Autonomous Man Overboard Rescue Equipment

A Major Qualifying Project Report:
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
By

_____

Frederick Hunter (RBE)

_____

Thomas Hunter (ME and ECE)

Date:_____

Approved:

_____

David Cyganski, Major Advisor

_____

Ken Stafford, Co-Advisor

Acknowledgements

Abstract

This project addressed the problem of rescuing people who have fallen overboard at sea. Large vessels with slow response times and limited turning capabilities are poorly equipped to handle man overboard cases. To solve this problem, a robotic rescue boat was developed with support from systems for the vessel and victim that autonomously help pilot the rescue boat via GPS, a magnetic compass, and a project-specific terminal location system. The same system also supports the return of the victim to the vessel.

Executive Summary

Ever since man has attempted to travel across the ocean there has been a risk of persons falling overboard. With today's new technology, the risk has slowly been diminishing but is still presents a problem for larger ships. With increasing vessel size, if someone were to fall overboard, it would be nearly impossible to keep track of them and turn the boat around for rescue. Some of the larger vessels such as oil tankers may take upwards of ten to twenty minutes to stop and have a turning radius of nearly two kilometers. There are various forms of safety equipment available today such as a beacon one can wear to notify the crew of their position once fallen overboard. Most ships also have a rescue team on standby for such cases, but it takes some time to mobilize any crew and dispatch a rescue mission. In the minutes that pass, the victim may be out of sight. Especially in cold waters every minute counts as survival time is low.

The purpose of this project was to create a robotic rescue system that could autonomously locate and return the victim to the vessel they had fallen from. To achieve this, the equipment was designed in three modules.

The first module is the personal locator device that the victim would be wearing to provide the other modules with his or her location. It features a GPS for general localization, an AM transmitter for homing purposes when the rescue unit approaches the victim, and an XBee transceiver for relaying information to the other modules. The software running on the victim unit executes on an ATMEGA 328P.

The second module is the rescue unit. This is located on a 10 foot rescue craft that is capable of locating and returning the victim autonomously. It also has a GPS for localization, three AM receivers for homing in on the victim, and an XBee for transmitting coordinates and messages to the other modules. The rescue unit also has power electronics for controlling the motors as well as an ATMEGA 328P that runs the software.

The third module is the mothership unit. This is used to relay information about the current status of the rescue to the end-user on the mothership. It features a monitor for displaying the current location of all the modules as well as a GPS for location and an XBee for transmitting information. The software is executed on the Raspberry Pi which also has a keyboard allowing for the user to take over control of the rescue craft should the need arise. These three modules are designed to work together and autonomously find, rescue, and return the victim to the mothership.

Each of the components was tested on the water in Maynard, MA and it was found that it successfully demonstrated the effectiveness of the autonomous rescue mission. The rescue unit was placed at one end of the lake while the victim unit was at the other while the mothership module was on a canoe. The victim unit was then activated and the rescue craft successfully homed in on it to within a meter. The return to home button was then pressed and the rescue unit returned to the mothership unit. It is recommended that a more powerful outboard motor be used for future versions of this project as well as a transmitter with longer range capabilities.

Authorship

Both members of the project worked on mechanical portions, testing of equipment, research, analysis and writing of the report. Thomas was responsible for mechanical-related tasks as well as electrical design and analysis. Frederick wrote all the code for the three modules and integrated the hardware with the software. Each member wrote the respective section pertaining to his aforementioned role in the project.

Table of Contents

Table of Figures

Abbreviations

Abbreviation	Meaning
ADC	Analog to Digital Converter
AHND	Ad-Hoc Network Device
AM	Amplitude Modulation
BJT	Bipolar Junction Transistor
DPDT	Dual Pole Dual Throw
DWT	Dead Weight Tons
FCC	Federal Communications Commission
GPS	Global Positioning System
IP	Ingress Protection Rating (Maxim Incorperated, 2007)
MCC	Master Control Computer
MOB	Man Overboard
MOBI	Man Overboard Indicator
MOD	Man Overboard Detect
Op-Amp	Operational Amplifier
PLD	Personal Locator Device
SPDT	Single Pole Dual Throw
TLD	Terminal Locator Device
USCG	United States Coast Guard
VLF	Very Low Frequency
VRD	Victim Retrieval Detect

Introduction

For centuries, one of the greatest fears of all seafaring people has been the prospect of falling overboard at sea. Thankfully, today there are various technologies that address this issue. Different forms of rescue devices include trailing lines, man overboard detection transmitters, chase boats, rescue buoys and life rings. One of the problems with these devices is that they only work for specific situations and certain sized vessels.

Due to the limitations of the aforementioned man overboard rescue devices, there is a need for a new device or product that is capable of saving the lives of persons operating in this poorly protected niche. Due to the dynamics of larger vessels and their respective handling capabilities, it becomes increasingly difficult to both monitor and protect the crew on board. Because of the vast size of larger vessels, there is a greatly increased chance that a person gone overboard will go unnoticed for several minutes. This is particularly the case for cruise ships and naval vessels.

This project addressed the problem of rescuing people who have fallen overboard at sea. The large amount of time it takes for a vessel to turn around puts a man overboard in great risk. To solve this problem, a robotic rescue boat was developed with support from systems for the vessel and victim that autonomously help pilot the rescue boat via GPS, a magnetic compass, and a project specific terminal location system. The same system also supports the return of the victim to the vessel.

Background

In the year 2001, the United States Coast Guard initiated in 39,486 search and rescue operations (United States Coast Guard, 2012). The 39,486 search and rescue cases are comparable to the four years prior to 2001 where the cases were fluctuating between 36,000 and 42,000 operations. Often, a man overboard situation can be handled easily when a person falls from a smaller vessel. As the size of the ship is increased, the difficulty in retrieving a man overboard is much greater. This poses a large problem, especially for cruise ships, military vessels and tankers or other crafts of this order of magnitude.

A supertanker has a stopping distance of 3 nautical miles when decelerating from a cruising speed of 16 knots using full reverse. For tankers at 17000 dead weight tons (DWT), the stopping distance is 1 nautical mile (Environmental Law Institute, 1991). Additionally, such a vessel has a turning radius of two kilometers. From this information, one can speculate that it becomes wildly impractical to turn the ship around to search for the victim. For this reason, many larger crafts make use of a man overboard boat that is readied by the crew to carry out a search and rescue while the vessel continues along its course or comes to a stop while the search is carried out. One of the problems with this type of setup is that it may take some time for someone to realize that a person has gone overboard. Additionally, it takes a crew several minutes to deploy the craft during which critical time is lost.

Figure 1 – Man Overboard Rescue Boat (Nautic Expo, 12)

The scope of this project was to develop a vessel that automatically and autonomously carries out the search as soon as it detects that a person is overboard. Once the rescue boat has located the victim and he or she secured him or herself, it returns to the mothership.

Problem Statement and Project Goals

This section defines the goals and requirements of the project. The final product should be a vessel that autonomously carries out a search for a man overboard as soon as a distress signal is detected from a personal locator device. When the rescue boat has collected the overboard person, it should return to the mothership, regardless if it has changed its location since the deployment of the rescue boat.

Goals and Objectives

The goal of this project was to design and create a fully functional system that can be integrated in a boat to locate and retrieve a person who has fallen in the water. At this stage of the design, the person will still need to enter the craft under his or her own power.

Tasks

To complete the project in the required timeframe, specific tasks were assigned to the project partner that they were most suited for. The major coding and programming as well as navigation aspects of the project have been researched and implemented by Frederick. Most electrical and mechanical portions of the project have been researched and implemented by Thomas. Both members were familiar with both portions, but the primary effort by each pertained to their major.

Project Requirements

In order for the project to be successful and realizable, certain requirements had to be met. These related to the operating environment of the craft as well as implicit and explicit expectations. For ease of presentation, the requirements are split up into the three main components below. Please note that these requirements were for the final experimental project and do not reflect those of a real full-scale device.

Mothership Module

The Mothership Module was designed to be located on the primary vessel where it is continuously running and checking for any distress signals. It is responsible for displaying what the system is currently doing and updating the rescue boat with its current location. This module is equipped with an XBee ad-Hoc network device capable of transmitting and receiving data to the rescue boat. It is also equipped with a GPS device to determine its current location for transmitting to the rescue boat. It updates its current position every three seconds. The electronics were to be contained in an IP67 container no larger than $0.5m^2$ to prevent the electronics from getting wet. For IP codes, see Appendix L – IP Code Chart.

Personal Locator Device

The personal Locator device was used for determining the location of the victim. It was required to have an IP68 rating and be activated upon a button press to be augmented with a conventional salt water immersion switch in the production version. It had to be capable of sending distress signals and a GPS location using the Ad-Hoc network with a range of up to one kilometer. In later versions, the Ad-Hoc system could be upgraded to cover greater distances. The GPS data was to be acquired from the internal

GPS device. Once the personal locator device was within thirty meters of the rescue boat, it had to activate the Terminal Locator Device to transmit a signal for the rescue boat to follow. The personal locator device had to be small enough to be affixed to a standard life vest without hindering the user. It was to be contained in a 6.8" x 4.5" x 1.8" case.

Rescue Boat

The rescue boat had to be capable of rescuing a victim from a one kilometer distance. The electronic components on the ship had to be contained in a housing rated for IP68. The boat was designed to be fully autonomous, to operate without interaction from humans for location and navigation. The boat was also to be battery powered and driven by two trolling motors, each with a nominal thrust of thirty pounds. It was decided that differential thrust would be used for heading control. The boat had to be capable of travelling one kilometer for testing purposes. In future iterations, a larger motor and boat could be used for traversing larger distances at greater speeds. Once the rescue boat was within thirty meters of the target, it had to activate the terminal locator device circuitry for a more accurate reading on the victim's location due to GPS error. The boat had to be capable of carrying three hundred pounds. This was to accommodate a passenger and the hardware on the boat. The boat also had to be capable of reaching a speed of approximately five knots to overcome expected wind drift conditions during the prototype testing.

Summary

The specifications set make the system durable for use in moderate conditions relating to a level two on the Beaufort scale. These conditions consist of waves up to half a meter in height and winds of up to five and a half kilometers per hour (National Meteorological Library and Archive , 2010). With a larger rescue boat, the system should later be capable of handling more extreme conditions. The above goals and specifications have been set to determine the functionality of such a system. All components of the system such as the personal locator device, the mothership module, and the rescue boat module can be upgraded and installed in a larger boat for increasing the range and performance of the rescue system.

Methods

This chapter provides a background for the robotic autonomous rescue boat. Data from this section was used to perform design and analysis for the respective modules. It begins with an overview of current products on the market that are used to retrieve or collect persons overboard. From the abovementioned project requirements, more solidified specifications were derived for the respective modules. A black box representation of the modules was also determined from the system requirements.

Review of Current MOB Systems and Methods

Current methods of man overboard retrieval may be divided into the two basic categories listed below.

- Rescue from the vessel the person has fallen from, herein referenced as domestic rescue

- Another boat or other craft, herein referred to as foreign rescue

The following sections examine the techniques that fall in the above mentioned categories.

Domestic Rescue Technologies

Current technologies that fall in the category of domestic rescue include but are not limited to the following devices.

- Buoys such as dan buoys or horseshoe buoys
- radio direction finders
- life rings
- trailing lines
- Man overboard setting of a GPS
- Rescue cradles

This section examines the viability and limits of the listed devices as they pertain to the project.

Buoys

Buoys are excellent devices for use in man overboard situations in smaller vessels. When a person is detected as having gone overboard, another crew member deploys the buoy in the direction of where the man overboard is seen in the water. This then provides a visual object for the helmsman to guide the boat toward and to carry out a search.

In the case of this project, such a device would be of limited benefit as the large vessel will continue along the direction of travel for some time until it can effectively loop back to where the man overboard is. Maneuverability of the ship is not effective enough so as to make a sharp turn to loop back towards the lost crew member and the path of travel may be too great where the helmsman will surely lose sight of the marker.

Radio Direction Finders

Radio direction finders are useful in a variety of vessels, though they require all crew members to wear a transmitter. When a person goes overboard, the transmitter, also known as a beacon, sends a distress signal that is picked up by the ship which automatically sounds an alarm. An onboard display system indicates the direction that the person is relative to the boat (Marine Rescue, 2012).

If used in conjunction with larger vessels, radio direction finders would not be effective. Until the boat has turned around and headed toward the person, there will be many precious minutes lost and there is a risk of running the person over with the ship. In most cases, this device would be used in conjunction with a separate man overboard boat to be described in the next section.

Life Rings

Life rings are the most common and well known man overboard rescue devices. When a person falls from the ship, a life ring is thrown toward them in the hope that they may be able to grab onto it to aid in staying afloat while the craft is turned around.

One of the downfalls of this device is that it requires another crew member at the scene who has a fast reaction time. If there is nobody there, then the man overboard will go unnoticed for some time. Additionally, in larger vessels this becomes a problem as there is a greater distance that the device must be thrown (United States Coast Guard, 2012).

Trailing Lines

Trailing lines are nylon ropes that are tied to the stern of the boat and as their name implies, trail the boat. When a person falls overboard, if they react in time, they can swim toward the line and grab hold of it while another crew member stops the vessel or turns around.

This is also a very effective method for saving a person's life, but again, only for smaller vessels. This becomes impractical for larger ships which would require much longer lines. Additionally, assuming the person had caught the trailing line, they would require much endurance and strength to maintain their hold on the line while the ship comes to a stop or someone retrieves them. This is also a very unconventional method for larger vessels (Tor Pinney, 2012).

GPS Man Overboard Function

More applicable to a wide range of boats is the GPS man-overboard function. Some boating GPS-enabled devices contain a button that, when pressed, will mark a location where the man overboard was called.

The GPS device may be used on larger vessels but will not be effective for a timely rescue if that ship will be used to retrieve the person. For this reason, it should be used in conjunction with a smaller rescue craft on the larger vessel (Garmin, 2011).

Rescue Cradles

The rescue cradle is optimal for smaller vessels and is attached to the side of the boat. When the rescue craft is next to the person overboard, the person is rolled into the boat. The downfall of this rescue cradle is that it requires a low hull and persons to operate it. For larger vessels, use of technology like this would be out of the question.

Foreign Rescue Technologies

Some technologies that require the help of an outside influence or vessel are listed below.

Coast Guard Rescue

Used only in the most extreme cases, a coast guard rescue involves a rescue craft deployed and operated by the USCG. Sometimes this may involve a helicopter as well. Use of this technique becomes limited due to the time it takes for a rescue aircraft to reach the victim in the open ocean. There are long range surveillance aircrafts in use by the coast guard such as the HC-130H/J, but they cannot help get the victim out of the water, only to pinpoint their location (United States Coast Guard, 2012).

Man Overboard Boat

Most common on larger scale vessels is the inclusion of a man overboard boat. When the call is sounded for a man overboard, a team assembles and deploys the man overboard boat to commence a search and rescue. Upon retrieval of the person, the boat returns to the mothership.

One of the issues with this setup is that it takes time for a team to assemble and lower the boat into the water to locate the person. Aside from this, the method for rescue is an effective one that can and has saved the lives of many seafaring people. Because of this, the current project is using this method as a basis, but focuses on improving the response time by removing the time needed for a crew to prepare for launching in the rescue boat.

Determination of System Requirements

Based on the capabilities and limitations of current technologies on the market today, the requirements for the project were determined. These took the components of technology that works well and addressed areas that need improvement for large vessel man overboard missions to create a superior method for rescuing sailors.

Module Design from System Requirements

The system requirements provide a concise direction for the project to take. The final project plan was based on these guidelines. As seen in the system requirements, the project was broken down into three main systems termed modules. These may be seen in the Figure 3 below. The modules in question are the Mothership module on the primary vessel, the Personal Locator Device on the victim, and the Rescue Boat Module on the rescue boat.

The Mothership module consists of a GPS device, the Raspberry Pi computer, and an Ad-Hoc network device. The Personal locator device similarly consists of an Ad-Hoc network device, a GPS device, and a man overboard detect button. Finally the Rescue Boat module consists of an Ad-Hoc network device, a GPS, an ATMEGA 328P control module, a victim retrieval detector, 10 foot Sea Rogue inflatable boat and a terminal locator device.
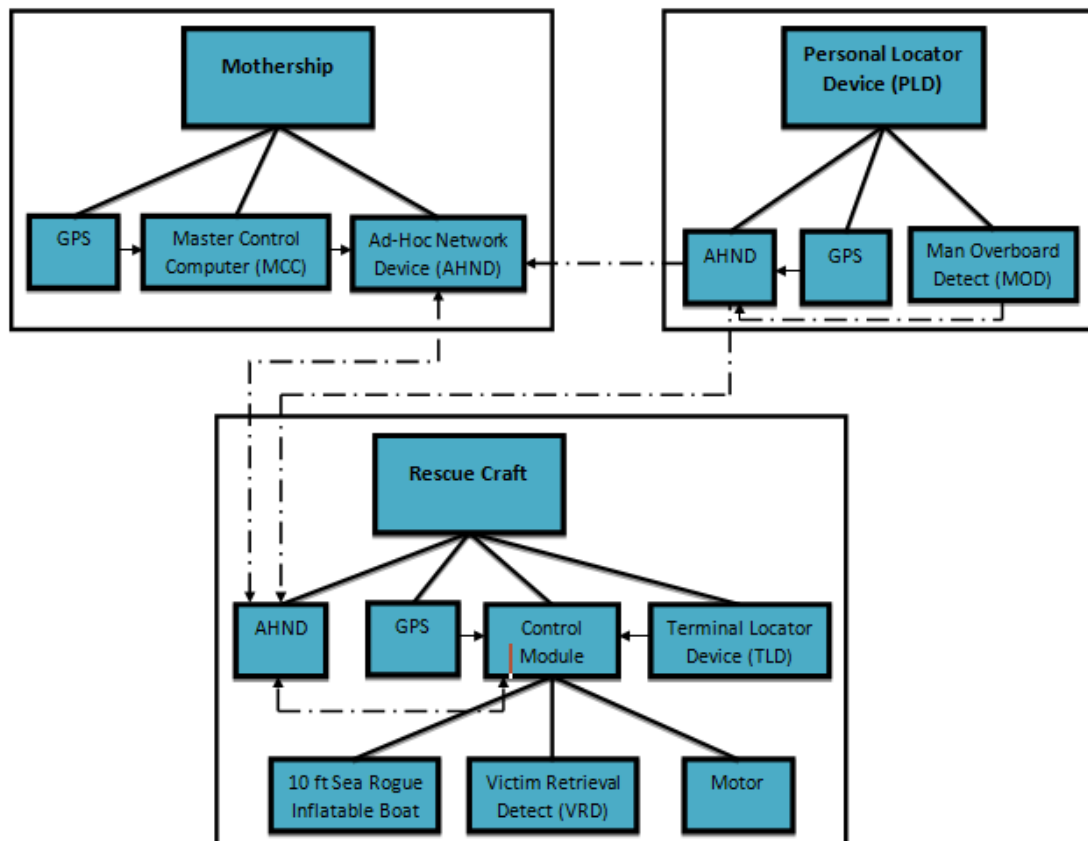
Figure 3 – Main System Modules

All three modules communicate with one another by means of the XBee Ad-Hoc network. The Mothership Module transmits and receives information from the rescue boat and simply receives data from the personal locator device. The rescue boat receives information from the personal locator device as well. Data is indicated by the dashed lines in the figure.

Summary

This section developed an overview of the autonomous robotic rescue boat showing how the three main systems interact with each other. It also shows what each module consists of and what their functions are. For a more detailed explanation of each module see their respective sections in the System Design section of the document.

System Design Overview

Upon developing a basic structure and modules for the overall system, the modules needed to be broken down into subsystems. Each subsystem is made up of various devices that perform the required tasks to make the system work as a whole. For each of the modules of the previous section, a more detailed description was developed.

Mothership

The Mothership module consists of four components: the GPS device, the Raspberry Pi, and the XBee Ad-Hoc Network Device (AHND). These systems will all interact with each other to successfully meet the system requirements as described in the Mothership Module section in the Project Requirements.
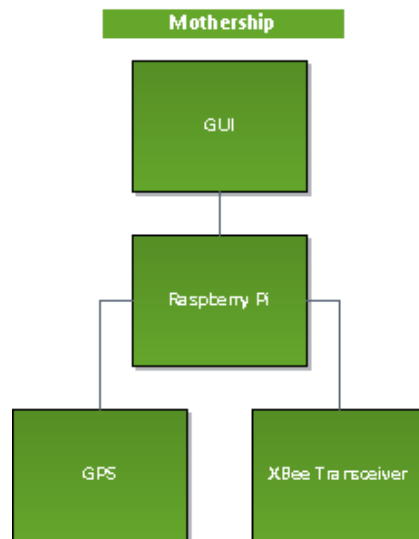
Figure 4 – The Mothership Module System Diagram

GPS

The first component is a GPS device for acquiring the current position to be transmitted to the rescue boat. The device being used is an EM406a GPS receiver which was readily available from a previous version of this project. It operates at five volts with TTL serial output and costs $60 from the Adafruit website.
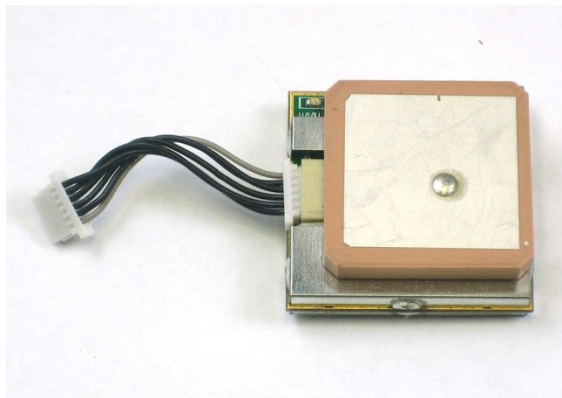
Raspberry Pi

The Mothership Module also has a master control computer which runs the software displaying the current status of the system for a user on a Pyle PLHR77 12V monitor with composite video input. The Master control computer is a Raspberry Pi and is in charge of relaying the mothership's current position from the GPS with the Ad-Hoc network device. The software displaying the status of the system for a user is capable of displaying current locations of the man overboard and rescue craft with respect to the mothership. The software is written in Python allowing it to be executed on a wide range of devices.

Figure 6 – Mothership Module Raspberry Pi and Pyle Monitor

The Raspberry Pi was chosen for the mothership computer because of its small credit-card size, processing power, and ability to integrate with the GPS, XBee, and monitors. The Raspberry Pi is a small computer running Linux with sixteen general purpose I/O pins as well as serial pins for interfacing with the GPS. It also has two USB ports used for a keyboard connection and FTDI cable hookup to interface with the XBee module. The Raspberry Pi runs off of five volts and can consume up to 700mA. It runs on a 700MHz processor and has 512Mb of ram (Premier Farnell plc., 2009).

In order to not clutter the SD memory card that stores all the programs and operating system, Arch Linux was chosen to be installed. Arch Linux is a bare bones operating system and any programs that were needed were downloaded from the packet manager pacman. Some of the required programs were

XTerm and Enlightenment for a graphical user interface. This was chosen as it provided the user with a simple, low resource, but aesthetically pleasing GUI. Python 2.7 was used for programming the software on the mothership unit and pyqtgraph was also installed to create a GUI for displaying all the units on a graph. The system was set up to boot and automatically log in as the root user. The boot time takes approximately thirty seconds, but loading the mothership program and GUI adds another twenty seconds to the start.

To interface with the XBee module, the XBee was connected to the Raspberry Pi with an FTDI cable on the USB port. In the python code, the XBee was accessed with the pyserial module. To interface with the GPS, the GPS was connected to the Raspberry Pi's GPIO UART pins. Again the pyserial module was used to connect to this and a library was written to collect the desired information from the GPS.

Ad-Hoc Network Device (AHND)

The Ad-Hoc network device is responsible for transmitting the GPS data to the rescue boat and receiving the locations of the rescue boat and victim. The transmission is accomplished through the use of an XBee PRO. It is capable of achieving a line of sight range of 3200 meters, has a data transfer rate of 250 Kbps and runs in the 2.4GHz frequency range. The logic levels on the chip are 3.3 volts and it also runs at this voltage. By utilizing a breakout board, it is able to be powered from a 5V source.

Figure 7 – XBee Ad-Hoc Network Device (Mouser, 2013)

Graphical User Interface

The graphical user interface is written in Python using the pyqtgraph module that allows for easy plotting to display the status of the rescue operation with each of the modules on the same screen. It displays messages indicating the current status of the rescue mission as well as the path traveled by each module. An arrow is displayed to indicate the course of each module. A legend on the bottom right of the screen indicates what color corresponds to each module. The graph can also be overlaid on an image of the current location such as a map downloaded from GoogleMaps. The coordinate locations are determined by indicating the GPS coordinates of the bottom left and top right of the map image

used as the background. This may be used to determine where each respective module is and aid in the rescue effort if a manned team would be deployed.
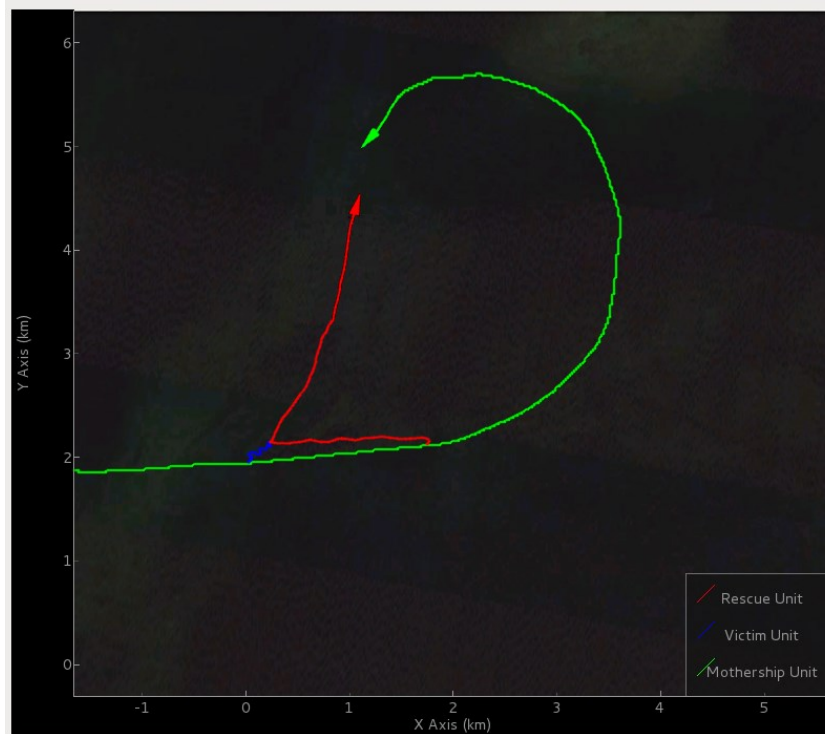
Figure 8 – Screenshot of a typical view from the GUI

Enclosure

To protect the Raspberry Pi from foreign objects and to allow for a more aesthetically pleasing look, a case was made for it. The case design was found online from the Adafruit website. A Solidworks .dwg file was downloaded and the case was lasercut in the Washburn shops, see Appendix G – Drawing File for Pi Box. The case still allowed for the monitor, power cable, GPS, and XBee to be connected to the Raspberry Pi without blocking any ports.

Figure 9 – Raspberry Pi enclosure

Personal Locator Device

The Personal Locator Device consists of an Ad-Hoc Network Device, a GPS, a Man Overboard Detector, an AM Transmitter, a Ferrite Core Loopstick Antenna and an Atmega 328P. These subsystems are designed to achieve the goals as stated in the Personal Locator Device section of the Project Requirements portion of this document.

Figure 10 – Personal Locator Device
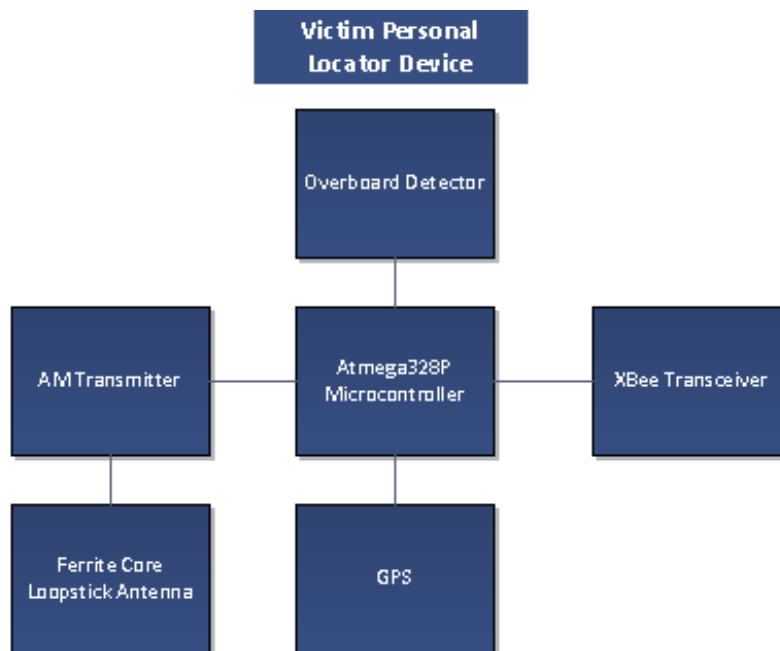
Figure 11 – Personal Locator Device Module System Overview

Ad-Hoc Network Device

The Ad-Hoc network device for the Personal Locator Device is used for transmitting the location data of the victim. For more information about the XBee Pro chip being used, see Ad-Hoc Network Device (AHND) in the Mothership system design section.

GPS

The GPS device used in the Personal Locator device is used to obtain the victim's current location. With intrinsic 10m accuracy, this system requires augmentation for terminal location. For more information about the chip being used see the GPS section in the System Design Mothership section of the report.

Man Overboard Detector

The man overboard detector is capable of meeting the system requirements as expressed in the Personal Locator Device section of the Project Requirements. A MOB victim simply presses a button to initiate the rescue sequence.

The switch used on the personal locator device is an illuminated, IP67 rated switch from E-Switch. It costs $3.32 from Mouser.com and provides a momentary OFF - (ON) signal to the microcontroller (Mouser, 2013). Current rating is up to 125 mA which will not be exceeded and has a blue LED which may be illuminated to indicate the functionality or state of the personal locator device.

Figure 12 – Personal Locator Device Switch for sending Distress Signal

AM Transmitter

The AM transmitter is used to send a radio frequency signal which may be picked up by the rescue boat when it is within a 30 meter range of the personal locator device. This is turned on by the microcontroller by means of Bipolar Junction Transistors which only allow the H-Bridge device which is the main transmitter driver to have power when a 5V logic signal is applied to the controlling BJT. This allows for power to be conserved and the capability of controlling when the signal is transmitted. See Transmitter Circuit for schematic and operation explanation.

Frequency Selection

One important aspect of the antenna design is the tuned frequency. This had to match the frequency that the transmitter and receiver circuit was running at to provide the most effective signal transmission and reception. Because this experiment is carried out in the Massachusetts area and using the AM low frequency band, a quick search of the FCC website was conducted. See for result of the AM band radio station search (FCC, 2011). Competing the search for the stations, it was found that the largest gap with no broadcasting stations occurred just above the 900 kHz band, see Appendix C – AM Station Query (FCC). By searching for oscillators around and in multiples of this frequency, an acceptable operating frequency was found at 921.6 kHz with no stations +18.4 kHz and -31.6 kHz. This was determined to be our best choice operating frequency as there is a larger gap between these frequencies so that a simple band pass filter should be capable of filtering out the desired signal from nearby interfering signals. Additionally, the oscillator being used, when divided by 8 with a modulo counter, is within the required

frequency (Digi-Key, 12). This oscillator has a frequency of 7.3728 MHz that resulted in a transmitted frequency of 921.6 KHz.

Ferrite Core Loopstick Antenna

The Ferrite Core Loopstick antenna is used to transmit the signal generated by the transmitter circuitry. This antenna has been tuned to the radio frequency of the transmitter for maximum transmission range. It is inductively coupled through primary and secondary windings on the same ferrite core for impedance matching which allows for increased signal amplitude on the antenna. See Appendix D – Antenna Tuning for a detailed explanation on antenna tuning.

Atmega 328P Microcontroller

The microcontroller used in the Personal Locator Device is an Atmega 328P which uses a 7.3728MHz crystal. Additionally, the output of the clock signal on a pin of the microcontroller is used to drive the transmitter circuitry which is activated by the Atmega applying a logic HIGH to the controlling BJTs. The microcontroller samples for the button press to activate the distress signal and sends information over the XBee network. It also takes the coordinates from the GPS device and relays them to the rescue boat and mothership units, respectively.
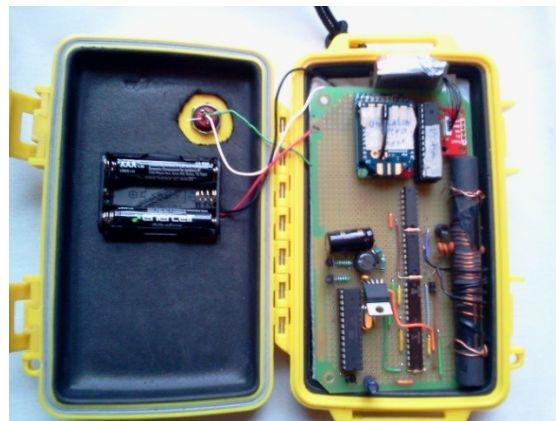
Figure 13 – Inside the PLD

The ATMEGA 328P was chosen for the victim module because of the low profile size and easy integration with the XBEE, compass, and GPS modules. The ATMEGA 328P has 32Kbytes of flash memory, and 28 pins. The ATMEGA 328P is just 3.4cm by 0.8cm in size making it possible to mount it virtually anywhere. The microcontroller also has UART pins for communication with the XBee module. Using the Arduino NewSoftwareSerial library, it is possible to emulate another serial port for communication with the GPS on pins four and five of the ATMEGA 328P. From Mouser.com, the 8-Bit microcontroller costs $2.24 and runs off of a 1.8-5.5V supply (Mouser, 2013).

Figure 14 – Atmega 328P Microcontroller (Mouser, 2013)

On the victim module the same pins were used to interface with the GPS and XBee modules to keep consistency. Pin 12 was used for the distress signal button and pin 13 was used for activating the terminal locator device transmitter.

Enclosure

In an effort to protect the enclosed circuitry for the victim module from water, an Otterbox was used. Because its size was large enough to enclose the circuit, but small enough to be worn on a lifejacket, it was a perfect fit. The Otterbox is 6.85" x 4.57" x 1.82" in size and is able to contain the batteries, GPS, XBee, and other components of the victim unit. The box is rated with an ingress protection of 68, meaning is can be submersed up 100 meters (NPD Group, 2013). The button was also purchased with an IP-68 rating and mounted to the front of the Otterbox through a drilled hole. The unit was tested for water tightness by submersing it three feet in a pool of water with a paper towel on the inside of the case. The button
was also pressed while underwater. After drying the outside of the box, it was opened and the paper towel was dry, indicating it was capable of protecting the circuitry from submersion underwater.

Figure 15 – Otterbox Dry Box used as Personal Locator Device Enclosure (NPD Group, 2013)

Rescue Boat Module

Many of the components used in the PLD and the Mothership Modules as well as others are seen below.
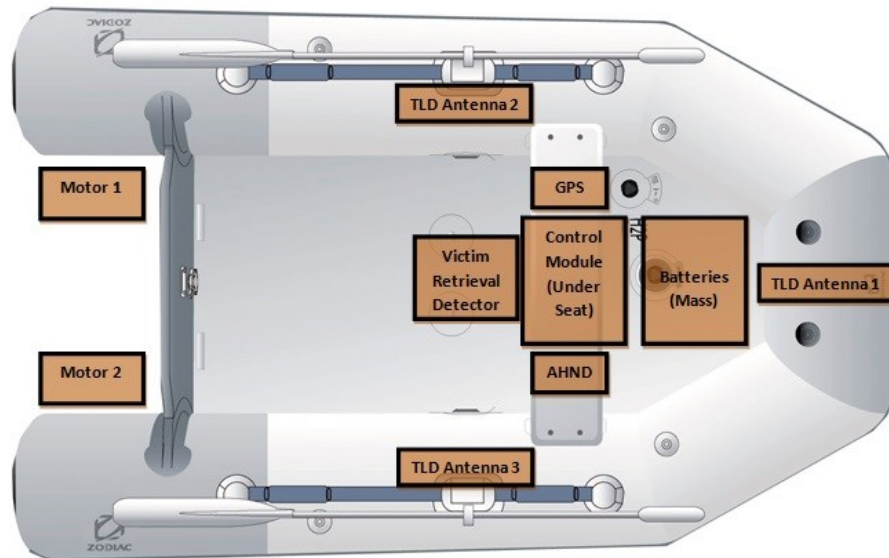
Figure 16 – The rescue boat with basic component layout

The Control Module houses the core microprocessor (Atmega 328P), the motor driver circuitry and terminal locator circuitry. Alongside these components are the Ad-Hoc Network Device (XBee) and victim retrieval button. Strategically placed about the rescue boat are the terminal locator device receivers (Grundig portable radios). The motors are attached as far apart from one another a possible at the stern of the boat.
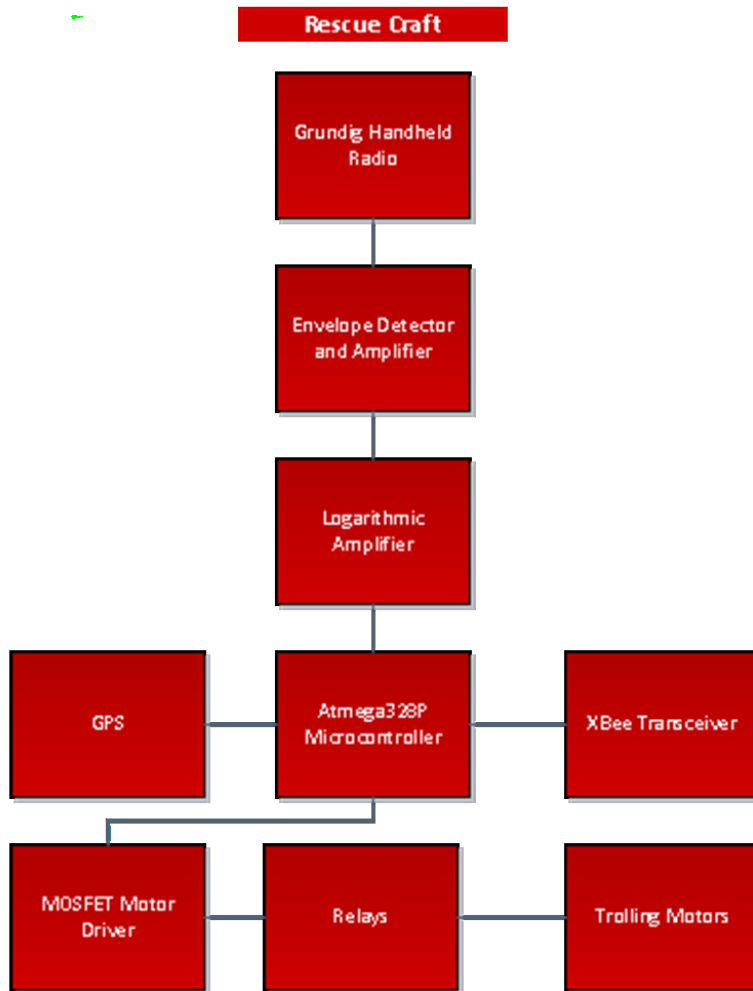
Figure 17 – Rescue Craft Module System Overview

Ad-Hoc Network Device

The Ad-Hoc network device is used for receiving data from the Personal Locator Device as well as the Mothership Module. For more information about the device used see the Ad-Hoc Network Device (AHND) section in the System Design Mothership section.

The Boat

The craft being used as the rescue boat is an inflatable displacement-hull boat. It houses all the electronics and power equipment listed in this section. Its task is to carry the person to safety after he or she has climbed into the boat. The craft used for this project is a 10 foot Sea Rogue. An image of the boat may be seen in Figure 18.

Figure 18 – A view of the Rescue Boat from Stern

GPS

The GPS for the rescue module is used to determine the location of the boat. For more information about the device being used see the GPS section in the System Design Mothership section of the report.

Compass module

A compass was used to provide the rescue craft with the most accurate heading information to increase the effectiveness of navigation. The compass module used was an HMC6352 from Sparkfun Electronics. The module was chosen because it could easily be integrated with the I2C interface on the ATMEGA328P. It also runs on the five volt logic levels available on the rescue unit. The resolution is up to 0.5 degrees and can be read at a rate of up to 20Hz. The low current draw of 1mA was also attractive. The small size of 1.5x1.5cm made it possible to mount it virtually anywhere without being intrusive to other systems. The low price of $35 made the compass a great option for the features it exhibits.

The compass provided the rescue craft with important heading information that would be used to more accurately determine the current heading than the GPS. Due to the electromagnetic interference exhibited by the relays and high current carrying wires, the compass could not be placed near these devices. Using a mechanical compass and running the motors to simulate operation, the compass was moved to various positions on the craft to determine where there was no electromagnetic interference. It was determined that the best location for the compass would be on the left side of the cross-board seat. To protect the compass from splashing water, it was mounted in a project box purchased from RadioShack. A telephone cable was used to connect the compass to the five volt power supply and the I2C port of the ATMEGA328P.

Atmega 328P Microcontroller

The Atmega 328P controls the trolling motors by means of a motor driver circuit, activates the terminal locator device, and determines if the victim is in the boat. From the Ad-Hoc Network Device, it collects GPS data from the rescue boat, mothership and personal locator device. It then uses this data to analyze which path it must take to travel both to the mothership and victim. Depending on the distance between the personal locator device and the rescue boat, it also uses the terminal locator device.

The Atmega 328P was chosen for the rescue unit because of the low profile size and easy integration with the XBEE, compass, and GPS modules. The Atmega 328P has 32Kbytes of flash memory, 28 pins, and runs on an internal 8 MHz crystal with the Arduino bootloader. The Atmega 328P is 3.4cm x 0.8cm in size. The microcontroller also has UART pins for communication with the XBee module. Using the Arduino NewSoftwareSerial library, it is possible to emulate another serial port for communication with the GPS on pins four and five. To interface with the compass on the rescue module, the I2C pins on the microcontroller were used.

On the rescue module, pins 14 - 17 control the MOSFETS that switch the relays for the motor control. Pin 13 was also used for reading the return to home button. Pins 23-25 are three of the six 10-bit analog to digital pins on the microcontroller and were used for reading the values from the receiver circuit for use in the terminal location algorithm.

Victim Retrieval Detector

The victim retrieval detector signals when the victim is on the rescue craft. It consists of a switch which the victim must activate. Once the switch is pressed, the rescue craft navigates to the current primary vessel location as reported by the Mothership Module's GPS. The switch is placed in a visible location and protected from accidental triggering. Possibilities for making it more visible include using an illuminated switch or including an audible device indicator near the switch.

For the victim retrieval detector, the GCX3226-24 mushroom style illuminated pushbutton emergency stop switch was used. The switch was ordered from AutomationDirect.com for $12.50 and with its 40mm head, is easily visible and depressible.

Figure 20 – Return Home Button on the Rescue Craft (automationdirect, 1999-2013)

Terminal Locator Device

The terminal locator device is used to obtain a better fix on the location of the victim due to the GPS error. It consists of three receiver circuits connected to the output of three commercial radios. The circuit can be seen in the Receiver subsection of the Design and Testing section of the report. The positions of the three radios form a triangular shape as denoted in Figure 16. This allows for determining the angle as well as the direction to where the victim is located. This arrangement can be used to align the rescue boat's bow directly to the victim's PLD. When the bow is pointing toward the victim, the front receiver antenna will have the largest peak envelope signal and the aft antennas will have a lower voltage, but equal to one another.

TLD Mounting

To obtain the best signal for homing of the receivers, the proper mounting location was to be found. Ideally the front receiver would be placed in the bow of the boat and the other two receivers would be mounted in the stern as far apart from each other as possible, but the same distance apart from the center of the boat.
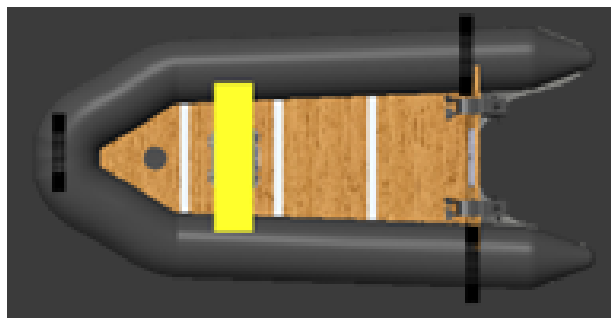
Figure 21 – TLD Mounting, Ideal Mounting Locations

Due to the electromagnetic interference from the motors in the stern of the boat, the receivers were moved up to the center of the boat as seen in Figure 22. This location saw the least interference from the motors. Additionally, on the rescue craft, there were oarlocks present which provided an easy mounting option as indicated in the figure below by the orange outline.

Figure 22 – TLD Mounting on the Rescue Craft – View from Stern

To better protect the receivers from the water, they were placed in a project box purchased from RadioShack. A 3.5mm male to male headphone jack cable was used to connect the receivers to the bandpass filter on the rescue unit.

Grundig Handheld Radio

The handheld radio receiver is a small commercial radio with good sensitivity and selectivity that receives the signal from the PLD transmitter. This signal is fed to the following circuitry which filters it to remove noise from other stations and only read information that is relevant to terminal location.

The radio used for the receiver device of the terminal location module was an Eton Grundig Mini 400 AM/FM Shortwave Radio. This was purchased at Radio Shack for $34.99 and had an accessible headphone jack which was used to connect to the terminal locator circuitry. The digital display provided an easy and accurate means to tune to the desired station. A lock button was also an added feature of the radio which prevents the user from accidentally turning it off.

Figure 23 – Radio Receiver used for the Terminal Locator Device (RadioShack Corporation, 2013)

Envelope Detector and Bandpass Filter

The output of the radio receiver is sent to an envelope detector and bandpass filter. The modulation signal which carries information for the rescue boat is a 1 kHz tone which is extracted from potential background noise by means of a bandpass filter. See Deliyannis Bandpass Filter for further explanation and design.

Logarithmic Amplifier

The logarithmic amplifier allows for operation over a greater range than would be achieved if the received signal envelope was simply captured by the linear ADC. It amplifies small signals with a much higher gain than larger signals. It also ensures that the output of the bandpass filter does not exceed 5V which is the maximum allowable input magnitude to the analog to digital converter of the microcontroller. See Logarithmic Amplifier for circuit design.

MOSFET Motor Driver

The MOSFET motor driver takes the logic level from the Microcontroller and provides the high current drive required for the relays which are used to drive the 14 A trolling motors. These relays operate from a 12V signal which is provided by the battery on board the rescue craft. The MOSFETs used in the circuit are four T1N60 N-Channel MOSFETs with a maximum continuous drain current of 1.3A. The TO-220 package was purchased from DigiKey for $0.72 (Digi-Key, 1995-2013). For more information on the implementation and placement in the circuit for the T1N60 MOSFETS, see the section Motor Control.

Relays

The relays drive the trolling motors with bi-directional capability. Because the motors do not require a rapid response time on the order of milliseconds, relays were deemed a plausible component to use for driving the trolling motors. Additionally, there is less power loss from heat in comparison to the use of MOSFETs with a heat sink for driving the motors. This alone makes the method for propulsion more economical and efficient. The single pole, double throw relays used for the rescue craft were Song Chuan 896H-1CH-D1SW-R1-12VDC relays as seen in the image below. These were purchased from Mouser.com for $4.31 per relay. They each can handle up to 50A and have a coil current of 133mA. Their small size of 1.1" x 1.25" x 2.7" is desirable as there is the necessity of four relays for bi-directional drive functionality. See MOSFET Motor Driver for the use of the relays in the final motor driver circuit.

Figure 24 – Song Chuan SPDT 896h-series Relay (Mouser, 2013)

23

Trolling Motors

The motors being used for propelling the rescue craft are two 12V trolling motors, each with a rated thrust of 30 pounds from MinnKota. The motors were chosen based on their operating voltage of 12V and simplicity of applying to the current design due to the clamp mounting apparatus which allows simplicity in fastening the motors to the backstay of the rescue boat. Additionally, there are several options for adjusting the depth that the motor sits in the water as well as the angle at which it propels the boat. A quick release mechanism allows the motor to be propped up out of the water during transport and when not in use. A single motor costs $109.99 new from the MinnKota website.

Figure 25 – MinnKota Endura C2 Trolling Motor (MinnKota, 2008-2013)

Enclosure

For the final rescue craft system, the circuitry described above was fit into a tower configuration which was placed into a marine enclosure box alongside the battery. This provides adequate protection for testing and keeps all the main components together in a single location making it easier to transport and remove it from the rescue craft. An isometric view of the Solidworks model for the component holder is seen in the Figure 26 below. This was used to lasercut acrylic while standoffs and screws were used to put the final pieces together.
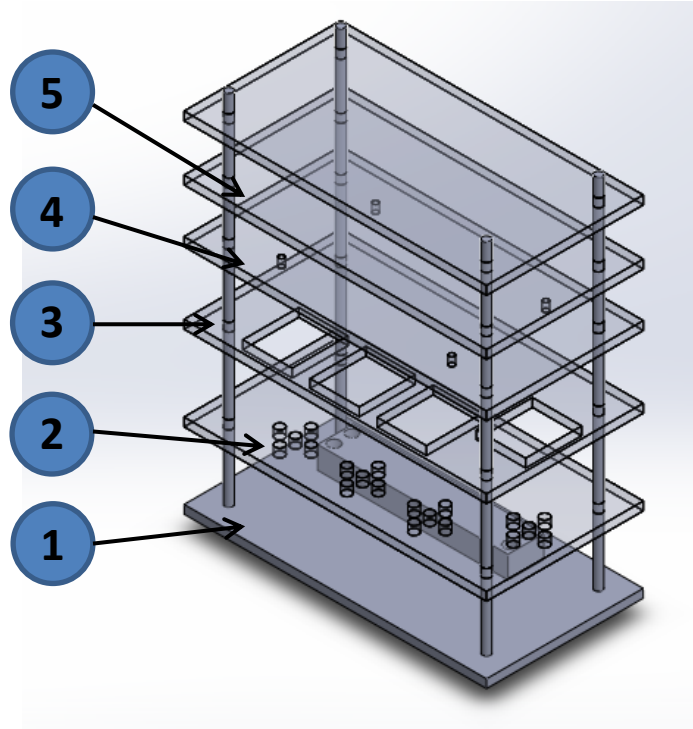
Figure 26 – Solidworks Model - Stacked Components Holder

The lower compartment 1 contains the wire bus which connects the battery power and ground terminals to the rest of the circuits, including the relays. The second level contains the relays with a support layer indicated by the square cut-outs on layer 3. Next, the motor controller circuitry with the GPS, Atmega and MOSFETs are fastened to the acrylic layer 4. Finally, the terminal locator circuitry is mounted to the layer 5. Dimensions for the CAD model may be found in the Appendix J – Dimensions for Circuit Tower.
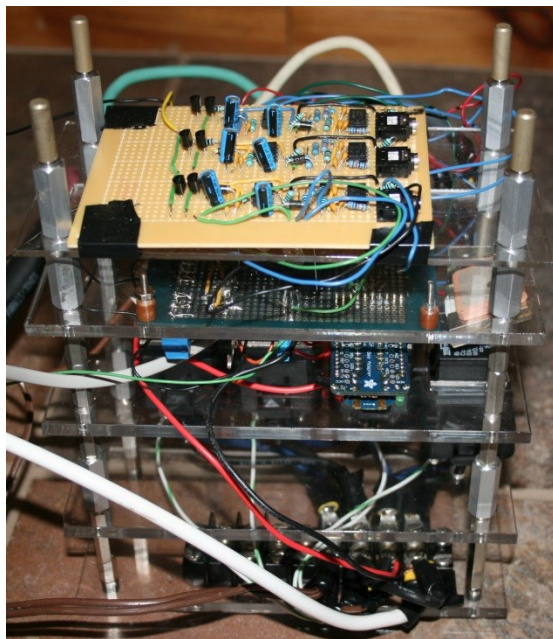
Figure 27 – Tower with all Circuit Elements Attached

The tower with the attached circuitry was placed into a Snap-Top Everstart Marine Battery Box purchased at Wal-Mart (Wal-Mart Stores, Inc., 2013). The 17.6" x 10" x 10.5" box was adequately large to fit both the battery and the circuitry tower inside. There also was an adjustable separator provided with the box which allowed the circuitry to be separated from the battery. The original and final enclosures are pictured in the Figure 28 below.
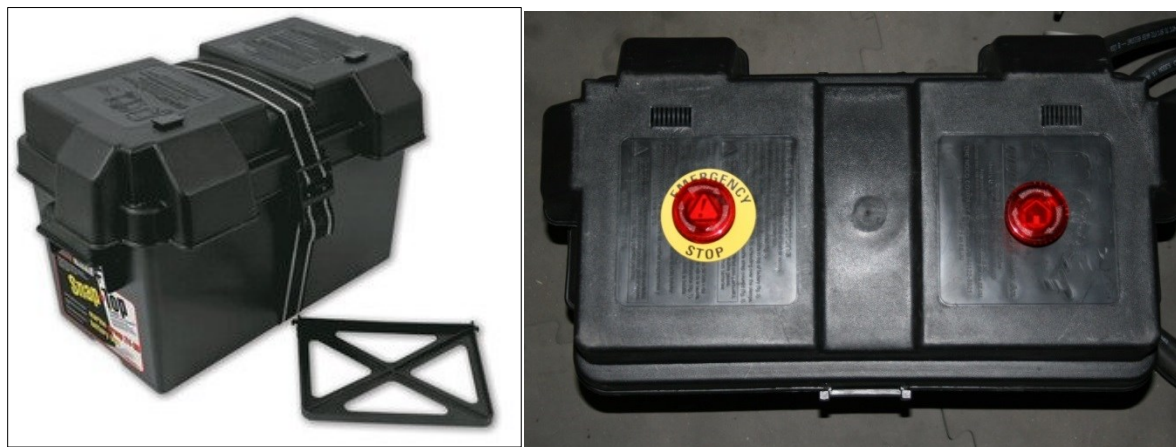
Figure 28 – Marine Battery Box used for Rescue Craft

Summary

The above sections describe the system design of all the modules contained in the project. The entire system consists of three main modules, the Mothership Module, the Personal Locator Device Module, and the Module. Each of these modules consists of multiple components or smaller modules responsible for subtasks to accomplish the goal of creating an autonomous robotic rescue boat.

Design and Testing

This section covers the process and methodology behind the design and testing of various main components of the rescue equipment.

Software Design

Careful planning was required to effectively execute the task of locating and retrieving a man overboard. Because there were three main locations of interest, it was decided there should be three units: the Mothership Module, the Victim Module, and the Rescue Module. Having three units communicating with each other at the same time was another challenge to be addressed, but one that could be handled by the XBee modules and simple coding practices. For example, limiting the amount of information that was being transmitted over the air was necessary to minimize packet loss. One such case was when the rescue craft was heading to the victim and did not need to know the location of the Mothership. Here the Mothership did not broadcast its location until the Rescue unit requested it. Each of the three modules acted as a type of state machine, waiting on information from the other to change states. For more detail, see each Module's respective section.

Mothership Module

The Mothership module uses a state machine to determine what it should do at any given time. While in state 1, the Mothership is continuously waiting for a distress signal from the PLD module and once that is received, it initializes the Rescue Module. Once the Mothership receives a confirmation that the Victim and Rescue Modules initialized properly, it initializes the Graphical User Interface (GUI) thread to begin plotting the coordinate locations of all the units in respect to one other. The plotting thread runs in the background until the victim is returned to the mothership, continuously updating the locations as new ones are received. At the same time the thread is started, the Mothership is set to state two. The process in full is seen in Figure 30.

In state two a thread is started to read and store the Mothership Module's current GPS coordinates to a file for plotting. It runs in the background and, without interruption stores a new coordinate location every 2.5 seconds. In these 2.5 seconds, it collects ten GPS coordinates, takes the average of them, and stores this location in an attempt to remove some GPS jitter. In state two, the Mothership also reads the XBee for any new coordinates from the victim module as well as any messages it may send to be stored and filed for plotting. Should the Rescue unit be close to the victim, it will be printed to screen on the GUI as seen in Figure 32. The Mothership unit will also poll the XBee for any message indicating the victim is on the rescue craft and has pressed the return to home button. Upon receiving this message, the Mothership unit ends the GPS thread that reads and stores the Mothership unit coordinates to file and sets its state to state three.

Once in state three, the thread to store the current GPS coordinates is initialized again, but also set to broadcast the coordinate locations to the rescue craft. After this is initialized, the Rescue unit reads the XBee and continuously checks for a message saying that the rescue unit is home. Once this condition is met, state one is initialized again.
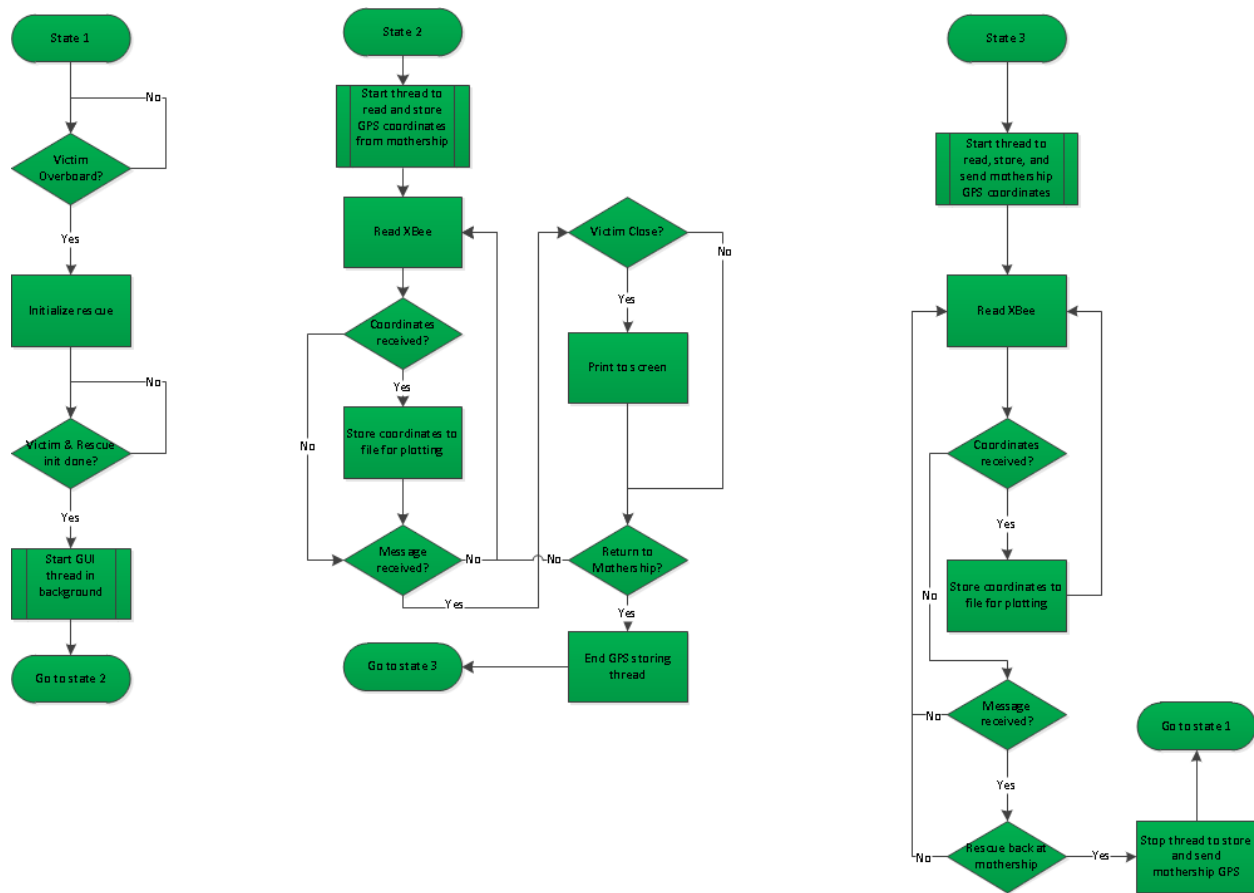
Figure 30 – Mothership Unit State Diagram

Another background process running on the Raspberry Pi is the user input process. The keyboard that is plugged into the Raspberry Pi is polled for key strokes. If the user presses the spacebar at any time, the rescue craft will be stopped and put into a remote control override mode. The arrow keys can be used to steer the craft. The right shift key can also be used to kill the motors. The "r" key restarts the program on the rescue module.

Graphical user interface

The graphical user interface (GUI) is used to relay information from the programs on all three units to the users. In order to accomplish this, the Raspberry Pi was used along with pyqtgraph running in a background process. Pyqtgraph made it possible to easily plot the locations of all the modules, place an arrow in the direction each module was heading, and overlay a map of the current area to the background as seen in Figure 32. The Mothership module stores all the GPS coordinate locations as they are received and stored to a comma separated values (CSV) file in the root directory of the program along with the distance from the first received coordinate in meters. An example of a line in this file can be seen in Figure 31.

```
B,42.254962,-71.255098,E,2000.26164234,32.0715245628
C,42.254934,-71.254999,E,2008.87041304,28.7821374281
C,42.254826,-71.254719,E,2033.21893642,16.0945013377
```

Figure 31 – Example of Coordinate Log File

The first value in each line is the sending modules identifier, the second value is the latitude coordinate (in degrees), the third the longitude coordinate, the fourth value is a validity bit, and finally the x and y coordinate location in meters of the current module in respect to the first received coordinate. Any messages received from the units were also stored to a file, for example: if the victim is close to the rescue unit.

These files were then periodically read by the background process of the GUI which would re-plot all the coordinates and write the most recent message to the screen. The GUI also plots the general location the modules head in by calculating the angle between the last two received coordinate locations of each module. Should a module not be online yet and there are no coordinates stored in the log file, the unit will not be displayed on the screen.
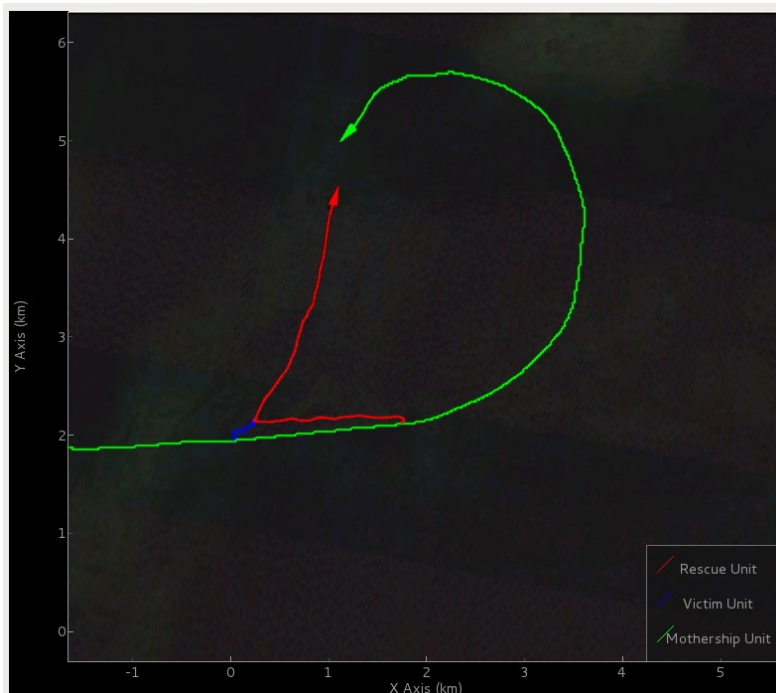
Figure 32 - Mothership GUI

Personal Locator Device Module

The PLD Module, much like the Mothership module, runs on a state machine type program structure. The entire flowchart can be seen in Figure 33. While in state zero the unit continuously polls the overboard button for a key press. Once the button is pressed, the victim unit checks for a GPS lock and sends a message to the mothership indicating it has completed the initialization process. It then moves on to state one.

While in state one, the Victim unit first reads the GPS for a lock. If a lock is present, it will store the current position in an array of ten coordinate locations that will be used to determine the average position in an attempt to remove some GPS jitter. Next, the average of the last ten positions is sent to the rescue unit and mothership for calculating the path to plotting in the GUI. The XBee is then polled for any new messages from the Mothership or Rescue modules. Should the message indicate that the rescue unit is close, the PLD module enters state two upon it activation of the terminal locator device. If the message is not received, the process is repeated.

While in state two, the XBee is read for any messages. If a message is received stating that the terminal locator device is out of range, the Victim module reverts back to state one. The PLD module also checks if the rescue module sends a message indicating it is close to it. If this is the case, the victim unit changes to state three, otherwise it repeats the process.

In state three, the PLD sends a message to the mothership module stating it is on the rescue module.
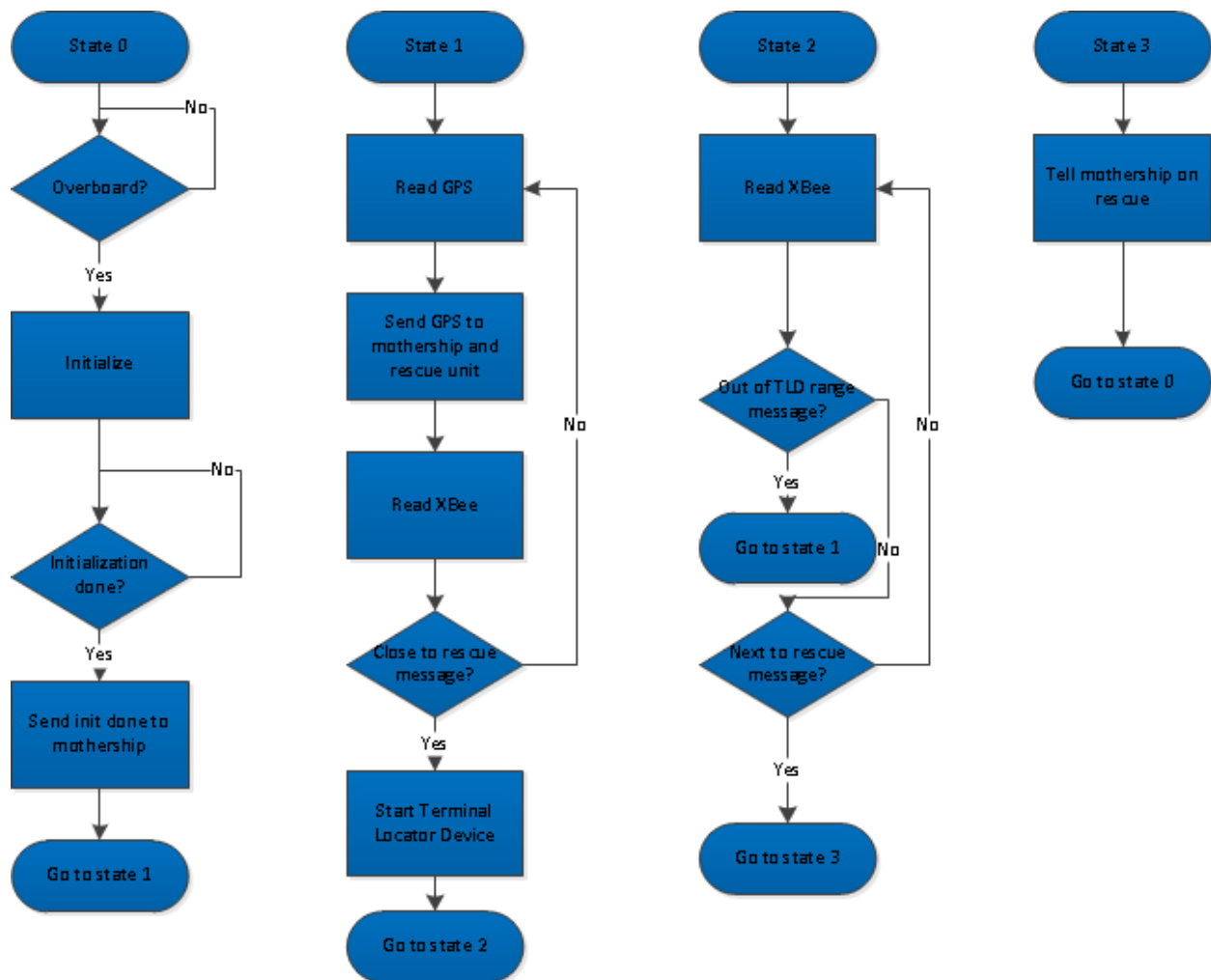
Figure 33 - Personal locator Module state diagram

30

Rescue Module

The rescue module is perhaps the most complicated of all three units as it is tasked with many processes in a short time. In order to organize the code in a more structured manner, it is also put in a state machine as seen in Figure 34. In state zero, the rescue module waits until it receives a boot message from the mothership. Once this is received, it reads the XBee and determines if the remote kill switch message has been sent. If it has been sent, it goes into the remote control mode state. If not, the rescue module sends a message to the mothership that the initialization has been completed and enters state one.

In state one, the rescue module checks if it is in range of the terminal locator device (TLD) in which case it goes to state two. If this is not the case, the rescue unit reads its local GPS coordinate location and sends it to the mothership as an average of the last ten coordinates. Next, the rescue module reads the XBee for a new message or coordinate data. If a message is received indicating that it should go to the remote control state, the rescue module will enter the remote control state. Otherwise, if coordinates from the PLD were received, it stores these, calculates the path to the victim, and corrects the course by means of control signals to the motors. This process is then repeated.

In state two, the rescue unit is close to the victim (within range of the TLD) and attempts to home in on the victim. It first notifies the victim and the mothership that it is close to the victim. In this state, the rescue unit continuously checks if the return to home button has been pressed and if it is out of signal range of the victim TLD. If the return home button has been pressed, state three is activated. If the TLD is out of range, the code reverts back to state one. Furthermore, in state two, the rescue unit checks if it should be put into the remote control state. It will correct the motors based on the peak envelope signal read from the receiver circuit. For example, if the right motor has a larger peak envelope signal, then the left motor is turned on and the right turned off to correct the direction of travel.

While in state three, the rescue unit attempts to go back to the mothership. It first sends the mothership a message indicating that it is heading home with the victim. Next it checks if it is close to the mothership, in which case it will notify the mothership that the rescue unit is home and enters state zero. If this is not the case, the rescue unit will find and store its local GPS coordinates to send to the mothership. Next the rescue unit reads the XBee for any messages or coordinates. If a message has been received that it should go into remote control mode, the remote control state is activated. Otherwise if the received packet has coordinates of the mothership, it will store them and use them to calculate the new path to the mothership, correct the direction, and set the motors.
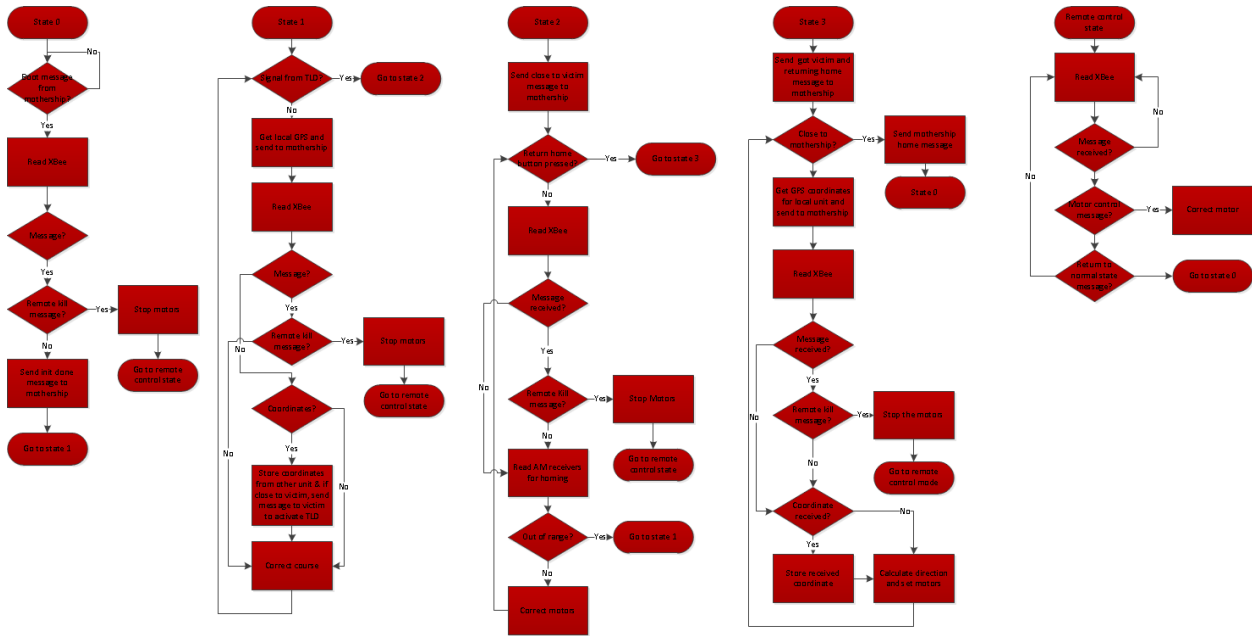
Figure 34 - Rescue unit state diagram

Navigation

The navigation algorithms used by the rescue unit involve two main methods of path planning. The first method is cross track error using the GPS coordinates of all three modules. It bases the direct course as being between the last position of the mothership module prior to rescue initialization and the current position of the PLD Module. The Rescue boat module's position is then adjusted to continually minimize the cross track from this track while proceeding to the PLD. The cross track error algorithm is used in order to reduce the effects of the wind on rescue craft as well as the spiraling effect when homing in on the victim using a simple straight path algorithm. The terminal locator function simply checks the peak envelope signals of the three receivers to determine which way to correct the motors for homing in on the victim.

Cross-track error algorithm

The cross track error algorithm simply determines how far from the straight line path to the victim the rescue unit is. If it is more than five meters from the path, it corrects the rescue craft to stay in a straight line path. The equation used to accomplish this can be seen in the equation below.

$$D = \left(asin\left(\sin\frac{d_{13}}{R}\right) * \sin(\theta_{13} - \theta_{12})\right) * R$$

Where R is the earths radius, d_{13} is the distance from the starting location of the rescue craft, θ_{13} is the angle from the starting location of the rescue craft to the current location of the rescue craft, θ_{12} is the angle from the starting location of the rescue craft to the current location of the victim module, and D is the cross-track distance error.

Using this in the code, every time new Rescue Boat or PLD GPS coordinates are received, the cross-track is recalculated. If the value is negative, the rescue craft knows it is to the left of the desired path and should it be greater than negative five meters, it corrects right. The same principle is true should the value be positive and greater than five. In that case, the rescue unit would be to the right of the desired path and have to correct left.

The cross-track error algorithm helps prevent the rescue craft from traveling too far from the shortest path to the victim. It also allows the craft to quickly correct its course in response to wind or currents that may move it off course.

Terminal location algorithm

The terminal location algorithm allows for the rescue craft to home in on the victim using the three AM receivers located on the boat. If the receiver on the bow of the boat has a weaker peak envelope signal than the receivers on the center of the boat, the rescue unit knows that the bow is pointing away from the victim and has to turn around. Should the bow have a larger peak envelope signal strength than the receivers on the center of the boat, then the boat knows it is pointing towards the victim and if the right receiver has a larger peak envelope signal strength than the left, then the boat is pointing to the left of the victim and will have to correct right as seen in Figure 35. The ideal condition would be when both the center receivers have the same peak envelope signal strength and the bow has the greatest peak envelope signal. Once both the left and right center receivers have a peak envelope signal strength that is greater than a set threshold, the boat knows that it is within one meter of the victim and kills the motors.
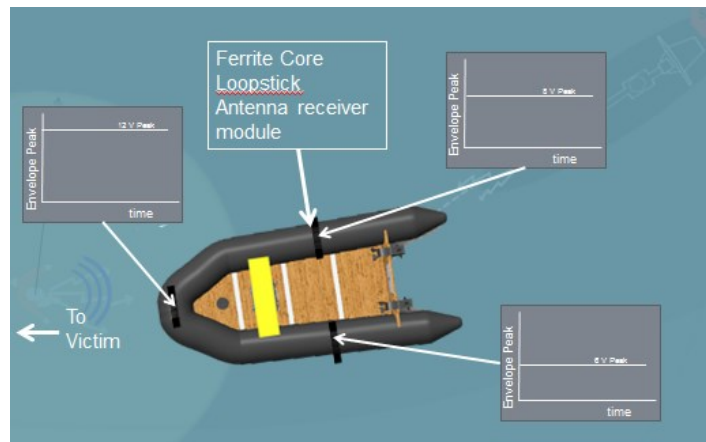
Figure 35 - Terminal locator device peak envelope signal example

Messages Between Modules

In the table shown below are all the messages that are sent between the three units.

Table 1 – Relayed Messages Between Modules

Message	Sender	Meaning
OVERBOARD	Victim	Victim is overboard
INITDN	Victim	Initialization complete
ONRESCUE	Victim	Victim is on rescue craft
INIT	Rescue	Initializing
SEARCHV	Rescue	Rescue is searching for victim
CLOSEV	Rescue	Rescue is close to victim
RETURN	Rescue	Rescue is returning to mothership
FORWARD	Rescue	(Debug) Both motors on
LEFT	Rescue	(Debug) Right motor on (turning left)
RIGHT	Rescue	(Debug) Left motor on (turning right)
WAITING FOR VICTIM TO BOARD	Rescue	Next to victim and waiting for home button press
MCOORD	Rescue	Tell the motherhship to send GPS coordinates
HOME	Rescue	Rescue craft returned to mothership
BOOT	Mothership	Notify rescue unit to initialize
FF	Mothership	Set both motors on
SS	Mothership	Stop the boat
LO	Mothership	Turn the boat right (left motor on)
RO	Mothership	Turn the boat left (right motor on)
RESTART	Mothership	Restart the rescue craft in state 0

Antenna Design

A ferrite core antenna was chosen for the signal transmission based on its size and signal reception capabilities. The ferrite core antenna being used is 4 inches long, allowing for inconspicuous mounting on the rescue craft within a watertight enclosure. Its small profile also reduces the size of the victim's locator device allowing for a non-intrusive safety system that will not hinder the wearer's actions. The antennas were tuned to a low frequency in the AM radio band of 921 kHz allowing the signal to penetrate waves, should the victim be on the other side of a large wave. For the method of tuning the antenna, see Appendix D – Antenna Tuning. Other antenna systems would be impractical of achieving these goals provided the expected operating conditions.

Frequency Selection

One important aspect of the antenna design is the tuned frequency. This had to match the frequency that the transmitter and receiver circuit was running at to provide the most effective signal transmission and reception. Because this experiment is carried out in the Massachusetts area and using the AM low frequency band, a quick search of the FCC website was conducted. See for result of the AM band radio station search (FCC, 2011). Competing the search for the stations, it was found that the largest gap with no broadcasting stations occurred just above the 900 kHz band, see Appendix C – AM Station Query (FCC). By searching for oscillators around and in multiples of this frequency, an acceptable operating frequency was found at 921.6 kHz with no stations +18.4 kHz and -31.6 kHz. This was determined to be our best choice operating frequency as there is a larger gap between these frequencies so that a simple band pass filter should be capable of filtering out the desired signal from nearby interfering signals. Additionally, the oscillator being used, when divided by 8 with a modulo counter, is within the required frequency (Digi-Key, 12). This oscillator has a frequency of 7.3728 MHz that resulted in a transmitted frequency of 921.6 KHz.

Transmitter Circuit
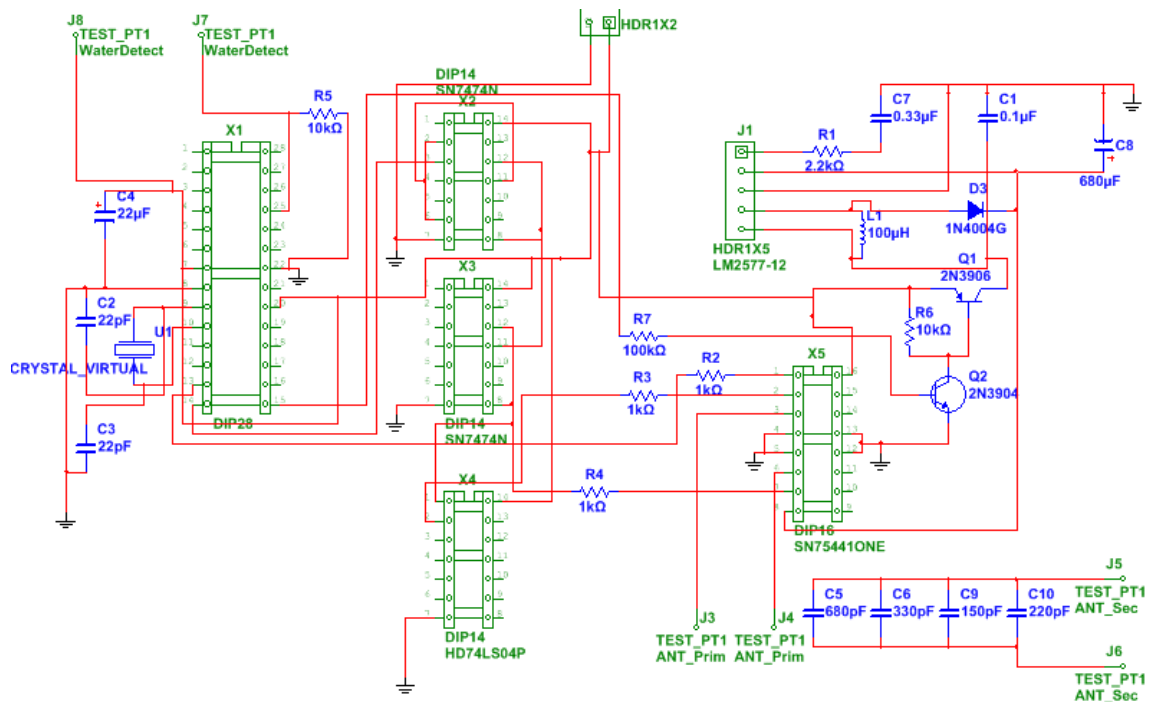
Figure 36: Transmitter Circuit Design

The AM transmitter is made up of 5 integrated circuits (IC). The main part of the transmitter is made up of the SN75441 IC which is a quadruple Half-H Driver. Two of the half bridges are used to drive a square wave into the primary coil of an LC antenna. The Atmega328P microcontroller IC is driven by a 7.3728MHz crystal and runs off of a 5 volt supply. It generates the clock output on Pin 14 and a 1 kHz square wave at pin 13. The clock output is fed into two SN7474N Dual D-Type Flip Flops which divide the frequency down from 7.3728 MHz to 921.6 kHz. To transmit a signal, an antenna had to be tuned for the desired frequency of 921 kHz. See Antenna Design. This square wave is sent to pin 7 of the H-Bridge

35

driver and an inverter, HD74LS04P. This inverted waveform is applied to the Pin 2 of the H-Bridge driver. For higher transmitting strength, a boost converter is used to boost the 5V supply to 12V for the H-Bridge IC. This is achieved by the LM2577 (Texas Instruments, 2013).

The square wave radio frequency signal applied to the Pins 7 and 2 of the H-Bridge are applied to the Pins 6 and 3 of the same IC when the signal on Pin 1 is high. When this is low, the output to Pins 6 and 3 are low. By applying the 1 kHz signal to the controlling Pin 1 of the H-Bridge driver, the radio frequency signal may be modulated in an ON-OFF Keying fashion.
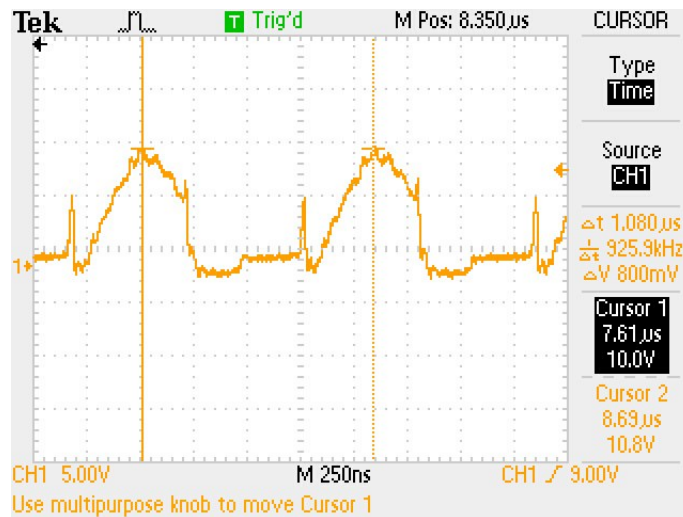
Figure 37 – H-Bridge Driver output Pin 3

Figure 37 shows the output from the H-Bridge circuit at the primary winding coil. As can be seen, the frequency of the waveform is at the radio frequency which the signal is applied to the inputs of the H-Bridge.
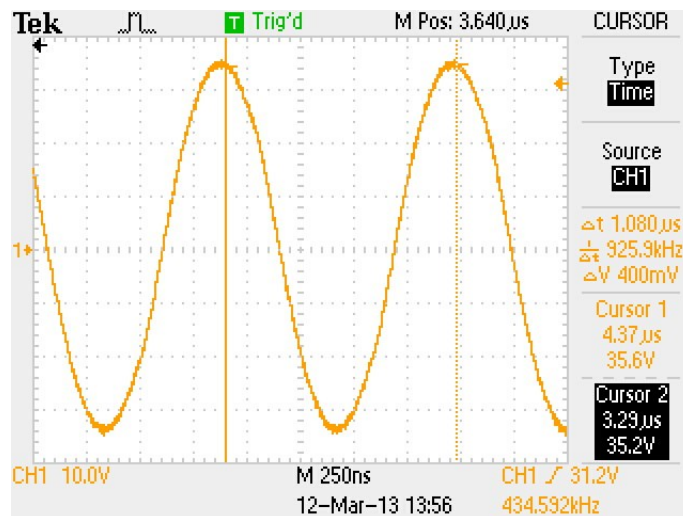
Figure 38 – Tuned LC Antenna Circuit Output Waveform

Figure 38 shows the waveform at the tuned LC circuit antenna which radiates the signal. This also is at the radio frequency as expected.
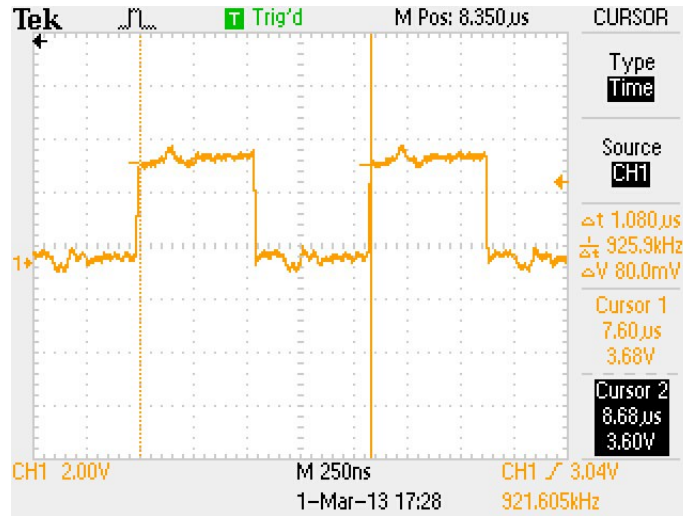
Figure 39 – Divided Clock Signal from Atmega328P

The divided clock signal from the Atmega IC is seen in the Figure 39. This is at the radio frequency that the circuit is operating and is the signal that is applied to Pins 3 and 4, one of which is inverted.
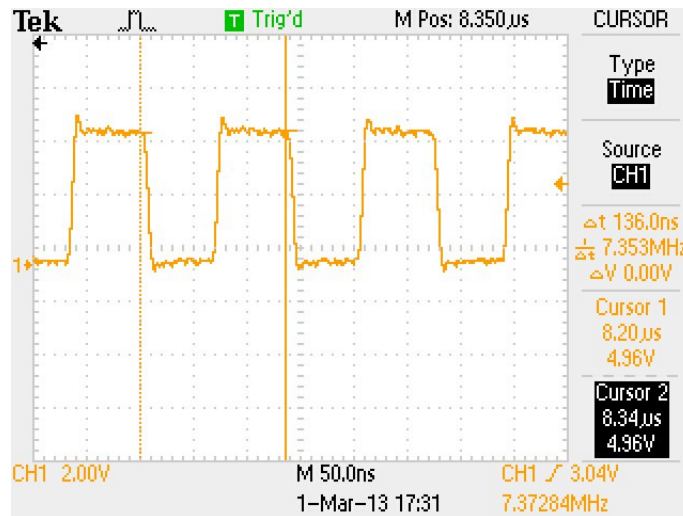
Figure 40 – Clock Signal Output from Pin 14 of Atmega328P

The Figure 40 shows the clock signal output of the Atmega328P which is used to derive most of the other signals being used in the transmitter.
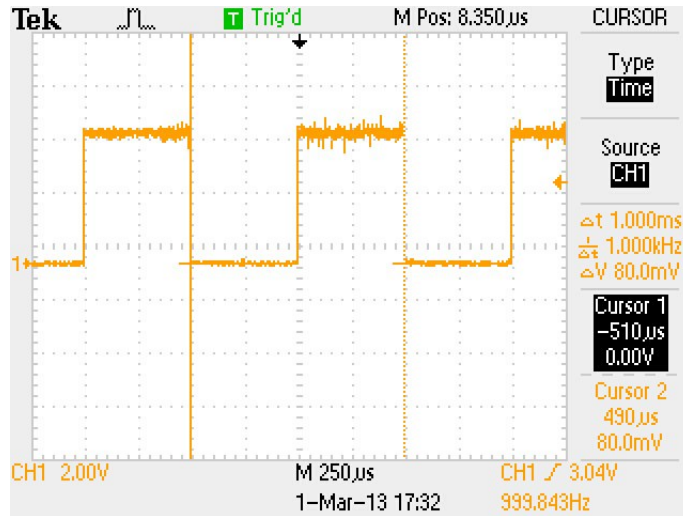
Figure 41 – 1 kHz modulating signal from Atmega328P Pin 13

Finally, Figure 41 shows the modulating signal that is used to ON-OFF key the radio frequency signal.

Additionally, the XBee and GPS devices are connected to the on the Atmega microcontroller which carries out the transmitter processes. These are connected to the physical pins as seen in the Figure 42 below.
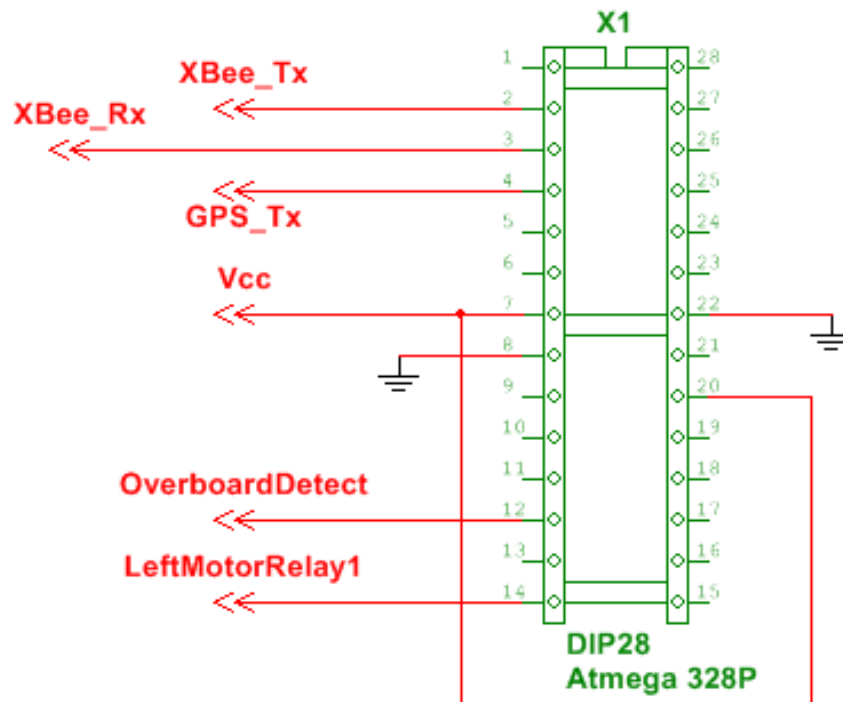
Figure 42 – Atmega Pinout for Personal Locator Device

Receiver Topology

For the terminal location to function properly there had to be three identical receivers that are positioned on the rescue craft. Due to time constraints, the basic receiver is made up of a commercial handheld Grundig M400 Radio which has been tuned to the transmitting frequency, a band pass filter and a final logarithmic amplifier stage. The schematic of Figure 43 shows the post AM receiver stage with the output of the final Operational Amplifier, U3B going to the Atmega328P analog to digital converter.
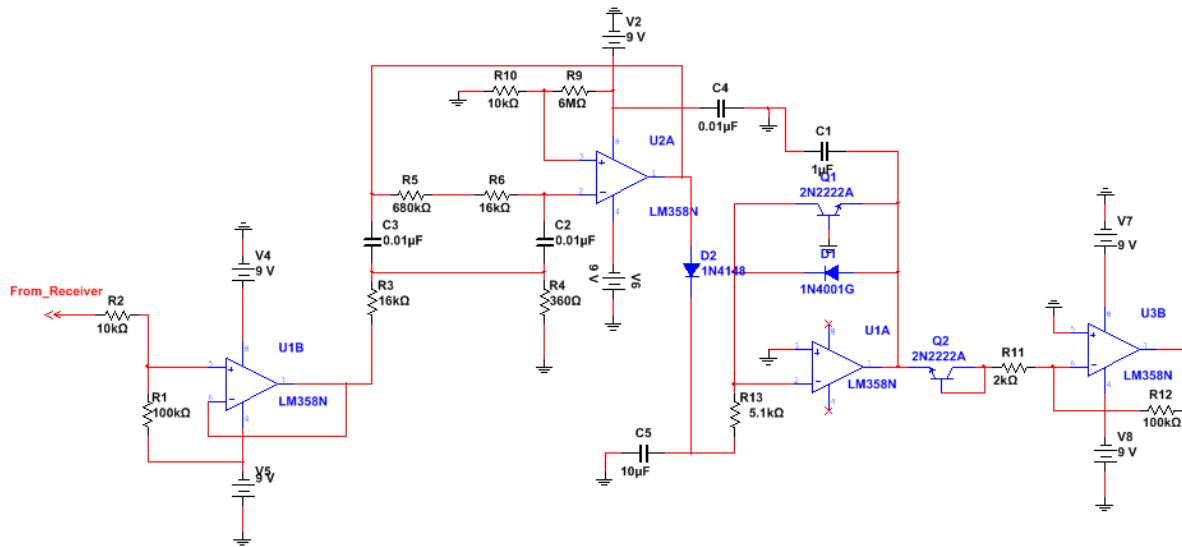
Figure 43 – Post AM Receiver stage

The LM358N Op-Amp denoted by U1B serves as a non-inverting level shifter which offsets the DC bias seen from the output of the receiver. The LM358N Op-Amp denoted by U2A is an active narrow band pass filter of the Deliyannis type with a center frequency at 1 kHz which is the signal being sent by the transmitter denoted by the AC signal source V2 (Carter, 2001). The output of this signal is sent into a logarithmic amplifier and finally to the Atmega328P analog to digital converter.

The respective pinouts of the Atmega Microcontroller of the rescue craft are seen in the Figure 44. As may be seen, the outputs of the logarithmic amplifier circuits are denoted by the text "From_Receiver".
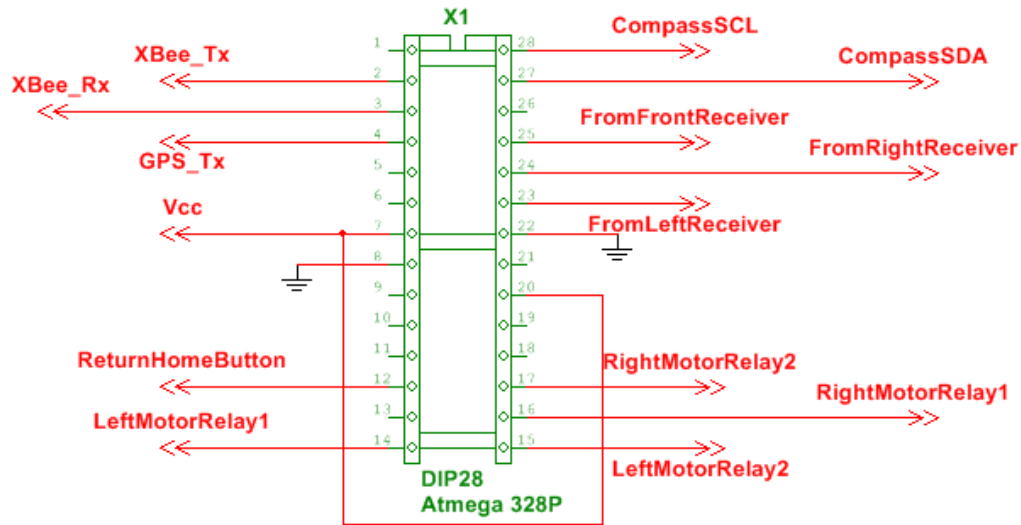
Figure 44 – Rescue Boat Microcontroller Pinout

Deliyannis Bandpass Filter

To decide on the component values for the filter, a quick analysis of the governing equations for the filter are investigated. For this particular configuration, the center frequency is equal to the following:

$$f_0 = \frac{1}{2\pi R_0 C_0}$$

In this equation, the following relations hold true for the circuit seen in Figure 45.

$$R3 = R5 = R_0$$

$$C3 = C2 = C_0$$

Gain and the Q factor are determined with the following equation. For higher values of Q and therefore also gain, the greater the selectivity of the filter will be.

$$Q = \frac{V_{out}}{V_{in}} = \frac{R5 + R6}{2 \times R3}$$

For this equation, note should be taken that R4 and R5 are related by the following relation:

$$n \times R6 = R_0 = \frac{R4}{n}$$

The method applied for solving for component values was an iterative approach which made use of an excel spreadsheet. Here, the values for the capacitors were chosen first and using the selected center frequency, the value of R_0 was solved for. This is equal to R3 and R5 from the relation above. From here, a value for n was chosen and in conjunction with the formula for gain and Q, the respective resistances R6 and R5 could be solved for.

40

In the case of the 1 kHz band pass filter needed for the receiver, a capacitor value of 0.01 µF was chosen for C2 and C3. After some iteration, a value of 44 was chosen for n which resulted in real resistance values. Solving the equations with these inputs, the following values were determined for the band pass filter.

R3 = R5 = 15,915Ω
R6 = 700,282Ω
R4 = 361Ω

For the simulation and in the actual circuit, the values used were rounded to the following:

R3 = R5 = 16 kΩ
R6 = 680 kΩ
R4 = 360 Ω

Through simulation it was found that the resistance that affects the center resistance the most was R4. Any small variation in this value could shift the center frequency drastically which meant that any resistors that were purchased for the filter should have a low error tolerance, especially those used for R4.
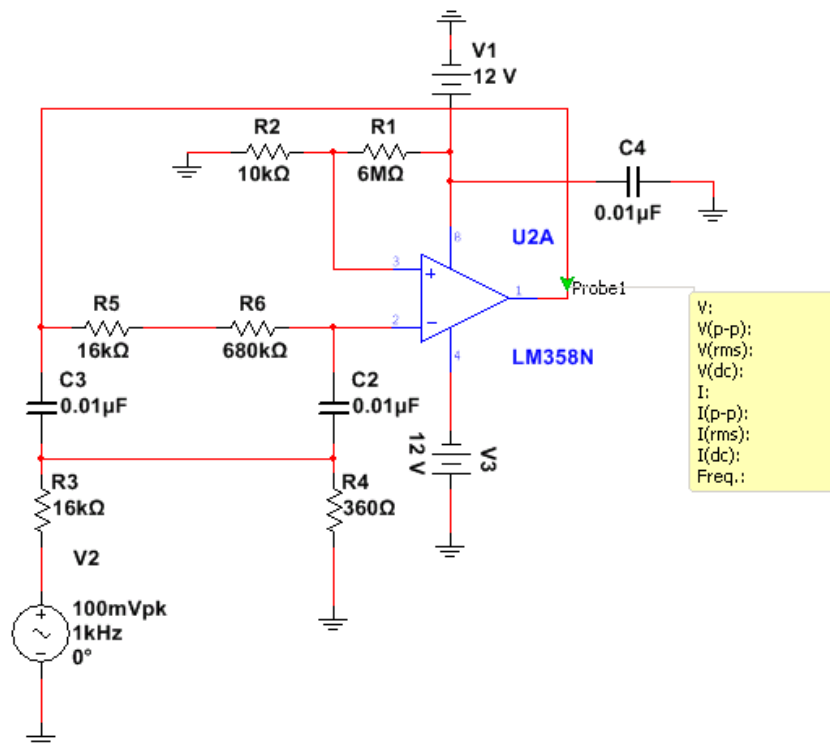
Figure 45 – Deliyannis-Type Band pass filter

Simulation of the filter in Figure 45 may be seen in the Figure 46. Parameters used for the simulation are an input signal of 100mV peak swept from a frequency of 1 Hz to 2 kHz. The peak gain seen by the circuit

is experienced at 1 kHz which is the signal that is being sent by the transmitter. This effectively attenuates all signals above and below this 1 kHz frequency.

AC Sweep Analysis of Bandpass Filter
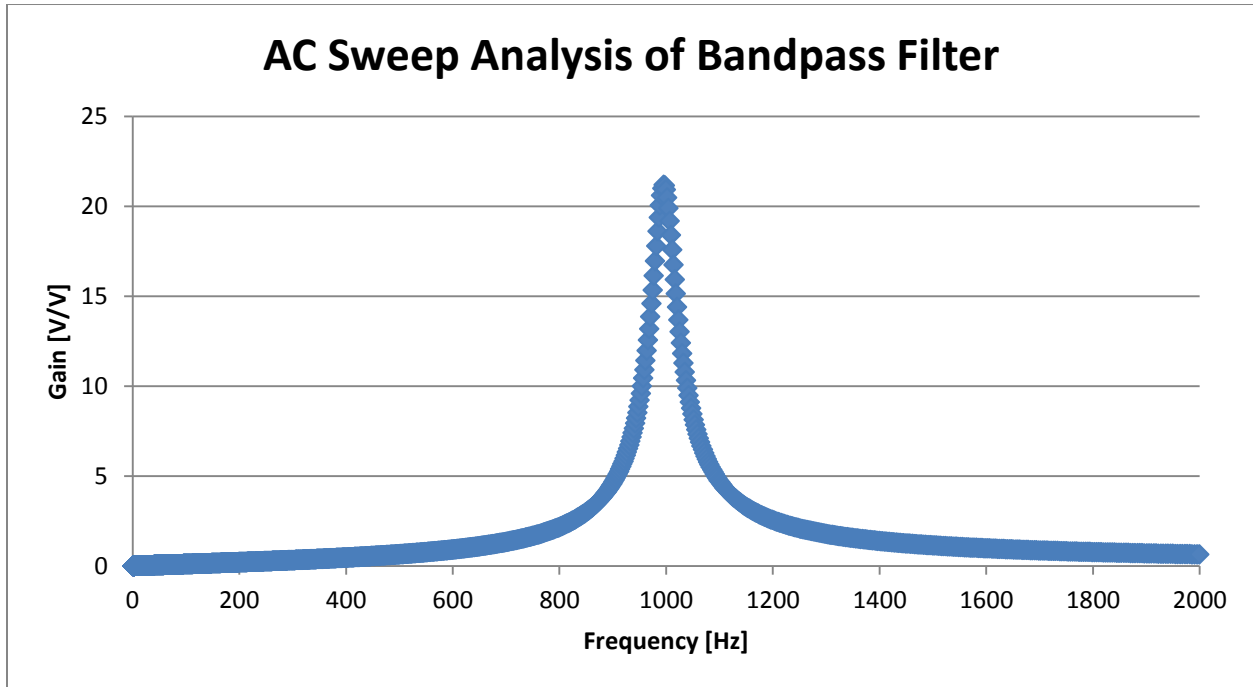
Figure 46 – Band pass filter frequency sweep with 1mV input signal

The topology of radio receiver to band pass filter to logarithmic amplifier is adopted for each of the three receivers whose signals are compared and processed by the Atmega. Using this method, the digitized input can be read and compared in the software to apply signals to the motors for homing in on the victim.

Logarithmic Amplifier

The transdiode-type logarithmic amplifier was necessary to reduce the non-linear characteristic of the received signal from the transmitter versus distance of the receiver to the transmitter. This amplifies low amplitude signals much more than higher amplitude signals resulting in a logarithmic characteristic. By using this amplifier, the small signals received due to greater distances of separation between transmitter and receiver are amplified with high gain factors. As the separation distance is decreased, the gain is reduced as may be seen in the Figure 48. An additional feature of the logarithmic amplifier is that its maximum output signal amplitude is 5V which is the maximum allowed input to the ADC of the Atmega.
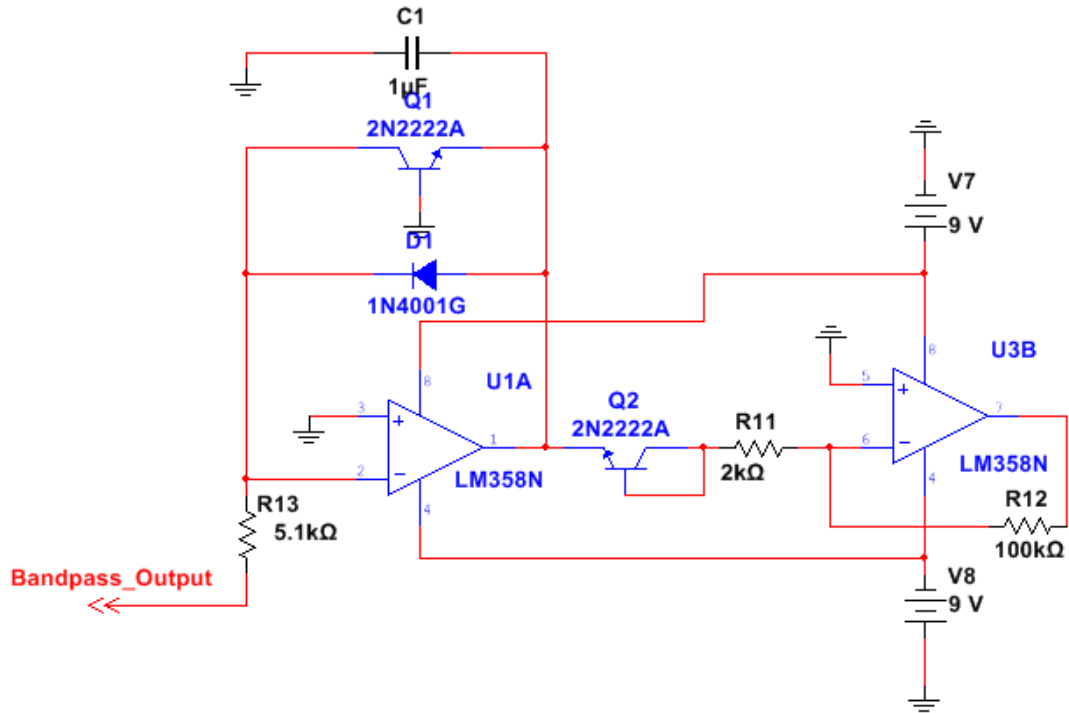
Figure 47 – Logarithmic Amplifier

The Logarithmic Amplifier is used to convert the output of the Bandpass Filter to a 0-5V level with variable gain. Due to the addition of a second BJT transistor, there is the added benefit of Temperature Compensation in the Transdiode Logarithmic Amplifier which compensates for offset voltage (Webster, 1999).

The design of the amplifier was based upon the Figure 16 configuration from the datasheet for the MC1556 operational amplifier (Motorola Semiconductor, 2003-2013). For temperature compensation, the circuit from Figure 6.16 was used from the text *Analog Signal Processing* by Pallas-Areny and Webster on page 299 (Webster, 1999).

To test the Logarithmic amplifier, the setup as seen in Figure 47 was utilized. The voltage source V3 was swept from 0-12V and the output voltage measured where probe 1 is seen in the Figure. The plot of the characteristic is seen in the Figure 48 below.

43
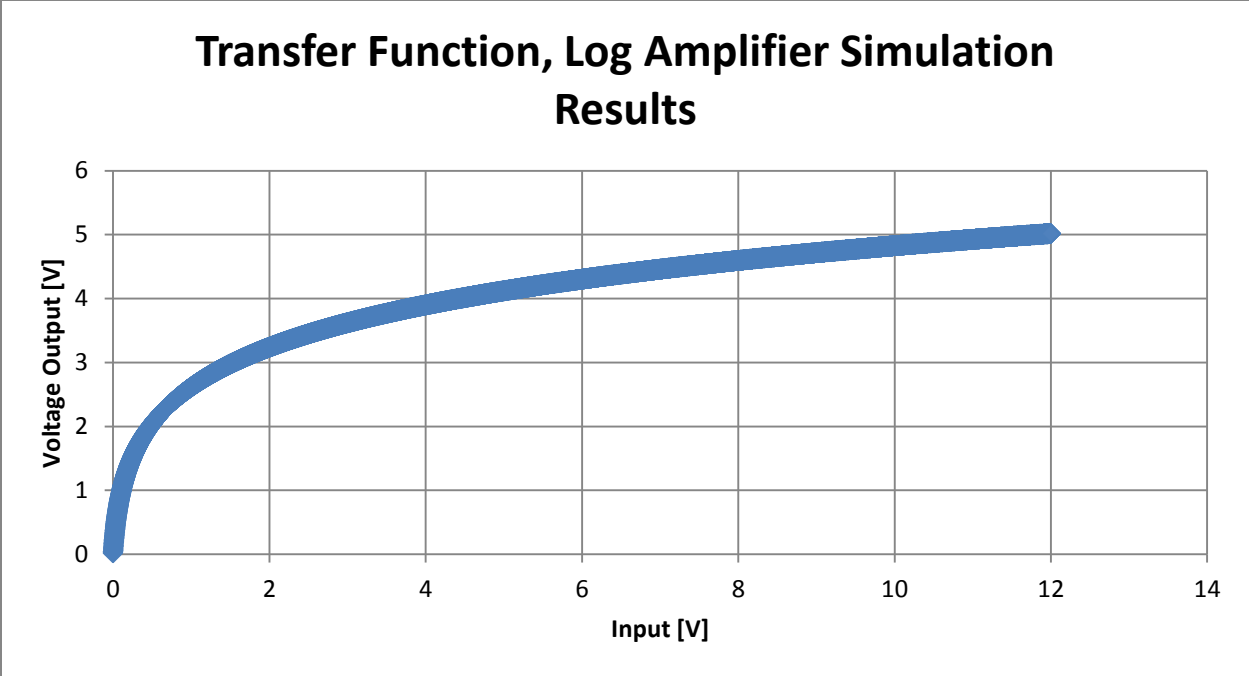
Figure 48 – Logarithmic Amplifier Simulation Results

To compare the actual circuit to the simulated one, the voltage source V3 was adjusted at small increments from 0-10V on a power supply and the output voltage measured where probe 1 is seen in the Figure. The results of this may be seen in the Figure 49 below.
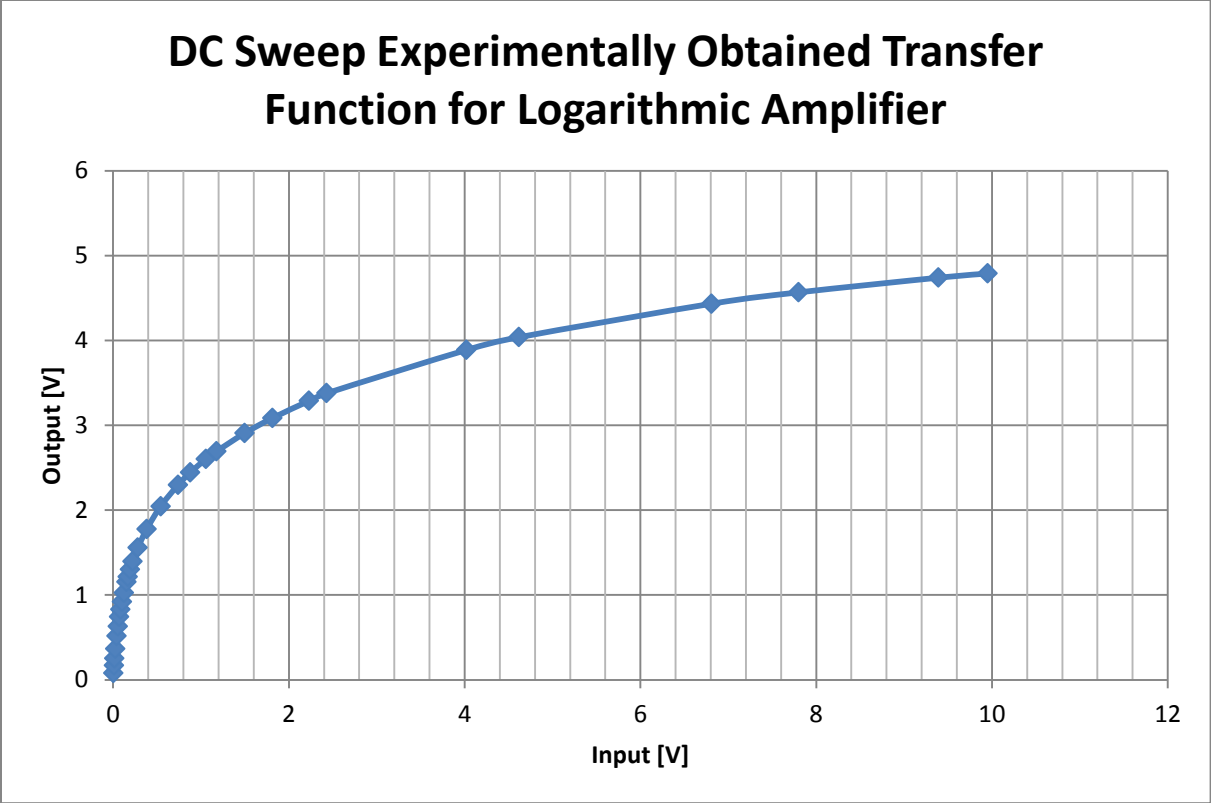
Figure 49 – Transfer Function of Logarithmic Amplifier, Experimentally Obtained

As can be seen, the two plots are nearly identical, demonstrating the circuit's functionality. Any small signal is amplified much more than the larger signals i.e. for every change in input voltage, there is a decreasing change in output voltage. It is the output of this logarithmic amplifier that is read by the ADC of the Atmega.
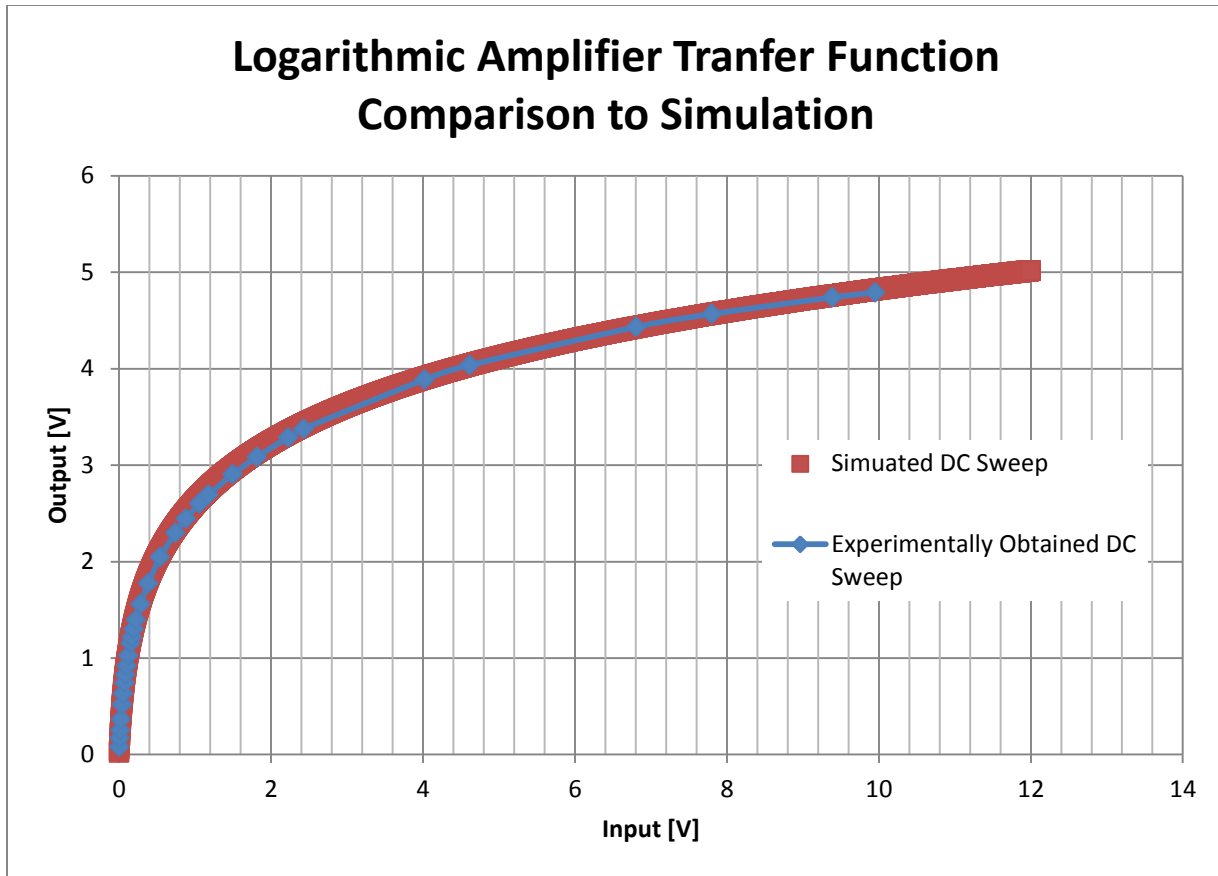
Figure 50 – Comparison of Experimentally Obtained Transfer Function to Simulation

This circuit is utilized for each of the three receiver devices, the outputs of which are compared internally by the Atmega microcontroller. The digitized values of the output signal are used for determining which motor to drive in the final terminal location stage.

Motor Control

Relays were used to drive the trolling motors of the rescue craft. This has been deemed the most efficient and fitting solution as reaction time is not critical and heat dissipation need not be considered due to the nature of relays. In the circuit below four, single pole, dual throw (SPDT) relays are implemented. The open connections to the MOSFETs come from the controlling pins of the Atmega microcontroller which provides logic signals to drive the Relays.
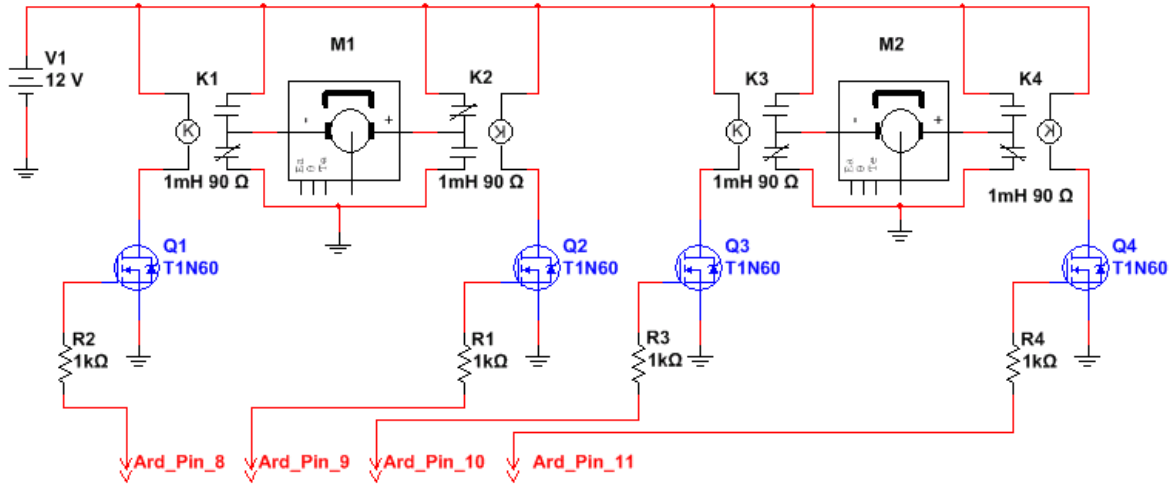
Depending on the state of the input pins, the trolling motors represented by M1 and M2 in the figure above may be driven forwards or in reverse. The table below summarizes what direction the motor will turn, depending on the inputs. MOSFETs Q1 and Q2 together control the state of the motor M1 and Q3 and Q4 the state of motor M2, respectively. For this example, the motor M1 represents the motor on the port side of the boat and M2 the starboard side. In the table, 1 represents a logic HIGH at the MOSFET, or 5V applied to the gate. 0 represents a logic low with 0V applied.

Table 2 – Boat Reaction per MOSFET Gate Input

State of Q1	State of Q2	State of Q3	State of Q4	Motor M1	Motor M2	Boat Reaction
0	0	0	0	OFF	OFF	Coast
0	0	0	1	OFF	FWD	Left Turn
0	0	1	0	OFF	REV	Right Turn
0	0	1	1	OFF	OFF	Coast
0	1	0	0	FWD	OFF	Right Turn
0	1	0	1	FWD	FWD	Forward
0	1	1	0	FWD	REV	Hard Right
0	1	1	1	FWD	OFF	Right Turn
1	0	0	0	REV	OFF	Left Turn
1	0	0	1	REV	FWD	Hard Left
1	0	1	0	REV	REV	Reverse
1	0	1	1	REV	OFF	Left Turn
1	1	0	0	OFF	OFF	Coast
1	1	0	1	OFF	FWD	Left Turn
1	1	1	0	OFF	REV	Right Turn
1	1	1	1	OFF	OFF	Coast

The Rescue Boat

To obtain a basic idea of what speeds and power requirements the rescue boat will need, some basic analysis was done on the craft. For any displacement hull boat, there is a characteristic measure called hull speed which is dependent on the waterline length of the boat which is the dimension B in the figure below.
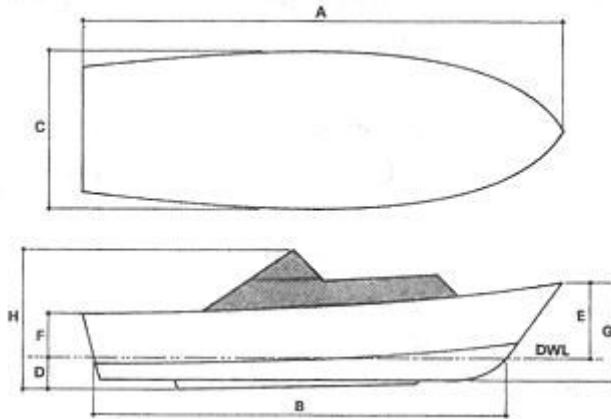
Figure 52 – Waterline length of a boat (glen-I, 2008)

Hull speed is defined as the speed of a displacement hull boat when the wave-making drag becomes dominate over the surface friction drag. The thrust required to exceed hull speed increases dramatically after this point. These may be found in Appendix B – Hull Speed Calculations. The formula for the hull speed may be found below (Savitsky, 2003).

$$v_{hull} = k \times \sqrt{L_{waterline}} \qquad\qquad where \ \ k = 1.34$$

The craft being used to carry out the rescue in this scenario is 10 feet long. While cruising, the craft has an angle of attack that changes with speed. Due to this, the waterline length will decrease. Because of this phenomenon, the hull speed will change. After some calculation over the various waterline lengths up to 10 feet, the resulting hull speeds were graphed in the Figure 53 – Graph of Hull Speed vs. Waterline Length below.

Figure 53 – Graph of Hull Speed vs. Waterline Length

49
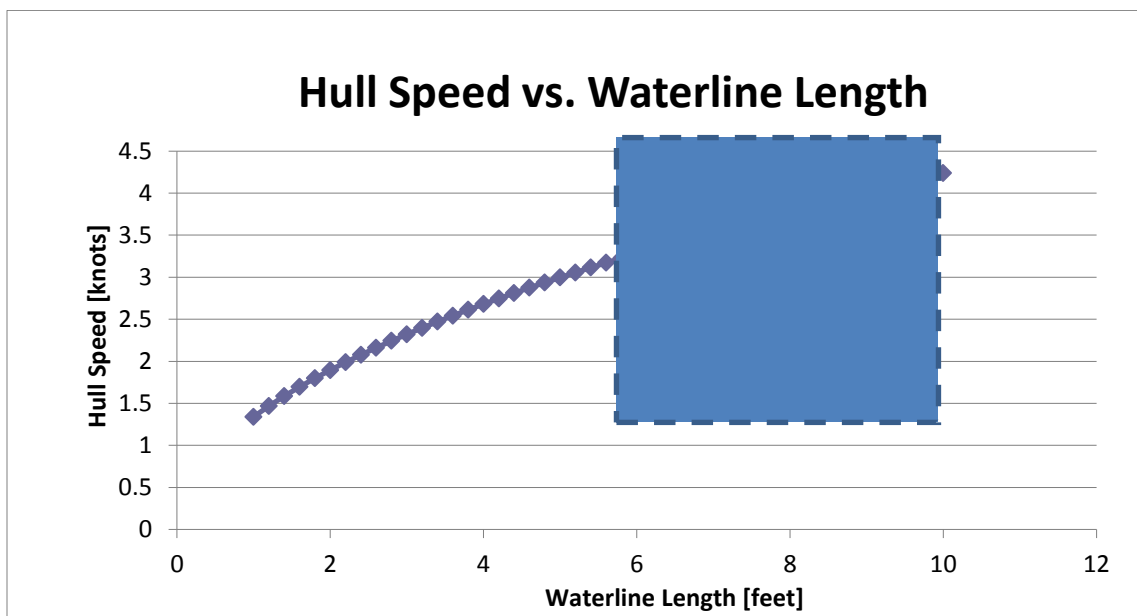
Based on this calculation, our craft will be capable of efficiently running up to 4 knots, provided the waterline length does not drop below 9 feet. Additionally, C_T is a parameter required to find characteristic specifications of the boat's performance. The formula is stated below (Anthony F. Molland, 2011).

$$C_T = 2 \times \frac{R_T}{\rho \cdot S \cdot V_s^2}$$

In the equation above, C_T is the coefficient of hull resistance in calm water; R_T is the total hull resistance in [lbs] and ρ is the fluid density in [slugs/ft^3]. Finally, S is the wetted surface area of the submerged hull in [ft^2] and V_s is the speed of the ship in [ft/s]. For the rescue boat traveling at 1.9 ft/s with 3.9 lbs. of drag, a coefficient of hull resistance was calculated to be 0.036.

Boat Testing

To test the rescue boat, a series of experiments were formulated to determine the handling of the craft itself. These results were crucial in creating a control scheme for the boat such as turning, stopping, and backing up.

Figure 54 – The Rescue Boat

The tests that were performed were to determine the top speed and stopping distance. These were useful in calculating the time of rescue and when the rescue craft will need to stop so as to not hit the victim. The drag force was also measured to determine how the craft can be run to achieve the most efficient traveling velocity. The current draw at maximum velocity was measured to approximate a running time on the boat based on battery characteristics. The turning circle of the boat was measured

to better model the handling. Finally the static thrust was determined. The experimental procedure for the testing may be found in the Appendix A – Boat Water Maneuverability Experimentation.

Test Results

The results of the test are summarized in the table below.

Table 3 – Water Test Results for the Rescue Craft

Parameter	Result	Units
Maximum Speed (With Person in Boat)	2.4	[knots]
Maximum Speed (Without Person in Boat)	3	[knots]
Stopping Distance (From Full Speed Coasting)	9.5	[ft]
Current Draw (Max Speed)	14	[A]
Drag Force (1.6 knot tow speed)	3.9	[lbs]
Static Thrust (Both Motors) With Person	28	[lbs]
Static Thrust (Both Motors) Without Person	25.7	[lbs]
Turning Circle, Radius (with slight Wind)	11.5	[ft]

One major noticeable difference between the motor specifications and the test results is the static thrust of the rescue craft and measured current draw. The trolling motors used for the rescue boat were designed to move heavy craft at slow speeds and therefore have propellers with a small pitch (Burns, 1999-2013). A decrease in pitch on the propeller results in higher motor RPM. This means that the motor reaches its top speed very rapidly and experiences less resistance from the water as the mass of the boat is much less than designed for. As seen from the graph of a typical DC motor speed and current versus torque graph in Figure 55, for higher motor speeds, there is a lower current draw and torque.
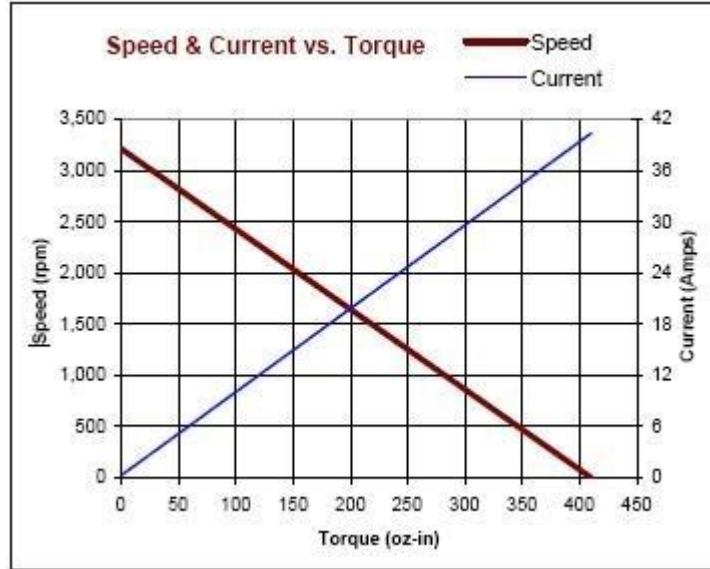
Figure 55 – Speed and Current vs. Torque Graph (Simple Machines, 2010-2011)

To increase the speed and static thrust of the boat, the current propeller should be replaced with one that has a higher pitch.

The Rescue Boat – Draft

To find the draft, or distance from the water level to the bottom of the hull of the boat, it is necessary to calculate several parameters in conjunction with the Archimedes principle (Cimbala, 2010). These are of benefit for the simulation of the rescue craft in the Flow Simulation software of Solidworks. This information is used to determine how deep the craft sits in the water.

To determine the draft of the sea rogue, the total mass of the boat, motors, battery and all other equipment is to be determined. Using this, the following formulas are used to determine how deep the boat sits in the water (arizona.edu, 2010).

$$m = v_{sub} \times \rho_w$$

Where m is the mass of the boat and all equipment on it, v_{sub} is the volume that is submerged and ρ_w the density of water. Solving this equation for the submerged volume and using the following relation allows one to solve for the water level.

$$v_{sub} = v_{disp}$$

Where v_{disp} is the volume of water that is displaced by the laden boat. To determine the water level, the following equation is used.

$$L = \frac{v_{disp}}{A_{hull}}$$

For the boat being used in this project, the following properties were used for the calculations.

52

Table 4 – Calculation Parameters

Parameter	Value	Unit
Dry mass of boat	50	kg
Mass of Battery	16	kg
Mass of Motors	14	kg
Mass of other equipment	5	kg
Density of sea water	1027	Kg/m^3

Utilizing these parameters, the submerged volume of the boat was calculated to be 0.083 m^3. This resulted in a draft of 2.17 cm. Using these values as a basis, a simulation was conducted in the Flow Simulator add on of Solidworks. Drag force was found by solving for a global goal of force in the axis parallel to the direction of travel for the boat.

Solidworks

To provide a base for calculations to determine proper motors for use with the craft and a comparison for expected drag forces from the vessel, a Solidworks model was drafted. This was modeled using the actual rescue boat in shape and size. Using the Flow Simulation 2012 wizard and solving for the reaction force in the Z-Direction as indicated in the model under global goals, the drag force was able to be calculated. To run this model there were some assumptions and conditions that were applied. For a complete procedure of the simulation, see Appendix F – SolidWorks Flow Simulation 2012 Procedure.
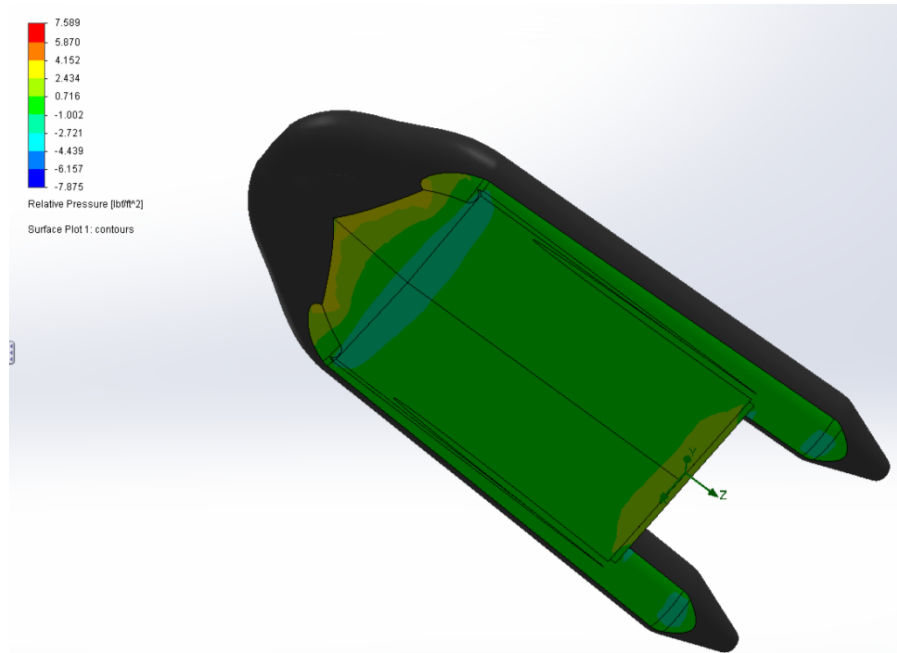
Figure 56 – Solidworks Simulation for Relative Pressure on the Rescue Craft | Drag Force Simulation

Water was used as the fluid and simulated to run at 3kmh toward the boat model. The water level was set as calculated in the draft section above by adjusting the computational domain (Cudacountry.net Tech Ed, 2010). Cavities without flow conditions and internal spaces were excluded from the calculations as well as any components that were not necessary for determining the drag force. Due to the limited availability of another craft to carry out testing, the trolling motors were excluded from the model. This was decided because they were fastened to a canoe which was used to tow the rescue craft for determining the drag force experienced.

When applying the initial conditions, it is important to take care in setting the correct value for flow direction. In this case, making note of the global coordinate system is useful as well as plotting flow trajectories and running an animation to ensure that the fluid is flowing in the correct direction.

After the calculations have been carried out for the global goal(s) as set in the goal table, the resulting drag force is displayed by choosing *Insert, Goal Table.* As may be seen in the Figure 57, the resulting drag force experienced by the rescue boat moving at 3kmh in water was determined to be 2.8 lbs. This was comparable to the experimentally obtained drag force of 3.9 lbs. The percent error of 40% from this experiment showed that the simulation was fairly accurate for a flow simulation. There are other factors that contribute to the drag including wind conditions, turbulence from the towing craft and others that contribute as sources of error.

Goal Name	Unit	Value
GG Force (Z) 1	[lbf]	2.8

To determine the drag force experimentally, a digital luggage scale was used in conjunction with a tow line and canoe upon which the trolling motors were attached. The scale used may be seen in the Figure 58 seen below.

Figure 58 – Luggage Scale used to determine Drag Force on Rescue Craft (Bed Bath & Beyond, 2013)

Other methods for determining drag such as towing the boat from land were considered but determined to be less feasible due to the nature of the lake that testing was completed on.

The Rescue Boat – Power Requirements

To determine the power requirements and therefore specifications for a battery, each of the respective electronic components were necessary to take into account. The following table summarizes the data that was used for the calculations as taken from the provided MinnKota website (Ltd, 2011) and online source (Smith, 2011).

Table 5 – Power Requirement Calculation Parameters

Parameter	Value	Units
Motor Voltage	12	[V]
Max Current Draw	30	[A]
Max Force per Motor	30	[Lbf]
Runtime at Max Power	30	[min]

The necessary battery ampere-hours required may be found using the following equation:

$$C = I_{draw} \times t_{run}$$

Where C is the necessary Ampere-hour capacity needed from the battery, I_{draw} is the total current draw from the battery and t_{run} is the time that the system is operational and requires power from the battery. Additionally, due to the nature of lead acid batteries which should not be discharged more than 50% their capacity, the required capacity will need to be doubled. Solving the equation above and accounting for the allowable discharge, the final capacity required for the motors is 60 Ah for total runtime of 30 minutes at full power.

Based on these calculations, the battery used for the rescue craft was a Die Hard Deep Cycle battery with an 80Ah capacity. This would provide more than the necessary energy required to run the rescue craft as specified.

Figure 59 – Deep Cycle Battery Used to provide Power for the Rescue Boat

After conducting the tests on the water and using a shunt resistor to measure the current draw from a single motor, it was found that less current was drawn than expected. To measure the current draw from one of the trolling motors, a shunt resistor was placed in series with the motor and battery. The battery is represented in the Figure 60 as V1 and the motor represented as a current sink with the resistor R1 representing the shunt resistor, respectively.
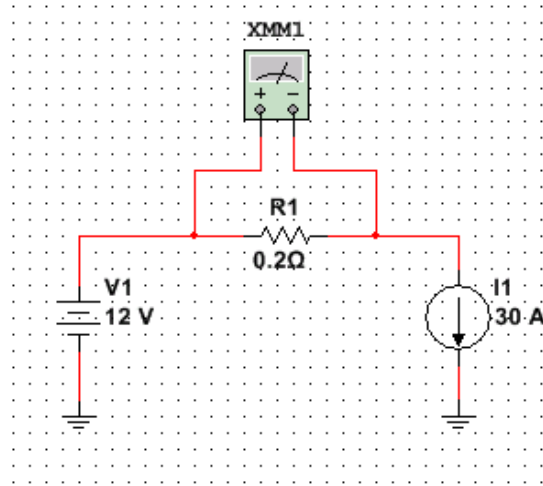
Figure 60 – Setup to measure current draw from single Motor

In measuring the voltage drop across the resistor R1 and knowing the resistance of the shunt resistor, the current draw from the motor may be calculated. This is accomplished by means of ohm's law.

$$I = \frac{V}{R}$$

For this setup, a shunt resistor with a resistance of 0.2 Ω was used and a voltage drop of 2.79V was measured. This resulted in a current draw from the motors of 14A. This is more than half the expected draw as stated by the datasheet from MinnKota. This was determined to be due to the fact that the motors are designed to move heavy craft. By being used to propel our light rescue craft, the motor speed is higher, resulting in a lower torque and therefore a lower current draw.

The Rescue Boat – Turning Circle

To calculate the turning circle of the rescue unit, a few preliminary tests were conducted on the water to obtain values for the angular velocity as well as the static forward thrust. To test the angular velocity, the rescue unit was remotely driven in a circle for a full rotation and timed. This time was then converted to a value of radians per second which was determined to be 0.2244 radians per second. For the turning radius calculation, the motors documented thrust of 30lbs was used. Measurements of the exact position of each motor were also made in order to obtain the most accurate experimental turning radius calculation as possible. Assumptions used in the calculation were that the motion is planar and a constant velocity is maintained throughout the turn.

The equations used to determine the radius can be seen below (Fossen, 1994) (Furfaro, 2012):

$$X_{control} = T_{left.Motor} \times \cos\theta_{left.Motor} + T_{right.Motor} \times \cos\theta_{right.Motor}$$

$$Y_{control} = T_{left.Motor} \times \sin\theta_{left.Motor} + T_{right.Motor} \times \sin\theta_{right.Motor}$$

$$Z_{control} = [T_{left.Motor} \times Dx_{left} \times \sin\theta_{left.Motor} + T_{right.Motor} \times Dx_{right} \times \sin\theta_{right.Motor}]$$
$$+ [T_{left.Motor} \times Dy_{left} \times \cos\theta_{left.Motor} - T_{right.Motor} \times Dy_{right} \times \cos\theta_{right.Motor}]$$

$$\dot{x} = X_{control} \times ratio$$

$$\dot{y} = Y_{control} \times ratio$$

$$R_{turning} = \frac{\sqrt{[\dot{x}^2 + \dot{y}^2]}}{\dot{\varphi}}$$

The figure below shows the respective variables and components of the equations as they relate to the rescue craft.
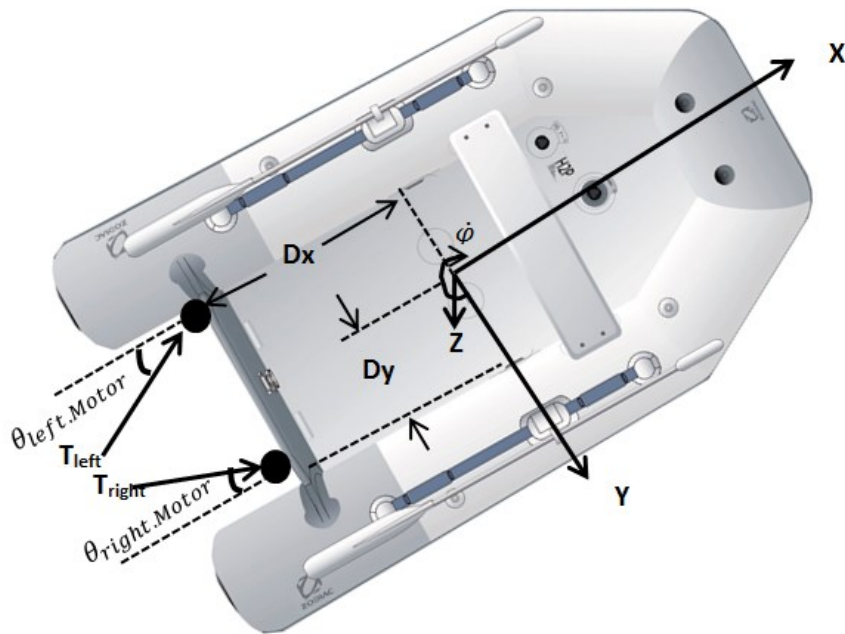
Figure 61 – Model of the Rescue craft for analysis

Additionally, the meaning of each of the symbols may be seen in the table below. The equations used the thrust of each motor as well as the angle that the respective motor was turned at to determine the force in the x and y direction of travel. These values were then converted using a ratio experimentally obtained between the force and the velocity of the boat to determine the speed the boat would be moving in the x and y direction while turning. Finally the velocity vector was divided by the angular velocity to determine the experimentally calculated expected turning radius. This was found to be approximately four meters.

Symbol	Description
$X_{control}$	Surge force control
$Y_{control}$	Sway force control
$Z_{control}$	Yaw control
$T_{x.Motor}$	Torque from motor x
$\theta_{x.Motor}$	Angle of applied torque from x-axis
$Dx_{\square}$	Displacement of motor from the center of gravity for the craft in the x-axis
$Dy_{\square}$	Displacement of motor from the center of gravity for the craft in the y-axis
$\dot{x}$	Linear velocity along the x-axis – surge velocity
$\dot{y}$	Linear velocity along the y-axis – sway velocity
$ratio$	Experimentally obtained relation between expected thrust to experimentally obtained velocity
$R_{turning}$	Calculated Turning radius
$\dot{\varphi}$	Yaw velocity

Results

With a completed system created, several experiments were conducted to demonstrate the functionality of the MOB equipment. These involved a water test which simulated the rescue of a MOB on the Assabet River in Maynard, MA as well as a land demonstration of the personal locator devices' terminal location. The tests were headquartered at the Ice House Landing near the Maynard Department of Public Works.

Water Test

The main test for functionality involved a simulated autonomous rescue effort by the rescue craft. Due to the cold water, the test was broken into two segments as a victim was not readily willing to swim in the lake to be rescued. The segments included victim location and return to mothership. This was segmented so as the "victim" could be placed in the rescue craft to press the return home button. The following series of figures show the path traveled by the three modules as seen on the mothership GUI as well as from an observer's perspective on the lake. A log file of the GPS coordinates can be found in Appendix N - Coordinates.log.

Victim Location

Victim location was achieved by starting the rescue craft 226m downstream from the personal locator device as measured from GPS log data. The personal locator device was then activated by pressing the distress button on the unit which was attached to a life vest as seen in the Figure 62.

Figure 62 – Personal Locator Device Fastened to the Life Vest

From the GUIs perspective, the following information is displayed in Figure 63. When the victim module sends a distress signal, the GUI displays a message indicating that the victim overboard condition has been detected. It also displays the position of any of the modules that have sent their respective GPS coordinates to the mothership. The test showed that by pressing the button on the victim module, the mothership successfully initialized the rescue unit. The rescue unit then began to navigate towards the victim using the cross-track error algorithm.
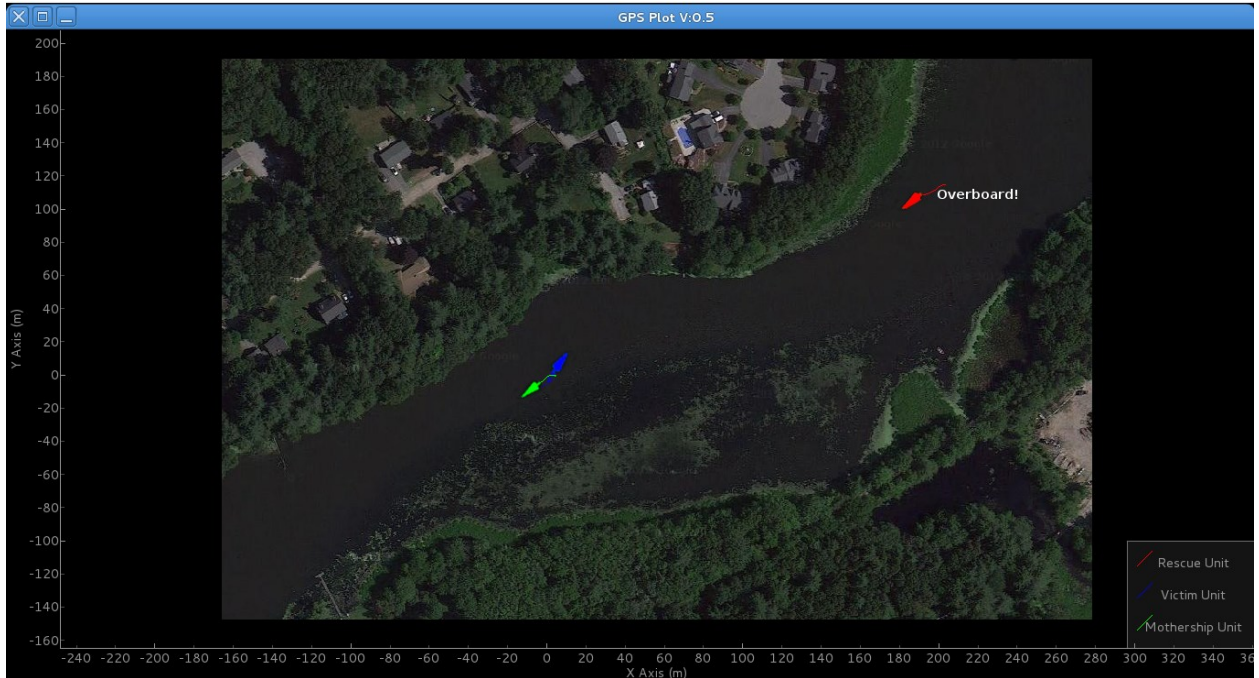
Figure 63 – Mothership GUI: Victim overboard condition detected

The next message the mothership GUI displays is that the rescue unit is initialized and navigating to the victim. This can be seen in Figure 64. In this stage the rescue craft successfully used the cross-track error algorithm to head in the direction of the victim using the GPS coordinates it received from the victim unit.
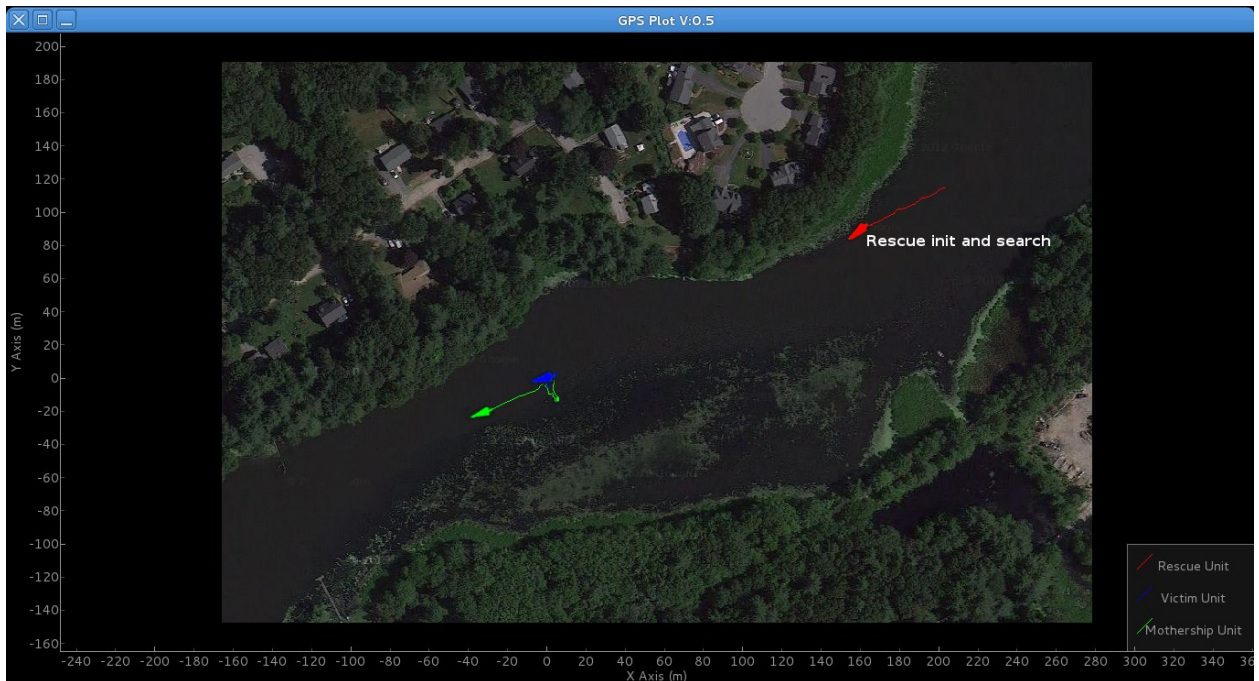
Figure 64 – Mothership GUI: Rescue unit initialized and heading off

In Figure 65 below, the rescue craft - denoted by the red outline, and victim - denoted by the blue outline may be seen. Here, the rescue craft is headed in the direction of the personal locator device.
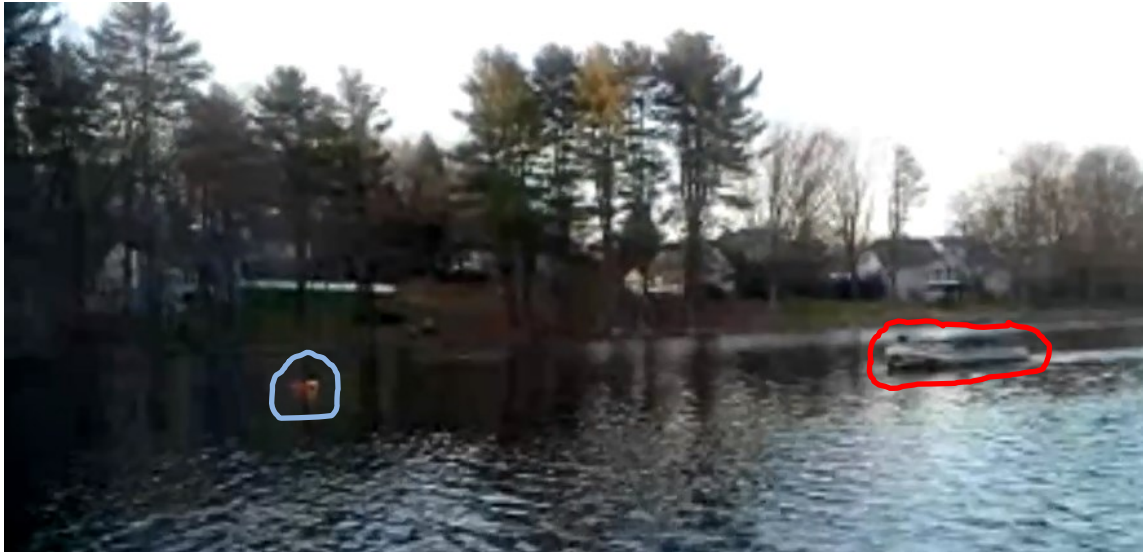
Figure 65 – Rescue Craft and Personal Locator Device

Additionally, as the rescue craft was within 30 meters of the PLD, the radio transmitter was activated and terminal location was standing by for a signal. Figure 66 shows the GUI telling the user that the victim unit has activated the terminal locator device and the rescue craft is searching for a signal from it. In this state, the rescue unit is still using the cross-track error algorithm to navigate to the victim, but is about to transition to the terminal location algorithm using the three AM receivers to home in on the victim.

Figure 66 – Mothership GUI: Terminal locator device activated

In this particular test scenario, the rescue craft passed the personal locator device and homed in on the simulated victim. This homing action may be seen in the Figure 67 below.

Figure 67 – Terminal Location on the Victim

63

In the figure below, the GUI is notifying the user that the rescue unit has successfully reached the victim within one meter. Here the rescue craft is in a state of idleness, waiting for the victim to board the boat. Should the victim drift away, the rescue unit will again home in on the man overboard.

Figure 68 – Mothership GUI: Waiting for the victim to board

In the next state, the rescue craft waits for the victim to press the return to home button located near the bow of the boat. Once this button is pressed, the GUI on the mothership displays a message indicating that the victim is on the rescue unit as demonstrated in Figure 69.

Figure 69 – Mothership GUI: Victim is on the rescue craft

Return to Mothership

For this stage of the test, a canoe was paddled to the rescue craft and a person entered the boat to press the return home button as indicated in Figure 70.

Figure 70 – Return Home Button

Once the return home button was pressed, the recue craft headed toward the GPS coordinates of the simulated mothership 133m away. For this test, the mothership was a canoe denoted by the green outline in the Figure 71.
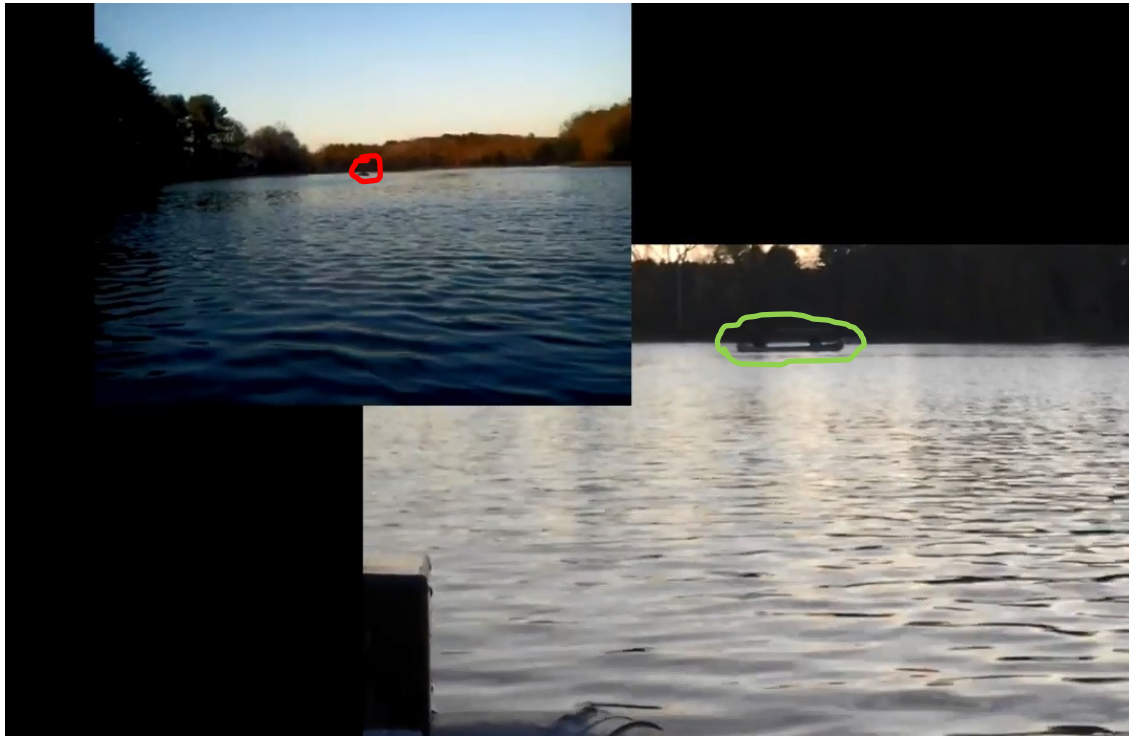
Figure 71 – Rescue Boat Return Home Trip

The output on the GUI is seen in Figure 72 for the condition after the return home button has been pressed.
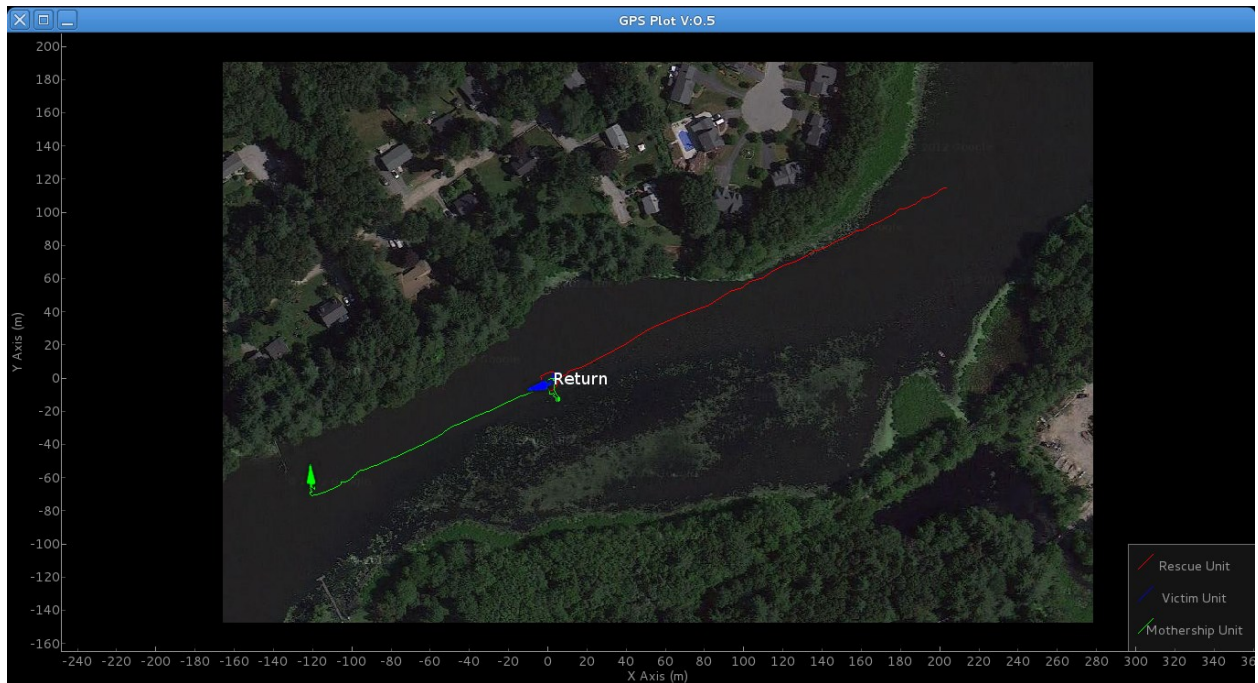
Figure 72 – Mothership GUI: The rescue craft is returning home

Once the victim and rescue unit are within thirty meters of the mothership, the GUI displays a message indicating the victim has returned home as seen in Figure 73. The test demonstrated that the rescue unit was capable of using the GPS coordinates to return to the mothership and realize when to stop once it was within thirty meters of it. With the mothership successfully reunited with the rescue craft and victim, the test was completed demonstrated that all three modules worked together properly to autonomously return the victim to the mothership.
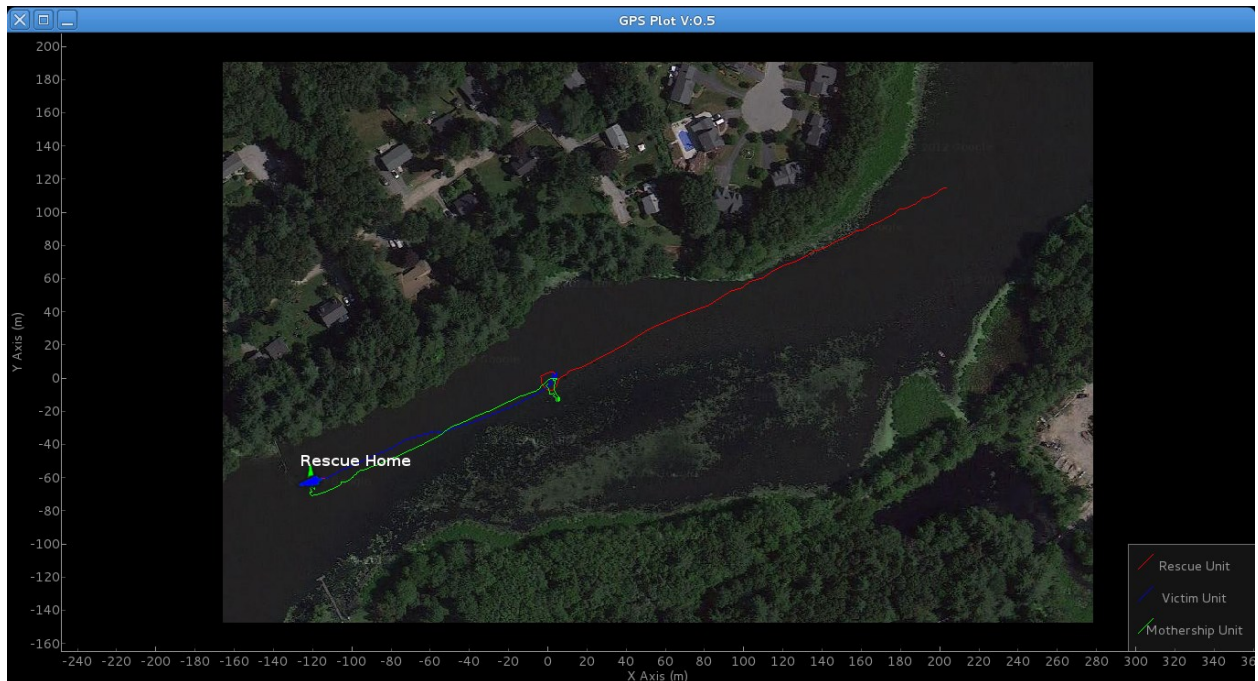
Figure 73 – Mothership GUI: The rescue craft is back at the mothership

Terminal Location – Land-based Demonstration

Before the Terminal Locator device could be tested, the receivers must be fine-tuned and volume levels from each of the devices adjusted such that proper functionality is achieved. See Appendix K – TLD Radio Fine-Tuning for TLD tuning.

In order to test the operation of the receivers, the victim module was held in front of the rescue craft on land and moved from side to side. This test was conducted to determine the distance of operation for the terminal locator device as well as its functionality. First the rescue unit was put in the GPS navigation mode and the victim module was slowly moved towards the boat beginning at a distance of fifty feet measured from the bow of the boat. Once the victim unit was within eighteen feet of the bow of the boat, the terminal location took over as seen in Figure 74. When the victim unit with the AM transmitter was moved back to twenty three feet from the bow of the boat, the rescue unit automatically reverted back to using the GPS navigation. This was repeated four times and accurately resulted in similar results.

Once within the terminal location mode, the victim unit was moved to the left and to the right of the rescue boat. The motors changed direction with an error of just three feet either to the left or right of the boat. This demonstrated functionality of the terminal location in being able to home in on the victim using the AM transmitter and receivers. Finally, to determine if the rescue unit would stop before hitting the victim, the PLD was moved closer to the rescue boat. Once it was within five feet of the bow of the boat, the rescue unit motors turned off and the rescue unit indicated that it was waiting for the victim to board.

68
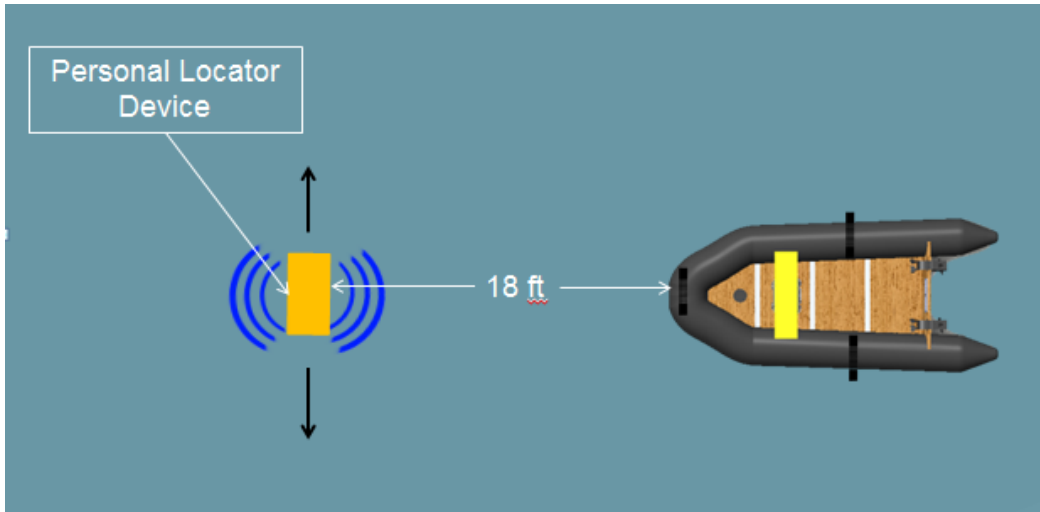
Figure 74 – Experimental Setup for Terminal Locator Test

A frame from the video recorded of the demonstration seen in Figure 75 shows that, when the personal locator device is positioned to the port of the rescue boat, the right motor is running and the vessel would be making a port turn.

Figure 75 – Terminal Locator Demonstration – Port Turn

As the personal locator device is moved to the starboard side, the left motor is turned on and the right motor turned off. This makes for a starboard turn to head toward the victim as seen in the Figure 76.

69

Figure 76 – Terminal Locator Demonstration – Starboard Turn

When the personal locator device is within 5 feet of the personal locator device, the motors are turned off to coast toward the victim. This is seen in the frame in Figure 77 below.

Figure 77 – Terminal Locator Demonstration – Coasting

The results of this test demonstrated that the terminal location by means of AM radio transmission and the tri-antenna setup is a feasible method for homing on the victim.

Conclusions and Further Research

The following sections describe the conclusions reached for the autonomous man overboard rescue equipment as well as suggested future research.

Conclusions

It was determined that the autonomous man overboard rescue equipment was successful in demonstrating an autonomous search and rescue mission for a man overboard case. The test was conducted on the Assabet River in Maynard, Mass where the rescue craft was located on one end of the river and the victim on the other, separated by a distance of 226 meters. The mothership was also located a little farther away from the two modules. When the victim unit was activated, the rescue unit successfully used GPS and the magnetic compass readings to head directly to the victim and then stop upon reaching it. Once the victim was on the rescue craft, the craft was able to head back to the mothership unit using the GPS coordinates received from the mothership unit. All functional requirements of the initial design were met.

The system was also able to operate well over the thirty minute time requirement. Additionally, it operated over a 500 meter range using the XBee modules although the full range of the XBees listed in the specifications was not achieved due to the enclosed location of the XBee transceiver on the rescue craft. The victim module was watertight and could be submerged without harm to the internal electronics. The mothership GUI also performed well, displaying the locations of all three modules relative to each other as well as notifying the user of the current state of the rescue. Overall the system performed as expected and demonstrated that the scaled up model could be used to autonomously find, rescue, and return a man overboard to the mothership.

Compromises

One of the compromises that was made to successfully complete the project was the use of a commercial AM receiver was used to replace a tuned radio receiver. In the original project design specifications, the receivers were to be designed and implemented specifically for the 921 kHz operating frequency by the project team. An initial receiver circuit was designed and underwent some revisions, but did not meet the sensitivity and selectivity requirements necessary for the performance at the distances required for the project. Due to time constraints caused by the design and troubleshooting of the transmitter and other circuits related to the receiver module (such as the band pass filter and the logarithmic amplifier), it was decided that the receiver circuit should be replaced with an off-the-shelf alternative. This allowed for more time to be allocated to other portions of the project.

A second compromise that was made for the project was the use of AAA batteries in the victim module. This was due to the limited available space in the enclosure for the victim module. Because of this, the battery life of the victim module was reduced with the AM transmitter active. Better batteries with a higher amp-hour rating would allow the victim module to operate longer.

A final compromise was that a PCB was not printed for the modules. With a PCB, the modules would be more compact, fit in a smaller space, and reduce the number of external wires used. Furthermore, the modules would have a more professional look and would not be prone to short circuits.

Next Steps

Further development for this project should be focused on obtaining a more powerful motor for the rescue unit. With a more powerful motor like an 8hp outboard motor, the rescue time for the victim could be significantly reduced. For example, the current trolling motors on the rescue craft require approximately half an hour to go 2.5km. If the motors were to be replaced with an 8 hp motor, then the rescue time would be reduced down to approximately ten minutes. Furthermore, a more powerful transmitter should be designed to increase the range for a better homing capability. More powerful transceivers to replace the XBees would also allow for a longer range of communication between the modules. A more modular design for the personal locator device would also allow for a smaller sized box for the victim to have to carry around. A deployment system for the rescue unit could also be helpful in making the entire process autonomous as the current design requires the boat to be placed in the water by someone.

The code could also be optimized for more advanced tracking allowing for less delay. Also the receivers could be tuned better for noise reduction capabilities and hopefully increase the range for the homing on the victim. With all these modifications added to the system, the autonomous man overboard rescue equipment would be much more effective and a reliable solution for the field.

Outlook

Based on the functionality of the MOB rescue equipment described in this report, a potential solution to the MOB problem that would surpass exiting systems has been demonstrated to be feasible. The successful PLD developed in this project may be used to create a more modular and higher performance device to be commercialized and mass produced and carried by the crew of a large vessel. Similarly, the rescue module may be made more robust and incorporate an outboard motor to overcome strong winds and currents experienced in the open ocean. The GUI may be designed to be incorporated on the control deck of a vessel and provide both audible and visual indicators of MOB cases as well as mission status updates. Finally, to make the system more autonomous, the distress button on the PLD should be supplemented by a salt water detector. The combination of these systems would result in a viable, commercial system that could save many lives.

Bibliography

What is the "MOB" function of my marine GPS device? (2011, March 28). Retrieved 12 10, 12, from
      Garmin: https://support.garmin.com/support/searchSupport/case.faces?caseId={71655ee0-
      104c-11dc-4b06-000000000000}

Anthony F. Molland, S. R. (2011). *Ship Resistance and Propulsion, Practical Estimation of Ship Propulsive*
      Power. New York, NY: Cambridge University Press.

arizona.edu. (2010, 6 7). *Archimedes*. Retrieved 2 2013, from physics.arizona.edu:
      http://www.physics.arizona.edu/physics/gdresources/documents/13_Archimedes.pdf

automationdirect. (1999-2013). *GCX3226-24*. Retrieved March 2013, from automationdirect.com:
      http://www.automationdirect.com/adc/Shopping/Catalog/Pushbuttons_-z-_Switches_-z-
      _Indicators/22mm_Plastic/Emergency_Stop_Pushbuttons_Illuminated_-a-_Non-
      Illuminated/GCX3226-24

Bed Bath & Beyond. (2013). *Digital Luggage Scale* . Retrieved March 2013, from bedbathandbeyond:
      http://www.bedbathandbeyond.com/product.asp?SKU=14704795

Burns, T. (1999-2013). *The Effect of Propeller Pitch on Outboard Motors*. Retrieved April 20, 2013, from
      ehow.com: http://www.ehow.com/info_8584258_effect-propeller-pitch-outboard-motors.html

Carter, B. (2001, December). *More Filter Design*. Retrieved February 2013, from Texas Instruments:
      http://www.ti.com/lit/an/sloa096/sloa096.pdf

Cimbala, Y. A. (2010). *Fluid Mechanics, Fundamentals and Applications.* New York, NY: McGraw-Hill.

Connolly, J. P. (2004, January 28). *Man Overboard Retrieval Device*. Retrieved 11 15, 12, from Google
      Patents :
      http://www.google.com/patents?id=QQMUAAAAEBAJ&printsec=abstract&zoom=4#v=onepage
      &q&f=false

Cudacountry.net Tech Ed. (2010, October 7). *Boat Flow Sim.* Retrieved February 2013, from
      Cudacountry.net: http://www.cudacountry.net/assets/applets/sw10_FlowSim_boat.pdf

Digi-Key. (12, 12 10). *ATS073B CTS-Frequency Controls*. Retrieved 12 10, 12, from Digi-Key:
      http://www.digikey.com/product-search/en?x=0&y=0&lang=en&site=us&KeyWords=CTX1040

Digi-Key. (1995-2013). *AOT1N60*. Retrieved April 23, 2013, from digikey.com:
      http://www.digikey.com/product-detail/en/AOT1N60/785-1184-5-ND/2353849

Environmental Law Institute. (1991). *Oil Pollution Deskbook, The Environmental Law Reporter.*
      Washington, DC: Environmental Law Institute.

FCC. (2011, November 3). *AM Query Broadcast Station Search*. Retrieved December 12, 12, from FCC:
      http://www.fcc.gov/encyclopedia/am-query-broadcast-station-search

Fossen, T. I. (1994). *Guidance and Control of Ocean Vehicles.* West Sussex, England: John Wiley & Sons Ltd.

Furfaro, T. C. (2012). *A Modular Guidance, Navigation and Control System.* Boca Raton: Florida Atlantic University.

glen-l. (2008, May 29). *Boat Design Characteristics*. Retrieved December 14, 2012, from glen-l.com: http://www.glen-l.com/desn-char.html

Google. (2011). *My Tracks*. Retrieved February 2013, from google.com: http://www.google.com/mobile/mytracks/

ladyada. (2012, August 14). *GPS datalogging shield for Arduino*. Retrieved February 2012, from ladyada.net: http://ladyada.net/make/gpsshield/modules.html

Ltd, B. L. (2011). *MinnKota Endura Manual*. Retrieved March 2013, from MinnKota: http://www.minnkota.com.au/community/user-manuals/motor-manuals/endura.html

Makerbot Industries, LLC. (2013). *Adafruit Pi Box: Enclosure for Raspberry Pi*. Retrieved April 2013, from thingiverse.com: http://www.thingiverse.com/thing:24461

Marine Rescue. (2012, December 1). *Rhotheta High-Precision Man Overboard Direction Finder Systems*. Retrieved 12 2, 12, from Marine Rescue: http://www.manoverboardsystems.com/Rhotheta-SAR-receivers.html

Maxim Incorperated. (2007, October 31). *What are IP Ratings?* Retrieved 12 05, 12, from Maxim Integrated: http://www.maximintegrated.com/app-notes/index.mvp/id/4126

MinnKota. (2008-2013). *Endura C2*. Retrieved April 2013, from minnkotamotors.com: http://www.minnkotamotors.com/products/trolling_motors/freshwater_transom_mount/enduraC2.aspx

Motorola Semiconductor. (2003-2013). *MC1556 pdf.* Retrieved March 2013, from alldatasheet.com: http://pdf1.alldatasheet.com/datasheet-pdf/view/128952/MOTOROLA/MC1556.html

Mouser. (2013). *893-896H1CHD1SW12VDC*. Retrieved April 2013, from Mouser.com: http://www.mouser.com/ProductDetail/Song-Chuan/896H-1CH-D1SW-R1-12VDC/?qs=sGAEpiMZZMt98bArVJter4JSFvwfmI4SJSjElpVYzv0%3d

Mouser. (2013). *ATMEGA328P-PU*. Retrieved January 2013, from mouser.com: http://www.mouser.com/ProductDetail/Atmel/ATMEGA328P-PU/?qs=sGAEpiMZZMtVoztFdqDXO6rEZqxeooRg

Mouser. (2013). *Digi International XBee*. Retrieved March 2013, from mouser.com: http://www.mouser.com/new/digi/digiXBeeZB/

Mouser. (2013). *RP8100B2M1CEBLKBLUBLU*. Retrieved March 2013, from mouser.com:
    http://www.mouser.com/ProductDetail/E-
    Switch/RP8100B2M1CEBLKBLUBLU/?qs=sGAEpiMZZMvxtGF7dlGNpqqmTFthXBmfiFV%252bWV9
    Ee7w%3d

National Meteorological Library and Archive . (2010). *Beaufort.* Retrieved December 14, 2012, from
    metoffice: http://www.metoffice.gov.uk/media/pdf/4/4/Fact_Sheet_No._6_-
    _Beaufort_Scale.pdf

Nautic Expo. (12, December 5). *MOB davit for ships*. Retrieved 12 06, 12, from Nautic Expo:
    http://www.nauticexpo.com/prod/vestdavit/mob-boat-davits-for-ship-31773-195721.html

NPD Group. (2013, February). *Otterbox 2000 Waterproof iPhone Case*. Retrieved March 2013, from
    otterbox.xom: http://www.otterbox.com/OtterBox-Drybox-2000/OTR3-
    2000S,default,pd.html?dwvar_OTR3-2000S_color=05&start=2&cgid=otterbox-2000-cases

Premier Farnell plc. (2009). *Group: Raspberry Pi*. Retrieved January 2013, from element14.com:
    http://www.element14.com/community/groups/raspberry-pi

RadioShack Corporation. (2013). *Eton Grundig Mini 400*. Retrieved April 2013, from radioshack.com:
    http://www.radioshack.com/product/index.jsp?productId=4394908

Rescue1Tech. (2012, November 28). *Fibre-Lite Rescue Cradle*. Retrieved 11 30, 12, from Rescue1Tech:
    http://www.rescuetech1.com/fibre-literescuecradle.aspx

Savitsky, D. (2003, October 2). *On the Subject of High-Speed Monohulls*. Retrieved 11 20, 12, from
    Legacy: http://legacy.sname.org/newsletter/Savitskyreport.pdf

Simple Machines. (2010-2011). *Speed and Current vs. Torque*. Retrieved April 23, 2013, from
    societyofrobots.com: http://www.societyofrobots.com/robotforum/index.php?topic=4324.0

Smith, J. P. (2011, August). *Trolling Motor Performance*. Retrieved March 2013, from tufox.com:
    http://tufox.com/hobie/TrollingPerformance.html

Sparkfun. (2013). *Compass Module - HMC6352*. Retrieved February 2013, from sparkfun.com:
    https://www.sparkfun.com/products/7915

Texas Instruments. (2013, April). *LM1577 / LM2577.* Retrieved April 2013, from ti.com:
    http://www.ti.com/lit/ds/symlink/lm2577.pdf

Tor Pinney. (2012, December 1). *Attention Cruising Sailors: This Could Save Your Life!* Retrieved 12 12,
    12, from Tor Pinney's Homeport: http://www.tor.cc/articles/last.htm

United States Coast Guard. (2012, December 6). *HC-130H/J Long Range Surveillance Aircraft*. Retrieved
    12 11, 12, from United States Coast Guard: http://www.uscg.mil/acquisition/lrs/

United States Coast Guard. (2012, December 6). *Ring Life Buoys & Buoyant Cushions*. Retrieved 11 12, 12, from United States Coast Guard: http://www.uscg.mil/hq/cg5/cg5214/ringlb.asp

United States Coast Guard. (2012, February 6). *U.S. Coast Guard SAR Statistics*. Retrieved December 12, 12, from United States Coast Guard: http://www.uscg.mil/hq/cg5/cg534/sarfactsinfo/USCG_SAR_Stats.asp

Wal-Mart Stores, Inc. (2013). *Snap Top Everstart Marnine Battery Box*. Retrieved April 2013, from walmart.com: http://www.walmart.com/ip/Snap-Top-Everstart-Marine-Battery-Box/16781380

Webster, R. P.-A. (1999). *Analog Signal Processing.* New York: John Wiley & Sons, Inc.

Williams, M. C. (2007, November 9). *Man Overboard Device Saves Sailors Lives*. Retrieved 12 01, 12, from Americas Navy: http://www.navy.mil/submit/display.asp?story_id=33180

wiseGeek. (12, 12 06). *How Long Does it Take a Supertanker to Stop?* Retrieved 12 10, 12, from wiseGeek: http://www.wisegeek.com/how-long-does-it-take-a-supertanker-to-stop.htm

Appendices

This section contains applicable material referenced in the report above.

Appendix A – Boat Water Maneuverability Experimentation

Goals

- Determine the turning characteristics
  - One driving motor
- Drag force
  - With/without victim
- Craft top speed
  - With / Without person
- Static thrust

Materials

- The Rescue Craft

- 12 Volt Batteries
- Two trolling motors
- High Current Switches
- Multimeter
- GPS
- Luggage Scale

- Rope
- Camera
- Timer
- Canoe Paddles
- Buoy (With Anchoring)

Experimental Setup

The rescue craft was set up with a battery and control box near the bow of the boat with software running allowing for remote control access to the motors. The motors were attached at the stern of the boat at various angles to test for the best maneuverability.

Procedure

Each of the following tests was conducted three times to ensure comparable results. If there was any deviation in the results of the three runs, additional tests were conducted in an attempt obtain more consistent data.

Water Test

The water test was conducted to roughly determine the maneuverability of the boat. The test was used to determine the best means of propelling the boat in anticipation of turning and maximumspeed capabilities.

The distance between the two mounted motors was 23.5 inches. While the motors were angled towards each other as seen in Figure 78, the best maneuverability was observed. The mass of the boat with all the electronics in it was measured to be 161 lbs.
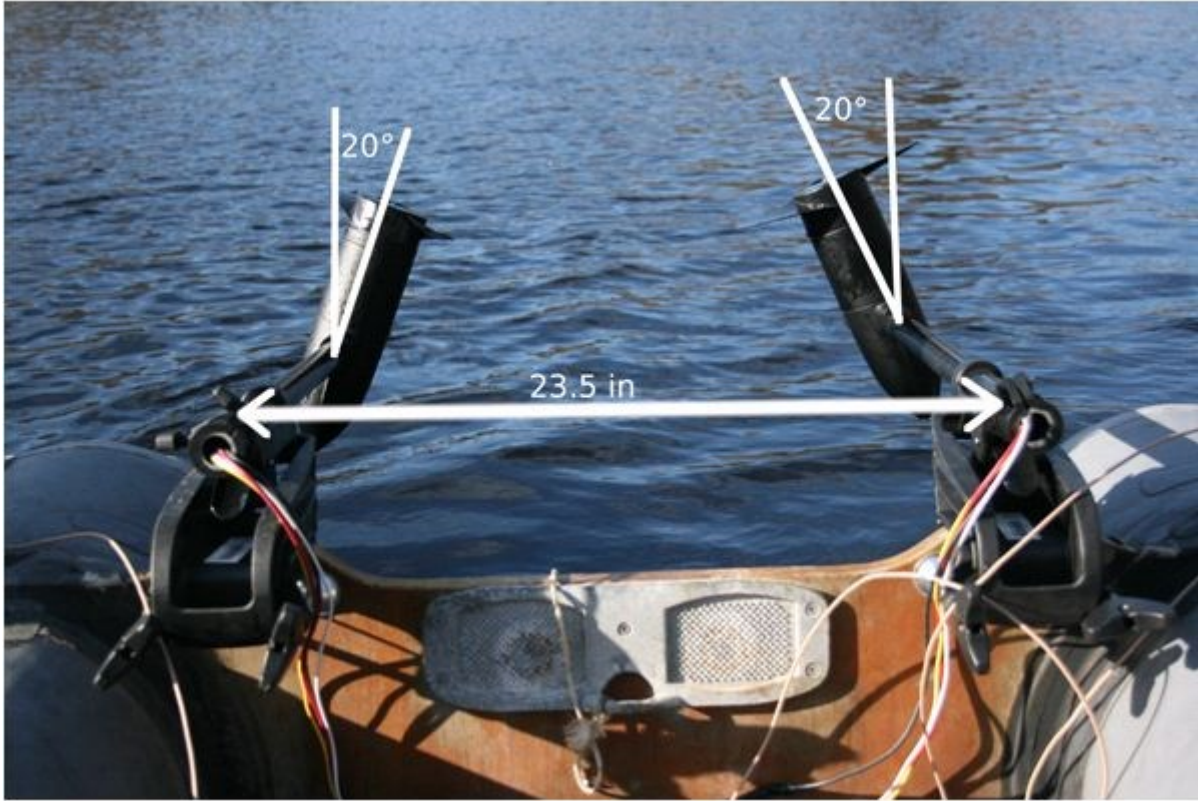
Figure 78 – Motors mounted towards each other
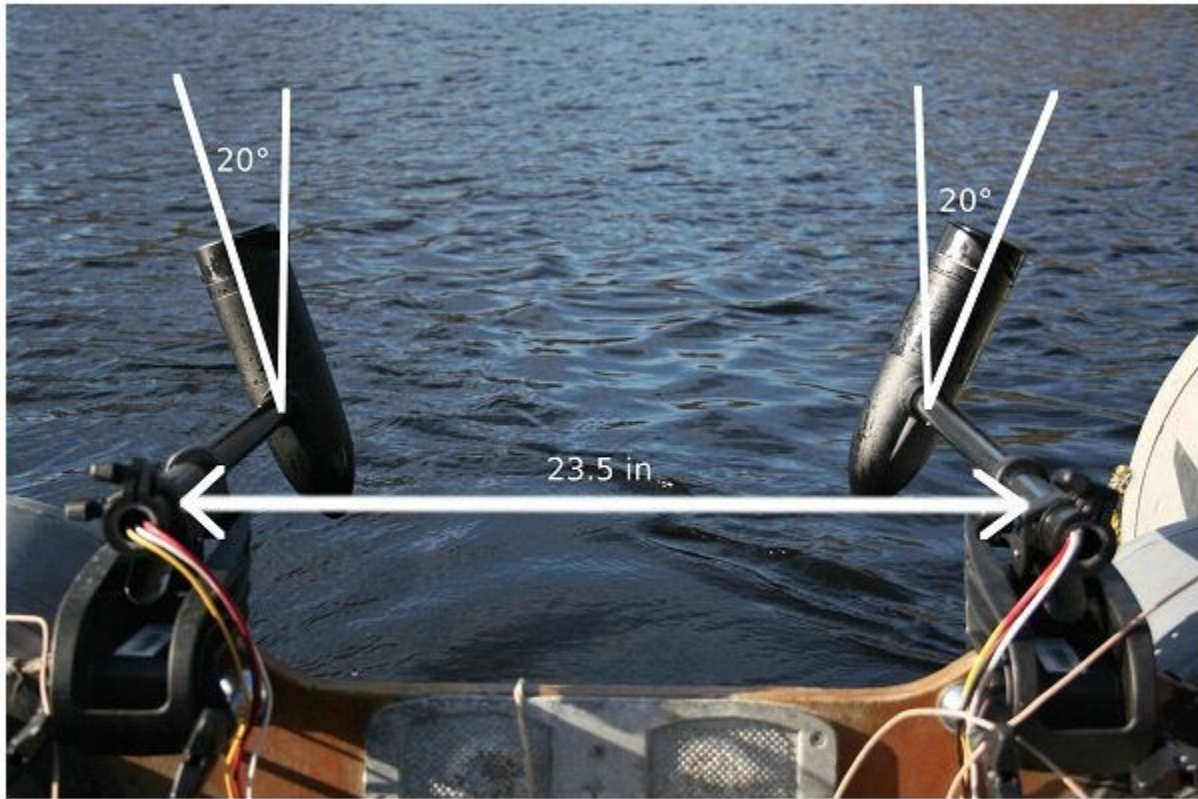
Figure 79 – Motors angled away from each other

Pulling Force

To determine the pulling force of the rescue craft, the boar was placed perpendicular to the shore and a rope was attached to the stern of the boat. Next, a luggage scale was attached to the rope and both motors were engaged. With the person weighing 150lbs in the boat, the Sea Rogue had a pulling force of approximately 28lbs.The pulling force of the boat without a person in it was 25.7 lbs.

Turning Maneuverability

To test the turning radius of the boat, the motors were set in three positions. In one position they were angled towards each other, in another they were pointed away from each other, and in the third they were parallel with one another. Each test was conducted both with and without a person in the boat.

Motors angled in

The below image shows a rough plot of the path traveled by the boat using an android app MyTracks (Google, 2011) with the motors angled in and no one in the boat. As can be seen, the boat has the best turning ability with the smallest turning circle diameter.
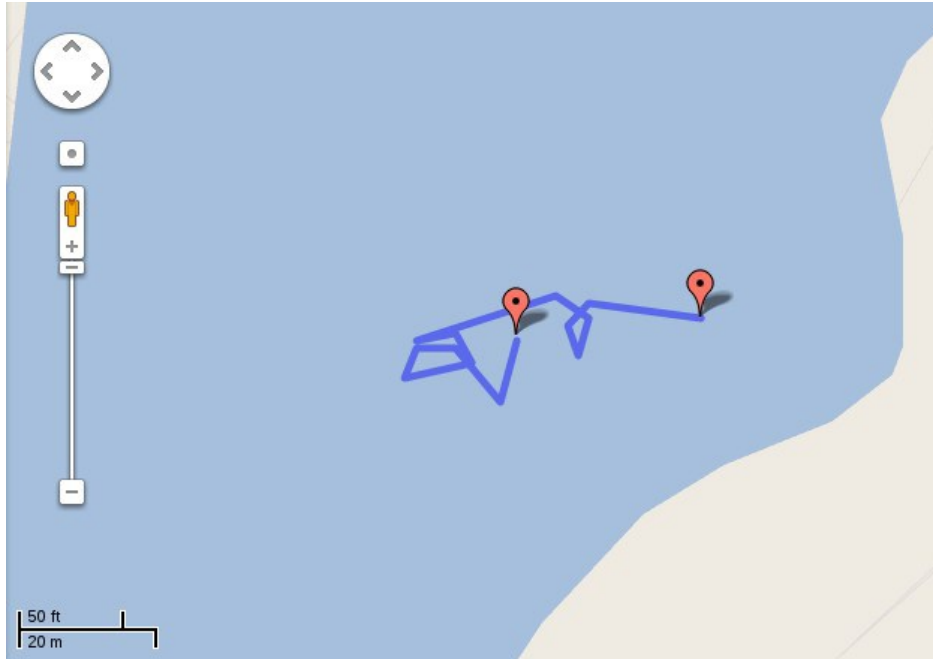
Figure 80 – Motors angled in and no one on the boat

Figure 81 shows the rough path traveled by the boat with one 150lb person in the boat and the motors angled towards each other. Again this exhibits the best tuning radius and forward travel speed.
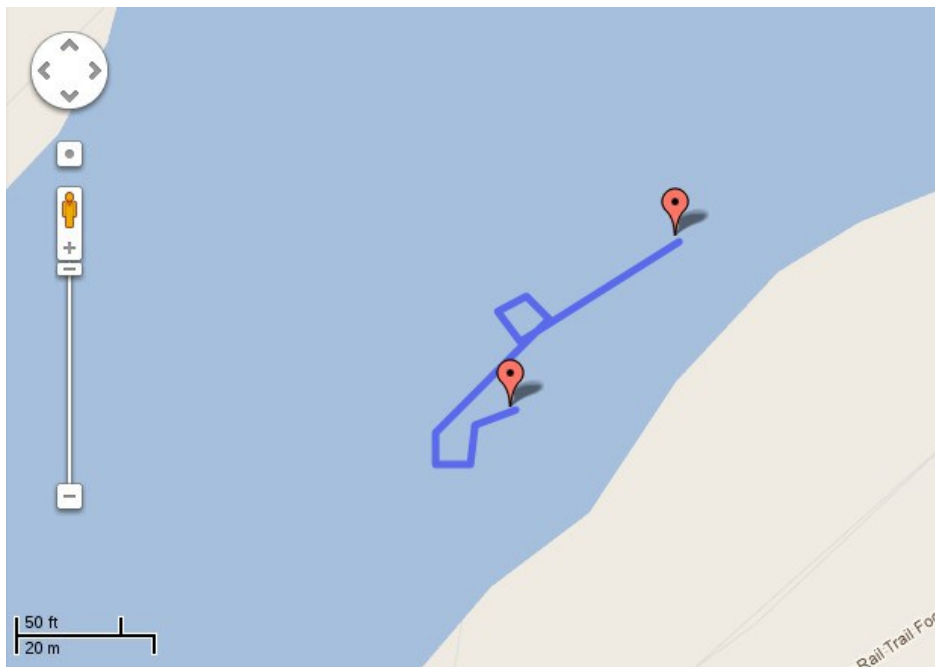
Figure 81 – Motors angled in and 150lb person on boat

Motors straight

Figure 5 shows the approximate path traveled by the boat while the motors were fixed straight without a person in the boat. Due to the wind, turning in the wind was deemed not practical with the motors in this configuration.

Figure 82 – Motors straight and no one in the boat

Motors angled out

The below figure shows the path the boat traveled with the the motors angled out and no one in the boat.

Figure 83 – Motors angled away with no one on the boat

As can be seen in Figure 7, the turning radius of the boat is not as good as the one seen in Figure 4, where the motors were angled in.
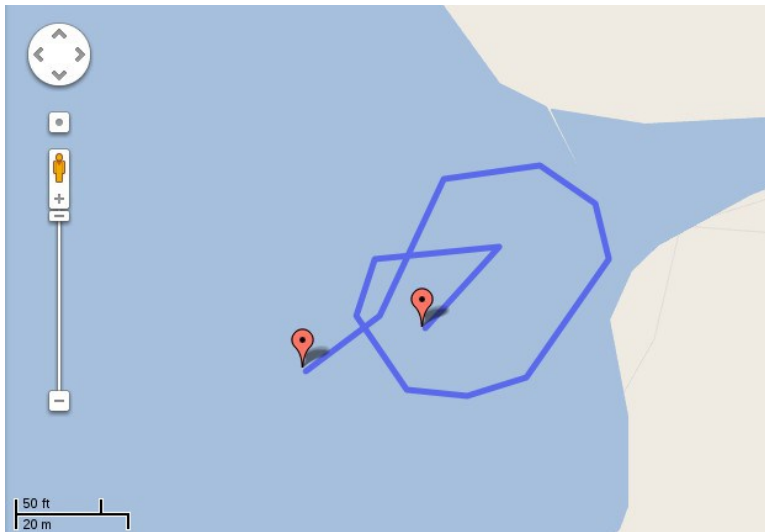
Figure 84 – Motors angled away and a 150lb person on the boat

Max speed

The maximum speed was also roughly calculated using the MyTracks app on the android. The boat was driven in a straight line at full throttle for approximately 30 seconds and the gps logger was later reviewed for the speed. Additionally, to verify the results of the maximum speed test, a distance of 62 feet was marked off on shore based off of landmarks. The time for the boat to travel this distance was recorded and by dividing the distance by travel time, the velocity was determined.

Motors angled in

While the motors were angled towards each other, the maximum achieved speed was 4.44 km/hr (2.39 knots) with a 150lb person in the boat. With no one in the boat, the speed was 5.21 km/hr (2.81 knots). The test was conducted both with and against the wind. Due to the wind and inaccuracy of the GPS long with the slow refresh rate of the android app, the calculated speed may be inaccurate and was validated by the method described above.

Motors straight

With the motors straight and no one in the boat, the android app clocked in a maximum speed of approximately 5.67 km/hr (3.06 knots). Again the validity of the results were verified as the GPS is not 100% accurate.

Motors angled out

While the motors were angled out, the maximum approximate speed was 5.23 km/hr (2.82 knots) without anyone in the boat. With someone in the boat, the maximum speed measured was 4.87 km/hr (2.63 knots).

Drag force

In order to determine the drag force exhibited by the rescue boat, a simple test was conducted. The rescue craft was towed by another boat without the motors attached. The towing boat was then run at a constant velocity of 1.6 knots and the drag force was measured on a digital luggage scale that was held from the stern of the towing boat and attached to the rescue boat by a rope. This measured force was 3.9lbs without the motors attached to the rescue boat.

82

Appendix B – Hull Speed Calculations

$$v_{hull} = k \times \sqrt{L_{waterline}} \qquad where \; k = 1.34$$

L.wl - Length of waterline [ft]	V.h - Hull Speed [Knots]	V.h [km/h]		constant
1	1.34	2.48		1.34
1.2	1.47	2.72		
1.4	1.59	2.94		
1.6	1.69	3.14		
1.8	1.80	3.33		
2	1.90	3.51		
2.2	1.99	3.68		
2.4	2.08	3.84		
2.6	2.16	4.00		
2.8	2.24	4.15		
3	2.32	4.30		
3.2	2.40	4.44		
3.4	2.47	4.58		
3.6	2.54	4.71		
3.8	2.61	4.84		
4	2.68	4.96		
4.2	2.75	5.09		
4.4	2.81	5.21		
4.6	2.87	5.32		
4.8	2.94	5.44		
5	3.00	5.55		
5.2	3.06	5.66		
5.4	3.11	5.77		
5.6	3.17	5.87		
5.8	3.23	5.98		
6	3.28	6.08		
6.2	3.34	6.18		
6.4	3.39	6.28		
6.6	3.44	6.38		
6.8	3.49	6.47		
7	3.55	6.57		
7.2	3.60	6.66		
7.4	3.65	6.75		
7.6	3.69	6.84		
7.8	3.74	6.93		
8	3.79	7.02		
8.2	3.84	7.11		
8.4	3.88	7.19		
8.6	3.93	7.28		
8.8	3.98	7.36		
9	4.02	7.45		

9.2	4.06	7.53
9.4	4.11	7.61
9.6	4.15	7.69
9.8	4.19	7.77
10	4.24	7.85

Appendix C – AM Station Query (FCC)

Search Parameters	
State:	MA
Lower Frequency	530
Upper Frequency	1700

```
WHYN      AM 560  kHz DA2  Daytime        B B LIC  SPRINGFIELD
MA
WHYN      AM 560  kHz DA2  Nighttime      B B LIC  SPRINGFIELD
MA
WTAG      AM 580  kHz DA2  Daytime        B B LIC  WORCESTER
MA US
WTAG      AM 580  kHz DA2  Nighttime      B B LIC  WORCESTER
MA US
WEZE      AM 590  kHz DA1  Unlimited      B B LIC  BOSTON
MA
WNNZ      AM 640  kHz DA2  Daytime        B B LIC  WESTFIELD
MA
WNNZ      AM 640  kHz DA2  Nighttime      B B LIC  WESTFIELD
MA
WSRO      AM 650  kHz DA2  Daytime        D B LIC  ASHLAND
MA
WSRO      AM 650  kHz DA2  Nighttime      D B LIC  ASHLAND
MA
WSRO      AM 650  kHz DA2  Daytime        D B APP  ASHLAND
MA US
WSRO      AM 650  kHz DA2  Nighttime      D B APP  ASHLAND
MA US
WRKO      AM 680  kHz DA2  Daytime        B B LIC  BOSTON
MA
WRKO      AM 680  kHz DA2  Nighttime      B B LIC  BOSTON
MA US
WTUB      AM 700  kHz NDD  Daytime        D B LIC  ORANGE-ATHOL
```

```
MA US
NEW           AM 720  kHz DA2  Daytime         B B -    BILLERICA
MA US         ---
NEW           AM 720  kHz DA2  Nighttime       B B -    BILLERICA
MA US
WACE          AM 730  kHz ND1  Daytime         D B LIC  CHICOPEE
MA
WACE          AM 730  kHz ND1  Nighttime       D B LIC  CHICOPEE
MA
WJIB          AM 740  kHz ND1  Daytime         D B LIC  CAMBRIDGE
MA US
WJIB          AM 740  kHz ND1  Nighttime       D B LIC  CAMBRIDGE
MA
WVNE          AM 760  kHz NDD  Daytime         D B LIC  LEICESTER
MA
WVNE          AM 760  kHz NDD  Critical Hours  D B LIC  LEICESTER
MA
WNNW          AM 800  kHz ND2  Daytime         B B LIC  LAWRENCE
MA
WNNW          AM 800  kHz ND2  Nighttime       B B LIC  LAWRENCE
MA US
WCRN          AM 830  kHz DA2  Daytime         B B LIC  WORCESTER
MA US
WCRN          AM 830  kHz DA2  Nighttime       B B LIC  WORCESTER
MA US
WEEI          AM 850  kHz DA2  Daytime         B B LIC  BOSTON
MA US
WEEI          AM 850  kHz DA2  Nighttime       B B LIC  BOSTON
MA US
WSBS          AM 860  kHz ND2  Daytime         D B LIC  GREAT BARRINGTON
MA
WSBS          AM 860  kHz ND2  Nighttime       D B LIC  GREAT BARRINGTON
MA
WSBS          AM 860  kHz ND2  Critical Hours  D B LIC  GREAT BARRINGTON
MA US
WAMG          AM 890  kHz DA2  Daytime         B B LIC  DEDHAM
MA US
WAMG          AM 890  kHz DA2  Nighttime       B B LIC  DEDHAM    > Lower Freq
MA
WGFP          AM 940  kHz ND2  Daytime         D B LIC  WEBSTER   > Upper Freq
MA
WGFP          AM 940  kHz ND2  Nighttime       D B LIC  WEBSTER
MA
WROL          AM 950  kHz NDD  Daytime         D B LIC  BOSTON
MA
WROL          AM 950  kHz ND1  Nighttime       D B LIC  BOSTON
MA
WROL          AM 950  kHz ND2  Daytime         D B APP  BOSTON
MA
WROL          AM 950  kHz ND2  Nighttime       D B APP  BOSTON
MA
WFGL          AM 960  kHz DA2  Daytime         B B LIC  FITCHBURG
MA
WFGL          AM 960  kHz DA2  Nighttime       B B LIC  FITCHBURG
MA
```

85

WESO MA	AM 970	kHz	ND1	Daytime	D	B	LIC	SOUTHBRIDGE
WESO MA	AM 970	kHz	ND1	Nighttime	D	B	LIC	SOUTHBRIDGE
WCAP MA	AM 980	kHz	DA2	Daytime	B	B	LIC	LOWELL
WCAP MA	AM 980	kHz	DA2	Nighttime	B	B	LIC	LOWELL
WCMX MA	AM 1000	kHz	NDD	Daytime	D	B	LIC	LEOMINSTER
WBZ MA	AM 1030	kHz	DA1	Unlimited	A	A	LIC	BOSTON
WQOM MA	AM 1060	kHz	DA2	Daytime	B	B	LIC	NATICK
WQOM MA US	AM 1060	kHz	DA2	Nighttime	B	B	LIC	NATICK
WILD MA	AM 1090	kHz	NDD	Daytime	D	B	LIC	BOSTON
WILD MA	AM 1090	kHz	NDD	Critical Hours	D	B	LIC	BOSTON
WUPE MA US	AM 1110	kHz	DAD	Daytime	D	B	LIC	PITTSFIELD
WBNW MA US	AM 1120	kHz	DA2	Daytime	B	B	LIC	CONCORD
WBNW MA	AM 1120	kHz	DA2	Nighttime	B	B	LIC	CONCORD
WWDJ MA US	AM 1150	kHz	DA2	Daytime	B	B	LIC	BOSTON
WWDJ MA	AM 1150	kHz	DA2	Nighttime	B	B	LIC	BOSTON
WDIS MA US	AM 1170	kHz	DAD	Daytime	D	B	LIC	NORFOLK
WFPB MA	AM 1170	kHz	DAD	Daytime	D	B	LIC	ORLEANS
WXKS MA US	AM 1200	kHz	DA2	Daytime	B	B	LIC	NEWTON
WXKS MA	AM 1200	kHz	DA2	Nighttime	B	B	LIC	NEWTON
WNAW MA US	AM 1230	kHz	ND1	Unlimited	C	C	LIC	NORTH ADAMS
WNEB MA	AM 1230	kHz	ND1	Unlimited	C	C	LIC	WORCESTER
WESX MA US	AM 1230	kHz	ND2	Daytime	C	C	LIC	NAHANT
WESX MA	AM 1230	kHz	ND2	Nighttime	C	C	LIC	NAHANT
WBUR MA US	AM 1240	kHz	ND1	Unlimited	C	C	LIC	WEST YARMOUTH
WHMQ MA US	AM 1240	kHz	ND2	Daytime	C	C	LIC	GREENFIELD
WHMQ MA US	AM 1240	kHz	ND2	Nighttime	C	C	LIC	GREENFIELD
WARE MA US	AM 1250	kHz	DA2	Daytime	B	B	LIC	WARE
WARE	AM 1250	kHz	DA2	Nighttime	B	B	LIC	WARE

86

```
MA US
WMKI         AM 1260 kHz DAN  Daytime          B B LIC  BOSTON
MA US
WMKI         AM 1260 kHz DAN  Nighttime        B B LIC  BOSTON
MA
WSPR         AM 1270 kHz DA2  Daytime          B B LIC  SPRINGFIELD
MA
WSPR         AM 1270 kHz DA2  Nighttime        B B LIC  SPRINGFIELD
MA US
WSPR         AM 1270 kHz ND2  Daytime          B B CP   SPRINGFIELD
MA US
WSPR         AM 1270 kHz ND2  Nighttime        B B CP   SPRINGFIELD
MA US
WSPR         AM 1270 kHz ND2  Daytime          B B LIC  SPRINGFIELD
MA US
WSPR         AM 1270 kHz ND2  Nighttime        B B LIC  SPRINGFIELD
MA US
WPKZ         AM 1280 kHz DA2  Daytime          B B LIC  FITCHBURG
MA US
WPKZ         AM 1280 kHz DA2  Nighttime        B B LIC  FITCHBURG
MA
WJDA         AM 1300 kHz ND1  Daytime          D B LIC  QUINCY
MA
WJDA         AM 1300 kHz ND1  Nighttime        D B LIC  QUINCY
MA US
WORC         AM 1310 kHz DA2  Daytime          B B LIC  WORCESTER
MA US
WORC         AM 1310 kHz DA2  Nighttime        B B LIC  WORCESTER
MA
WARL         AM 1320 kHz DA2  Nighttime        B B LIC  ATTLEBORO
MA US
WARL         AM 1320 kHz DA2  Daytime          B B LIC  ATTLEBORO
MA
WRCA         AM 1330 kHz DA2  Daytime          B B LIC  WATERTOWN
MA
WRCA         AM 1330 kHz DA2  Nighttime        B B LIC  WATERTOWN
MA US
WGAW         AM 1340 kHz ND1  Unlimited        C C LIC  GARDNER
MA US
WBRK         AM 1340 kHz ND1  Unlimited        C C LIC  PITTSFIELD
MA US
WNBH         AM 1340 kHz ND1  Unlimited        C C LIC  NEW BEDFORD
MA US
WLYN         AM 1360 kHz ND2  Daytime          D B LIC  LYNN
MA US
WLYN         AM 1360 kHz ND2  Nighttime        D B LIC  LYNN
MA
WPLM         AM 1390 kHz DA2  Daytime          B B LIC  PLYMOUTH
MA
WPLM         AM 1390 kHz DA2  Nighttime        B B LIC  PLYMOUTH
MA US
WHTB         AM 1400 kHz ND1  Unlimited        C C LIC  FALL RIVER
MA
WHMP         AM 1400 kHz ND1  Unlimited        C C LIC  NORTHAMPTON
MA
```

```
WLLH          AM 1400 kHz ND1  Unlimited        C C LIC  LAWRENCE
MA US
WLLH          AM 1400 kHz ND1  Unlimited        C C LIC  LOWELL
MA
WMSX          AM 1410 kHz DA2  Daytime          D B LIC  BROCKTON
MA
WMSX          AM 1410 kHz DA2  Nighttime        D B LIC  BROCKTON
MA US
WBSM          AM 1420 kHz DA2  Daytime          B B LIC  NEW BEDFORD
MA
WBSM          AM 1420 kHz DA2  Nighttime        B B LIC  NEW BEDFORD
MA US
WBEC          AM 1420 kHz DAN  Daytime          B B LIC  PITTSFIELD
MA US
WBEC          AM 1420 kHz DAN  Nighttime        B B LIC  PITTSFIELD
MA US
WPNI          AM 1430 kHz DA2  Daytime          D B LIC  AMHERST
MA US
WKOX          AM 1430 kHz DAN  Daytime          B B LIC  EVERETT
MA
WKOX          AM 1430 kHz DAN  Nighttime        B B LIC  EVERETT
MA US
WPNI          AM 1430 kHz DA2  Nighttime        D B LIC  AMHERST
MA
WPNI          AM 1430 kHz ND2  Daytime          D B APP  AMHERST
MA
WPNI          AM 1430 kHz ND2  Nighttime        D B APP  AMHERST
MA US
WVEI          AM 1440 kHz DAN  Daytime          B B LIC  WORCESTER
MA US
WVEI          AM 1440 kHz DAN  Nighttime        B B LIC  WORCESTER
MA
WNBP          AM 1450 kHz ND1  Unlimited        C C LIC  NEWBURYPORT
MA
WHLL          AM 1450 kHz ND1  Unlimited        C C LIC  SPRINGFIELD
MA US
WXBR          AM 1460 kHz DAN  Daytime          B B LIC  BROCKTON
MA
WXBR          AM 1460 kHz DAN  Nighttime        B B LIC  BROCKTON
MA US
WAZN          AM 1470 kHz DA2  Daytime          B B LIC  WATERTOWN
MA
WAZN          AM 1470 kHz DA2  Nighttime        B B LIC  WATERTOWN
MA
WSAR          AM 1480 kHz DA1  Unlimited        B B LIC  FALL RIVER
MA
WSAR          AM 1480 kHz DA2  Daytime          B B CP   FALL RIVER
MA US
WSAR          AM 1480 kHz DA2  Nighttime        B B CP   FALL RIVER
MA US
WCEC          AM 1490 kHz ND1  Unlimited        C C LIC  HAVERHILL
MA US
WMRC          AM 1490 kHz ND1  Unlimited        C C LIC  MILFORD
MA US
WACM          AM 1490 kHz ND1  Unlimited        C C LIC  WEST SPRINGFIELD
```

```
MA
WACM        AM 1490 kHz ND2   Daytime          C C LIC   WEST SPRINGFIELD
MA US
WACM        AM 1490 kHz ND2   Nighttime        C C LIC   WEST SPRINGFIELD
MA
WUFC        AM 1510 kHz DA3   Daytime          B B LIC   BOSTON
MA
WUFC        AM 1510 kHz DA3   Nighttime        B B LIC   BOSTON
MA US
WUFC        AM 1510 kHz DA3   Critical Hours   B B LIC   BOSTON
MA US
WIZZ        AM 1520 kHz DAD   Daytime          D B LIC   GREENFIELD
MA
WVBF        AM 1530 kHz ND3   Daytime          D B LIC   MIDDLEBOROUGH CENTER
MA
WVBF        AM 1530 kHz ND3   Nighttime        D B LIC   MIDDLEBOROUGH CENTER
MA
WVBF        AM 1530 kHz ND3   Critical Hours   D B LIC   MIDDLEBOROUGH CENTER
MA US
WNTN        AM 1550 kHz ND2   Daytime          D B LIC   NEWTON
MA US       WNTN    AM 1550 kHz ND2  Nighttime    D B LIC   NEWTON
MA US
WMVX        AM 1570 kHz ND1   Daytime          D B LIC   BEVERLY
MA US
WMVX        AM 1570 kHz ND1   Nighttime        D B LIC   BEVERLY
MA
WMVX        AM 1570 kHz ND2   Daytime          D B APP   BEVERLY
MA
WMVX        AM 1570 kHz ND2   Nighttime        D B APP   BEVERLY
MA
WHNP        AM 1600 kHz NDD   Daytime          D B LIC   EAST LONGMEADOW
MA US
WUNR        AM 1600 kHz DA1   Daytime          B B LIC   BROOKLINE
MA
WUNR        AM 1600 kHz DA1   Nighttime        B B LIC   BROOKLINE
MA
```

*** 142 Records Retrieved ***

Appendix D – Antenna Tuning

Tuning the antenna to the desired frequency was accomplished by hooking up the ferrite core antenna to an oscilloscope and function generator as depicted in Figure 85 – Antenna Tuning. To find the natural frequency of the antenna, the frequency from the signal generator was swept to find the maximum amplitude on the oscilloscope. Once this frequency was found, the inductance of the antenna could be found due to the known capacitance and frequency using the formula below.

$$L_{ant} = \left(\frac{1}{2 \cdot \pi \cdot f_0}\right)^2 \cdot \frac{1}{C}$$

In the above formula, L_{ant} is the inductance of the antenna, f_0 is the natural frequency and C is the

89

capacitance of the capacitor.

This in turn allowed for calculation to find the correct capacitance to match the desired frequency the antenna should be tuned to. The formula is rearranged below.

$$C = \left(\frac{1}{2 \cdot \pi \cdot f_0}\right)^2 \cdot \frac{1}{L_{ant}}$$
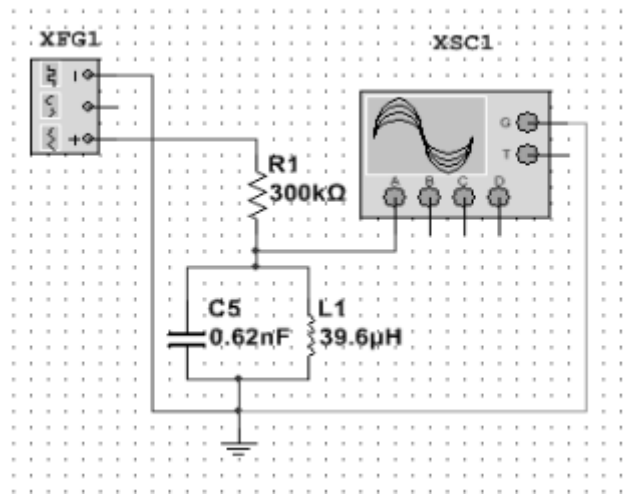
Figure 85 – Antenna Tuning

Appendix E – Turning Circle Code

```
#Octave script used for calculating the turning radius of the Zodiac
boat
#Written by Frederick Hunter on 25 March, 2013 based on Equations
    from:    What is the "MOB" function of my marine GPS device? (2011, March 28). Retrieved
    12 10, 12, from Garmin:
    https://support.garmin.com/support/searchSupport/case.faces?caseId={71655ee0-104c-11dc-
    4b06-000000000000}

pgs 206-209


leftMotorAngle = -20*(pi/180); # the angle of the left motor from the
perpendicular line of the stern of the boat
rightMotorAngle = 20*(pi/180); # the angle of the right motor from the
perpendicular line of the stern of the boat

#distance from motors to the center of the boat in y direction
DxLeft = -0.9144; # the distance in meters
DxRight = -0.9144;
```

```
#distance from motors to the center of the boat in x direction
DyLeft = 0.299; #distance in meters
DyRight = 0.299;
#mass of the boat in kgs
Mass = 73.03; #in kg

leftMotorThrust =133.4;
rightMotorThrust = 0;#133.4; # the thrust in Newtons (30lbs)
#print the information of the experiment
printf("The left motor thrust is %.2f N (%.2f lbs)\n",leftMotorThrust,
leftMotorThrust/4.448)
printf("The right motor thrust is %.2f N (%.2f
lbs)\n",rightMotorThrust, rightMotorThrust/4.448)

#Assuming linear speed and full thrust = 250.71 N = 1.543 m/s use the
following ratio
ratio = 0.0061545212;

#find the thrust in the x (bow of the boat/surge) direction
Xcontrol =
leftMotorThrust*cos(leftMotorAngle)+rightMotorThrust*cos(rightMotorAng
le);
#find the thrust in the y (leftor right/sway) direction
Ycontrol =
leftMotorThrust*sin(leftMotorAngle)+rightMotorThrust*sin(rightMotorAng
le);

#find the thrust in the z (yaw)
Ncontrol =
leftMotorThrust*DxLeft*sin(leftMotorAngle)+rightMotorThrust*DxRight*si
n(rightMotorAngle)+leftMotorThrust*DyLeft*cos(leftMotorAngle)-
rightMotorThrust*DyRight*cos(rightMotorAngle);

# print results to screen
printf("Calculated X thrust: %.2f N (%.2f lbs)\n", Xcontrol,
Xcontrol/4.448)
printf("Calculated Y thrust: %.2f N (%.2f
lbs)\n",Ycontrol,Ycontrol/4.448)
printf("Calculated rotational thrust about the turning center is: %.2f
N-m\n",Ncontrol)

#calculate the approximate speed using ratio
xVelocity = Xcontrol*ratio;
yVelocity = Ycontrol*ratio;
#nVelocity = Ncontrol*ratio;
nVelocity = 0.2244; #value calculated from water test (rad/sec)
printf("Approximate X velocity: %.2f m/s\n",xVelocity)
printf("Approximate Y velocity: %.2f m/s\n",yVelocity)
printf("Approximate Yaw velocity: %.2f rad/s\n",nVelocity)
```

```
#calculate the radius
radius = sqrt(xVelocity*xVelocity+yVelocity*yVelocity) / nVelocity;
printf("The turning radius of the boat is: %.2f m (%.2f ft)\n",radius,
radius*3.281)
printf("..and the diameter is %.2f m (%.2f
ft)\n",radius*2,radius*2*3.281)

#calculate the acceleration of the boat F = Ma
accelX = Xcontrol / Mass;
accelY = Ycontrol / Mass;
#printf("Calculated X acceleration: %.2f m/s^2\n", accelX)
#printf("Calculated acceleration: %.2f m/s^2\n", accelY)

#plot the graph
x = 0;
y = 0;
#get a time vector
theta =[0:0.1:2*pi];
x = radius*cos(theta);
y = radius*sin(theta);
plot(x,y);
title("Turning Radius of Zodiac calculation");
xlabel("X-axis (meters)");
ylabel("Y-axis (meters)");
axis ([-5, 5, -5, 5], "square"); #force square axis ratio
```

Appendix F – SolidWorks Flow Simulation 2012 Procedure

To set up a new flow simulation to determine the drag on the rescue craft, the following procedure was implemented. First, the **add-on** for **SolidWorks Flow Simulation 2012** was selected from the **Tools** -> **Add-Ins** menu in Solidworks.
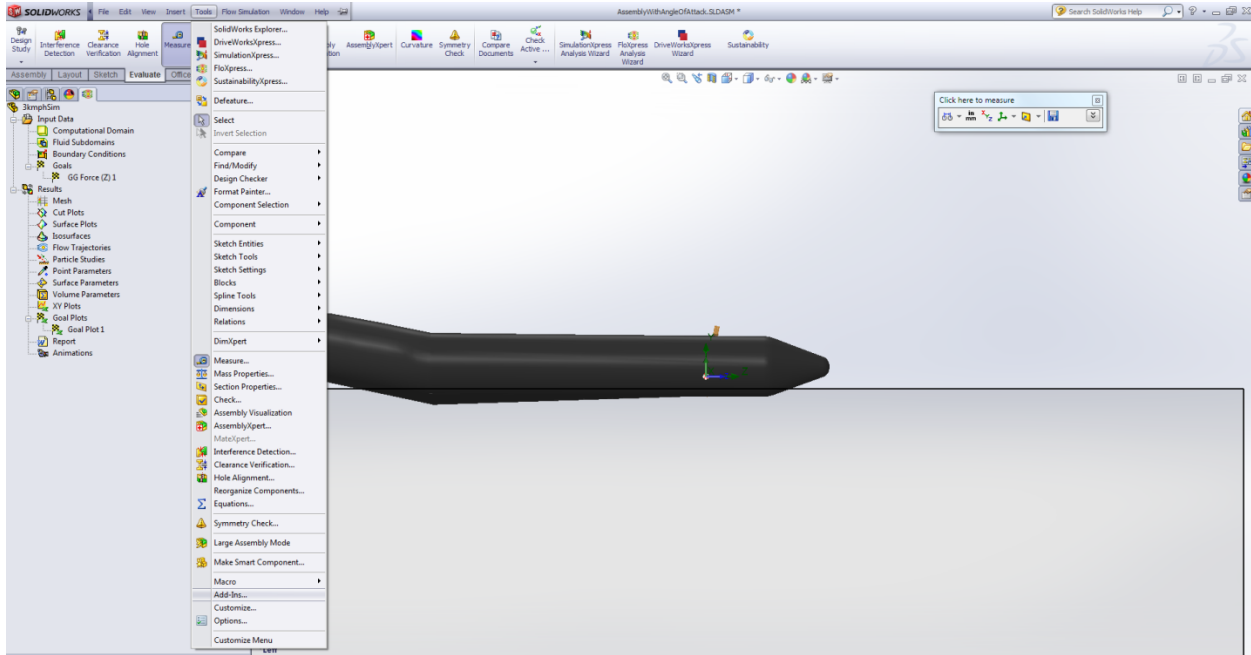
Figure 86 – Selecting Flow Simulation Add-In for SolidWorks 2012
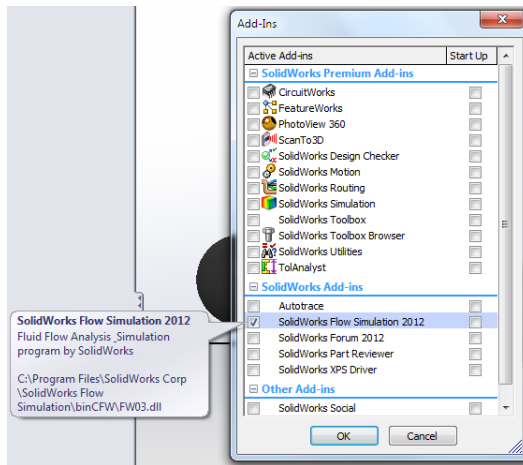
Figure 87 – Flow Simulation 2012

From here, the **Flow Simulation** menu was clicked and a new project was created by selecting **Project** - > **Wizard**.
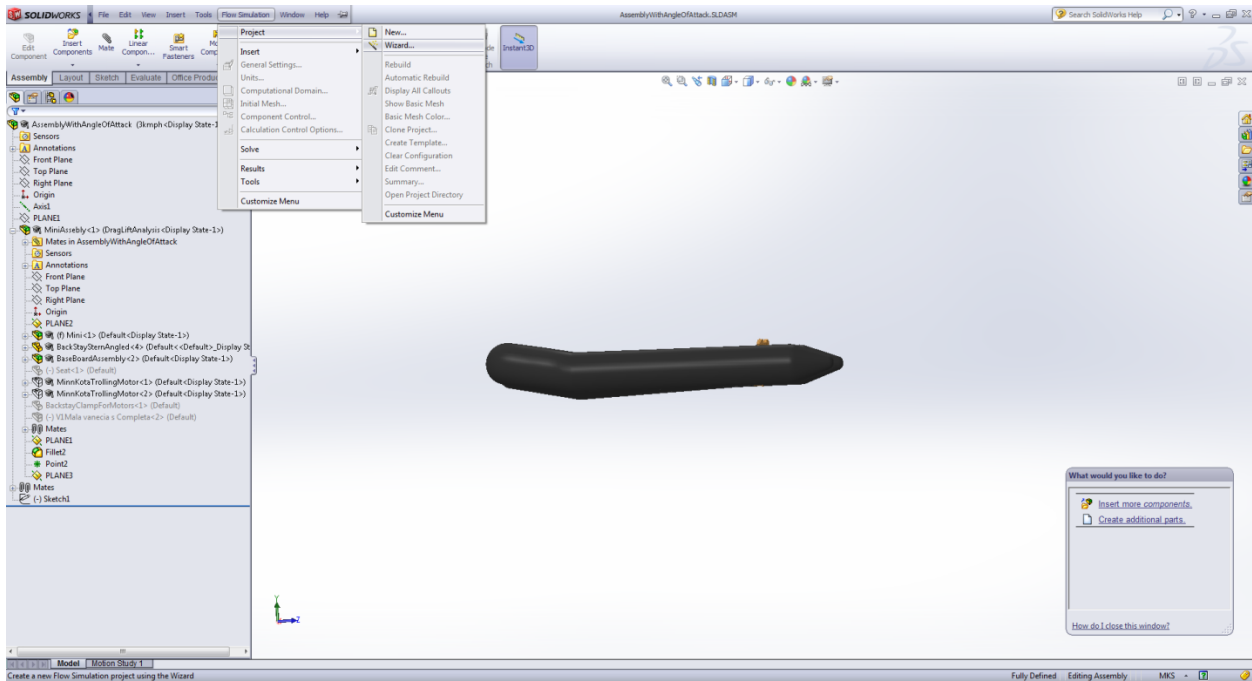
Figure 88 – Flow Simulator 2012 | New Project Wizard

The wizard opens a new dialog window and the project name is entered under **configuration name**.
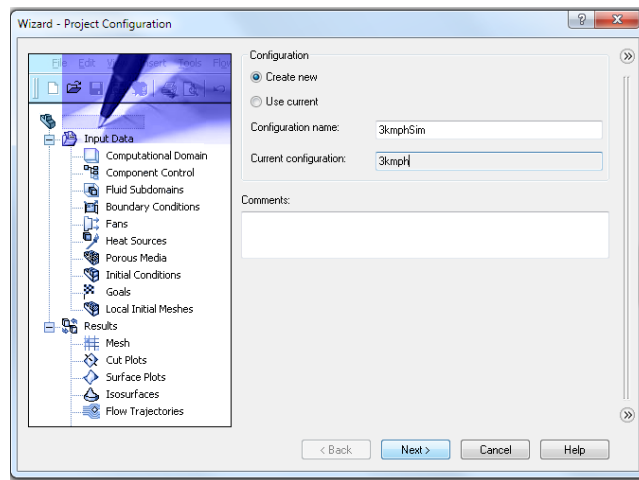
Figure 89 – Flow Simulator 2012 | Project Configurations

Next, the unit system is chosen, for this configuration the FPS system was used with the velocity measured in kmh.
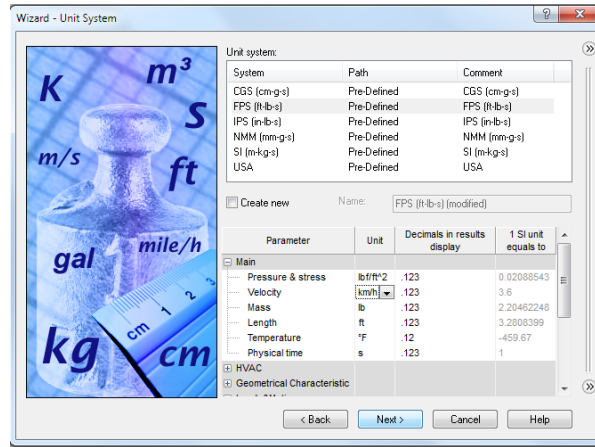
Figure 90 – Flow Simulator 2012 | Unit System

Clicking **next** brings the user to the analysis type selection window. Here an external flow analysis is selected with the **exclude internal space** and **exclude cavities without flow conditions** check boxes selected. Please note that for this model, the **reference axis** is the Z-axis which is the direction that the fluid will travel in.
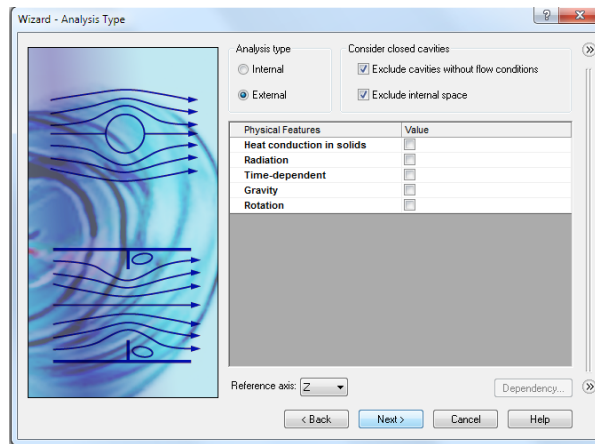
Figure 91 - Flow Simulator 2012 | Analysis Type

The default fluid is selected next. Because the rescue boat will be traveling through water, this simulation uses water as the fluid type.
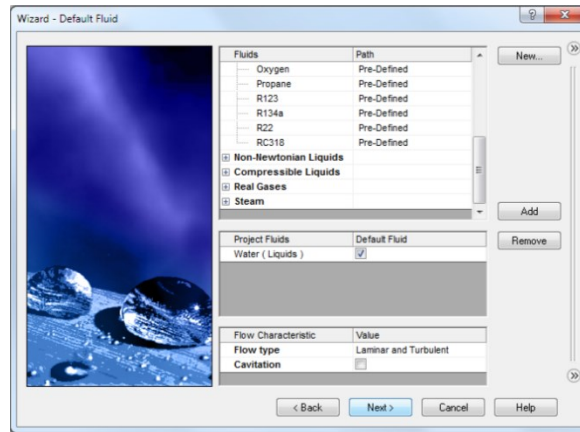
Figure 92 - Flow Simulator 2012 | Default Fluid Selection

The next window accepts the default values and the **next** button is pressed.
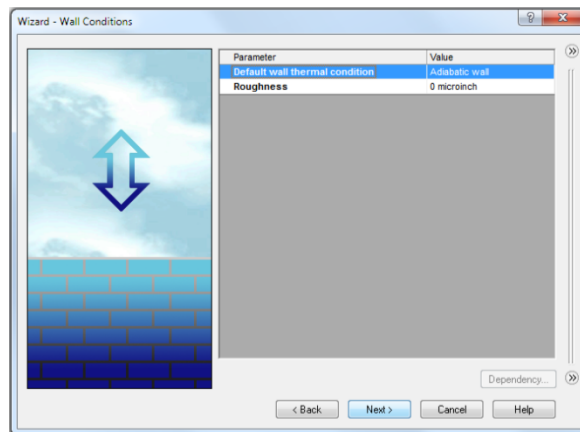
Figure 93 - Flow Simulator 2012 | Wall Conditions

In the initial and ambient conditions dialog box, the velocity of the traveling fluid is set to 3 kmh. This is the speed at which the rescue boat traveled during the water tow test and will provide a reasonable comparison to experimentally obtained values for drag.
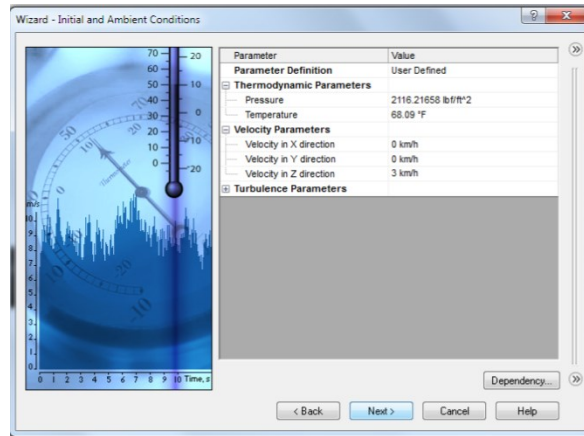
Figure 94 - Flow Simulator 2012 |Initial and Ambient Conditions

Defaults are accepted in the results and geometry resolution dialog box. Adjusting the result resolution scale will change the calculation time and accuracy of a solution.
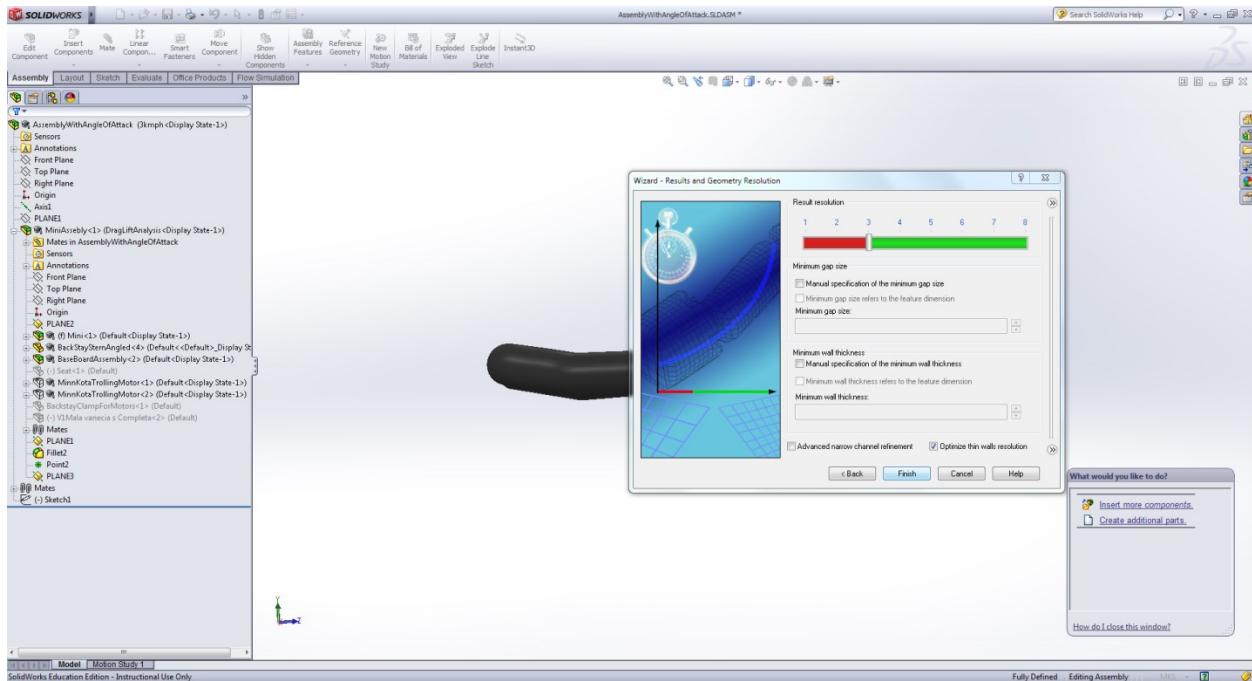
Figure 95 - Flow Simulator 2012 | Simulation Resolution

Next, the computational domain is set by right clicking on **computational domain** and choosing **edit** definition under the flow simulation tab.
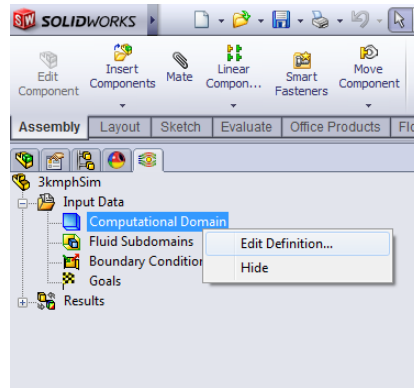
Figure 96 - Flow Simulator 2012 | Computational Domain

The computational domain determines where the flowing water is and therefore must include the draft of the boat. From prior analysis, the boat was calculated to have 0.85 inches of the hull exposed to the water. The coordinates of the computational domain are based off of the local coordinate system on the boat and therefore the values are different from the calculated one stated above. To ensure that the correct distance is set, a reference geometry plane was created and the offset distance set equal to that from the local coordinate system.
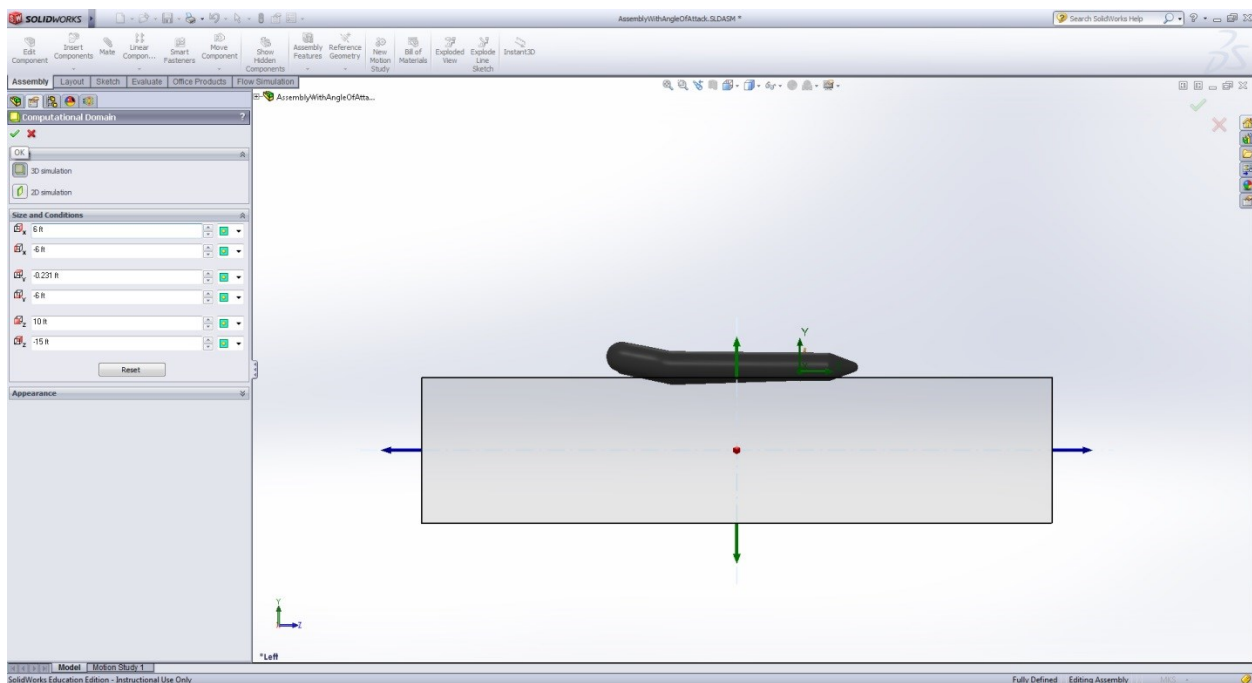
Figure 97 - Flow Simulator 2012 | Final Computational Domain

As verification for the offset distance, the measure tool was used to measure how low the boat is sitting in the water.
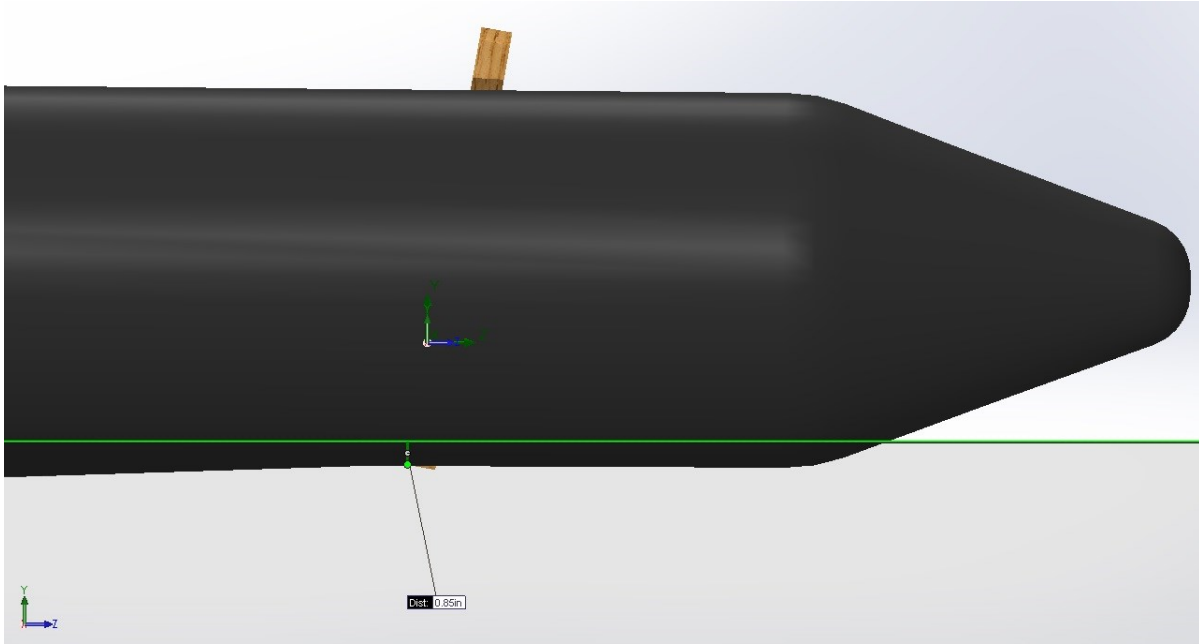
Figure 98 - Flow Simulator 2012 | Water Level Check

Next, global goals had to be defined such that the computer knows what to solve for. This was accomplished by right clicking **goals** and selecting **insert global goals**.
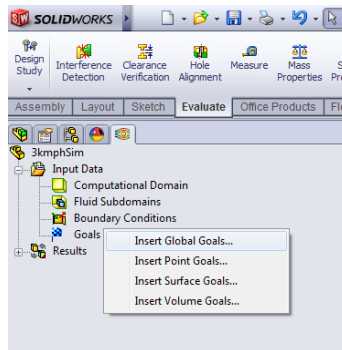
Figure 99 - Flow Simulator 2012 | Insert Global Goals

This opens the global goals that may be solved for in the simulation. To select a global goal, simply check the box next to the parameter of interest. For the drag, the user is interested in the maximum force in the Z-direction. For this reason, the **Force (Z)** parameter is checked off.
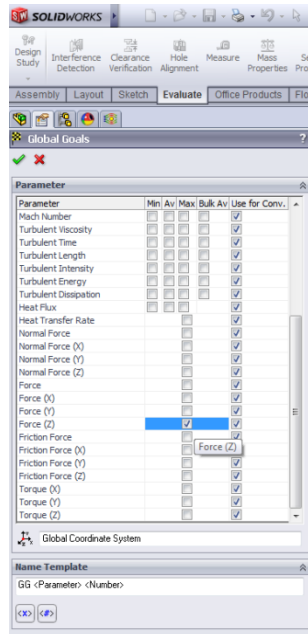
Figure 100 - Flow Simulator 2012 | Selecting Global Goals

Upon accepting the global goal, the simulation is run by right clicking the simulation file name, in this case **3kmphSim** and selecting **Run**.
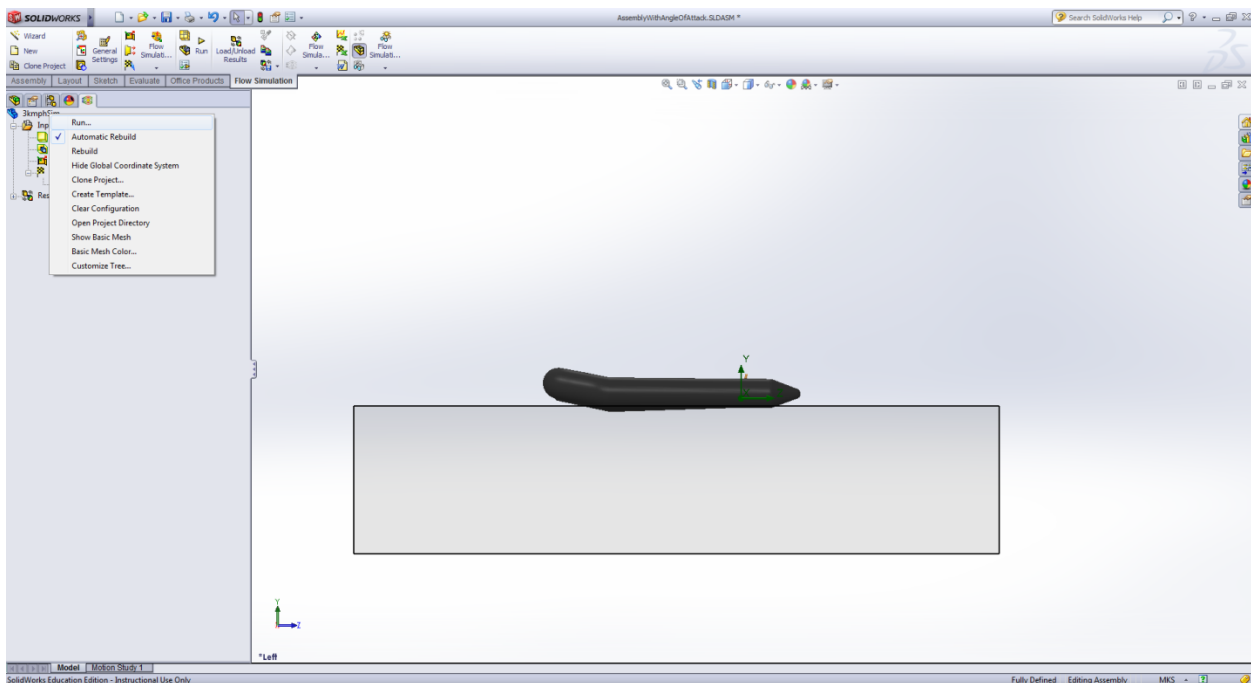
Figure 101 - Flow Simulator 2012 | Run Simulation

The run dialog box opens and the user may accept default values and click **Run**.
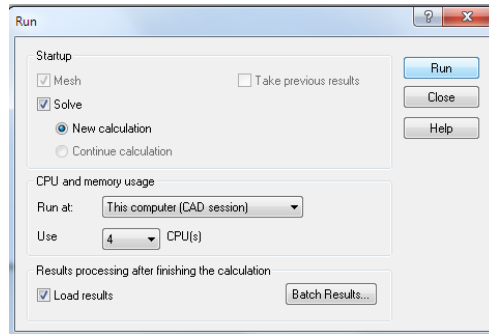
Figure 102 - Flow Simulator 2012 | Run Dialog Box

This opens the calculation window which displays the related information and updates for the calculations. The calculation will run until convergence is reached.
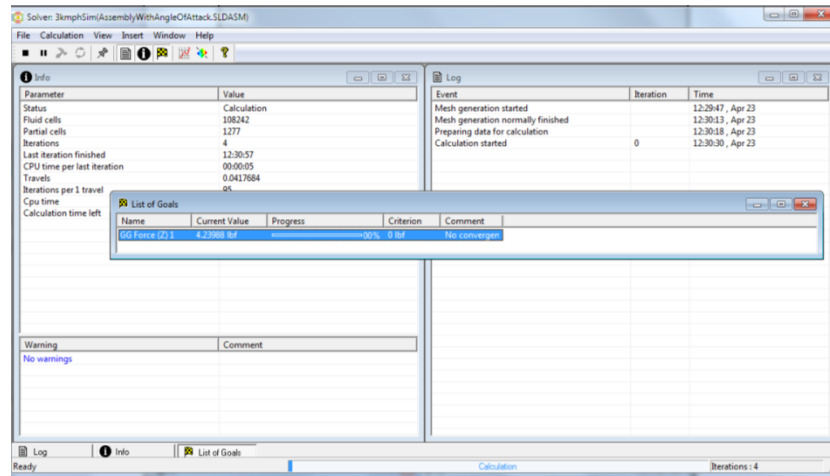
Figure 103 - Flow Simulator 2012 | Calculations Dialog Box

When the calculation has completed, the dialog box may be closed and the user can return to the Solidworks file. Here, a new goal plot may be generated by right clicking on **Goal Plots** and selecting **Insert.** This will open an excel document that shows what the solutions to the goals are. In this case, the calculated drag was determined to be 2.8 lbs.
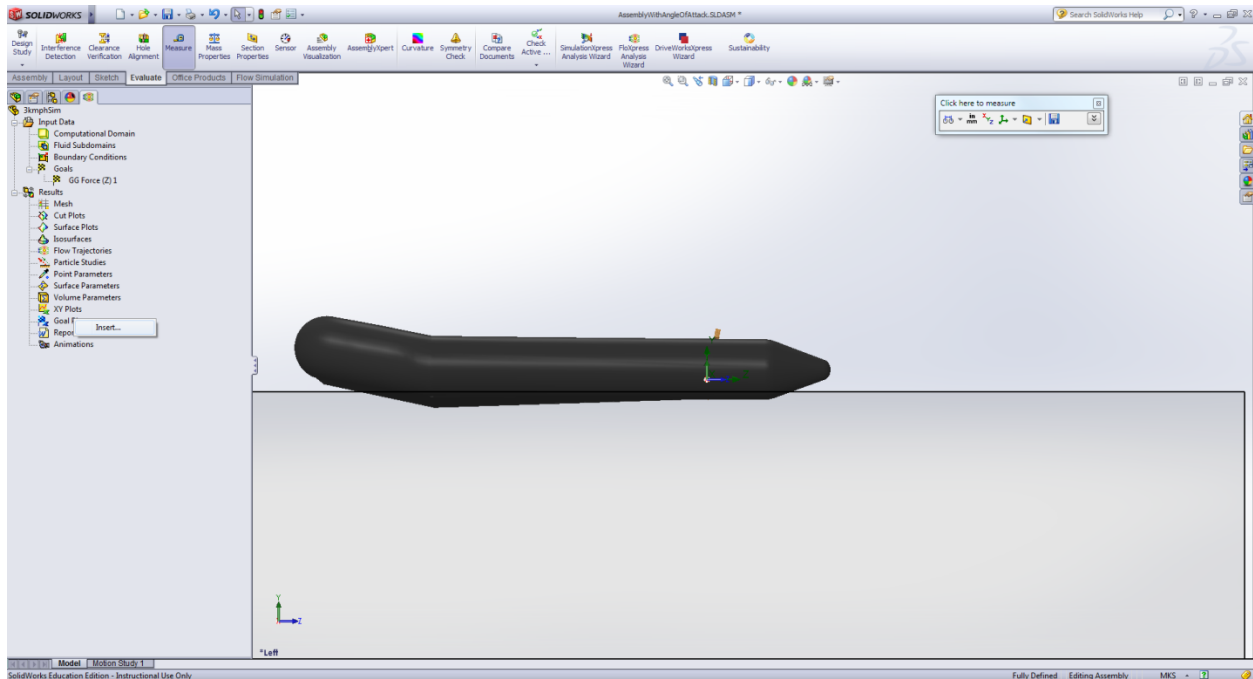
Figure 104 - Flow Simulator 2012 | Inserting Goal Plots

Other options are also available to visually display forces on the surface of the boat. By right clicking on the parameters and selecting **insert**, one has the option to display the calculated results visually. Some popular visual graphs include the contour plot and the flow lines.

Appendix G – Drawing File for Pi Box

The screenshot from Solidworks is a .dwg file that was downloaded from thingiverse.com (Makerbot Industries, LLC, 2013). The file was used to lasercut 1/8th inch thick acrylic to make the enclosure for the Raspberry Pi.
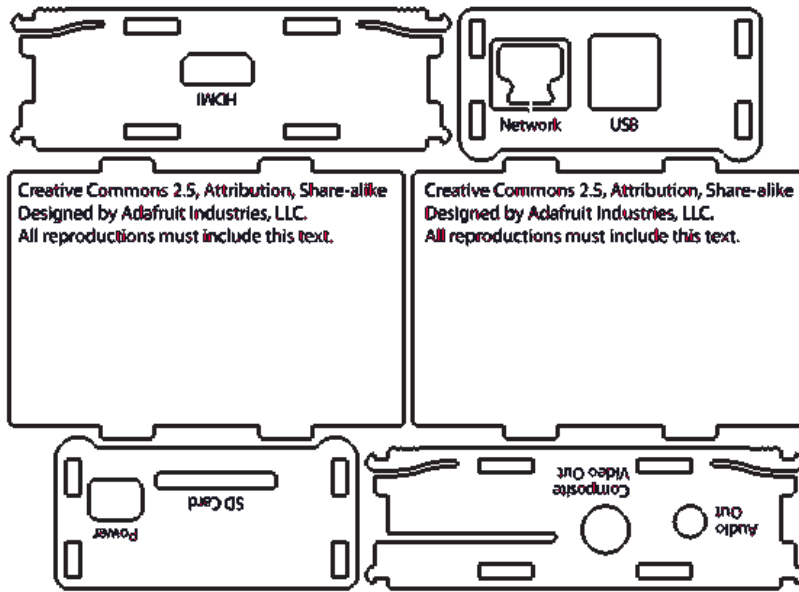
Figure 105 – PiBox Lasercut Template from Adafruit

Appendix H – MQP Poster

Appendix I – AMORE MQP Presentation

Problem Statement

- Man overboard (MOB) fatalities make up 24% boating deaths (Boatus.com)
- Of these, 57% of MOB cases (from 2000-2010) were not witnessed

The MORE Solution

- Fully Autonomous System that
  - Tracks and locates person
  - Returns them to the mothership upon button press
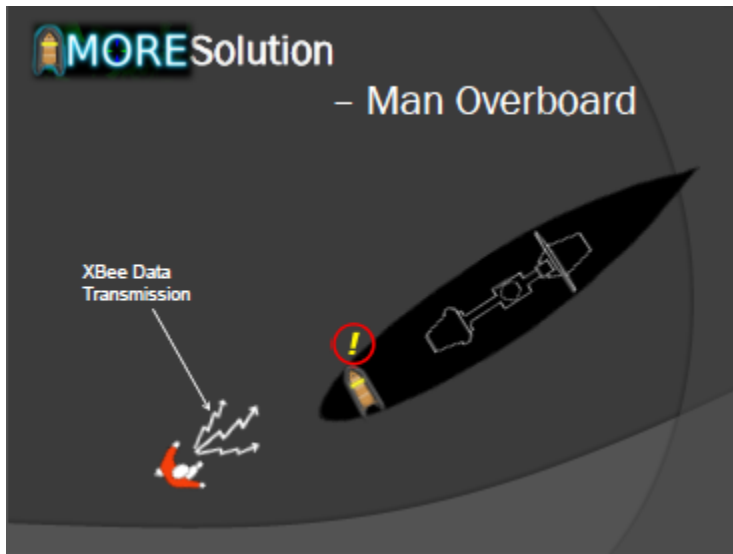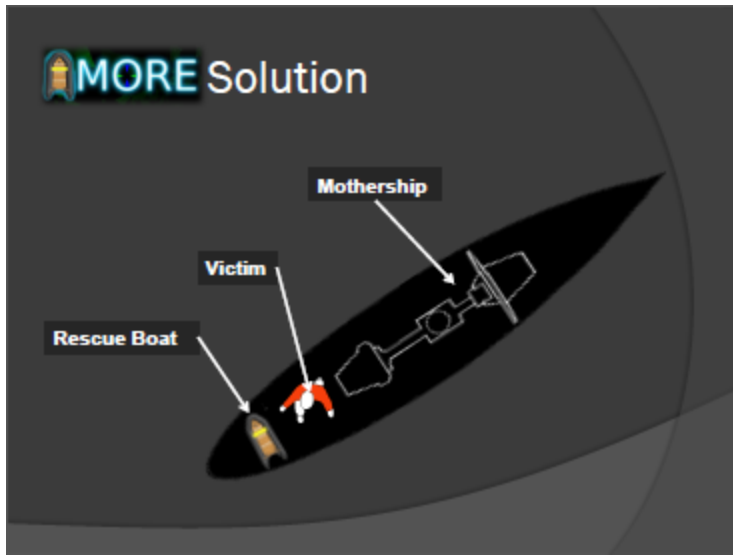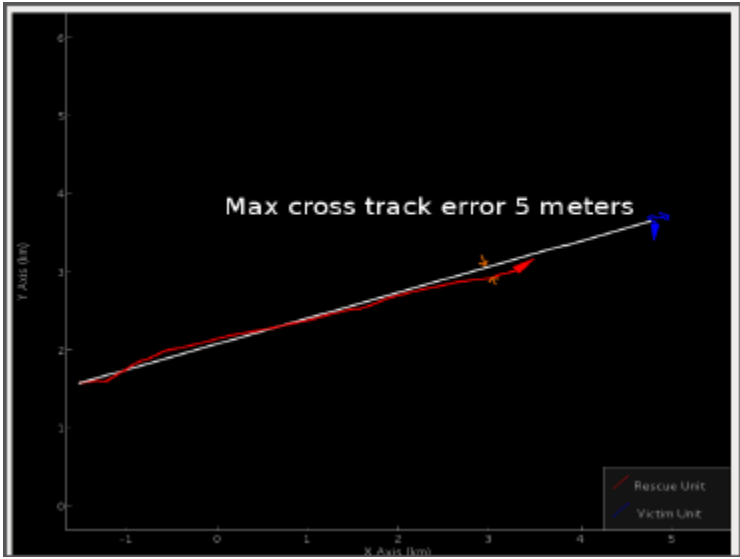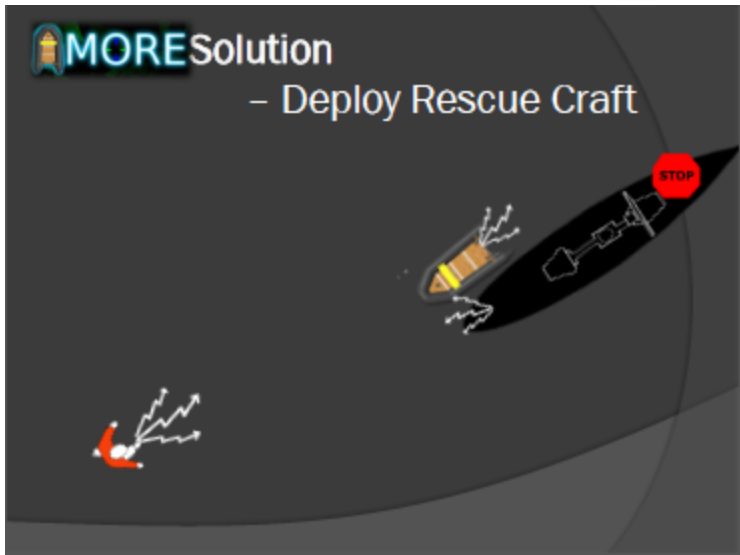
Max cross track error 5 meters
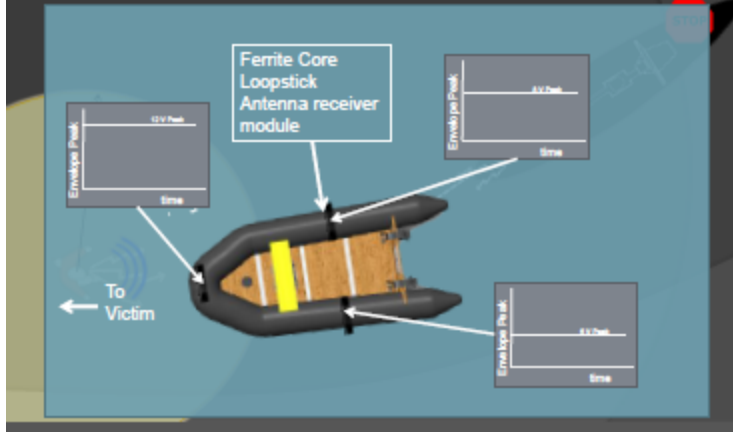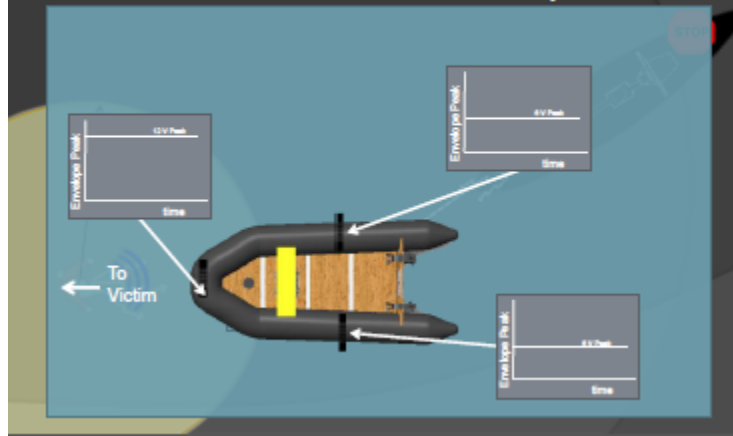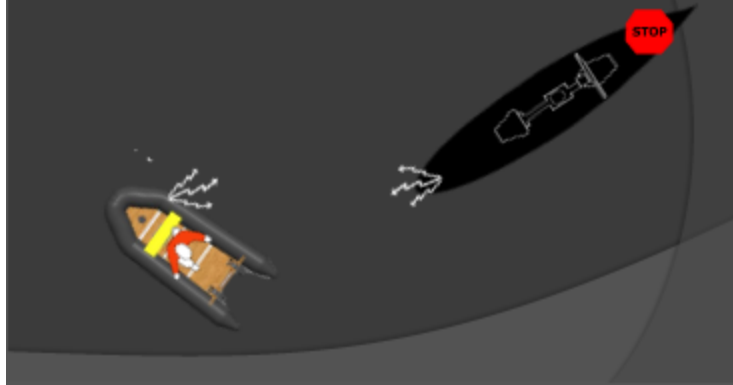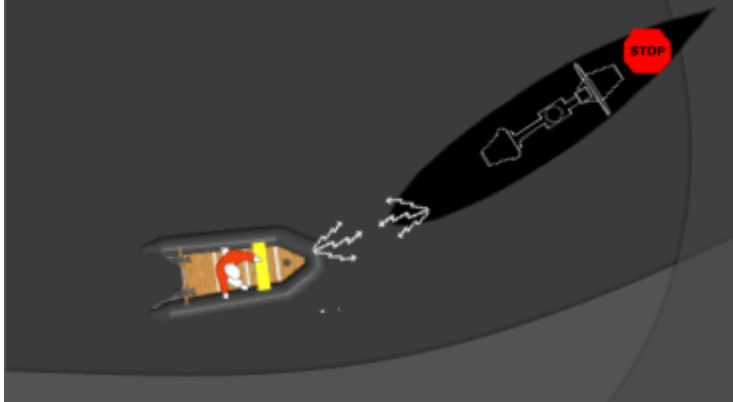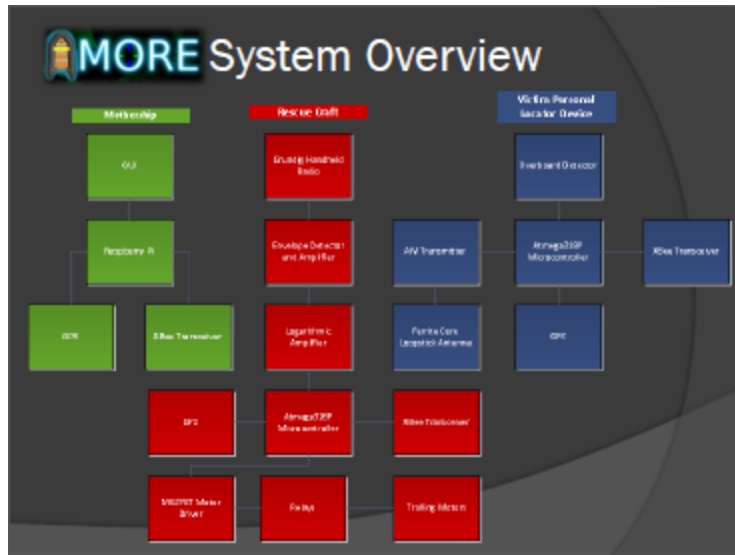
The Rescue Equipment

Mothership Unit

Victim Unit

Rescue Unit

Simulation and Setup
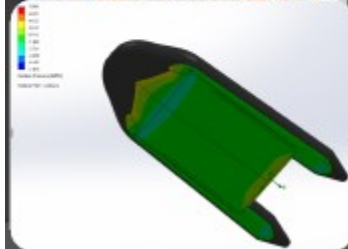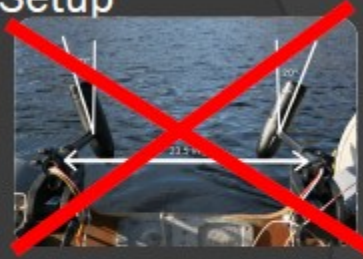
Goal Name	Unit	Value
GG Force (Z) 1	[bf]	2.8
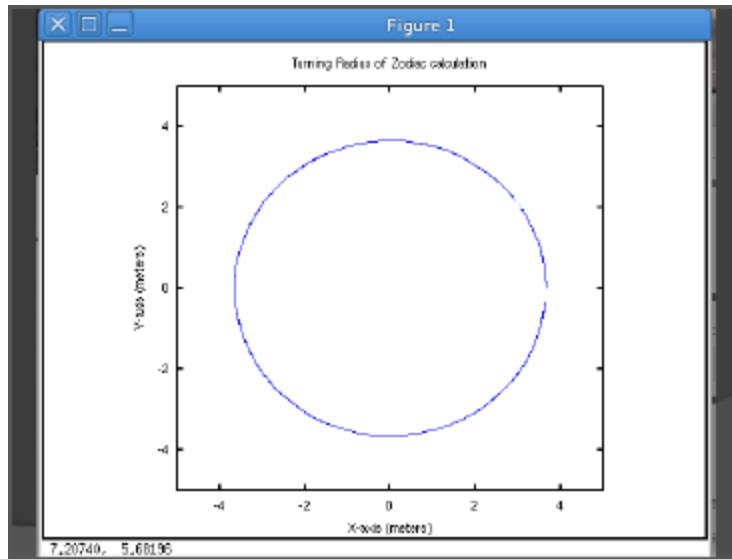
Conclusions

- Constructed a system that utilizes GPS, localized radio-location and magnetic compass
- Linked three independent microcontroller based units via radio to support a coordinated rescue effort
- Used computer analysis for fluid models to anticipate expected boat performance and optimize mechanical aspects
- Integrated a GUI to display real time information on rescue status
- Successfully demonstrated a coordinated operation of the system on the water

Demonstration

- Video Demonstration of the system
- Video of the terminal locator device test
- MQPFinalVideoWithDemo.mp4

Future Work

- Rescue time may be reduced by a factor of 8 utilizing a higher thrust, 8hp motor and more seaworthy craft
- Extended range of transmitter for increased homing capability
- Modular design of personal locator device
- Rescue boat deployment system design
- Code optimization for more advanced tracking and less delay
- More sensitive receivers with noise reduction capabilities

References

- http://fwlearn.net/fw-projects/5138/news/teamounts-project-by-... (illegible URL)
- http://www.cdc.gov/niosh/topics/fishing/
- http://en.wikipedia.org/wiki/Horizon
- http://i1.bp.blogspot.com/-mEzvoqYPoMk/T9_JsjMHErI/AAAAAAAAULo/qZy8ljeNdRQ/s1600/2-1C511MQR1P01
- http://freeoceasnd.rs-online.com/web/generalDisplay.html?id=centre/mro_techref_iprating
- http://www.ladyada.net/make/gpsshield/modules.html
- http://www.adafruit.com/products/957
- http://www.ugusst.net/store/index.php?route=product/product&product_id=78
- http://www.raspberrypi.org/faqs
- http://3.bp.blogspot.com/-edGMdMMeJjc/Tz-O2ID7wII/AAAAAAAAT4/aL8kP-EtA55MWk1600/rough_alalb.jpg
- http://www.amazon.com/Minn-Kota-Freshwater-Transom-Trolling/dp/B00C43dXKY/ref=sr_4_?ie=UTF8&qid=13605517868&sr=1-2&keywords=minn+kota+trolling
- http://www.geek.com/chips/adafruits-transperent-raspberry-pi-case-uses-dragon-claws-1495280/
- Fossen, T. I. (1994). Guidance and Control of Ocean Vehicles. West Sussex, England: John Wiley & Sons Ltd. pgs 206-209

- Furfaro, T. C. (2012). A MODULAR GUIDANCE, NAVIGATION AND CONTROL SYSTEM. Boca Raton, Fl: Florida Atlantic Univerity.

Appendix J – Dimensions for Circuit Tower
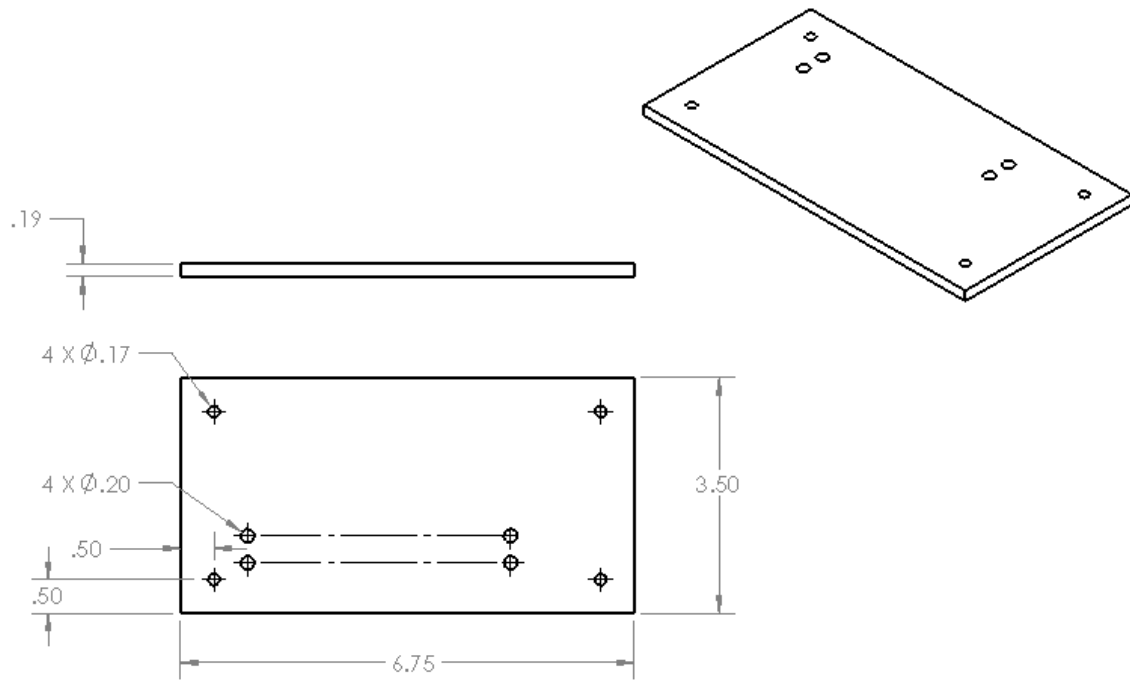
Figure 107 – Level 1 Dimensions

117
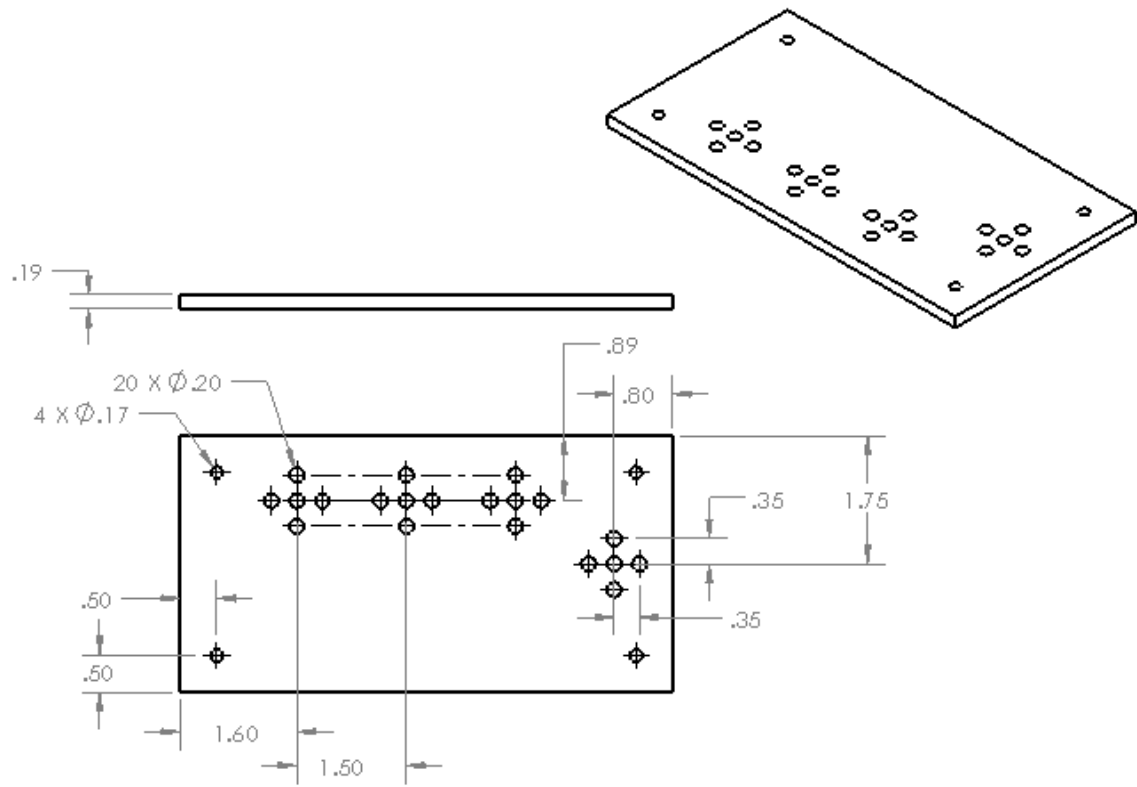
Figure 108 – Level 2Dimensions
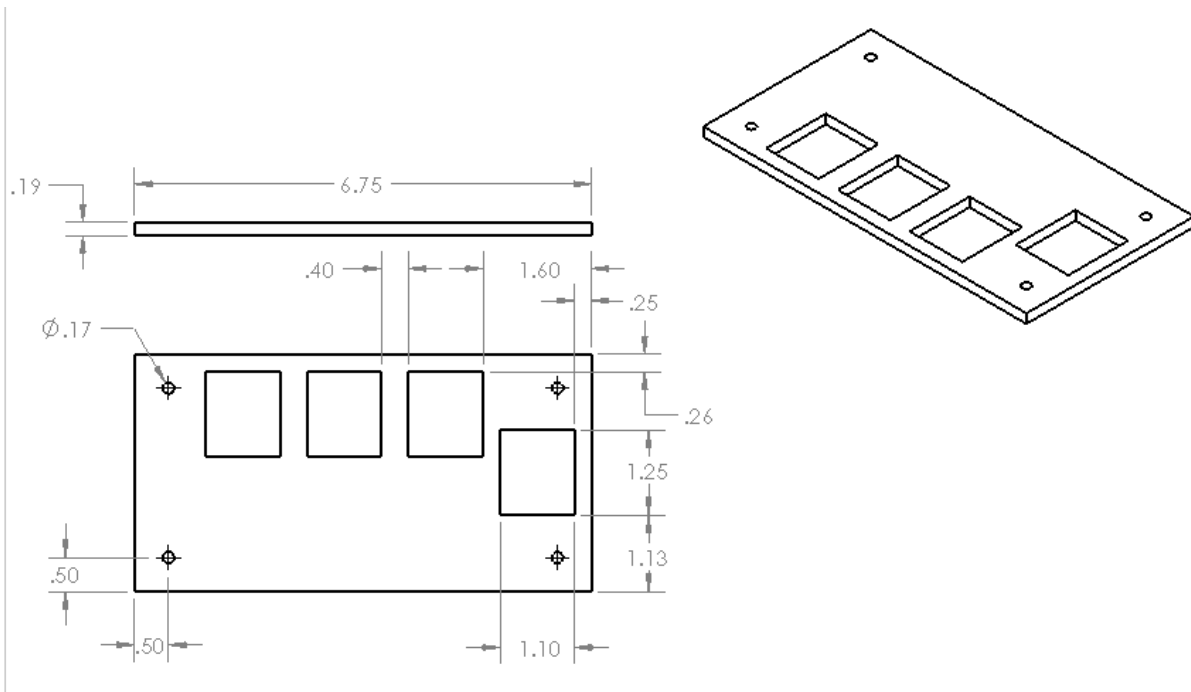
Figure 109 – Level 3 Dimensions

118
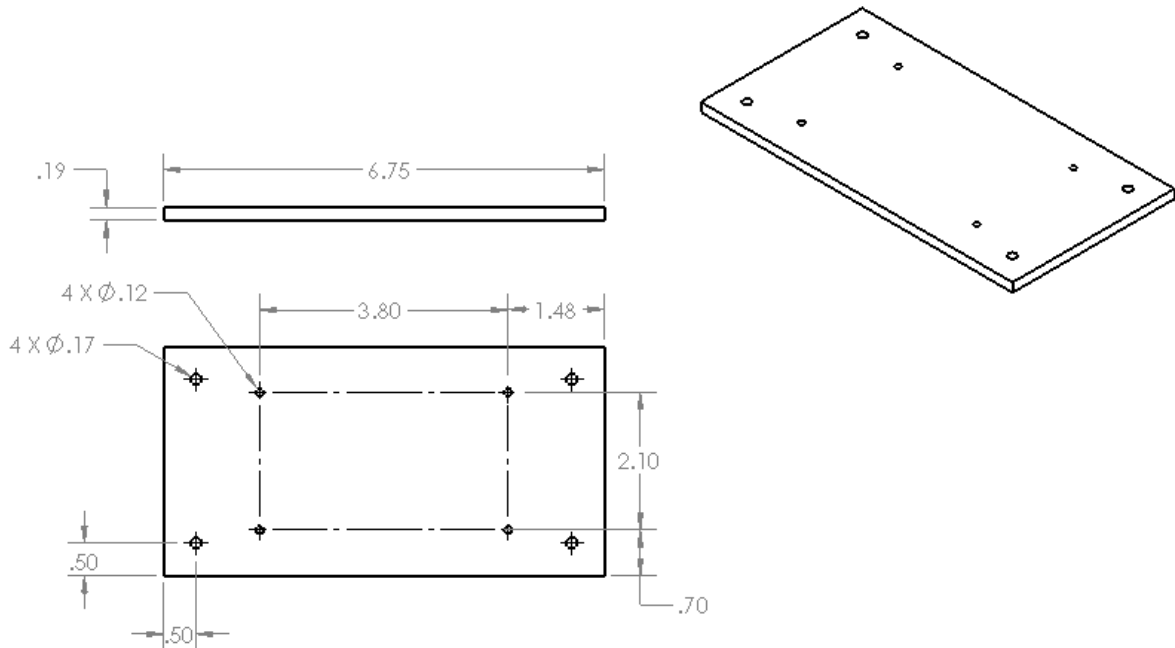
Figure 110 – Level 4 Dimensions

Appendix K – TLD Radio Fine-Tuning

For a successful rescue mission, proper fine-tuning of the receivers is critical. Due to post receiver amplification, any small difference in signal strength can result in major errors in homing in on the victim. For this reason, care should be taken to tune each of the receivers to the proper frequency before carrying out the rescue mission. For the case of this project the receivers were tuned to 922 kHz on the AM band.

Figure 111 – Grundig Radio Receiver used for Terminal Location

Next, with the transmitter broadcasting the 1 kHz tone, the PLD is placed 15 feet from the bow of the boat. A second person adjusts the volume levels of the receivers until the motors react properly as the PLD is moved 2 ft in the left and right direction.
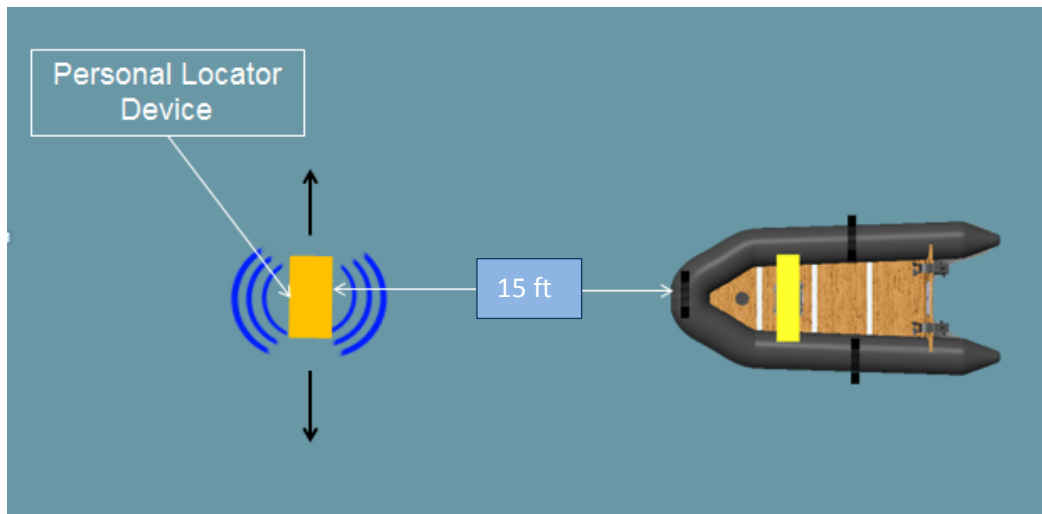
Figure 112 – Terminal Locator Device Radio Tuning Setup

Appendix L – IP Code Chart

Appendix M – Code

Rescue Module Code

```
//The following code is for the Rescue module
//Code written by Frederick Hunter for the Autonomous Man Overboard Rescue Equipment MQP


#include <XBee.h>  //library for sending in API mode with XBees
#include <string.h>
#include <ctype.h> //library for testing characters
#include <SoftwareSerial.h> //library for simulating a serial port
#include <TinyGPS.h> //include for GPS library
#include <Wire.h>

int HMC6352Address = 0x42; // the address of the comapss
int slaveAddress; //the address of the compass slave
byte headingData[2]; //array used to store the incoming data for the compass
float latitudeRescue;
float longitudeRescue;
float lastLat=0.0;
float lastLon;
#define AVG 10 //number of positions to take average from
```

```
float lstCpleLatPos[AVG+2]; //stores the last lat positions for calculating the averagwe
float lstCpleLonPos[AVG+2];
int avgArrayPos = 0; //the position of the lstCpleXXXPos array for updating
int rescueNew = 0; //variable to determine if new coordinates for the rescue module have been
received
int newMes=0; //variable to store whether or not a new message has been recived
String message; //variable to store the message received
float victimLat = 0.0;//lat and lon of the victim
float victimLon;
int victimNew = 0; //bolean to determine if victim coordinates were received
float motherLat=0.0;//lat and lon of the mothership
float motherLon;
int motherNew=0; //boolean to determine if mothership coordinates were received
unsigned long age;
int state = 0; //The current state of the state machine////////////////////change in state 4 also
or remote control restart state
int heading; //The current heading of the rescue module
int accError = 15; //acceptable error in heading to the victim
int desHeading; //the desired heading to go to


float rescueStartLat = 0; //the starting location of the rescue boat
float rescueStartLon = 0;

//To drive the motors forward, A has to be high and B has to be Low
//vice versa for reverse
#define LEFTMOTORF 8 //the left motor relay 1
#define LEFTMOTORR 9 //the left motor relay 2
#define RIGHTMOTORF 10 //the right motor relay 1
#define RIGHTMOTORR 11 //the right motor relay 2
#define GPSPOW 4 //the pin for powering the GPS
#define XBEEPOW 5 //the pin for powering the XBEE
#define BUTTON 6 //the pin for the return home button
#define TLDPOW 7 //the power for the terminal locator device
#define FRONTRECEIVER A2
#define RIGHTRECEIVER A1
#define LEFTRECEIVER A0
#define RADIUS 6372795 //radius of the earth

//XBee address to send to (SH + SL) as defined in X-CTU software
//XBeeAddress64 motherAddr = XBeeAddress64(0x13a200,0x405c2ca0);
//XBeeAddress64 victimAddr = XBeeAddress64(0x13a200,0x4099233c);
XBeeAddress64 victimAddr = XBeeAddress64(0x13a200,0x409f3a9d);
XBeeAddress64 motherAddr = XBeeAddress64(0x13a200,0x406cb54d);


SoftwareSerial GPS(2,3); //RX, TX
XBee XBee = XBee(); //create XBee object
XBeeResponse response = XBeeResponse();
ZBRxResponse rx = ZBRxResponse();
TinyGPS gps;

void setup(){
  //Serial.begin(9600);
  GPS.begin(4800); //set up the serial port for GPS reading (pins 2,3)
  XBee.begin(9600); //print to XBEE (Uses serial ports 1)
  //Serial.begin(9600);
  pinMode(GPSPOW, OUTPUT);
  pinMode(XBEEPOW, OUTPUT);
  pinMode(TLDPOW, OUTPUT); //the power for the terminal locator devices
  pinMode(BUTTON, INPUT);
  pinMode(FRONTRECEIVER,INPUT);
  pinMode(LEFTRECEIVER,INPUT);
  pinMode(RIGHTRECEIVER,INPUT);
  digitalWrite(GPSPOW, LOW);
  digitalWrite(XBEEPOW, LOW);
  digitalWrite(TLDPOW, LOW);
  slaveAddress = HMC6352Address >> 1;    //set the slave address for the compass
  Wire.begin(); //set up the i2c communication for the compass
```

```
    pinMode(LEFTMOTORF, OUTPUT); //set up the motor ports and let them to low to avoid injury
    pinMode(LEFTMOTORR, OUTPUT);
    pinMode(RIGHTMOTORF, OUTPUT);
    pinMode(RIGHTMOTORR, OUTPUT);
    digitalWrite(LEFTMOTORF, LOW);
    digitalWrite(LEFTMOTORR, LOW);
    digitalWrite(RIGHTMOTORF, LOW);
    digitalWrite(RIGHTMOTORR, LOW);
}

/////////////////////////setMotors(int desCourse, int currCourse)///////////////////
//Corrects the motors to correct the course
//desCourse -> The desired course to head to in degrees ex 90 for East
//currCourse -> The current course being traveled
//nextToTarg -> boolean if we are next to the target
void setMotors(int desCourse, int currCourse, boolean nextToTarg){
  //determine error

  int error = desCourse - currCourse;

  /*
  Serial.print("Des Course: ");
  Serial.println(desCourse);
  Serial.print("Curr Course: ");
  Serial.println(currCourse);
  */


  //fix wrap around error
  if(error < -180)
    error = error + 360;
  if(error > 180)
    error = error - 360;
  /*
  Serial.print("Error: ");
  Serial.println(error);
  */
  //correct the course if off
  if(accError < abs(error)){
    if(nextToTarg){
      //if next to victim, we just stop
      sendMess("O,STOP",motherAddr);
      digitalWrite(LEFTMOTORF, LOW);
      digitalWrite(LEFTMOTORR, LOW);
      digitalWrite(RIGHTMOTORF, LOW);
      digitalWrite(RIGHTMOTORR, LOW);
      return;
    }
    if(error > 0){
      /*//if we need to make a hard right turn, make\
      one motor go backwards
      if(error > 90){
        sendMess("O,HARDRIGHT",motherAddr);
        digitalWrite(LEFTMOTORF, HIGH);
        digitalWrite(LEFTMOTORR, LOW);
        digitalWrite(RIGHTMOTORF, LOW);
        digitalWrite(RIGHTMOTORR, HIGH);
        return;
      }*/
      //if the boat should turn right
      sendMess("O,RIGHT",motherAddr);
      digitalWrite(LEFTMOTORF, HIGH);
      digitalWrite(LEFTMOTORR, LOW);
      digitalWrite(RIGHTMOTORF, LOW);
      digitalWrite(RIGHTMOTORR, LOW);
      return;
    }
    else if(error < 0){
      /*//if we need to make a hard left turn, make\
```

```
      one motor go backwards
      if(error < -90){
        sendMess("O,HARDLEFT",motherAddr);
        digitalWrite(LEFTMOTORF, LOW);
        digitalWrite(LEFTMOTORR, HIGH);
        digitalWrite(RIGHTMOTORF, HIGH);
        digitalWrite(RIGHTMOTORR, LOW);
        return;
      }*/
      //else the boat should correct left
      sendMess("O,LEFT",motherAddr);
      digitalWrite(LEFTMOTORF, LOW);
      digitalWrite(LEFTMOTORR, LOW);
      digitalWrite(RIGHTMOTORF, HIGH);
      digitalWrite(RIGHTMOTORR, LOW);
      return;
    }
  }
  //if the course is ok, full throttle if we are not close to the victim
  if(nextToTarg){
    //if next to victim, we just stop
    sendMess("O,STOP",motherAddr);
    digitalWrite(LEFTMOTORF, LOW);
    digitalWrite(LEFTMOTORR, LOW);
    digitalWrite(RIGHTMOTORF, LOW);
    digitalWrite(RIGHTMOTORR, LOW);
  }
  else{
    //otherwise go straight
    sendMess("O,FORWARD",motherAddr);
    digitalWrite(LEFTMOTORF, HIGH);
    digitalWrite(LEFTMOTORR, LOW);
    digitalWrite(RIGHTMOTORF, HIGH);
    digitalWrite(RIGHTMOTORR, LOW);
  }
}

///////////////////////readCompass//////////////////
//retrieves the heading from the compass (0-360 degrees, where 0 = North)
int readCompass(){
  Wire.beginTransmission(slaveAddress);
  Wire.write("A");                // The "Get Data" command
  Wire.endTransmission();
  delay(10);                      // The HMC6352 needs at least a 70us (microsecond) delay
  // after this command.  Using 10ms just makes it safe
  // Read the 2 heading bytes, MSB first
  // The resulting 16bit word is the compass heading in 10th's of a degree
  // For example: a heading of 1345 would be 134.5 degrees
  Wire.requestFrom(slaveAddress, 2);       // Request the 2 byte heading (MSB comes first)
  int i = 0;
  while(Wire.available() && i < 2)
  {
    headingData[i] = Wire.read();
    i++;
  }

  int headingValue = headingData[0]*256 + headingData[1];  // Put the MSB and LSB together
  return headingValue/10; //return the compass heading Output: 0 - 360 where 0 & 360 = North
}

///////////////////////////crosstrackerror////////////////////////////////////
/*Calculates the distance from the perpendicular to a straight line path to a target
 * Distance: dist from current location to goal location
 * theta13: angle from start to current point
 * theta12: angle from start to goal location
 * If return is +, we are on righ on line, if -, we are on left of line
 */
float crossTrackError(float distance, float theta13, float theta12){
  float angleError = radians(theta13-theta12);
```

124

```
    float distRad = distance/RADIUS;
    float sAngleError = sin(angleError);
    float sDistRad = sin(distRad);
    float aSin = asin(sDistRad*sAngleError);
    return aSin*RADIUS;
}

//////////////////////feedgps/////////////////////
//determines if there is data available
static bool feedgps(){
  while (GPS.available()){
    if (gps.encode(GPS.read()))
      return true;
  }
  return false;
}

//retrieves the GPS data into the buffer array
void getGPS(){
  //unsigned long start = millis();
  int i = 0;
  while (i<4/*millis() - start < 1500 && victimNew == 0*/){
    if (feedgps()){
      rescueNew = 1;
      //Serial.println("New Data!");
      gps.f_get_position(&latitudeRescue, &longitudeRescue, &age);
      i++;
    }
  }
  if(rescueNew){
    lstCpleLatPos[avgArrayPos] = latitudeRescue;
    lstCpleLonPos[avgArrayPos] = longitudeRescue;
    //Serial.println(latitudeRescue,6);
    //Serial.println(longitudeRescue,6);
    if(avgArrayPos >= AVG)
      avgArrayPos = 0;
    else
      avgArrayPos++;
  }
}

////////////////////////////sendGPS/////////////////////
//Send the GPS coordinates via XBee to other modules
void sendGPS(XBeeAddress64 module, float latitude, float longitude){
  //if the start position has not been set for the rescue craft, do so now
  if(rescueStartLat == 0 && state != 3){
    //not while state = 3 because there we are also sending the victim coordinates
    rescueStartLat = latitude;
    rescueStartLon = longitude;
  }
  uint8_t payload[30];
  //store as string
  char latc[16];
  char lonc[16];
  dtostrf(latitude,10,5,latc);
  dtostrf(longitude,10,5,lonc);
  String packet = "B,"+(String)latc+","+(String)lonc+",";
  //convert to uint8_t for transmission
  //Serial.println(packet);
  int i;
  for(i = 0; i<packet.length();i++){
    payload[i] = (uint8_t)packet[i];
  }
 //attempt to send the packet
 ZBTxRequest zbTx = ZBTxRequest(module, payload,i);
 XBee.send(zbTx);
 delay(50);
 if(state == 3){
   //if we are heading back, simuklate victim on mothership too
```

125

```
   packet = "C,"+(String)latc+","+(String)lonc+",";
   //convert to uint8_t for transmission
   //Serial.println(packet);
   for(i = 0; i<packet.length();i++){
     payload[i] = (uint8_t)packet[i];
   }
   //attempt to send the packet
   ZBTxRequest zbTx = ZBTxRequest(module, payload,i);
   XBee.send(zbTx);
  }
}

////////////////////////////////readXBee///////////////////
//retrieves the data from another module and returns the string if
//successfull, if not, returns NULL
String readXBee(){
  int i = 0; //check four times in case a message was missed
  for(i=0;i<2;i++){
    newMes = 0;
    XBee.readPacket();
    if(XBee.getResponse().isAvailable()){
      //got some message
      if(XBee.getResponse().getApiId() == ZB_RX_RESPONSE){
        //got a zb rx packet
        //fill the zb rx class
        XBee.getResponse().getZBRxResponse(rx);

        int len = rx.getDataLength(); //number of char received
        char buff [len+2];
        int i = 0;
        for(i = 0; i<len;i++){
          buff[i] = (char)rx.getData(i);
        }
        buff[i+1] = '\0';
        //determine if the received data is a message, or coordinate
        if(buff[0] == 'M' || buff[0] == 'N' || buff[0] == 'O'){
          //sent data is a message
          newMes = 1;
        }
        return buff;


      }
    }
  }
  //else //if no packet received
  return NULL;
}

//////////////////////parseXBeeMess//////////////////////
//parse the message returned from readXBee
//if a message is received, store the message and sender, else return NULL
//if a coordinate was received, store the coordinates
//also update the variable if a new message has been recived or coordinate
//return the message string, null if error, and "coord" if coordinates received
String parseXBeeMess(String mes, char *sender){
  //reset the newMes variable
  *sender = 'Z'; //reset the sender variable
  newMes = 0;
  //determine if a message is being sent
  if(mes.startsWith("M",0) || mes.startsWith("N",0) || mes.startsWith("O",0)){
    //store the message
    newMes = 1;
    //get rid of the start delimiter
    char buf[mes.length()]; //temporary buffer to store the string
    mes.toCharArray(buf,mes.length());
    String ret; //return string
    int i = 2;//get rid of the X, where X, is the sending module
    for(i=2;i<mes.length()-1;i++){
```

```
      //Serial.println(buf[i]);
      ret = ret + buf[i];
    }
    *sender = buf[0]; //return who the sender was
    //Serial.print("In Parse Mess, received: ");
    //Serial.println(ret);
    return ret; //return the message
    //Serial.println(mes[]);
  }
  else{
    //otherwise we will return null after storing the lat/lon positions
    int i = 0;
    int index[4]; //index to store where data begins
    index[0] = '\0'; //empty the array
    index[1] = '\0';
    index[2] = '\0';
    index[3] = '\0';
    index[4] = '\0';
    char c; //stores each character in the string array
    int j =0; //keeps the place in the index array
    char buf[mes.length()+2]; //buffer for converting the string to a char array

    if(mes.startsWith("A",0)){
      //coordinates received from the mothership unit
      motherNew = 1; //let program know a new coordinate was recived
      //get the coordinates
      for(i=0;i<mes.length();i++){
        //get the character at the ith element
        c = mes.charAt(i);
        if(c == ','){
          //if a comma is found, remember where it is
          index[j] = i;
          j++;
          buf[i] = '\0'; // replace with end of string char
        }
        else{
          buf[i] = c; //store the character into the buffer
        }
      }
      //store the lat pos
      char coordBuf[10]; // buffers for storing the individual lat and lon pos
      char coordBuf2[10];
      for(i=index[0]+1;i<index[1];i++){
        //store the lat position in a char buffer
        coordBuf[i-(index[0]+1)] = buf[i];
      }
      coordBuf[i+1] = '\0'; //null terminator
      motherLat = atof(coordBuf); //convert to long and store to memory

      for(i=index[1]+1;i<index[2];i++){
        //store the lon position to buffer
        coordBuf2[i-(index[1]+1)] = buf[i];
      }
      coordBuf2[i+1] = '\0'; //null terminator
      motherLon = atof(coordBuf2); //convert to long and store

      *sender=buf[0]; //tell who the sender was
      return NULL;
    }
    if(mes.startsWith("C",0)){
      //coordinates received from the victim unit
      victimNew = 1; //let program know new coordinates have been received
      for(i=0;i<mes.length();i++){
        //get the character at the ith element
        c = mes.charAt(i);
        if(c == ','){
          //if a comma is found, remember where it is
          index[j] = i;
          j++;
```

```
          buf[i] = '\0'; // replace with end of string char
        }
        else{
          buf[i] = c; //store the character into the buffer
        }
      }
      //store the lat pos
      char coordBuf[10]; // buffers for storing the individual lat and lon pos
      char coordBuf2[10];
      for(i=index[0]+1;i<index[1];i++){
        //store the lat position in a char buffer
        coordBuf[i-(index[0]+1)] = buf[i];
      }
      coordBuf[i+1] = '\0'; //null terminator
      victimLat = atof(coordBuf); //convert to long and store to memory

      for(i=index[1]+1;i<index[2];i++){
        //store the lon position to buffer
        coordBuf2[i-(index[1]+1)] = buf[i];
      }
      coordBuf2[i+1] = '\0'; //null terminator
      victimLon = atof(coordBuf2); //convert to long and store
      *sender =buf[0]; //tell who the sender was
      return NULL;
    }
    //otherwise if no message has been sent, return null
    *sender =NULL; //if no message or coordinates were received
    return NULL;
  }
}
/*Calculates the average of the last 25 positions
for the given module, unless the module doesn't have
that many past recorded values in the avg array
to calculate latitude, set lat to true
for longitud, set lat to false*/
float calcAverage(boolean lat){
  if(lstCpleLatPos[0] == '\0'){
    return 0; //return 0 if nothing in array
  }
  int i;
  float avg = 0.00000;
  if(lat == true){
    for(i = 0; i<AVG; i++){
      if(lstCpleLatPos[i] == '\0')
        break; //if reached end of array, break
      avg = avg + lstCpleLatPos[i];
      //Serial.println(module.lst_cple_lat_pos[i]);
    }
  }
  else{
    for(i = 0; i<AVG; i++){
      if(lstCpleLonPos[i] == '\0')
        break; //if reached end of array, break
      avg = avg + lstCpleLonPos[i];
      //Serial.println(module.lst_cple_lon_pos[i]);
    }
  }
  if(i ==0)
    return 0;
  else{
    //Serial.println(avg,5);
    //Serial.print("Average: ");
    //Serial.println(avg/(float)i,5);
    return avg / (float)i;
  }
}

////////////////////////////resetAvg////////////////////////
//resets the avg buffer
```

```
void resetAvg(){
  int j = 0;
  for(j=0;j<AVG;j++){
    lstCpleLonPos[j] = '\0';
    lstCpleLatPos[j]= '\0';
  }
  avgArrayPos = 0; //update the average array position index
}

/////////////////////////sendMess/////////////////////////
//this function is used to send a message to the mothership that the victim is in danger
void sendMess(String message, XBeeAddress64 module){
  uint8_t payload[30];
  //convert message to uint8_t for transmission
  int i ;
  //Serial.println(message);
  //Serial.println(message.length());
  for(i = 0; i<message.length();i++){
    payload[i] = (uint8_t)message[i];
  }
  //Serial.println(message);
  //attempt to send the packet
  ZBTxRequest zbTx = ZBTxRequest(module, payload, i);
  XBee.send(zbTx); //send the message
}
void resetArray(float array[]){
  int i = 0;
  for(i = 0; i<AVG+2;i++){
    array[i] = '\0';
  }
}

/////////////////////////readReceivers///////////7/
//retunrs a negative if we need to go left and a positive to go right
float readReceivers(){
  int loops = 500; //number of loops to go through
  int i=0;
  float leftAVG = 0;
  float rightAVG = 0;
  float frontAVG = 0;
  for(i = 0;i<loops;i++){
    leftAVG = leftAVG + analogRead(LEFTRECEIVER);
    rightAVG = rightAVG + analogRead(RIGHTRECEIVER);
    //frontAVG = frontAVG + analogRead(FRONTRECEIVER);
  }
  leftAVG = leftAVG/loops;
  rightAVG = rightAVG/loops;

  //frontAVG = frontAVG/50;
  if(rightAVG < 15 || leftAVG < 15){
    //if the signal is too low, go to state 2
    sendMess("O,OUT OF RANGE!!(GO_STATE1)",motherAddr);
    state = 1;
  }
  //TODO://determine if we should go straight

  return rightAVG-leftAVG; //want a negative for a desired left turn
}
/////////////////////////checkSignalStrength/////////////////
//retunrs the average signal reading of the front receiver
float checkSignalStrength(){
  int i=0;
  float frontAVG=0;
  float leftAVG =0;
  float rightAVG=0;
  for(i=0;i<50;i++){
    //frontAVG=frontAVG+analogRead(FRONTRECEIVER);
    leftAVG = leftAVG + analogRead(LEFTRECEIVER);
    rightAVG = rightAVG + analogRead(RIGHTRECEIVER);
```

129

```
  }
  if(leftAVG > rightAVG)
    return rightAVG/50;
  else
    return leftAVG/50;
}


//////////////////////////////////Loop//////////////////////////
void loop(){
  boolean leftOn = false;
  boolean rightOn = false;
  //Reset the arrays for calculating the averages
  //resetArray(lstCpleLatPos);
  //resetArray(lstCpleLonPos);
  String rv; //the return value of the XBee
  char sender; //variable to hold the sender

    // 0 = SLEEP (Wait for mothership to wake me up)
    // 1 = HEADING TO VICTIM (send GPS coord to mothership & check if we are within 50m of the
victim)
    // 2 = HEADING TO VICTIM (within 50m of victim, Activate the receiver and follow that, send
GPS and notify we are close)
    // 3 = HEADING TO MOTHERSHIP (Victim retrieved and heading back to the mothership)
  long distToTarget = 0; //the distance to the target
  while(1){
    while(state == 0){
      //initialize things for state 0
      //turn the GPS off and turn the XBEE on
      digitalWrite(GPSPOW, LOW);
      digitalWrite(XBEEPOW, HIGH);
      resetAvg();
      delay(1000);
      while(state == 0){
        //waiting for the mothership to send a wakeup signal
        rv = readXBee();
        if(rv != NULL && newMes == 1){
          //mesage received
          message = parseXBeeMess(rv,&sender);
          //if message from mothrship
          if(message.equals("STOP") && sender == 'M'){
            //if the kill switch has been thrown
            setMotors(0,0,true); //stop the motors
            state = 4;//go to state 4 for remote control
            break; //break out of loop
          }
          if(message.equals("BOOT") && sender == 'M'){
            //signal received
            sendMess("O,INIT",motherAddr);
            digitalWrite(GPSPOW,HIGH);
            //wait for GPS to get a lock
            while(rescueNew == 0){
              rescueNew= 0;
              latitudeRescue = 0;
              longitudeRescue = 0;
              //get the gps data
              getGPS();
              rv = readXBee();
              if(rv != NULL && newMes == 1){
                //mesage received
                message = parseXBeeMess(rv,&sender);
                //if message from mothrship
                //Serial.println(message);
                if(message.equals("STOP") && sender == 'M'){
                  //if the kill switch has been thrown
                  setMotors(0,0,true); //stop the motors
                  state = 4;//go to state 4 for remote control
                   sendMess("O,State 4",motherAddr);
                  break; //break out of loop
```

```
              }
            }
          }
          //tell the mothership, the search for the victim has begun
          sendMess("O,SEARCHV",motherAddr);
          state = 1;
          //Turn on the GPS in case it was off
          digitalWrite(GPSPOW,HIGH);
          break;
        }
      }
    }
  }

while(state == 1){
  //Serial.println("Sate1");
  //heading to the victim following the gps coordinates from the victim
  sendMess("O,Rescue STATE 1",motherAddr);
  delay(50);
  sendMess("O,BOOM!",motherAddr);
  resetAvg();
  resetArray(lstCpleLatPos);
  resetArray(lstCpleLonPos);
  //Activate the transmitter on the victim module
  sendMess("O,CLOSEV",victimAddr);
  while(state == 1){//75 was good
    if(checkSignalStrength()>30){
      //determine if there is a signal present from tld
      setMotors(0,0,true);//////////////////////////
      state = 2;
    }
    //reset variables
    rescueNew= 0;
    latitudeRescue = 0;
    longitudeRescue = 0;
    //get the gps data
    getGPS();
    //send the gps data to the mothership if new data received
    if(rescueNew){
      //send the average (10 pos) of the lat and lon positions respectively
      //Serial.println("New rescue coord!");
      lastLat = calcAverage(true);
      lastLon = calcAverage(false);
      //Serial.println(lastLat);
      //Serial.println(lastLon);
      //sendMess("O,Sending coord",motherAddr);
      //(50);
      sendGPS(motherAddr, lastLat, lastLon);
      rescueNew=0; //reset the variable
    }
    //retrieve the coordinates from the victim
    rv = readXBee();
    if(rv != NULL){
      //Received a Coordinate
      message = parseXBeeMess(rv, &sender); //store the coordinates
      if(message.equals("STOP") && sender == 'M'){
        //if the kill switch has been thrown
        setMotors(0,0,true); //stop the motors
        state = 4;//go to state 4 for remote control
        break; //break out of loop
      }
      if(victimLat > 1){
        //if coordinates from the victim received
        victimNew = 0;
        //calculate the distance to the victim
        distToTarget = gps.distance_between(victimLat, victimLon, lastLat,lastLon);
        if(distToTarget < 30){
          //if we are close to the victim, switch to state 2
          if(checkSignalStrength()>30){ //75 was good
```
131

```
                 //determine if there is a signal present
                 setMotors(0,0,true);////////////////////////////
                 state = 2;
               }

            }
          }
        }
        //if some coordinates from the victim and rescue boat were stored, correct direction
        if(victimLat != 0 && lastLat){
          //get desired heading and get the current heading & pass to the set motors function
          int cou = (int)gps.course_to(lastLat,lastLon,victimLat,victimLon);
          //int cou = (int)gps.course_to(victimLat,victimLon,lastLat,lastLon);
           int currcourse = readCompass();
          float CTE =
crossTrackError(distToTarget,gps.course_to(rescueStartLat,rescueStartLon,victimLat,victimLon),gps
.course_to(rescueStartLat,rescueStartLon,lastLat,lastLon)); //find the cross track error
          if(CTE > 5)
            cou = cou +(accError/2); //make boat correct to be within 5m of straight line path to
victim
          if(CTE < -5)
            cou = cou -(accError/2); //need to correct right more
          setMotors(cou,currcourse, false);
        }
      }
    }

  }

  while(state == 2){
    //within 50 meters of the victim and activating the receiver to follow that
    //Activate the transmitter on the victim module
    sendMess("O,CLOSEV",victimAddr);


    setMotors(0,0,true);////////////////////////////


    delay(100); //allow message to send
    sendMess("O,CLOSEV", motherAddr); //tell mothership we are close
    //activate the recievers on this module
    digitalWrite(TLDPOW, HIGH);
    delay(100); //allow the receivers to activate
    //home in on the victim
    while(state == 2){
      //sendMess("O,State2 waiting", motherAddr); //tell mothership we are close
      //once the victim has been retrieved and button pressed, send message to mothership & go
to state 3
      if(digitalRead(BUTTON)==HIGH){
        setMotors(0,0,true); //stop the motors
        delay(50);
        sendMess("O,RETURN",motherAddr); //tell mothership we are heading back
        delay(50);
        sendMess("O,RETURN",victimAddr); //tell mothership we are heading back
        state = 3;
        break;
      }
      rv = readXBee();
      if(rv != NULL){
        //Received a Coordinate
        message = parseXBeeMess(rv, &sender); //store the coordinates
        //see if mothership has sent a message
        if(message.equals("STOP") && sender == 'M'){
          //if the kill switch has been thrown
          setMotors(0,0,true); //stop the motors
          state = 4;//go to state 4 for remote control
          break; //break out of loop
        }
      }
```

132

```
    //correct the heading here and set motors
    //ALSO need to tell the motors when we are next to the victim to turn them off!!
    //determine the readings from the receivers
    float go = readReceivers();
    if(go == 0){
      //if we need to go straight
      sendMess("O,FORWARD",motherAddr);
      digitalWrite(LEFTMOTORF, HIGH);
      digitalWrite(LEFTMOTORR, LOW);
      digitalWrite(RIGHTMOTORF, HIGH);
      digitalWrite(RIGHTMOTORR, LOW);
    }


    else if(go<0){
      //if negative, go left
      sendMess("O,LEFT",motherAddr);
      digitalWrite(LEFTMOTORF, LOW);
      digitalWrite(LEFTMOTORR, LOW);
      digitalWrite(RIGHTMOTORF, HIGH);
      digitalWrite(RIGHTMOTORR, LOW);
    }
    else if(go>0){
      //if positive, go right
      sendMess("O,RIGHT",motherAddr);
      digitalWrite(LEFTMOTORF, HIGH);
      digitalWrite(LEFTMOTORR, LOW);
      digitalWrite(RIGHTMOTORF, LOW);
      digitalWrite(RIGHTMOTORR, LOW);
    }
    //need to determine if we are close to the victim and stop
    if(checkSignalStrength()>=550){
      //we are next to the victim, so stop and wait to be picked up
      setMotors(0,0,true); //stop the motors
      sendMess("O,WAITING FOR VICTIM TO BOARD",motherAddr);
      while(1){
        if(digitalRead(BUTTON)==HIGH){
          setMotors(0,0,true); //stop the motors
          delay(50);
          sendMess("O,RETURN",motherAddr); //tell mothership we are heading back
          delay(50);
          sendMess("O,RETURN",victimAddr); //tell mothership we are heading back
          state = 3;
          break;
        }
      }
    }
    delay(500);
    //TODO: we need to determine if e need to turn around
    //TODO: Need to account for error

  }

}

while(state == 3){
  //got the victim, return to the mothership
  //tell mothership to send it's gps location
  sendMess("O,MCOORD", motherAddr);
  delay(100);
  sendMess("O,3 BOOM!",motherAddr);
  resetAvg();
  resetArray(lstCpleLatPos);
  resetArray(lstCpleLonPos);
  while(1){
    //send and store this modules coordinates
    //reset variables
    rescueNew= 0;
    latitudeRescue = 0;
```

133

```
          longitudeRescue = 0;
          //get the gps data
          getGPS();
          //send the gps data to the mothership if new data received
          if(rescueNew){
            //send the average (10 pos) of the lat and lon positions respectively
            lastLat = calcAverage(true);
            lastLon = calcAverage(false);
            sendGPS(motherAddr, lastLat, lastLon);
            rescueNew=0; //reset the variable
          }

          //retrieve coordinates from the mothership of its location and store them
          rv = readXBee();
          if(rv != NULL){
            //Received a Coordinate
            message = parseXBeeMess(rv, &sender); //store the coordinates
            if(message.equals("STOP") && sender == 'M'){
              //if the kill switch has been thrown
              setMotors(0,0,true); //stop the motors
              state = 4;//go to state 4 for remote control
              break; //break out of loop
            }
            if(motherNew && message == NULL){
              //if coordinates from the victim received
              motherNew = 0;
              //calculate the distance to the victim
              distToTarget = gps.distance_between(motherLat, motherLon, lastLat,lastLon);
              if(distToTarget < 30){
                //if we are close to the mother, switch to state 0 and notify crew
                sendMess("O,HOME",motherAddr);
                delay(10000);
                state = 0;
                break;
              }
            }
          }

          //plan angle to go to mothership
          //correct heading
          //if some coordinates from the victim and rescue boat were stored, correct direction
          if(motherLat != 0 && lastLat){
            //get desired heading and get the current heading & pass to the set motors function
            setMotors((int)gps.course_to(lastLat,lastLon,motherLat,motherLon),readCompass(),
false);
          }
        }
      }
    }

    while(state == 4){
      //state for remote control operation of the boat
      rv = readXBee();
      if(rv != NULL && newMes == 1){
        //check if the message got sent to restart
        message = parseXBeeMess(rv, &sender); //store the coordinates
        if(message.equals("RESTART") && sender == 'M'){
          //if the kill switch has been thrown
          setMotors(0,0,true); //stop the motors
          state = 1;//go to state 0 for restart/////////////////////////////////////////
          break; //break out of loop
        }
        //mesage received
        if(message.equals("LO")){
          Serial.println("Turning right");
          leftOn = 1;
          rightOn = 0;
        }
        if(message.equals("SS")){
          Serial.println("Stop");
```

```
      leftOn = 0;
      rightOn = 0;
    }
    if(message.equals("RO")){
      Serial.println("Turning left");
      rightOn = 1;
      leftOn = 0;
    }
    if(message.equals("FF")){
      Serial.println("Going straight");
       digitalWrite(LEFTMOTORF, HIGH);
      digitalWrite(LEFTMOTORR, LOW);
      digitalWrite(RIGHTMOTORF, HIGH);
      digitalWrite(RIGHTMOTORR, LOW);
      rightOn = 4; //just don't do anything
      leftOn = 4;
    }
    if(message.equals("BB")){
      //go backwards
      digitalWrite(LEFTMOTORF, LOW);
      digitalWrite(LEFTMOTORR, HIGH);
      digitalWrite(RIGHTMOTORF, LOW);
      digitalWrite(RIGHTMOTORR, HIGH);
       rightOn = 3;
      leftOn = 3;
    }
  }
  if(leftOn == 1){
    //turn on left motor
    digitalWrite(LEFTMOTORF, HIGH);
    digitalWrite(LEFTMOTORR, LOW);
    digitalWrite(RIGHTMOTORF, LOW);
    digitalWrite(RIGHTMOTORR, LOW);
  }
  if(rightOn == 1){
    //turn on right motor
    digitalWrite(LEFTMOTORF, LOW);
    digitalWrite(LEFTMOTORR, LOW);
    digitalWrite(RIGHTMOTORF, HIGH);
    digitalWrite(RIGHTMOTORR, LOW);
  }
  if(leftOn == 0){
    //turn off left motor
    digitalWrite(LEFTMOTORF, LOW);
    digitalWrite(LEFTMOTORR, LOW);
  }
  if(rightOn == 0){
    //turn off right motor
    digitalWrite(RIGHTMOTORF, LOW);
    digitalWrite(RIGHTMOTORR, LOW);
  }
  if(rightOn == 3 && leftOn == 3){
     //reverse
     digitalWrite(LEFTMOTORF, LOW);
    digitalWrite(LEFTMOTORR, HIGH);
    digitalWrite(RIGHTMOTORF, LOW);
    digitalWrite(RIGHTMOTORR, HIGH);
  }

}

while(state == 5){
  //testing state
  rv = readXBee();
    if(rv != NULL){
      //Received a packet
      //Serial.println("Packet received");
      message = parseXBeeMess(rv, &sender); //store the coordinates
      if(motherNew && message == NULL){
```

```
                //if coordinates from the victim received
                motherNew = 0;
                Serial.println("Lat and llon pos:");
                Serial.println(motherLat);
                Serial.println(motherLon);
                //calculate the distance to the victim
                //distToTarget = calcDist(motherLat, motherLon, lastLat,lastLon);
                if(distToTarget < 50){
                  //if we are close to the mother, switch to state 0 and notify crew
                  sendMess("O,HOME",motherAddr);
                  state = 0;
                }
            }
          }
        }
      }
}
```

Victim Unit Code

```
//The following code is for the Victim module written by Frederick Hunter for the Autonomous Man
Overboard Rescue Equipment MQP


#include <XBee.h>
#include <string.h>
#include <ctype.h> //library for testing characters
#include <SoftwareSerial.h>
#include <TinyGPS.h> //include for GPS libraries

float longitudeVictim;
float latitudeVictim;
unsigned long age;
#define AVG 10 //number of positions to take average from
float lstCpleLatPos[AVG+2]; //stores the last lat positions for calculating the averagwe
float lstCpleLonPos[AVG+2];
int avgArrayPos = 0; //the position of the lstCpleXXXPos array for updating
int victimNew = 0; //variable to determine if new coordinated for the victim module were recieved
int newMes = 0; //variable to determine if a new message has been received
String message; //variable to hold the message received
TinyGPS gps;

#define thisModule 'C' //Module identifier for this module A = Mother B = Rescue C = Victim
#define GPSPOW 4 //the pin for powering the GPS
#define XBEEPOW 5 //the pin for powering the XBEE
#define WATERSENSOR 6 //the sensor for detecting if the person fell in the water
#define TLD 7 //the terminal locator device can be turned on / off with this pin

//XBee address to send to (SH + SL) as defined in X-CTU software
//XBeeAddress64 rescueAddr = XBeeAddress64(0x13a200,0x405c2ca0);
XBeeAddress64 rescueAddr = XBeeAddress64(0x13a200,0x4099233c);
XBeeAddress64 motherAddr = XBeeAddress64(0x13a200,0x406cb54d);
//XBeeAddress64 motherAddr = XBeeAddress64(0x13a200,0x405c2ca0);



SoftwareSerial GPS(2,3); //RX, TX
XBee XBee = XBee(); //create XBee object
XBeeResponse response = XBeeResponse();
ZBRxResponse rx = ZBRxResponse();

//Setup script
void setup(){
  pinMode(GPSPOW, OUTPUT);
  pinMode(XBEEPOW, OUTPUT);
  pinMode(TLD, OUTPUT);
  pinMode(WATERSENSOR, INPUT);
  digitalWrite(GPSPOW, LOW);
```

```
   digitalWrite(XBEEPOW, LOW);
 //Serial.begin(9600);  //print to screen
 GPS.begin(4800); //read GPS
 XBee.begin(9600); //print to XBEE (Uses serial ports 1)
}


/////////////////////////////sendMess///////////////////////////
//this function is used to send a message to the mothership that the victim is in danger
void sendMess(String message, XBeeAddress64 module){
  uint8_t payload[30];
  //convert message to uint8_t for transmission
  int i ;
  for(i = 0; i<message.length();i++){
    payload[i] = (uint8_t)message[i];
  }
  //attempt to send the packet
  ZBTxRequest zbTx = ZBTxRequest(module, payload, i);
  XBee.send(zbTx); //send the message
}
///////////////////////////////readXBee//////////////////
//retrieves the data from another module and returns the string if
//successfull, if not, returns NULL
String readXBee(){
  int i = 0;
  for(i=0;i<4;i++){
    newMes = 0;
    XBee.readPacket();
    if(XBee.getResponse().isAvailable()){
      //got some message
      if(XBee.getResponse().getApiId() == ZB_RX_RESPONSE){
        //got a zb rx packet
        //fill the zb rx class
        XBee.getResponse().getZBRxResponse(rx);

        int len = rx.getDataLength(); //number of char received
        char buff [len+2];
        int i = 0;
        for(i = 0; i<len;i++){
          buff[i] = (char)rx.getData(i);
        }
        buff[i+1] = '\0';
        //determine if the received data is a message, or coordinate
        if(buff[0] == 'M' || buff[0] == 'N' || buff[0] == 'O'){
          //sent data is a message
          newMes = 1;
        }
        return buff;
      }
    }
  }
  //else //if no packet received
  return NULL;
}



//////////////////////parseXBeeMess///////////////////
//parse the message returned from readXBee
//if a message is received, store the message and sender, else return NULL
//if a coordinate was received, store the coordinates
//also update the variable if a new message has been recived or coordinate
//return the message string, null if error, and "coord" if coordinates received
String parseXBeeMess(String mes, char *sender){
  //reset the newMes variable
  *sender = 'Z'; //reset the sender variable
  newMes = 0;
  //determine if a message is being sent
  if(mes.startsWith("M",0) || mes.startsWith("N",0) || mes.startsWith("O",0)){
    //store the message
    newMes = 1;
```

```
    //get rid of the start delimiter
    char buf[mes.length()]; //temporary buffer to store the string
    mes.toCharArray(buf,mes.length());
    String ret; //return string
    int i = 2;//get rid of the X, where X, is the sending module
    for(i=2;i<mes.length()-1;i++){
      //Serial.println(buf[i]);
      ret = ret + buf[i];
    }
    *sender = buf[0]; //return who the sender was
    //Serial.print("In Parse Mess, received: ");
    //Serial.println(ret);
    return ret; //return the message
    //Serial.println(mes[]);
  }
  //else //otherwise if no message has been sent, return null
    return NULL;
}
//////////////////////feedgps/////////////////////
//determines if there is data available
static bool feedgps()
{
  while (GPS.available())
  {
    if (gps.encode(GPS.read()))
      return true;
  }
  return false;
}

//retrieves the GPS data into the buffer array
void getGPS(){
  //unsigned long start = millis();
  int i = 0;
  while (i<4/*millis() - start < 1500 && victimNew == 0*/){
    if (feedgps()){
      victimNew = 1;
      //Serial.println("New Data!");
      gps.f_get_position(&latitudeVictim, &longitudeVictim, &age);
      i++;
    }
  }
  if(victimNew){
    lstCpleLatPos[avgArrayPos] = latitudeVictim;
    lstCpleLonPos[avgArrayPos] = longitudeVictim;
    //Serial.println(latitudeVictim,6);
    //Serial.println(longitudeVictim,6);
    if(avgArrayPos >= AVG)
      avgArrayPos = 0;
    else
      avgArrayPos++;
  }
}

/////////////////////////////sendGPS/////////////////////
//Send the GPS coordinates via XBee to other modules
void sendGPS(){
  uint8_t payload[30];
  //determine if more data sent
  if(victimNew){ //make sure new data has been recieved before printing
      //getAverage pos
      float avgLat = calcAverage(true);
      float avgLon = calcAverage(false);
      //Serial.println(avgLat,5);
      //Serial.println(avgLon,5);
      //Store as string

      char latc[16];
      dtostrf(avgLat,10,5,latc);
```

138

```
        char lonc[16];
        dtostrf(avgLon,10,5,lonc);
        //Serial.println(lonc);
        //Serial.println(latc);
        //String lat = String(latc);
        //String lon = String(lonc);
        String packet = "C,"+String(latc)+","+String(lonc)+",";//ltoa(latitudeVictim, buf, 10) +
ltoa(longitudeVictim, buf, 10);
        //convert to uint8_t for transmission
        //Serial.println(packet);
        int i;
        for(i = 0; i<packet.length();i++){
          payload[i] = (uint8_t)packet[i];
        }
      //attempt to send the packet to rescue
      ZBTxRequest zbTx = ZBTxRequest(rescueAddr, payload,i);
      XBee.send(zbTx);
      delay(50);
      //XBee.flush();
      //attempt to send packet to mothership
      ZBTxRequest zbTx2 = ZBTxRequest(motherAddr, payload,i);
      XBee.send(zbTx2);
      //delay(50);
      //XBee.flush();
    }
}


/*Calculates the average of the last 25 positions
for the given module, unless the module doesn't have
that many past recorded values in the avg array
to calculate latitude, set lat to true
for longitud, set lat to false*/
float calcAverage(boolean lat){
  if(lstCpleLatPos[0] == '\0'){
    return 0; //return 0 if nothing in array
  }
  int i;
  float avg = 0.00000;
  if(lat == true){
    for(i = 0; i<AVG; i++){
      if(lstCpleLatPos[i] == '\0')
        break; //if reached end of array, break
      avg = avg + lstCpleLatPos[i];
      //Serial.println(module.lst_cple_lat_pos[i]);
    }
  }
  else{
    for(i = 0; i<AVG; i++){
      if(lstCpleLonPos[i] == '\0')
        break; //if reached end of array, break
      avg = avg + lstCpleLonPos[i];
      //Serial.println(module.lst_cple_lon_pos[i]);
    }
  }
  if(i ==0)
    return 0;
  else{
    //Serial.println(avg,5);
    //Serial.print("Average: ");
    //Serial.println(avg/(float)i,5);
    return avg / (float)i;
  }
}

/////////////////////////resetAvg/////////////////////////
//resets the avg buffer
void resetAvg(){
  int j = 0;
  for(j=0;j<AVG;j++){
```

```
      lstCpleLonPos[j] = '\0';
      lstCpleLatPos[j]= '\0';
  }
  avgArrayPos = 0; //update the average array position index
}


void resetArray(float array[]){
  int i = 0;
  for(i = 0; i<AVG+2;i++){
    array[i] = '\0';
  }
}



void loop(){
  int state = 0;   //states for the victim module
      //0 = victim is not in danger, still on mothership
      //1 = victim is in distess, fell off the mothership
      //2 = victim is close to the rescue craft
      //3 = victim is on the rescue craft
  //reset the arrays for calculating the averages
  String rv; //string received
  char sender; //variable to store who the sender is
  resetArray(lstCpleLatPos);
  resetArray(lstCpleLonPos);
  while(1){
    while(state == 0){
      resetAvg();
      //Serial.println("State 0");
      while(state == 0){
        //keep polling the WATERSENSOR to check if the victim is in danger
        if(digitalRead(WATERSENSOR)){
          //if in danger, send distress signal to the mothership, activate GPS and XBEE
          digitalWrite(GPSPOW, HIGH); //turn on the GPS
          digitalWrite(XBEEPOW,HIGH); //turn on the XBEE
          delay(1000); //allow XBEE to Power on
          //send distress signal to Mothership
          sendMess("N,OVERBOARD", motherAddr);
          //wait for GPS to get a lock
          delay(50);
          while(victimNew == 0){
            victimNew = 0;
            latitudeVictim =0;
            longitudeVictim=0;
            getGPS(); //get GPS data
            //sendGPS();
            //victimNew = 1;//////////////////////////////////////
          }
          sendMess("N,INITDN",motherAddr);
          //delay(50);
          //rv = readXBee();
          //resetAvg();
          state = 1; //now in danger state
        }
      }
    }
    while(state == 1){
      resetAvg();
      resetArray(lstCpleLatPos);
      resetArray(lstCpleLonPos);
      //Serial.println("State = 1");
      sendMess("N,STATE1",motherAddr);
      delay(50);
      sendMess("N,AHHHH! HELP ME!",motherAddr);
      while(state == 1){
        //victim is in danger, send GPS coordinates
        //reset variable
        victimNew = 0;
        latitudeVictim =0;
```

140

```
      longitudeVictim=0;
      getGPS(); //get GPS data
      sendGPS(); //send the GPS data via GPS
    //check if the rescue craft is close, then activate the TLD
    //sendMess("N,STATE1",motherAddr);////////////////////////////
    //delay(20);
    rv = readXBee();
    if(rv != NULL && newMes == 1){
      //message received
      //delay(50);
      //sendMess("N,MESSEGE?!",motherAddr);//////////////
      //delay(50);
      message = parseXBeeMess(rv,&sender);
      if(message.equals("CLOSEV") && sender == 'O'){
        //sendMess("N,CLOSERECEIVED!",motherAddr);//////////////
        //if rescue craft is close to the victim and message sent from rescue craft
        digitalWrite(TLD,HIGH); //turn on the TLD
        state = 2; // now close to the rescue craft
        resetAvg();
        delay(1000);
      }
    }
  }
}
while(state == 2){
  //Serial.println("State = 2");
  sendMess("N,STATE2",motherAddr);
  delay(50);
  sendMess("N,STATE2!",motherAddr);
  while(state == 2){
  //victim is close to the rescue craft
  //Send the mothership a message that the victim is close to the rescue craft
  //keep transmitting the TLD
  //check if rescue craft requires GPS coordinates again, if so, go to state 1
  rv = readXBee();
  if(rv != NULL && newMes == 1){
    //message received
    message = parseXBeeMess(rv,&sender);
    if(message.equals("VCOORD") && sender == 'O'){
      //if rescue craft sent message and requires gps coord again
      digitalWrite(GPSPOW,HIGH);
      state = 1;
      delay(1000);
    }
    //if the sender is the rescue craft and the home button has been activated, go to state 3
    if(message.equals("RETURN") && sender == 'O'){
      digitalWrite(TLD,LOW);
      state = 3;
      resetAvg();
      delay(1000);
    }
  }
}
}
while(state == 3){
  //Serial.println("state = 3");
  sendMess("N,STATE3",motherAddr);
  //send the mothership a message saying that victim is on rescue craft
  sendMess("N,ONRESCUE",motherAddr);
  while(1 && state == 3){
  //victim is on the rescue craft
  //turn off the GPS
  digitalWrite(GPSPOW,LOW);
  //keep transmitting the TLD, to make sure victim does not fall off again
}
}
while(state== 10){
  //Serial.println("State 10");
  //reset variable
```

```
        victimNew = 0;
        latitudeVictim =0;
        longitudeVictim=0;
        //Serial.println("Start Loop");
        getGPS(); //get GPS data
        //printGPS(); //print the received data
        //if(i > 3){ //5 = defaultonly send sometimes so as to not overflow the traffic (2 works)
          /*victimNew=1;
          latitudeVictim = 4251545;
          longitudeVictim = -71515454;*/
          sendGPS(); //send the GPS data via GPS
          //i = 0;

          //sendMess("TEST",rescueAddr);

        //}
        //i++;
        //readGPS(); //read GPS for incoming coordinates
      }
    }
}
```

Mothership Unit Code

```python
#!/usr/bin/python
#module for reading the data coming from the victim and rescue craft modules over XBEES
#Code written by Frederick Hunter for the Autonomous Man Overboard Rescue Equipment MQP

import serial

import time;

import math
import csv;
import XBeeMy #import module for XBee communication
import gps #import for gps manipulation
import gui #import for graphing and gui stuff
import calc #module for doing math
import threading #used for multithreading with the gps
import subprocess #module for starting another process in the background
import sys #import for starting a process in background


#degree_sign= u'\N{DEGREE SIGN}' #so a degree sign can be used

debug = 0;  #if debug statements should be printed, set to one.
HEADER_LINES = 11 #number of header lines (KEEP CONSISTANT WITH GUI TOO)
RADIUS = 6371000 #the radius of the earth in meters
"""
start_pos
Class for storing the start GPS positions of the different modules
"""
class start_pos:
  def __init__(self):
    self._lat = 0
    self._lon = 0


#class for storing needed GPS values
class gps_data:
  _XBee = '' #XBee identifier ex A, B, C.. etc A = Mothership, B = Victim, C = rescue
  _latitude = ''#latitude value ddmm.mmmm
  _longitude = '' #longitude value dddmm.mmmm
  _xpos = 0 #x-axis position in meters
  _ypos = 0 #y-axis position in meters
```

```
"""
isNum(num)
Checks if num is a number, if it isn't return false, if it is
return true

@param num: a string to check if it is a number
"""
def isNum(num):
  try:
    float(num)
    return True
  except ValueError:
    return False




"""
parse_coord()
returns the DMS coordinates in a graphable form
@param coord: in xxxdegyy'zz" form
"""
def parse_coord(coord):
  return coord.replace('deg','').replace("'",'').replace('"','')

'''
printMesToFile
prints the passed message to the messages file
these messages are received from the victim or rescue module
@param message: the message to be stored
'''
def printMesToFile(message):
  try:
    log = open('messages.log','a') #open the messages log file
  except Exception as e:
    print 'Could not open the messages.log file'
    print e
    exit() #kill the program
  log.write(str(message)); #write the message to file
  log.close #close the log file


"""
print_to_file
prints the given _gps coordinates to a file for later use
@param gpsXBee: the class with the data from the gps
"""
def print_to_file(gpsXBee):
  try:
    log = open('coordinates.log', 'a'); #open log file
  except Exception as e:
    print 'Could not open coordinates.log! Is the file in the current directory?'
    print e
    exit(); #kill the program
  #check if data is new before printing
  #print new data to log file
  if(int(float(gpsXBee._latitude)) != 0):
    #try:
    #check if the string for longitude and latitude is a number
    if(isNum(gpsXBee._latitude) and isNum(gpsXBee._longitude)):
      writeLog = gpsXBee._XBee + ',' + gpsXBee._latitude + ',' + gpsXBee._longitude + ',' + 'E'+
',' + str(gpsXBee._xpos) + ',' + str(gpsXBee._ypos) + '\n';
      log.write(writeLog);
    """except Exception as e:
      print 'Could not print data to file'
      print e"""
  log.close
```

143

```python
"""
print_gps(_gps)
prints the GPS data to screen
@param _gps the _gps data returned by read_serial()
"""
def print_gps(_gps):
  print 'XBEE = \t\t' + gpsXBee._XBee;
  print 'Latitude = \t' + gpsXBee._latitude;
  print 'Longitude = \t' + gpsXBee._longitude;
  print 'xpos = \t' + str(gpsXBee._xpos);
  print 'ypos = \t' + str(gpsXBee._ypos);
  print '' #print newline


"""
getStartPos()
returns 0 if none could be found, otherwise returns 1 and sets the startPosition in the start_pos
class.
"""
def getStartPos():
  try:
    log = open('coordinates.log', 'r');
  except Exception as e:
    print 'Could not open coordinates.log as read only. Is it in the current directory?'
    print e
    exit(); #kill the program
  #skip the header
  try:
    for i in range(0,HEADER_LINES):
      log.next();
  except Exception as e:
    print 'Error skipping the header file 0:' + HEADER_LINES + 'in getStartPos'
    print e
    exit(); #kill the program

  #try:
  reader = csv.reader(log) #open with csv reader
  for row in reader:#go through each row and look for the first module instance
    try:
      i = row[0] #if no module found at end, return 0
    except Exception as e:
      print e
      return 0
    #if the first input was found, save the lat & long values
    startPos._lat = (row[1])
    startPos._lon = (row[2])
    log.close()
    return 1
  #except:
  #  print 'Error in reading the log file in getStartPos'

"""
read_serial(_gps)
returns the data from the _gps needed for calculations in the parameter passed to it
returns two empty strings if no message has been received and if one has been received,
it returns the message, sender
"""
def read_serial(gpsXBee):
  #set gpsXBee._latitude to zero to notify the print_to_file function that no new data has been
received
  gpsXBee._latitude = 0;
  temp = XBeeMy.receive(xser);
  if(temp[0] == 'N'):
    #if message from victim received, print to screen
    print "Message from Victim: " + temp[2:]
    return temp[2:], 'N' #return the received message and the sender
  if(temp[0] == 'O'):
```

144

```
    #message from rescue received
    print "Message from Rescue: " + temp[2:]
    return temp[2:], 'O' #return the received message and the sender
  if debug:
    print 'Raw XBEE data received: ' + temp
  temp = temp.split(','); #store values in list
  #print temp
  if(len(temp) < 3): #make sure all the data has been received
    #print "data corrupted"
    return '', '' #return empty strings indicating that no message has been received
  while (temp[0].find('C') == -1) and (temp[0].find('B') == -1) and (temp[0].find('A') == -1):
#do nothing until desired data is found & not corrupt
    temp = XBeeMy.receive(xser);
    if(temp[0] == 'N'):
      #if message from victim received, print to screen
      print "Message from Victim: " + temp[2:]
      return temp[2:], 'N' #return the received message and the sender
    if(temp[0] == 'O'):
      #message from rescue received
      print "Message from Rescue: " + temp[2:]
      return temp[2:], 'O' #return the received message and the sender
    temp = temp.split(','); #store values in list
    if((len(temp) < 3) or (len(temp[2]) < 8) or (len(temp[1]) < 7)): #make sure all the data has
been received
      print "corrupt data"
      return '', '' #return empty strings indicating that no message has been received
    pass;
  if debug:
    print temp[0]
    print temp[1];
    print temp[2];

  #store values into _gpsXBee class
  if(temp[0].find('C') >= 0):
    if debug:
      print 'C RECEIVED'
    gpsXBee._XBee = 'C'; #if c found, then data is from module c
  if(temp[0].find('B') >= 0):
    if debug:
      print 'B RECEIVED'
    gpsXBee._XBee = 'B';
  if(temp[0].find('A') >= 0):
    if debug:
      print 'A RECEIVED'
    gpsXBee._XBee = 'A';
  #check if there is a start position for the Rescue craft yet
  if(startPos._lon == 0):
    #if it is 0, then check if there is already data logged,
    #otherwise, update it with the current data
    if(getStartPos() == 0): #if 0, then the start position should be updated with current
position
      startPos._lat = temp[1]#store lat position
      startPos._lon = temp[2].replace('\n', '').replace(' ', '') #store longitude position (get
rid of newline and any stray spaces)
  gpsXBee._latitude = temp[1]
  gpsXBee._longitude = temp[2].replace('\n', '').replace(' ','')
  #find the respective location in meters
  gpsXBee._xpos = calc.getxPos(gpsXBee._latitude, gpsXBee._longitude, startPos)
  gpsXBee._ypos = calc.getyPos(gpsXBee._latitude, gpsXBee._longitude, startPos)


  #plot_dms_gps(); #plot the datapoints to the _gps graph
  #gui.plotGps(False, True); #plot raw _gps values (1st boolean is for multiple plots, second is
for meters)
  if debug:
    print_gps(gpsXBee);
  return '','' #return empty strings indicating that no message has been received
```

```
"""
gpsThread(store, send, xser)
this function retrieves the gps coordinates in a background thread
it will store the values to the file if store == 1 it will send the
gps coordinates to the rescue module if send == 1
the thread exits if the globa variable done == 1
xser is the XBee serial object
"""
def gpsThread(store, send,xser):
  while done == 0:
    #get mothership coordinates a few times and grab the average position
    avgLat = 0
    avgLon = 0
    avgCount = 0
    for i in range(0,5):
      pos = gps.readGPS(gpsSerial)
      if(pos != -1):
        #split the return into lat and lon positions
        pos = pos.split(',')
        avgLat = avgLat + float(pos[0])
        avgLon = avgLon + float(pos[1])
        avgCount = avgCount + 1
    #get the positions
    latGps = str(avgLat/avgCount)
    lonGps = str(avgLon/avgCount)
    #if the start position is zero, update it with the current position
    if(getStartPos() == 0):
      startPos._lat = latGps;
      startPos._lon = lonGps;
    #calc the x and y pos in meters for storing
    latMeters = str(calc.getxPos(latGps,lonGps, startPos))
    lonMeters = str(calc.getyPos(latGps,lonGps, startPos))
    #if desired, store the the coordinates to file
    if(store):
      gps.saveToFile(latGps, lonGps, latMeters, lonMeters); #store the gps coordinates to file
      print 'Storing gps'
    #send the gps coordinates to the rescue unit if desired
    if(send):
      XBeeMy.sendToRescue(xser,"A,"+latGps+","+lonGps+",");
      print "Sending gps"
  print 'Terminating gps thread!'
  return #end the thread


############################GLOBAL VARIABLES#####################
lastTime = 0; #global variable for checking if time is same before printing to file to avaoid
repeats
done = 0; #variable to tell gps receive thread to stop


################################MAIN###########################

#have the user specify which serial port to use
#serialPort = raw_input('Enter the serial port the XBEE is connected to. Ex: ttyUSB0\n');
#serialPort = "/dev/" + serialPort;

try:
  ser, xser = XBeeMy.initXBee("ttyUSB0", 9600) #init the XBee


  print 'GPS is starting up, please wait...';
  time.sleep(1); #pause for 2 seconds
  print 'Systems online';
  gpsXBee = gps_data(); #instantiate the class for storing _gps data
  startPos = start_pos(); #instantiate the startpos class to store the start positions of the
modules

  gpsSerial = gps.initGPS("ttyAMA0",4800) #initiate the GPS class
```

146

```python
    #create a file for messages (if it exists already, it will clear the contents)
    l = open('messages.log','w+')
    l.close #close the file again




    '''
    Simulate the Victim Module here
    '''
    XBeeMy.sendToRescue(xser,"N,OVERBOARD");
    XBeeMy.sendToRescue(xser,"C,42.42504,-71.471025,") #simulated victim coordinates

    while 1:
      localtime = time.asctime( time.localtime(time.time())) #get current time
      printMesToFile("\n\nNew Test conducted on localtime!\n\n") #tell new test conducted
      state = 1
      print 'The mothership module is all set up and waiting for a distress signal!'
      while state == 1:

        XBeeMy.sendToRescue(xser,"C,42.42504,-71.471025,") #simulated victim coordinates


        #in this state, no one has fallen overboard, keep checking for an overboard condition
        message , unit = read_serial(gpsXBee); #read the XBee for gps coordinates or messages
        if(unit == ''): #determine if gps data received
          print_to_file(gpsXBee); #store the gps location to file
        if(unit == 'N' and message == 'OVERBOARD'):
          #if overboard condition
          printMesToFile('VICTIM OVERBOARD\n'); #store to file that the victim is overboard
          print "VICTIM OVERBOARD!"
          time.sleep(0.2) #allow packet to get sent
          #start the rescue boat unit
          XBeeMy.sendToRescue(xser,"M,BOOT");
          #start the plotting GUI
          process = subprocess.Popen([sys.executable, 'plotBG.py'], stdout=subprocess.PIPE,
    stderr=subprocess.STDOUT)
          message , unit = read_serial(gpsXBee); #read the XBee for gps coordinates or messages
          if(unit == 'N' and message == 'INITDN'):
            #victim unit initialized
            printMesToFile("Victim unit is online and Transmitting\n") #store to file
            print "Victim unit GPS is online and Transmitting!"
          #wait for the rescue unit to init
          while(unit != 'O' and message != 'INIT'):
            #waiting for the rescue craft to init
            message , unit = read_serial(gpsXBee); #read the XBee for gps coordinates or messages
            if(unit == 'N' and message == 'INITDN'):
              #victim unit initialized
              printMesToFile("Victim unit is online and Transmitting\n") #store to file
              print "Victim unit GPS is online and Transmitting!"
          #set state to 2
          printMesToFile("Rescue Craft initialized and heading off\n")
          print "Rescue Craft initialized and heading off"
          state = 2;
      while state == 2:
        #initialize things for state 2 here
        #start thread to store the mothership gps data
        done = 0; #make sure thread is continuing to run
        t1=threading.Thread(group=None,target=gpsThread,name="Thread-1",args=(1,0,xser))
        t1.daemon = True #allow program to terminate with ctrl^c
        t1.start()
        print "Started thread in state 2"
        while state == 2:

          XBeeMy.sendToRescue(xser,"C,42.42504,-71.471025,") #simulated victim coordinates


          #Here we are just collecting the gps coordinates and storing them to file for plotting
          message , unit = read_serial(gpsXBee); #read the XBee for gps coordinates or messages
```

147

```
      if(unit == ''): #check if gps data received, or message
        print_to_file(gpsXBee); #store the gps location to file
      #check if the rescue craft is returning home or close to victim
      if(unit == 'O' and message == 'CLOSEV'):
        #if rescue craft is close to victim (within 50m), notify the crew
        printMesToFile("The rescue craft is close to the victim (within 50m)\n")
        print 'The rescue craft is close to the victim (within 50m)'
      if(unit == 'N' and message == 'INITDN'):
        #victim unit initialized
        printMesToFile('Victim unit is online and Transmitting\n')
        print "Victim unit GPS is online and Transmitting!"
      if(unit == 'N' and message == 'STATE2'):
        #victim turned on TLD
        printMesToFile("Victim activated Terminal Locator Device\n")
        print "Victim activated Terminal Locator Device"
      if(unit == 'O' and message == 'RETURN'):
        printMesToFile("The rescue craft is returning to the mothership\n")
        print "The rescue craft is returning to the mothership"
        #if rescue craft is returning home, send the GPS coordinates to the rescue craft
        #stop the thread for storing mothership GPS
        done = 1;
        #wait for thread to exit
        print "Waiting for thread to exit"
        t1.join()
        #set to state 3
        state = 3
  while state == 3:
    #initialize stuff here for state 3
    #start a thread to retrieve GPS coorinates of mothership and send them to the rescue craft
    done =0; #set done to 0 so thread continues
    t2=threading.Thread(group=None,target=gpsThread,name="Thread-2",args=(1,1,xser))
    t2.daemon = True #allow program to terminate with ctrl^c
    t2.start()
    print "Started thread 2 in state 3"
    while state == 3:
      #Here the rescue craft is returning home, so send it the mothership GPS coordinates
      #read the XBee for incoming data
      message, unit = read_serial(gpsXBee);
      if(unit == ''): #check if gps data received, or message
        print_to_file(gpsXBee); #store the gps location to file
      if(unit == 'N' and message == 'ONRESCUE'):
        #message from victim
        printMesToFile("The Victim is on the rescue craft!\n")
        print "The Victim is on the rescue craft!"
      if(unit == 'O' and message == 'HOME'):
        printMesToFile("The rescue unit is back home!\n")
        print "The rescue unit is back home!"
        #Check if the rescue module is back home and notify the crew
        #set to state 1
        #stop the gps thread
        done = 1;
        t2.join() #wait for the thread to exit
        state = 1
        #stop the plot gui
        try:
          process.kill()
        except:
          pass
except KeyboardInterrupt: #kill the process in the background if it is running when ctl^c called
  try:
    process.kill()
  except:
    pass
```

Calc.py Module Code

```
#! /bin/python
#module calculations Filename: calculations.py
#Code written by Frederick Hunter for the Autonomous Man Overboard Rescue Equipment MQP
```

148

```python
"""
This module contains all the major methods for
calculating distances, angles, averages, etc..
"""


import math


RADIUS = 6372795
debug = 0

"""
calc_dist(module_a, module_b)
calculates the straight line distance to the coordinate
@param module_a: the current module where the distance is calculated from
@param module_b: the module the distance is supposed to be calculated to
Found the calculation on http://www.movable-type.co.uk/scripts/latlong.html
under Equirectangular approximation
"""
def calc_dist(module_a, module_b):
  #check if there is even an up to date current position for both modules
  if(module_a._current_lat_pos == '' or module_b._current_lat_pos == ''):
    #print 'No up to date position for module'
    return 0;
  '''
  a = math.radians(module_a._current_lon_pos - module_b._current_lon_pos) *
math.cos((math.radians(module_a._current_lat_pos + module_b._current_lat_pos))/2);
  b = math.radians(module_a._current_lat_pos - module_b._current_lat_pos);
  #print math.sqrt(math.pow(a,2) + math.pow(b,2)) * 1369.42
  return math.sqrt(math.pow(a,2) + math.pow(b,2)) * RADIUS
  '''



"""
calc_angle_to_target(module_a, module_b)
claculates the angle from the current module to the other module where 0deg is
@param module_a: the current module where the distance is calculated from
@param module_b: the module the distance is supposed to be calculated to
"""
def calc_angle_to_target(module_a, module_b):
  if(module_a._current_lat_pos == '' or module_b._current_lat_pos == ''):
    #print 'No up to date position for module'
    return 0;
  a = module_a._current_lon_pos - module_b._current_lon_pos;
  b = module_a._current_lat_pos - module_b._current_lat_pos;
  #print math.atan2(b,a)
  return math.atan2(b,a);



"""
get_average(num, lst)
calculates the average of the last num values in the list
@param num: the number of values to in list to go back and calculate with
@param lst: a list of numbers to calculate average from
"""
def get_average(num, lst):
  """Determine if the list even has the amount of numbers to get average of"""
  if(len(lst) > num + 1):
    #take average of the last num values
    avg = 0
    for i in range(len(lst)-(num+1),len(lst)-1):
      #print 'lst[i]: ' + str(lst[i])
      avg = (lst[i]-lst[i-1]) + avg
    avg = avg / num
    #print 'avg: ' +str(avg)
    return avg;
  #if list is basically empty, use the last value
```

```
      elif(len(lst) < 3):
        #check if list is empty
        if(len(lst) == 0):
          return 0;
        return lst[len(lst)-1];
      #otherwise if lst is not populated with enough numbers, use the max possible
      else:
        avg = 0
        for i in range(0,len(lst)-1):
          if(i == 0):
            pass #skip the first value
          else:
            #print 'lst[i]: ' + str(lst[i]) + ' ' + str(lst[i-1]) + 'Sub: ' + str(lst[i] -lst[i-1])
            avg = (lst[i]-lst[i-1]) + avg
        avg = avg / i
        #print 'avg: ' +str(avg)
        return avg


"""
getxPos(latitude, longitude, startPos)
@param latitude: The current gps latitude
@param longitude: the current gps longitude position
@param startPos: The class storing the start position of the modules
calculates the x position of the gpsData, if no new data passed, return 0
"""
def getxPos(latitude, longitude, startPos):

  #determine if the lat & lon is not zero
  if debug:
    print 'Acquiring the x pos in meters'
    print 'lat:'
    print latitude
    print 'lon:'
    print longitude
  if(latitude == '0'):
    return 0
  else:
    try:

      x =  (math.radians(float(longitude) - float(startPos._lon)) *
math.cos(((math.radians(float(startPos._lat) + float(latitude)))/2))) * RADIUS;
      if debug:
        print 'x = ' + str(x)
      return x
    except Exception as e:
      print 'ERROR calculating the x pos in getxPos'
      print e
  return 0

"""
getyPos(latitude, longitude, startPos)
@param latitude: The current gps latitude
@param longitude: the current gps longitude positions
@param startPos: The class containing the start long and lat positions
calculates the y position in meters using the passed data
"""
def getyPos(latitude, longitude, startPos):
  #determine if the lat & lon is not zero

  if debug:
    print 'Acquiring the y pos in meters'
    print 'lat:'+ latitude
    print 'lon:' + longitude
    print 'victim start pos: '+ str(startPos._lat)
  try:
    if(latitude == '0'):
      return 0
    else:
```

```
      y =  math.radians(float(latitude) - float(startPos._lat)) * RADIUS
      if debug:
        print 'y = ' + str(y)
      return y
  except Exception as e:
    print 'ERROR calculating the y pos in meters in getyPos'
    print e

'''
function to calculate the angle (in degrees) the arrow should be pointing in
@param xA: The x position of the last datapoint in the set
@param xB: The x position of the previous datapoint in the set
@param yA: The y position of the last datapoint in the set
@param yB: The y position of the previous datapoint in the set
'''
def calcArrowAngle(xA, yA, xB, yB):
  #determine the distances from the two points
  #where xA is the most recent point and xB is the last point
  x = xA - xB;
  y = yA - yB;
  #determine what quandrant we are in
  if((x > 0 and y > 0) or (x < 0 and y > 0)):
    #if in quadrants one or two, the same calculation can be used
    angleDeg = 180 - math.degrees(math.atan2(y,x))
    if debug:
      print "Q1 or Q2: " + str(angleDeg)
    return angleDeg
  else:
    #otherwise if in quadrant 3 or 4
    angleDeg = math.degrees(math.fabs(math.atan2(y,x)))+180
    if debug:
      print "Q3 or Q4: " + str(angleDeg)
    return angleDeg
```

GPS.py Module Code

```
#! /bin/python
#module gps Filename: gps.py
#Code written by Frederick Hunter for the Autonomous Man Overboard Rescue Equipment MQP


import serial
debug = 1

"""
Attempt to initialize the serial port for the GPS on the RPI
@param port: the port where the GPS is connected to on the RPI (usually ttyAMA0)
@param baud: the baud rate
"""
def initGPS(port, baud):
  try:
    gSer = serial.Serial("/dev/"+port, baud,bytesize=8,stopbits=1)
    return gSer
  except Exception as e:
    print "ERROR: in initGPS (Is the GPS connected to the RPI?)"
    print e
    exit()


"""
attempts to read the GPS data from the GPS and returns the lat & lon pos ("42.254455,-71.278220")
@param gSer: the serial port object initialized by the initGPS function
"""
def readGPS(gSer):
  try:
    #read a line from the GPS
    inLine = gSer.readline();
    #while the desired data is not found, continue reading lines
```

```python
    while(inLine.find("GPRMC")==-1):
      inLine = gSer.readline();
    #extract the data
    pos = extractPos(inLine);
    if(pos == -1):
      if(debug):
        print 'No GPS Lock!'
      return -1
    print 'Received GPS' + pos
    return pos; #return the pos if it was recived
  except Exception as e:
    print "ERROR reading the gps"
    print e

"""
Extracts the gps position from the line read from the GPS
@param line: the line read in from the gps
ex (with a lock): line = $GPRMC,170006.000,A,4225.4455,N,07127.8220,W,1.61,153.14,170213,,*19
"""
def extractPos(line):
  #Check if there is even a lock and data
  line = line.split(',')
  #determine if the extracted line is the the correct line
  if(line[2] == 'V' or line[3] == ''):
    #if there is no lock, return -1
    return -1
  else:
    #if there is a lock, extract the data and send it for storage
    lat = line[3]
    lon = line[5]
    #convert to decimal degrees
    deglat = int(lat[0:2]); #get the degree
    lat = lat[2:] #get the rest of the lat value
    latf = float(lat) #convert to float for math
    lat = deglat+(latf/60) #convert to decimal degrees
    lat = str(lat) #convert to string for return and trim the answer
    lat = lat[0:8]
    #convert lon to decimal degrees
    deglon = int(lon[0:3])
    lon = lon[3:] #get the rest of the longitude
    lonf = float(lon) #convert to float for calculation
    lon = deglon+(lonf/60) #convert to decimal degrees
    lon = str(lon) #convert to string for returning
    lon = '-'+lon[0:8] #trim the value and make negative
    return lat+','+lon;
    '''
    #move the decimal point to the correct location
    lat = lat.replace('.','')
    lat = lat[0:2]+'.'+lat[2:]
    lon = lon.replace('.','')
    lon = '-'+lon[1:3]+'.'+lon[3:]
    #return the lat and lon positions
    return lat+','+lon;
    '''


"""
Stores the longitude and latitude position into the log file
@param latitude: the string of the lat position from the GPS
@param longitude: the string of the lon position from the GPS
@param xMeters: the x position in meters calculated by getxPos in the calc module
@param yMeters: the y position in meters calculated by getYPos in the calc module
"""
def saveToFile(latitude, longitude, xMeters, yMeters):
  #store the longitude and lat position into the log file
  #try to open the log file for storage
  try:
    log = open('coordinates.log', 'a')
    #print to file
```

```
      log.write('A,'+str(latitude)+','+str(longitude)+',E,'+str(xMeters)+','+str(yMeters)+'\n');
      print "Stored gps pos:"
      log.close() #close the log file
    except Exception as e:
      print "Could not open the coordinates.log file in gps.py. Is it in the current directory?"
      print e
      exit() #kill the program due to a heavy error
```

XBeeMy.py Module Code

```python
#! /bin/python
#module XBee Filename: XBee.py
#Code written by Frederick Hunter for the Autonomous Man Overboard Rescue Equipment MQP

import time
import serial;
from XBee import ZigBee

"""
initXBee(port)
This function initializes the XBEE connected
to the passed serial port
It returns the serial object used for printing and
reading with the XBee
"""
def initXBee(port, baud):
  serialPort = '/dev/' + port
  try:
    ser = serial.Serial(serialPort,baud,bytesize=8,stopbits=1)
    XBee = ZigBee(ser, escaped = True) #escped = AP mode = 2 on XBee
    return ser, XBee
  except Exception as e:
    print e
    exit()

"""
sendToRescue(port, sendString)
this function sends the command passed to this function
to the RESCUE XBEE module
"""
def sendToRescue(xser, sendString):

  #send the message
  #TODO: the address here is for the mothership, need to change to
  #rescue address : 405c2ca0
  #dest = "\x00\x13\xa2\x00\x40\x6c\xb5\x4d"; #mothership address
  #dest = "\x00\x13\xa2\x00\x40\x5c\x2c\xa0" #rescue address
  dest = "\x00\x13\xa2\x00\x40\x99\x23\x3c"#victim address
  xser.send("tx",dest_addr="\xff\xff",dest_addr_long=dest,
data=sendString)


"""
sendToVictim(xser, sendString)
This function sends the string passed to this function
to the XBEE module called VICTIM
"""
def sendToVictim(xser, sendString):
  '''
  ser.write('+++'); #enter the command mode
  time.sleep(1); #wait a sec for the cbee to register
  ser.write('ATDN VICTIM\r'); #set XBee to send to the VICTIM
  time.sleep(0.5); #wait for XBee to set
  ser.write(str(sendString)) #send the message
  time.sleep(1) #wait a sec for XBee to exit the command mode
```

153

```
    '''
    dest = "\x00\x13\xa2\x00\x40\x9f\x3a\x9d"
    xser.send("tx",dest_addr="\xff\xff",dest_addr_long=dest,data=sendString)

"""
receive(ser)
this function returns the line on the serial buffer associated
with the ser.
ser is the return from the initXBee function
"""
def receive(xser):
  try:
    response = xser.wait_read_frame()
    return response['rf_data'] #return the XBee response
  except Exception as e:
    print e
    return '-1'
```

PlotBG.py Script Code

```python
#! /bin/python
"""
module to remotely control the boat written by frederick Hunter,
uses the event handler
NOTE: This must be run as root!
"""

from evdev import InputDevice
from select import select
import serial
from XBee import ZigBee
import time

#module we are using, can be either laptop or rpi
module = 'laptop'

#keyboard values from the laptop
LapKey = {'Q': 16, 'RIGHT': 205, 'SPACE': 57, 'UP': 200, 'DOWN': 208, 'R': 19, 'LEFT':
203,'RSHIFT':54}
#keyboard values from the USB Keyboard on RPI
rpiKey = {'Q': 458772, 'RIGHT': 458831, 'SPACE': 458796, 'UP': 458834, 'DOWN': 458833, 'R':
458773, 'LEFT': 458832,'RSHIFT':458981}

#set the dictionary to use
if(module == 'laptop'):
  dic = LapKey;
  dev = InputDevice('/dev/input/event3') #device registered as keyboard
else:
  dic = rpiKey;
  dev = InputDevice('/dev/input/event0') #device registered as keyboard


serialPort = "/dev/ttyUSB0" #serial port for XBee
baud = 9600

ser = serial.Serial(serialPort,baud,bytesize=8,stopbits=1)
XBee = ZigBee(ser, escaped = True) #escped = AP mode = 2 on XBee




def sendToRescue(xser, sendString):

  #send the message
  #TODO: the address here is for the mothership, need to change to
  #rescue address : 405c2ca0
  #dest = "\x00\x13\xa2\x00\x40\x6c\xb5\x4d"; #pro2sB revB
  #dest = "\x00\x13\xa2\x00\x40\x99\x23\x3c" #pro2sB rev A
  dest = "\x00\x13\xa2\x00\x40\x5c\x2c\xa0" #pro2sB
```

```
    xser.send("tx",dest_addr="\xff\xff",dest_addr_long=dest, data=sendString)

#send to rescue to boot it
sendToRescue(XBee, 'M,BOOT')
key = 0
while key != dic['Q']:
  r,y,x=select([dev],[],[])
  for event in dev.read():
    if(event.type == 4): #make sure it is a keyboard input
      key = event.value
      #print(event.value)
      if key == dic['UP']:
        #caught the up key
        print "GO" #go forwards
        sendToRescue(XBee, 'M,FF')
      elif key == dic['DOWN']:
        #caught the down key
        print "REVERSE" #Stop the boat
        sendToRescue(XBee, 'M,BB')
      elif key == dic['RIGHT']:
        #caught the right key
        print "RIGHT" #turn right
        sendToRescue(XBee, 'M,LO')
      elif key == dic['LEFT']:
        #caught the left key
        print "LEFT" #turn left
        sendToRescue(XBee, 'M,RO')
      elif key == dic['RSHIFT']:
        #caught the right shift key
        print "STOP"
        sendToRescue(XBee, 'M,SS')
      elif key == dic['SPACE']: #space is used to kill the boat
        print "Killing the motors and enetering remote control mode" #message to user
        sendToRescue(XBee, 'M,STOP') #stop the motors and enter control mode on the rescue craft
      elif key == dic['R']:
        #caught the r key
        print 'STOP' #stop the motors and enter control mode on the rescue craft
        sendToRescue(XBee, 'M,STOP')
        time.sleep(0.5)
        print "Restarting in State 0" #message to user
        #restart the rescue boat in state 0
        sendToRescue(XBee, 'M,RESTART')
exit() #end the program on 'q' press
```

PlotBG.py

```
#!/usr/bin/python
#module for handling the plotting in the background process, written by Frederick Hunter for
AMORE MQP

import gui
import gps
import pyqtgraph as pg
import time
from pyqtgraph.Qt import QtGui, QtCore
import csv
import calc #import for my calc module
import Image
import numpy
import calc #import for getx and gety
import serial

gpsSerial = gps.initGPS("ttyAMA0",4800) #initiate the GPS class

HEADER_LINES = 11

"""
start_pos
```

```
Class for storing the start GPS positions of the different modules
"""
class start_pos:
  def __init__(self):
    self._lat = 0
    self._lon = 0

#create a qt application
app = QtGui.QApplication([])
#create the main window
mw = QtGui.QMainWindow()
#resize if needed
mw.resize(400,400)
#create central widget
cw = QtGui.QWidget()
#set the central widget onthe main window
mw.setCentralWidget(cw)
#create andset the widget layout
l = QtGui.QVBoxLayout()
cw.setLayout(l)

#create a plot widget window
pWidget = pg.PlotWidget(name="GPS Plot V:0.5")
#add a legend
pWidget.addLegend(size=None,offset=(0,0))
#add the plot widget to the layout
l.addWidget(pWidget)
#show the main window
mw.show()

#create the arrows for later

#create variable to store info if the messages have been printed
global mState
mState = [0,0,0,0,0,0,0,0];
#the text object to write the messages to the plot
global text
text = pg.TextItem(text="Plot V:0.5!",color='w',html=None)
text.setPos(0.5,0.5) #set the position
pWidget.addItem(text) #add the text to the plot

#axis labels
pWidget.setLabel('left',"Y Axis",units='m')
pWidget.setLabel('bottom',"X Axis",units='m')

#image plotting
#store the lat and lon location of the top right and bottom left of the image
#also store the center
'''
latTopRight = 42.437779
lonTopRight = -71.428312
latCenter = 42.355869
lonCenter = -71.632958
latBottomLeft = 42.277087
lonBottomLeft = -71.856934
'''
latTopRight = 42.427485
lonTopRight = -71.466914
latCenter = 42.426418
lonCenter = -71.469027
latBottomLeft = 42.425686
lonBottomLeft = -71.470748

#instantiate the start pos class
startPos = start_pos()
startPos._lat = 42.42462
startPos._lon = -71.463632

#grab the image file
```

```python
imageFile = "42.426418,-71.469027.png"
src = Image.open(imageFile)
#convert to color
src = src.convert(mode="RGB")
#convert image to array
srcArray=numpy.asarray(src)
#rotate the array to be oriented correctly
rotArray=numpy.rot90(srcArray,3)

#set the image object
IIR = pg.ImageItem(image=rotArray,autoLevels=False)

#determine the location in meters (Bottom left and top right) from center
metersBLx = calc.getxPos(latBottomLeft,lonBottomLeft,startPos)
metersBLy = calc.getyPos(latBottomLeft,lonBottomLeft,startPos)
metersTRx = calc.getxPos(latTopRight,lonTopRight,startPos)
metersTRy = calc.getyPos(latTopRight,lonTopRight,startPos)



#set the size of the image (scale it)
IIR.setRect(QtCore.QRectF(metersBLx,metersBLy,metersTRx+abs(metersBLx),metersTRy+abs(metersBLy)))
#scale the image
pWidget.addItem(IIR) #add the image to the plot

#create an empty plot object for now
p1=pWidget.plot(name="Rescue Unit") #rescue plot
p2=pWidget.plot(name="Victim Unit") #victim plot
p3=pWidget.plot(name="Mothership Unit") #mothership plot
'''
getMesFromFile(messageState)
returns the new message on the logfile and the updated message state list
messageState = a list of states for the messages
ex-> [0(victim overboard?),
0(victim online and transmitting)),
0(Rescue craft intitalized and heading off?),
0(Rescue craft close to victim?),
0(Victim activated terminal locator device?),
0(Rescue Craft returning to mothership?),
0(Victim on the rescuecraft?),
0(Rescue unit home?)]
'''
def getMesFromFile(messageState):
  newMessageState = messageState; #store for returning
  #open the log file
  try:
    mlog =open('messages.log','r').read()
    if(messageState[0] == 0):
      #if the victim overboard condition was not yet read
      #check if it has been received
      if('VICTIM OVERBOARD' in mlog):
        newMessageState[0] = 1 #set to read
        return "Victim Overboard", newMessageState
    if(messageState[1] == 0):
      #if the victim online and transmitting condition not met
      #check if it has now been met
      if('Victim unit is online and Transmitting' in mlog):
        newMessageState[1] = 1
        return 'Victim Online',newMessageState
    if(messageState[2] == 0):
      #if the rescue craft has been init and heading off has not been met
      #check if it has now been met
      if('Rescue Craft initialized and heading off' in mlog):
        newMessageState[2] = 1
        return 'Rescue Searching', newMessageState
    if(messageState[3] == 0):
      #if the rescue craft is close to the victim has been met
      #check if it is now close
      if('The rescue craft is close to the victim (within 50m)' in mlog):
```

157

```
        newMessageState[3] = 1
        return "Rescue Close", newMessageState
    if(messageState[4] == 0):
      #if the victim activated the TLD yet
      #check if it is now activated
      if('Victim activated Terminal Locator Device' in mlog):
        newMessageState[4] = 1
        return "TLD Activated", newMessageState
    if(messageState[5] == 0):
      #check if the rescue craft has been returning to the mothership
      #check if needs to be updated
      if('The rescue craft is returning to the mothership' in mlog):
        newMessageState[5] = 1
        return 'The rescue craft is returning to the mothership',newMessageState
    if(messageState[6] == 0):
      #if the victim has not been on the rescue craft
      #check if it now is
      if('The Victim is on the rescue craft!' in mlog):
        newMessageState[6] = 1
        return 'Victim on Craft',newMessageState
    if(messageState[7] == 0):
      #check if the rescue unit has been home yet
      #if not, check again
      if('The rescue unit is back home!' in mlog):
        newMessageState[7] = 1
        return 'Home', newMessageState
  except Exception as e:
    print 'Error opening the messages +.log file in gui.py!'
    print e
  return None,messageState #return nothing


'''
updates the plot data
'''
def updateData():
  #instantiate variables for the modules to be graphed
  mothershipX = []
  mothershipY = []
  rescueX = []
  rescueY = []
  victimX = []
  victimY = []
  try:
    log = open('coordinates.log', 'rt');
  except Exception as e:
    print 'Could not open coordinates.log as read only. Is it in the current directory?'
    print e
    exit(); #kill the program
  #skip the header
  try:
    for i in range(0,HEADER_LINES):
      log.next();
  except Exception as e:
    print 'Error skipping the header file 0:' + HEADER_LINES + 'in plot_gps'
    print e
    exit(); #kill the program

  #try:
  reader = csv.reader(log) #open with csv reader

  for row in reader:#go through each row and add to array
    #check which XBee the data is from & add to corresponding graph
    if(row[0].find('A') >= 0):
      mothershipX.append(float(row[4]))
      mothershipY.append(float(row[5]))
    elif(row[0].find('C') >= 0):
      victimX.append(float(row[4]))
      victimY.append(float(row[5]))
```

158

```
      else:
         rescueX.append(float(row[4]))
         rescueY.append(float(row[5]))
   log.close(); #close the file
   #clear the plot
   #pWidget.clear()
   #update the plot datapoints
   p1.setData(x=rescueX ,y=rescueY, pen='r');
   p2.setData(x=victimX ,y=victimY, pen='b');
   p3.setData(x=mothershipX ,y=mothershipY, pen='g');
   #clear the legend to prevent duplicates from showing up
   pWidget.plotItem.legend.items=[]
   #add the plots to the widget
   pWidget.addItem(p1)
   pWidget.addItem(p2)
   pWidget.addItem(p3)
   #update the arrow positions and check if there is even data in the lists
   if(len(rescueX)>0):
      global a1
      #first we try to see if the variable has even been set
      try:
         pWidget.removeItem(a1) #get rid of old arrow
      except:
         pass #do nothing if this arrow has not yet been created to be removed
      a1=pg.ArrowItem(angle=calc.calcArrowAngle(rescueX[len(rescueX)-1],rescueY[len(rescueY)-1],
rescueX[len(rescueX)-2], rescueY[len(rescueY)-2]),brush='r', pen='r') #rescue arrow
      #update the arrow location
      a1.setPos(rescueX[len(rescueX)-1],rescueY[len(rescueY)-1])
      #add the arrow
      pWidget.addItem(a1)
      #add some padding to show all the drawn objects
      #pWidget.autoRange(padding=0.1, item=p1)
   if(len(victimX)>0):
      global a2
      #first we try to see if the variable has even been set
      try:
         pWidget.removeItem(a2) #get rid of old arrow
      except:
         pass #do nothing if this arrow has not yet been created to be removed
      a2=pg.ArrowItem(angle=calc.calcArrowAngle(victimX[len(victimX)-1], victimY[len(victimY)-1],
victimX[len(victimX)-2], victimY[len(victimY)-2]),brush='b', pen='b') #victim arrow
      #update the arrow position
      a2.setPos(victimX[len(victimX)-1],victimY[len(victimY)-1])
      #add the arrow
      pWidget.addItem(a2)
      #add some padding to show all the drawn objects
      #pWidget.autoRange(padding=0.1, item=p2)
   if(len(mothershipX)>0):
      global a3
      #first we try to see if the variable has even been set
      try:
         pWidget.removeItem(a3) #get rid of old arrow
      except:
         pass #do nothing if this arrow has not yet been created to be removed
      a3=pg.ArrowItem(angle=calc.calcArrowAngle(mothershipX[len(mothershipX)-1],
mothershipY[len(mothershipY)-1], mothershipX[len(mothershipX)-2], mothershipY[len(mothershipY)-
2]),brush='g', pen='g') #mothership arrow
      #update the arrow position
      a3.setPos(mothershipX[len(mothershipX)-1],mothershipY[len(mothershipY)-1])
      #add the arrow
      pWidget.addItem(a3)
   global text
   pWidget.addItem(text) #add the text again because it was cleared
   #determine if there is a new message received and if so, delete the last text and add the new
one
   #only check if we have coordinates for the rescue craft
   if(len(mothershipX) > 0 or len(victimX) > 0 or len(rescueX) > 0):
      global mState
      #print mState
```

```
    newMes, mState = getMesFromFile(mState)
    #print newMes
    if(newMes != None):
      #if there is a new message
      #remove the old one
      pWidget.removeItem(text)
      #create the new one
      text = pg.TextItem(text=newMes,color='w',html=None)
      #try to place the text in the last rescue coordinate
      if(len(rescueX) > 0):
        text.setPos(rescueX[len(rescueX)-1],rescueY[len(rescueY)-1]) #set the position of it
      #otherwise try to write to the last position of the mothership
      elif(len(mothershipX) > 0):
        text.setPos(mothershipX[len(mothershipX)-1],mothershipY[len(mothershipY)-1]) #set the
position of it
      elif(len(victimX)>0):
        #otherwise try the victim
        text.setPos(victimX[len(victimX)-1],victimY[len(victimY)-1]) #set the position of it
      else:
        text.setPos(0.5,0.5)#if none there, place in 0.5,0.5
      pWidget.addItem(text) #add the item

    #pWidget.autoRange(padding=0.5, item=a3)
  #add some padding to show all the drawn objects
  pWidget.setAspectLocked(lock=True,ratio=1)
  #pWidget.autoRange(padding=0.05)
  #set the view range (Zoomed in on map)
  xVuRng=  mothershipX[len(mothershipX)-1]
  yVuRng =  mothershipY[len(mothershipY)-1]
  #zoomed in to 100x100 meters
  pWidget.setRange(xRange=[xVuRng-50,xVuRng+50],yRange=[yVuRng-50,yVuRng+50])


## Start a timer to rapidly update the plot in pw
#while 1: #NOTE:Not sure if this will mess up the file
t = QtCore.QTimer()
t.timeout.connect(updateData)
t.start(500)
#updateData()



## Start Qt event loop unless running in interactive mode or using pyside.
if __name__ == '__main__':
    import sys
    if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
        QtGui.QApplication.instance().exec_()
```

Appendix N - Coordinates.log

```
#log file to store GPS data for reading
#data stored as: (Separated by comma)
#XBee.........#XBee identifier ex A=Mother, B=Rescue, C=victim.. etc
#latitude.....#latitude value (degrees)
#longitude....#longitude value
#Validity bit (E is valid)
#xPos.........#x-axis position in meters from the start position
#yPos.........#y-axis position in meters from the start pos


##############################################################
A,42.4250335,-71.4719553,E,0.0,0.0
A,42.4250291,-71.4719626,E,-0.599350793168,-0.489395522737
A,42.4250221,-71.4719593,E,-0.32841141134,-1.26797931027
A,42.4250172,-71.4719485,E,0.558299423196,-1.81298796131
A,42.4250033,-71.4719425,E,1.05091667713,-3.35903290994
```

```
A,42.4249880,-71.4719418,E,1.10838881822,-5.06079461623
C,42.4250306,-71.4719499,E,0.443355376886,-0.322556140201
C,42.4250370,-71.4719479,E,0.607561040413,0.389291894164
A,42.4249915,-71.4719298,E,2.09362326408,-4.67150272285
C,42.4250375,-71.4719378,E,1.43679975164,0.444005021676
B,42.4259339,-71.4694749,E,203.646435737,100.148120276
B,42.4259349,-71.4694843,E,202.874672912,100.259346532
B,42.4259319,-71.4694930,E,202.160388128,99.9256677661
C,42.4250454,-71.4719351,E,1.65847732397,1.32359243858
A,42.4249924,-71.4719177,E,3.08706800554,-4.5713990927
B,42.4259289,-71.4695010,E,201.503574892,99.5919890003
B,42.4259265,-71.4695111,E,200.674345914,99.3250459876
A,42.4249905,-71.4719097,E,3.74389104287,-4.78272897788
C,42.4250464,-71.4719439,E,0.935972343723,1.4348186936
B,42.4259156,-71.4695265,E,199.409988433,98.1126798048
B,42.4259106,-71.4695366,E,198.580763446,97.5565485281
A,42.4249865,-71.4719137,E,3.41547965655,-5.22763399876
C,42.4250518,-71.4719439,E,0.935972303413,2.03544047215
B,42.4259086,-71.4695426,E,198.088152978,97.3340960173
A,42.4249821,-71.4719204,E,2.86539048553,-5.71702952229
A,42.4249781,-71.4719150,E,3.30874613946,-6.16193454397
C,42.4250528,-71.4719358,E,1.60100524295,2.14666672797
B,42.4259042,-71.4695500,E,197.480603091,96.8447004938
B,42.4259007,-71.4695607,E,196.602114232,96.4554086004
A,42.4249746,-71.4719016,E,4.4089248794,-6.55122643734
B,42.4258968,-71.4695688,E,195.937091877,96.0216262044
A,42.4249672,-71.4719003,E,4.5156588948,-7.37430072673
B,42.4258888,-71.4695983,E,193.515086956,95.1318161626
C,42.4250518,-71.4719251,E,2.47950557436,2.03544047215
B,42.4258874,-71.4696237,E,191.429691042,94.9760994049
A,42.4249608,-71.4719003,E,4.51565912529,-8.08614876031
B,42.4258725,-71.4696452,E,189.664514423,93.3188282005
A,42.4249578,-71.4718942,E,5.01648689433,-8.41982752617
C,42.4250612,-71.4719271,E,2.315299734,3.08096727238
B,42.4258601,-71.4696673,E,187.850072255,91.9396226352
A,42.4249642,-71.4718895,E,5.40237022503,-7.70797949259
B,42.4258537,-71.4696888,E,186.084882211,91.2277746008
B,42.4258468,-71.4697089,E,184.43463596,90.4603134397
A,42.4249687,-71.4718909,E,5.28742598775,-7.2074613434
B,42.4258393,-71.4697330,E,182.45598113,89.6261165247
C,42.4250687,-71.4719284,E,2.20856592673,3.91516418664
A,42.4249746,-71.4718909,E,5.28742573895,-6.55122643734
B,42.4258379,-71.4697585,E,180.362374037,89.470399767
A,42.4249761,-71.4718848,E,5.78825326359,-6.38438705401
C,42.4250637,-71.4719177,E,3.08706625006,3.35903291073
B,42.4258294,-71.4697820,E,178.432981641,88.524976597
A,42.4249672,-71.4718835,E,5.89498742964,-7.37430072673
B,42.4258156,-71.4697961,E,177.275358378,86.990054274
A,42.4249603,-71.4718835,E,5.89498775405,-8.14176188782
B,42.4258096,-71.4698169,E,175.567638244,86.3226967423
C,42.4250612,-71.4719070,E,3.96556656556,3.08096727238
B,42.4258002,-71.4698423,E,173.482251881,85.277169942
B,42.4257844,-71.4698752,E,170.781106299,83.5197951082
A,42.4249533,-71.4718875,E,5.56657649161,-8.92034567535
B,42.4257725,-71.4699134,E,167.644812293,82.1962026704
A,42.4249593,-71.4718775,E,6.38760516597,-8.25298814363
B,42.4257621,-71.4699483,E,164.779453538,81.0394496152
C,42.4250652,-71.4718962,E,4.85227694697,3.52587229326
B,42.4257428,-71.4699892,E,161.421491515,78.892782888
B,42.4257354,-71.4700408,E,157.185017926,78.0697085986
A,42.4249712,-71.4718775,E,6.38760455973,-6.92939570505
A,42.4249811,-71.4718875,E,5.56657525739,-5.82825577811
B,42.4257196,-71.4700683,E,154.927221836,76.3123337648
C,42.4250751,-71.4719009,E,4.4663933184,4.627012221
A,42.4249875,-71.4718989,E,4.63060218985,-5.11640774374
B,42.4257082,-71.4701032,E,152.06186203,75.0443544546
C,42.4250776,-71.4719097,E,3.74388844211,4.90507785935
A,42.4249949,-71.4719009,E,4.46639617527,-4.29333345435
B,42.4256928,-71.4701474,E,148.432954014,73.3314701226
```

161

```
A,42.4250038,-71.4719097,E,3.74389064574,-3.30341978243
B,42.4256814,-71.4701776,E,145.953474386,72.0634908123
B,42.4256666,-71.4702145,E,142.92391191,70.4173422336
C,42.4250845,-71.4719184,E,3.02959376933,5.67253902044
B,42.4256488,-71.4702655,E,138.736707624,68.4375148889
B,42.4256468,-71.4702963,E,136.207954562,68.2150623789
A,42.4250122,-71.4719117,E,3.57968467512,-2.36911923801
C,42.4250736,-71.4719184,E,3.0295940327,4.46017283768
B,42.4256235,-71.4703399,E,132.628312652,65.62349063
A,42.4250207,-71.4719123,E,3.53042272066,-1.42369606794
C,42.4250771,-71.4718969,E,4.79480451444,4.84946473184
B,42.4256072,-71.4703848,E,128.941928994,63.8105026687
A,42.4250281,-71.4719130,E,3.47295051783,-0.600621778551
B,42.4255924,-71.4704277,E,125.41974784,62.1643540899
B,42.4255790,-71.4704854,E,120.682447664,60.6739222688
A,42.4250395,-71.4719083,E,3.85883355824,0.667357531723
B,42.4255533,-71.4705551,E,114.959927788,57.8154075073
B,42.4255221,-71.4706142,E,110.10769713,54.3451483423
C,42.4250830,-71.4718949,E,4.9590099159,5.5056996379
B,42.4255047,-71.4706980,E,103.227519807,52.4098114995
A,42.4250513,-71.4719056,E,4.08051084634,1.97982734464
B,42.4254760,-71.4707462,E,99.2702002455,49.2176179729
B,42.4254607,-71.4708173,E,93.4327199278,47.5158562666
A,42.4250538,-71.4719130,E,3.47294980598,2.25789298299
B,42.4253944,-71.4709568,E,81.9794553419,40.1415555399
A,42.4250583,-71.4719184,E,3.02959440239,2.75841113218
B,42.4253756,-71.4710306,E,75.9202941616,38.0505019402
A,42.4250523,-71.4719190,E,2.980332848,2.09105359966
B,42.4253503,-71.4711238,E,68.2683436074,35.2364776805
B,42.4253206,-71.4712176,E,60.5671304972,31.9330578981
C,42.4250890,-71.4719009,E,4.46639282326,6.17305716963
B,42.4252954,-71.4713001,E,53.7936720593,29.1301562649
B,42.4252721,-71.4713578,E,49.0563568798,26.538584516
A,42.4250568,-71.4718976,E,4.73733330942,2.59157174885
B,42.4252236,-71.4714570,E,40.9117859231,21.1441111339
B,42.4251795,-71.4715670,E,31.8804978161,16.2390332746
A,42.4250622,-71.4718983,E,4.67986112656,3.1921935274
B,42.4251122,-71.4717434,E,17.3975820587,8.75350629282
B,42.4251048,-71.4717843,E,14.0395786108,7.93043200343
C,42.4250707,-71.4719063,E,4.02303824004,4.13761669748
A,42.4250617,-71.4719231,E,2.64371103356,3.13658039989
C,42.4250588,-71.4719050,E,4.12977229908,2.81402425969
B,42.4250924,-71.4718265,E,10.5748415427,6.55122643734
A,42.4250608,-71.4719345,E,1.7077388163,3.03647676974
A,42.4250578,-71.4719445,E,0.886710560248,2.70279800388
B,42.4250815,-71.4718332,E,10.0247536072,5.33886025458
B,42.4250751,-71.4718513,E,8.53869310768,4.627012221
C,42.4250573,-71.4719144,E,3.35800574686,2.64718487636
B,42.4250677,-71.4718674,E,7.21683815868,3.80393793161
A,42.4250533,-71.4719519,E,0.27914963181,2.20227985548
B,42.4250622,-71.4718815,E,6.05918861748,3.1921935274
C,42.4250687,-71.4719170,E,3.14453810282,3.91516418664
B,42.4250276,-71.4718983,E,4.67986241798,-0.656234906063
B,42.4250048,-71.4719278,E,2.25782877032,-3.19219352661
A,42.4250464,-71.4719606,E,-0.435145036625,1.4348186936
B,42.4250147,-71.4719680,E,-1.04270629511,-2.09105359966
B,42.4250419,-71.4719928,E,-3.07885650211,0.934300544413
A,42.4250414,-71.4719687,E,-1.10017806144,0.878687416901
B,42.4250726,-71.4719955,E,-3.30053336303,4.34894658265
B,42.4250840,-71.4719640,E,-0.714294468428,5.61692589293
A,42.4250345,-71.4719774,E,-1.81447287241,0.111226255814
B,42.4250890,-71.4719412,E,1.15764961033,6.17305716963
B,42.4250909,-71.4719164,E,3.19379923143,6.38438705481
A,42.4250291,-71.4719881,E,-2.69297342941,-0.489395522737
B,42.4250810,-71.4719036,E,4.24471550831,5.28324712707
A,42.4250197,-71.4720002,E,-3.68641817149,-1.53492232296
A,42.4250103,-71.4720129,E,-4.72912477956,-2.58044912319
A,42.4250028,-71.4720297,E,-6.10845320565,-3.41464603745
A,42.4249994,-71.4720478,E,-7.59451528085,-3.79281530595
```

162

```
A,42.4249944,-71.4720659,E,-9.08057755251,-4.34894658186
A,42.4249885,-71.4720820,E,-10.4024342677,-5.00518148872
A,42.4249860,-71.4720941,E,-11.3958792835,-5.28324712707
A,42.4249830,-71.4721075,E,-12.49605813,-5.61692589293
A,42.4249821,-71.4721142,E,-13.0461475094,-5.71702952229
A,42.4249791,-71.4721249,E,-13.9246486508,-6.05070828815
A,42.4249761,-71.4721330,E,-14.5896823377,-6.38438705401
A,42.4249726,-71.4721444,E,-15.5256556396,-6.77367894818
A,42.4249687,-71.4721544,E,-16.3466850017,-7.2074613434
A,42.4249657,-71.4721651,E,-17.2251863159,-7.54114010927
A,42.4249608,-71.4721779,E,-18.2761040224,-8.08614876031
A,42.4249568,-71.4721893,E,-19.2120776169,-8.53105378198
A,42.4249538,-71.4722047,E,-20.4764627029,-8.86473254784
A,42.4249474,-71.4722154,E,-21.3549647982,-9.57658058142
A,42.4249429,-71.4722295,E,-22.5126165239,-10.0770987306
A,42.4249395,-71.4722389,E,-23.2843844611,-10.4552679983
A,42.4249360,-71.4722463,E,-23.8919466409,-10.8445598917
A,42.4249330,-71.4722604,E,-25.0495982632,-11.1782386576
A,42.4249261,-71.4722764,E,-26.3632462963,-11.9456998194
A,42.4249227,-71.4722845,E,-27.0282806483,-12.3238690871
A,42.4249202,-71.4722919,E,-27.6358427936,-12.6019347255
A,42.4249157,-71.4723033,E,-28.5718171036,-13.1024528747
A,42.4249123,-71.4723127,E,-29.3435853737,-13.4806221424
A,42.4249108,-71.4723281,E,-30.6079707835,-13.6474615257
A,42.4249078,-71.4723395,E,-31.5439448942,-13.9811402916
A,42.4249034,-71.4723516,E,-32.5373914628,-14.4705358151
A,42.4248974,-71.4723650,E,-33.6375723448,-15.1378933468
A,42.4248949,-71.4723764,E,-34.5735464902,-15.4159589852
A,42.4248885,-71.4723931,E,-35.9446673619,-16.1278070188
A,42.4248870,-71.4724039,E,-36.8313795601,-16.2946464021
A,42.4248816,-71.4724199,E,-38.1450282672,-16.8952681806
A,42.4248751,-71.4724340,E,-39.3026818299,-17.6182388399
A,42.4248712,-71.4724468,E,-40.3536008356,-18.0520212359
A,42.4248657,-71.4724629,E,-41.6754601882,-18.6637656401
A,42.4248623,-71.4724763,E,-42.7756409513,-19.0419349078
A,42.4248578,-71.4724897,E,-43.8758221589,-19.542453057
A,42.4248529,-71.4725024,E,-44.9185315142,-20.087461708
A,42.4248504,-71.4725178,E,-46.1829180789,-20.3655273464
A,42.4248464,-71.4725286,E,-47.0696316075,-20.8104323673
A,42.4248415,-71.4725440,E,-48.3340192046,-21.3554410183
A,42.4248355,-71.4725588,E,-49.5491455512,-22.02279855
A,42.4248291,-71.4725715,E,-50.5918559189,-22.7346465844
A,42.4248256,-71.4725795,E,-51.2486811843,-23.1239384778
A,42.4248187,-71.4725923,E,-52.299602229,-23.8913996389
A,42.4248093,-71.4726043,E,-53.2848420583,-24.9369264391
A,42.4248044,-71.4726204,E,-54.6067023695,-25.4819350901
A,42.4247969,-71.4726332,E,-55.6576240429,-26.3161320052
A,42.4247925,-71.4726459,E,-56.700334139,-26.8055275279
A,42.4247870,-71.4726593,E,-57.8005169199,-27.4172719321
A,42.4247831,-71.4726721,E,-58.8514372393,-27.8510543281
A,42.4247747,-71.4726929,E,-60.559183895,-28.7853548726
A,42.4247697,-71.4727063,E,-61.6593667554,-29.3414861493
A,42.4247633,-71.4727217,E,-62.9237564489,-30.0533341828
A,42.4247568,-71.4727338,E,-63.9172063353,-30.7763048429
A,42.4247519,-71.4727525,E,-65.4525354935,-31.3213134939
A,42.4247479,-71.4727639,E,-66.3885121535,-31.7662185148
A,42.4247425,-71.4727794,E,-67.6611120718,-32.3668402933
A,42.4247370,-71.4727921,E,-68.7038236455,-32.9785846975
A,42.4247336,-71.4728089,E,-70.0831566131,-33.3567539653
A,42.4247296,-71.4728196,E,-70.961661425,-33.8016589869
A,42.4247232,-71.4728343,E,-72.1685798647,-34.5135070205
A,42.4247167,-71.4728538,E,-73.7695931393,-35.2364776805
A,42.4247118,-71.4728679,E,-74.9272491152,-35.7814863316
A,42.4247073,-71.4728813,E,-76.0274327973,-36.28200448
A,42.4247024,-71.4728947,E,-77.1276168044,-36.827013131
A,42.4246994,-71.4729148,E,-78.7778902433,-37.1606918969
A,42.4246935,-71.4729276,E,-79.8288132239,-37.8169268037
A,42.4246890,-71.4729430,E,-81.0932033844,-38.3174449521
A,42.4246841,-71.4729537,E,-81.9717094496,-38.8624536032
```

```
A,42.4246801,-71.4729725,E,-83.5152499114,-39.307358624
A,42.4246762,-71.4729846,E,-84.5086997721,-39.7411410201
A,42.4246707,-71.4729980,E,-85.6088847932,-40.3528854243
A,42.4246623,-71.4730201,E,-87.4233687871,-41.2871859687
A,42.4246603,-71.4730375,E,-88.8519639471,-41.5096384795
A,42.4246564,-71.4730550,E,-90.2887708296,-41.9434208747
A,42.4246484,-71.4730731,E,-91.7748426588,-42.8332309173
A,42.4246460,-71.4730865,E,-92.8750259463,-43.10017393
A,42.4246425,-71.4730979,E,-93.8110038976,-43.4894658242
A,42.4246405,-71.4731173,E,-95.4038055906,-43.7119183342
A,42.4246331,-71.4731327,E,-96.6681990803,-44.5349926236
A,42.4246217,-71.4731468,E,-97.8258617943,-45.8029719339
A,42.4246163,-71.4731569,E,-98.6551075078,-46.4035937132
A,42.4246074,-71.4731663,E,-99.4268838814,-47.3935073851
A,42.4246024,-71.4731777,E,-100.362863503,-47.9496386618
A,42.4245955,-71.4731864,E,-101.077166279,-48.7170998229
A,42.4245910,-71.4732018,E,-102.34155819,-49.2176179721
A,42.4245866,-71.4732132,E,-103.277537558,-49.7070134956
A,42.4245866,-71.4732360,E,-105.149489108,-49.7070134956
A,42.4245757,-71.4732387,E,-105.371176741,-50.9193796784
A,42.4245712,-71.4732514,E,-106.413890507,-51.4198978268
A,42.4245653,-71.4732615,E,-107.243137352,-52.0761327337
A,42.4245618,-71.4732749,E,-108.343322615,-52.465424627
A,42.4245569,-71.4732897,E,-109.558453581,-53.0104332781
A,42.4245485,-71.4733171,E,-111.808087592,-53.9447338225
A,42.4245430,-71.4733339,E,-113.187425698,-54.5564782267
A,42.4245405,-71.4733446,E,-114.065931857,-54.834543865
A,42.4245361,-71.4733594,E,-115.281062795,-55.3239393886
A,42.4245311,-71.4733755,E,-116.602928502,-55.8800706645
B,42.4250454,-71.4719076,E,3.9163053632,1.32359243858
C,42.4250454,-71.4719076,E,3.9163053632,1.32359243858
A,42.4245296,-71.4733929,E,-118.031525151,-56.0469100478
B,42.4249989,-71.4719988,E,-3.57147476733,-3.84842843347
C,42.4249989,-71.4719988,E,-3.57147476733,-3.84842843347
A,42.4245277,-71.4734030,E,-118.860769026,-56.258239933
B,42.4249632,-71.4720981,E,-11.7242929213,-7.81920574762
C,42.4249632,-71.4720981,E,-11.7242929213,-7.81920574762
A,42.4245252,-71.4734204,E,-120.289366699,-56.5363055713
B,42.4249246,-71.4722013,E,-20.1973175532,-12.1125392028
C,42.4249246,-71.4722013,E,-20.1973175532,-12.1125392028
A,42.4245331,-71.4734298,E,-121.06112895,-55.6576181544
B,42.4248910,-71.4723167,E,-29.6720021132,-15.8497413804
C,42.4248910,-71.4723167,E,-29.6720021132,-15.8497413804
A,42.4245410,-71.4734305,E,-121.11859354,-54.7789307375
B,42.4248583,-71.4724307,E,-39.0317473268,-19.4868399295
C,42.4248583,-71.4724307,E,-39.0317473268,-19.4868399295
A,42.4245425,-71.4734177,E,-120.067671551,-54.6120913542
B,42.4248365,-71.4725272,E,-46.9546911594,-21.911572295
C,42.4248365,-71.4725272,E,-46.9546911594,-21.911572295
A,42.4245485,-71.4734224,E,-120.453550673,-53.9447338225
B,42.4248108,-71.4726171,E,-54.3357597103,-24.7700870558
C,42.4248108,-71.4726171,E,-54.3357597103,-24.7700870558
B,42.4248207,-71.4726453,E,-56.6510595951,-23.6689471288
C,42.4248207,-71.4726453,E,-56.6510595951,-23.6689471288
A,42.4245574,-71.4734298,E,-121.061105488,-52.9548201506
B,42.4247989,-71.4727485,E,-65.1240989617,-26.0936794943
C,42.4247989,-71.4727485,E,-65.1240989617,-26.0936794943
A,42.4245692,-71.4734231,E,-120.511002992,-51.6423503376
B,42.4247791,-71.4728397,E,-72.6119039548,-28.295959349
C,42.4247791,-71.4728397,E,-72.6119039548,-28.295959349
A,42.4245791,-71.4734150,E,-119.845958017,-50.5412104099
B,42.4247425,-71.4729215,E,-79.327953506,-32.3668402933
C,42.4247425,-71.4729215,E,-79.327953506,-32.3668402933
A,42.4245871,-71.4734104,E,-119.468275937,-49.6514003681
A,42.4245796,-71.4734077,E,-119.246604597,-50.4855972824
B,42.4246979,-71.4730637,E,-91.0030376136,-37.3275312802
C,42.4246979,-71.4730637,E,-91.0030376136,-37.3275312802
A,42.4245737,-71.4734077,E,-119.246610208,-51.1418321885
B,42.4246712,-71.4731388,E,-97.168998512,-40.2972722968
```

```
C,42.4246712,-71.4731388,E,-97.168998512,-40.2972722968
A,42.4245692,-71.4734137,E,-119.739233385,-51.6423503376
B,42.4246445,-71.4732246,E,-104.213465209,-43.2670133133
C,42.4246445,-71.4732246,E,-104.213465209,-43.2670133133
A,42.4245623,-71.4734137,E,-119.739239974,-52.4098114995
B,42.4246237,-71.4732796,E,-108.729153763,-45.5805194238
C,42.4246237,-71.4732796,E,-108.729153763,-45.5805194238
A,42.4245534,-71.4734030,E,-118.860744663,-53.3997251714
B,42.4246049,-71.4733306,E,-112.916429503,-47.6715730235
C,42.4246049,-71.4733306,E,-112.916429503,-47.6715730235
A,42.4245608,-71.4734003,E,-118.639059128,-52.5766508821
A,42.4245663,-71.4733996,E,-118.581581718,-51.9649064778
C,42.4246108,-71.4733520,E,-114.673431009,-47.0153381166
B,42.4246019,-71.4733815,E,-117.09548143,-48.0052517894
C,42.4246019,-71.4733815,E,-117.09548143,-48.0052517894
A,42.4245727,-71.4733996,E,-118.581575665,-51.2530584443
B,42.4245900,-71.4733815,E,-117.095492543,-49.3288442271
C,42.4245900,-71.4733815,E,-117.095492543,-49.3288442271
B,42.4245801,-71.4733990,E,-118.532306782,-50.4299841549
C,42.4245801,-71.4733990,E,-118.532306782,-50.4299841549
A,42.4245821,-71.4734003,E,-118.639038974,-50.207531644
```