Optical Coordination of Surface and Subsurface Autonomous Vehicles

by

Cody Charles White

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degrees of

Master of Science in Naval Architecture and Marine Engineering

and

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Signature redacted

Author ....

Department of Mechanical Engineering
May 15, 2019

Signature redacted

Certified by ....

Dick K. P. Yue
Philip J. Solondz Professor of Engineering
Thesis Supervisor

Signature redacted

Accepted by ....

Nicolas Hadjiconstantinou
Chairman, Department Committee on Graduate Theses

Optical Coordination of Surface and Subsurface Autonomous Vehicles

by

Cody Charles White

Submitted to the Department of Mechanical Engineering
on May 15, 2019, in partial fulfillment of the
requirements for the degrees of
Master of Science in Naval Architecture and Marine Engineering
and
Master of Science in Mechanical Engineering

Abstract

Much work has been done in the development of autonomous systems for use at sea. They provide a useful way of carrying out a variety of tasks, from exploration to data collection. The vast expanse of the ocean has naturally led to interest in networks of independent, autonomous vehicles that can act in a coordinated manner, or swarm.

Current swarm technology has focused on the surface of the water. However, it is desirable to be able to perform measurements at varying depths. In order to expand swarm capabilities, this work investigates using low-cost cameras and light communication to provide navigation for a subsurface sensing vehicle, by allowing it to follow a surface vehicle. This would allow for data collection at various depths, expanding the capability of the existing swarm, while maintaining the coordination of the swarm on the water's surface. This leverages the benefits of wireless communication and GPS.

This work focused on developing an understanding of the potential capabilities that could be expected as a result of utilizing low-cost cameras for guidance. Testing was performed to examine tracking algorithm performance in both relatively clear testing tank water as well as turbid pond water. Using a low-cost, 5.0 Megapixel USB camera, testing showed reliable tracking performance out to a maximum range of 40 feet in clear water.

Additionally, this work focused on creating an underwater platform to facilitate testing of various tracking algorithms. The platform utilizes a vertically facing USB camera, incorporated into the guidance and propulsion system of an autonomous surface vehicle. This platform allows further testing of tracking schemes as well as environmental effects, such as wave and target motion, water clarity, and ambient light.

Thesis Supervisor: Dick K. P. Yue
Title: Philip J. Solondz Professor of Engineering

Acknowledgments

I would like to thank my future wife, Brianna, who supported me during my MIT experience. Also, I would like to thank my Advisor, Professor Yue, for his support and encouragement. Additionally, I would like to thank Grgur Tokic, PhD, for his guidance throughout, along with Andrew Freeman for his help.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

Water covers over 70% of the earth. It impacts virtually every aspect of our lives. The world's water is directly tied to weather, climate, and atmospheric conditions. It is a vital source for a number of resources, including food, oil, and minerals. The marine environment is also being increasingly used as a source of renewable energy, from both the wind and waves. The economies of the world also ride the ocean currents. According the the International Chamber of Shipping, approximately 90% of world trade is carried over the world's waterways. This traffic is rapidly increasing, with research showing a fourfold increase between 1992 and 2012[Tournadre, 2014]. This growing reliance on the marine environment is driving a plethora of research as people strive to understand the dynamics of this complex environment, as well as our impact on it.

1.1 Exploring the Marine Environment

Oceanographers, environmental scientists, biologists, as well as companies from a number of industries, are interested studying the various marine environments. Their goals vary as much as the ocean itself. From studying aquatic life in coastal regions to exploring the deep sea for natural resources, the search for knowledge of the marine environment has led to a number of developments to facilitate data collection of various types. Both the variety of data and the large spatial scale of these quantities of interest provides a number of unique challenges.

1.1.1 Autonomy

The vast areas of interest require expeditious means of collecting data from the surface of the ocean. Some measurements are conveniently measured using satellites or airplanes. For instance, sea surface temperature measurements from using thermal infrared sensors or surface current measurements using high-frequency radar [Leonard et al., 2007]. Other types of measurements must be taken locally. In situ measurements require placing sensors in the water in fixed locations, letting them be carried by the current, or providing some type of propulsion to guide them as desired. Ships can be used to carry or tow sensors to the desired location, but having vehicles or sensors that can take themselves where needed provides many benefits.

Unmanned surface vehicles have been the focus of large amounts of research in the last few years [Manley, 2008]. Like progress in many fields, advances stem from the development of smaller, more affordable, computation and sensing technology. These technological advances are continually bringing more capability to various surface and subsurface platforms. Remote monitoring by an autonomous surface vehicle (ASV) or autonomous underwater vehicle (AUV), provides an effective means of data collection for certain scenarios; however, the flexibility and efficiency can be lacking.

Since the marine environment is constantly changing, variables of interest such as temperature, pH, salinity, and oxygen concentrations also can change rapidly, but the data is spread over a large area. As a result, it is often desired to simultaneously take measurements in multiple locations, which requires multiple platforms. This naturally leads to the desire for networks of distributed capability.

1.1.2 Swarming Networks

Having a multiple robot system (MRS) operating in a swarm with decentralized control provides many benefits. Individual robots operate based on local observations of the environment and coordination with neighboring robots. Other benefits include network robustness and fault tolerance, flexibility to perform various tasks in different environments, and ability to scale the swarm to the task at hand [Duarte et al., 2016]. Zoss developed a small autonomous buoy to designed to be low cost, portable, and highly maneuverable platform able to collect, share, and process environmental data [Zoss, 2016].

This network provides a good example of an attritable system. Attritability describes somehting that is relatively low cost, such that its loss can be tolerated [Colombi et al., 2017]. Each buoy is relatively low-cost, so the loss of one presents minimal impact. Additionally, the loss of a single unit doesn't disrupt the function of the swarm as a whole. This work culminated in a distributed multi-robot system referred to as "Bunch of Buoys" or BoB for short. The network exhibited real world swarm operations with up to 50 units. Figure 1-1 shows the system of buoys operating in a marine environment and stacked for transportation.

Currently, the capabilities of the BoB is limited to the surface of the water; however, the ability to collect data a various depths is desirable and would greatly expand the utility of the system. There are a variety of ways that data could be collected at different depths, including:

- Towed sensors that can be lowered to different depths.
- Expendable sensors that are dropped and report data as they descend.
- Subsurface vehicles or vehicles that can operate both on and below the surface.

1.2 Thesis Overview

This thesis explores a method of coordinating surface and underwater elements to develop an attritable ASV/AUV pair to expand the capabilities of the swarm network, which is based on wireless optical tracking. This work utilizes a low-cost USB camera to provide a navigational inputs to a subsurface vehicle, allowing it to follow a surface element of the BoB. Tracking is done using open source computer vision techniques contained in the OpenCV library.

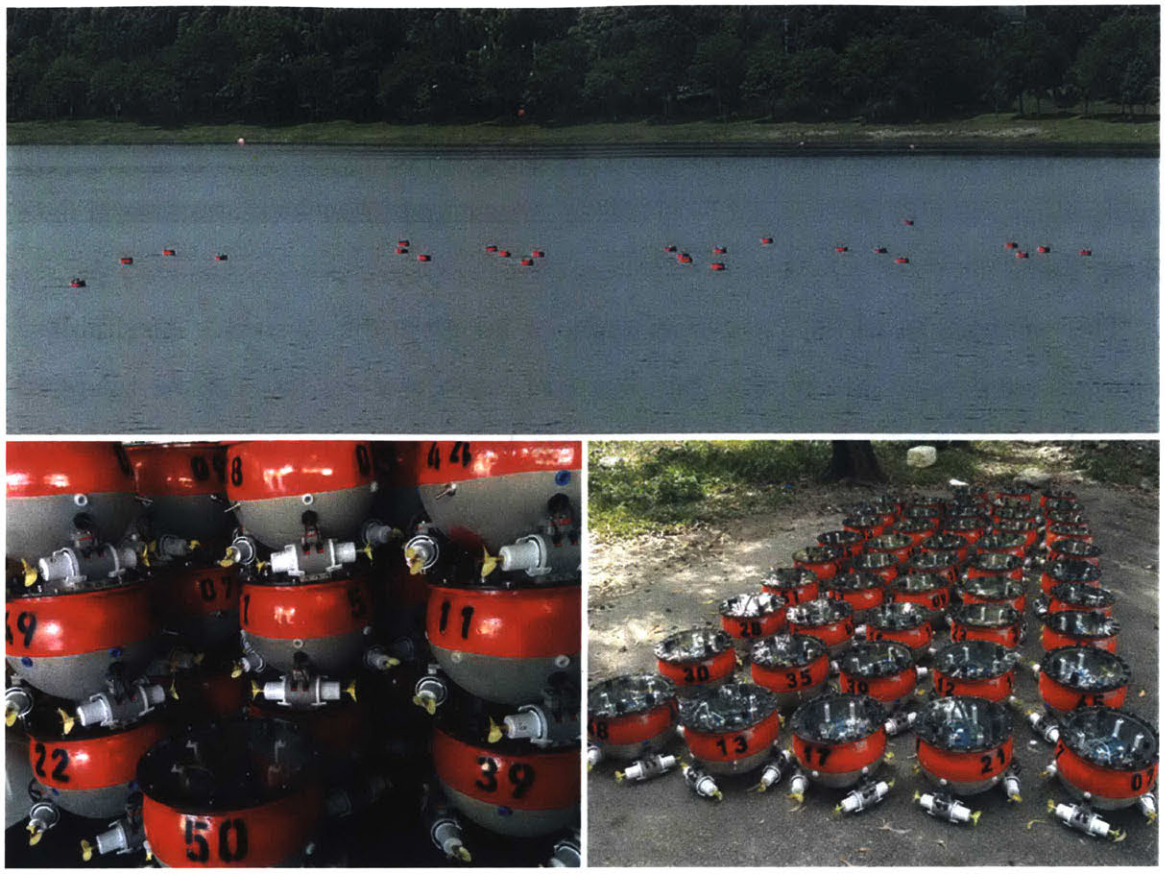


Figure 1-1: A small fleet of autonomous surface vehicles. Top panel: 25 buoys of a BoB system collectively operating at Bedok Reservoir, Singapore. Bottom left: Buoys stacked up during transportation to the field site. Bottom right: 48 buoys lined up before deployment [Zoss et al., 2018]

This provides a way of collecting data at additional depths, while keeping the overall swarm network on the surface. This facilitates positioning and communication between the different nodes of the network by maintaining the ability to use GPS and WIFI. Overall, it is consistent with the low cost, attritable concept of the swarm.

1.3 Thesis Contributions

This thesis focuses on the work done to incorporate this optical navigation into a subsurface vehicle to provide a platform for testing. Chapter 2 discusses background of the navigation of unmanned vehicles, as well as a brief discussion of the optical environment the a marine vehicle would experience. Chapter 3 provides a background on digital images and the manipulation of them in a computer vision environment. Also, a discussion is provided on the two algorithms that were selected as a starting point for testing. Chapter 4 shows the results obtained from testing done to obtain a preliminary understanding of the range that could be provided by an optical navigation system composed of low cost components. Chapter 5 goes into developing the platform for testing and Chapter 6 discusses the results gathered. Finally, Chapter 7 provides a summary of the results, as well as areas for further work.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Navigation of Unmanned Vehicles

Providing practical endurance for unmanned vehicles has long been a challenge. Significant research and development in renewable energy such as wind, solar, and even wave energy for unmanned and autonomous vessels is being carried out [Manley, 2008]. Additionally, improvements in battery capacity and fuel cells are allowing longer missions to be undertaken. Besides having sufficient energy to stay on station for a prolonged period of time, these vehicles must also have the ability to know where that station is in relation to themselves. These longer missions necessitate more advanced navigation systems to provide accurate vehicle control and minimize positional error that compounds as endurance increases [Stutters et al., 2008]. For instance, Wave Glider prototypes have demonstrated extended deployments of 142 days and over 2500 nautical miles [Manley, 2008]. Additionally, the job the vehicle sets out to perform has a lot to do with the types and quality of systems that are needed.type of mission generally dictates the requirements for navigational accuracy. As in most systems, there exists a complex relationship between requirements, system capabilities, size, endurance, and reliability, all of which have an associated cost.

2.1 Navigational Methods

There are several approaches to navigation depending on the what type of vehicle it will be used on and what type of mission will be conducted. There is generally not one solution that provides satisfactory results. This generally leads to typical navigation systems being made up of compilation of several of the following navigational approaches to provide for a robust picture of the environment.

- Dead Reckoning
- Acoustic Navigation
- Geophysical Navigation
- Satellite Navigation
- Visual Navigation

2.1.1 Visual Navigation

Visual navigation is used in a variety of applications. Autonomous vehicles are using it on land, in the air, and at sea. Aerial drones can use in for following a path and obstacle avoidance. Typical underwater applications include mapping seabeds and docking. These applications environments are generally very uniform in appearance and have little ambient light.

Utilizing visual navigation to direct an underwater vehicle that is focused on a surface target provides an interesting challenge. A variety of factors can complicate the problem, such as surface wave motion, significant ambient light from the sun, shadows, and floating objects like boats. One of the strongest factors that affect the ability of this type of system to function is the visual clarity of the water.

2.1.2 Transmission of Light Under Water

Anybody that has looked into different bodies of water knows that there is a wide variation in the clarity. From the crystal clear waters of the Caribbean to the Muddy Charles, the distance that you can see can range from a few inches to several meters. The optical properties of water vary widely depending on the where they are located, the composition, and even the time of day. In order to see what would impact the ability of a camera to see something, I first wanted to investigate what impacts the transmission of light in water. Not surprisingly, there are a large number of optical properties that impact the interaction of light to one extent or another.

Even at the same location, the optical properties of the water column will not be uniform. Over several miles of depth, pressure, temperature, and a number of other things will likely change. Because of these different properties the ocean is broken down into a number of different zones, which are shown in Figure 2-1. The epipelogic layer is the uppermost and corresponds to the photic zone. The photic zone receives sufficient sunlight to support photosynthesis. The bottom of this zone is typically defined by the depth at which the surface irradiance has been reduced to 1% [Jerlov, 1976]. Since this water experiences enough light for photosynthesis, there is a large amount of organic matter which both scatters and absorbs the light and plays an important role in the attenuation.

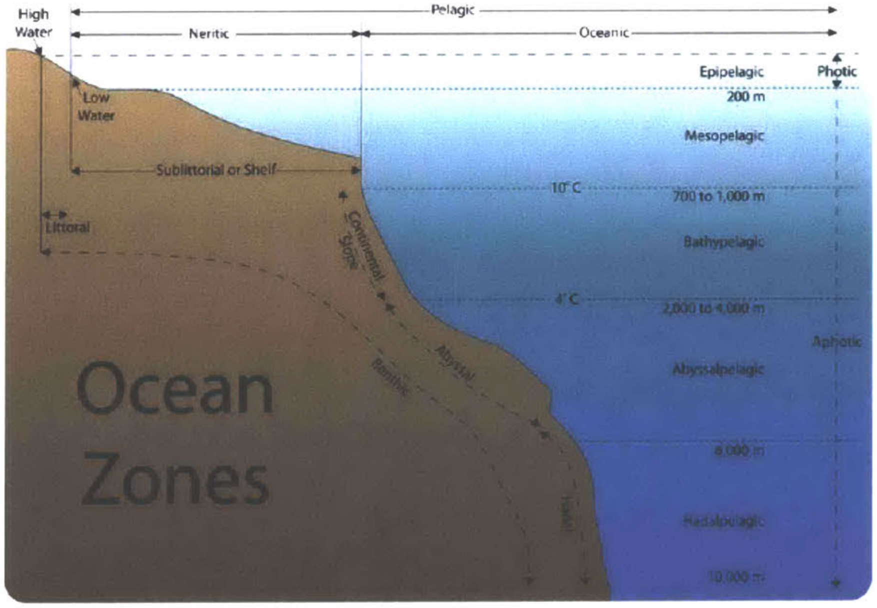


Figure 2-1: The various depth zones that the ocean is broken down into. [Allen Institute for Artificial Intelligence,]

In addition to the previously mentioned categories, Jerlov developed an optical classification for various waters based on the amount of transmittance they exhibit. This classification has three types, I, II, and III for ocean waters and an additional nine types, 1 through 9, for coastal waters [Jerlov, 1976]. This classification provides a convenient way of estimating various optical parameters based on the region of interest. Figure 2-2 shows the Jerlov classifications for various regions around the world [Jerlov, 1976].

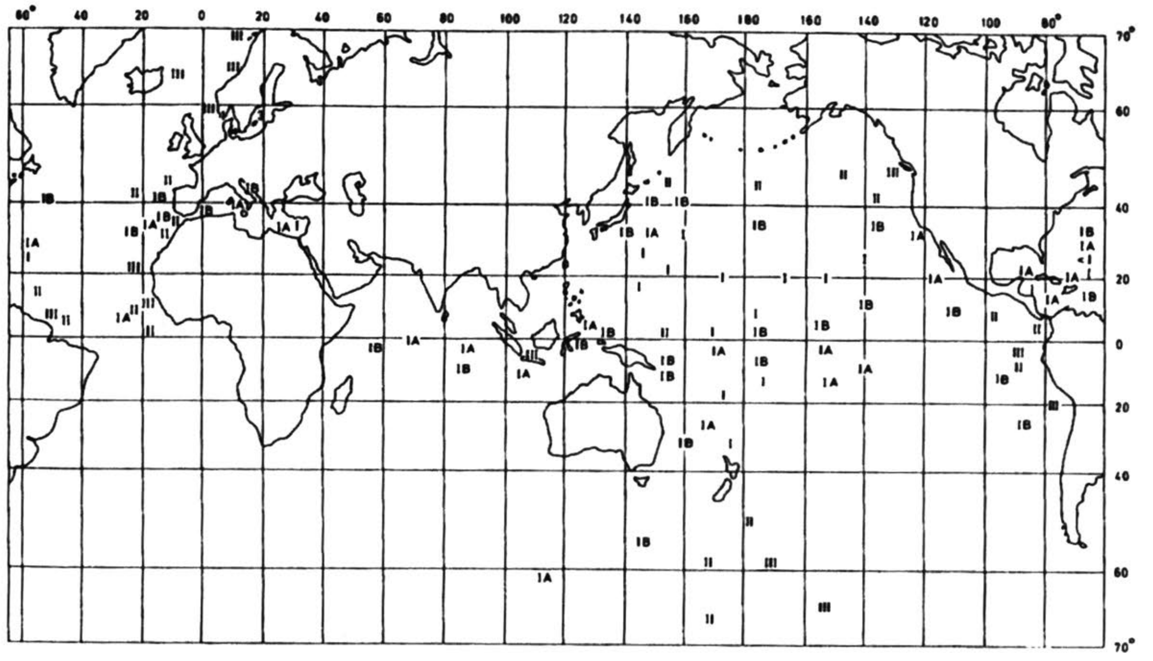


Figure 2-2: Regional distribution of optical water types.

2.2 Light Attenuation

As light is transmitted underwater, the intensity of the beam decreases via two main mechanisms, scattering and absorption [Jerlov, 1976]. This change in the intensity is known attenuation and the different Jerlov types can be used to estimate it.

The attenuation process is typically modeled using Beer's Law, shown in Equation 2.1. Here, r is the propagation distance and c is the total attenuation coefficient (in units m^{-1}) [P Bukata et al., 1995]. N is the photon flux, but the relationship can also be used for the energy flux or radiant flux.

$$N(r) = N_0 e^{-cr} \tag{2.1}$$

The total attenuation coefficient can be broken into two components, the absorption coefficient a and the scattering coefficient b. These are related by Equation 2.2 and are all dependent on the wavelength of the light in question.

$$c(\lambda) = a(\lambda) + b(\lambda) \tag{2.2}$$

Table 2.1 shows the various color of light contained in the visible spectrum, along with some of the associated properties. Each of these frequencies is effected by the environment to varying degrees due to a variety of mechanisms. For instance, Figure 2-3 shows the attenuation of light in ocean water and fjord water. It can be seen that a shift if the most penetrating wavelength takes place due to the change in water conditions.

Color	**λ (nm)**	**ν (THz)**
Infrared	>1000	<300
Red	700	428
Orange	620	484
Yellow	580	517
Green	530	566
Cyan	500	600
Blue	470	638
Violet (visible)	420	714
Near ultraviolet	<300	1000

Table 2.1: Colors and corresponding wavelengths and frequencies for the visible spectrum.

Depending on the water type, generally blue or green light tends to reach the farthest. Based on this, I decided to use both colors for the target to be used for the tracking algorithm. Additionally, I wanted to evaluate red as well. While red doesn't have the penetrating power of the other two, it does have the advantage that there is typically not a lot of red colored objects out in the marine environment. This is beneficial as it provides a high level of contrast with the surroundings, which is important for a color based tracking algorithm.

As is evident in the number of water types and wavelengths present, the interaction of light and water is a complex phenomenon that can change not only spatially, but

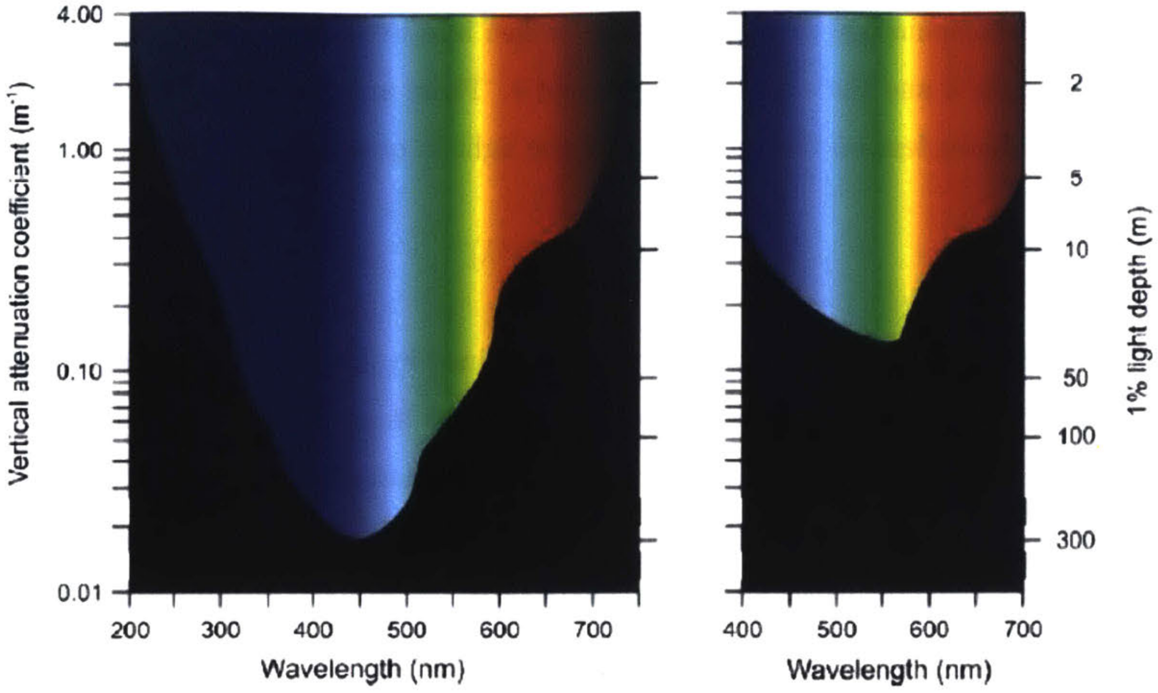


Figure 2-3: The left figure shows the absorption spectrum for ocean water, which appears blue because the deepest penetrating wavelength is approximately 450 nm. The right image show the spectrum for fjord water, which appears green due to the shift in the deepest penetrating wavelength to around 550 nm [Sakshaug and Slagstad, 1991].

temporally as well. This makes visual navigation a challenging problem. Adjusting to the navigational algorithms to environment can be cumbersome and computationally intensive. However, the ability to leverage computer vision for a subsurface vehicle could significantly improve the capabilities of the overall system. This thesis seeks to develop an understanding of these potential benefits and provide a platform for future work.

Chapter 3

Computer Vision

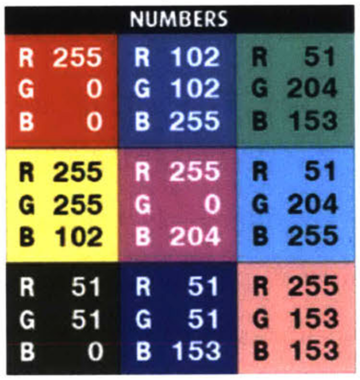
Large amounts of research have been done in the area of computer vision. It has a number of applications from facial recognition to managing traffic. There are a number of open source resources available. For this project, OpenCV (Open Source Computer Vision) was used since it provided a variety of tools. While written in C++, OpenCV also has Python libraries which allows for easy implementation of a number of common image processing operations. This project only uses a handful of the capabilites, which include neural networks, machine learning, and others. The ones that prove most useful for this task are object identification and motion tracking. Two tracking methods were selected for evaluation, Camshift and MOSSE.

3.1 Digital Images

Digital images are made up of an array of numbers, usually in three dimensions. For a typical digital image, two dimensions of the image represent how many pixels wide and how many pixels tall the image is, for instance 640x480. These coordinates then represent the location of a pixel. The remaining dimension is a vector containing a number of values. For a standard three channel image, the vector consists of three values, typical ranging from zero to 255. While there are a number of different formats, a common one is RGB. In an RGB image, each pixel location is defined by this the vector, resulting in a vector field:

$$\mathfrak{F}(R, G, B)$$

The numerical values represent the intensity of color, (red, green, or blue), that is present in the individual pixel. Figure 3-1 shows several colors and there numerical representations. This mathematical representation provides for a number of operations to be performed, such as establishing certain thresholds, inversion, addition, subtraction, and Gaussian filtering.


NUMBERS

R 255	R 102	R 51
G 0	G 102	G 204
B 0	B 255	B 153
R 255	R 255	R 51
G 255	G 0	G 204
B 102	B 204	B 255
R 51	R 51	R 255
G 51	G 51	G 153
B 0	B 153	B 153

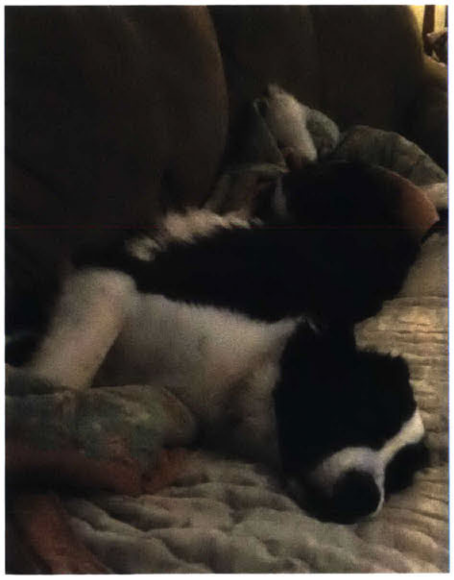
Figure 3-1: Example of RGB representation of several colors. [shutha.org]

Digital video is simply a string of digital images that are updated at sufficient rates as to appear continuous to the eye. As time advances, the digital images can change in a number of ways. The object under observation can change aspect, appearance, range, and position. Additionally, environment factors such as water clarity can change which leads to varying illumination. Tracking algorithms must be able to deal with these changes to some extent if the system is expected to be robust.

There are a number of ways of representing an object for use by an algorithm. The object can be represented using a collection of points, or a box that contains the object, as well as a contour of the outline. This area of interest is identified by key attributes, such as color, shape, or texture [Salhi and Yengui Jammoussi, 2012]. In order to provide accurate propulsion commands, the AUV must be able to accurately track the surface element in order to determine its relative position.

3.2 Camshift

The first algorithm selected is the Camshift algorithm. It is chosen because it focuses on tracking via color matching and utilizes relatively simple computations. Camshift stands for Continuously Adaptive Mean Shift, and is based on the Meanshift algorithm. The algorithm generates a target probability distribution function (PDF) using histogram back-projection, then iterates to climb the gradient of the PDF in order to find the mode [Allen et al., 2004]. Figure 3-2 shows an example of a histogram. Like normal histograms, color histograms simply show the color distribution of an image by indicating the number of pixels that have the value of each color bin. The algorithm takes an initial location in an image and determines the color



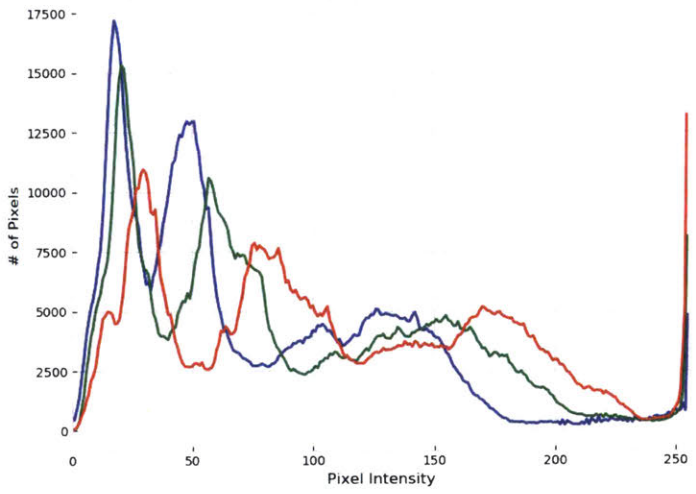


Figure 3-2: Left: Sample image. Right: Histogram of a digital image showing the distribution of all three colors (R,G,B)

histogram of this area. Given a histogram with m bins, the n pixel locations in the image are defined as $\{x_i\}_{i=1...n}$ and the histogram as $\{\hat{q}_u\}_{u=1...m}$. We also define a function $c : \Re^2 \rightarrow \{1...m\}$ that associates the histogram bin index $c(x_i^*)$ to the pixel at location x_i^* [Allen et al., 2004]. The unweighted histogram is computed as

$$\hat{q}_u = \sum_{i=1}^{n} \delta\left[c(x_i^*) - u\right] \tag{3.1}$$

Figure 3-3 shows an example of the probability image.

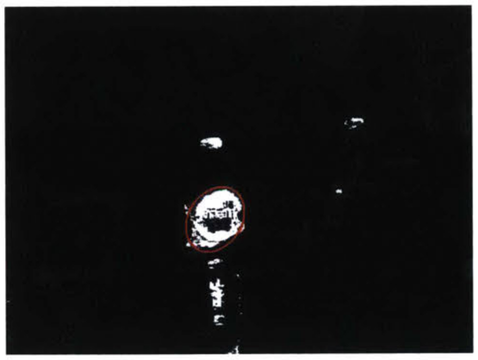


Figure 3-3: Sample of a probability distribution generated by histogram back-projection.

The location of the search window is moved by calculating the various moments and centroids associated with the probability image. These moment are analogous to geometric moments used in many applications. The center of the search window is then moved to the location of the centroid. The process is then iterated until it satisfies a maximum error value [Allen et al., 2004]. The following equations show the general idea. Here, $P(x, y)$ is the back-projected probability distribution at position x, y [Exner et al., 2010]. Equation 3.2 shows the zeroth moment, while Equation 3.3 shows the first moments.

$$M_{00} = \sum_x \sum_y P(x, y) \tag{3.2}$$

$$M_{10} = \sum_x \sum_y xP(x, y); \quad M_{01} = \sum_x \sum_y yP(x, y) \tag{3.3}$$

Using these, the location of the centroid can be calculated as shown in Equation 3.4.

$$x_c = \frac{M_{10}}{M_{00}}; \quad y_c = \frac{M_{01}}{M_{00}} \tag{3.4}$$

In order to update the search window, the Equation 3.5 is used to determine the second moments.

$$M_{20} = \sum_x \sum_y x^2 P(x,y); \quad M_{02} = \sum_x \sum_y y^2 P(x,y) \tag{3.5}$$

$$\mathrm{ar} = \frac{M_{20}/x_c^2}{M_{02}/y_c^2} \tag{3.6}$$

$$\mathrm{width} = 2M_{00} * \mathrm{ar}; \quad \mathrm{height} = 2M_{00}/\mathrm{ar} \tag{3.7}$$

These are then used to determine the aspect ration, ar,given in Equation 3.6. Then the new dimensions for the search window given in Equation 3.7 can be found [Exner et al., 2010].

3.3 Mosse

The second tracking algorithm selected for exploration was the Mosse algorithm. Mosse stands for Minimum Output Sum of Squared Error. It is a correlation based tracker that performs well with varying lighting, pose, scale, and even deformation [Bolme et al., 2010].

These types of trackers work by using an initial image of the object of interest to train a filter, which is then used to evaluate search windows in the next frame. The location of the window with the highest value of the correlation of the filter with the window is the new location of the target. The correlation is determined by first taking the fast fourier transform of both the image, f, and filter, h, $F = \mathscr{F}(f)$ and $H = \mathscr{F}(h)$. The transform into Fourier space allows the correlation to be evaluated by element wise multiplication of F with the complex conjugate of H [Bolme et al., 2010] as shown in the following equation. Here $\odot$ means element-wise multiplication and $*$ refers to the complex conjugate.

$$G = F \odot H^* \tag{3.8}$$

The filter can be determined using training images and desired output shapes, typically a 2D Gaussian function. Given that i images are used, Equation 3.9 shows the expression used to determine the filter [Bolme et al., 2010].

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*} \tag{3.9}$$

For instance, Figure 3-4 shows a frame from a video. The area inside the red square was chosen as the object of interest to be tracked. Figure 3-5 breaks down the process taken by the tracker. The first picture shows the area of interest. From this area, the filter is produced, which is shown in the middle picture. Finally, this filter is applied to the search windows throughout the image and one with the highest correlation is shown on the right. The filter determines a peak-to-sidelobe ratio (PSR) which provides a measure of the strength of the correlation. As time progresses, the filters are averaged with a time decay such that they basically slowly adapt over time.

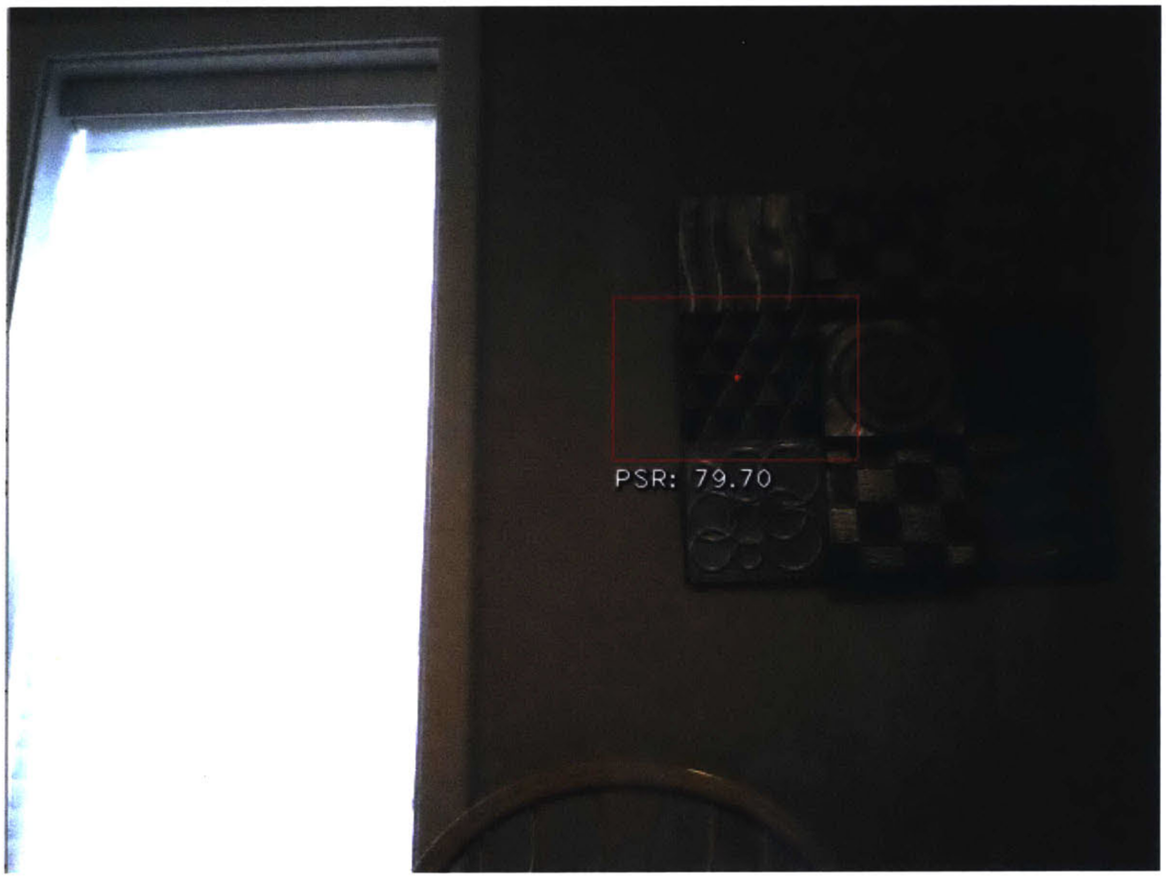


Figure 3-4: Example of a video frame used for initializing the Mosse tracker.

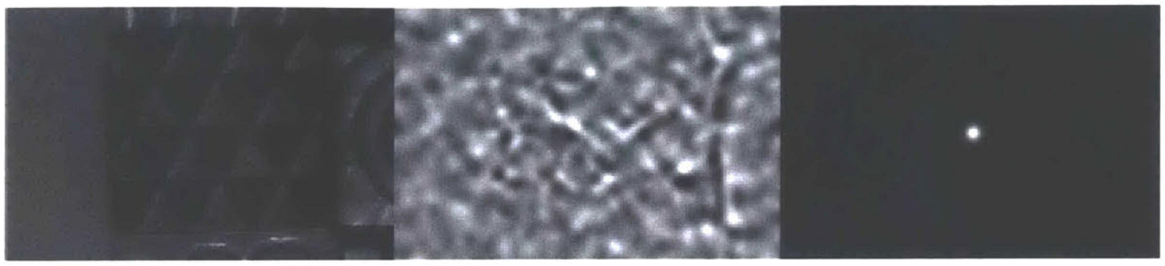


Figure 3-5: Left to Right: Search window input, filter, and correlation output for the frame shown in Figure 3-4.

THIS PAGE INTENTIONALLY LEFT BLANK

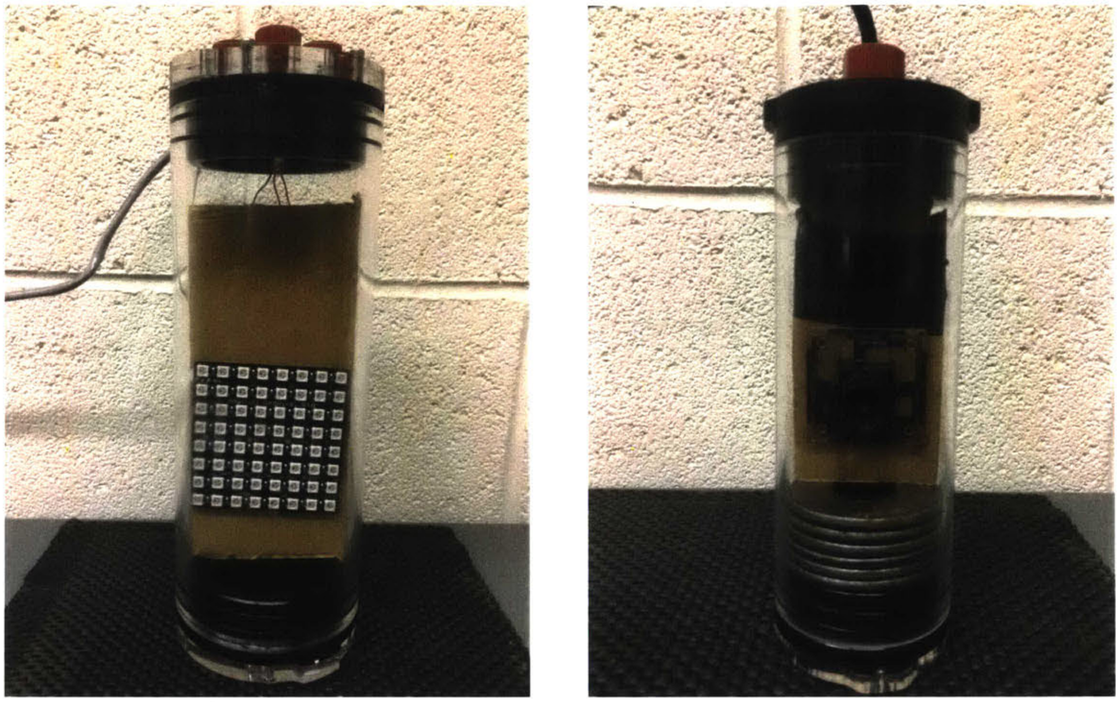
Chapter 4

Evaluating the Range of Tracking Algorithms

In order to get a better understanding of the utility the system could possibly provide, the maximum effective range was investigated. The MIT tow tank provided a good starting point for testing, as it provides a long span of water that is essentially as clear as can be expected. A pond on my family farm served as another, more real world, body of water for testing.

For the camera, an ELP 5.0 Megapixel USB camera (ELP-USB500W02M-L36) was used and placed in a waterproof tube. This provided an input via USB to a laptop running a Python script utilizing OpenCV. Appendix A contains the code that was used to run the Camshift algorithm.

An Adafruit 8x8 Neopixel LED matrix served as the target, which was also enclosed in a watertight tube. This provided a flexible light source, as the color and intensity of all 64 LEDs can be adjusted with only one signal wire. The Neopixel is controlled with a special library that generates specific pulse width modulation signals that were provided by an Arduino Mega 2560. Figure 4-1 shows the light and camera use. Appendix B contains the Arduino code that was used to control the color and intensity of the LED matrix.



(a) 8x8 Adafruit Neopixel (b) USB Camera

Figure 4-1: Equipment in waterproof canisters used to investigate range of effective tracking.

4.1 Tow Tank Testing

Testing in the tow tank revealed that with these low-cost components, the Camshift algorithm could track the 3"x3" Neopixel array out to a distance of over 40ft when using red as the target color in the clear tow tank water. The setup is shown in Figure 4-2. The depth was set at 24" for both the camera and the light. The light is mounted to a movable tray that can be rolled along the top of the tow tank, allowing the distance between the light and camera to be increased until the algorithm lost the track. Trials were done using both the Camshift and Mosse algorithm. This testing provided a good indication of the the maximum depth that could be expected. Additionally, environmental factors, such as color and reflections had an adverse effect on the performance of the system.

Testing was conducted with the light source producing red light, blue light, and green light. Each of these colors was tried at a 20%, 40%, 60%, 80%, and 100% of the output of the Neopixel LED matrix. In the relatively clear conditions of the tow

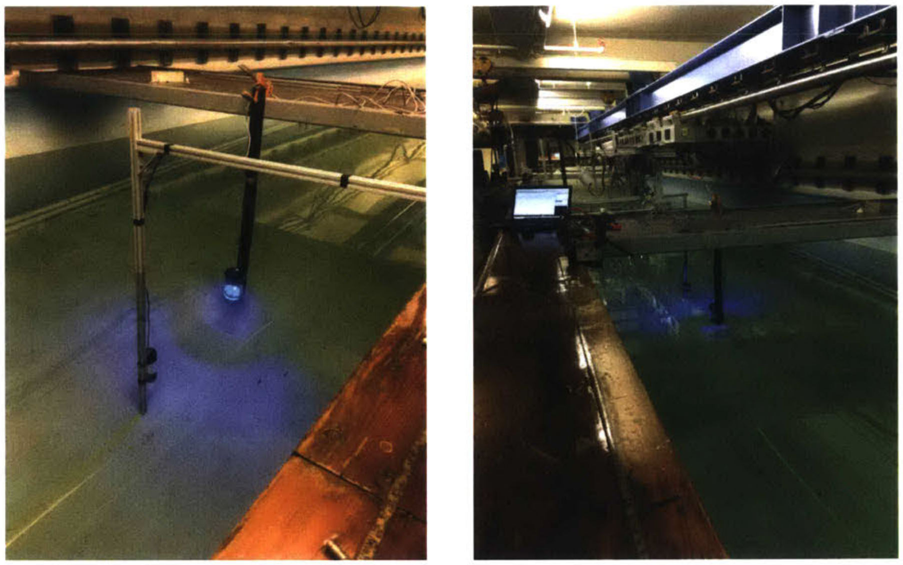


Figure 4-2: Equipment setup in the tow tank used to measure the effective range.

tank, the Mosse algorithm performed very well. It was insensitive to color and light intensity. The target was able to be tracked out to approximately 15 ft for all of the combinations tested. This can be seen in Figures 4-4, 4-5, and 4-3.

The performance of the Camshift algorithm was not as consistent as that of the Mosse. Figures 4-4 and 4-3 show significant variability in the distance achieved and overall poor results. Some of this is the result of the testing environment. When using green for the target, the water surrounding the target started to glow and produce a green halo that resulted in inaccurate tracking. There was significant green reflection on from the floor and walls as well which could be seen when looking at the probability images generated. Increasing the intensity of the light exacerbated the problem.

The blue results, shown in Figure 4-4, are better than the green, but still contained high variability. The orientation of the experiment resulted in surface reflections that hindered tracking and limited the range to approximately 30 ft. Like the green light, higher light intensity seemed to reduce performance. These effects would likely be less significant in a true marine environment with the camera oriented vertically and no walls to provide reflection.

The results using red light are shown in Figure 4-5. A red target exhibited the most consistent performance and was insensitive to the light intensity in the tow

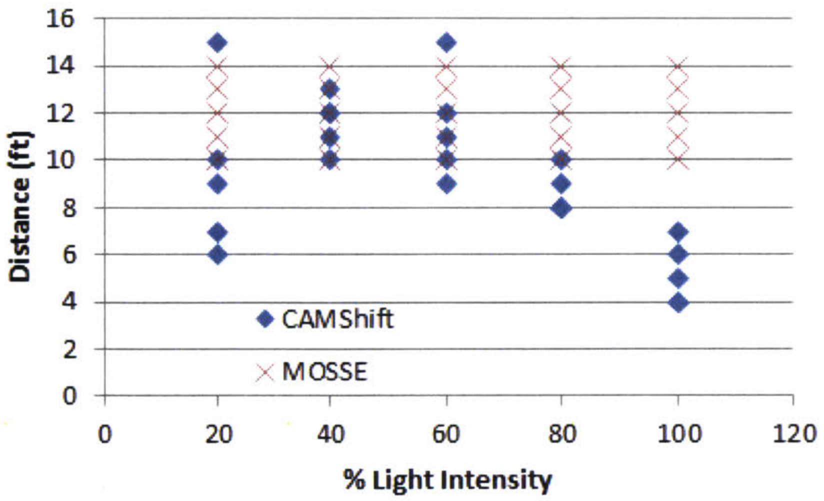


Figure 4-3: Results from testing using green light in tow tank.

tank waters. Reflections from the water surface and various walls did not impact the algorithm when using a red target. The increased performance is the result of red providing a high contrast with all of the surroundings. At approximately 45 ft, the area of the light source relative to the field of view was too small to have sufficient pixels to support the Camshift algorithm.

These results aided in the decision to utilize the Camshift algorithm with a red target for the navigational input for the buoy described in Chapter 5.

4.2 Pond Testing

In addition to testing in the tow tank, I wanted to see how the range of the tracking system would perform in more realistic conditions. I took the equipment to Indiana over the winter holidays to evaluate the system in our family's farm pond. It is approximately two acres of water, fed mostly by surface runoff.

The weather in the days leading up to my test had been rainy, and as a result, the water was quite cloudy. I built a Secchi disk to obtain a measure of the water clarity. The Secchi disk was made of a flat piece of sheet metal cut into a circle with a diameter of 20 cm. Alternating quadrants were paint black and white as shown in Figure 4-6. The disk was lowered until the sight of it was lost and the depth recorded.

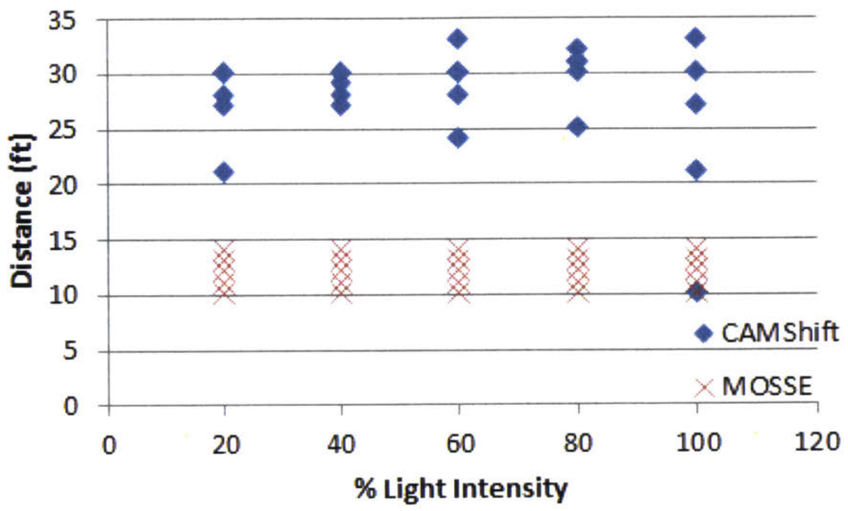


Figure 4-4: Results from testing using blue light in tow tank.

Then the disk was raised until it became visible and this depth was recorded. The average of the two measurements was 46 inches.

The water visibility was only 46", which severely limited the testing that could be performed. Tracks could only be maintained out to a range of 4ft, which coincides will with the visibility measurement using the Secchi disk. Like the testing in the tow tank, multiple colors were tested: red, green, and blue. Additionally, both the Camshift and Mosse algorithms were used. The depth of the test was 24". In the turbid water, no color exhibited superior performance. Additionally, neither algorithm performed noticeably better than the other. The turbid water provided a monochromatic environment, which served to eliminate the reflections that hindered the performance in the tow tank.

The water obscured the target and visibility dropped off quickly. Figure 4-7 illustrates the poor visibility and how quickly conditions deteriorated between two feet and four feet. At four feet, the target is basically no longer visible and the general area is such a haze that the algorithm can not effectively maintain a track.

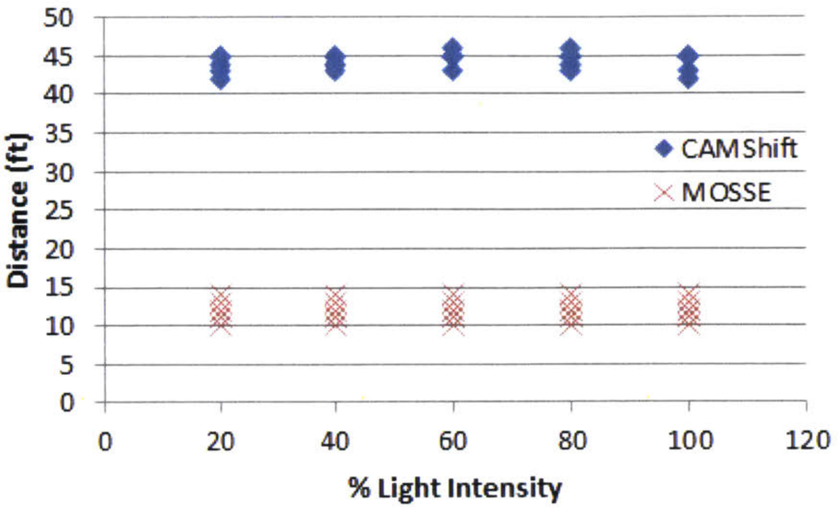


Figure 4-5: Results from testing using red light in tow tank.

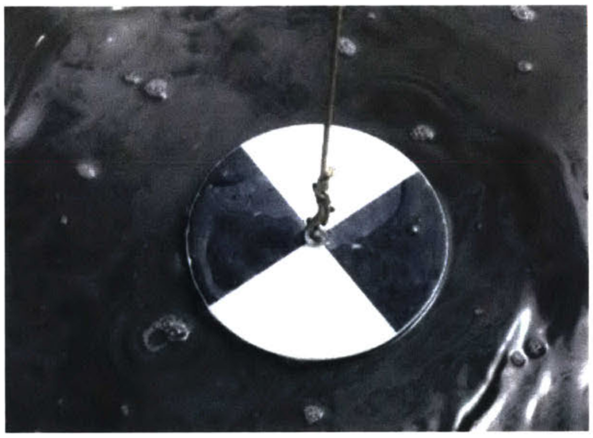
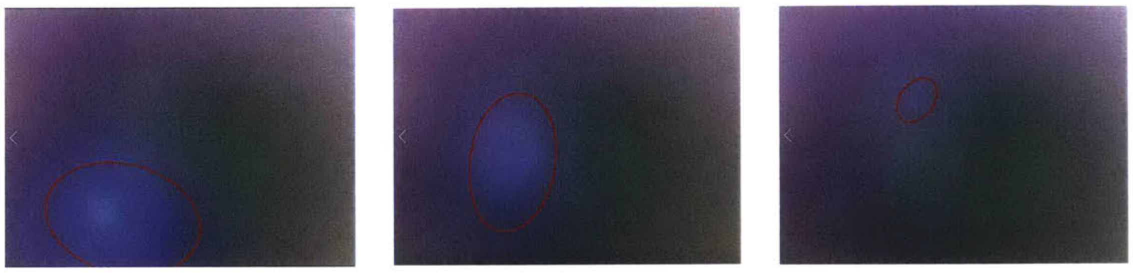


Figure 4-6: Secchi disk used to measure the clarity of the water.



(a) 2 ft. (b) 3 ft. (c) 4 ft.

Figure 4-7: Video captures taken during pond testing.

Chapter 5

Implementing the Tracker on a Vehicle

5.1 Creating a Testing Platform from a Buoy

With a basic idea of the maximum useful range in hand, as well as some of the pros and cons of the different algorithms, I wanted to implement the tracking into an actual control system to allow for further testing. I decided to utilize an existing buoy and modify it to serve underwater. The buoy provided a great starting point as it already had the necessary hardware and software to operate on the surface of the water. The buoys are driven with a Beaglebone Black (BBB) which runs a Debian Linux Distribution. The BBB accepts a standard USB input to connect the camera. It also provides for relatively easy implementation of Python code, allowing OpenCV to be added and used. The buoy was upgraded with some new components and updated software. Appendix D goes through some of the steps that were required to set up the buoy.

5.2 Modifying the Buoy

By simply submerging the buoy and operating it on the bottom of a body of water, the performance of the camera and tracking software could be tested without having

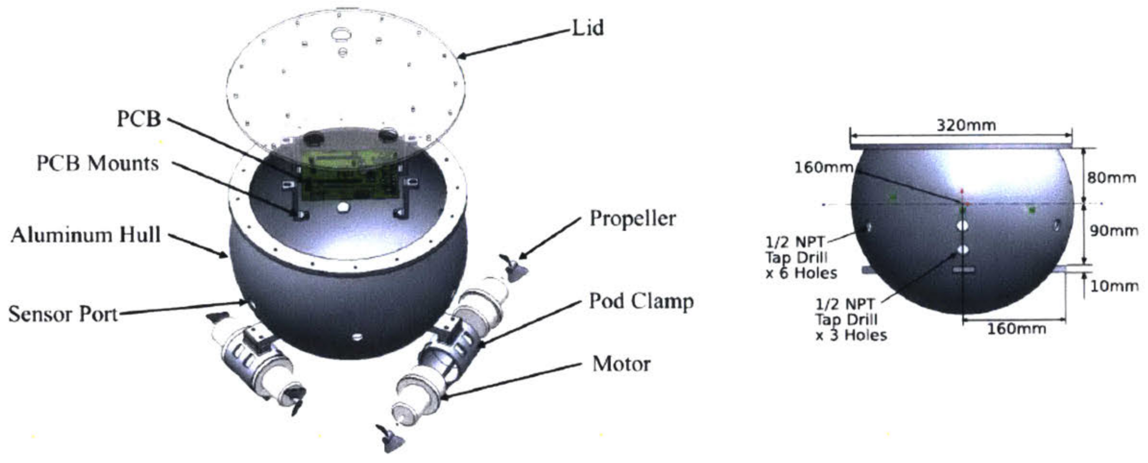


Figure 5-1: Buoy components and dimensions [Zoss et al., 2018].

to worry about controlling the buoy in three dimensions, just controlling its horizontal position. The buoy was modified with a base and set of wheels as shown in Figure 5-2. The buoy could then be ballasted down and submerged, then simply roll along the floor of a tank or pool for testing purposes. The buoy was kept just slightly negatively buoyant, which minimizes any extra resistance from the wheels and provided an easy transition to the suspension system I implemented later.

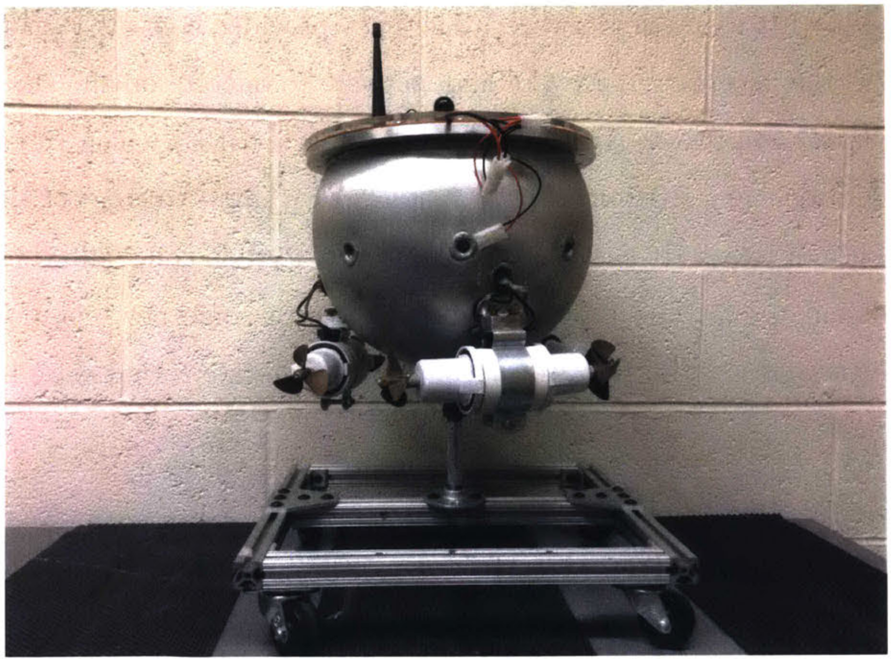


Figure 5-2: Modified buoy for sub-surface operation.

Since I decided to sink the buoy, I wanted to verify the acrylic lid would not break while submerged. In order to keep the buoy watertight, the original lid was replaced with one without any penetrations for the plugs and antennas. This minimized the potential for leaks. Additionally, the original lid was relatively thin, so there was concern of brittleness as it was several years old. A new lid was made using 12 mm acrylic to provide added strength, but still allow the camera to see through it. Appendix C contains the calculations done to see what depth it would be safe at.

The camera was mounted to the board containing the circuitry and aligned with the heading of the buoy, as shown in Figure 5-3. The BBB contains a USB port, allowing for an easy connection of the components. The original code was modified to incorporate guidance from the tracking algorithm, which can be seen in Appendix E. This code reads an image of the target, converts it to the HSV spectrum, and determines the color histogram. The sample code below shows the necessary commands.

```
###########  Import Target   ####################
target_picture = cv2.imread('target.png')
hsv = cv2.cvtColor(target_picture, cv2.COLOR_BGR2HSV)
hist = cv2.calcHist([hsv], [0], None, [16], [0, 180])
cv2.normalize(hist, hist, 0, 255, cv2.NORM_MINMAX)
```

The algorithm then reads a frame from the video stream. The image is blurred, and converted to HSV as well. Pixel values that are extreme are filtered out. Once this is complete, the back projected probability is calculated for use by the actual Camshift algorithm.

```
######### Prepare Image #######################
# Read the image from video stream
grabbed, frame = cam.read()
# Apply Gaussian blur it
blurred = cv2.GaussianBlur(frame, (11, 11), 0)
# Convert the HSV color space
```

```
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
# Remove extreme values
limit=cv2.inRange(hsv,np.array((0,60,32)),np.array((180,255, 255)))
# Calculate Probability map
mask = cv2.calcBackProject([hsv], [0], hist, [0, 180], 1)
# Limits for Convergence
term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)
# Apply Camshift algorithm
track_box, track_window = cv2.CamShift(mask, track_window, term_crit)
```

This tracking code was ran in parallel with the other buoy functions by utilizing the Python Threading package. The relative position of the target provides a heading for the buoy, along with a speed that is computed using the range to the target.

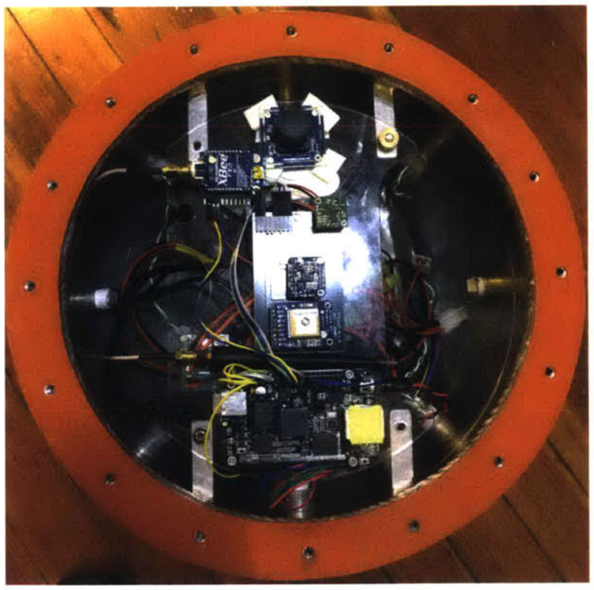


Figure 5-3: Location of the camera inside of the buoy, top center.

Although the buoy cannot communicate while under water, the module for wireless communication wireless capabilities were taken advantage of. The motors can still be armed or disarmed when the buoy is out of the water, by using the "Turn Motors On" and "Turn Motors Off" buttons. The signal is able to penetrate the acrylic lid at close range, allowing the motors to be turned on and off when moving the buoy in and out of the water and working on it. This can extend the battery life during testing and keep everything drier and safer. Additionally, the wireless communications can be used to change various tuning constants and enable data recording before submerging the buoy.

5.3 Propulsion Commands

The buoy is driven by three pods with forward and reverse propellers on each, shown in Figure 5-4. These are controlled using a desired heading and motor speed. The heading and speed are used to solve for the motor output which generates the thrust in the proper direction and a net moment of zero on the buoy. This is done in order to make the buoy translate in the proper direction without turning.

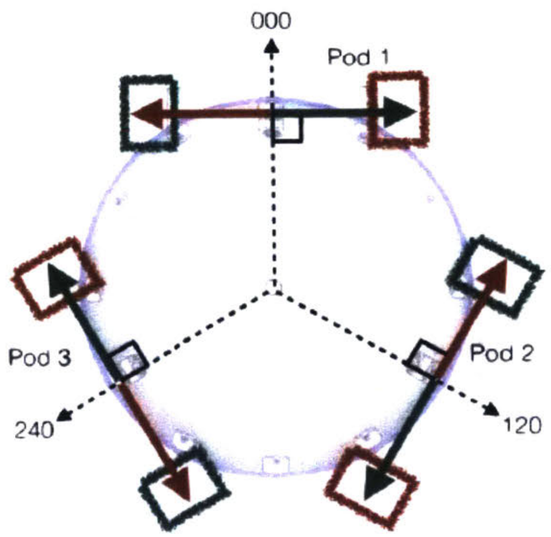


Figure 5-4: Orientation of the propeller pods. Green indicates forward and red indicates reverse. [Zoss, 2016].

In order to determine the speed and direction for the buoy to go, the location of the target in the image is used. The center of the target, x, y, is provided by the tracking algorithm and is used as an input to a PID controller which is used to determine propulsion commands.

The tracking algorithm returns a rotated rectangle in the form of a center point, height and width, and angle of rotation. The x and y coordinates for the center location are given the following labels: box_x and box_y. The digital image is referenced from the top left corner, so the frame of reference is shifted to the center of the image. In the following equations $FrameWidth = 320$ and $FrameHeight = 240$.

$$Target_x = box_x - \frac{1}{2}FrameWidth \tag{5.1a}$$

$$Target_y = -(box_y - \frac{1}{2}FrameHeight) \tag{5.1b}$$

The motor output is measured as a percentage, from 0 to 100. In order to have K values on the order of 1, the coordinates of the target location were converted to percentages as shown in Equations 5.2. This results in an x of zero being the center and an x of 100 being the edge of the frame.

$$X = \frac{Target_x}{\frac{1}{2}FrameHeight} * 100\% \tag{5.2a}$$

$$Y = \frac{Target_y}{\frac{1}{2}FrameHeight} * 100\% \tag{5.2b}$$

These locations were then used as inputs to a PID control algorithm. The general form for a PID controller is shown in Equation 5.3

$$Output = K_p e(t) + \int K_I e(t)dt + K_D \frac{d}{dt}e(t) \tag{5.3}$$

$$Where : e = Setpoint - Input$$

For this application, the target coordinates were substituted in as the error values. These were then used to calculate the thrust vector needed. The thrust in the x direction, T_x and the thrust in the y direction, T_y, were determined using Equations 5.4. K_p, K_I, and K_D are all constants that can be used to tune the response of the buoy. They can all be adjusted using the graphical user interface.

$$T_x = K_P x + \sum K_I x \Delta t + K_D(dx) \tag{5.4a}$$

$$T_y = K_P y + \sum K_I y \Delta t + K_D(dy) \tag{5.4b}$$

The two thrust vectors can be combined to determine the net thrust. This is done using Equation 5.5. This value is limited to 100% using a simple if statement in the buoy code. Additionally, the angle of thrust is found using Equation 5.6.

$$Thrust = \sqrt{T_x^2 + T_y^2} \tag{5.5}$$

$$\theta = \arctan \frac{T_y}{T_x} \tag{5.6}$$

The full implementation is shown in the code given in Appendix E.

5.4 Improving System Performance

The first camera that was utilized provided an inadequate field of view. As a result, it was replaced with a wide angle version. Originally, an ELP 5.0 Megapixel USB camera (ELP-USB500W02M-L36) was used on the buoy. This camera has a 3.6 mm lens and a field of view of 30°. In order to improve the ability to maintain the track on the target, this camera was replaced with an ELP 8.0 Megapixel USB camera (ELP-USB8MP02G-L170). This new camera has a 170°field of view which is achieved with the use of a fish eye lens. Figure 5-5 shows a comparison of the two cameras and it is clear that the new camera should provide a much better view of the surroundings.

While the new camera is advertised as having 170 degrees of viewing angle, actual performance did not seem to be that good. The installation affects the viewing angle, as the body of the buoy can limit the field of view if the camera is mounted too deep inside the shell. In order to measure the installed field of view, the camera was mounted and connected to a laptop. The lid was placed on the buoy and the camera was used to see when an object could be seen to enter the frame. Using some simple trigonometry, the horizontal field of view was found to be 114°, while the vertical field was 88°. This confirmed that advertised value was overly optimistic.

Since the limiting field of view of view is 88°, this was used to determine the

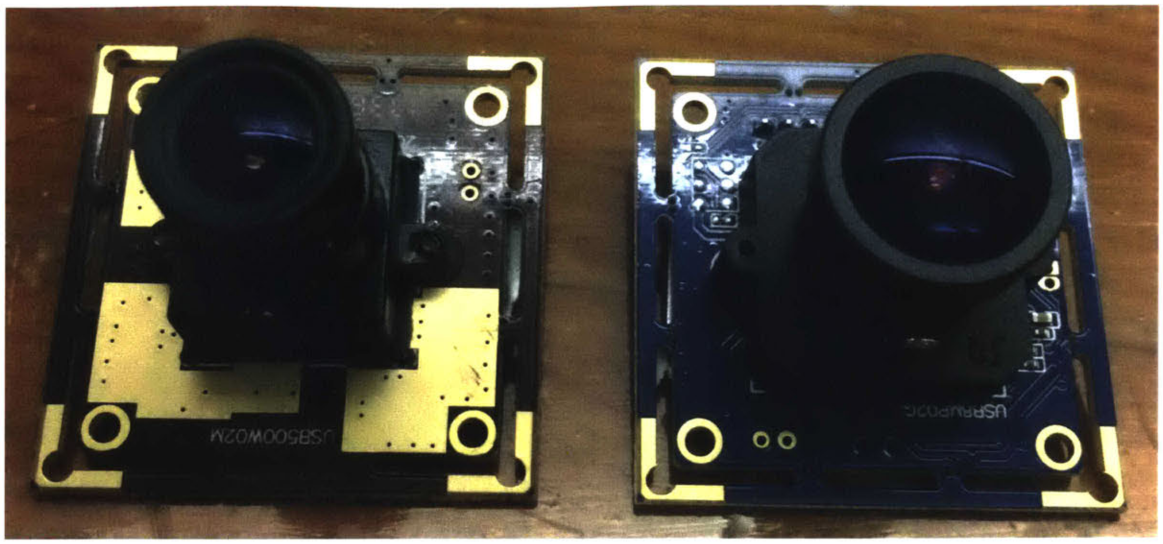


Figure 5-5: Comparison of the 30 degree lens on the left and the 170 degree lens on the right.

maximum horizontal displacement that the target could reached before leaving the frame. The radius of the field of view r is a function of both the depth d and the angle of the field of view θ.

$$r = d \tan \frac{\theta}{2} \tag{5.7}$$

In the tow tank, the depth the buoy operates is 24", which gives a circle of radius 23" that the target must remain inside.

The image processing added a significant computational load on the BBB. To improve the ability to handle this, a couple of changes were made to the BBB operating system. First, the graphical desktop that can be used via an HDMI output to a monitor was turned off to reduced to processing load. Secondly, the BBB has a processor that supports a variety of cpu frequencies. This was taken advantage of by increasing the frequency to 1 GHz.

5.5 Support Equipment

Since the Camshift algorithm was chosen for the initial testing, a target was needed that could provide the required color. The Adafruit 8x8 LED matrix that was previously used for evaluating the range was again used. A 5V voltage regulator was wired up with a 12V LiPo battery to supply power to the LED board as well, as the Arduino needed to drive the matrix. Figures 5-6 and 5-7 show the wiring and equipment that was used. The components were placed in a waterproof container which could easily be moved around with light nylon line to lead the buoy around.

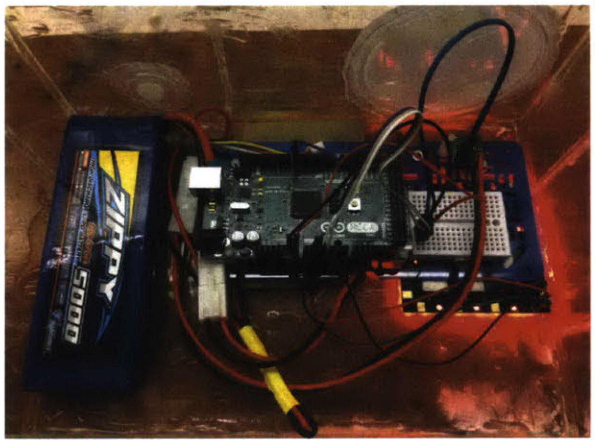


Figure 5-6: Components used for the LED target.

5.6 Buoy Suspension System

In order to test the setup in the Charles River, the buoy was modified again to be free floating. The base and wheels provided a convenient setup for testing in the tow tank, but were not suited for the muddy, uneven surfaces found in natural environments. Additionally, the base provided no way of adjusting the operating depth of the buoy. Since the buoy was only slightly negatively buoyant, the base was removed and a system of nylon line was attached to suspend the buoy from a piece of high density foam from the surface. The length of the suspension system can be adjusted to allow for testing at various depths.

5.7 Graphical User Interface

In order to facilitate testing and make tuning the constants easier, the wireless capabilities of the buoy were utilized. The graphical user interface of the existing buoy

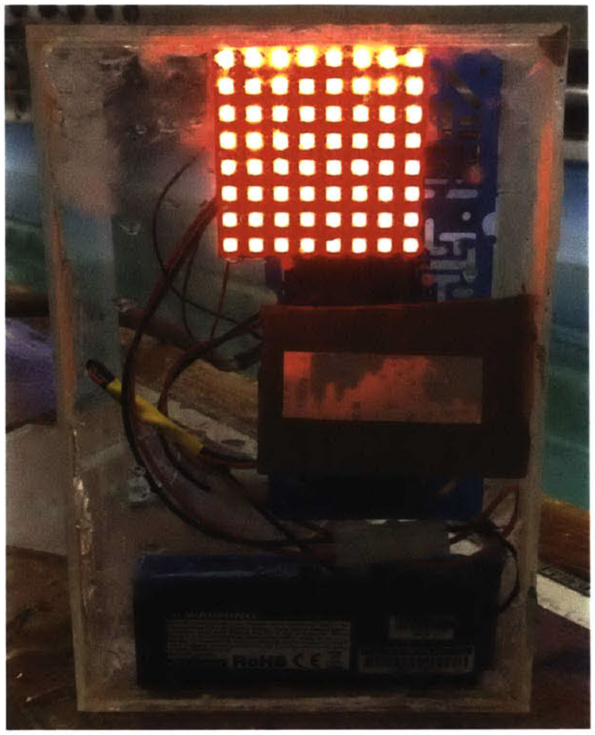
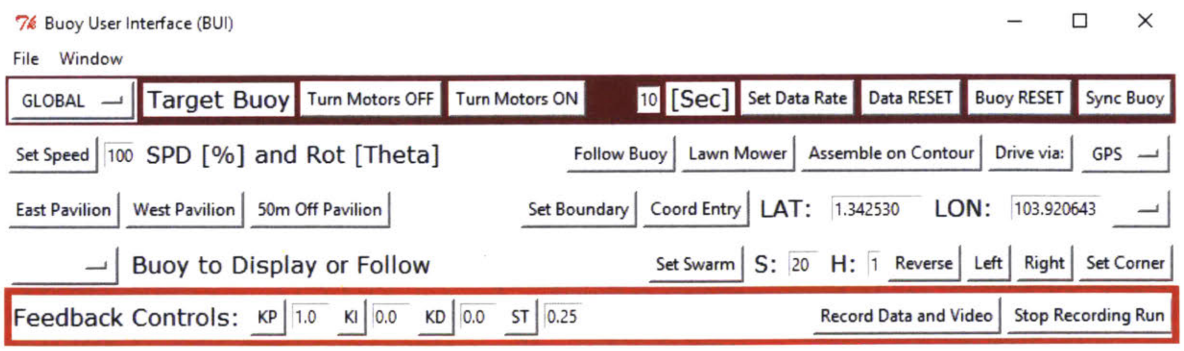


Figure 5-7: Components used for the LED target.

was modified to provide a way to pass settings to the buoy when it is removed from the water. Buttons and commands for the constants K_p, K_I, K_D, and the time between samples, ST, were added to the interface, shown in Figure 5-8. Additionally, in order to better understand the buoy's ability to identify the target and maintain position, data recording was added. This provides video and a variety of data from the tracking and propulsion calculations.



GLOBAL

10 [Sec]

Set Speed 100 SPD [%] and Rot [Theta]

Drive via: GPS

LAT: 1.342530 LON: 103.920643

S: 20 H: 1

Feedback Controls: KP 1.0 KI 0.0 KD 0.0 ST 0.25

Figure 5-8: Modified GUI which supports changing propulsion parameters and recording camera data and video.

Chapter 6

Testing the Platform

Once the buoy was modified, testing was performed to help understand and improve the buoy's performance. Additionally, it is desired to develop an understanding of how the buoy responded to various frequencies of target movement. Testing was conducted in the tow tank and Charles River.

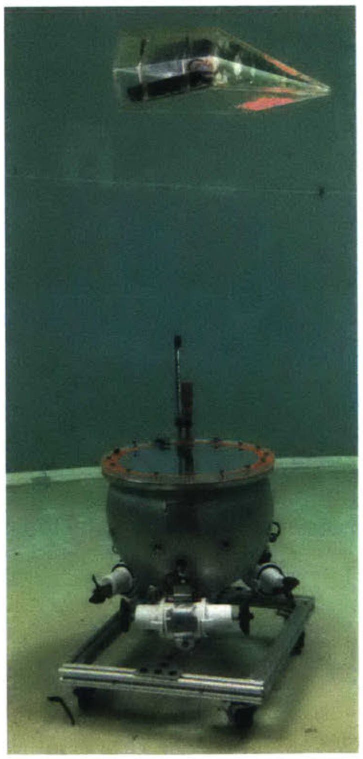


Figure 6-1: Testing setup used in the Tow Tank.

6.1 Tow Tank

The buoy was submerged and the target was 42" from the camera. The target was moved horizontally by 3ft at a speed of 1ft/sec, first in one direction and then back to the original location after a specified period. This was then repeated approximately 10 times. The buoy was allowed to track the target and collect data. This process was repeated for a variety of oscillation periods.

Visibility in the tow tank was in excess of 50ft when measured with the camera during horizontal testing. Figure 6-2 shows the world from the buoy's point of view.

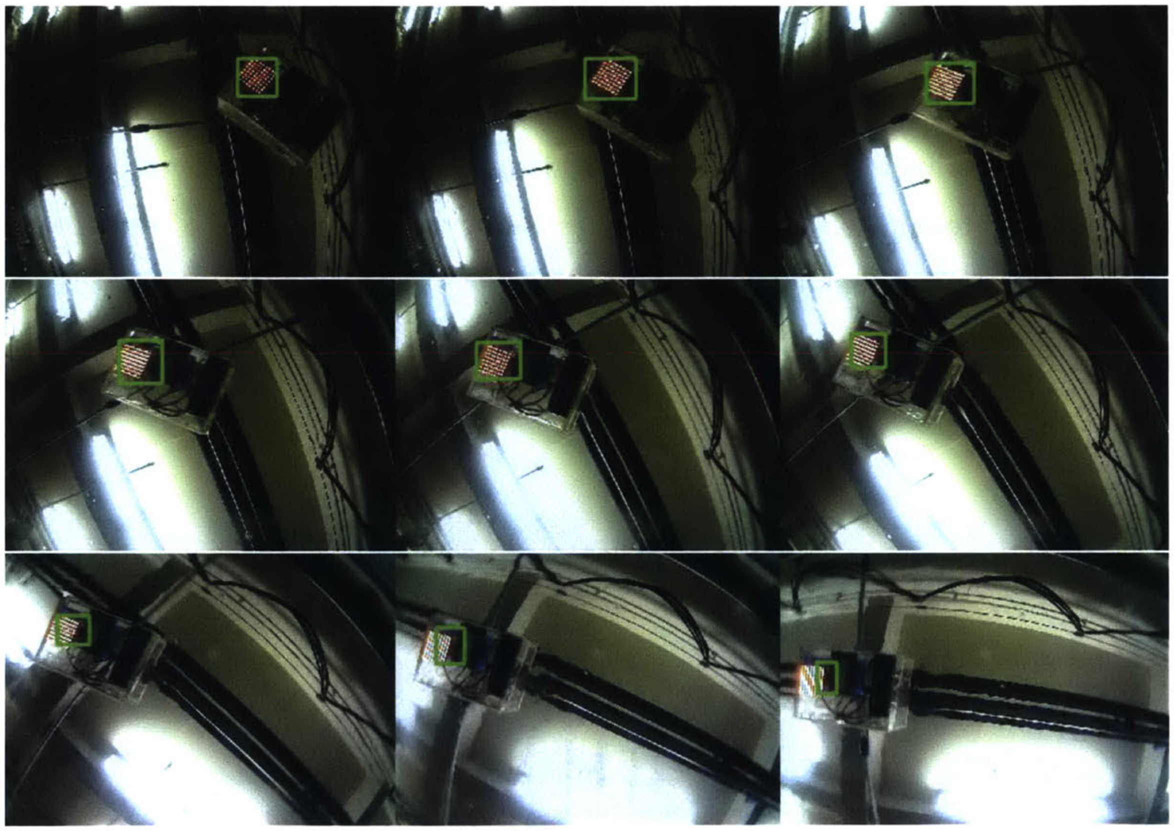


Figure 6-2: A video stream from the buoy while tracking underwater in the Tow Tank. Each image is 10 frames later than the previous image.

Figure 6-3, 6-4, 6-5, 6-6, 6-7, 6-8, 6-9, and 6-10 show a number of runs carried out with a periods of $T = 15s$, $T = 10s$, $T = 8s$, and $T = 5s$, respectively. The figures contain dashed vertical lines which represent the start of the target motion. The propulsion parameters used are shown in Table 6.1.

K_P	K_I	K_D	ST
1.0	0	0	0.25

Table 6.1: PID constants and sample time used for testing shown in Figures 6-3 through 6-10.

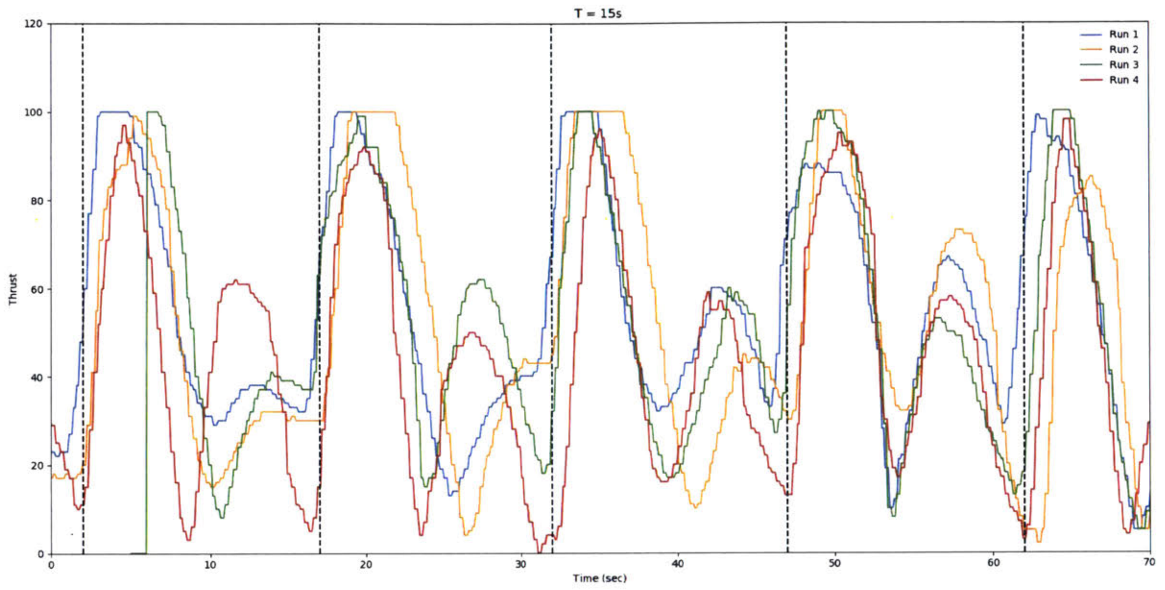


Figure 6-3: Thrust for input target period of $T = 15s$.

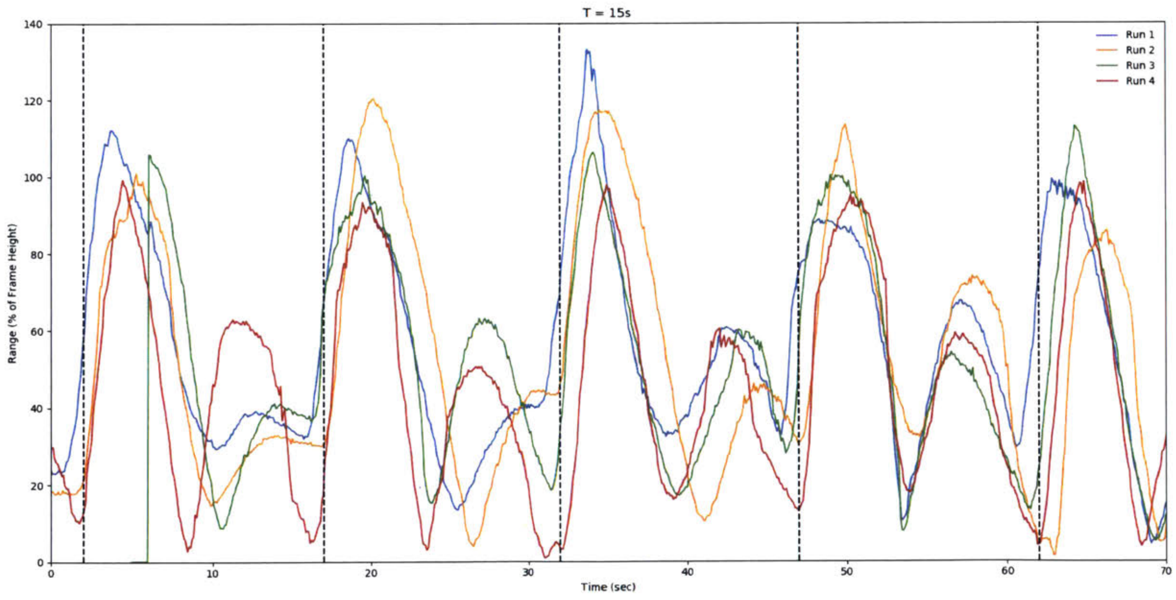


Figure 6-4: Target range for target period of $T = 15s$.

Figure 6-4 shows that there was a significant overshoot. This can be seen in the large burst between the initial movement. Later runs utilized derivative control as well in order to try to minimize this overshoot. Also of note is the flat portion at the top of the peaks. This results from limiting the output to 100%. As mentioned in the previous chapter, the target location is measured as a percentage of the frame height, as shown in the following equation.

$$X = \frac{P_x}{\frac{1}{2}H} * 100\% \tag{6.1a}$$

$$Y = \frac{P_y}{\frac{1}{2}H} * 100\% \tag{6.1b}$$

In Equation 6.1, P_x and P_y represent the pixel locations in the x and y directions and H represent the total height of the frame. The frame height was used to scale both the y and x directions so that with a K_p value of 1, the buoy output is 100% when the target reaches the closest edge of the frame of the image. As a result, the x location can be greater than 100% since the frame is wider than it is tall. The range is then calculated using Equation 6.2.

$$R = \sqrt{P_x^2 + P_y^2} \tag{6.2}$$

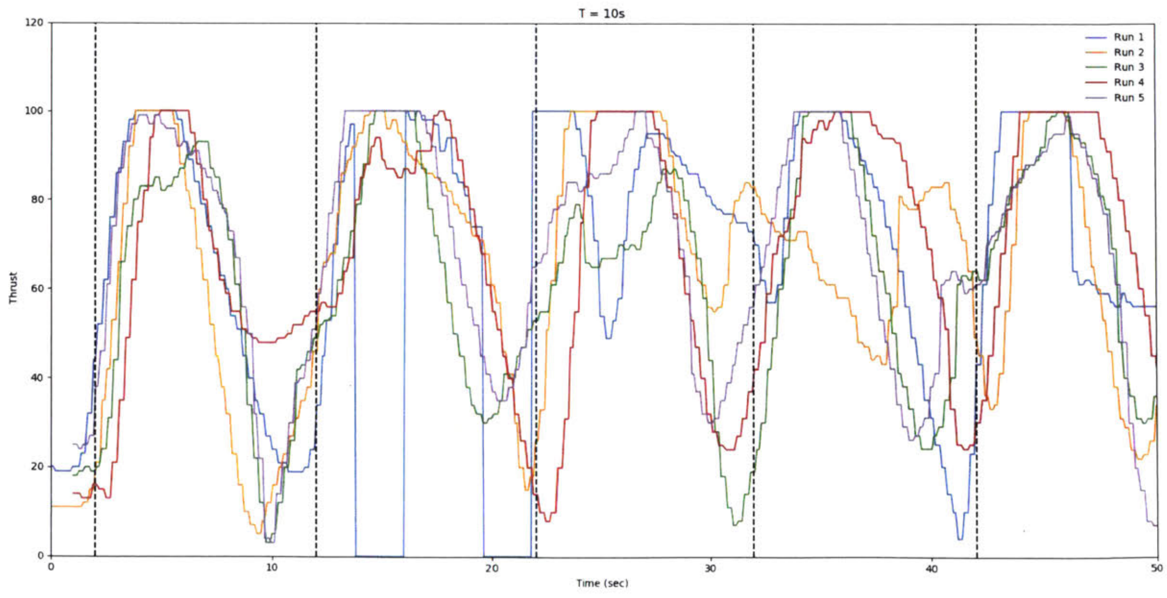


Figure 6-5: Thrust for input target period of $T = 10s$.

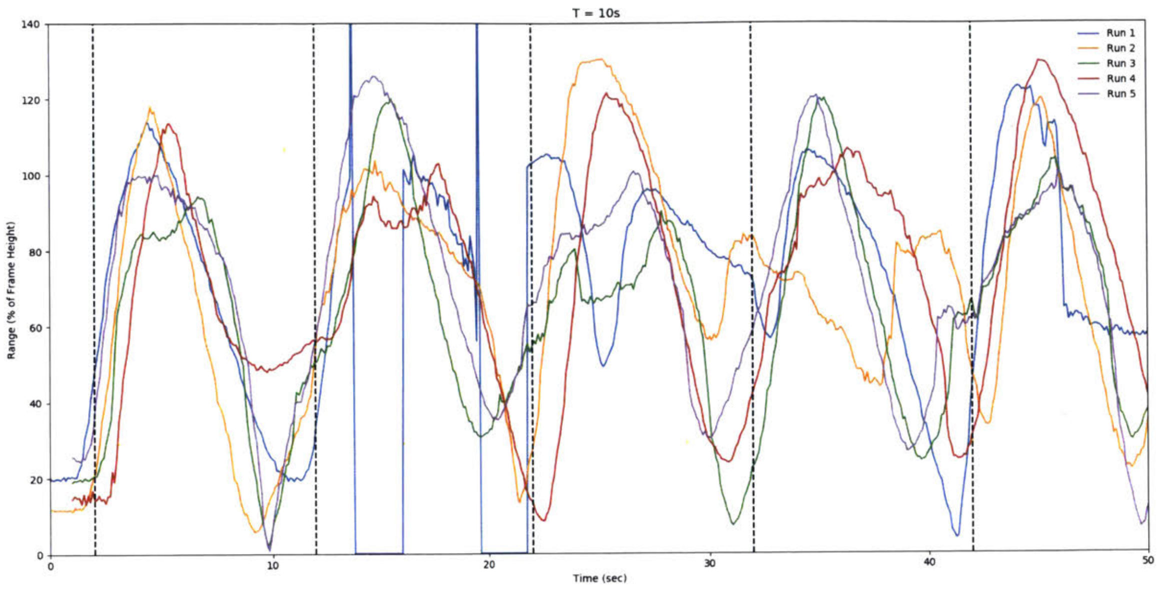


Figure 6-6: Target range for target period of $T = 10s$.

Figure 6-5 and 6-6 shows the results from runs with a period of $T = 10s$. The shorter period drove the buoy in the other direction quickly, giving it little time to overshoot. In two locations, Run 1 dropped the thrust to zero. This is the result of the buoy losing the target. It is programmed to set the outputs to zero if the target is lost in order to prevent the buoy from driving off in a random direction. It can be seen that the target was reacquired in approximately 2 seconds. This acquisition process was modestly reliable but could be improved in the future.

Figure 6-7 and 6-8 contains the runs with $T = 8s$. The responses are consistent and the buoy was able to maintain the target. On the other hand, Figure 6-9 and 6-10 shows the results for $T = 5s$. It can be seen that the buoy output rarely reached the full 100%. This is because the buoy sat virtually in the middle of the oscillating target and didn't move. For input periods less than 5 seconds, the buoy can not follow the target. While the target moved back and forth a distance of 3ft, the buoy effectively sat stationary, while the target moved 1.5ft to each side. It can also be seen that the buoy occasionally lost the target, approximately 10% of the time.

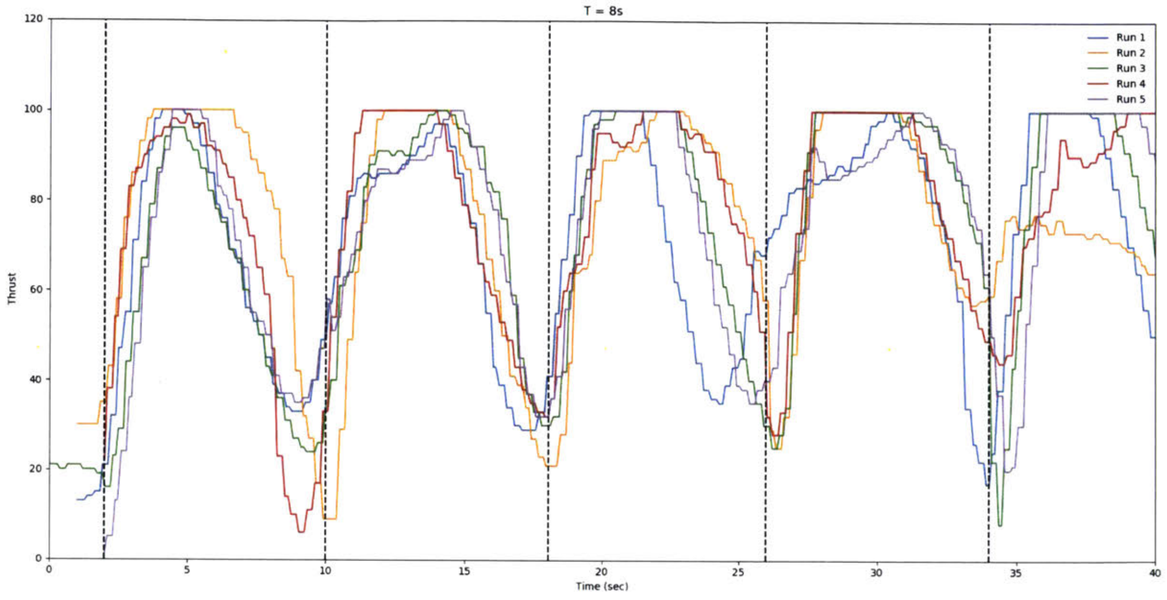


Figure 6-7: Thrust for input target period of $T = 8s$.

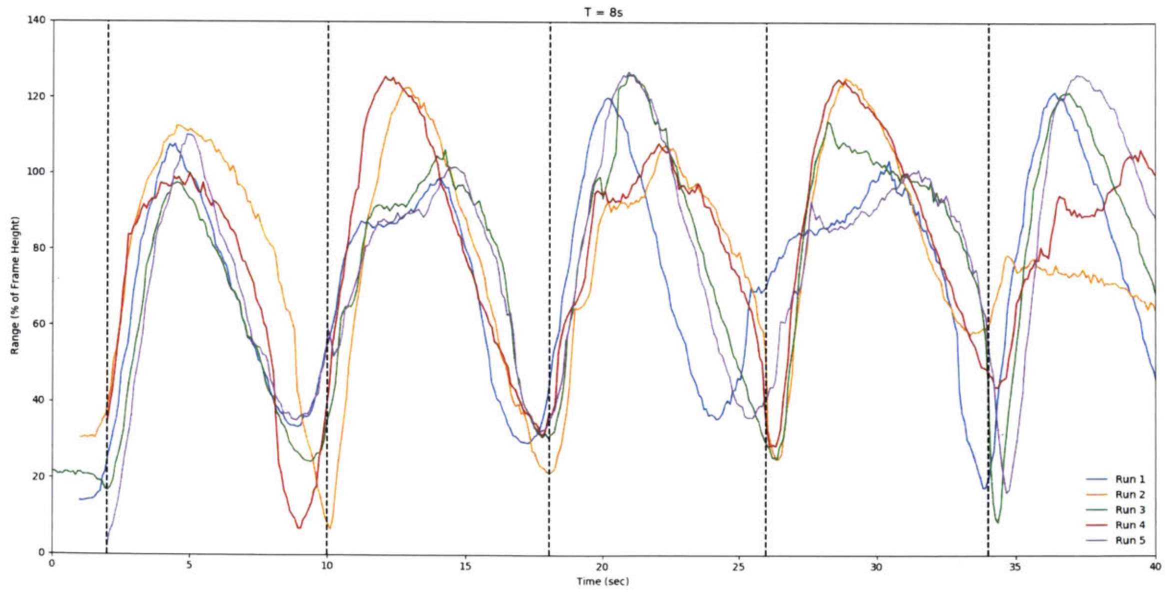


Figure 6-8: Target range for target period of $T = 8s$.

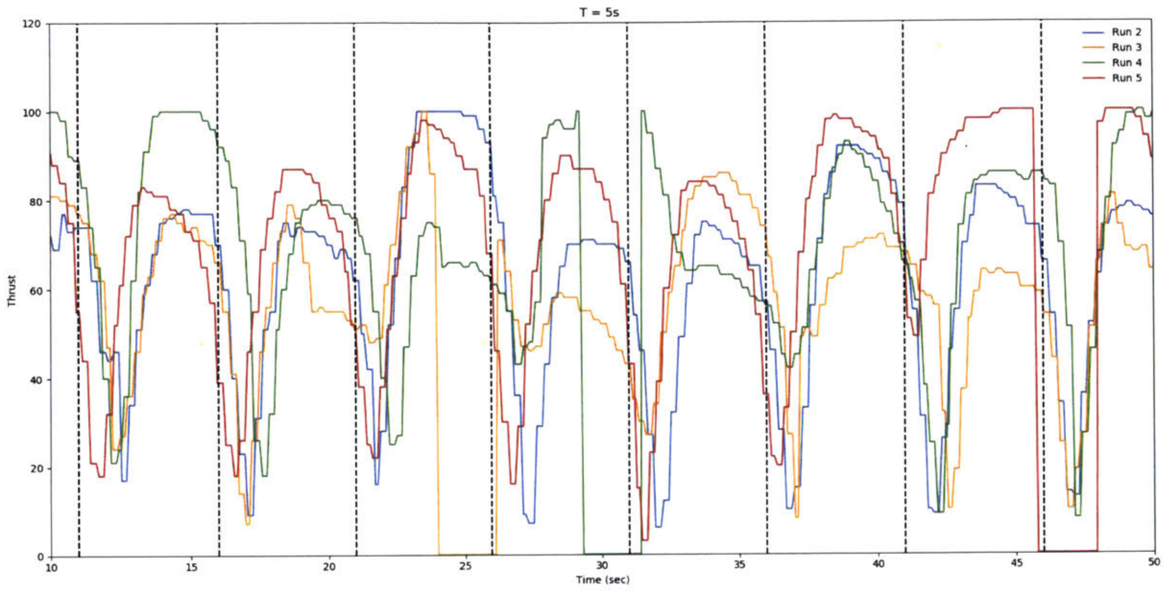


Figure 6-9: Thrust for input target period of $T = 5s$.

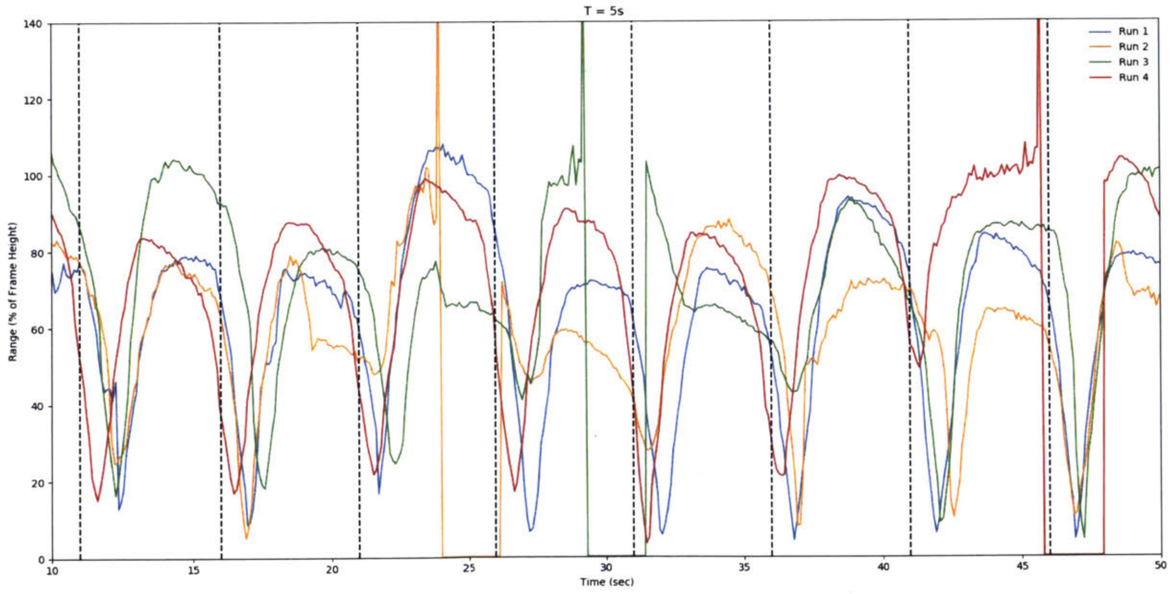


Figure 6-10: Target range for target period of $T = 5s$.

The same runs were also performed with a higher value of $K_p = 1.5$. Other values were held constant and are shown in Table 6.2.

K_P	K_I	K_D	ST
1.5	0	0	0.25

Table 6.2: PID constants and sample time used for testing shown in Figure 6-11 and 6-12.

These runs showed increased overshoot. The most significant effect of this can be seen in Figure 6-11 and 6-12. The locations where the thrust drops to zero are the result of losing the target, which occur about 30% of the time. The same unreliability was apparent in runs with longer periods as well; however, for the longer runs the buoy had longer to correct itself so the results were not as severe.

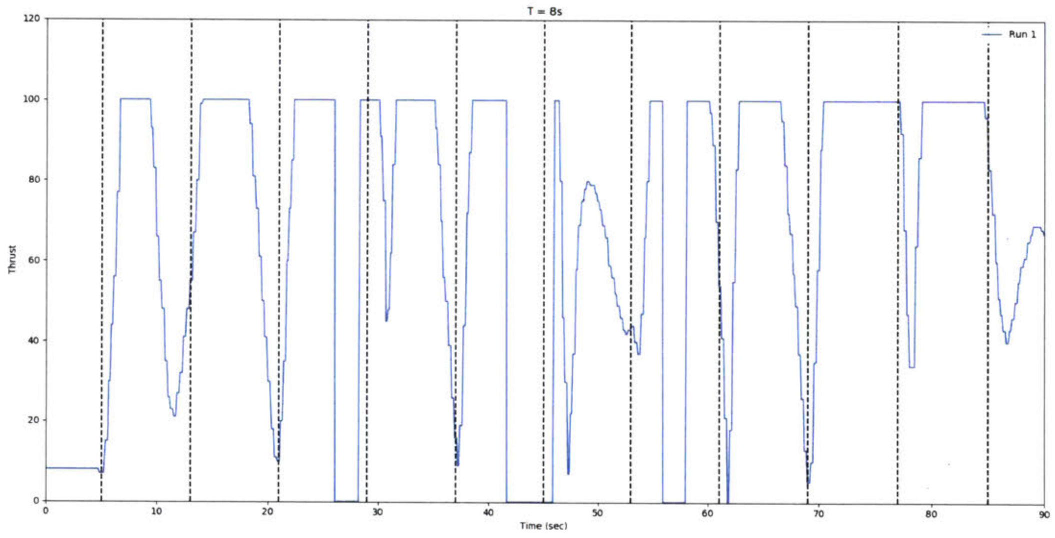


Figure 6-11: Thrust for input target period of $T = 8s$ showing the significant loss of target.

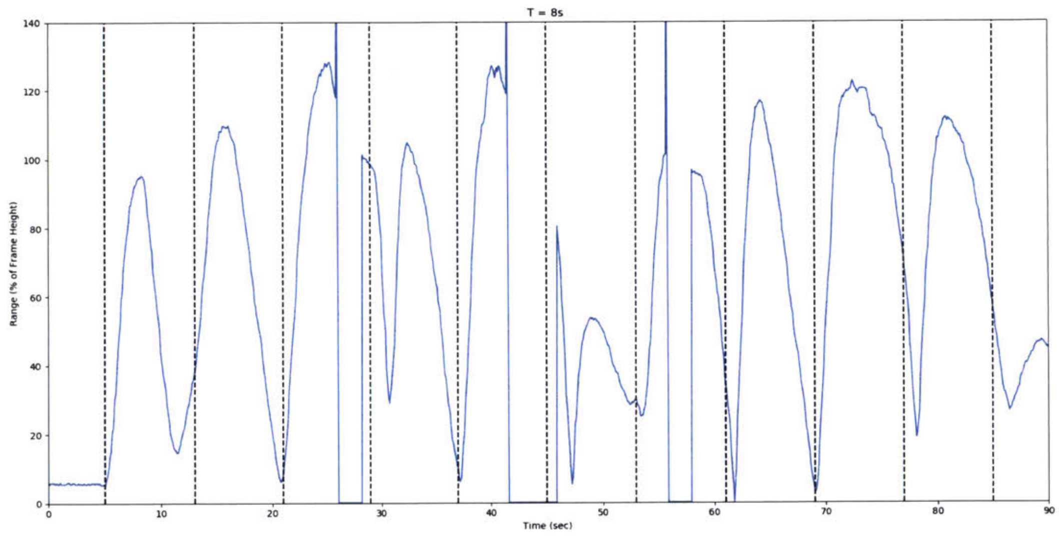


Figure 6-12: Target range for target period of $T = 8s$ showing the significant loss of the target.

The next set of runs tested the use of derivative control on the buoys response. Table 6.3 shows the constants that were utilized for testing. Figure 6-13 and 6-14 show a comparison of the runs with and without derivative control for a period of $T = 15s$ and Figure 6-15 and 6-16 show shows a comparison at a period of $T = 5s$.

K_P	K_I	K_D	ST
1.0	0	0.5	0.25

Table 6.3: PID constants and sample time used for testing shown in Figures 6-13 - 6-16.

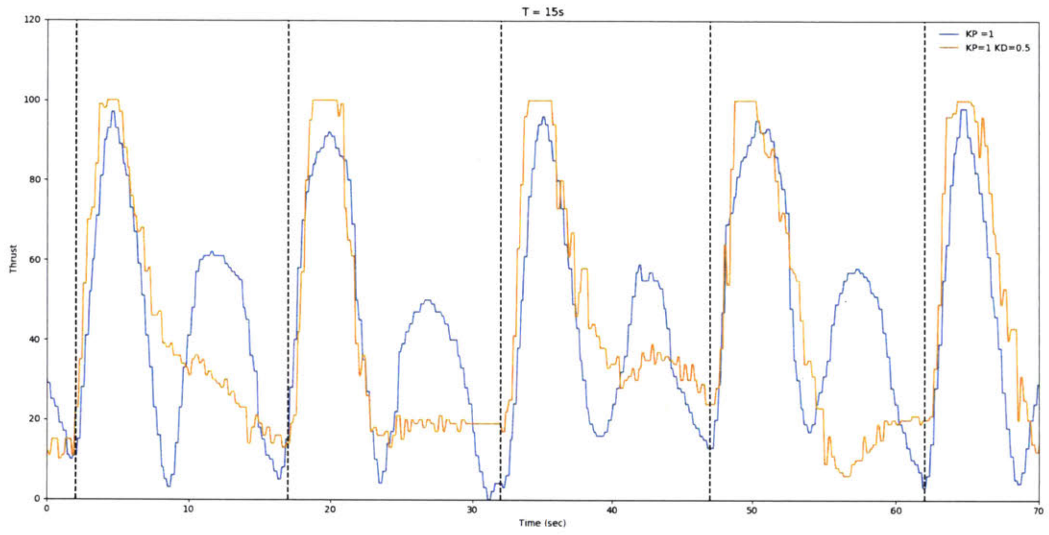


Figure 6-13: Thrust comparison of response with and without derivative control.

Figure 6-13 shows that the derivative control dramatically reduced the overshoot the buoy experienced. This significantly helped to stabilize the overall response and movements were visibly smoother.

Figure 6-15 shows that the derivative control provided a significant increase in the output of the buoy; however, the buoy still remained in the same location when tested at this frequency.

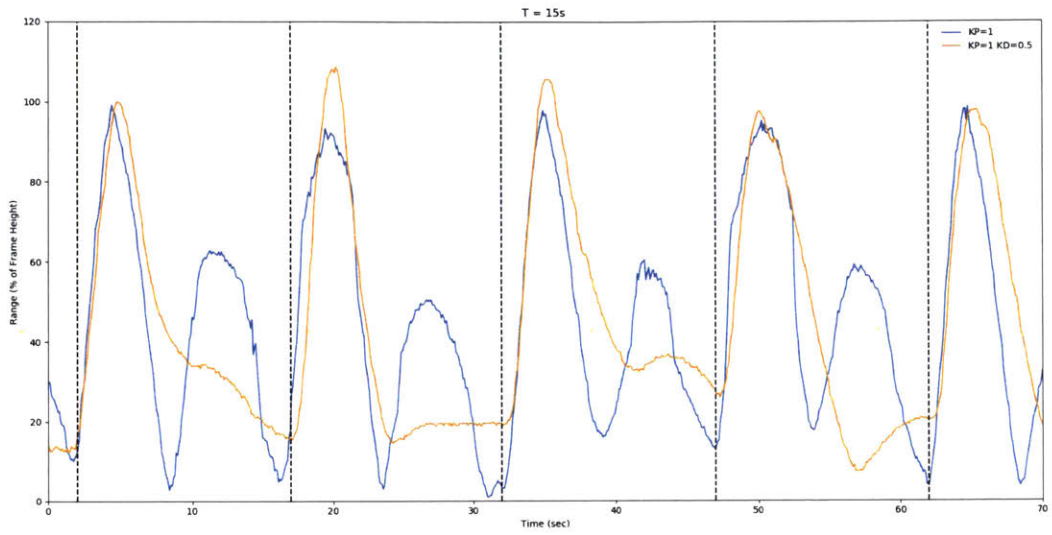


Figure 6-14: Range comparison of response with and without derivative control.

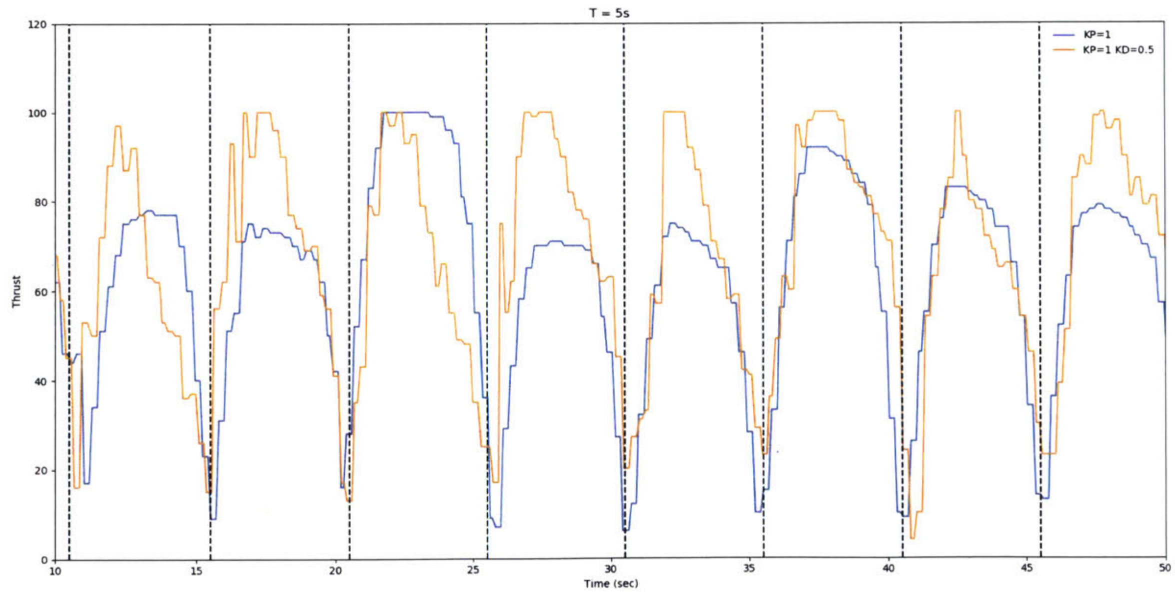


Figure 6-15: Thrust comparison of response with and without derivative control.

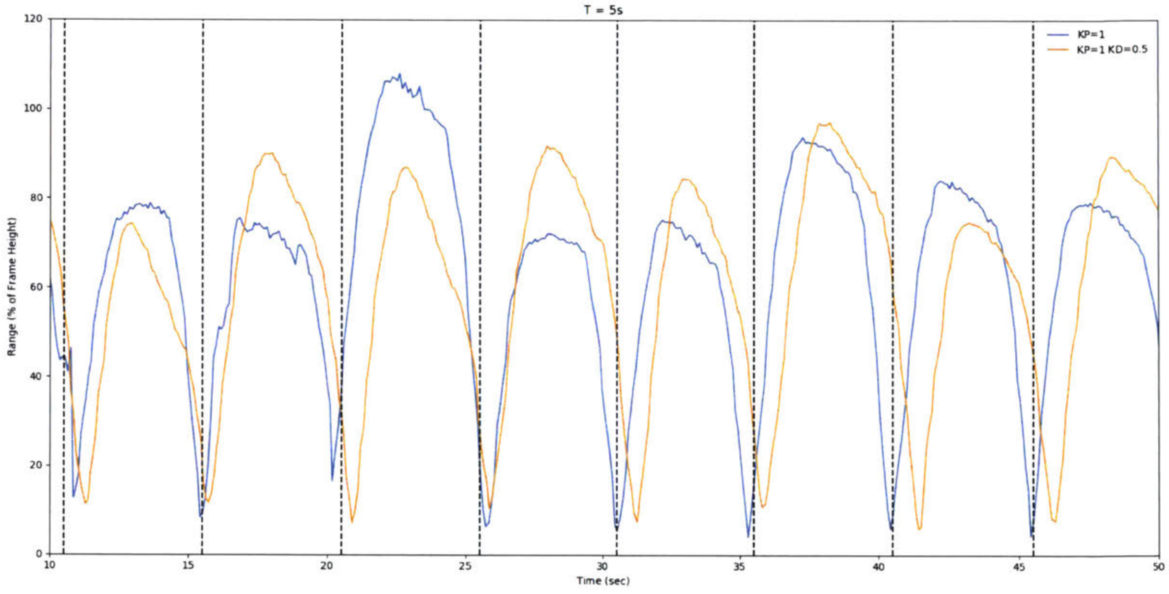


Figure 6-16: Range comparison of response with and without derivative control.

6.1.1 Buoy Stall

One observation from the experimentation was that after the target motion completed, the buoy would sometimes come to a rest with some horizontal offset from the target. A sample of this data is shown in Figure 6-17. When this would happen, the propellers would often still be rotating slowly. The thrust value, $\vec{T}_{Thrust}$, and heading, HDG, from the tracking algorithm is used to determine the output from each of the three pods. This is done so that Equation 6.3 and 6.4 are both satisfied.

$$\sum \vec{F}_{\text{buoy}} = \vec{F}_{\text{Pod 1}} + \vec{F}_{\text{Pod 2}} + \vec{F}_{\text{Pod 3}} = \vec{T}_{\text{Thrust}} \tag{6.3}$$

$$\sum \vec{M}_{\text{buoy}} = \vec{F}_1 + \vec{F}_2 + \vec{F}_3 \tag{6.4}$$

The pods are all equally spaced at a common distance from the buoy centerline. This leads to the following matrix equation which can be used to solve for the thrust from each pod.

$$\begin{bmatrix} \cos(\theta_{1x}) & \cos(\theta_{2x}) & \cos(\theta_{3x}) \\ \sin(\theta_{1y}) & \sin(\theta_{2y}) & \sin(\theta_{3y}) \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ M_z \end{bmatrix} \tag{6.5}$$

$$\begin{bmatrix} 0 & \cos 210^\circ & \cos 330^\circ \\ 1 & \sin 210^\circ & \sin 330^\circ \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \begin{bmatrix} \cos(HDG) \\ \sin(HDG) \\ 0 \end{bmatrix} \cdot |\vec{T}_{Thrust}| \tag{6.6}$$

The matrix is invert to solve for the output from each pod. This was was done on the data shown in Figure 6-17. The top two graphs show that the range and heading to the target are constant, which indicates the buoy was not moving relative to the target. The angle and thrust were collected from the buoy and used to calculate the pod outputs. It can be be seen that the net thrust determined by the tracking algorithm was approximately 25%. However, once the net thrust was distributed among the individual pods using Equation 6.6, the individual pod outputs were all much lower. Even though the buoy was trying to move, nothing was happening.

This could partly be attributed to friction from the test setup, but could also be due to poor DC motor performance at these low operating voltages. The red area in the lower two plots indicate motor outputs less than 20%. It is believed that below this value the output of the motor becomes nonlinear. Buoy performance could be improved in the future by increasing the minimum output, utilizing integral control, or possibly changing motors.

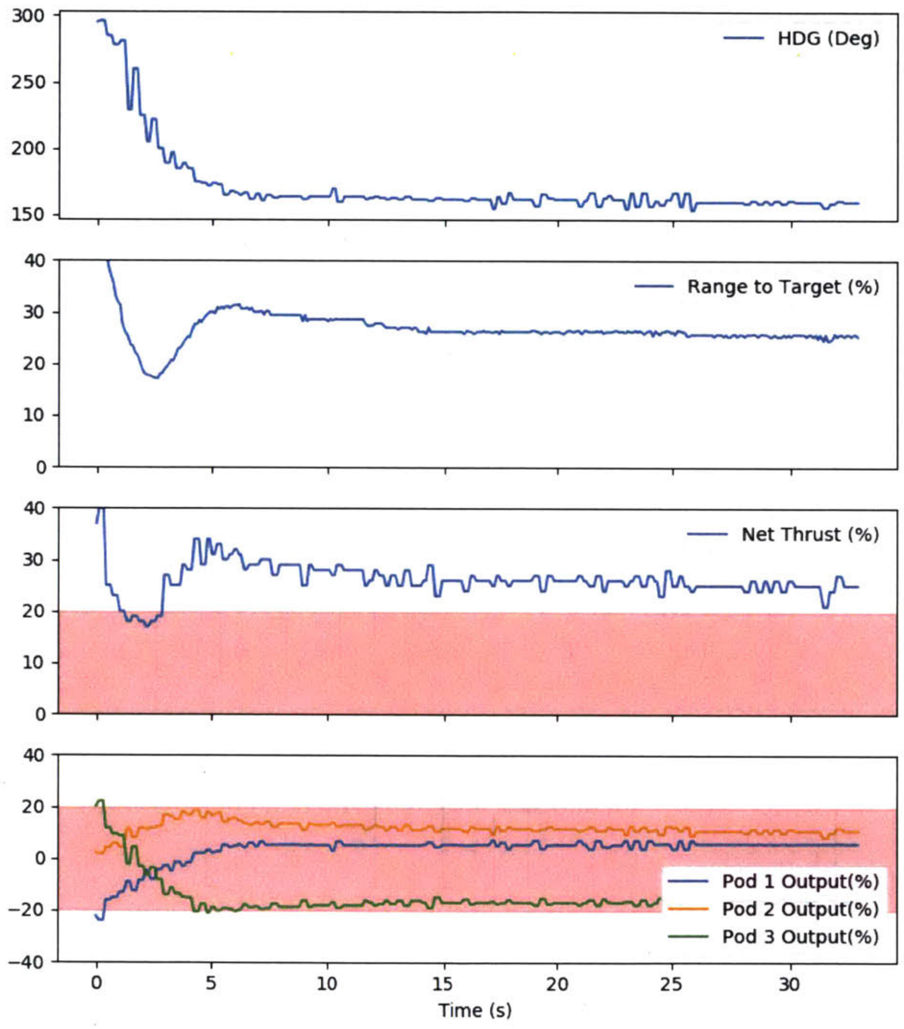


Figure 6-17: Top to Bottom: Heading to the target, range to target, net thrust calculated by the tracking algorithm, and individual pod outputs.

6.1.2 Long Distance Tracking

Runs were also done to test the buoys ability to follow in a straight line and see what speeds could be achieved. Figure 6-18 shows one such run. This run was carried out at a speed of 1ft/sec. This represents the maximum speed that the buoy would reliably follow at. The heading to the target, HDG, can be seen in the upper plot. It fluctuates by about 40°during the run. The second plot shows that the net thrust was almost nearly 100% in order to keep up with the target.

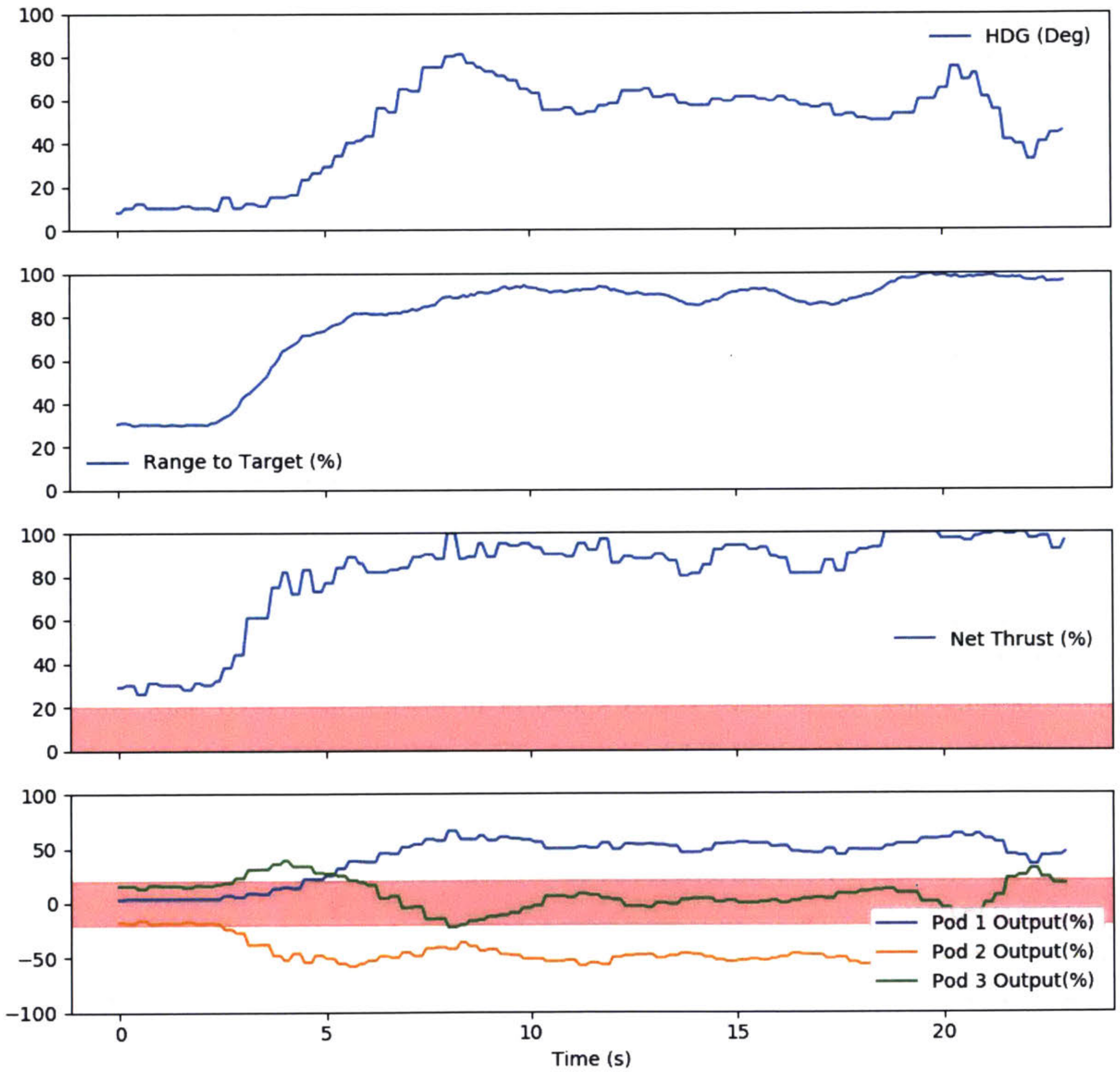


Figure 6-18: Buoy performance during a straight run at a speed of 1ft/sec. Thrust output and target angle are shown.

The bottom two plots in Figure 6-18 shows the net thrust and individual pod outputs for the three propeller pods during the straight run. It can be seen that Pod 3 has an output that fluctuates around zero. The fact that the pod output is likely nonlinear at low values and is producing very little thrust could be a possible cause for the change in the target heading. The without the proper output on Pod 3, the buoy would be unable to balance the moments from the pods. This would result in the buoy rotating about its vertical axis. Another cause for this rotation could be the fluctuations in the target track window location and size.

6.2 Charles River

In addition to testing in the tow tank, testing was carried out in the Charles River. This was done in order to assess the visibility and determine what depth could be achieved. The sky was heavily overcast and the river had limited visibility of 36". Figures 6-19, 6-20, and 6-21 show the view from the buoy in the Charles River during this testing.

The light intensity is much more important in the turbid river water. With the target set at 8% light, the buoy could still identify the target at a depth of 2ft. Increasing the light to 100%, the buoy was only able to gain another foot of depth.

In addition to the light intensity, the river had a noticeable current. Another buoy was used on the surface to tow the target along at a constant speed. Even at a speed of only 30% of the buoy's max speed (1 m/s) the submerged buoy was only able to follow reliably for short distances, on the order of 5 ft. This could possibly be improved by providing higher power propulsion pods for the buoy. One positive observation was that the buoy would reliably acquire the target again once it was lost.

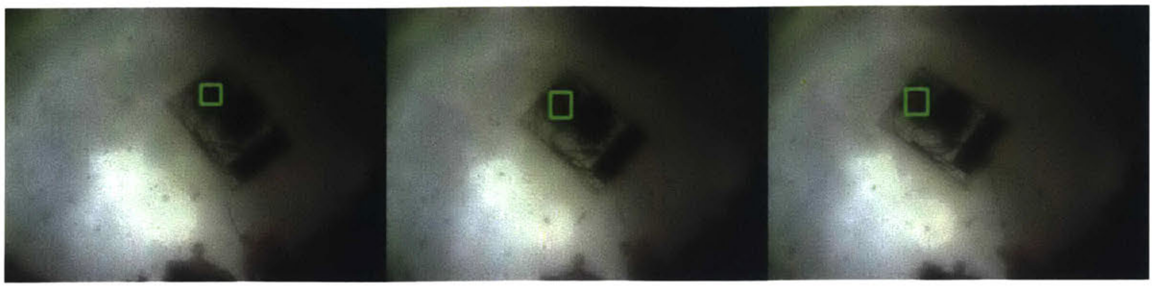


Figure 6-19: A video stream from the buoy while tracking underwater in the Charles River. Each image is 10 frames later than the previous image. Depth of 24" with LED at 8% intensity.

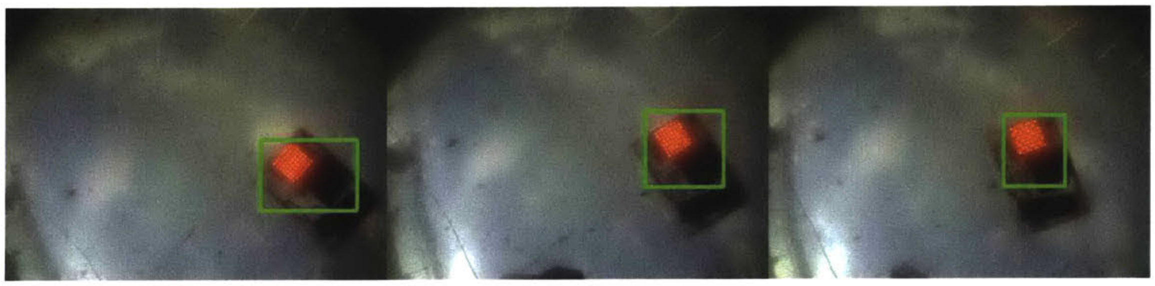


Figure 6-20: A video stream from the buoy while tracking underwater in the Charles River. Each image is 10 frames later than the previous image. Depth of 24" with LED at 100% intensity.

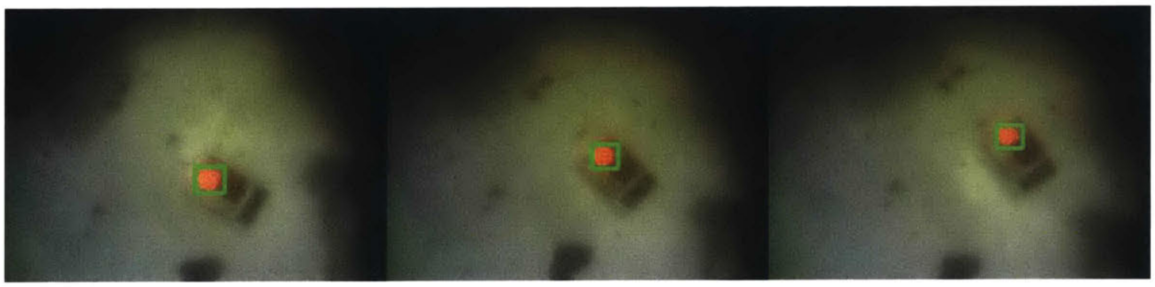


Figure 6-21: A video stream from the buoy while tracking underwater in the Charles River. Each image is 10 frames later than the previous image. Depth of 36" with LED at 100% intensity.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Conclusions

7.1 Project Accomplishments

7.1.1 Distance Experiments

This work showed that tracking can be achieved at distances of 40 feet in clear water using a low cost, USB camera incorporated into the guidance system of an autonomous vehicle. This shows that the technique could be utilized to provide a significant benefit to capability of the buoy swarm in certain scenarios.

At greater distances, camera resolution limited the ability of the tracking algorithms. However, this distance is highly dependant on the water conditions as well as other environmental conditions. Surface conditions are highly variable. Wave motion, ambient lighting conditions, and surface reflections all contribute to the performance of the tracking system. Higher quality cameras and more sophisticated image processing techniques could make this system more robust and possibly increase the range.

7.1.2 Testing Platform

Additionally, a modified buoy was created which incorporates a camera into the navigation system. This camera faces upwards, allowing tracking of a surface target. This underwater platform supports testing of various tracking methods, as well as experimentation involving varying water and environmental conditions and provides a means of data collection. More powerful propulsion would improve this platform and allow for a greater range of testing.

This platform was tested to gain an understanding of the buoy's ability to track targets with oscillating motion at different frequencies. Below a period of 5s the buoy was unable to response to the target motion.

Additionally, the buoy was tested in the Charles River to evaluate its performance. Even in the turbid river water, the buoy could clearly see the target at a depth of 3ft. However, improved propulsion is needed for further testing.

7.2 Areas for Further Work

This topic of optical tracking provides a large number of possibilities for future work. These fall into three broad categories: improving the image acquisition process, improving the image processing and tracking techniques, and improving the robustness of the system.

7.2.1 Improving Image Acquisition

The original camera used for testing had a field of view angle of 30 degrees. This was replaced with a 170 degree fish eye lens in order to increase the area that is able to be seen. This showed an improvement in tracking performance. Another way to accomplish this would be to use multiple cameras. Stitching software is available to allow for combining multiple digital images into one larger image. This would provide a greater understanding of the surroundings, especially in shallower depths. Additionally, utilizing a second processor specifically for the image processing could help speed up the performance of the system.

7.2.2 Improving Image Processing

In turbid water, the image quality is degraded and can hinder the ability for target identification. Image recovery techniques could be explored to improve contrast and color correction. Image processing algorithms have been studied which can dramatically improve the range of visibility in turbid conditions [Schechner and Karpel, 2004].

The Camshift algorithm was utilized as a starting point for developing the vehicle; however, there are a large number of other algorithms that could be explored. Trackers that are available in various versions of OpenCV include: BOOSTING, MIL, KCF, CSRT, MedianFLow, TLD, MOSSE, and GOTURN.

Target acquisition is another area that could benefit from further research. Research into machine learning and neural networks could be explored to provide more robust way of acquiring the target, vice the color detection that is currently employed.

7.2.3 Improving System Robustness

In order to field the system on a seagoing vehicle, the system would likely need to incorporate a method for reestablishing the visual link in the event that the target is obscured or passes from the field of view. An simple acoustic pinging system could provide a way of localizing the target in order to direct the subsurface vehicle back to the correct area.

Another modification that could benefit the system would be the incorporation of a better filter for the navigation inputs, such as a Kalman filter. The Kalman filter is a recursive algorithm that consists of basically two steps, a prediction step and and update step [Musoff and Zarchan, 2009]. A system model is used to take the current state of the system and predict the state at the next time increment. Measurements are then taken and compared to the prediction. These are then combined based on a weighting function to update the model and errors in preparation for the next prediction. Figure 7-1 shows the basic idea.

Adding this ability to collect data below the surface increases the utility of the swarm; however, a way to transmit the data from the underwater vehicle to the

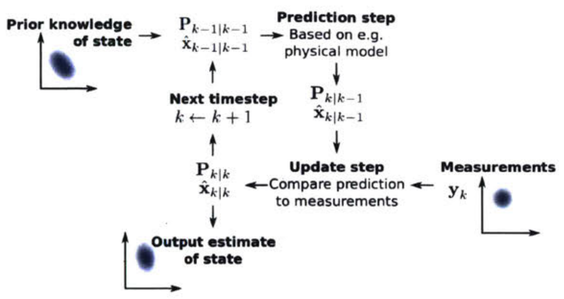


Figure 7-1: A graphical description of how a generic Kalman Filter works.

surface is also needed. Short range methods for data transfer could be explored to improved the system, including acoustic and optical variants. For instance, Sonardyne produces the BlueComm Underwater Optical Communication system; however, this is a highly capable system which does not fit in to the attritable concept of the swarm. Leveraging similar technologies to provide for a low-cost method of data transmission is an important area that needs further study.

Appendix A

Python Code for CAMSHIFT Algorithm

```
#!/usr/bin/env python

'''
Camshift tracker

Cody White 2018-2019
Modified from tutorials by: https://casis.llnl.gov/seminars/opencv

Uses OpenCV camshift tracking algorithm.
Select object with mouse.
This reads from video camera (0 by default)

To initialize tracking, select the object with mouse

Keys:
-----
    ESC   - exit
    b     - toggle back-projected probability visualization
'''

# Python 2/3 compatibility
from __future__ import print_function
import sys
PY3 = sys.version_info[0] == 3

if PY3:
    xrange = range

# import modules needed
import numpy as np
import cv2

# local module
import video
from video import presets


class App(object):
    def __init__(self, video_src):
        self.cam = video.create_capture(video_src, presets['cube'])
        ret, self.frame = self.cam.read()

        # Default resolutions of the frame are obtained.
        #The default resolutions are system dependent.
        # We convert the resolutions from float to integer.
        self.frame_width = int(self.cam.get(3))
        self.frame_height = int(self.cam.get(4))

        cv2.namedWindow('camshift')
        cv2.setMouseCallback('camshift', self.onmouse)

        self.selection = None
        self.drag_start = None
        self.show_backproj = False
        self.track_window = None

    def onmouse(self, event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            self.drag_start = (x, y)
            self.track_window = None
        if self.drag_start:
            xmin = min(x, self.drag_start[0])
            ymin = min(y, self.drag_start[1])
            xmax = max(x, self.drag_start[0])
            ymax = max(y, self.drag_start[1])
            self.selection = (xmin, ymin, xmax, ymax)
        if event == cv2.EVENT_LBUTTONUP:
            self.drag_start = None
            self.track_window = (xmin, ymin, xmax - xmin, ymax - ymin)

    def show_hist(self):
        bin_count = self.hist.shape[0]
        bin_w = 24
        img = np.zeros((256, bin_count*bin_w, 3), np.uint8)
        for i in xrange(bin_count):
            h = int(self.hist[i])
            cv2.rectangle(img, (i*bin_w+2, 255), ((i+1)*bin_w-2, 255-h),
                          (int(180.0*i/bin_count), 255, 255), -1)
        img = cv2.cvtColor(img, cv2.COLOR_HSV2BGR)
        cv2.imshow('hist', img)

    def run(self):
        # Define the codec and create VideoWriter object.
        #The output is stored in 'outpy.avi' file.
        out = cv2.VideoWriter('codyvid.avi',
                              cv2.VideoWriter_fourcc('M','J','P','G'),
                              30, (self.frame_width,self.frame_height))
        out2 = cv2.VideoWriter('codyvid2.avi',
                               cv2.VideoWriter_fourcc('M','J','P','G'),
                               30, (self.frame_width,self.frame_height))

        while True:
            ret, self.frame = self.cam.read()
            vis = self.frame.copy()
            hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)
            mask = cv2.inRange(hsv, np.array((0., 60., 32.)),
                               np.array((180., 255., 255.)))

            if self.selection:
                x0, y0, x1, y1 = self.selection
                hsv_roi = hsv[y0:y1, x0:x1]
                mask_roi = mask[y0:y1, x0:x1]
                hist = cv2.calcHist([hsv_roi],[0],mask_roi,[16],[0,180])
                cv2.normalize(hist, hist, 0, 255, cv2.NORM_MINMAX)
                self.hist = hist.reshape(-1)
                self.show_hist()
                vis_roi = vis[y0:y1, x0:x1]
                cv2.bitwise_not(vis_roi, vis_roi)
                vis[mask == 0] = 0

            if self.track_window and self.track_window[2] > 0
                                 and self.track_window[3] > 0:
                self.selection = None
                prob = cv2.calcBackProject([hsv],[0],self.hist,[0,180],1)
                prob &= mask
                term_crit = ( cv2.TERM_CRITERIA_EPS |
                        cv2.TERM_CRITERIA_COUNT, 10, 1 )
                track_box, self.track_window =
                    cv2.CamShift(prob, self.track_window, term_crit)

                if self.show_backproj:
                    vis[:] = prob[...,np.newaxis]
                try:
                    cv2.ellipse(vis, track_box, (0, 0, 255), 2)
                except:
                    print(track_box)
            cv2.imshow('camshift', vis)

            # Write the frame into the file 'output.avi'
            out.write(self.frame)
            out2.write(vis)

            ch = 0xFF & cv2.waitKey(5)
            if ch == 27:
                break
            if ch == ord('b'):
                self.show_backproj = not self.show_backproj
        cv2.destroyAllWindows()

if __name__ == '__main__':
    import sys
    try:
        video_src = sys.argv[1]
    except:
        video_src = 0
    print(__doc__)
    App(video_src).run()
```

Appendix B

Ardruino Code for NeoPixel Array

```
\*
Code to drive 8x8 Neopixel array with Arduino
Cody White
2018-2019
*/

// Import FastLED package to drive the Neopixel array
#include <FastLED.h>

// How many leds in your strip?
#define NUM_LEDS 64

// Define output pin on Arduino that will be PWM pin to NEOPIXEL
#define DATA_PIN 6

// Define the array of leds as RGB format
CRGB leds[NUM_LEDS];

// Build a pattern of 4 LEDs in each corner of the array
int pattern1[] = {0,1,6,7,8,9,14,15,48,49,54,55,56,57,62,63};
```

```
void setup() {
  //
  FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);

  // Make all the LEDs be off to start with
  for (int i = 0; i < 64; i++){
     leds[i] = CRGB::Black;
    }

  // Set up the pattern for all 64 LEDs and assign color/brightness
  for (int i = 0; i < 64; i++){
    leds[i] = CRGB(0,0,255);
    }

  // Set up pattern and assign brightness for 4 corner pattern if used
//  for( int i = 0; i < 16; i++) {
//    leds[pattern1[i]] = CRGB::White;
//    }
}

void loop() {
  // Turn on the LEDs in the pattern built above
  FastLED.show();
}
```

Appendix C

Lid Calculations

Since I decided to sink a buoy on a set of wheels, I wanted to verify the lid would not break while submerged. In order the buoy watertight I want to replace the current lid with one without all the various penetrations for the plugs and antennas. This minimized the potential for leaks. Additionally, the original lid was relatively thin. The original lid was made out of acrylic and there was some concern of brittleness as it was several years old. A new lid was made using acrylic with a thickness of 12 mm.

The strength of the new lid was calculated to ensure it wouldn't break due to water pressure at depth. This was done using plate theory [Ventsel and Krauthammer, 2001]. The lid is a circular plate under axially symmetric loading. A solid plate a radius a, subjected to a uniform load, is governed by the differential equation given in Equation C.1.

$$\frac{1}{r}\frac{d}{dr}\left\{r\frac{d}{dr}\left[\frac{1}{r}\frac{d}{dr}\left(r\frac{dw}{dr}\right)\right]\right\} = \frac{p}{D} \tag{C.1}$$

D is the flexural rigidity of the plate. It is a function of plate thickness, modulus of elasticity and Poisson ratio of the material. It is given by:

$$D = \frac{Eh^3}{12(1-\nu^2)} \tag{C.2}$$

This equation can be solve to yield the deflection w, which is a combination of the particular and homogeneous solutions shown in the following equations:

$$w_h = C_1 ln(r) + C_2 r^2 ln(r) + C_3 r^2 + C_4 \tag{C.3}$$

$$w_p = \frac{p_0 r^4}{64D} \tag{C.4}$$

The lid for the buoy is clamped by a ring of bolts which leads to the following boundary conditions:

$$w = 0|_{r=a} \quad \text{and} \quad \frac{dw}{dr} = 0|_{r=a} \tag{C.5}$$

Using the boundary conditions to solve yields the following for the maximum deflection, which obviously occurs and the center of the lid:

$$w_{max} = \frac{p_0 a^4}{64D} \tag{C.6}$$

The real concern though is the maximum bending moments which lead to stresses in the material. The maximum bending moment in the radial direction occurs at the outer radius and the maximum bending moment in the tangential direction is at the center. Both are shown below, along with the equations to determine the maximum stress, which occurs and the surfaces of the lid.

$$M_{rmax} = -\frac{p_0 a^2}{8} \tag{C.7}$$

$$M_{tmax} = \frac{p_0 a^2 (1+\nu)}{16} \tag{C.8}$$

$$\sigma_r = \frac{12 M_r}{h^3} z \qquad \sigma_t = \frac{12 M_t}{h^3} z \tag{C.9}$$

The calculations were carried out using MATLAB as shown below. The new lid should withstand a depth of 50 meters easily. Even applying a safety factor of 2, it will handle the depths that I expect testing to occur at.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculations to determine stress in buoy lid
% Cody White
% 2018-2019
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Problem Setup
clear all
clc
% Modulus of Elasticity
E = 2.76e9;
% Poisson ratio
v = 0.3;
% Thickness of lid
h = 0.012;
% Radius of lid
a = 0.15;
% Density of water (kg/m^3)
rho = 1000;
% Gravitational Constant
g = 9.8;
% Outer surface of lid
z = h/2;
% Flexural rigidity
D = E*h^3/(12*(1-v^2));

%% Loop through various depths
s = 100
d = linspace(0,s,s);

for i = 1:s
```

```
    % Calculate the pressure at this depth
    p = rho*g*d(i);
    % Calculate maximum radial bending moment
    Mr_max(i) = p*a^2/(8);
    % Calculate maximum tangential bending moment
    Mt_max(i) = p*a^2*(1+v)/16;
    % Calculate radial stress at surface of lid
    sig_r(i) = 12*Mr_max(i)/h^3 * z /(10^6);
    % Calculate tangential stress at surface of lid
    sig_t(i) = 12*Mt_max(i)/h^3 * z /(10^6);
end

figure
plot(d,sig_r,d,sig_t,[0,100],[60,60])
xlabel('Depth (m)')
ylabel('Stress (MPa)')
title('Stress in Lid of Submerged Buoy')
legend('Stress in Radial Direction', 'Stress in Tangential Direction',...
'Material Limit for Acrylic','Location','best')
```

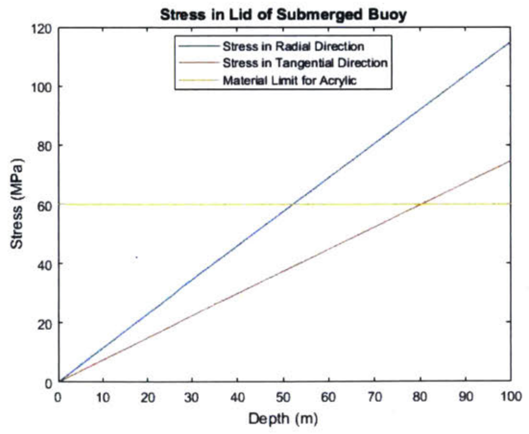


Figure C-1: Stresses in the lid show it should withstand a depth of 50 meters.

Appendix D

Updating the Beaglebone Black

The new BBB had an old image from 2014. This resulted in a number of the repositories being no longer supported, so updating would not work smoothly. To solve this, the new BBB was updated with a new Debian 9.5 image, the 2018-10-07 Stretch version. After downloading the new image, Win32DiskImager was used to flash an SD card with the new image. This could then be inserted into the BBB and used to boot the system.

Following this, the BBB was connected to the internet. Utilizing the ethernet port proved more successful, and more simple, than using the USB through a laptop. Some guidance for this can be found at the following link:
http://ofitselfso.com/Beagle/NetworkingSetupConnectingTheBeagleboneBlack.php

D.1 Updating the Kernel and Packages

Once the BBB was Then, the kernel was updated. Guidance for this can be found at:
https://elinux.org/Beagleboard:BeagleBoneBlack_Debian
This was done using the following commands:

```
cd /opt/scripts/tools/
git pull
sudo ./update_kernel.sh
sudo reboot
```

Even though the image was relatively new, there were still a large number of packages that required updating. Once this was completed, the python setup could be installed, along with the pip installer. This was completed using:

```
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get install build-essential python-dev -y
sudo apt-get install python-setuptools python-pip python-symbus -y
sudo pip install Adafruit_BBIO
```

Before the necessary libraries could be added, the memory was repartitioned to provide more room for the archives. This can be done using:

```
cd /opt/scripts/tools/
git pull || true
sudo ./grow_partition.sh
sudo reboot
```

D.2 Adding Python Libraries

The following packages were installed. This was done using either 1) sudo pip install [packagename], or 2) sudo apt-get install [packagename].

```
numpy
scipy
pynmea2
matplotlib
```

The imutils package was also installed. It required the following command:

```
sudo pip install –index-url https://pypi.python.org/simple/ imutils
```

D.3 Useful Commands for Copying Files

Copy Directory from Computer to BBB:

```
scp -r ./Directory/ root@192.168.7.2:/home/debian/Python
```

Copy File from Computer to BBB:

```
scp *.py root@192.168.7.2:/home/debian/Python
```

Copy from BBB to Computer:

```
scp -r root@192.168.7.2:/home/debian/Python/Output ./Directory
```

While these can be used to transfer files, WinSCP provides a more user friendly approach because it allows the file structures to be seen from both systems. I found it easier than the Linux command line.

D.4 Setting up the BBB to Run

After putting the necessary buoy code into the /home/debian/Python directory, the system needed to be told to initialize this code on startup. This was done by making a file to start the scripts:

```
nano /lib/systemd/system/Buoy.service
```

File Contents:

```
[Unit]
Description=Buoy Init Service
```

```
[Service]
WorkingDirectory=/home/debian/Python/
ExecStart=/usr/bin/python Buoy_Main.py
StandardOutput=null
Restart=always (or on-failure)
RestartSec=30

[Install]
WantedBy=multi-user.target
Alias=Buoy.service
```

Enable the service:

```
sudo systemctl enable Buoy.service
```

verfiy by:

```
sudo systemctl start Buoy.service
```

Stop the running script by determining it's number and killing:

```
top (read the PID number from list)
kill [PID number]
```

Appendix E

Camshift Tracker Buoy Code

The the majority of the buoy was driven by code presented in [Zoss, 2016]. This file contains the code that generated the heading and speed needed based on the tracking algorithm. This code was integrated into the existing buoy to provide navigation.

```
# -*- coding: utf-8 -*-
"""
Camshift tracker
Cody White 2018-2019
Uses OpenCV camshift tracking algorithm.
This reads from video camera (0 by default)
Done using Python 2.7
"""
# Import the necessary modules
import numpy as np
import cv2
from threading import Thread
import imutils
import time
from os.path import exists

class Camshift(Thread,object):
    def __init__(self, video_src = 0):
        Thread.__init__(self)
        self.video_src = video_src

    def _start_thread(self, name):
        if name == 'main':
            self.main_loop = Thread(target=self.run)
            self.main_loop.daemon = True
            self.main_loop.start()
        print 'Camera Thread'

    ##################################################################
    ############    Initialize values            ###################
    ##################################################################
    def reset(self):
        self.frame_count = 0
        self.initBB = None
        self.angle = 0
        self.Thrust = 0
        self.lastx = 0
        self.lasty = 0
        self.d_x = 0
        self.d_y = 0
        self.x = 0
        self.y = 0

    def pack_Data(self):
        """
        Assemble Data for packing into CamBuoyOUT file
        Line Structure:KP,KI,KD,ST,Tracked, Time,X,Y,Angle,Thrust
        """
        s = []
        s.append(self.KP)
        s.append(self.KI)
        s.append(self.KD)
        s.append(self.sampletime)

        if self.initBB == None:
            s.append(0)
        else:
            s.append(1)
        seconds = time.time()-self.time_started
        s.append(round(seconds,3))
        s.append(round(self.x,3))
        s.append(round(self.y,3))
        s.append(self.angle)
        s.append(self.Thrust)
        """
        Join all list data into a single comma
        separated data string.
        """
        s = ','.join(map(str, s)) + '\n'
        print('this is s from pack',s)
        return s

    def write_Data(self):
        """
        Pack and write Buoy data into BuoyOUT file
        """
        data_output=open(self.data_label[0:-4]+
                         '_'+str(self.file)+'.txt','a+')
        string = self.pack_Data()
        data_output.write(string)
        data_output.flush()
        data_output.close()
        if self.line >= 200:
            self.line = 0
            self.file +=1
        else:
            self.line +=1

    def init_dataandvid(self):
        self.time_started = time.time()
        try:
            VidFileString =
                "/home/debian/Python/Video/CamBuoyOUT00.avi"
            DataFileString =
                "/home/debian/Python/TrackData/CamBuoyOUT00.txt"
            dataChar = list(DataFileString)
            vidChar = list(VidFileString)
            for m in range(0, 100):
                dataChar[-6] = str(int(m / 10))
                dataChar[-5] = str(int(m % 10))
                vidChar[-6] = str(int(m / 10))
                vidChar[-5] = str(int(m % 10))

                self.vid_label = ''.join(vidChar)
                self.data_label = ''.join(dataChar)
                if (exists(self.data_label) == False):
                    break
        except:
            pass

        try:
            ##############Commented out to stop vid
            #label = str(self.vid_label)
            #self.out = cv2.VideoWriter(label,cv2.cv.CV_FOURCC
        #                    ('M','J','P','G'), 10,
        #                     (self.frame_width,self.frame_height))
        #######################
        self.started_writer = True
        holder = open(self.data_label, 'w')
        holer.close()

    except:
        print 'Failed to start Video Writer'

####################################################################
##############    PID Functions      ##############################
####################################################################
def set_PID(self):
   # Compensates the controller for new K values, or sample time
   if self.sampletime > 0:
       self.KP = self.KP
       self.ki_t = self.KI * self.sampletime
       self.kd_t = self.KD / self.sampletime

def control_calc(self):
    # Find the time since the last PID calc
    now = time.time()
    timechange = now - self.lastTime
    # Get the pixel location from the tracker
    (self.pix_x, self.pix_y), junk, junk = self.track_box
    self.pix_x = self.pix_x - self.frame_width/2
    self.pix_y = -(self.pix_y - self.frame_height/2)
    # Determine % across the frame
    self.x = self.pix_x/(self.frame_height/2)*100
   # print('x',self.x)
    self.y = self.pix_y/(self.frame_height/2)*100
   # print('y', self.y)
    # If the time since last is larger than sample time do calcs
    if timechange >= self.sampletime:
        # Error is difference between setpoint and input.
        # Input is buoy position, (0,0) and the setpoint ends up
        # being the target position.

        # Calulate the integral term
        self.ITerm_x = self.ki_t * self.x
        self.ITerm_y = self.ki_t * self.y
        # Cap the integral term
        if self.ITerm_x >100: self.ITerm_x = 100
        if self.ITerm_x < -100: self.ITerm_x = -100
        if self.ITerm_y >100: self.ITerm_y = 100
        if self.ITerm_y < -100: self.ITerm_y = -100
        # Determine the derivative term
        self.d_x = self.x - self.lastx
        self.d_y = self.y - self.lasty
        #print('p term ',self.KP*self.x)
        #print('i term ', self.ITerm_x)
        #print('d term', self.kd_t*self.d_x)
        # Calculate the thrust for the x and y directions
        self.Tx = self.KP*self.x + self.ITerm_x + self.kd_t*self.d_x
        self.Ty = self.KP*self.y + self.ITerm_y + self.kd_t*self.d_y
        # Calculate the net thrust and cap at 100%
        self.Thrust = int(np.sqrt(self.Tx**2 + self.Ty**2))
        #print('thrust',self.Thrust)
        if self.Thrust > 100: self.Thrust = 100

        # Determine the heading to
        angle = np.arctan2(self.Ty,self.Tx)*180/np.pi
        if angle >= 0:
            self.angle = round(angle)
        else:
            self.angle = round(angle + 360)

        # Store x and y for next round
        self.lastx = self.x
        self.lasty = self.y
        self.lastTime = now

################################################################
###########     Main Loop      ##################################
################################################################

def run(self):
    print 'Starting Run Function of Camera'
    # Create a capture from the video source
    self.cam = cv2.VideoCapture(self.video_src)

    # Set resolution to support BBB USB ops
    self.cam.set(3,320); self.cam.set(4,240)

    # Find the resolution of the camera
    self.frame_width = int(self.cam.get(3))
    self.frame_height = int(self.cam.get(4))

    # Create a window to display the video
    #cv2.namedWindow('Camshift')

    ########### CamShift Section ###########
    self.reset()
    ########### Propulsion Parameters ###########
    self.sampletime = .2
    self.KP = 1
    self.KI = 0
    self.KD = 0
    self.ki_t = self.KI * self.sampletime
    self.kd_t = self.KD / self.sampletime
    self.lastTime = time.time()
    ########### Video and Data Recording ###########
    self.write = False
    self.started_writer = False
    self.line = 0
    self.file = 0

    ###########  Import Target   ####################
    self.target_picture = cv2.imread('messigray.png')
    #cv2.imshow('initial',self.target_picture)
    hsv = cv2.cvtColor(self.target_picture, cv2.COLOR_BGR2HSV)
    #cv2.imshow('target',hsv)
```

```
self.hist = cv2.calcHist([hsv], [0], None, [16], [0, 180])
#self.hist=cv2.calcHist([hsv],[0,1],None,[16,2],[0,180,0,256])

cv2.normalize(self.hist, self.hist, 0, 255, cv2.NORM_MINMAX)
###############################################################
#############  Start the Loop   ###############################
###############################################################
while True:
    # capture the next frame of the video
    if self.write == True and self.started_writer == False:
        try:
            self.init_dataandvid()
        except:
            print 'Data not Started'
    grabbed, self.frame = self.cam.read()
    # if we are viewing a video and we did not grab a frame,
    # then we have reached the end of the video
    if self.frame is None:
        break
    else:
        self.frame2 = self.frame.copy()

    # resize the frame, blur it, and convert it to the HSV
    # color space prior to finding contours
    self.blurred = cv2.GaussianBlur(self.frame, (11, 11), 0)
    self.hsv = cv2.cvtColor(self.blurred, cv2.COLOR_BGR2HSV)
    self.limit = cv2.inRange(self.hsv, np.array((0., 60., 32.)),
                             np.array((180., 255., 255.)))
    self.mask = cv2.calcBackProject([self.hsv], [0], self.hist,
                                    [0, 180], 1)

    if self.frame_count < 20:
        self.frame_count += 1

        self.mask = cv2.erode(self.mask, None, iterations=2)
        self.mask = cv2.dilate(self.mask, None, iterations=2)
        # find contours in the mask and initialize the current
        # (x, y) center of the ball
        self.cnts = cv2.findContours(self.mask.copy(),
                        cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        self.cnts = imutils.grab_contours(self.cnts)
        self.center = None

        self.vis = self.frame.copy()
        #print('contours',self.cnts)

        # only proceed if at least one contour was found
        if len(self.cnts) > 0:
            # find the largest contour in the mask, then use
            # it to compute the minimum enclosing circle and
            # centroid
            self.c = max(self.cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(self.c)
            self.track_window = cv2.boundingRect(self.c)
            M = cv2.moments(self.c)
            self.center = (int(M["m10"] / M["m00"]),
                           int(M["m01"] / M["m00"]))
            # only proceed if the radius meets a minimum size
            if radius > 10:
                # draw the circle and centroid on the frame,
                # then update the list of tracked points
                (x, y, w, h) = [int(v) for v in self.track_window]
                cv2.rectangle(self.vis, (x, y), (x + w, y + h),
                        (0, 255, 0), 2)

    else:
        self.initBB = self.track_window

        if self.initBB is not None:

            self.vis = self.frame.copy()
            #self.mask = cv2.calcBackProject([self.hsv], [0],
            #     self.hist, [0, 180], 1)
            term_crit = (cv2.TERM_CRITERIA_EPS |
                    cv2.TERM_CRITERIA_COUNT, 10, 1)
            self.mask &= self.limit
            if (self.track_window[2] <= 0 or
                self.track_window[3]<= 0 or
                self.mask.all() == None):
                print('Lost Track')
                self.reset()
            try:
                self.track_box, self.track_window = cv2.CamShift(
                        self.mask, self.track_window, term_crit)
                self.initBB = self.track_window
            except:
                print'back to acq from camshift part1'
                self.reset()
                self.track_box=[(self.frame_width/2,
                                 self.frame_height/2)]
                pass
            try:
                (x, y, w, h) = [int(v) for v in self.track_window]
                cv2.rectangle(self.vis, (x, y), (x + w, y + h),
                              (0, 255, 0), 2)
            except:
                print('exception',self.track_box)

    ############################################################
    ################### Determine Propulsion Controls ##########
    ############################################################
            if self.initBB is not None:
                self.control_calc()
            else:
                self.angle = 0
                self.Thrust = 0
            print ('Thrust',self.Thrust, 'angle',self.angle)

            info = [("KP: ", self.KP),
                    ("KI: ", self.KI),
                    ("KD: ", self.KD),
                    ("ST: ", self.sampletime)]
    ############################################################
```

```
            #################  Check and try to write video###########
            ##########################################################
            if self.write == True and self.started_writer == True:
                ###############Commented out to stop vid
#                output = self.vis.copy()
#                for (i, (k, v)) in enumerate(info):
#                        text = "{}: {}".format(k, v)
#                        cv2.putText(output,text,
#                                    (10, self.frame_height-((i*20)+20)),
#                                    cv2.FONT_HERSHEY_SIMPLEX,
#                                    0.4, (0, 0, 255), 2)
                try:
                    #self.out.write(output)
                    self.write_Data()
                except:
                    pass

            elif self.write == True and self.started_writer == False:
                print 'No writer available'

            elif self.write == False and self.started_writer == True:
                try:
                    #self.out.release()
                    self.started_writer = False
                    self.line = 0
                    self.file = 0
                    print'Released Video Writer'

                    print'Exported Data'
                    #break
                except:
                    try:
                        #label = './Video/z_junkvid.avi'
                        #self.out=cv2.VideoWriter(label,cv2.cv.CV_FOURCC
                        #        ('M','J','P','G'), 10,
                        #        (self.frame_width,self.frame_height))
                        self.started_writer = False

                    except:
                        pass
                    print'Failed to release Video Writer'
                    pass
            else:
                pass

if __name__ == '__main__':
    # Set the default video source for the camera input
    video_src = 0
    # Start the Tracker
    Camshift(video_src).run()
```

Bibliography

[Allen et al., 2004] Allen, J. G., Xu, R. Y., and Jin, J. S. (2004). Object tracking using camshift algorithm and multiple quantized feature spaces. In *Proceedings of the Pan-Sydney area workshop on Visual information processing*, pages 3–7. Australian Computer Society, Inc.

[Allen Institute for Artificial Intelligence,] Allen Institute for Artificial Intelligence. Ocean Zones. `http://data.allenai.org/tqa/ocean_zones_L_0258/`. Online; accessed 4-February-2019.

[Bolme et al., 2010] Bolme, D. S., Beveridge, J. R., Draper, B. A., and Lui, Y. M. (2010). Visual object tracking using adaptive correlation filters. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550.

[Colombi et al., 2017] Colombi, J., Bentz, B., Recker, R., Lucas, B., and Freels, J. (2017). Attritable design trades: Reliability and cost implications for unmanned aircraft. In *11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings.*

[Duarte et al., 2016] Duarte, M., Costa, V., Gomes, J., Rodrigues, T., Silva, F., Oliveira, S. M., and Christensen, A. L. (2016). Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLoS ONE.*

[Exner et al., 2010] Exner, D., Bruns, E., Kurz, D., GrundhÃűfer, A., and Bimber, O. (2010). Fast and robust camshift tracking. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, pages 9–16.

[Jerlov, 1976] Jerlov, N. (1976). Marine Optics.

[Leonard et al., 2007] Leonard, N. E., Paley, D. A., Lekien, F., Sepulchre, R., Fratantoni, D. M., and Davis, R. E. (2007). Collective motion, sensor networks, and ocean sampling. *Proceedings of the IEEE.*

[Manley, 2008] Manley, J. E. (2008). Unmanned surface vehicles, 15 years of development. In *OCEANS 2008.*

[Musoff and Zarchan, 2009] Musoff, H. and Zarchan, P. (2009). *Fundamentals of Kalman Filtering: A Practical Approach, Third Edition.* American Institute of Aeronautics and Astronautics.

[P Bukata et al., 1995] P Bukata, R., H Jerome, K., Kondratyev, K., and V Podzdnyakov, D. (1995). *Optical Properties and Remote Sensing of Inland and Coastal Waters.*

[Sakshaug and Slagstad, 1991] Sakshaug, E. and Slagstad, D. (1991). Light and productivity of phytoplankton in polar marine ecosystems: a physiological view. *Polar Research*, 10(1):69–86.

[Salhi and Yengui Jammoussi, 2012] Salhi, A. and Yengui Jammoussi, A. (2012). Object tracking system using Camshift, Meanshift and Kalman filter. Technical report.

[Schechner and Karpel, 2004] Schechner, Y. Y. and Karpel, N. (2004). Clear Underwater Vision. Technical report.

[Stutters et al., 2008] Stutters, L., Liu, H., Tiltman, C., and Brown, D. J. (2008). Navigation technologies for autonomous underwater vehicles. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews.*

[Tournadre, 2014] Tournadre, J. (2014). Anthropogenic pressure on the open ocean: The growth of ship traffic revealed by altimeter data analysis. *Geophysical Research Letters.*

[Ventsel and Krauthammer, 2001] Ventsel, E. and Krauthammer, T. (2001). *Thin Plates and Shells: Theory, Analysis, and Applications.*

[Zoss, 2016] Zoss, B. M. (2016). Design and Analysis of Mobile Sensing Systems an Environmental Data Collection Swarm. Master's thesis, Massachusetts Institute of Technology.

[Zoss et al., 2018] Zoss, B. M., Mateo, D., Kuan, Y. K., Tokić, G., Chamanbaz, M., Goh, L., Vallegra, F., Bouffanais, R., and Yue, D. K. (2018). Distributed system of autonomous buoys for scalable deployment and monitoring of large waterbodies.