

# Designing and Constructing B.O.A.T.O.S

## A Buoyant Object for Autonomous Travel On a Surface

A thesis submitted in partial fulfillment of the requirement  
for the degree of Bachelor of Science in Physics from  
The College of William and Mary

by

Adam C. Abate and Simon R. Hetzler



Advisor: Prof. William E. Cooke



Advisor: Prof. Jonathan Frey



Senior Thesis Coordinator: Prof. Seth Aubin

Williamsburg, VA

May 2, 2019

# Designing and Constructing B.O.A.T.O.S.

A Buoyant Object for Autonomous Travel On a Surface

Simon Hetzler and Adam Abate

Advised Dr. William E. Cooke

With help from Jonathan Frey

27 April 2019

## Abstract

To design a buoyant vessel used for development of swarm behavior, we have first designed a prototype autonomous boat. Our prototype is equipped with an accelerometer/gyroscope/magnetometer module and a GPS module for navigation. Given a desired heading, a Voith-Schneider Drive (VSD) produces a thrust vector in any direction parallel to the surface of the water. Future students will use this prototype to make more vessels, using them to examine swarm behavior. In the future, swarms of Autonomous Surface Vehicles may automate undesirable jobs on the water such as long range shipping, oil spill and garbage clean ups, sample collecting and mapping, depth mapping of bodies of water, offshore maintenance, or smart fishing.

## 1 Introduction (Hetzler)

Our design project is motivated by its applications. In 2019, autonomous technology is being developed to automate work that people would otherwise do. Consumers are increasingly willing to buy robots and smart electronics to perform tasks they may consider mundane or repetitive. However, smart robots will surpass mere novelty marketed to consumers. Autonomous robots can perform jobs that are undesirable or have yet to be imagined. As storms (on the ocean) become increasingly dangerous, autonomous boats can be used for shipping or servicing offshore wind farms. Autonomous shipping boats would alleviate danger of pirates. By transitioning people away from these potentially dangerous jobs we allow (and should provide) these people with better quality of life. In a fully automated society people would not have to work for a living, rather autonomous robots (for farming, manufacture, and distribution) would provide for all people.

A swarm of robots can perform tasks that single robot would not be able to accomplish. For example, many ocean cleanup techniques involve finding a patch of high density trash (or oil) and gathering it by encircling it with a net or floating boundary. This is the kind of task that is simpler with several boats compared to one. Many boats can easily spread out to find where the spills occur. They can communicate and gather at the spill and encircle a patch of trash or oil. Many boats can evenly stretch nets and barriers for cleanups, realizing geometries that are inaccessible to a single or few cleanup boats. Since autonomous surface vessels (ASVs) do need to hold humans, they need not be shaped like conventional boats. Their form can be designed specifically for their task. Provided there is a source of power at sea (solar, wind or otherwise), autonomous boats can be away from land indefinitely to complete their task because robots do not need time off. Robots do not crave the land after weeks at sea or miss their families. Autonomous swarms can be used to gather data. Part of our motivation to do this project here at William and Mary in Williamsburg, Virginia is to make a depth map of the Matoka Lake. Additionally, ASVs could collect water/plant samples across a large and detailed area. We believe that these potential applications justify the development of a autonomous surface vessels. Though ours is not seaworthy, if this project were scaled up in size and number of ASVs, the software developed by us and any successors could be a framework that can direct any fleet of ASVs.

Another use-case of a swarm of small boats is in topographically mapping the bottoms of bodies of water. This has been recommended by Professor Donglai Gong as a possible end goal of William Mary research into autonomous swarms. Such a use-case retains the advantages listed in the ocean cleanup example, and

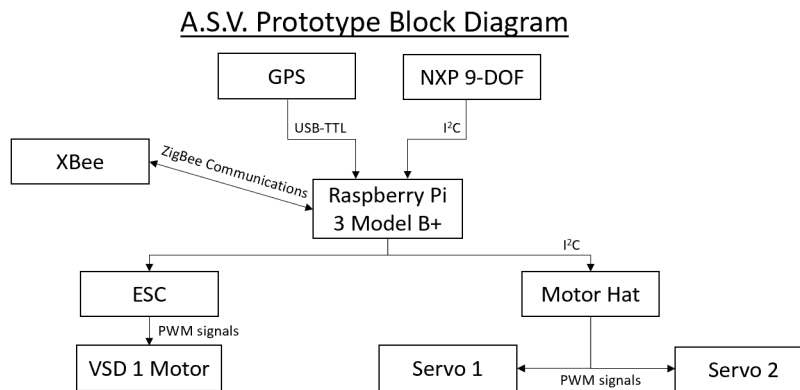


Figure 1: A block diagram of communications to and from the Raspberry Pi

though it may be less impactful, the mapping application is one that future students could implement at William & Mary’s Lake Matoaka. As described below, the components of our prototype vessel are motivated by these use-cases.

For our project, we are designing a small autonomous boat. The vessel is currently programmed to travel North, but may be easily modified for waypoint finding. This is but one step on the path to an autonomous fleet. We complete this in good faith that future students will continue the project, building upon the ground work we have laid in tasks such as obstacle avoidance and multi-boat coordination.

A Raspberry Pi 3 B+ is at the heart of our design. This allows us the processing power needed to read data from our various types of sensors and control six motors. Figure 1 is a block diagram that shows how the pi communicates with and controls each module in our design. The system that the Pi coordinates has numerous hardware and software requirements. These are enumerated below, along with our current solutions to them.

- To know the boat’s location, we use a GPS module.
- With a Magnetometer/Accelerometer/Gyroscope (NXP 9-DOF) module, we determine the boat’s direction.
- A camera can help the vessel avoid obstacles via object recognition, though we have not proceeded with the camera module and code beyond taking pictures.
- The pi operates its brushless motor (VSD) with an electronic speed controller (E.S.C.) and steers them using a motor hat connecting to 2 servos.
- In the future, a Zigbee could be used to give new waypoints or a stop command directly to the pi. For now, communication between Pi and ground station occurs via WiFi.

During operation, the pi continuously runs a master program to call all protocol and data needed to run the boat. The master program includes protocols to read each sensor. Code for these functions is stored on the Pi and referenced when needed by the master code. Information from the sensors is stored in a globally-accessible CSV file on the Pi. The master code reads these global variables and uses yet another protocol to operate the motors such that the boat is steered to the desired waypoint.

In the following sections we detail how each module is used and how the working code is structure. In the last section, we suggest starting points for other students to investigate. This thesis is a summary of our senior research project at William and Mary so it is right to credit us both Adam and me as authors. However, obviously we did not each collaborate on every sentence. For clarity the primary author(s) for each section (or subsection) are/is listed after the title in parenthesis (as suggested by Dr. Cooke). Please note that we have both read and edited this document in its entirety and it would be misguided to say that when a section is credited to me that Adam has necessarily not influenced it and vice versa.

## 2 System Components (Abate & Hetzler)

The purpose of this Section discuss each module in the boat. We examine the Raspberry Pi, the Voith-Schnieder drive, the GPS, the accelerometer/gyroscope/magnetometer, the camera, Zigbee communication, the hull and power supply. For each piece that interfaces with the raspberry pi, we examine the command protocol and hardware needed for full integration.

### 2.1 Raspberry Pi (Abate)

The Raspberry Pi (Pi for short) is a small computer that serves as a center of communication for each sensor that is added. We chose the Pi above other similar micro-controllers (i.e. Arduino or Mbed) because it can easily run a python script to communicate with our sensors using I<sup>2</sup>C, SPI, USB-TTL, CSI and Pulse Width Modulated Signals. Our Pi is powered by the 10 amp hour battery and connected to each sensor as described in the following sections. Our data-collecting and motor-running loops are stored and run by the Pi.

### 2.2 Voith-Schnieder Drive/Servo Setup (Hetzler)

#### 2.2.1 Method

To move our boat, we use a Voith-Schnieder Drive (VSD, pictured on the following page). Despite their complexity, VSDs are an elegant alternative to traditional propeller drives. Traditional motors create a thrust vector in one direction relative to the motor then move the motor to change the thrust relative to the boat. Alternatively a rudder can be used to change the direction of the water moved by the propeller and hence the thrust vector of the boat-motor-rudder system. Instead VSDs can create a thrust vector in any (horizontal) direction relative to the drive (which is oriented vertically in the hull). A VSD operates by spinning 5 foils around an axis perpendicular to the surface of the water. To create a thrust vector the foils are each rotated on their own axis (still perpendicular to the water) as they rotate around the central axis. The thrust vector is controlled by a central joystick or pin. The X,Y coordinate of the joystick maps to the set of all possible thrust vectors by a cross product. Suppose we define a vector that comes out of the surface of the water as  $\vec{k} = [0, 0, 1]$  and the expression of any normalized vector on the surface as  $[x, y, 0]$  such that  $x^2 + y^2 = 1$ . Given a desired normalized thrust vector,  $\vec{t} = [x', y', 0]$ , we can write the direction of our pin as a unit vector  $\vec{p}$ :

$$\vec{p} = \vec{k} \times \vec{t}$$

Where  $\times$  is a cross product. Conversely, given  $\vec{p}$  but not  $\vec{t}$  we can find  $\vec{t}$  using:

$$\vec{t} = \vec{p} \times \vec{k}$$

Figure 3 below is a bottom view of the VSD. Take some time to examine it now. Putting the pin in one location, each of the foils move around their own axis as foils spin around the central axis of rotation. Each of the three drawings are freeze frames separated by 1/10th of a turn. Over 1/5th of a turn the foils rotate such that the net force vector is 90° from the pin. After the drive has completed one cycle it is reset to exert another push. These pushes occur continuously given constant rotation. As a result of this radical design, the VSD can create a thrust vector in any direction parallel to the surface of the water.

For our prototype boat, we use one VSD with a drag fin attached to the stern of the boat so that the back tends to point opposite of the thrust vector. To run the VSD, we need to be able to A) spin the brushless motor that rotates the foils and B) move a central pin to the X,Y coordinate that results in the expression of our desired thrust vector.

A) Brushless motors work when an alternating current is applied through the central armature (copper coils) creating an alternating magnetic field and an electromotive force. The magnetic field exerts torque on the alternating north/south magnets around the armature. The speed of the motor is proportional to the rate and amplitude of the supplied alternating current. One way to spin a brushless motor is to use an electronic speed controller(ESC) as shown in figure 2. The ESC supplies a pulse width signal that is calibrated by the inductance of the motor. When the motor is attached, the ESC generated three approximate pulse waves with a phase offset of 120 degrees. The speed of the motor depends on the frequency of these pulse waves.

B)To move the central X,Y pin of the VSD, we use two servo motors (one per axis) attached to the pin by two orthogonal rods (one per servo). Servo motors work by mapping a supplied pulse width signal to the

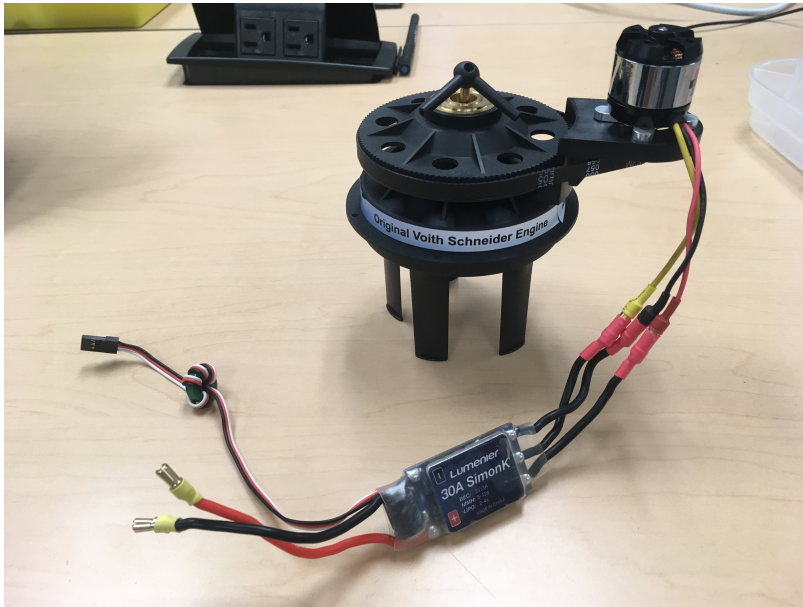


Figure 2: A Voith-Schneider Drive connected to an electronic speed controller. The red and black lead to the left are power leads. The small wires to the right read signals from the raspberry pi. The three leads to the right supply the 120 degree phase shifted pulses to power the brushless motor.

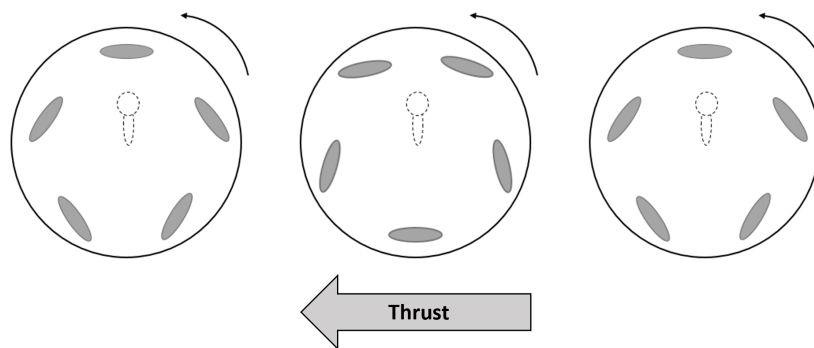


Figure 3: These three images show a bottom view of the same VSD in three stages of a cycle. Each image is separated by 1/10th of a full rotation. For this example the pin is set at north. The resulting thrust vector is a sum of thrust vectors from each foil.

domain of the servo arm's angles between 0 and 180 degrees. Our servos read pulse width signals at 60 hz. If the pulses are divided into 4096 "ticks", where each tick corresponds to about 0.41% of a duty cycle or  $4.07\mu s$ . A duty cycle of 375/4096 ticks high corresponds to 0 degrees and 600/4096 ticks high corresponds to 180 degrees. Intermediate positions are mapped linearly to this range. When supplied a pulse width in this range, the servos move to their mapped position and pull the center pin to its desired location. Since the center pin has a relatively small range of motion, it needs to be very precisely positioned. To achieve this precision we intend to use a mechanical advantage (a lever) such that large movements on the servo are translated to small and exact movements on the center pin.

The VSD uses these mechanisms simultaneously to move our boat in the desired direction.

### 2.2.2 Results

Recall we have opted to control the VSD's brushless motor with an Electronic Speed Controller (ESC). We operate the ESC using a piece of code that we borrowed from an online source [AGTx] . The code has a calibration phase, an arming phase and a run phase among other functionality. Our final integrated code involves these three steps. The adapted python3 code from author AGTx can be found in our github repository in the control folder under motor\_controller.py.

For each pulse width inputted to the ESC, the ESC outputs three 120-degree-out-of-phase signals that power the brushless motor. The frequency of these pulses determines motor speed. The following figures demonstrate that the ESC increases output frequency as the average of the input signal increases. The images also demonstrate, however, that significant noise appears in ESC outputs as the input signals approach very low and very high pulse widths. This explains why the brushless motor ceases to function at inputs that are very low and very high, yet still producible by the Pi.

The x-axis of each of these scope traces is time. The y-axis of each of these scope traces is voltage, though note that the zero-voltage centers of the traces have been shifted to distinguish the traces.

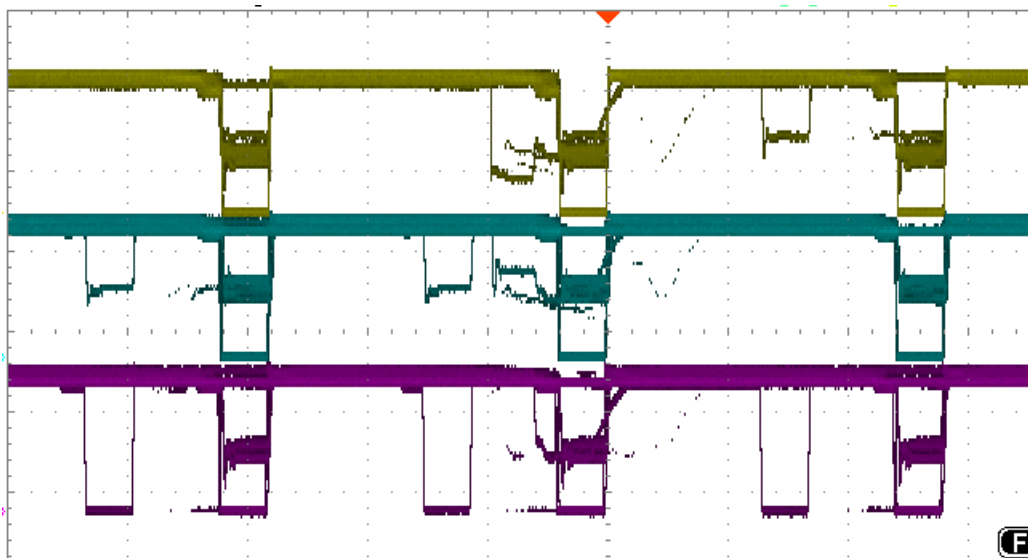


Figure 4: The pulses sent from the ESC to the VSD brushless motor under a Pi-to-ESC pulse width of 800 ms. Note that the three VSD inputs are noisy and in phase. The 800 ms pulse width is the point below (and including) which the VSD ceases to turn. Given that the DC brushless motor of the VSD requires three out-of-phase inputs, it is intuitive from this graph that the VSD should not turn.

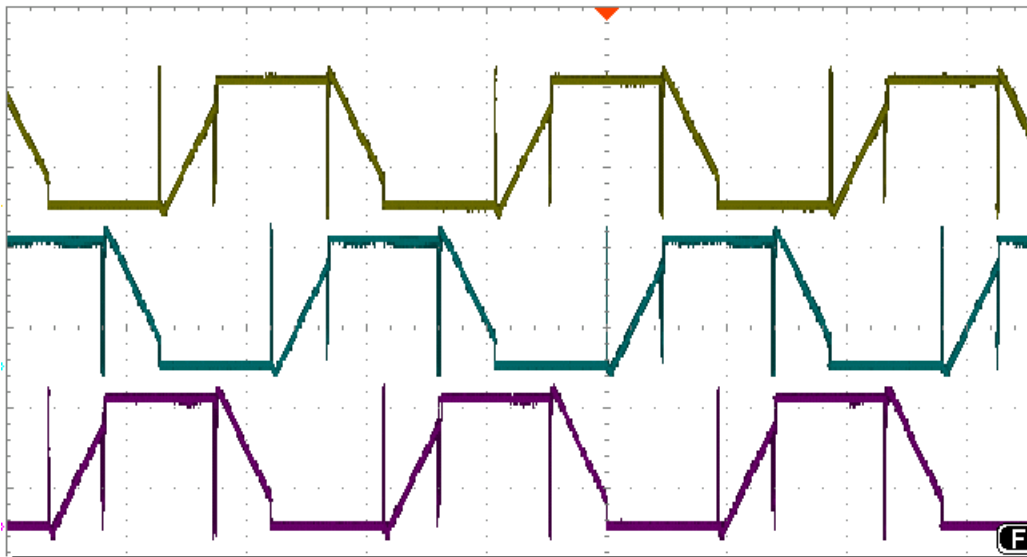


Figure 5: The pulses sent from the ESC to the VSD brushless motor under a Pi-to-ESC pulse width of 1400 ms. Of these three figures, this one's traces clearly have the least noise and are 120 degrees out of phase. This results in the most consistent VSD speed.

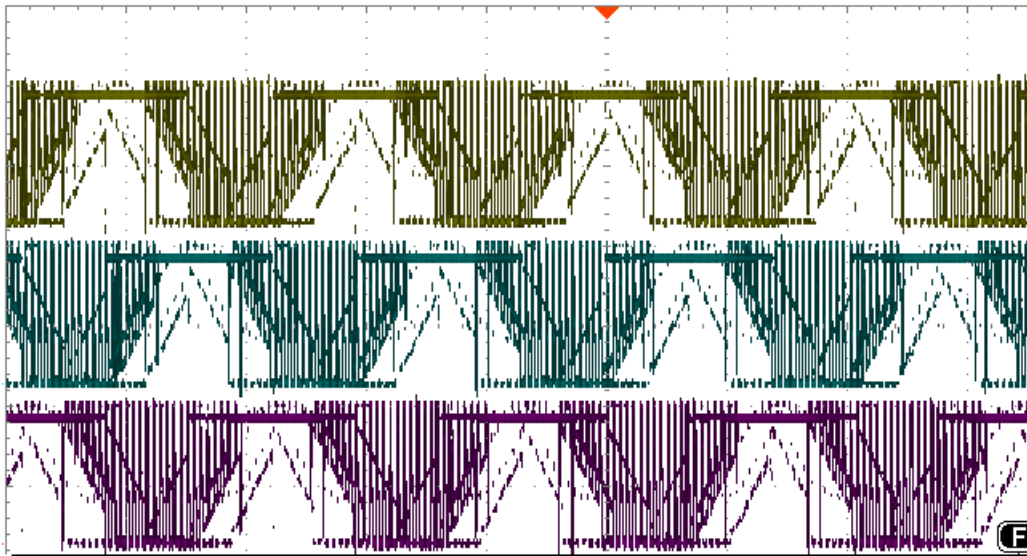


Figure 6: The pulses sent from the ESC to the VSD brushless motor under a Pi-to-ESC pulse width of 2000 ms. The traces here still are 120 degrees out of phase. As expected, they have a higher frequency than in the previous two figures (Pi-to-ESC pulse width of 800 ms and 1400 ms), resulting in a higher rate of motor rotation. However, this Pi-to-ESC pulse width of 2000 ms clearly results in more noise in the VSD input than does the 1400 ms Pi-to-ESC pulse width. Indeed, above a Pi-to-ESC pulse width of 2000 ms, the VSD ceases to turn, likely because the noise obfuscates the coil phase differences necessary to turn the VSD.

To run the servos, we use a servo motor hat specifically designed for raspberry pis from Adafruit.com. The hat is another board that nests onto the pi's pins and converts the signals it needs to pulses that can drive the servos. Essentially the hat converts I<sup>2</sup>C signals to puslewidth modulated signals. We use the word hat because it rests on top of the pi and it looks like the pi is wearing a "hat". The servo hat connects directly to an external power source, and prevents high current from being drawn through the pi. The current needed to run four servos has the potential to surge to levels that would damage the circuitry of the pi. Using the servo hat also streamlines the process of writing code. We interface with

the hat using the following python code adapted from a code written by Tony DiCola [DiCola]. It is designed to turn the servos and express a thrust vector that the user enters. In our final integrated code, this vector is the difference between the boat's position and the target waypoint. The code is given in our git hub repository in the control folder under `direc_controller.py` [https://github.com/AdamAbate-6/boat-os/blob/master/control/direc\\_controller.py](https://github.com/AdamAbate-6/boat-os/blob/master/control/direc_controller.py). Figure 7 shows some annotated (and offset) oscilloscope traces of example signals that

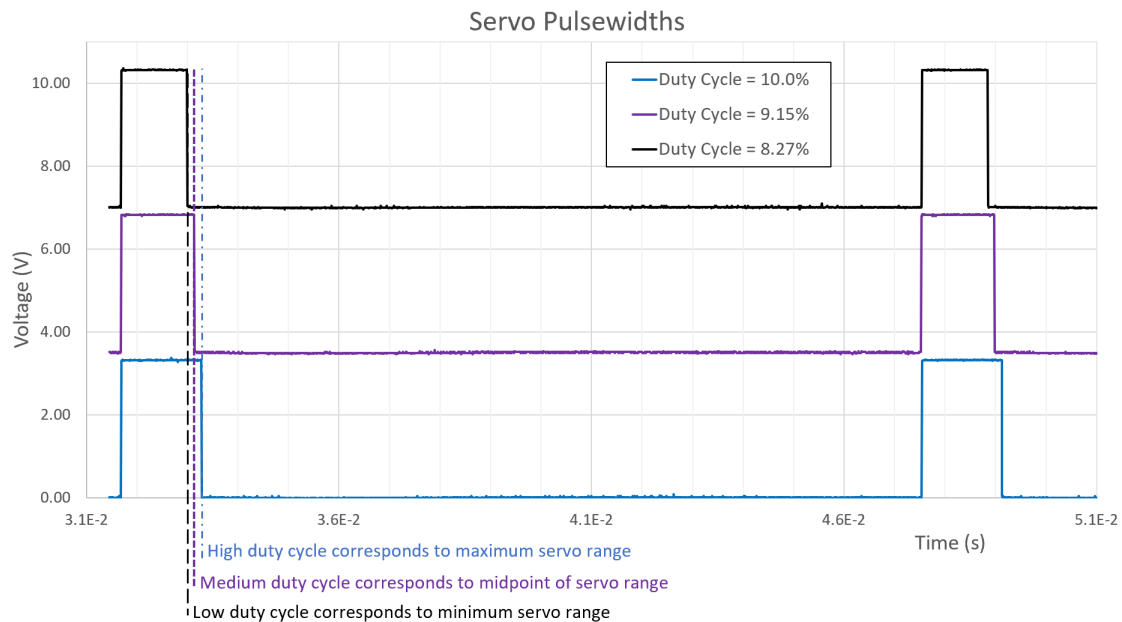


Figure 7: This image shows oscilloscope traces of example pulse width signals that could be sent to our servo motors. The signals are sent at 63 hz with a low of 0 Volts and a high of roughly 3.3 volts. The 3 signals are offset in this figure by 0V, 3.5V and 7V respectively. We will be sending signals with a duty cycle between 10% and 8.3%. We show the maximums of this range and a trace of the midpoint of this range which will result in no thrust in that direction.

Figure 8 shows the servos connected to the VSD inside our boat hull. As the two servos move, the central pin is moved to various XY positions.

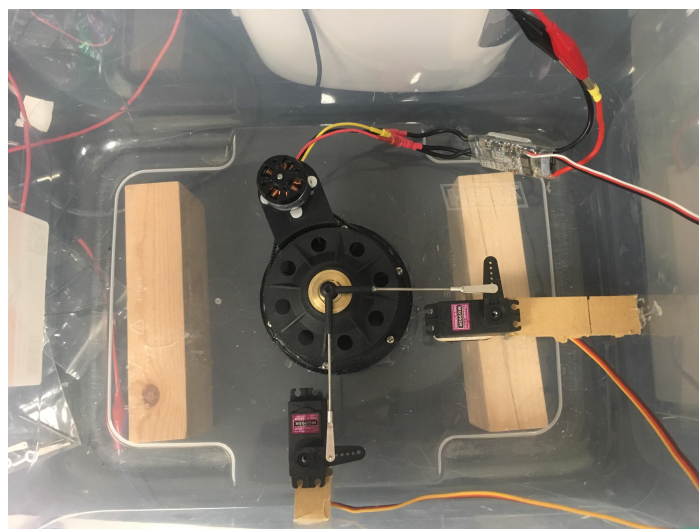


Figure 8: A test bed to show the connection between servos, ESC and VSD. Moving the servos will move the VSD's central pin.



The servos will pull and push on the pin of the VSD orthogonal to each other. This way, with two servos, we can push the pin to any position such that the VSD exerts our desired thrust vector.

### 2.2.3 Future

Although our design only incorporates one VSD and two servos. For more nuanced maneuvering like pivoting, future students may consider adapting our code for navigation with two VSDs and 4 servos. This added agility will be beneficial for swarm maneuvering and avoiding obstacles.

## 2.3 GPS (Abate)

### 2.3.1 Method

A GPS allows the vessel to know its position absolutely, rather than relying on estimates of distance traveled from a known starting point (dead reckoning). Given the difficulty of such estimates when traveling on the surface of water, GPS positions measurements likely have lower error than position measurements produced by dead reckoning.

Initially, we chose the Adafruit Ultimate GPS Breakout [Adafruit, a] because of its ease of integration with the Pi. An Adafruit Python library for the module simplifies the module's software requirements. The module uses TTL Serial communication to transmit data. For simplicity, we attempted to use a TTL Serial-to-USB cable to connect the module into one of the Pi's four USB ports. Due to the Adafruit module's failure (explained below), we now use a 3DR GPS, which communicates with the Pi via I2C and the Python library "gps".

Most standard GPS modules give position measurements accurate to within 3 meters. For topographic mapping applications, however, this uncertainty is unacceptable. Rather, we intend to use differential GPS to reduce uncertainty by one or two orders of magnitude.

### 2.3.2 Results

We initially tested the Adafruit GPS module with an Arduino microcontroller. Jonathan Frey connected the module to the Arduino's Universal Asynchronous Receiver/Transmitter (UART) serial port. The Adafruit example code produced latitude and longitude output, but these were all zero. We initially interpreted this as the GPS not having a fix, but as further discussion below shows, it was a first indication of the module's faultiness.

We attempted to connect the Pi to the GPS' USB port via a TTL-to-USB adapter. Querying the Pi's USB connections in its Terminal, we saw that the Pi recognized the make and model of the adapter; the Pi did not, however, receive any data from the GPS. We did not observe GPS data either using the Python "gps" library or by looking at output of the serial port directly in Terminal. Not being a software issue, this was a problem either with the chip's physical location or with chip-to-Pi communications. We ruled out the former by attaching u.FL to SMA adapter to the chip, allowing us to connect a longer-range antenna. We observed that, after, a calibration period, the module's red LED started blinking at 15-second intervals, indicating a satellite fix. Knowing now that the problem lay in chip-to-Pi communications, we tested transmitting serial data to the Pi's UART port (rather than USB). This, however, was not successful. We even adjusted baud rate, but had no success receiving data on the Pi. In light of this, we inspected the Adafruit example code used in the Arduino test and found that actual data transmission between Pi and module is not necessary to produce the output we saw. The issue, then, was not with GPS location and likely not with all of the Pi's serial ports, so we concluded it must be a GPS module issue. Using an oscilloscope, we observed what appeared to be bits coming from the module's TX pin, but for whatever reason, the Pi could not "read" these bits. To ensure this wasn't really a Pi issue, we next tested a different GPS Module: the UBlox 3DR GPS module.

We scavenged the 3DR module from an unused quadcopter. This module, pictured in Figure 9, has both GPS and magnetometer capabilities, though we only used the latter. Looking at Figure 9, one can see that only four wires are relevant of the six ports this module makes available. We connected these four wires to the TTL-to-USB adapter, whose USB end we connected to the Pi. Again using the Python "gps" module, we successfully observed latitude and longitude data. The issue with the Adafruit module must therefore have had to do with its output – a hardware issue that we could not fix. Until more vessels are needed, we advocate continued use of the 3DR module.



Figure 9: The U-blox 3DR GPS and Magnetometer module

### 2.3.3 Future

Once we have GPS data from the Pi, there are a couple ways in which we could increase accuracy of our position estimations: differential GPS (DGPS) and Kalman Filtering.

DGPS (illustrated in figure 10) is a way of reducing uncertainty in GPS position from meters to decimeters. It requires a base station with a known location that can communicate with the vessels. We must ensure that the vessels' GPS modules and the base station GPS module are connected to the same satellites, which is doable using the RTKLIB library [Maker, 2016]. This also requires the vessels and base station be within several hundred miles of each other – not an issue for lake use cases. Given these assumptions, the difference between the base station's actual and measured locations should be the same difference as for the vessels. This difference can be represented as a vector with components in the N and E directions. Subtract these components from the corresponding components of the vessel's measured position vector, and the result is a greater-accuracy estimate of the vessel's position.

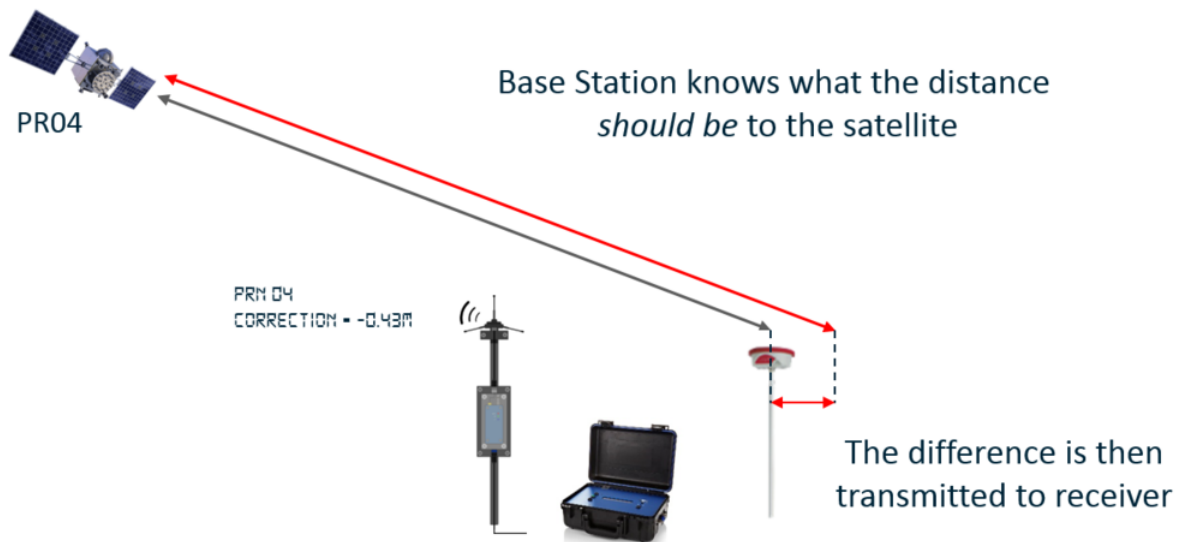


Figure 10: An illustration of the DGPS concept, where known position of the base station helps correct for the satellite’s mismeasurement of the vessel’s location. [RACELOGIC, 2018].

Kalman filtering is a method of combining dead reckoning data (e.g. accelerometer data, velocity estimates from GPS) with GPS position data to estimate current position better than either dead reckoning or GPS positioning would do alone. It does this by creating a weighted average of predictions (from dead reckoning) and position measurements (from GPS). The weights measure how noisy the average’s components are by considering their covariances. For more reading, Welch and Bishop of UNC provide a mathematical introduction [Welch and Bishop, 2006].

In the short term, before pursuing DGPS or Kalman filtering as is addressed in section 2.3.3, we are collecting a large set of GPS data in order to observe patterns in latitude and longitude change. By simply averaging latitude and longitude over small time scales, we may eliminate any random components of the error.

## 2.4 Magnetometer/Accelerometer/Gyroscope (Hetzler & Abate)

### 2.4.1 Method

This subsection combines three functionalities because we chose a single module that could execute each: the Adafruit Precision NXP 9-DOF Breakout Board [Adafruit, b]. This module communicates with the Pi via I2C. This board really contains two chips: an FXOS8700CQ magnetometer/accelerometer and an FXAS21002 gyroscope. Since our goal is to have a waypoint finding robot, we first investigate the functionality of the magnetometer in the FXOS8700CQ chip. The magnetometer is useful for establishing the boat’s orientation in the earth’s magnetic field. Future magnetometer work could include using the accelerometer for dead-reckoning. Eventually, the gyroscope could be used for creating a still image in other conditions than flat water but for now we do not use it.

The magnetometer reads Earth’s magnetic field. This helps the vessel determine its orientation relative to a target position at a given attitude and longitude. Depending on target distance and initial test results, this orientation may require adjustment to account for the difference between magnetic and true poles. Our vessel can translate the orientation directly into a thrust vector, which is translated into commands for servo motors directing the VSDs.

Our primary method of determining location is GPS. However, if we instruct the raspberry pi to integrate the acceleration data twice with respect to time, we can draw a map for displacement vs time. Given an initial position (provided by the GPS or otherwise), we can calibrate our displacement vs time to our actual position. Using this method we can check our final position against the GPS readings. Should these readings disagree, we could compare the two data sets and make a decision about which one should be trusted more. One possible technique for reconciling these sensor inputs is Kalman Filtering. This is left for future students to work on.

The gyroscope could assist in calibration of topographic sensing. If the boat is rolling, pitching, and yawing, any image of the bottom will be at an angle relative to the true representation. The gyroscope measures roll, pitch, and yaw, and so could help adjust the image to account for such motion. Again, in the interest of a timely prototype, we leave this to future students.

Figure 11 shows the Magnetometer/Accelerometer/Gyroscope. This module is mounted on our boat such that when it is floating on the surface, the z-axis points upwards. The x and y axes are oriented so that they evenly straddle the bow. This allows us to find north via the maximum of the magnetic field felt in the x and y directions.

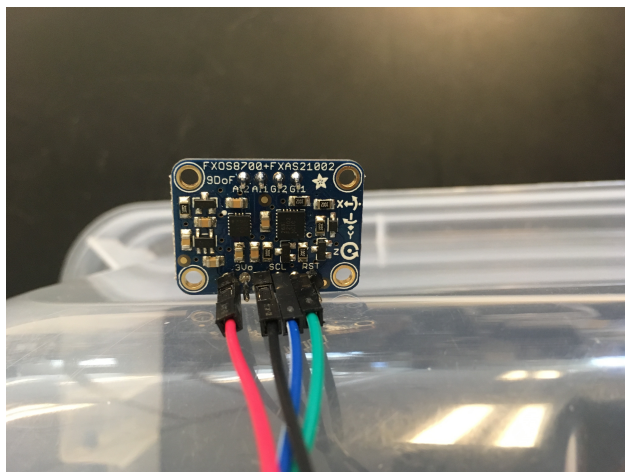


Figure 11: We use the Adafruit 9DOF NXP module which houses both the FXOS8700CQ and FXAS21002 chips in one board.

## 2.4.2 Results

We have used the Adafruit FXOS8700 library to read values from the magnetometer. To test our sensor, we set it to collect almost 1931 data points over 19.31 seconds. As we started collecting the data the magnetometer was spun and twisted in an attempt to trace out a sphere. In a perfect experiment we would hit all the possible orientations of this device and determine if the output of the magnetometer needed adjustment. With the help of Dr. Cooke, we used statistical analysis to calculate corrections for our magnetometer. There are two types of corrections: hard iron ( $\beta_i$ ) which centers the data and soft iron ( $\alpha_i$ ) which scales the data. These corrections are applied to a magnetometer reading ( $m'_i$ ) to retrieve a corrected magnetometer reading ( $m_i$ ) as is shown below. In this notation, the index  $i$  shows that this can be used in different dimensions. This magnetometer measures  $m_x$ ,  $m_y$  and  $m_z$ .

$$m_i = (1 + \alpha_i) * (m'_i - \beta_i)$$

To find the values for  $\alpha_i$  and  $\beta_i$ , we examine the total field ( $m_t$ ). Where

$$m_t = \sqrt{m_x^2 + m_y^2 + m_z^2}$$

We then vary parameters  $\alpha_i$  and  $\beta_i$  to minimize

$$\frac{\sigma_{m_t}}{\bar{m}_t}$$

To find hard and soft iron corrections, we took data for roughly 20 seconds. To get a range of readings, we spun the magnetometer (as far as the wires would allow) in as many directions as possible. In figure 12, we show 4 plots. 1) Mag x vs Mag y shows magnetometer data in the x-y plane. 2) Mag y vs Mag z shows magnetometer data in the y-z plane. 3) Mag x vs Mag z shows magnetometer data in the z-y plane. 4) Corrected Total Field shows the total magnetic field after hard and soft iron corrections have been applied.

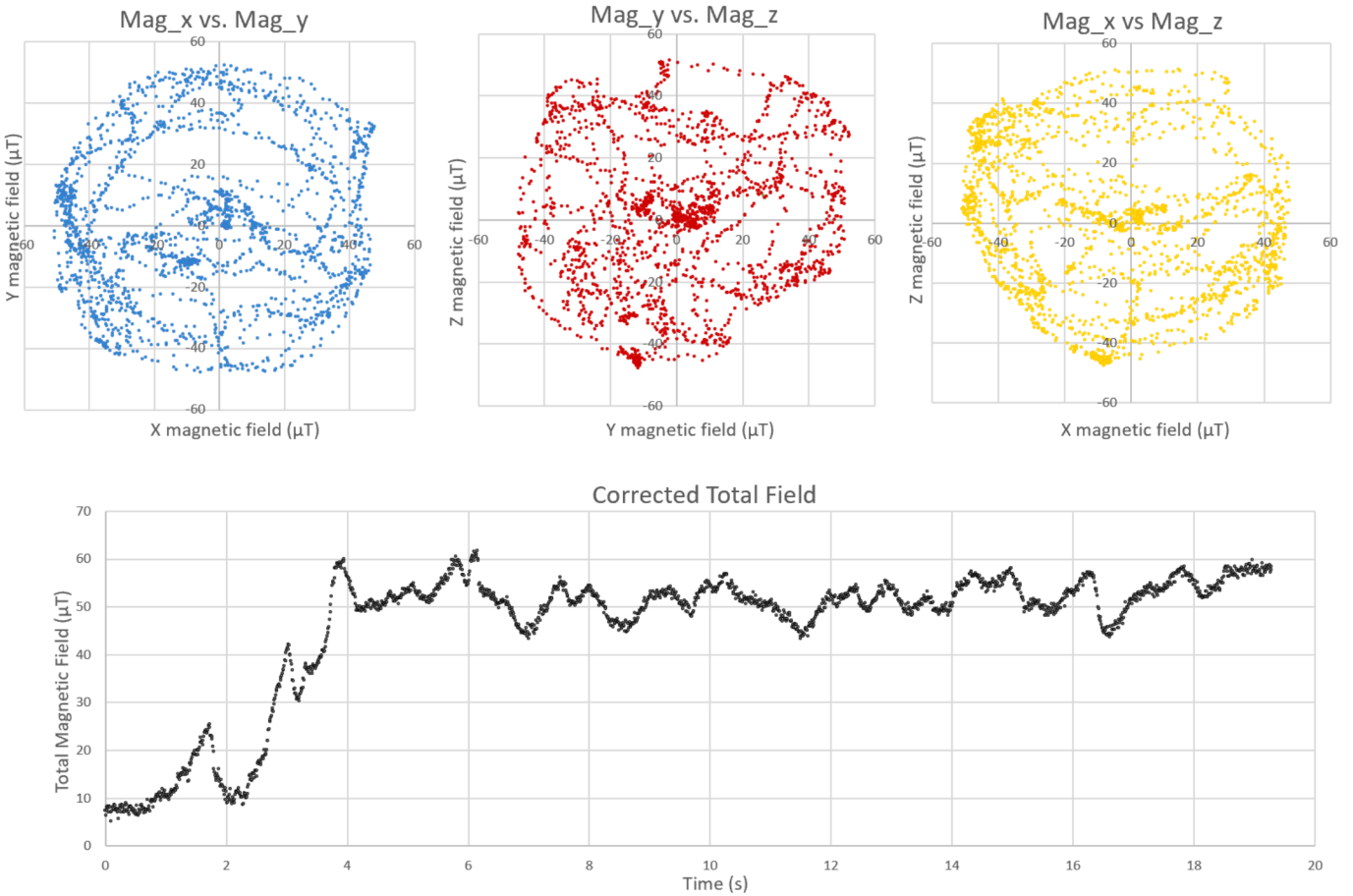


Figure 12: 4 graphs describe a test of the magnetometer. From this test, we retrieve our hard and soft iron corrections.

From these data we retrieve  $\alpha_x = 0.062$ ,  $\alpha_y = 0.039$ ,  $\alpha_z = 0.038$ .  $\beta_x = 0$ ,  $\beta_y = 1.55$ ,  $\beta_z = 7.75$ . These graphs demonstrate pretty good magnetometer response. The distributions are roughly symmetric around 0 and the minimum and maximum readings are roughly a  $-50\mu\text{T}$  for each axis. Although the data is not perfectly circular, this is not entirely surprising. The magnetometer was attached to wires running in the  $+z$  direction. This made spinning the magnetometer easy in the  $xy$  plane but difficult in the  $xz$  and especially difficult in the  $yz$  plane. I suspect this is why the  $xy$  data looks pretty good but the  $xz$  and  $yz$  data is a little uneven. It is plausible that the magnetometer was not spun evenly in every direction.

For short-term ease, we have placed the 9-DOF module on a mast (see Figure 16) above the vessel. Ideally, this has isolated it from most VSD and servo magnetic field effects.

Indeed, in implementing north-seeking functionality, we have found that the mast results in less erratic VSD direction changes than if the module were in the vessel. The north-seeking functionality is simple to implement: input a direction vector to the VSDs that is opposite the north-pointing vector output by the magnetometer. This works if 1) the magnetometer is parallel to the VSD servo arms; and 2) if the  $x,y$  reference frame of the magnetometer data is the same as the  $x,y$  reference frame used by the servos. The latter condition is not satisfied, but has nearly been accounted for. Inexact servo alignments and rotational servo arm motion make matching these reference frames difficult. At the very least, our vessel pursues a fixed cardinal direction.

### 2.4.3 Future

Now that we have used magnetometer data to create a north-seeking vessel, the most important remaining task is to combine accelerometer data with GPS data in Kalman Filtering. This will improve accuracy of

the vessel’s location estimates. Beyond this, gyroscope use is based on vessel application, as discussed in subsection 2.4.1.

## 2.5 Camera (Abate)

### 2.5.1 Method

Three requirements of autonomous navigation necessitate a camera: first, that we can identify obstacles near the boat; second, that we can check the IR distance sensor readings; and third, that we can troubleshoot boat issues remotely.

We chose to use the Camera Module V2 produced by the Raspberry Pi foundation. Because it is produced specifically for the Pi, this module is easy to use. The Pi has a designated port for this module between the Ethernet and HDMI ports. Communications between the Pi and the camera module rely on the MIPI CSI-2 protocol [Alliance] [Picamera]. Given the size of picture data, this protocol is likely more efficient than using SPI or I2C communications with the Pi’s GPIO pins. The module also requires little software, as the picamera Python library abstracts away pin-level communications into a simple set of picture-taking methods.

In order to satisfy the requirements that motivated our use of cameras, multiple modules are needed to monitor all areas around the vessel for obstructing or approaching objects. This is problematic, as a Pi only has one of the aforementioned camera ports. One solution is to use multiple Raspberry Pis – to network cheaper Pi models with cameras to a master Pi on the boat. Another option is to use the Arducam Multi Camera Adapter Module [Arducam, 2015]. One downside of this is the adapter’s use of many GPIO pins, which we also need to control the VSDs, IR distance sensors, a 9-DOF module, and Zigbee modules. We advise future students to try the Arducam module first for simplicity.

### 2.5.2 Results

As discussed in section 2.5, the picamera library simplified the software needed to take pictures. After connecting the module to the CSI port of the Pi, we may run the code in Appendix A to take 5 pictures. This is easily modifiable to take any number of pictures, with any length delay in-between the pictures.

### 2.5.3 Future

Above, we discussed three elements of autonomous navigation necessitating a camera. Here, we go over methods of accomplishing each.

The Python OpenCV library has unsupervised learning algorithms that can detect contours in an image. That is, if we can ensure it ignores the air-water boundary, OpenCV will identify the most distinguished objects in an image. Its `findContours()` method does this by identifying changes in pixel intensity and RGB values. This is one straightforward approach to object identification, but it fails to be useful under a couple scenarios: one, if it identifies the water-air boundary as marking an object; two, if the camera is so close to an oncoming object that it identifies no object.

If, however, contours are successfully drawn around relevant objects in an image, we can calculate the pixel widths of these objects. Using these, and if we know the camera focal length (which can be determined experimentally) and the actual widths of those objects, we can calculate distances to those objects by the following equation:

$$D = \frac{W_{act} * F}{W_{pix}} \quad (1)$$

Here,  $D$  is the distance to that object in the same units as  $W_{act}$ , which is the object’s actual width.  $F$  is the focal length, and  $W_{pix}$  is the pixel width of the object in the photo. Assuming a circular boat hull, we then know the actual widths of boat hulls and therefore can tell the distance to them.

This method for determining distance-to-object does not work when we do not know the object’s actual width or length. Another option is to determine the object’s contour’s lowest y-value in the image, which is (in calm water) its intersection with the water. Knowing the camera’s height above the water and the camera’s focal length, it should be possible to devise a function that relates y-position in the image to distance from the camera. This requires more research.

As for troubleshooting vessel issues, there are several possible examples. For instance, if the accelerometer detects a collision, but in previous seconds the cameras had not identified any objects nearby, one explanation is that the vessel ran aground. This can be cross-checked with GPS data. Another example is listing detection. If the water-air boundaries of two pictures taken from cameras on opposite sides of the boat are different (beyond the gyroscope's prediction), the vessel is listing. There are more ways to cleverly interpret pixels to help troubleshoot vessel issues. Future students should explore these as the project progresses.

## 2.6 Zigbee (Abate)

### 2.6.1 Method

We considered Bluetooth and Zigbee when considering communications protocols. Zigbee is a communications protocol designed for mesh networks. This means that every vessel in a swarm of boats could both relay and send its own data to other boats with just a single Zigbee module on each. This is a distinct advantage over one-to-one protocols over Bluetooth. Zigbee also allows lower power consumption than Bluetooth.

Importantly, Zigbee also transmits data at lower frequencies than Bluetooth [Ray, 2015]. This means that Zigbee transmits less data per second, but it also ensures that Zigbee communications do not interact with water – a problem with Bluetooth's near-microwave frequencies (2.4 GHz [SIG]). This means that transmissions between zigbee modules will reflect off of the water, rather than be absorbed by it. The fact that more absorption occurs as frequency approaches the microwave range is demonstrated in the following graph from Jackson's textbook *Classical Electrodynamics* [Jackson, 1999]. At  $2.4 \times 10^9$  Hz, the absorption coefficient is slightly less than 1. At  $0.9 \times 10^9$  Hz (our Zigbee module – see following section), the absorption coefficient is slightly less than  $10^{-1}$ . That is, for a factor of 2.6 reduction in frequency, we achieve 10 times less signal absorption by water. This is demonstrated in figure 13.

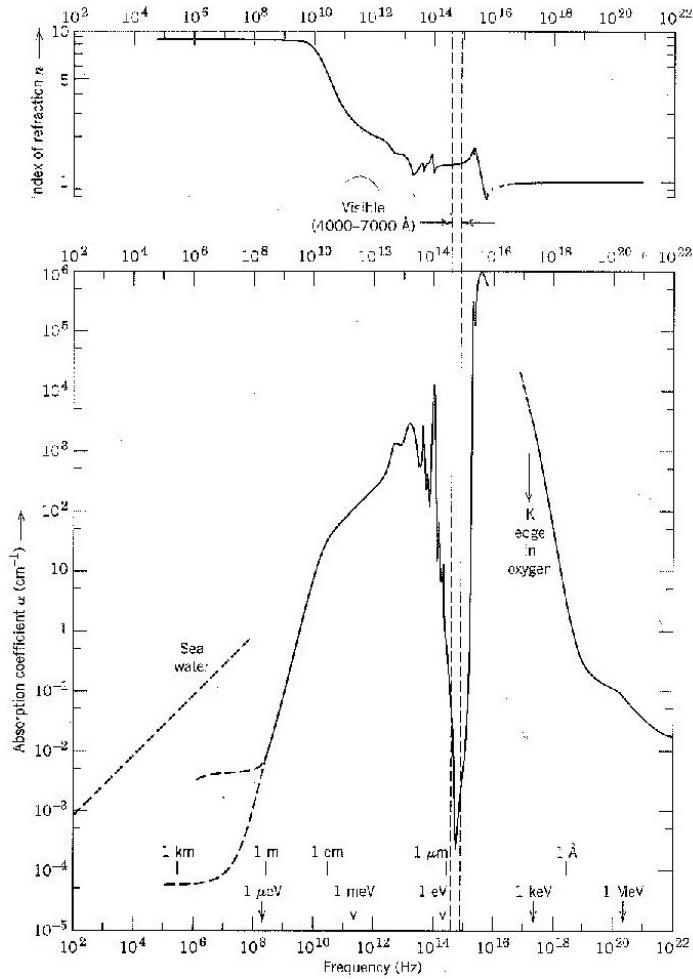


Figure 13: Absorption coefficient and index of refraction of water for various frequencies, courtesy of Jackson's *Classical Electrodynamics* [Jackson, 1999]

### 2.6.2 Results

We chose the Digi XBee-Pro 900HP module, which connects to the USB port of a Pi via the SparkFun XBee Explorer Dongle. The XBee-Pro communicates with the Explorer Dongle via SPI. The XBee-Pro operates in the 900 MHz range. The XBee-Pro has a 9-mile line-of-sight range, and 2000-ft range otherwise, transmitting data at 10 Kbps. As we do not anticipate having to transmit raw image data, this data transmission rate should fit the vessel's needs (coordinates of obstacles, direction vectors, etc.). A 2000-ft range is suitable for Lake Matoaka, and in larger bodies of water, some vessels could serve purely as relays to ensure a swarm does not bifurcate. Further, this module can connect to external antennas, giving a maximum possible range of 28 miles line-of-sight [Digi].

### 2.6.3 Future

To use the Zigbee protocol with its devices, Digi created the X-CTU software. This software abstracts away the need to code serial communication between computer and Explorer Dongle, and SPI communication between Explorer Dongle and XBee chip. This software is flexible, however – it allows modification of Baud Rate, Data Bits, Parity, Stop Bits, and Flow Control. The software has both a GUI/terminal for easy testing and a Python API. This API will ease integration of Zigbee communications into the project code base as future students apply the module.



## 2.7 Hull and Power (Hetzler)

### 2.7.1 Method

We require a watertight hull that displaces enough water to float the weight of all the components of the boat (likely between 2 and 3 kg). To accommodate our battery, VSD and steering servos we require the hull to have a flat bottom of roughly 150mm square.

Originally, we had a few options for manufacture. First we examined pre-made hulls. Most flat bottom hulls designed for RC boats were too expensive because they were made to look like a real ship. This is not what we needed. We decided this method was not cost effective. We examined using other manufactured objects for a hull such as a baby bathtub or a drink cooler but these shapes were too bulky to be hydrodynamic or made from metal which is a material that can be difficult to cut holes in. There was also little room for customizing the shape of the hull. If we manufactured our own hull, we could design the hull to fit our needs,

A relatively easy way to manufacture objects is 3D printing. We made a 3D model that fit our design specifications but found it was too big for the 3D printers we have access to. We attempted to print the hull piece-wise. Although we would have been able to use acetone vapor to seal our print, making it waterproof, this printing method still presented two issues. First, the piece was warped such that it would not mate flush to other segments of the hull. This is likely due to an uneven temperature gradient in the 3D printer or the fact the test did not adhere fully to the print bed and peeled up as the print progressed. Secondly and similarly, the structural integrity of the print was inadequate. There were sections of the print where layers did not adhere to each other, there were sections where the infill was too weak. These imperfections could be solved by more supports or more infill but both of these solutions would make for longer prints. Our one section which was intended to be 1/7 of the full hull took 36 hours to print. Even if the prints ran perfectly each time, to manufacture a boat by 3D printer would take over a week of continuous printing. This estimate does not include the requisite mending, assembly and waterproofing that would be required for each boat. We concluded that we should not use 3D printing to manufacture the hull because it was wasteful and time consuming.

We next considered a fiberglass hull. We planned to construct a positive mold (buck) from slices of wood and shape layers of wet fiberglass around the mold. When the fiberglass is dry, we will cut holes for the VSD and drop our components into place, securing them with screws, epoxy or hot glue.

To prepare this, we made a 3D drawing of the buck from our 3D drawing of the hull. The buck would have been made from 10 slices cut from Medium Density Fiberboard (MDF) using a CNC router. Toolpaths were drawn using Fusion 360 so the slices could be cut out efficiently as possible. Figure 14 shows some 2D representations of the buck and its dimensions. The horizontal lines show where the slices divide the buck. In practice, each slice would be individually cut from MDF and glued together. The buck would then be smoothed and sanded in preparation for fiber glassing.

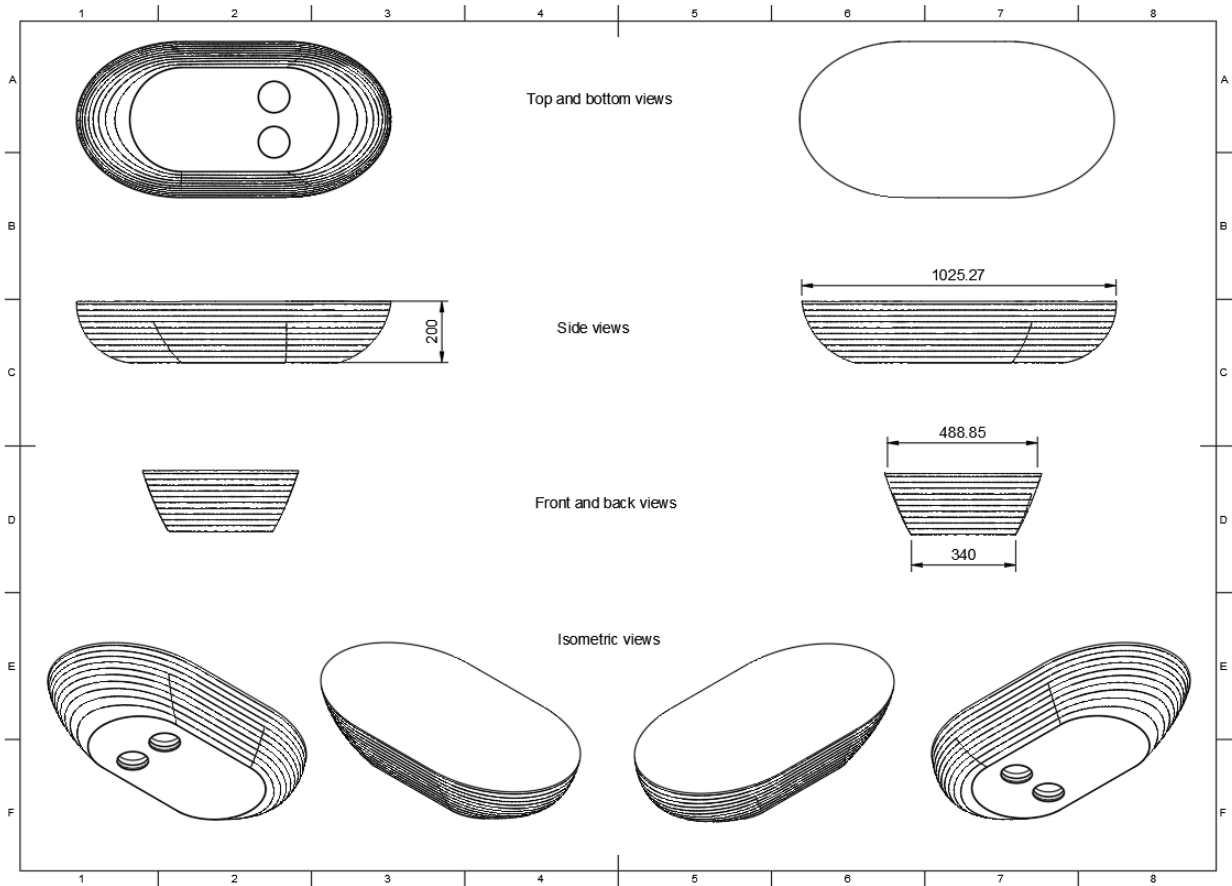


Figure 14: The buck that we planned to assemble piece wise from slices cut on a CNC router. Note this design has two holes for VSDs. Our original design specification for the hull was for two VSDs. When we moved from this design we abandoned the second VSD for simplicity. This is further explained in the results section.

After some preparation, it was clear that this method would be too time intensive to make practical sense. The process of making the buck alone could have taken up to 10 hours. The fiber-glassing would then add time on top of that. It was soon clear this was time that we did not have. In order to prioritize the function of this boat over its cosmetic appearance, we opted for a less intense construction process detailed in the results portion of this section.

### 2.7.2 Results

We chose to instead use a small bin that had been holding our project thusfar. We cut hole in the bottom of the bin large enough for the VSD and caulked it into place. We hot glued the servos into place and a board above the drive to hold our pi and GPS. We put a lid on the boat to keep the inside dry as possible and 3D printed a mast which holds the magnetometer.

A drag fin on the stern that aligns the boat with the VSD's thrust vector. This drag fin allows us to bypass the added complication that would come with programming a "turning" protocol. In order to fit our unique motor, the boat has a flat bottom that is wide enough to accommodate a central VSD and one servo (roughly 150mm). The hull is the exact tub used for the test bed in figure 6.

Power is required the E.S.C. which drives the VSD. The Servo hat which drives the steering servos and the pi which supplies power for the NXP-9DOF (magnetometer) and the GPS. The servo hat and the pi both run on a 5 volt supply. We could use the same source for both of these components but the pi's power adapter is a standardized connector and the hat takes a +5V and a ground wire. For simplicity, we use separate 5V supplies for these two pieces. For the ESC, we use a rechargeable 11.1V, 10 Amp-hour battery

connected directly to the E.S.C.'s leads. Figure 15 shows our boat floating and figure 16 shows the assembly of BOATOS in its full glory!

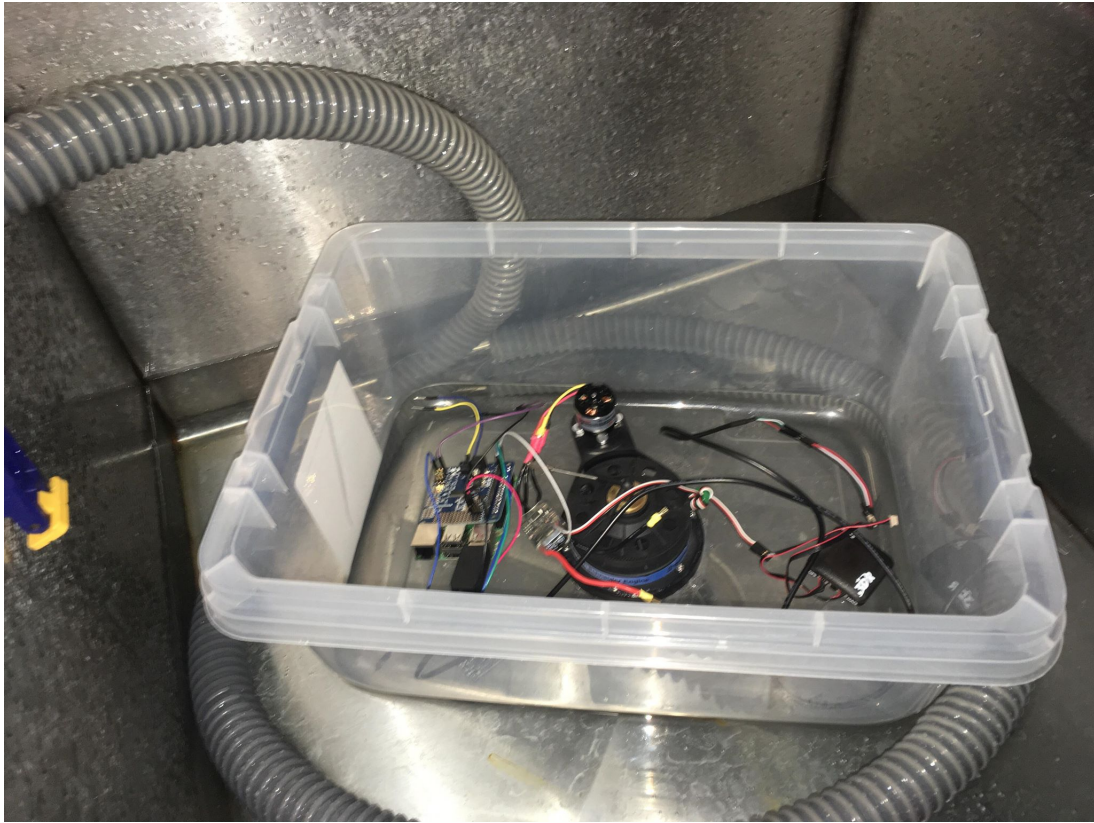


Figure 15: It floats!

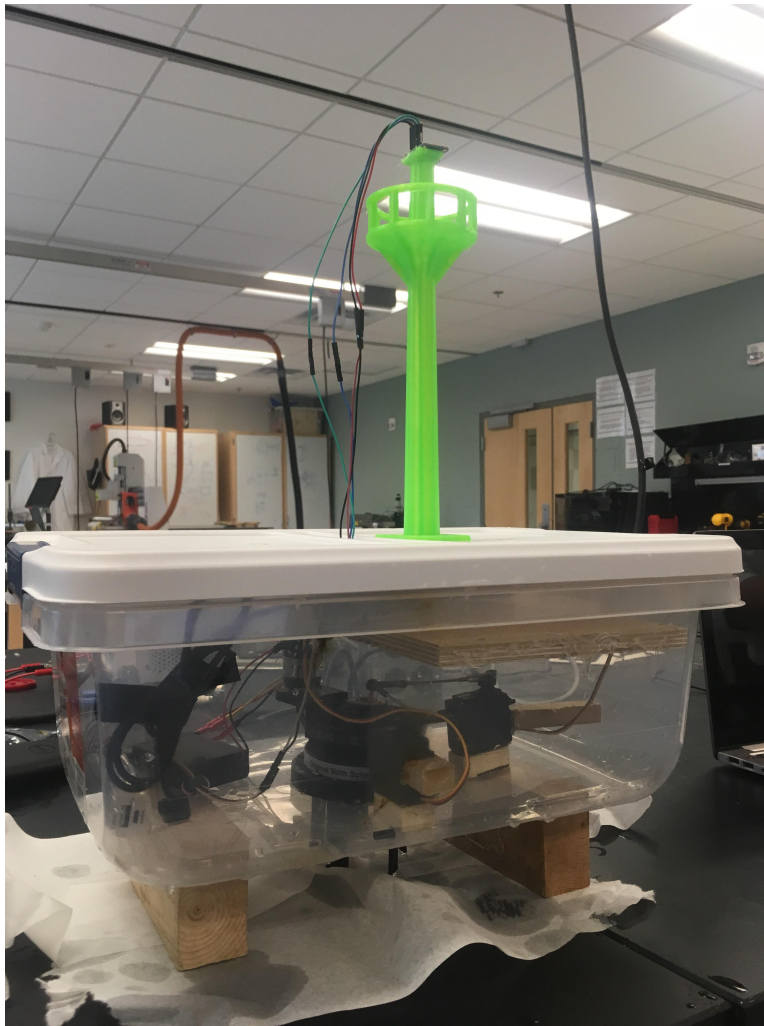


Figure 16: Our ship worthy of the calm Matoka Lake

### 2.7.3 Future

Future students may find as we did that using a plastic bin as a hull is the simplest and most time efficient method for hull construction. If they decide to use two servos however, they may wish to use our 3D designs or make their own. Future students may also want to divide the power from the large 11.1V battery into a bus that also outputs 5V. This would allow them to reduce the weight of the boat (by reducing the number of batteries) and eliminate the kludge-y wiring of 3 separate power supplies. If this bus were controlled by a regulator connected to the pi, in principle, the pi could control the power flow to the E.S.C., allowing the pi to turn off the motor.

## 3 Code Architecture (Abate)

This section describes the software architecture integrating data collection with navigation tasks.

### 3.1 Overview

Successful autonomous navigation can be separated into two tasks: data gathering and motor control. Accordingly, the Python files responsible for these tasks are separated into "observing" and "control" folders. The directory containing these two folders has a single Python file: "conductor.py". This file orchestrates the two tasks and how frequently they each run. This architecture is shown below in Figure 17.

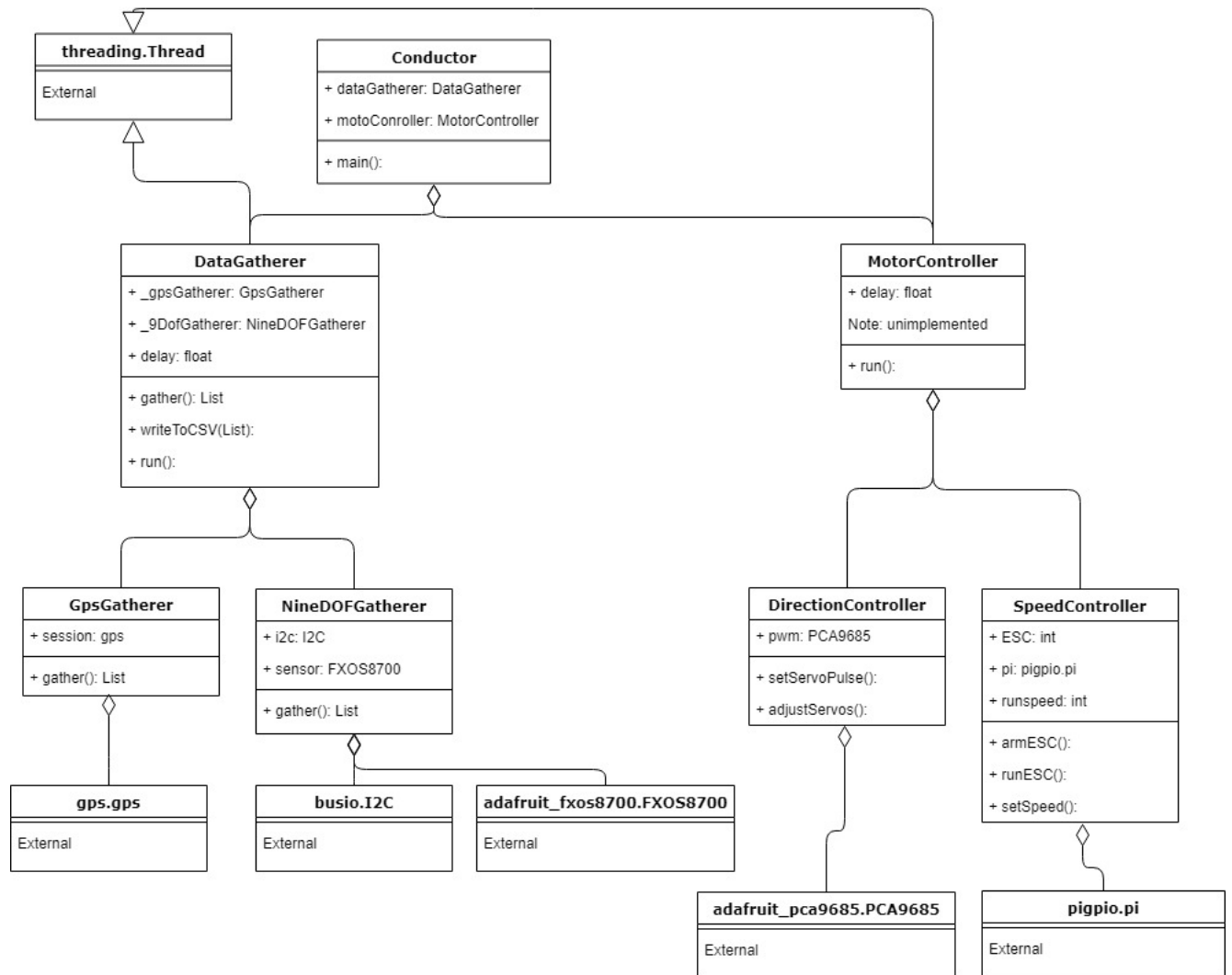


Figure 17: Unified Modeling Language (UML) class diagram of the boat-os software architecture

### 3.2 Multithreading

The conductor.py file orchestrates data gathering and motor control by running them in separate threads in what is referred to as the "Master-Slave Architecture". We multithread the two tasks using Python's native library "threading". Note that conductor.py does *not* depend on this library. An example from conductor.py is necessary to illustrate this:

```
dataGatherer = dg.DataGatherer(0.1)
motorControl = mc.MotorController(0.1)
```

```
dataGatherer.start()
motorControl.start()
```

"DataGatherer" is a class. It is contained in the data\_gathering.py file, which is here abbreviated dg, of the "observing" folder. Importantly, "DataGatherer" is a subclass of "threading.Thread" – that is, it inherits the methods of threading.Thread and can overwrite them. One such method is "run()". "DataGatherer" overwrites "run()" to include all data gathering activities, executed in each iteration of an infinite while loop. When, in conductor.py, "start()" is called on the object "dataGatherer" of class "DataGatherer", "run()" is executed, as well as other "threading.Thread" methods relevant to multithreading. The same argument holds for the "motorController" object, whose "MotorController" class is a subclass of "threadingThread".

In the infinite while loops in the "run()" methods of "DataGatherer" and "MotorController", each iteration ends with a call to "time.sleep(param)", where param is a floating-point number of seconds for execution to pause. In reality, execution may pause for longer, as a "time.sleep()" call in "DataGatherer.run()" cues the multithreading library to run another iteration in "MotorController.run()", and vice versa. In this way, conductor.py trades running time between "dataGatherer" and "motorControl". By adjusting the relative numbers of seconds slept in the two while loops, we can adjust which object's "run()" method is resumed more often.

In this implementation, "DataGatherer" writes, and "MotorController" accesses data. Locking is, therefore, unnecessary. Instead, our use of threading is to coordinate and time module execution.

For detailed descriptions of how a module functions, see the comments in its file via our Github repository: <https://github.com/AdamAbate-6/boat-os>.

### 3.3 Version Control

We use Git as our version control infrastructure. The reason is that, other than its popularity, it is the system with which I am most familiar. To hold our code, we have created the Github repository boat-os, whose URL is given above.

## 4 Steps Towards Autonomy (Abate)

In this section, we describe the work of combining the system components into a vessel that can path-find and avoid obstacles.

### 4.1 Module Integration and Path-Finding

We are considering three approaches to path-finding, in the order in which we believe they should be tested. First, the well-known Proportional Integrative Derivative (PID) controller; second, a hydrodynamics-based approach; third, the newer field of reinforcement learning.

A PID controller is an intuitive approach to navigating between two locations, given external forces that make a straight line between the points intractable. Call the error  $e$  the difference between the target end location and your location. In a 2-dimensional space (latitude and longitude),  $\mathbf{e}$  is simply a vector going from starting to target location. The larger the magnitude of this error (distance to the target), the faster the boat wants to go along the vector  $\mathbf{e}$ . This is the proportional term. This, however, risks the boat overshooting the target. The derivative term, adjusts for this, dampening the vessel's response (speed) more as the error decreases more rapidly. Lastly, the integrative term ensures that the pressures of proportional term decrease and derivative term dampening as the vessel approaches the target do not prevent the vessel from reaching the target. This is because the integrative term increases as time goes on so long as the error is non-zero. These three terms can be expressed in the relationships of Equation 2 and Equation 3.

$$r_{lat}(t) = K_p e_{lat}(t) + K_i \int_0^t e_{lat}(\tau) d\tau + k_d \frac{de_{lat}(t)}{dt} \quad (2)$$

$$r_{lon}(t) = K_p e_{lon}(t) + K_i \int_0^t e_{lon}(\tau) d\tau + k_d \frac{de_{lon}(t)}{dt} \quad (3)$$

The outputs of the PID functions  $r_{lat}$  and  $r_{lon}$  are "responses". In our case, these are changes in velocity in the latitudinal and longitudinal directions. The coefficients  $k_p$ ,  $k_i$ , and  $k_d$  require tuning to ensure that 1) certain terms are more emphasized, and 2) the output is a reasonable change in velocity. Emphasizing certain terms over others may reduce the time it takes to reach a target. This tuning must be done in the environment or via simulation. For our case, Lake Matoaka testing is the simplest way to optimize these coefficients.

For the next method, we look at a paper deducing control algorithms from hydrodynamics [Gaiduk et al., 2015]. These algorithms maintain a designated angular change and velocity. To our knowledge, this has only been tested in simulation. The algorithms' grounding in classical mechanics, however, possibly makes them more reliable than the second option we are considering.

Reinforcement learning is a growing field at the intersection of control theory and machine learning. A robot trained via reinforcement learning has a "reward function" – in our case, possibly a linear combination of inverse time-to-target, inverse battery-consumption, and inverse number-of-collisions. The robot aims to maximize this reward function, and ideally learns to do so after thousands of simulations. The more components of the reward function, and the more dynamic the environment, however, the less likely it is for this training to converge. That is, the robot will not find even a local maximum of the reward function. Future students should look into previous attempts at reinforcement learning in autonomous surface vehicles, such as one by Sterne [Sterne, 2004].

As of the writing of this report, we have completed a purely north-seeking vessel. This has allowed us to ensure our code architecture's flexibility, preparing for later use of a PID controller.

## 4.2 Obstacle Avoidance

Our Autonomous Surface Vessels will not be very effective if they are oblivious to their surroundings. Since our ultimate goal is to create a swarm of A.S.V.s, we must consider their spatial awareness. We believe that the first, most trivial step of spatial awareness is stationary obstacle avoidance. To avoid an obstacle, our A.S.V. must first be able to understand what a stationary obstacle looks like. Since we have planned to equip our boats with cameras, this is a matter of interpreting the pictures we take. If the `findContours()` method is successful in finding obstacles and we are able to determine the distance between the boat and the obstacle, then we can add a piece of code to our navigation that draws a path away from the obstacle. Once the obstacle has passed, the boat should return to its original trajectory. This could work for one obstacle but when more than one obstacle is present or an obstacle is moving, avoiding them becomes more challenging. However since the path of the boats are relatively unconstrained, the A.S.V. has a lot more freedom to choose paths. It may be wise for swarm behavior to add short range infra-red distance sensors to determine the distance between A.S.V.s and obstacles or other A.S.V.s. [Erik F. Wilthil and Brekke, 2017]

Since obstacles also may exist underwater and our VSDs draw roughly 5 cm, we advise that future students look into additional underwater cameras. Similar object avoidance algorithms would likely apply to input from underwater cameras. More research is required for this.

## 5 Conclusion (Abate & Hetzler)

For this project we have worked to understand and operate the VSDs, the camera, the GPS and magnetometer modules. We have also created conductor code that operates both a datagatherer and motorcontroller loop. The motorcontroller loop takes as input a desired heading and the boat's current heading, translating them into a directional expression for the servos while spinning the VSD. The datagatherer operates and records data from the vessel's GPS, 9-DOF, and eventually the camera module. Using this structure, we have first made a boat that seeks north. Within our framework, future students may implement code that will allow navigation to specific GPS coordinates and stop on arrival. Potential future projects will investigate the construction of an operating system that can be uploaded to the Pi so that more processing power can be dedicated to the processing of graphics (and other data). Additionally, future students may examine obstacle identification and avoidance. After stationary obstacles can be avoided, it may be worthwhile to investigate moving obstacles or obstacles just below the surface. When the boats can navigate without collision, future groups should make more prototypes in order to build the software mechanisms needed to examine swarm behavior on a surface.

# Appendices

## A Camera Code

```
# Author: Professor William Cooke  
# Adapted on 9/25/18 by Adam Abate  
# License: Public Domain
```

```

from picamera import PiCamera
from time import sleep

camera = PiCamera()

#camera.start_preview()
#sleep(10)
#camera.stop_preview()

#camera.start_preview()
# Needs 5 seconds to set levels
#sleep(5)

for i in range(5):

    camera.capture('/home/pi/Desktop/image%s.jpg' % i)
    sleep(2)

#camera.stop_preview()

```

Initializing an object of type `PiCamera` allows the developer to take a picture simply by calling that object's `capture()` method. This method's argument is the directory and file name of the resulting image. Run as is, this code will take five pictures, one for each iteration of the for loop. The file name will be `image0.jpg`, `image1.jpg`, `image2.jpg`, `image3.jpg`, or `image4.jpg` – the number is the for loop iteration. Between each capture, the Pi "sleeps" for two seconds, delaying execution of the next iteration by two seconds. Such delays allow the camera to adjust for resolution and brightness. Above the for loop, the commented code would allow the user to preview their image. That is, a call to `start_preview()` displays a window on the Pi's video output (does not require the GUI of an operating system) showing what a call to `capture()` would return at that instant. Likewise, calling `stop_preview()` closes this window. As we don't aim for humans to pilot the boat, such previews are not necessary.

## References

- Adafruit. Adafruit ultimate gps breakout - 66 channel w/10 hz updates - version 3, a. URL <https://www.adafruit.com/product/746>.
- Adafruit. Adafruit precision nxp 9-dof breakout board - fxs8700 + fxs21002, b. URL <https://www.adafruit.com/product/3463>.
- AGTx. Driving an esc/brushless-motor using raspberry pi. URL <https://www.instructables.com/id/Driving-an-ESCBushless-Motor-Using-Raspberry-Pi/>.
- MIPi Alliance. Mipi camera serial interface 2 (mipi csi-2). URL [https://www.overleaf.com/learn/latex/Bibliography\\_management\\_with\\_natbib](https://www.overleaf.com/learn/latex/Bibliography_management_with_natbib).
- Arducam. Raspberry pi multi camera adapter module user guide. 2015.
- Tony DiCola. Adafruit 16-channel pwm/servo hat bonnet for raspberry pi. URL <https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi?view=all#using-the-python-library>.
- Digi. Digi xbee-pro® 900hp. URL <https://www.digi.com/products/xbee-rf-solutions/sub-1-ghz-modules/xbee-pro-900hp#specifications>.
- Andreas L. Flåten Erik F. Wilthil and Edmund F. Brekke. *Sensing and Control for Autonomous Vehicles*. Springer, Cham, Switzerland, 2017. ISBN 978-3-319-55371-9.
- Anatoliy Gaiduk, Boris Gurenko, Elena Plaksienko, Igor Shapovalov, and Maksim Beresnev. Development of Algorithms for Control of Motor Boat as Multidimensional Nonlinear Object. *Matec Web of Conferences*, 34(04005):1–4, 2015. doi: <http://dx.doi.org/10.1051/matecconf/20153404005>.



John D. Jackson. *Classical Electrodynamics*. Wiley, New York, 1999. ISBN 047130932X.

Maker. Accurate gps positioning with the raspberry pi computer and rtklib - gps base station configuration. 2016. URL <https://custom-build-robots.com/raspberry-pi-elektronik/praezise-gps-positionierung-mit-dem-raspberry-pi-computer-und-rtklib-konfiguration-gps-basisstation/7956>.

Picamera. Camera hardware. URL <https://picamera.readthedocs.io/en/release-1.13/fov.html>.

RACELOGIC. How does dgps (differential gps) work? 2018. URL [https://racelogic.support/01VBOX\\_Automotive/01General\\_Information/Knowledge\\_Base/How\\_does\\_DGPS\\_\(Differential\\_GPS\)\\_work%3F](https://racelogic.support/01VBOX_Automotive/01General_Information/Knowledge_Base/How_does_DGPS_(Differential_GPS)_work%3F).

Brian Ray. Zigbee vs. bluetooth: A use case with range calculations. 2015. URL <https://www.instructables.com/id/Driving-an-ESCBushless-Motor-Using-Raspberry-Pi/>.

Bluetooth SIG. Radio versions. URL <https://www.bluetooth.com/bluetooth-technology/radio-versions>.

Philip Jonathan Sterne. Reinforcement Learning. Master's thesis, University of Edinburgh, School of Informatics, 2004.

Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. 2006.