];           //Creates an array of buoys that is pointed at by buoys
    for(int i = 0; i < stuff.size(); i++)  //A loop for assigning data from the list to the array
    {
        buoys[i] = *bitr;                //Assigns the data in list position i to buoys position i

        if(size > i)                     //Determines if the iterator can move forward
        {
            bitr++;                      //Moves the iterator to the next piece of data
        }
    }

    if(size == 0)                        //If the list shows no buoys, then search
    {
        target_theta = 0;
        target_distance = 0;
        search();
    }
    else                                //If the list is not empty run through navigation steps
    {
        if(buoys[0].getDistance() < too_close &&
           buoys[0].getTheta() > -30 && buoys[0].getTheta() < 30) //If the first buoy is too close
                                                                    //and within a certain angle,
                                                                    //then go in Reverse
        {
            target_theta = -180;
            target_distance = 0;
            reverse();
        }
        else
        {
            red_distance = 0; red_theta = 0;           //Sets baseline red values
            green_distance = 0; green_theta = 0;       //Sets baseline green values

            for(int i = 0; i < size; i++)              //assign red and green distance/theta to
                                                                    //the appropriate target
            {
                if(buoys[i].getColor() == 'R' && red_distance < next_buoy)
                {
                    red_distance = buoys[i].getDistance();
                    red_theta = buoys[i].getTheta();
                }
                else if(buoys[i].getColor() == 'G' && green_distance < next_buoy)
                {
                    green_distance = buoys[i].getDistance();
                    green_theta = buoys[i].getTheta();
                }
            }

            if(red_distance == 0 && green_distance == 0) //If red or green buoys are not
                                                                    //detected then Search
            {
                search();
            }
        }
    }
}
```

```
        else if(red_distance == 0 && green_distance > 0) //If green is listed and red is
        {                                                                                       not, target visible object
            target_theta = green_theta;
            target_distance = green_distance;
            target_green();
            drive();
        }
        else if(green_distance == 0 && red_distance > 0) //If red is listed and green is
        {                                                                                       not, target visible object
            target_theta = red_theta;
            target_distance = red_distance;
            target_red();
            drive();
        }
        else if(green_distance > 0 && red_distance > 0) //If red and green are listed,
        {                                                                                       target both objects
            target_red_and_green();
            drive();
        }
    }
}
delete [] buoys;
}
```

#### APPENDIX F-4: Setting Target Point – Buoy Scenarios Code

```
void Navigation::target_red() {           //offsets target theta to the right by 90cm
    target_theta = target_theta + (180 / 3.14) * atan(90.0 / target_distance);
}

void Navigation::target_green() {         //offsets target theta to the left by 90cm
    target_theta = target_theta - (180 / 3.14) * atan(90.0 / target_distance);
}

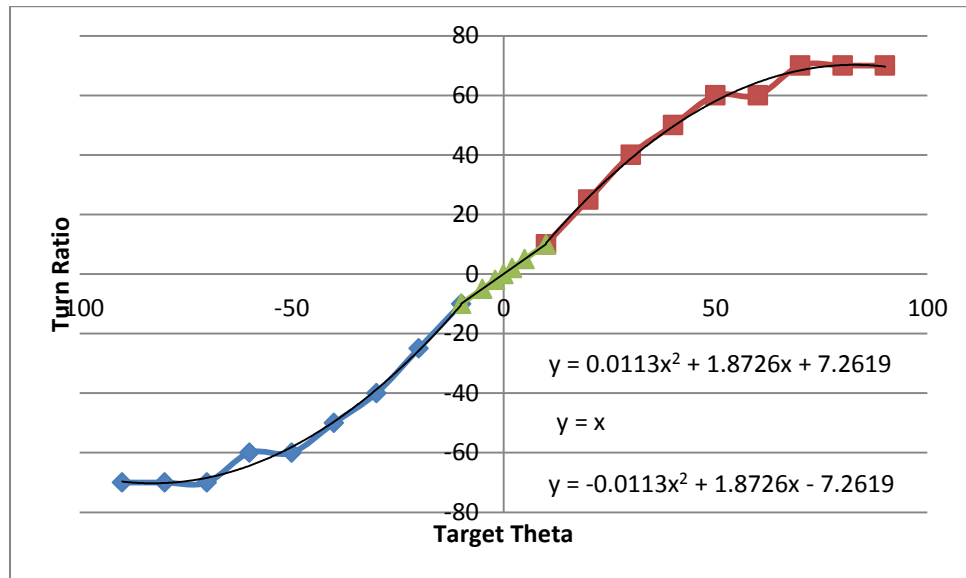
void Navigation::target_red_and_green() { //aims for center of gate by taking the average
    target_theta = (red_theta + green_theta) / 2;
    target_distance = (red_distance + green_distance) / 2;

    percent_skew = (red_distance - green_distance) / //adjusts the target theta in order to bring the
                  (red_distance + green_distance);    boat straight through the gate
    target_theta = target_theta - percent_skew * square_rating;
}
```

## APPENDIX F-5: Proportional Controller Code and Graphs

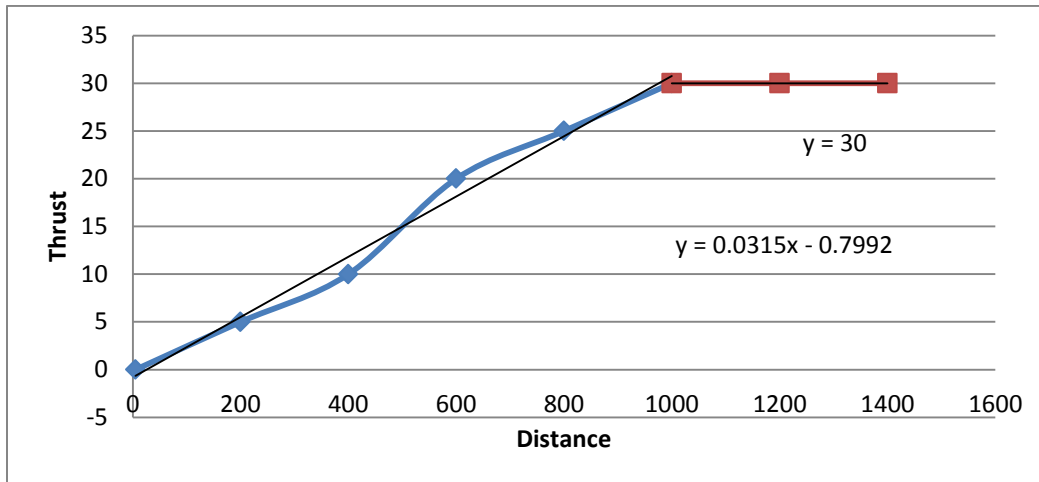
```
void Navigation::drive() {                                //(-35 full reverse; 35 full forward)
    prop_correction = -1;                                //to adjust thrust for non-symmetrical prop (-1-no correction
                                                         //to -10-large correction)
```

\*\*\*Step 1: Control turn speed proportionally to theta\*\*\*



```
double x = target_theta;
if (target_theta < -10)
{
    LeftThrust = (0.0113*x*x + 1.8726*x + 7.2619); //built from excel plot
    RightThrust = LeftThrust * prop_correction;
}
else if ((target_theta >= -10) && (target_theta <=10))
{
    LeftThrust = (x); //built from excel plot
    RightThrust = LeftThrust * prop_correction;
}
else if (target_theta > 10)
{
    LeftThrust = (-0.0113*x*x + 1.8726*x - 7.2619); //built from excel plot
    RightThrust = LeftThrust * prop_correction;
}
```

\*\*\*Step 2: Control forward speed proportionally to distance\*\*\*



```
x = target_distance;
if (target_distance > 1000)    //if target is this far limit addition to 30
{
    LeftThrust = LeftThrust + 30;    //full forward
    RightThrust = RightThrust + 30; //full forward
}
else
{
    LeftThrust = LeftThrust + (0.0315*x - 0.7992); //built from excel plot
    RightThrust = RightThrust + (0.0315*x - 0.7992); //built from excel plot
}
```