Introduction to Engineering Design with Professional Development 1

Final Report for SABBL-Lite

Semi-Autonomous Black Box Locator - Lite

Team: Non_Ame SRG

Section 3

Instructors: Robert Niemiec, Graham Knowles

Version 3.5

December 12, 2018

Prepared by

Andrew Chafy 2021 / Mechanical Engineering

Jonathan Beck 2021 / Aerospace Engineering

Jack Ottinger 2021 / Aerospace Engineering

Adam Li 2021 / Mechanical and Electrical Engineering

Justin Krasinski 2018 / Aerospace Engineering

William Parker 2020 / Aerospace Engineering

Executive Summary

The SABBL-Lite is a semi-autonomous underwater vehicle whose purpose is to locate the flight data recorder from downed aircraft, then relay its position back to a larger search force for retrieval. The project addresses the current issue of extremely slow, expensive, and outdated black box search methods that unnecessarily put human lives in possible danger. By integrating autonomous underwater vehicles (AUVs) into search parties for downed aircraft, the Non_Ame team asserts that searches will be faster, cheaper, and more effective.

The SABBL-Lite is a proof of concept for the use of AUVs in black box searches. The system is semi-autonomous, meaning it can be either be autonomous or controlled manually. It is designed to move in an underwater environment efficiently, locate a black box based on its emitted signal, and subsequently retrieve this black box.

The SABBL-Lite system is broken into six separate subsystems. These subsystems center on the frame, communications, navigation, guidance, propulsion, and object retrieval. During the course of the project, each individual subsystem experienced successes and failures, leading to each subsystem experiencing varying levels of success in integration onto the overall SABBL-Lite vehicle.

Following extensive design, development, and testing, the system was able to move through the water effectively and retrieve a black box. The black box was able to emit sound and be submerged without issue. The guidance sensors, while not integrated, could hear the black box and distinguish the black box unique frequency. The navigation sensors, while not integrated fully, could effectively read the depth of the SABBL-Lite system and give commands that would regulate depth.

Overall, the SABBL-Lite proves that it is possible for an AUV to locate and retrieve a black box. By using an AUV like the SABBL-Lite in actual operations, black box search teams will more effectively carry out their missions. Investigations into accidents only made possible by the AUV contribution to black box searches will save lives by keeping the future of air travel safe.

Table of Contents

1 Introduction

When aircraft and oceangoing ships are involved in accidents, it is only by a thorough investigation that potential dangers can be addressed to prevent future failures. Catastrophic failure in these vehicles represents a unique problem. In the absence of a well-defined crash site, investigators turn to a vessel's flight data recorder (or voyage data recorder for ships), often called its "black box," to determine the cause of failure. Flight data recorders emit high-frequency pings for approximately thirty days when submerged, and the objective of search teams is to locate the source of these pings, thereby locating the black box and some of the wreckage from the crash.

Current methods for determining the location of black boxes are inefficient and costly. The current state of the art system, as demonstrated in Figure 1, uses hydrophones towed by large ships and submarines. Ships follow a predetermined search pattern and use the hydrophone array to determine the amplitude of the pings at a given location.

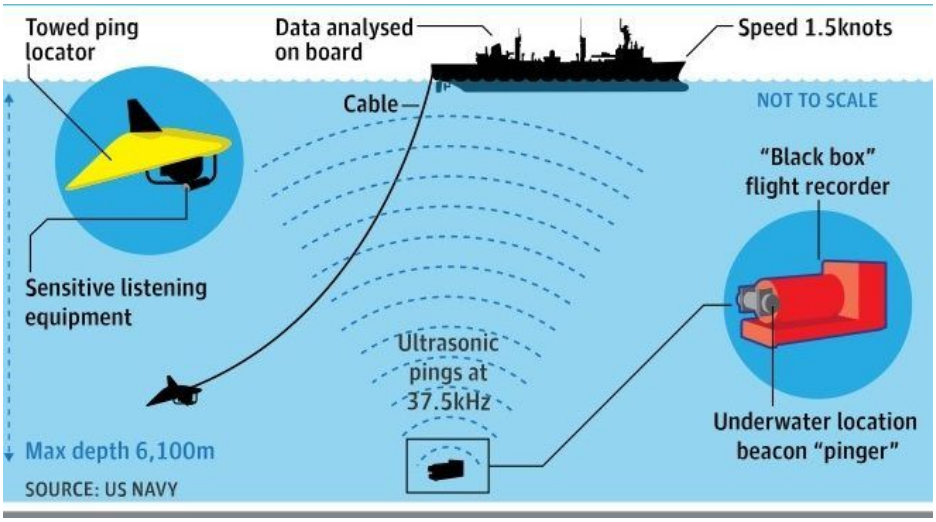


Figure 1. Current state of the art black box search method [1].

For search efforts like the one for Air France Flight 447, this search method was insufficient, and the black box stopped emitting its signal before search teams could determine its location. Flight 447 was eventually recovered years later, with considerably more effort and funding than would have been required if the most had been made of the thirty days that the black box was actively sending out signals.

Autonomous underwater vehicles (AUVs) have been used for decades in a variety of applications including seafloor mapping, mine countermeasures, anti-submarine warfare, and archeology. Leading AUV manufacturers like Hydroid, Bluefin Robotics, and BAE Systems have developed modular adaptable systems which can be used in each of these applications.

Over time, as the enabling technology has matured, an increasing number of applications for AUVs has become apparent. These vehicles have proven extremely effective in making ocean-based data collection more efficient and economically palatable. Building on these successes, an AUV designed specifically to aid in black box search efforts will serve to significantly decrease black box recovery time. The Semi-Autonomous Black Box Locator - Lite (SABBL-Lite) is a proof of concept, built to demonstrate the potential effectiveness of autonomous vehicles in marine search efforts. By improving black box locating and recovery success, potential aircraft and ocean vessel dangers will be discovered and addressed before they cause further failure and loss of life.

2 Project Objectives and Scope

2.1 Team Mission Statement

To build devices that are effective, efficient, and applicable to search situations by automating tasks normally requiring large amounts of manpower and human risk.

2.2 Customer Requirements

The following customer requirements, compiled in Table 1, represent the critical needs for the proposed SABBL-Lite concept.

Table 1. Customer Requirements

Requirement
Buoyant
Can mount all subsystems
Can lift black box
Can move in xy-plane and independently on the z-axis
Can "hear" pings
Balance
Must be able to determine and maintain depth
Must be able to roughly determine self-location/direction
Responsive
Waterproof

2.3 Technical Requirements

The following technical requirements compiled in Table 2 are necessary for the SABBL-Lite to address the problems associated with current black box search and recovery methods.

Table 2. Breakdown of Prioritized Requirements and Associated Specifications by Subsystem

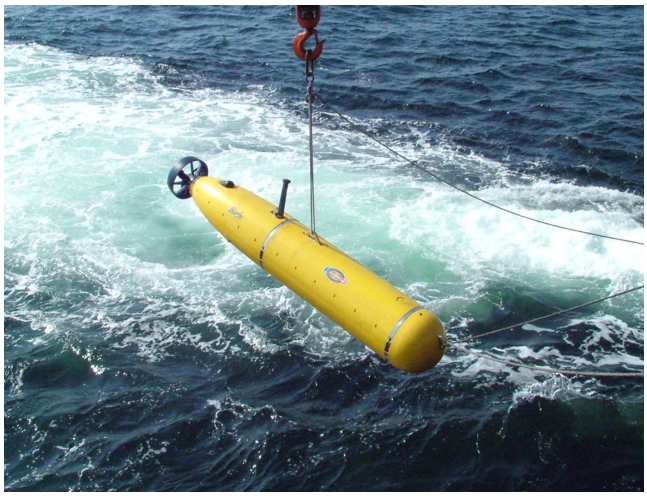
Subsystem	Requirement	Specification
Frame	Mounting Space	Subsystems mounted on frame: 6
	Balance	Frame list in water < + - 5 degrees
	Waterproof Section	Waterproof section allows < 1 oz of water to enter at up to 5 m underwater
	Buoyancy	System has density < 997 kg/m^3
Communications	Communication Latency	Communication latency of <= 100ms
	Bandwidth	Demonstrated bandwidth >= 25kbps
Navigation	Determination of Depth	Depth measurement accurate +/- 1.5’
	Regulation of Depth	Able to maintain a depth of 5’ +/- 1.5’
Guidance	Can “Hear” Pings	Able to hear the black box within 1.5 meters
Propulsion	Can Propel System in 3D Space	Can operate on the z-axis, independent of xy-plane
	Can Lift Black Box	Can provide enough thrust to pick up black box. (total vertical thrust > weight vehicle + weight black box)
Object Retrieval	Generates Audio	Noise produced is audible to humans (20 hz to 20 khz)
	Waterproof	Waterproof at 3 m depth

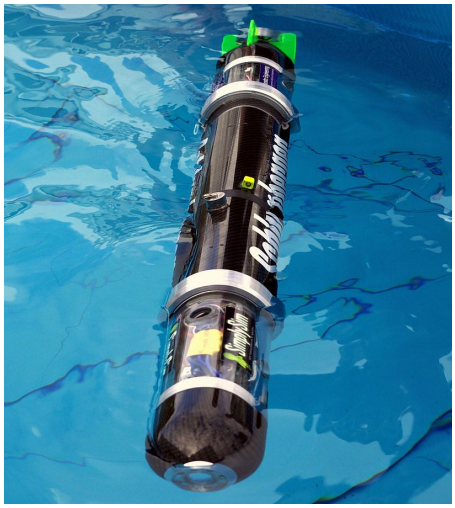
3 Assessment of Relevant Existing Technologies

Other existing AUV’s were researched and benchmarked before beginning their process of designing the SABBL-Lite. Table 3 details the various products reviewed. The existing AUV’s benchmarked in Table 3 were used by the Non_Ame team as inspiration for the possible applications of AUV’s, as well as to see where the gaps in AUV usage were. Additionally, the sensor arrays used in the benchmarked vehicles were examined for use in the SABBL-Lite, and though many of the sensors were far too expensive or were simply impractical for use in the SABBL-Lite, they provided a starting point for possible sensor types to be used in the project to solve various subsystem problems.

Table 3. Benchmarking [2]

Competitive Product	**Title/Description**	**Relation to the Project**
Bluefin Robotics BPAUV	Battlespace Preparation Autonomous Underwater Vehicle	The BPAUV (figure 2 b.) is an example of the successful employment of AUVs in the real world. Used by the U.S. Navy, the BPAUV demonstrates the independent mapping capability of AUV's in a combat environment and the ability for AUV's to be useful to a larger force at set [2].
Blackghost AUV	Undertakes an assault course without any outside control or intervention	The Blackghost AUV (figure 2 a.) demonstrates the ability for AUV's to perform complex tasks, like taking an underwater assault course to a target, completely autonomously. Additionally, the Blackghost AUV again provided an example of a larger at sea force benefitting from using AUV's [2].
Pluto Plus UUV	Norwegian minehunter	The Pluto Plus UUV (figure 2 c.) gives an example of an unmanned underwater vehicle being able to locate another object underwater and give that object's location back to a larger at sea force. The Pluto Plus also proves that there is a market for UUVs and AUVs on the world scale [2].

a) 

b) 

c) 

Figure 2. Various AUVs [2]. **a)** Blackghost AUV. **b)** Bluefin Robotics BPAUV. **c)** Pluto Plus UUV.

4 Professional and Societal Considerations

The impact of the SABBL-Lite system has been evaluated on a global, economic, environmental, and societal contexts. Table 4 shows the team's evaluations.

Table 4. Engineering Solutions Impact

Area of Impact	Impact	Description of Impact
Global	Yes	Air travel is ubiquitous. Safer planes impact nearly everyone on Earth.
Economic	Yes	Commercial airlines depend on aircraft safety to sell tickets.
Environmental	Yes	AUV deployment saves fuel burned by large ships towing hydrophones.
Societal	Yes	Located wreckage from lost aircraft provides closure for the families of victims

It became clear in the impact analysis that there is a vested interest in solving the problem of black box recovery. SABBL-Lite serves as a solution that satisfies the needs and desires of people everywhere.

5 System Concept Development and Selection

When posed with the problem of poor black box location and recovery, a variety of solutions were explored before deciding on the final AUV-based solution. The most basic requirement for any solution is an overall increase in the probability of black box recovery. This requirement can be satisfied through multiple methods of approach. Redesigning black boxes to be more locatable and recoverable, developing an emergency black box deployment system, and using new tools to make current search processes more efficient were all valid approaches to increasing the black box recovery rate. Ultimately, these were the three approaches investigated in the system concept development phase.

5.1 Redesigned Black Box

Flight data recorders constantly collect large amounts of information during a typical flight. Any information that an investigator may need in order to make a determination as to what caused an accident is recorded. This information is critical to prevent future accidents, which is why so much time and effort is spent on recovering black boxes after the loss of an aircraft at sea. In order to protect this information better, and prevent the need for search efforts altogether, the team examined the prospect of a flight data uplink using satellites. This concept is demonstrated in the original concept drawing in figure 3 below. By delocalizing the critical information required to investigate accidents, considerable amounts of time and money could be saved by preventing the need for searches, and there would always be data to analyze after each accident. Ultimately, this proposed concept was ruled out as a solution for a few reasons. Firstly, this solution was not conducive to a semester-long project, it would require considerably more resources than the team had available. Secondly, a lack of expertise on the team in the area of satellite data transmission left a question of where to even begin a prototyping effort. Ultimately, this idea shows promise at solving the problem posed, but this team felt unprepared to adequately explore this solution.

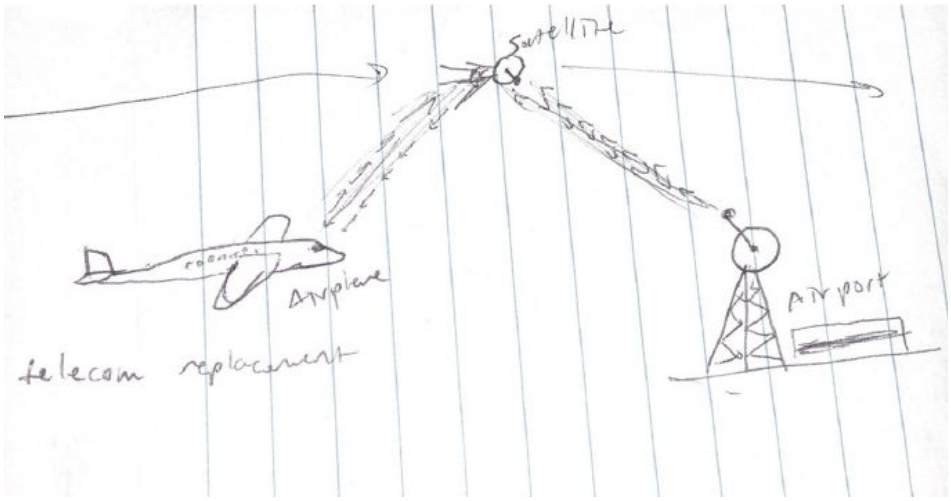


Figure 3. Original drawing for a redesigned method of data collection

5.2 Emergency Black Box Deployment System

Considering that locating objects from the surface of the ocean is a much simpler task than at the bottom of the ocean, another considered system concept involved developing a deployment mechanism, which would jettison a black box from an aircraft upon sudden impact. This approach is demonstrated in the initial system concept drawing in figure 4 below. By ensuring that the black box is visible and buoyant, or can deploy buoyancy support, this mechanism would allow for search teams to utilize aircraft to assist search operations through simple spotting. Without the need for underwater acoustic searches, black box search efforts would become drastically less complex and recovery efforts more reasonable. The group abandoned this idea, however, after considering that there is value in locating the actual wreckage of a crash. Viewing wreckage can assist accident investigators, and often provides closure to the families of the victims, and by jettisoning black boxes, all plane wreckage would be hopelessly lost.

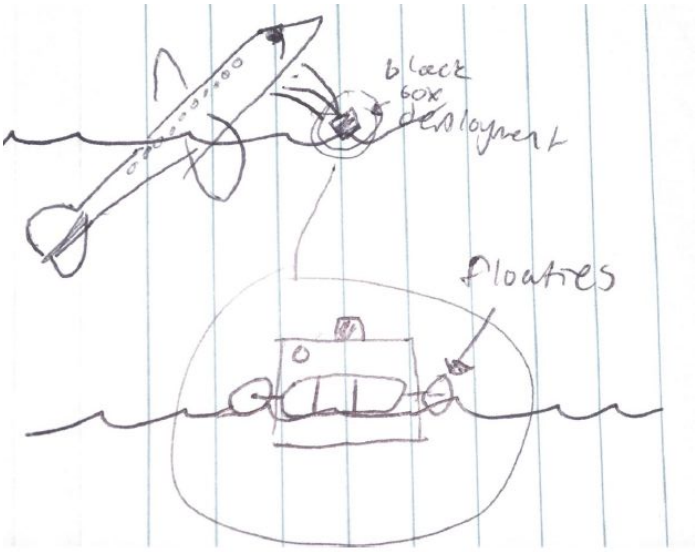


Figure 4. Original drawing for black box deployment system

5.3 Final Selection - Updating Search Tools

Ultimately, the current state of black boxes and their recovery is unlikely to change unless a major shift occurs in this field. In order to assess the best concept, the team generated the concept selection matrix in Table 5 below. Ultimately, it was decided that the simplest and most practical solution was the most likely to be implemented. Recognizing this, the team sought to develop a new tool to aid in the current method of searching for black boxes. The result of this effort was the concept for the SABBL-Lite. By pairing current search methods with new technology used to automate the process, major strides can be made to improve mission success. By changing the way that things are done currently, this solution requires the least amount of change on the part of aircraft manufacturers, airlines, and safety regulatory bodies. This concept appeared in comparison to the others to be the simplest solution to the problem at hand, and the most likely change that the industry would willingly make.

Table 5. Concept Selection Matrix

	Simplicity	**Effectiveness**	**Likelihood of Implementation**	**Practicality**	**Total**
Redesigned Black Box	1	3	2	1	7
Black Box Deployment Mechanism	2	2	1	2	7
Updating Search Tools	3	1	3	3	10

6 Subsystem Analysis and Design

The development of SABBL-Lite was broken down into six distinct subsystems as described in the subsystem hierarchy diagram below in Figure 5. Analysis and design for each subsystem is described in further detail below.

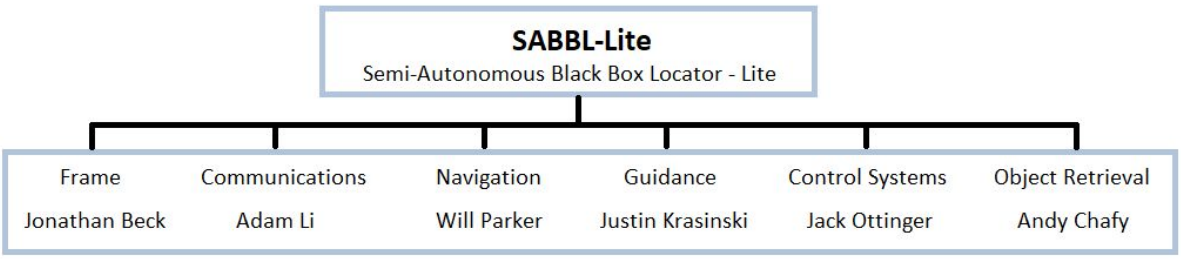


Figure 5. Subsystem hierarchy diagram

6.1 Frame

6.1.1 Purpose

The purpose of the frame subsystem is to create a platform that can effectively mount and contain all other subsystems. Additionally, the frame must serve as a stable and buoyant base such that other subsystems may perform their function. The frame subsystem also includes the waterproof box mounted to the frame which contains all onboard electronics. The frame had to accomplish the customer requirements of being balanced in the water (+/- 5 degrees), being able to mount all other subsystems to it, being slightly positively buoyant, and being waterproof in the onboard electronics section.

6.1.2 Initial Design

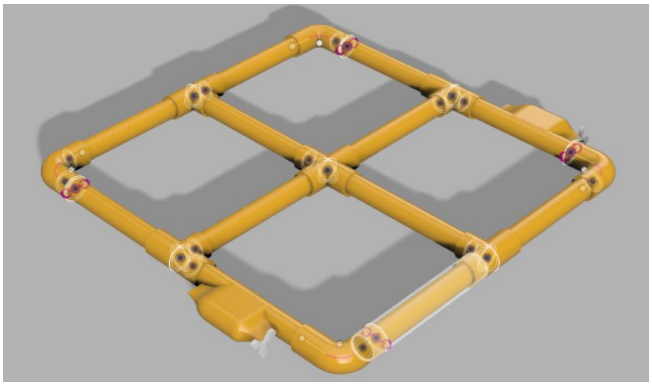


Figure 6. Square Frame CAD

The initial design for the frame of the SABBL-Lite was the square frame. Figure 6 shows the square frame, a 2 ft. x 2 ft. square design made from 1 in. diameter PVC (Polyvinyl Chloride)

piping, and was designed to create the maximum amount of options for mounting of other subsystems, as well as to keep the frame as well balanced as possible in the water. Additionally, since the frame was a square shape, the team would know exactly where the center of gravity for the frame was (the center of the square). However, while the square frame was constructed, it never made it to the testing phase. Before testing occurred, the Non-Ame team decided that in order to increase system stability and to better accommodate the guidance and propulsion subsystems, the frame design needed to change, leading to the second frame concept.

6.1.3 Testing and Iterations

The goals for the frame subsystem during the first pool test were to test balance in the water, test stability while moving through the water, and test the waterproofness of the waterproof compartment. The test included placing the SABBL-Lite system in the pool, having moved in the x and y-axis in the pool, and having it dive to a variety of depths (from 2 ft. to 8 ft.) for a variety of times (5 to 10 minutes). During this test, the frame design tested was called the horseshoe crab frame. This frame is shown in figure 7.

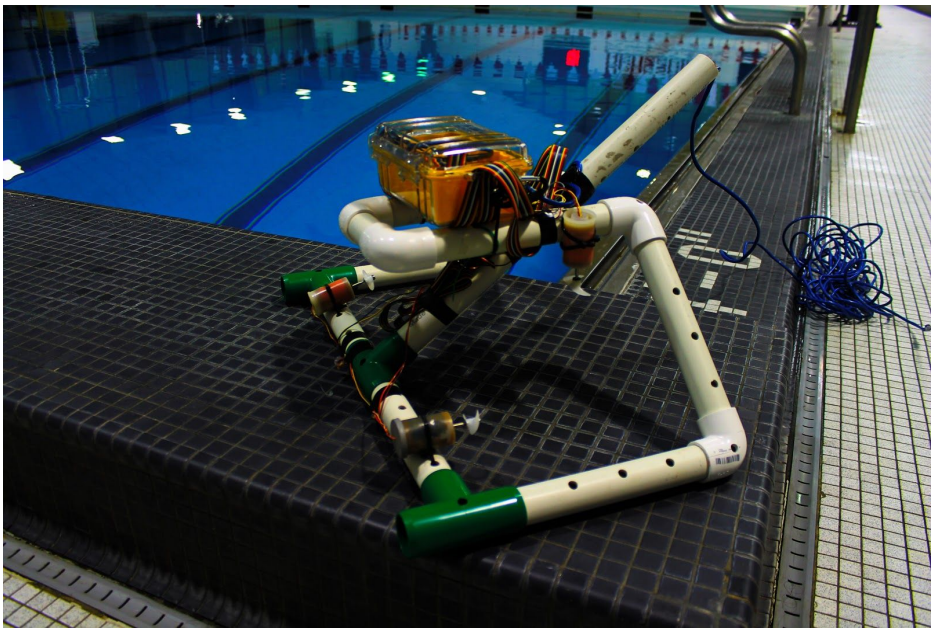


Figure 7. Horseshoe Crab Frame

The horseshoe crab frame is a lifted design consisting of a large rectangular base with a smaller rectangular mounting space raised above the base. The purpose of the horseshoe crab design is to increase the stability of the square design by placing the waterproof onboard electronics box, which is the most buoyant and heaviest individual part of the frame, farther forward on the frame. This move allowed for simple frame balancing. Additionally, the intention of raising the onboard electronics box was to, theoretically, increase the stability of the frame by making the

center of gravity for the frame as far from the center of buoyancy as possible. The horseshoe Crab frame was noted for its static balance in the water. However, the horseshoe crab design was also noted for pitching upwards when in forward motion. Due to the distance between the center of gravity and the horizontal propulsion, the forward motion would cause the frame to rotate, slightly but noticeable, around the center of gravity of the frame. Simply put, the lower portion of the frame would move forward immediately while the top portion of the frame would not. Additionally, the onboard electronics box leaked during the test, and was observed to leak significantly more at deeper depths than at shallower ones.

The goals for the frame subsystem during the second pool test was to perfect the balance and stability, ensure the proper mounting of all other subsystems, and ensure the waterproofness of the onboard electronics box.

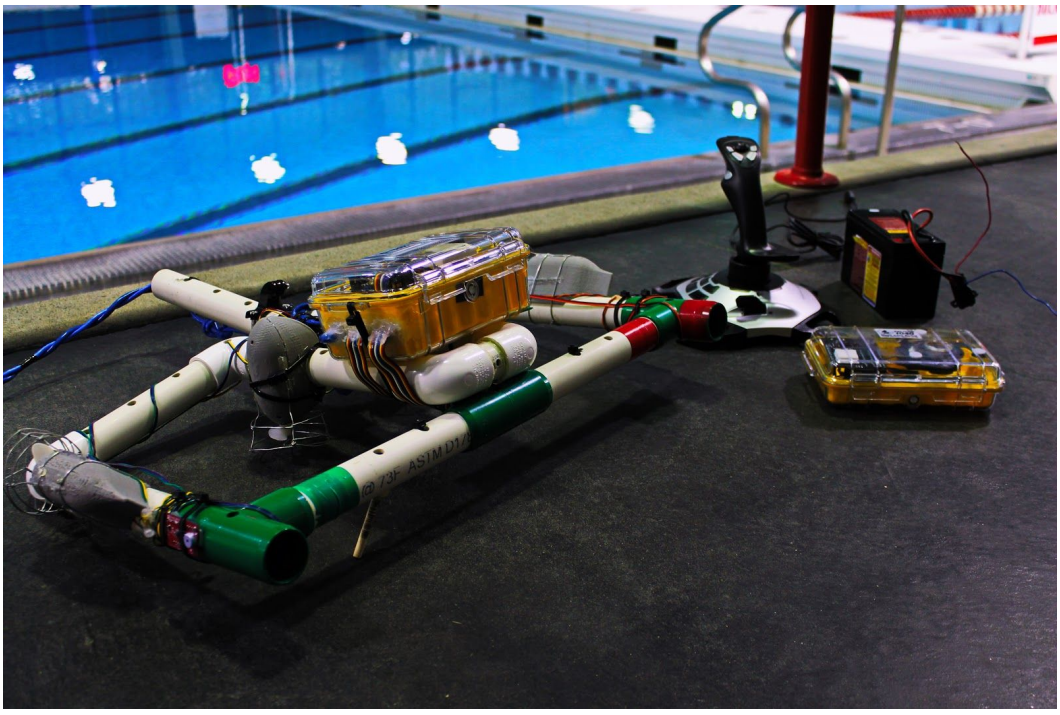


Figure 8. Stingray Frame

The final frame design, shown above in figure 8 dubbed the stingray frame, is a combination of the benefits from the square and horseshoe crab frames. The stingray frame forward loads the system to allow for easier fine tuning of balance (like the horseshoe crab frame) and has the heaviest portion of the frame lowered and nearly level with the rest of the frame (like the square frame). The stingray frame solved the pitching problem encountered during the first test with the horseshoe crab frame. The stingray frame proved to be highly stable when moving through the water and to have a good static balance. Additionally, all waterproofing problems were fixed in

the intermittent period between the first and second tests, as more silicone and hot glue were added to problem areas on the electronics box.

6.1.4 Conclusion

The stingray frame design for the SABBL-Lite was the result of an iterative process of trial and error in frame design. While the square frame did not make it to testing, it was the basis on which the following frame designs would stand on. The horseshoe crab frame refined the concepts from the square frame and experimented with new concepts, some of which succeeded (front-loading the frame) and some of which failed (creating a two-tier frame design). Overall, the frame performed excellently during the final test, fulfilling all of the customer requirements save the balance requirement, which it nearly accomplished.

6.2 Communications

6.2.1 Purpose

Communications as a system aimed to provide the platform which can interface with the vehicle's sensors and systems. As such, communications acts as the backbone to all electrical systems on the vehicle. A combination of hardware and software was required to accomplish this.

6.2.2 Initial Design

Hardware-wise, communications initially used a Raspberry Pi microcomputer and an Arduino Mega 2560 microcontroller combination to minimize the number of electronics in the water while maximizing inputs and outputs. These two boards would then be connected to a shared power source and a two-wire communication protocol, specifically the Universal Asynchronous Receiver-Transmitter protocol at a baud rate, or bit rate of 1 Mbps. Since the Raspberry Pi operated on 3.3V logic whilst the Arduino Mega operated on 5V logic, a simple voltage divider was used to prevent damage to the Raspberry Pi while saving costs. In order to integrate this initial design into the system, an 8-wire 24 AWG per wire cable was used, where 6 of the wires would be symmetrically dedicated to power and ground, while the other two wires would make up the hardware component of this 2-wire communication method. A basic overview diagram describing this method of communication is below in Figure 9.

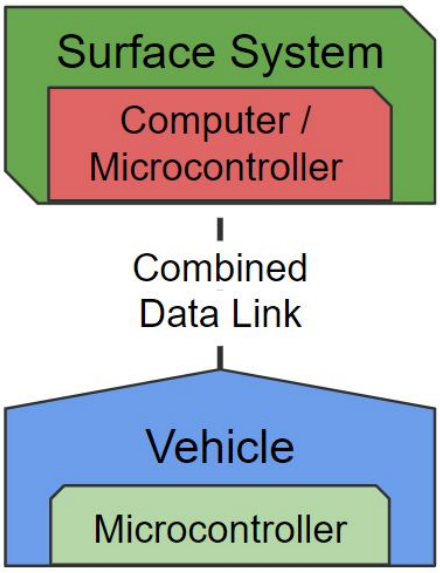


Figure 9. Diagram describing the configuration utilizing a microcontroller to preprocess the data into a combined data link back to a surface system.

Software-wise, a custom ”packet”-based communication system was considered before using ROS and ROSSerial as the core for communication. ROS is the Robot Operating System, which is actually not an operating system itself, but a set of software libraries and tools that act as a platform for robotics. In other words, ROS is a platform that allows for discreet software “nodes” to communicate over a universal messaging system. In this case, ROS ran on the Raspberry Pi to take advantage of its greater processing capabilities over the Arduino Mega. As a platform, ROS is very developed, with an extensive selection of open-source software that can easily extend its functionality. One such software is ROSSerial which serviced the communication link between the Arduino Mega and the Raspberry Pi. ROSSerial opened the universal messaging system on the Pi to the Mega over the UART line, allowing communications to easily ”publish” sensor data from the Mega and act on information it is ”subscribed” to from the Pi. See Appendix H for more detailed information on the software setup used.

6.2.3 Testing and Iterations

Initial testing showed communications working as expected on land and over short wires. In this optimal situation, an average 12 ms delay and 82 kbps transfer speed could be measured, which was more than enough to handle the load expected over the combined data link. Thus control programs and interfaces were developed on top of this system; however, once integrated with the vehicle over the 8-wire cable, issues with the hardware selection became evident.

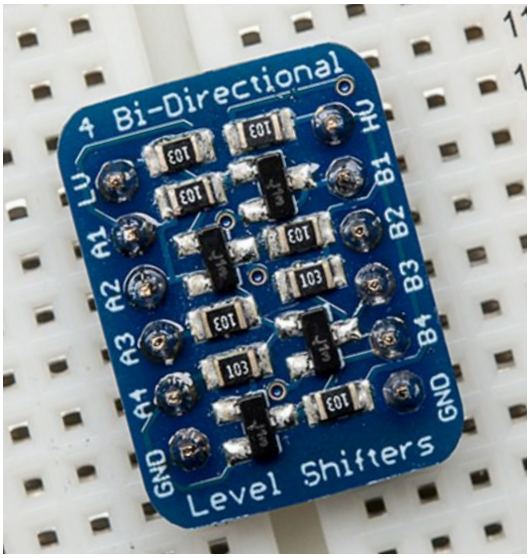


Figure 10. The logic level converter used to increase the signal to noise ratio. [3]

Once integrated, the communication link was experiencing massive amounts of interference, which not only caused occasional glitches, but it caused the underwater Arduino Mega system to crash, rendering the programmed software protections useless. In solving this issue, an oscilloscope was used to analyze the signal over the link. It became apparent that under motor load, the power lines running alongside the signal lines for 50 feet caused enough interference to drop voltages below the logic high level of 2.5V for the Arduino and corrupted the signal to the Mega enough that communication could not be sustained. As a result, logic level converters like the one shown in Figure 10 were used to increase the voltage of the signal over the signal wires in order to increase the signal-to-noise ratio; however, a second wire was still necessary to isolate power from communication since running motors anywhere above an approximate 40% power level was enough to corrupt communications. Figure 11 shows the two cables connected to the vehicle.

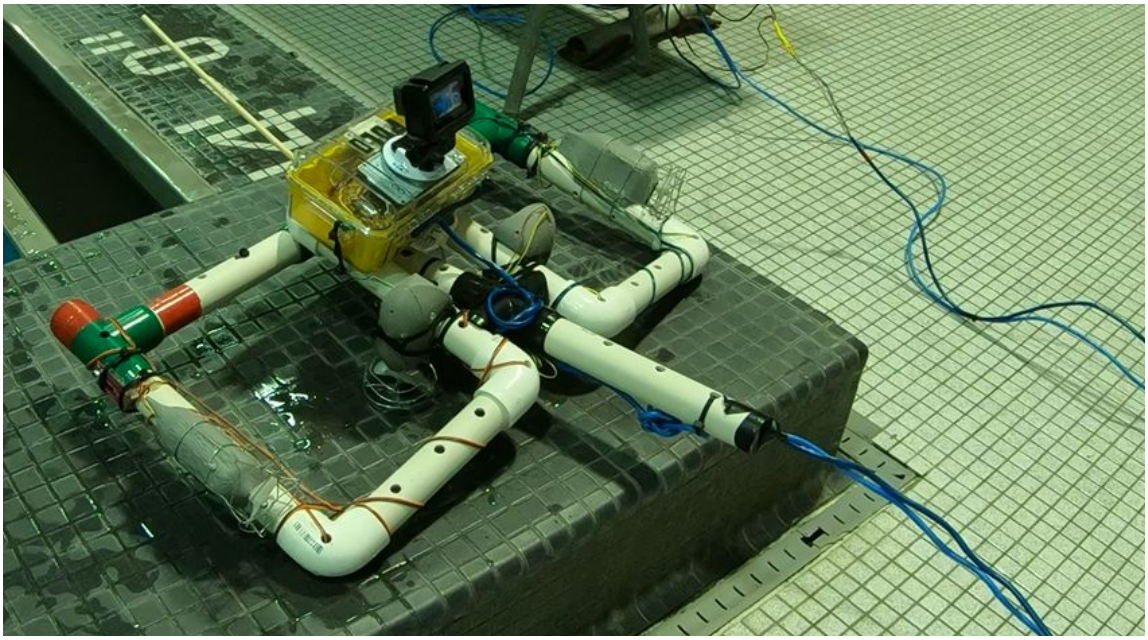


Figure 11. Two cables can be seen clearly attached to the vehicle

However, it became apparent again that the two signal wires in the same line interfered with each other, especially when at one point, opposite wires were connected at both ends, but the system was still able to receive data. With the twisting of the wires and the length of the cable, one wire was capable of inducing a signal and any of the 8 other wires. Further research revealed that the type of cable being used, one with twisted pairs of wires, did not inherently reduce interference, but required special circuitry to do so. As such, the signal wires were split between this additional cable and the power cable, which was enough to demonstrate reliable communications on land with a power level of up to around 95% on the motors; however, this once again failed once the system entered the water as it was not a resilient enough system and the wires were in closer proximity. Additionally, it is likely that the viscosity of the water applied enough load to the motors that enough current was running through the wires to interfere with communications. Finally, a backup system involving the use of a hardwired remote like the one shown in figure 12 directly sent power to the motors was implemented in order to demonstrate the proof of concept.

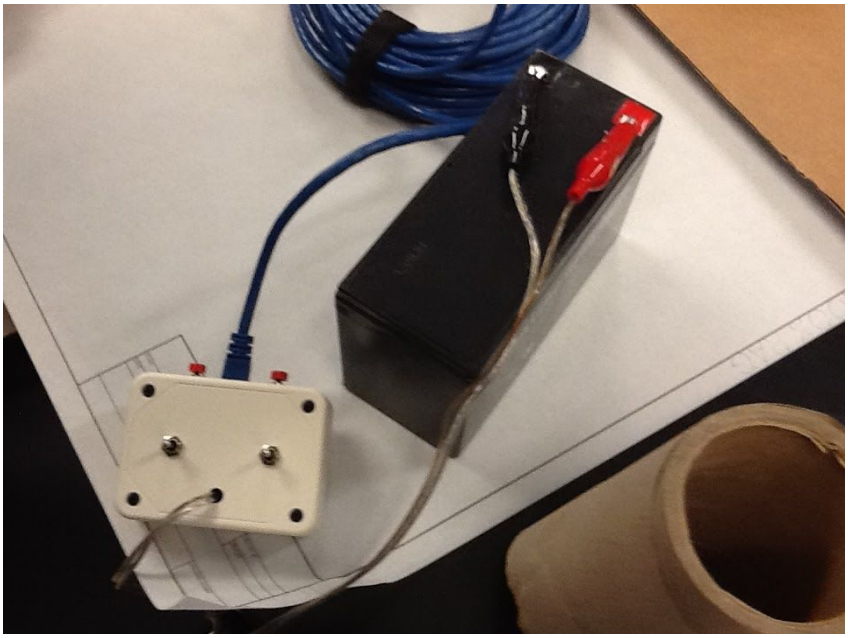


Figure 12. A picture of a similar controller to the hardwired one used to demonstrate the proof of concept [4]

6.2.4 Conclusion

The original design for communications was incapable of satisfying the design requirements once integrated. This is primarily because of hardware issues as demonstrated by the number of problems encountered in the prior section once integrated. Moving forward, a need for better interference-resilient cabling and any supporting hardware is the primary improvement needed. For example, in order to continue using the existing cabling, a set of two balanced receiver and driver pairs would be needed to take advantage of the twisted pairs in interference reduction. It is worth noting that this is an artificial problem caused by the need for a surface system. As the intended design product would not have required such a surface system, future iterations of this project will eventually avoid this problem of cabling and interference altogether as communications became fully integrated with the vehicle.

6.3 Propulsion

6.3.1 Purpose

The propulsion sub-system was designed to meet three customer requirements. These requirements include diving and resurfacing, as well as being able to move in three dimensions. These requirements were translated into the following statements: Free and independent movement along the Z-axis, and free movement on the XY-plane with rotation around the Z-axis. These axes are explicitly shown in figure 13.

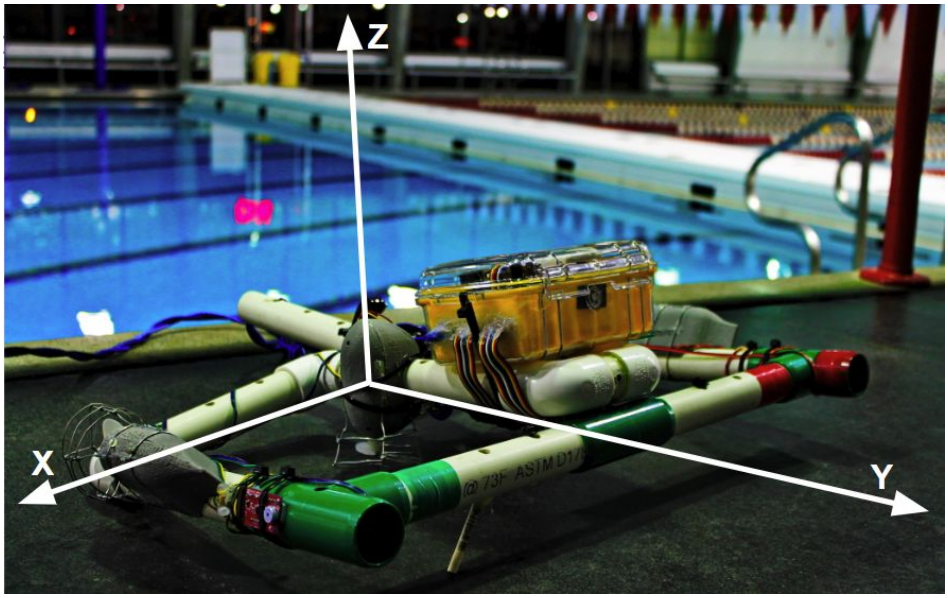


Figure 13. Defining Polo's Coordinate System

It was later apparent that the customer also required retrieval of the flight data recorder, adding a constraint that vertical thrust had to be greater than the combined drag and weight of the vehicle and the black box. Finally, it was required that, in the event of a power failure, the vehicle would float to the surface. Without power, it would be impossible to retrieve the device, and the customer would take a loss on their humanitarian investment

It should also be noted that the subsystem was originally named "Control Mechanisms", as it was not confirmed during the project's inception that varied-directional thrusters would be used, and other designs including hydrofoils or ballast tanks were also considered but were dismissed due to projected cost and increased complexity.

6.3.2 Initial Design

The initial design for propelling POLO in the water was to place three to four 12-volt DC motors, cased in wax-filled film canisters (to ensure waterproofing, shown in figure 14), with omnidirectional propellers attached to the drive shaft of each motor.

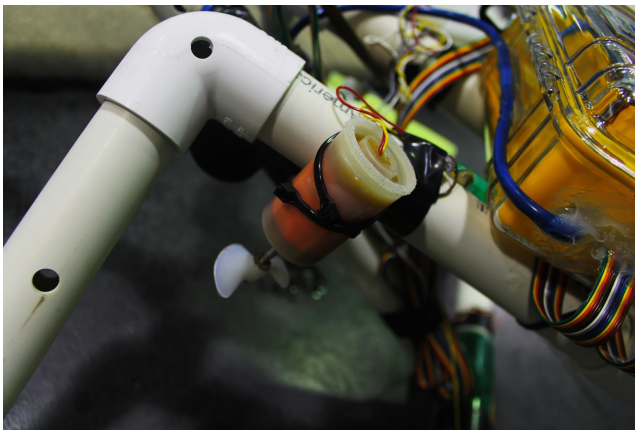


Figure 14. Waterproofed Motor Assembly

Two motors were to be placed horizontally on opposite sides of the vehicle, facing to the rear. These could be independently controlled to spin POLO around the Z axis, or used in tandem to move the vehicle forward or backward. These allowed for complete utilization of the XY-plane. One to two more motors were to be placed vertically near the center of mass to move the vehicle along the Z-axis. It was unclear during brainstorming whether one or two motors would be used, but it was ultimately decided that a second motor would add not only power but stability, as the motors could be placed in line with the center of gravity and buoyancy, instead of having to be placed exactly on these locations. These two motors were linked to a single switch so they would dive and resurface simultaneously, so as to not risk flipping the vehicle.

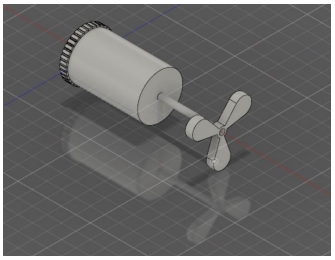
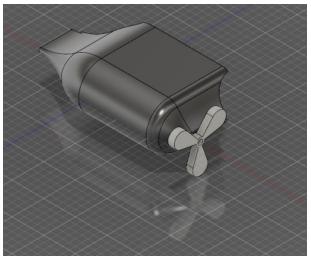
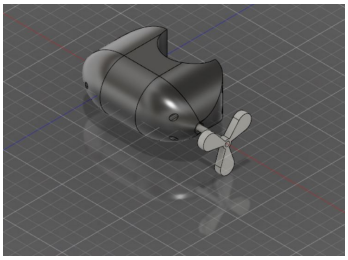
a)  **b)**  **c)** 

Figure 15. Renderings of the motor and fairings. **a)** Motor Rendering, **b)** Horizontal Motor Fairing Rendering, **c)** Vertical Motor Fairing Rendering

All four motors were attached using two types of 3D printed fairing, differing depending on whether they were horizontal or vertical. The original renderings are shown in figure 15**.** The horizontal motors were given fairings that were positioned parallel to the tube frame, while the vertical motors had to be attached perpendicular to the tubing, which caused more complexity in terms of design and manufacturing. Due to this, the horizontal motor fairings could be printed as a single piece, while the vertical motor fairings had to be designed as two pieces, and were later hot glued together, sealing the motor inside. These were attached to the frame using a combination of 18-gauge wire and zip-ties and were satisfactory for prototyping purposes. The

fairings featured a streamlined design to reduce drag in the water and increase thrust and efficiency, as can be seen in figure 16.

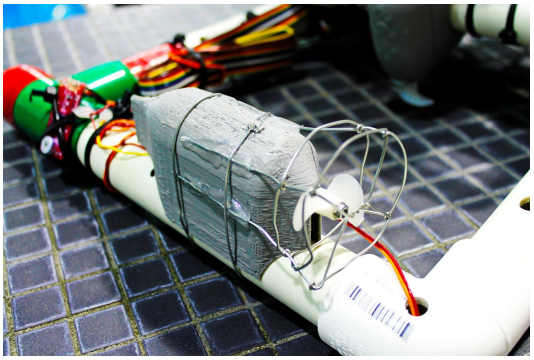
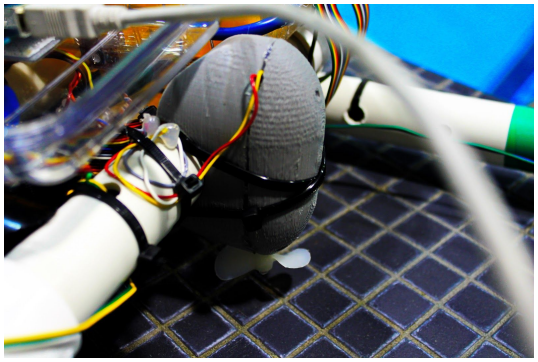
a)  **b)** 

Figure 16. 3D Printed components. **a)** Horizontal and **b)** Vertical Motor Fairings

Creating a slightly positively-buoyant craft was a combined effort of the entire team, but was overseen by the propulsion and frame subsystems. The density of the craft is inherently a control mechanism because it changes how the vehicle maneuvers in the water. The initial design was generous as to how dense the craft would be, and two large pool noodles were attached on opposite sides of the frame

6.3.3 Testing and Iterations

This design was largely successful, though a few problems were noted during the testing phase. During the first test, POLO was so buoyant that diving was impossible. Experiments were done to ensure only slight positive buoyancy, making diving effortless. While attempting to recover the black box for the first time, it was discovered that retrieval was very difficult. Small amounts of foam were added to certify that this was an easy task as well. Once these corrections were in place, the vehicle could now operate in three dimensions, even with the added weight of the black box. There was one problem that was unforeseen during the design phase, and that was safety; spinning propellers pose an inherent risk for humans, sea life, and in this application, for the PACT (power and communications tether).

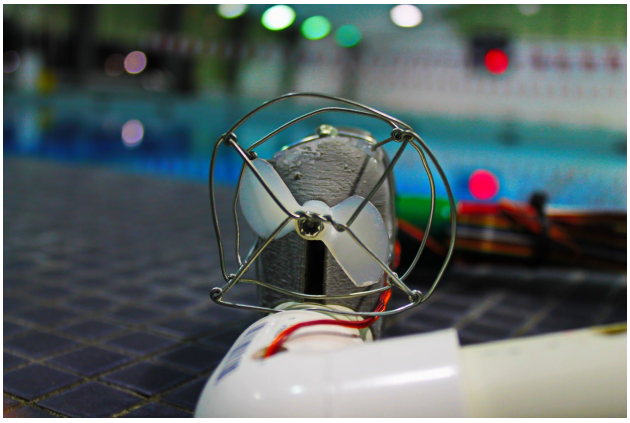


Figure 17. Motor Cage Detail

Therefore, a circular, 2-inch diameter cage was developed from 18 gauge wire, shown in figures 16a and 17 which was then glued to the fairings. Great care was taken to ensure that the cage would not interfere with the propellers, as this could cause the propellers, or worse, the motors to fail.

This vehicle differs from other AUVs and UAVs in respect to power consumption. While others have to deal with battery life and keeping things efficient for this constraint, the SABBL-Lite prototype was powered via a tether. In concept, this would mean the vehicle could operate for as long as possible. What wasn't foreseen, however, was that only so much power could be pushed through the tether to the motors. When trying to operate all four motors simultaneously, less thrust was observed all around. In addition, since the two vertical motors were connected to the same switch, they only received half as much power as each horizontal motor. This was fixed by adding more wires alongside the existing wires, to increase the amount of power that each motor would receive.

Initial testing showed that the POLO craft was too buoyant, and could not dive, even with full thrust. Density was then increased using various weights, and it was now too dense, and could not retrieve the black box. If there had been a power failure during this stage, it could not easily be pulled out of the water. Using a variety of pool noodles and weights, Polo was balanced and slightly buoyant in the pool before the second day of testing was finished.

6.3.4 Conclusion

The initial design of the propulsion subsystem satisfied the customer requirements. Some amendments were made to the design to increase efficiency, safety, and power in order to better meet the needs of the customer, and provide for a longer life in the harsh sea environment. Future testing will incorporate research into alternative control mechanisms, specifically ballast

tanks, as these use significantly less power in order to dive. Similarly, the frame will become a more hydrodynamic shape, increasing efficiency.

6.4 Navigation

6.4.1 Purpose

The purpose of the navigation subsystem is to provide enough information about the vehicle's state at any moment to determine its position. The vehicle's position relative to its last GPS (Global Positioning System) - verified location, relative to the flight data recorder, and relative to the surface of the water are all relevant in a search effort. By collecting reliable time-stamped data about the vehicle's location, it is possible to correlate that location data with the corresponding time-stamped acoustic amplitude readings from the flight data recorder. It is important that search teams are able to correlate location and acoustic signal strength, since this information is what will allow teams to identify the areas in which the flight data recorder is most likely to be found.

6.4.2 Initial Design

The navigation subsystem was required to allow the vehicle to (1) avoid obstacles, (2) determine and maintain depth, (3) determine its location and direction, and (4) log navigation data over time. Challenges pertinent in this project included the fact that more established navigation methods, like those which use GPS, were not viable underwater because GPS signals are incapable of penetrating the water by more than a few inches. As a result, initial research centered on similar underwater autonomous systems, where the same challenges existed. The most common methods of underwater navigation were discovered to be (1) dead-reckoning through the use of an Inertial Measurement Unit (IMU), and (2) using a number of preplaced acoustic transponders of known location to trilaterate position relative to those transponders. While trilateration is a more accurate method of navigation, it is far less practical for a search mission than dead reckoning, since search teams would not be required to install transponders on the ocean floor for this navigation method to work. Because the vehicle is more useful when it navigates independently using dead reckoning, this was determined to be the best approach for horizontal position determination. Therefore, an inertial measurement unit, made up of a three-axis accelerometer, a three-axis gyroscope, and a three-axis magnetometer is necessary for this approach. After benchmarking against other readily available IMUs, the Adafruit LSM9DS1 was determined to be the most cost-effective and reliable unit for this vehicle. I2C compatibility was also a major driver for the acquisition of this unit, as it made interfacing with the vehicle's Arduino Mega simpler and more robust.

For vertical position determination (depth), a digital pressure sensor was determined to be the best method of data collection. Because the density of water is high, as the vehicle changes depth, a considerable change in hydrostatic pressure arises. With a sensor that can read the changes in hydrostatic pressure, it is possible to determine depth through use of the relationship,

$$P = \rho g h$$

P is the gauge pressure (absolute pressure minus atmospheric pressure), p is the density of water at the pool temperature of 80 F, and h is the depth of the sensor in the water. After substantial benchmarking between pressure sensor options, the MS5803-14BA Pressure Sensor was determined to be the best option. This sensor was advertised as reading accurately +/- 2” to a depth of beyond 100m. The MS5803 has a gel-filled pressure cavity, so all that needed to be done to make the sensor waterproof was (1) apply silicone conformal coating and (2) apply extra silicone around the leads. This sensor also was I2C compatible, which made prototyping with the Arduino Mega simple and robust.

In order to avoid obstacles, the vehicle must be capable of determining distances to objects underwater, which involved a search for sensing methods that were capable of operating in this environment. Viable distance determination tools included optical rangefinders and ultrasonic rangefinders. Due to cost limitations and precedent in the field towards ultrasonic ranging, ultrasonic rangefinders were determined to be the optimal option for obstacle detection. There were not many available options in the search for a waterproof ultrasonic rangefinder, but a viable sensor unit was found in the JSNSR04T. This sensor was not I2C compatible, but appeared easy to integrate onto the onboard microcontroller. In order to make intelligent obstacle avoidance maneuvers, the initial design required two sensors set at 2’ apart. Since the sensors had a range of about 50 degrees, at this distance apart, they would be capable of reliably determining the vehicle’s orientation as it approaches a wall, and an optimal maneuver could be determined based on this understanding of not only where the wall is, but also how the vehicle is approaching the wall. This basic method of obstacle detection and avoidance is outlined in figure 18 below.

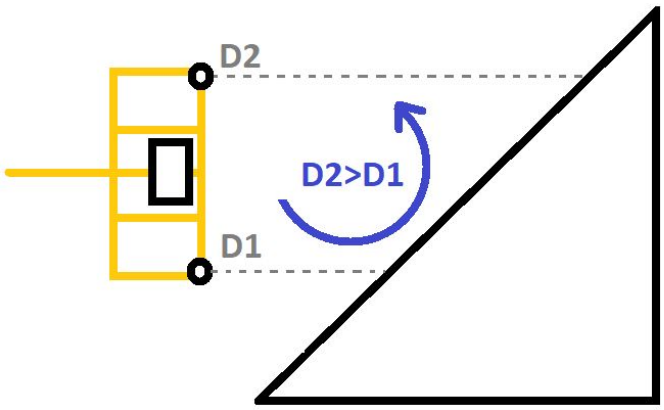


Figure 18. Obstacle detection and avoidance algorithm visualization

Together, these three sensors provide all of the information needed to satisfy the requirements for this subsystem. A considerable amount of research went into benchmarking these devices to ensure that navigation information would be viable for use on the vehicle. Once these sensors were acquired, testing immediately began in order to verify that their use on the vehicle would satisfy the requirements.

6.4.3 Testing and Iterations

A number of testing procedures were performed to verify the viability of each component for use on the vehicle. Each sensor was assessed on land, then if a component passed dry testing, it was waterproofed and a submersion test was performed. Following a successful submersion test, a pool test was performed to assess waterproofness and component efficacy at the maximum vehicle depth of 15'. Assuming that a component has passed all of these tests, a final test was performed with the component integrated onto the vehicle.

The LSM9DS1 IMU was promising initially. Data was easily collected by the microcontroller and passed to the Raspberry Pi via the communications infrastructure for the vehicle. However, issues in testing soon arose as a double integration from accelerometer data to yield a position estimate was determined to be too error-prone to proceed. However, the magnetometer and gyroscope data were still useful in the vehicle, so as testing proceeded on the other components, the team planned on implementing the IMU in the demonstration vehicle. This sensor never needed to be waterproofed or water tested because it was to be housed in the already waterproofed control box.

The JSNSRO4T distance sensors were dry tested by a similar method. After connecting to the microcontroller, a code was written to read distance measurements. Once these results were reliable and consistent, a second distance sensor was added, and a script was written to compare the two sensor outputs, and recommend a direction to turn when the distance to a wall fell below

two feet. A testing mechanism, shown in figure 19 below, was built from a 2’ length of 1/2 inch PVC pipe, with a distance sensor mounted on each side. When the sensors detected an obstacle, a red LED would light up on one side indicating the optimal direction in which to rotate to avoid the obstacle. This test setup worked well, but was prone to error, and noise in the sensor readings meant that, at times, a user would be instructed to turn when there was no obstacle nearby. This error happened infrequently, and can most likely be attributed to issues in the sensors, not with the setup of the test. While dry tests for the distance sensor were fruitful, pool tests were not pursued due to lack of time. Given more time, these distance sensors could have successfully been used to help the vehicle navigate the pool walls.

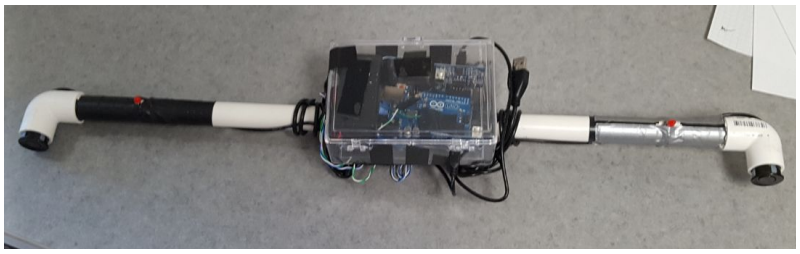


Figure 19. Testing Mechanism for Obstacle Avoidance

The MS5803 pressure sensor set up with the microcontroller was simple due to the I2C capability. A code was written to read absolute pressure, then from this code, a separate code was written to derive depth using the depth equation from hydrostatic pressure. Upon inducing stagnation pressure on the sensor by blowing through a straw, a considerable change in pressure and corresponding estimated depth was read. This successful test was encouraging, so the sensor was waterproofed with silicone modified conformal coating, and a shallow submersion test was done. This test was successful, but testers noted that atmospheric pressure varied substantially by location with a range of 98 kPa to around 101.3 kPa. Because of this difference in atmospheric pressure, it became clear that the depth sensor would need to be calibrated for atmospheric pressure at each testing location before each test. This change was implemented into the code and depth readings were all accurate. Upon passing shallow submersion tests easily, a 10’ length of extension cable was attached and waterproofed at the leads of the sensor. By attaching a tape measure to the end of this cable, it was possible to submerge the sensor to a depth of 7’. This depth would suffice to prove that the sensor was waterproof at this depth and gave accurate readings. The readings from this test are compiled in the table below. These measurements were all accurate to +/- 2 inches, just as had been advertised for the sensor.

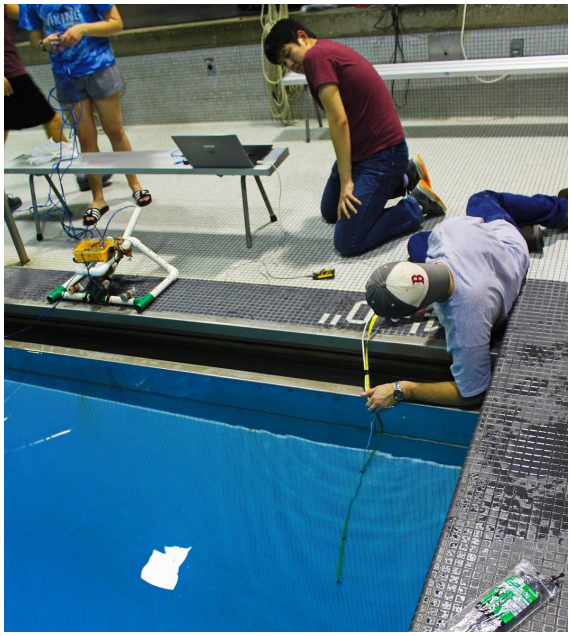


Figure 20. Pressure Sensor Independent Pool Test

Table 6. Measured vs. Actual Depth

Measured Depth (in)	86	74	61	49.5	37	25	13	2
Actual depth (in)	84	72	60	48	36	24	12	2

Upon passing the pool test, the depth sensor was deemed ready for integration onto the vehicle. A code was written which actively regulated depth through providing thrust based on the actual depth compared to the desired depth. PID control was investigated for this depth regulation method. One issue that arose in preparing for this test was that as the vehicle moves through the water, stagnation pressure due to the movement could contribute to error in depth measurements. In order to combat this, a PVC shield, shown in figure 21 below, was attached to surround the depth sensor and prevent stagnation pressure interference. While the integrated depth sensor motor control unit worked well in dry testing, it failed in pool testing. This failure can most likely be attributed to poor waterproofing on the leads, and issue which was discovered minutes before the final demonstration, and which was not able to be adequately addressed prior to final testing.

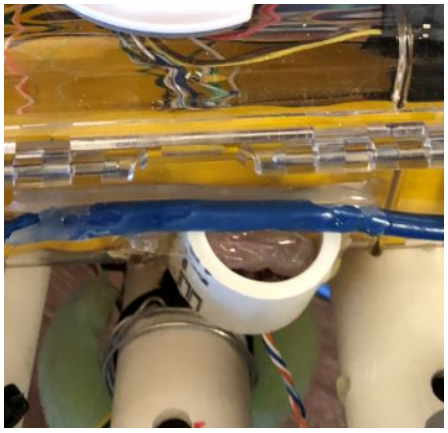


Figure 21. Depth sensor mounting with PVC shield

6.4.4 Conclusion

Through collecting and processing a large amount of data, the navigation subsystem components were able to determine an estimate of the vehicle's horizontal position, depth, and distance and orientation relative to obstacles. The only component that was integrated onto the final vehicle was the depth sensor, which was tested in the final demonstration for accuracy at depth and ability to regulate depth. Ultimately, issues arising from waterproofing impeded the sensor's ability to collect data underwater, but *some* data was sent sporadically. This limited data allowed the team to demonstrate that the sensor could read depth accurately and that the vehicle program developed was able to determine how to correct vehicle depth given depth data.

6.5 Guidance

6.5.1 Purpose

The purpose of the guidance subsystem is to locate the submerged flight data recorder. When flight data recorders are submerged under water they start to send out acoustic pings for thirty days. Within that time frame, the SABBL-Lite will be deployed to search for the flight data recorder. Using acoustic listening devices, guidance will detect the acoustic ping signals and be able to track the vehicle towards the source. Once the vehicle has reached the flight data recorder, the object retrieval subsystem will pick up the flight data recorder.

6.5.2 Initial Design

The initial design for the guidance subsystem was to use three omnidirectional microphones to be able to locate the black box pings under water. Two of the microphones were to be mounted on the front sides of POLO, and the last one was to be centered on the backside of the craft. Since the microphones are omnidirectional there will be some form soundproofing cone around the three microphones. The purpose of this is to allow the microphones to sense sound only in a preset angle, in attempt to get a clear difference of sound levels from all three microphones, in

order to tell where in relation to the craft the flight data recorders are. For the prototype, the use of hydrophones will not be used, since they are too expensive to fit in the project's budget. Microphones are not waterproofed out of the box, the three microphones would have to be waterproofed, yet still able to hear acoustic signals while submerged. For waterproofing the microphones the use of a special 3M silicone adhesive, that is dispensed out of a hot glue gun, is to be implemented on the entire surface of the microphone unit.

6.5.3 Testing and Iterations

The first test that was conducted was a test to ensure that the microphones were working and that audio signals were able to be sent to computer software to be analyzed. During this test, the microphones worked correctly and decibel readings were provided from the microphone using the “gate” and “envelope” pins on the microphone. After the dry test, the microphones were waterproofed using a silicone conformal coating that is applied with a brush. The entire surface of the microphone unit received two layers of this coating as well as the wire leads coming off the unit. Once the coating was applied, the next test was performed. The microphone was submerged shallow freshwater to see if it can still pick up acoustic signals. During the shallow water test, the conformal coating around the wire leads on the microphone unit became cracked and were no longer waterproofed. This showed that the conformal coating would alone not be enough to waterproof the microphone units. The use of hot glue to encapsulate the wire leads into the microphone was implemented. This ensured that even if the wires were bent and moved around the microphone wires would remain waterproof. During both the dry test and the shallow water test, decibel readings were provided as an average over a given set period of time. Since it was an average reading there was no way to fully understand what source the sound was coming from. To overcome this disadvantage, the use of the “audio” pin was used along with a Fourier transform. With this discrete audio frequencies were able to be displayed on a computer. The previous two tests were redone. The only step that was changed procedurally was to let the microphones listen to the ambient sound beforehand. After this, the ambient noise levels were able to be compared to the noise levels when the black box was going off to show a visible change on the correct frequency that the black box sends out. On land, the microphones were able to display the correct frequencies at a distance of 1.5 meters away.

Figure 22. Microphone mounted on the front left side of POLO

6.5.4 Conclusion

The guidance subsystem performed correctly for the prototype demonstration. The microphones were able to hear discrete frequencies at a distance of at least 1.5 meters away. Due to communication difficulties the microphones that were attached to the POLO frame were not able to be tested. A single microphone attached directly to a computer on dry land was tested instead.

6.6 Object Retrieval

6.6.1 Purpose

The object retrieval subsystem's main purpose was to design an economically feasible object to simulate a black box and develop a mechanism by which the vehicle can pick that object up. To accomplish this, a combination of mechanical engineering, electrical engineering, and computer science was required to create a feasible final project.

6.6.2 Initial Design

At a fundamental level there were four main requirements for this subsystem. First, the box must be waterproof. Second the black box must be able to generate an audible acoustic signal while underwater. Third, the black box must be as close to neutral buoyancy as possible to limit the extra weight it adds to the drone in the lifting process. Finally the black box and drone frame must both have a system making object retrieval possible. The first priority was to create a functioning black box that satisfied the first and second requirements. Initially, research on underwater speakers led to the conclusion that they were far too expensive to consider using in the project. Given this, the black box design was now required to consist of two components, a waterproof casing, and a speaker loud enough to be detectable through the waterproof casing. After some extra research on possible speaker solutions, a piezoelectric surface transducer was

determined to be the best option. This surface transducer works by vibrating a surface at a certain frequency effectively turning the surface into a speaker. Luckily, Will, the team's navigation lead, was familiar with surface transducers and owned one. After obtaining the transducer, to move forward the speaker had to generate a specific frequency. Quickly, it was determined that a battery powered arduino was the best option to generate the signal. Any other option was either much to expensive and complex, or unreliable, with the arduino board generating and modifying the frequency of the signal being generating only required a couple of lines of code. After designing the signal generation, the next step was to find waterproof housing. To find the correct kind of housing, two factors were considered, (1) the inherent buoyancy of the waterproof container, and (2) the surface area available to attach the surface transducer. The inherent buoyancy of the box had to be at a minimum to make achieving a near neutrally buoyant black feasible. However, the box's dimensions must still be large enough to contain all of the necessary components for the signal generation. After careful consideration of box dimensions, it was decided that a pelican 1040 would be the most ideal container. The Pelican case is shown in figure 23.

Figure 23. Waterproof Case [5]

The pelican container provided the largest surface area for the surface transducer , while having a relatively low level of buoyancy at 1.25 lbs. Finally, the box needed to be negatively buoyant, this was accomplished by adding a steel plate to the bottom of the box that weighs 1.5lbs, making the overall weight of the case in the water .25lbs. The final iteration of the black box can be seen in figure 24.

Figure 24. Final BlackBox

After the design of the simulated black box was finalized, it was time to consider possible solutions for object retrieval. When designing the object retrieval mechanism, a couple factors were considered. The most important of these factors were (1) reliability of the mechanism and (2) simplicity of the mechanism. When considering these factors a couple of concepts were developed. The first idea was an electromagnet scoop, which utilized a magnetic field created by the steel plate on the black box and an electromagnet on the scoop to guide the black box onto a scoop reliability. Unfortunately, while this idea was promising, as it turns out the permittivity of water is incredibly high and the amount of amperage required to make an appreciable magnetic field would put any living organism at a high risk of electrocution and would be nearly impossible to generate using a battery. Another idea was just a scoop in itself, while simple the scoops reliability is incredibly small due to the fact that getting a scoop to position itself underneath the black box would result in bumping the black box around more than scooping underneath. The final design was the hook and loop, essentially a wire hook was attached below the drones center of gravity and a long wire loop was attached to the box as seen in figure 25 .

Figure 25. Final hooking mechanism

6.6.3 Testing and Iterations

After completing the initial design, it was time to test the concept. The first test was a survivability test, the black box was dropped to a depth of 15 ft and left there for 10 minutes to

make sure the electronics would be safe. This test was the most crucial test and was mostly successful. During this test the initial surface transducer separated from its base and broke, requiring an emergency order of another surface transducer. Other than that ,the test went well and no water was found within the waterproof casing. Other subsequent testing involved testing the consistency of the retrieval mechanism. This mechanism was surprisingly consistent and forgiving; however, during the testing, the team determined the weight of the black box should be closer to neutral buoyancy. To accomplish this, a pool noodle was cut up and trial and error method was used until the ideal buoyancy was obtained. In the end about 2 inches of pool noodle was used to get as close to neutral buoyancy as possible. When testing with the guidance subsystem, it was determined that the microphones used could not detect the noise, through some extra testing the reason for this was determined to be the conformal coating on the microphones. However, it was noted that for future iterations of the black box, a amplifier and a surface transducer with a higher impedance would be helpful in creating a louder ping for guidance to hear.

6.6.4 Conclusion

Overall, the design process of the object retrieval subsystem went relatively smoothly. While there were some minor inconveniences along the way, the simulated black box and the black box retrieval mechanism worked well for demo day. In the future, it would be better if a real black box were obtained, given the stark weight difference between the simulated black box and real black box , a real black box would require a different approach to retrieval. In conclusion the subsystem worked well and accomplished its goal for demo day.

7 Results and Discussion

7.1 Results

Due to the nature of this project, the team only had two opportunities to perform tests on the system in the pool were available. To make effective use of limited testing time blocks, strict plans for each test were developed. The first test plan is shown in its entirety below.

Test Plan:

Waterproof Tests – vehicle and black box (Duration: ~35 mins)

- Submerge to ~ 3 ft. for 1 minutes, then check waterproofing. If dry, proceed.
- Submerge to ~ 3 ft. for 5 minutes, then check waterproofing. If dry, proceed.
- Submerge to ~ 6 ft. for 5 minutes, then check waterproofing. If dry, proceed.
- Submerge to max Depth for 2 minutes, then check waterproofing. If dry, proceed.
- Submerge to max depth for 5 minutes, then check waterproofing. If dry, proceed.

- Submerge to max depth for 10 minutes. If waterproof still, then confidence in waterproofness is high.

Sound Detector Test

- Drop sound detector to full depth to check waterproofness. Keep at depth for 5 minutes and determine whether it is working by listening for motor noise from the vehicle/black box.
- If waterproof, continue by moving closer to/further from black box. Check to see that amplitude of signal increases as distance to box decreases.
- Try turning sound detector in different directions, does amplitude increase/decrease when sound detector is rotated towards/away from source?
- At what distance can the signal from the black box be heard from? (may need a beep pattern to distinguish between signal and noise)

Depth Sensor Test:

- Calibrate depth sensor code for atmospheric pressure in pool area. Very important step.
- Drop depth sensor to known depth of 1m (using tape measure). Ensure that depth reading is accurate.
- Submerge depth sensor to max depth. Ensure reading is accurate.
- Assess whether depth sensor can be used as "float switch" by seeing if accurate to a few centimeters at surface of water. Report results.

Propulsion Systems Test:

- Connect manual controller to battery and to vehicle. Submerge vehicle.
- Move vehicle in 2 dimensions on surface using controller.
- Attempt a dive, determine whether vertical motors can provide enough thrust to counteract buoyancy.
- Modify frame where necessary to make propulsion more efficient or neutralize buoyancy.

After the initial tests leads of each subsequent subsystem took notes on successes and failures. The resulting notes were summarized and the following summary was used to dictate the next tests needed to finalize the project.

- Depth sensor was calibrated to pool atmospheric pressure of 99.75 kpa
- Sensor was accurate to +/- 2 inches to full depth possible with testing apparatus (about 7 feet)
- Sensor was accurate at shallow depths, reliably able to read down to the inch - capable of being used as safety "float switch"
- Vehicle frame originally too buoyant, positive pitch in water. To level vehicle out, weight was added to the front of the vehicle. This weight made the vehicle slightly negatively buoyant. To add buoyancy, 1 in of pool noodle foam added to right and left arm. Following this, vehicle was both sitting level in water, but also was slightly positively buoyant, exactly what was desired.

- When horizontal thrust occurred, vehicle pitched up due to high center of buoyancy. In order to fix this, group proposes that frame is modified slightly such that the box sits lower, closer to the center of thrust.
- Vertical thrust lacking. Will need to modify controller circuit to allow for full current to reach both vertical thrusters. When thrusters are operated off of motor controller, they will receive twice as much current and will hopefully be more capable of lifting and submerging the vehicle in a more expedient way.
- Vehicle waterproofness was serviceable, but could use more waterproofing measures. After 5 minutes at depth of ~10' and 3 minutes at 15', very small amount of water entered the control box in both tests. Enough to create a small pool, but not enough to indicate a serious problem. Recommended action includes:
    - Adding more waterproofing silicon to electronics inlets.
    - Bagging electronics within control box to add extra layer of waterproofing
    - Attaching a sponge to the bottom of the control box to absorb any water that enters, preventing water from moving in control box and possibly damaging electronics.
- Black box was tested and waterproof at depth 15' for 5 minutes.
- Black box surface transducer broke during testing. Surface mount broke away from diaphragm, needed support in back. Another surface transducer ordered, will support in the future to prevent break again.

OVERALL: Partial success. Got vehicle to move

Overall the first test went relatively well. Some key successes were the effective waterproofing of the black box, successful testing of the depth sensors, and overall functionality of the system. During testing some quick fixes were applied as well, one key fix was fine tuning the buoyancy of the vehicle using pool noodles. There were, however, many failures during the first round of testing as well. For example, during the first round of testing there was not enough vertical thrust, which required in depth modification of the controller circuit, and the electronics box was not fully waterproof which luckily did not end as catastrophically as it could have.Finally, the black box surface transducer used had completely fallen off, which required an emergency order of a new surface transducer. Using these lessons learned, the second round of testing was designed as follows in table 7.

Table 7. Testing Roles for Round 2 of Testing

Name	Task
Jonathan	- Reassess frame with updates - Add/remove buoyancy where required
Adam	- Work with Will to test ability to control vehicle with joystick apparatus through raspberry pi - Ensure that data can be reliably sent and received
Will	- Work with Adam to ensure that depth sensor is operable and meets all of the specifications for the subsystem test.
Jack	- Attach motor fairings to more effectively secure motors to frame - Ensure that vehicle can more easily move in 3 dimensions (vertical thrust is more suitable as compared to test 2).
Andy	- Check that black box is waterproof at 15' for 5 mins - Ensure that surface transducer is well attached and supported - Work with Justin to check that sound detectors can be heard from 1.5 meters away
Justin	- Work with Andy to ensure that sound detectors can be heard from 1.5 meters away

The second iteration of testing was designed to ensure that any changes made on the vehicle between tests was thoroughly vetted to assure the team of the functionality of the vehicle. In concurrence with the first round of testing, notes were taken after the second and any emergency repairs needed were made based on the summarized notes, which were formatted as follows.

- Slight hitch with communication, unintended interference from power in tether prevented efficient communication
    - Proposed solution: Run a second tether cable to shield from interference from power
- New frame design sat flat in water, slight adjustments made with buoyancy
- Vehicle able to dive and resurface with some trouble. Vertical thrusters moved to better location
- Depth sensor not tested
- Sound Detectors integrated onto vehicle, able to pick up signal above water, but unable to collect data underwater. Failure attributed to inadequate waterproofing.
- Concern: placement of depth sensor is important because fluid flow will affect depth reading.
- Motors fit nicely in casings
- Casings have solid attachment to frame.
- Thrust is similar, though it depends on how they are powered
    - Through arduino, more power
    - Interference when power is running through tether
- Cages effectively prevent motor-tether entanglement
- Moving Forward:
    - Integrate cages onto vertical motor thrusters
    - Test vertical movement with black box attached

Given the second test notes, it is prevalent that while not perfect,the vehicle was running at a more than acceptable standard. However, there were some failures, which prompted discussions on possible next steps that the team could potentially take in the future. For example while the implemented code and design of the communications subsystem did not take into account how an ethernet cable sends data. This resulted in inverted waves canceling each other out, which severely hindered the functionality of the communications subsystem. In the future, to fix this issue a shielded cable should have been used instead of an ethernet cable. Another notable failure was the sound detectors failing to collect data while underwater. This was due to the inadequate waterproofing of the sound detectors. Overall, this test was a success and any changes integrated into the vehicle were minimal. Given the teams design goals, this test proved that we had successfully met the goals, and the only thing left was to work on perfecting the kinks in the vehicle before demo day.

7.2 Significant Technical Accomplishments

Throughout the process of developing the vehicle used, the team was successfully able to establish robust communications between the vehicle and the controller. The vehicle was able to collect large amounts of data from multiple sensors, integrate that data, then act on it autonomously. The team designed and built a frame and propulsion system which allowed for seamless navigation in the water. The simulated black box built demonstrated a new use for surface transducers and worked well for the purposes of the project and considering what was desired. These accomplishments did not come without learning opportunities. One notable example would be how an ethernet cable functions, while this initial oversight resulted in the failure of the communications subsystem, a memorable lesson was learned in data transfer and in the future cable selection as a factor will take a higher level of precedence than it had during this project. Another example that comes to mind was designing for purposeful manufacturability. At the beginning, many subsystems were reliant on 3D printing to generate functioning components to be used in the project. However, this was not necessary as some parts that were to be 3D printed were not necessary to 3D print considering they would be cheaper to buy as a pre existing non custom component. One final notable lesson learned was effective waterproofing of electronics. Conformal coating is something no one had experience with; however, due to the nature of the project the team was forced to learn this waterproofing method and overall it was an educational experience we were all happy to have learned.

8 Conclusions

Overall, the Non_Ame team's SABBL-Lite System accomplished nearly every single customer requirement considered at the onset of the project. The few requirements that were not

accomplished failed due to unforeseen hardware problems, which easily could be resolved given more time and money. Two examples of these problems arose in the hydrophones and the tether components. The problem with the hydrophones was that they weren't hydrophones at all, but waterproofed microphones, acquired in the hope of decreasing expenses. They worked in shallow water, but in the pool (with extensive background noise) they failed. Considering these issues, demonstrating the hydrophones' ability to pick up the black box signal in a dry environment was considered a success for the team. Another critical system component which could have benefitted from more development time and funding was the PACT (Power and Communications Tether). Again, the PACT worked in concept and was demonstrated to work on the land, but ended up failing when power was distributed to the motors, as this power created too much interference with communications signal. Again, a simple hardware insufficiency limited the success of the SAABL-Lite.

Other subsystems on the SABBL-Lite system did satisfy all of their customer requirements, but still need substantial work before being suitable in a consumer product. For example, the frame system in its current form is a tubular PVC body. Before this product reaches the market, the frame must be incorporated into a single, hydrodynamic, waterproofed container, capable of storing all of the necessary sensors and control mechanisms inside. A properly scaled iteration of this project would incorporate solar panels for mobile charging, large internal batteries, and a slight positive buoyancy. This way, the vehicle could operate for days or weeks autonomously without needing a recharge, as it would float to the surface and recharge itself, before submerging and continuing the search. Submerging would become a much more efficient endeavor, using ballast tanks either instead of or in tandem with vertical thrusters. Rudders and hydrofoils could also help POLO change its orientation in the water, but this would require further testing to determine which is more efficient.

9 References

[1] Patton, Sean, "Locator for a submerged black box," *techlinkcenter.org* [online], https://techlinkcenter.org/technologies/locator-for-a-submerged-black-box/ [retrieved 10 Oct. 2018]

[2] Wikipedia contributors. "Autonomous Underwater Vehicle," *wikipedia.org* [online], https://en.wikipedia.org/wiki/Autonomous_Underwater_Vehicle [retrieved 11 Dec. 2018]

[3] Industries, A., "4-channel I2C-safe Bi-directional Logic Level Converter," *adafruit industries blog* [online], https://www.adafruit.com/product/757 [retrieved 12 Dec. 2018]

[4] “My SeaPerch: Journal and Images - STEM Legacy,” Google Sites Available: https://sites.google.com/site/stemlegacy/home/seaperch.

[5] “1040 Micro Case,” *Pelican* Available: https://www.pelican.com/us/en/product/cases/micro/1040.

10 Appendix A: Selection of Team Project

This team took note of opportunity for great improvements in black box search methods underwater. To improve the current search methods, a novel approach would have to be faster, less expensive, and more effective at searching a given area. Through the use of specially designed autonomous underwater vehicles in search efforts, search teams will be able to more reliably recover lost flight and voyage data recorders at sea, and potential aircraft and ocean vessel dangers will be addressed before they cause repeated failure. These autonomous underwater vehicles (AUVs) will be launched from ships, given an area to search, and left alone to perform their task without the need for human intervention. The AUVs will be outfitted and programmed with collision avoidance and self tracking sensors and software, along with the capability to communicate with other manned or unmanned search vehicles. Finally, the AUVs will be solar powered, so as to not require refueling during searches, be faster than manned vessels, and be less expensive to procure and operate than current methods

11 Appendix B: Customer Requirements and Technical Specifications

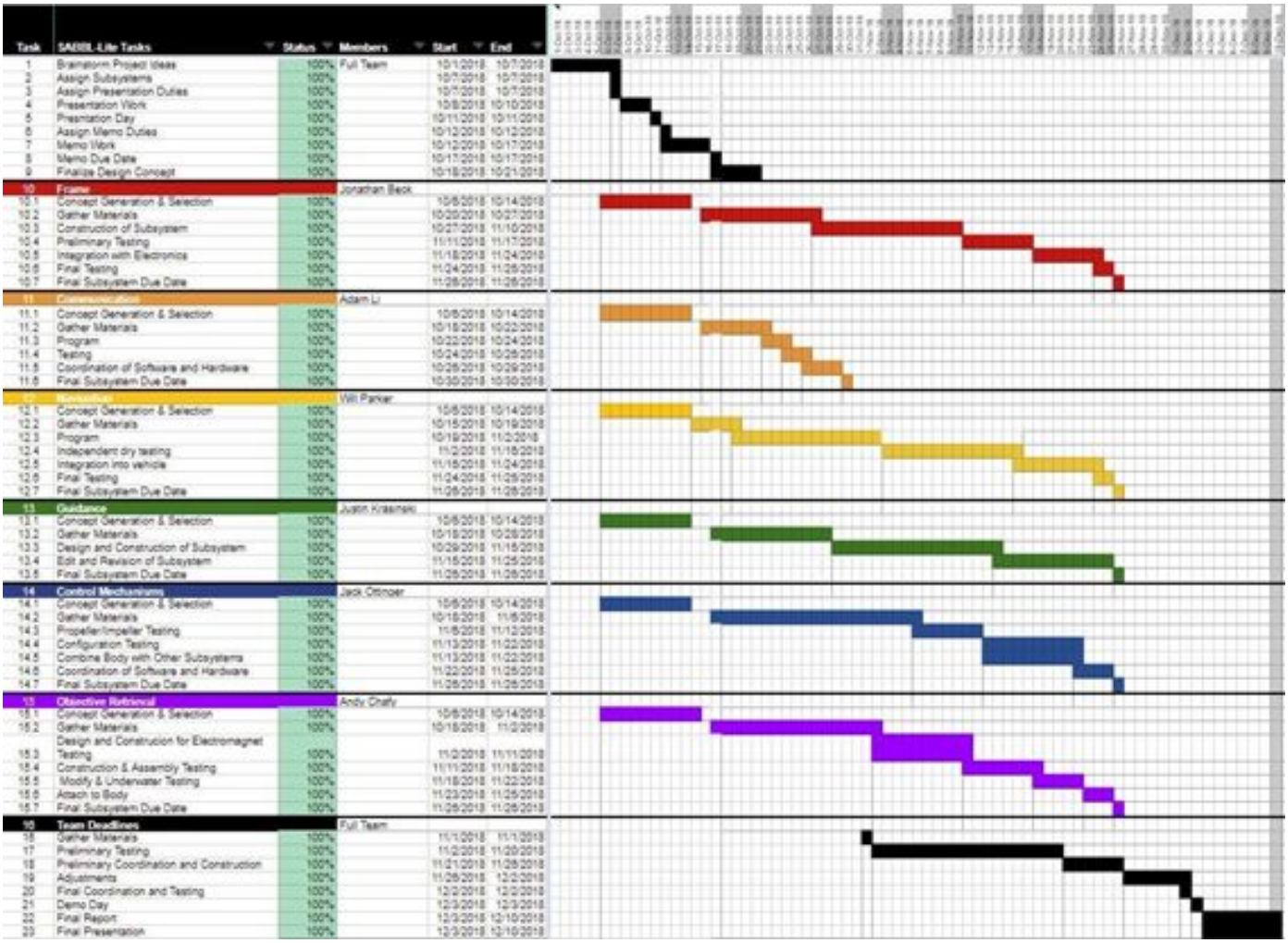
Table 8. Requirements and Associated Specifications by Subsystem Breakdown

Subsystem	Requirement	Specification
Frame	Buoyant	Must have a density < 997 kg/m^3
	Ease of movement via attached propulsion	Must have 4+ locations for propulsion attachment
	Good platform for navigation sensors	0.4 meter front end
	Lightweight	Weight < 7 kg
	Durable	Must survive 50 kpa of force
	Waterproof section for onboard electronics	Waterproof up to 5m
	Inexpensive	Cost < $50
Communications	Can interface with many devices	Supports 12+ sensors and 5+ motors/servos
	Responsive	Communication delay <= 100ms.

	Small physical footprint in frame	Less than 8x8x3 cm
	Reliable	Contains failsafe operation and fallback capability
	Comprehensible	Capable of producing real time visualizations
	Controllable	Support a manual controller that overrides autonomy
	Turns on in water only	A float switch or similar protection is added
Navigation	Must avoid obstacles (walls)	Does not operate < 0.5m from a wall
	Must be able to determine and maintain depth	Able to deduce depth +/- 0.5m - Capable of regulating depth autonomously - Capable of triggering alarm at 3m
	Must be able to roughly determine location/direction	Able to determine heading (direction), translation in x and y to error of <0.5m/min
	Must be able to log navigation data over time	Able to record and correlate amplitude data from sound sensors to location data over time
	Must be inexpensive	Components must cost <$100
Guidance	Path determination from current location to beacon	Able to tell AUV to move in the x-y-z direction
	Autonomous to manual control	When within 1m of the beacon switch to manual
	Identify amplitude of beacon	Be able to pick up 8.5khz/9.5khz frequency.
Control Systems	Will float if the power fails	997 kg/m^3 < Density < 1050 at all times
	Move freely in the pool	One propulsion method per axis
	Maintain a predictable orientation in the water	The sum of torques must be equal to 0
	Waterproof	Waterproof to 5 Meters
	Reasonable Power	1N < Propellor thrust < 3N
	Small	27mm diameter by 46mm long
Object Retrieval	Buoyant	Must have a density < 997 kg/m^3
	Compact	Must create no more than 1 N of drag
	Reliable	Able to retrieve the object at least 75% of the time
	Waterproof section for wiring	Waterproof up to 5m
	Cheap	Cost < $50

12 Appendix C: Gantt Chart

Although some subsections set very rigorous deadlines for themselves in the Gantt Chart as you can see in Figure 26, all tasks were completed on or before their due dates. If it weren't for the Non-Ame team's communication skills (i.e. reminding each other to get stuff done) then this would not have been possible.



Task	SAERL-Lite Tasks	Status	Members	Start	End
1	Brainstorm Project Ideas	100%	Full Team	10/1/2018	10/1/2018
2	Assign Subsystems	100%		10/1/2018	10/1/2018
3	Assign Presentation Duties	100%		10/1/2018	10/1/2018
4	Presentation Work	100%		10/8/2018	10/10/2018
5	Presntation Day	100%		10/11/2018	10/11/2018
6	Assign Memo Duties	100%		10/12/2018	10/12/2018
7	Memo Work	100%		10/12/2018	10/17/2018
8	Memo Due Date	100%		10/17/2018	10/17/2018
9	Finalize Design Concept	100%		10/18/2018	10/21/2018
10	Frame		Jonathan Beck		
10.1	Concept Generation & Selection	100%		10/8/2018	10/14/2018
10.2	Gather Materials	100%		10/20/2018	10/27/2018
10.3	Construction of Subsystem	100%		10/27/2018	11/10/2018
10.4	Preliminary Testing	100%		11/11/2018	11/17/2018
10.5	Integration with Electronics	100%		11/18/2018	11/24/2018
10.6	Final Testing	100%		11/24/2018	11/26/2018
10.7	Final Subsystem Due Date	100%		11/26/2018	11/26/2018
11	Communication		Adam Li		
11.1	Concept Generation & Selection	100%		10/8/2018	10/14/2018
11.2	Gather Materials	100%		10/18/2018	10/22/2018
11.3	Program	100%		10/22/2018	10/24/2018
11.4	Testing	100%		10/24/2018	10/26/2018
11.5	Coordination of Software and Hardware	100%		10/26/2018	10/29/2018
11.6	Final Subsystem Due Date	100%		10/30/2018	10/30/2018
12	Navigation		Will Parker		
12.1	Concept Generation & Selection	100%		10/8/2018	10/14/2018
12.2	Gather Materials	100%		10/15/2018	10/19/2018
12.3	Program	100%		10/19/2018	11/2/2018
12.4	Independent dry testing	100%		11/2/2018	11/18/2018
12.5	Integration into vehicle	100%		11/18/2018	11/24/2018
12.6	Final Testing	100%		11/24/2018	11/26/2018
12.7	Final Subsystem Due Date	100%		11/26/2018	11/26/2018
13	Guidance		Justin Krasinski		
13.1	Concept Generation & Selection	100%		10/8/2018	10/14/2018
13.2	Gather Materials	100%		10/18/2018	10/29/2018
13.3	Design and Construction of Subsystem	100%		10/29/2018	11/18/2018
13.4	Edit and Revision of Subsystem	100%		11/18/2018	11/26/2018
13.5	Final Subsystem Due Date	100%		11/26/2018	11/26/2018
14	Control Mechanisms		Jack Ottmyer		
14.1	Concept Generation & Selection	100%		10/8/2018	10/14/2018
14.2	Gather Materials	100%		10/18/2018	11/6/2018
14.3	Propeller/Impeller Testing	100%		11/6/2018	11/12/2018
14.4	Configuration Testing	100%		11/13/2018	11/22/2018
14.5	Combine Body with Other Subsystems	100%		11/13/2018	11/22/2018
14.6	Coordination of Software and Hardware	100%		11/22/2018	11/26/2018
14.7	Final Subsystem Due Date	100%		11/26/2018	11/26/2018
15	Objective Retrieval		Andy Chaty		
15.1	Concept Generation & Selection	100%		10/8/2018	10/14/2018
15.2	Gather Materials	100%		10/18/2018	11/2/2018
15.3	Design and Construcion for Electromagnet Testing	100%		11/2/2018	11/11/2018
15.4	Construction & Assembly Testing	100%		11/11/2018	11/18/2018
15.5	Modify & Underwater Testing	100%		11/18/2018	11/22/2018
15.6	Attach to Body	100%		11/23/2018	11/26/2018
15.7	Final Subsystem Due Date	100%		11/26/2018	11/26/2018
16	Team Deadlines		Full Team		
16	Gather Materials	100%		11/1/2018	11/1/2018
17	Preliminary Testing	100%		11/2/2018	11/20/2018
18	Preliminary Coordination and Construction	100%		11/21/2018	11/26/2018
19	Adjustments	100%		11/26/2018	12/3/2018
20	Final Coordination and Testing	100%		12/2/2018	12/3/2018
21	Demo Day	100%		12/3/2018	12/3/2018
22	Final Report	100%		12/3/2018	12/10/2018
23	Final Presentation	100%		12/3/2018	12/10/2018

Figure 26. Completed Non_Ame Gantt Chart

13 Appendix D: Expense Report

The goal for the SABBL-Lite Project was prototype completion for under four hundred dollars. After creating an initial design, the projected cost of the prototype was three hundred forty dollars. After breaking components, or realizing that they were not viable and replacing them, the final cost of all materials was just under three hundred sixty dollars. The final bill of materials can be found table 9. Also to note is that the team already owned most of these components, and the actual amount they spent is significantly less than what is reported below.

Table 9. Final Bill of Materials

Subsystem	Function	Part Name	Qty	Price	Total Price	Retailer
Black Box	Buoyancy	pool noodle	1	$1.00	$1.00	N/a
Frame	Joint	1 in. PVC 90 Degree Elbow	1	$1.14	$1.14	HomeDepot
Propulsion	Waterproofing	Film Canister	4	$0.50	$2.00	Amazon
Team	Integration	8 inch ZipTies (50 ct)	1	$2.50	$2.50	
Propulsion	Waterproofing	Paraffin Household Gulf Waxf	1	$2.95	$2.95	Amazon
Team	Integration	Hot Glue Sticks (30 ct)	1	$2.99	$2.99	
Frame	Sealant	2.8 oz. Silicone Sealant	1	$5.24	$5.24	HomeDepot
Frame	Joint	1 in. PVC 90 Degree Elbow (5 pack)	1	$5.58	$5.58	HomeDepot
Team	Wiring	Ethernet Cable (100 ft)	1	$5.95	$5.95	Monoprice
Team	Integration	18 guage Steel wire (110 ft)	1	$6.48	$6.48	Home Depot
Team	Wiring	Ribbon Wire (3 ft)	2	$4.95	$9.90	Mouser
Team	Misc.	3d Printer Filament	1	$9.99	$9.99	Makergeeks
Team	Wiring	Wire (spool)	1	$11.35	$11.35	Home Depot
Frame	Joint	1 in. PVC Tee (4 pack)	1	$11.45	$11.45	HomeDepot
Frame	Joint	1 in. PVC Cross (4 pack)	1	$13.71	$13.71	HomeDepot
Black Box	Container	pelican box	1	$15.00	$15.00	amazon
Communications	Microcontroller	Arduino Mega	1	$15.00	$15.00	Robotshop
Frame	Pipe	1in.x10ft. PVC Plain-End Pipe	2	$7.94	$15.88	HomeDepot
Frame	Controls Box	Pelican case	1	$16.20	$16.20	Amazon
Navigation	IMU	IMU	1	$16.92	$16.92	Amazon
Propulsion	Propellor	1.8 inch water propeller	4	$4.23	$16.92	RobotShop
Black Box	Signal generator	daytona surface transducer	2	$10.00	$20.00	amazon
Black Box	Weight	steel plate	2	$10.00	$20.00	home depot
Propulsion	Motor	12V DC Motor	4	$6.52	$26.08	Amazon
Black Box	Signal generator	elegoo	1	$35.00	$35.00	amazon
Communications	Microcomputer	Raspberry Pi	1	$35.00	$35.00	
Navigation	Depth Sensing	Pressure Sensor	1	$35.89	$35.89	Amazon
				Total	***$358.76***	

14 Appendix E: Team Members and Their Contributions

14.1 Jonathan Beck

Jonathan was the frame subsystem lead. He took charge of designing and fabricating the frame as well as balancing and achieving proper buoyancy of the frame in testing. In addition to being frame lead, Jonathan was also the point of contact between the Non_Ame team and the pool facilities coordinator Joe Vrablic. Jonathan organized and scheduled all of the teams pool testing dates.

14.2 Jack Ottinger

Jack was the Chief Propulsion Systems Engineer at Non_Ame Special Research Group. He also acted as the Director of Photography, and as the Keeper of the Budgets. He was in charge of the design, as well as research and development for the motors, propellers, and casings. Designed the waterproof casings for the motors, as well as the fairings and cages.

14.3 Justin Krasinski

Justin was the guidance subsystem lead. Was in charge of researching what type of microphones to use, and eventually ordering them. Researched the best way to waterproof electronic equipment outside of the waterproof box. During testing video recorded all the test for documentation in the event of a failed subsystem during demonstration day.

14.4 Andy Chafy

Andy was the object retrieval subsystem lead. He was responsible for developing the simulated black box and the mechanism by which the vehicle picked up that black box.

14.5 Adam Li

Adam was the communications subsystem lead. He developed an interface which allowed the vehicle to be controlled and regulated through a series of microcontroller interfaces. This subsystem was designed to allow the vehicle to be controlled while also collecting and processing data.

14.6 Will Parker

Will was the navigation subsystem lead for the Non_Ame team. He researched methods of location determination underwater, and did extensive testing on his subsystem components. Because of his prior experience in the development of underwater vehicles, Will was able to

provide insights and offerer advice and recommendations at various stages in the development of the frame, propulsion, and object retrieval subsystems.

15 Appendix F: Statement of Work

Activity 10-1 Statement of Work
Statement of Work (10/1/2018)

Team
Jonathan Beck, Justin Krasinski, Will Parker, Andy Chafy, Adam Li

Semester Objectives
1. Design, construct, and deliver a proof of concept for a submersible device that is able to locate black boxes deep underwater.
2. Perform analysis on current systems as well as systems currently in the research stages in order to better design the final product.
3. Create a working land-based prototype for proof of concept purposes.
4. Generate ideas to generalize the basic design to work for other triangulated signal-based searches
5. Research and evaluate advanced solutions that could be implemented in the real world, including automation, advanced sensors, and improved longevity.

Approach
Develop a simple and robust base vehicle. Design a system which can determine relative amplitude of flight data recorder pings. Combine these components into a system that can be quickly deployed in swarm, allowing autonomous vehicles to aid in determining the position of a lost flight data recorder.

Deliverables and Dates
1. System Concept Review presentation (10/11)
2. (basically informal) preliminary design review (10/15)
3. Memos due (10/17)
4. Fabricate and demonstrate proof-of-concept (11/5, 11/8)
5. Project Demos (12/3)
6. Team Presentation (12/10)
7. Report & Peer Evaluations (12/12) (peer eval. after report)

16 Appendix G: Lessons Learned

ISTJ WP	ISFJ	INFJ JO	INTJ
ISTP JK	ISFP	INFP AL	INTP AC
ESTP	ESFP	ENFP	ENTP
ESTJ	ESFJ	ENFJ JB	ENTJ

Figure 29. Myers Briggs Table for Non_Ame Team

Overall , we worked very well as a team. It is clear that we are mostly type P as shown in the MBTI chart in Figure 29. While in a big group project this is usually a flaw due to the lack of planning that would be prevalent, this worked out very well for us. We left the initial planning up to the type J group members and the initial meetings were meant to build momentum. Afterwards the team was incredibly excited to work on the project, which is a great strategy to use when working with type P people. The general excitement and atmosphere of our group made it very easy for everyone to want to work on the project, this strategy effectively utilized the type P individuals who usually do things based on feeling. Having a majority type P group made our meeting times and overall efficiency very adaptable. For example for the milestone 1 memo due to our majority type P we were able to effectively reformat our essay one hour before it was due, something that type Js may struggle with. The type N personality trait was also valued especially during testing. If something failed during testing, someone usually had an ad lib solution. One final positive thing to note is that all but one of the mbti types in our group are often associated with open mindedness, a quality that is extremely valuable when going through the design process. Overall, our team worked together very well. However, there would be some new strategies that could be employed to make a team like ours more effective in the future. For example, we started off quite slow initially, mostly due to the fact that we often found ourselves gridlocked in design concepts dude to the number of different solutions we fabricated for our problem. This might be addressed by adding an ENTJ to our group, a personality type often associated with skillful executive decision making. Overall generating moral at the beginning was the essential problem, however once the vision was seen by everyone each of our individual personality types quickly formed a high functioning talented team. In conclusion, our team worked together very well and we all agreed that we were lucky to get each other as group partners.

17 Appendix H: Software Breakdown of Communications

17.1 Overview of Software Setup

For the software side of communications, ROS was used as the primary platform to simplify the integration of various programs. ROS, short for Robot Operating System, is a set of software libraries and tools that act as a platform for robotics. All of the code for this platform is licensed under the 3 clause BSD license, which can be found in Appendix I. As a platform, ROS provides a number of tools for ROS packages to interact with each other and the hardware system through a "messaging" system, where package nodes can publish or subscribe to topics with structured message types. One of the key external packages used that was not part of the core ROS code was ROSSerial, which handled the communication between the Arduino Mega and the Raspberry Pi. All of this was installed on top of the Raspbian Distribution for the Raspberry Pi.

On the Arduino, the use of C++ classes was heavily taken advantage of, with each class specifically handling a singular key software subsystem and various functions within the class acting as hooks for the main loop or other classes to interact with the specific class through. In figure 28 below, an overview diagram of the intended software setup can be found. As communications could not be completed, the software development aspect of communications was not completed.

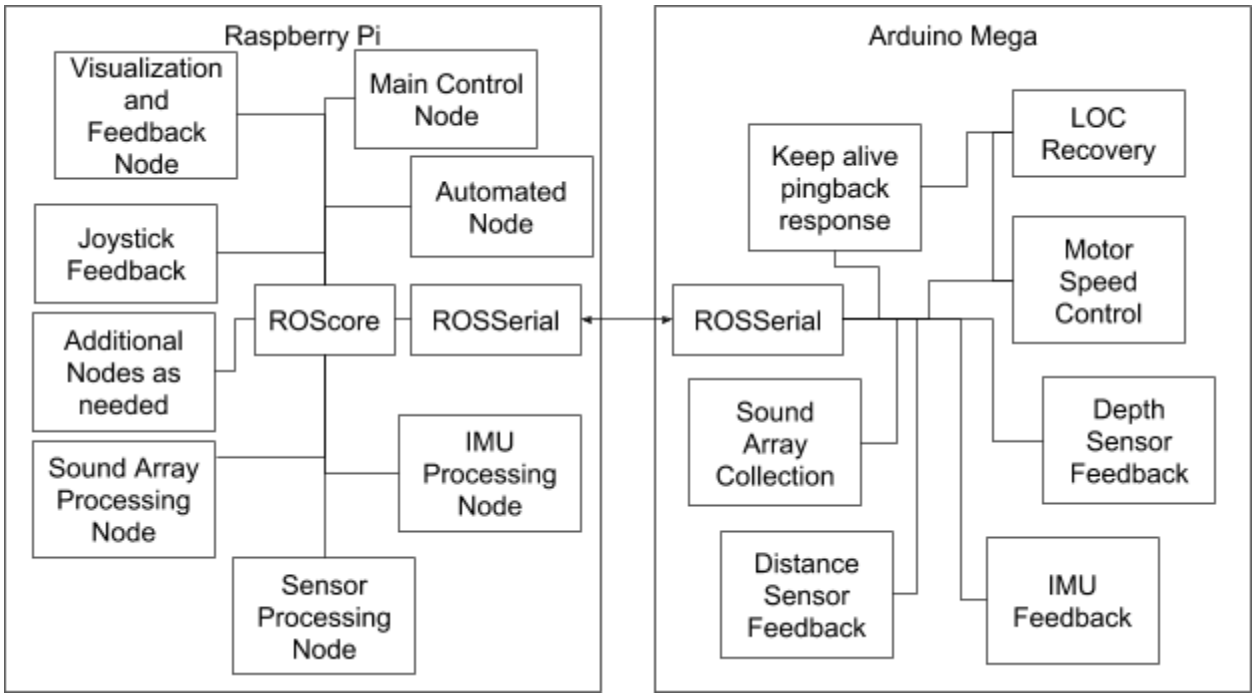


Figure 28: Diagram of Intended System Setup

17.2 Description of Software Components on the Raspberry Pi

On the Pi, software was developed in the form of nodes and messages, taking advantage of the vast platform offered by ROS. Python was the primary language used on the Pi. All external ROS packages used are license under the BSD license in Appendix I.

17.2.1 ROScore

This is the core program of ROS. It provides the hook-ins for all ROS nodes and handles cross communication through the messaging system. All nodes communicate through ROScore. As such all messages are accessible by any node.

17.2.2 ROSSerial

This node is a packet from ROS. It exists in two parts, one of which is the server on the Raspberry Pi, and the other is a class that defines an object on the Arduino Mega with hook-ins for other code to communicate to the Raspberry Pi server. The Raspberry Pi then opened up the messaging channels requested from the Mega and handles the publishing and forwarding of data to and from the Mega.

17.2.3 Joystick Feedback

Joystick Feedback was obtained through the use of the Joy package, which provided an interface with the attached joystick and published the feedback from the joystick to the ROS system.

17.2.4 Main Control Node

This node handles all core functionality required to maintain communication with the Arduino Mega, such as making sure to send KeepAlive packets at regular intervals, as well as acting on the data received from the other Nodes. It takes precedence over the Automated Node so that any manual input immediately overrides any automation.

17.2.5 Automated Node

All automation and autonomous functionality would be handled through this node. It reads the data returned by the processing nodes to determine actions.

17.2.6 Sound Array Processing Node

This node would handle the fast fourier transform of the audio received and work in conjunction with a localization node (not displayed) to determine object location as well as make public the relevant sound data.

17.2.7 IMU Processing Node

This node would handle the processing of the IMU data to a more standardized message since the IMU data was compressed in communication between the Mega and the Pi. It would also handle the processing of the distance and depth sensors in conjunction with the ros package, robot_localization (not displayed), to determine the location of the vehicle in water.

17.2.8 Sensor Processing Node

Any additional sensors would be processed through this/these nodes.

17.2.9 Visualization and Feedback Node

All user feedback would be provided through this node, and it would handle the whole system setup and use on the software end.

17.3 Description of Software Components on the Arduino Mega

On the Mega, software was developed in terms of classes and object, taking advantage of the object oriented parts of C++ to modularize development of code. ROSSerial as a component maintains the same description from the Pi, which can be found in section 17.2.2 above.

17.3.1 LOC Recovery and Keep alive pingback response

The LOC Recovery component, short for Loss-Of-Communication Recovery was not developed as an isolated object for the reason of simplicity. It worked in tandem with the Keep alive pingback component, also not developed as an isolated object, to ensure communication continued to exist with the Pi. If a long enough time had gone by before the next keep alive packet, the LOC Recovery system begins thrusting up to bring the system to the surface.

17.3.2 Motor Speed Control

This component of the Mega system listened for motor control signals from the Pi. As all automation was handled on the Pi, with the Mega acting as the interface to hardware, motors speeds were obtained from messages sent from the Pi. This component also opened up an internal interface to LOC Recovery so that built in recovery systems can access the motors and control them when communication with the Pi is lost. The Adafruit Motor Shield v2 Library, licensed under the BSD license in Appendix I, was used to control the motor controller.

17.3.3 Depth Sensor Feedback

The depth sensor would undergo minor preprocessing here before the data was published over the ROSSerial object on the Mega. This components isolated the depth sensor codes and published readings to the Pi when called on from the main loop in the Mega. The SparkFun Pressure Sensor Breakout - MS5803-14BA Arduino Library was used to interface with the

sensor used, and it is license under the Creative Commons Share-alike 3.0 license. A summary of this license can be found in Appendix J.

17.3.4 Sound Array Collection

When called on this component stops all movement on the vehicle to obtain a sample of sound readings in the water. This array of sound is then communicated back to the Pi as a large array of data.

17.3.5 Distance Sensor Feedback

This components polls the distance sensors for data and sends it back as a list to the Pi.

17.3.6 IMU Feedback

The 3-axis accelerometer, gyroscope, and magnetometer are polled in this object, and their data compressed into a simple 3x3 array to send back to the Pi for processing when called on. The IMU was interfaced through the Adafruit LSM9DS1 Library, licensed under the BSD license which can be found in Appendix I.

17.4 Completed Software

Due to the fact communications fell apart when integration was tested, only code remained incomplete across both platforms. On the Raspberry Pi, only the main control node was partially implemented with test programs made for certain nodes and subsystems. These test programs can be found in Appendix K. For the Arduino Mega, the Distance Sensor Feedback node was the only unwritten node, with the others being either completed or partially completed.

17.5 Special Note on ROS Messages

As ROS uses a messaging system with structured messages, these messages and the data they contain have to be defined prior to use. These messages are defined in plaintext documents in the order and structure of datatypes used. Message definitions will be included with the Pi code in the next section.

17.6 Completed Pi Code

17.7.1 Main Control Node

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

# This is the main user control interface for sabbl

import rospy
import math
from sensor_msgs.msg import Joy
from sabbl_msgs.msg import UInt8List
```

```
from std_msgs.msg import Empty

manualControl = True
lastButtonTime = 0 #1 sec debounce, I'm too lazy to deal with additional vals
lastButtonState = 0 #too lazy to do better too, this prevents rapid toggling

invertY = 1

verboseOut = False

max_power = (460 / 1020.0) * 4

msg = UInt8List()
msg.data = [0,0,0,0,0] #[directions, LMot, RMot, VLMot, VRMot]

#the subscribers and publishers
pub = rospy.Publisher('Mot_Ctrl', UInt8List, queue_size=1)

def isclose(a, b, rel_tol=1e-09, abs_tol=0.0):
    return abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)

def callback(data):
    global manualControl, lastButtonState, lastButtonTime, invertY, verboseOut, max_power, msg
    #print(data.axes);
    #print(invertY);
    # toggle manual control
    if data.buttons[1] is 1 and lastButtonState is 0:
        if (data.header.stamp.secs > lastButtonTime + 1):
            manualControl = not manualControl
            print("Manual Control: " + str(manualControl))
        lastButtonTime = data.header.stamp.secs
    lastButtonState = data.buttons[1]

    if data.buttons[10] is 1:
        invertY = -invertY
        print("Invert Y: " + str(invertY is -1))

    if data.buttons[6] is 1:
        verboseOut = not verboseOut
        print("Verbose out: " + str(verboseOut))

    if not manualControl: return

    #because I'm using the alt value temporarily, this is to shim the variable name
    vert_pow = data.axes[3] * invertY;

    #reset directions
    msg.data[0] = 0
    #calculate all control values
    #first map forward-backward and left-right rot into circle from square
    #unfortunately my math is reversed for horizontal x axis
    horiz_x = -data.axes[2] * math.sqrt(1 - (data.axes[1]**2) / 2)
    horiz_y = data.axes[1] * math.sqrt(1 - (data.axes[2]**2) / 2)
    #then calculate thetas and such to get powers with atan2
    theta = math.atan2(horiz_y, horiz_x)
    radius = math.sqrt(horiz_x**2 + horiz_y**2)
    if (radius > 1.0): radius = 1.0
    right_mot = math.sin(theta - math.pi/4) * radius
    left_mot = math.cos(theta - math.pi/4) * radius
    #now map left_right and up_down
    vert_x = data.axes[0] * math.sqrt(1 - (vert_pow**2) / 2)
    vert_y = vert_pow * math.sqrt(1 - (data.axes[0]**2) / 2)
    theta = math.atan2(vert_y, vert_x)
    radius = math.sqrt(vert_x**2 + vert_y**2)
    if (radius > 1.0): radius = 1.0
```

```
    vright_mot = -math.sin(theta - math.pi/4) * radius
    vleft_mot = -math.cos(theta - math.pi/4) * radius
    #scale power
#    print(max_power)
    tot_power = abs(right_mot) + abs(left_mot) + abs(vleft_mot) + abs(vright_mot)
#    print(tot_power)
    scale = 1 if (max_power > tot_power) else max_power / tot_power
    left_mot *= scale
    right_mot *= scale
    vleft_mot *= scale
    vright_mot *= scale
#    print(scale)
    #update vector
    #(this first part is just multiplying the bool result to the appropriate bit)
    msg.data[0] += ((left_mot < 0.0)*1) * 8
    msg.data[0] += ((right_mot < 0.0)*1) * 4
    msg.data[0] += ((vleft_mot < 0.0)*1) * 2
    msg.data[0] += ((vright_mot < 0.0)*1)
    msg.data[1] = int(abs(left_mot*255))
    msg.data[2] = int(abs(right_mot*255))
    msg.data[3] = int(abs(vleft_mot*255))
    msg.data[4] = int(abs(vright_mot*255))
    #if verbose out
    if verboseOut:
        print([int(left_mot*255), int(right_mot*255), int(vleft_mot*255), int(vright_mot*255)])
    pub.publish(msg)

def recover(data):
    pub.publish(msg)

def controller():
    global manualControl, invertY, verboseOut

    rospy.init_node('sabbl_controller')
    rospy.Subscriber('joy', Joy, callback)
    rospy.Subscriber('keepalive_recover', Empty, recover)

    print("Manual Control: " + str(manualControl))
    print("Invert Y: " + str(invertY is -1))
    print("Verbose out: " + str(verboseOut))

    pub = rospy.Publisher('keepalive', Empty, queue_size=10)
    while not rospy.is_shutdown():
        pub.publish()
        rospy.sleep(.5)

if __name__ == '__main__':
    try:
        controller()
    except rospy.ROSInterruptException:
        pass
```

17.6.2 sabbl_msgs\RawImu.msg

```
Header header

float32[3] orientation
float32[3] angular_velocity
float32[3] linear_acceleration
```

17.6.3 sabbl_msgs\SensorArray.msg

```
Header header

float32[] readings
```

17.6.4 sabbl_msgs\UInt8List.msg

```
uint8[] data
```

17.6.5 sabbl_msgs\UInt16List.msg

```
uint16[] data
```

17.6.6 Command used to launch ROSSerial

```
# rosrun rosserial_python serial_node.py _port:=/dev/Serial0 _baud:=1000000
```

17.7 Completed Mega Code

17.7.1 sabbl.ino

```
#include "ros.h"
#include <std_msgs/Empty.h>
#include <std_msgs/String.h>
#include "IMU.h"
#include "MotorController.h"
#include "PressureSensor.h"
#include "SoundDetectorArray.h"

ros::NodeHandle nh;

std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

Sabbl::IMU imu(nh);
Sabbl::MotorController motors(nh);
Sabbl::PressureSensor depth(nh);
Sabbl::SoundDetectorArray sound(nh, motors);

char hello[13] = "hello world!";

unsigned long lastTime = 0;
void pingCallback(const std_msgs::Empty& ping){
  lastTime = millis();
}
void soundCallback(const std_msgs::Empty& empty){
  sound.sendReading();
}

std_msgs::Empty empty;

ros::Subscriber<std_msgs::Empty> ping("keepalive", pingCallback);
ros::Subscriber<std_msgs::Empty> getSound("getSound", soundCallback);
```

```
ros::Publisher keepalive_recover("keepalive_recover", &empty);
void setup()
{
  nh.initNode();
  //nh.advertise(keepalive_recover);
  //nh.advertise(chatter);
  imu.begin();
  motors.begin();
  depth.begin();
  //sound.begin();
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  nh.subscribe(ping);
  //nh.subscribe(getSound);
  lastTime = millis();
}

void loop()
{
  while(!motors.safeToSendData());
  //str_msg.data = hello;
  //chatter.publish( &str_msg );
  imu.run();
  depth.run();
  unsigned long time = millis();
  while(time + 500 > millis()){
    /*if (!nh.connected()){
      motors.set(0,0,0,0,0);
    }*/
    if (lastTime + 1500 < millis()){
      if (lastTime + 3000 < millis()){
        //recover HERE
        digitalWrite(13, HIGH);
        motors.set(0,0,0,255,255);
      } else{
        //giver communication a chance to recover
        motors.set(0,0,0,0,0);
        //keepalive_recover.publish(&empty);
      }
    }
    nh.spinOnce();
  }
  //delay(100);
}
```

17.7.2 ArduinoHardware.h

```
#ifndef ROS_ARDUINO_HARDWARE_H_
#define ROS_ARDUINO_HARDWARE_H_

#if ARDUINO>=100
  #include <Arduino.h>
#else
  #include <WProgram.h>
#endif

#include <HardwareSerial.h>
#define SERIAL_CLASS HardwareSerial

class ArduinoHardware
{
  public:
    ArduinoHardware()
```

```
    {
      iostream = &Serial1;
      baud_ = 1000000;
    }

    void setBaud(long baud)
    {
      this->baud_= baud;
    }

    int getBaud()
    {
      return baud_;
    }

    void init()
    {
      iostream->begin(baud_);
    }

    int read()
    {
      return iostream->read();
    };

    void write(uint8_t* data, int length)
    {
      for(int i=0; i<length; i++)
      {
        iostream->write(data[i]);
      }
    }

    unsigned long time()
    {
      return millis();
    }

  protected:
    SERIAL_CLASS* iostream;
    long baud_;
};

#endif
```

17.7.3 IMU.h

```
#ifndef SABBL_IMU
#define SABBL_IMU

#include "ros.h"
#include <sabbl_msgs/RawIMU.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>

namespace Sabbl{
class IMU {
public:
  IMU(ros::NodeHandle &nh) : lsm(), rawIMU("IMU_raw", &msg){
    this->nh = &nh;
  }
```

```
  void begin(){
    (*nh).advertise(rawIMU);

    lsm.begin();

    lsm.setupAccel(lsm.LSM9DS1_ACCELRANGE_2G);
    lsm.setupMag(lsm.LSM9DS1_MAGGAIN_4GAUSS);
    lsm.setupGyro(lsm.LSM9DS1_GYROSCALE_245DPS);
  }

  void run(){
    sensors_event_t a, m, g, t;

    lsm.getEvent(&a, &m, &g, &t);

    msg.orientation[0] = m.data[0];
    msg.orientation[1] = m.data[1];
    msg.orientation[2] = m.data[2];

    msg.angular_velocity[0] = g.data[0];
    msg.angular_velocity[1] = g.data[1];
    msg.angular_velocity[2] = g.data[2];

    msg.linear_acceleration[0] = a.data[0];
    msg.linear_acceleration[1] = a.data[1];
    msg.linear_acceleration[2] = a.data[2];

    msg.header.stamp = nh->now();

    rawIMU.publish(&msg);
  }
private:
  Adafruit_LSM9DS1 lsm;
  ros::Publisher rawIMU;
  sabbl_msgs::RawIMU msg;
  ros::NodeHandle* nh;

};
};

#endif
```

17.7.4 MotorController.cpp

```
#include "MotorController.h"

#include "ros.h"
#include <sabbl_msgs/UInt8List.h>
#include <Wire.h>
#include <Adafruit_MotorShield.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield();
unsigned long motor_lastTime = 0;

#define SAFE_DELAY 10 //1ms needed after motor change for clean data

namespace Sabbl{
MotorController::MotorController(ros::NodeHandle &nh)
                                : mot_ctrl("Mot_Ctrl", MotorControllerCallback),
                                  nh(&nh){}

void MotorController::begin(){
```

```
  nh->subscribe(mot_ctrl);

  AFMS.begin();
}

void MotorController::set(uint8_t directions, uint8_t left, uint8_t right, uint8_t v_left, uint8_t
v_right){
  uint8_t params[5] = {0, left, right, v_left, v_right};
  MotorControllerControl(directions, params);
}

bool MotorController::safeToSendData(){
  return millis() > motor_lastTime + SAFE_DELAY;
}
};

void MotorControllerControl(uint8_t directions, const uint8_t* params){
  digitalWrite(13, HIGH);
  motor_lastTime = millis();
  for (int i = 0; i < 4; i++){
    AFMS.getMotor(4-i)->run((directions&0x01)+1);
    AFMS.getMotor(4-i)->setSpeed(params[4-i]);
    /*Serial.print(4-i);
    Serial.print(": ");
    Serial.print((directions&0x01) ? "Backwards" : "Forwards");
    Serial.print(" at ");
    Serial.println(params[4-i]);*/
    directions >>= 1;
  }
  digitalWrite(13, LOW);
}

void MotorControllerCallback(const sabbl_msgs::UInt8List& powers){
  if (powers.data_length != 5) return;
  uint8_t dir = powers.data[0];
  MotorControllerControl(dir, powers.data);
}
```

17.7.5 MotorController.h

```
#ifndef SABBL_MOTOR_CONTROLLER
#define SABBL_MOTOR_CONTROLLER

#include "ros.h"
#include <sabbl_msgs/UInt8List.h>

namespace Sabbl{
class MotorController {
public:
  MotorController(ros::NodeHandle &nh);
  void begin();
  void set(uint8_t directions, uint8_t left, uint8_t right, uint8_t v_left, uint8_t v_right);
  bool safeToSendData();
private:
  ros::Subscriber<sabbl_msgs::UInt8List> mot_ctrl;
  ros::NodeHandle* nh;
};
};

void MotorControllerCallback(const sabbl_msgs::UInt8List& powers);
void MotorControllerControl(uint8_t directions, const uint8_t* params);
```

```
#endif
```

17.7.6 PressureSensor.cpp

```
#include "PressureSensor.h"

#include <Wire.h>
#include <SparkFun_MS5803_I2C.h>

// Begin class with selected address
// available addresses (selected by jumper on board)
// default is ADDRESS_HIGH

//  ADDRESS_HIGH = 0x76
//  ADDRESS_LOW  = 0x77

MS5803 sensor(ADDRESS_HIGH);

namespace Sabbl{

PressureSensor::PressureSensor(ros::NodeHandle &nh) : fluidPressure("FluidPressure", &msg){
  this->nh = &nh;
}

void PressureSensor::begin(){
  (*nh).advertise(fluidPressure);

  sensor.reset();
  sensor.begin();
}

void PressureSensor::run(){
  msg.header.stamp = nh->now();
  msg.fluid_pressure = ((float)sensor.getPressure(ADC_4096)*100.0);

  fluidPressure.publish(&msg);
}
};
```

17.7.7 PressureSensor.h

```
#ifndef SABBL_PRESSURE_SENSOR
#define SABBL_PRESSURE_SENSOR

#include "ros.h"
#include <sensor_msgs/FluidPressure.h>

namespace Sabbl{
class PressureSensor {
public:
  PressureSensor(ros::NodeHandle &nh);
  void begin();
  void run();
private:
  ros::Publisher fluidPressure;
  sensor_msgs::FluidPressure msg;
  ros::NodeHandle* nh;

};
```

```
};

#endif
```

17.7.8 SoundDetectorArray.cpp

```
#include "SoundDetectorArray.h"

#include "ros.h"
#include <sabbl_msgs/UInt16List.h>

namespace Sabbl{

SoundDetectorArray::SoundDetectorArray(ros::NodeHandle &nh, Sabbl::MotorController &motors) :
audioTransfer("audio_transfer", &msg), audioTransfer2("audio_transfer2", &msg){
  this->nh = &nh;
  this->motors = &motors;
  msg.data_length = 48;
}

void SoundDetectorArray::begin(){
  nh->advertise(audioTransfer);
  nh->advertise(audioTransfer2);
}


void SoundDetectorArray::sendReading(){
  motors->set(0,0,0,0,0);
  while(!motors->safeToSendData());

  for(uint8_t i = 0; i < 48; i++){
    audio_data[i] = analogRead(0);//readADC();
    //delayMicroseconds(500-13);
    //Approx 10,000 hz
  }

  msg.data = audio_data;
  audioTransfer.publish(&msg);

  for(uint8_t i = 0; i < 48; i++){
    audio_data[i] = analogRead(1);//readADC();
    //delayMicroseconds(500-13);
    //Approx 10,000 hz
  }

  msg.data = audio_data;
  audioTransfer2.publish(&msg);
}

};
```

17.7.9 SoundDetectorArray.h

```
#ifndef SABBL_SOUND_DETECTOR_ARRAY
#define SABBL_SOUND_DETECTOR_ARRAY

#include "ros.h"
#include <sabbl_msgs/UInt16List.h>
#include "MotorController.h"
```

```
namespace Sabbl{
class SoundDetectorArray {
public:
  SoundDetectorArray(ros::NodeHandle &nh, Sabbl::MotorController &motors);
  void begin();
  void sendReading();
private:
  ros::Publisher audioTransfer;
  ros::Publisher audioTransfer2;
  sabbl_msgs::UInt16List msg;
  uint16_t audio_data[48];
  ros::NodeHandle* nh;
  Sabbl::MotorController* motors;
};
};

#endif
```

17.7.10 ros.h

```
#ifndef _ROS_H_
#define _ROS_H_

#include <ros/node_handle.h>
#include "ArduinoHardware.h"

namespace ros
{
  // default is 25, 25, 512, 512
  typedef NodeHandle_<ArduinoHardware, 15, 15, 128, 256> NodeHandle;
}

#endif
```

18 Appendix I: The 3-clause BSD License

The 3-Clause BSD License

SPDX short identifier: BSD-3-Clause

Note: This license has also been called the "New BSD License" or "Modified BSD License". See also the 2-clause BSD License.

Copyright <YEAR> <COPYRIGHT HOLDER>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Source [https://opensource.org/licenses/BSD-3-Clause]

19 Appendix J: Creative Commons Attribution-ShareAlike 3.0 Unported Human Readable Summary

This is a human-readable summary of (and not a substitute for) the license.

You are free to:

Share — copy and redistribute the material in any medium or format
Adapt — remix, transform, and build upon the material for any purpose, even commercially.

This license is acceptable for Free Cultural Works.
The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Source [https://creativecommons.org/licenses/by-sa/3.0/]

20 Appendix K: Code Used for Testing

20.1 Latency Test Code

20.1.1 Code on the Pi

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

import rospy
from std_msgs.msg import Empty, String

pingTime = count = 0

def callback(data):
    global count, pingTime
    print("%d: %fms" % (count, (rospy.get_time() - pingTime)*1000))

def ping():
    global count, pingTime
    pub = rospy.Publisher('ping', Empty, queue_size=10)
    rospy.init_node('pinger')
    rospy.Subscriber('pingback', Empty, callback)
    rospy.sleep(1)
    while not rospy.is_shutdown():
        pingTime = rospy.get_time()
        count += 1
        pub.publish()
        rospy.sleep(.5)

if __name__ == '__main__':
    try:
        ping()
    except rospy.ROSInterruptException:
        pass
```

20.1.2 Code on the Mega

ArduinoHardware.h and ros.h are retained from the completed code in Appendix H, section 17.7.

```
#include "ros.h"
#include <std_msgs/Empty.h>

ros::NodeHandle nh;

std_msgs::Empty packet;
ros::Publisher pingback("pingback", &packet);

void pingCallback(const std_msgs::Empty& ping){
  pingback.publish( &packet );
  digitalWrite(13, HIGH);
  delayMicroseconds(50);
  digitalWrite(13, LOW);
}

ros::Subscriber<std_msgs::Empty> ping("ping", pingCallback);

void setup()
{
  pinMode(13, OUTPUT);
  nh.initNode();
```

```
  nh.advertise(pingback);
  nh.subscribe(ping);
}

void loop()
{
  nh.spinOnce();
}
```

20.2 Bandwidth Test Code

20.2.1 Code on the Pi

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

import random
import string
import rospy
from std_msgs.msg import Empty, String

pingTime = count = total = 0
global_time = 0
packet_size = 100 # n bytes = n*8 bits

pub = rospy.Publisher('bandwidth', String , queue_size=10)

def callback(data):
    global count, pingTime, global_time, total, pub

    global_time += (rospy.get_time() - pingTime)*1000
    total += packet_size
    est = total * 8 / global_time
    print("%d bytes, %d total, %fms in transfer, %fkbps avg" % (packet_size, total, global_time, est))

    packet_data = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in
range(packet_size))
    count += 1
    pingTime = rospy.get_time()
    pub.publish(packet_data)

def ping():
    global count, pingTime, pub
    rospy.init_node('bandwidth_test')
    rospy.Subscriber('bandwidth_back', Empty, callback)
    rospy.sleep(1)

    packet_data = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in
range(packet_size))
    count += 1
    pingTime = rospy.get_time()
    pub.publish(packet_data)
    rospy.spin()

if __name__ == '__main__':
    try:
        ping()
    except rospy.ROSInterruptException:
        pass
```

20.2.2 Code on the Mega

```
#include "ros.h"
#include <std_msgs/Empty.h>
#include <std_msgs/String.h>

ros::NodeHandle nh;

std_msgs::Empty packet;
ros::Publisher pingback("bandwidth_back", &packet);

void pingCallback(const std_msgs::String& ping){
  pingback.publish( &packet );
  digitalWrite(13, HIGH);
  delayMicroseconds(50);
  digitalWrite(13, LOW);
}

ros::Subscriber<std_msgs::String> ping("bandwidth", pingCallback);

void setup()
{
  pinMode(13, OUTPUT);
  nh.initNode();
  nh.advertise(pingback);
  nh.subscribe(ping);
}

void loop()
{
  nh.spinOnce();
}
```

20.3 Audio Transfer with FFT Test Code

20.3.1 Code on the Pi

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

import rospy
import numpy as np
from sabbl_msgs.msg import UInt16List

def callback(data):
    part_a = np.array(data.data, int)
    part_a -= 512
    #part_b = np.empty(96, int)
    #for i in range(96):
        #part_a[i] = ord(data.data[i])
# // 16
        #part_b[i] = ord(data.data[i]) #np.remainder(ord(data.data[i]), 16)
    #part_a[:] = ord(data.data)
    #part_b[:] = ord(data.data)
    #part_a = part_a // 16
    #part_b = np.remainder(part_b, 16)
#    print (part_a)
    wavedata = part_a #np.empty((part_a.size + part_b.size,), dtype=int)
    #wavedata[0::2] = part_a
    #wavedata[1::2] = part_b
```

```
    w = np.fft.fft(wavedata)
    sampling_rate = 9800
    freqs = np.fft.fftfreq(len(w))
    ps = np.abs(w)**2
    #idx = np.argpartition(np.abs(w), -2)[-2:]
    #idx = idx[0]
    #freq = abs(freqs[idx] * sampling_rate)
    #print('{f} at {p}'.format(f=freq,p=ps[idx]))
    avg = 0
    for coef, freq in zip(ps, freqs):
        if coef and (abs(freq*sampling_rate) > 600) and (abs(freq*sampling_rate) < 3400):
            print('{c} \t for {f}'.format(c=int(coef/100),f=abs(freq*sampling_rate)))
        if (abs(freq*sampling_rate) > 3400):
            print('\r\n\r\n')
            break
             #avg += coef
#            avg = coef
#    print(int(coef/100));

def run():
    rospy.init_node('test_audio')
    rospy.Subscriber('audio_transfer', UInt16List, callback)
    rospy.spin()

if __name__ == '__main__':
    try:
        run()
    except rospy.ROSInterruptException:
        pass
```

20.3.2 Code on the Mega

```
#include "ros.h"
#include <sabbl_msgs/UInt16List.h>

ros::NodeHandle nh;

sabbl_msgs::UInt16List output;
ros::Publisher msg("audio_transfer", &output);

uint16_t data[128];

void setup(){
  nh.initNode();
  nh.advertise(msg);
  output.data_length = 128;
}

void loop()
{
  for(uint8_t i = 0; i < 128; i++){
    data[i] = analogRead(0);
    //Approx 10,000 hz
  }

  output.data = data;
  msg.publish(&output);

  nh.spinOnce();
  delay(100-14);
}
```

20.3 Depth Test Arduino Code

```
#include <Wire.h>
#include <SparkFun_MS5803_I2C.h>
int atm = 99.7;

// Begin class with selected address
// available addresses (selected by jumper on board)
// default is ADDRESS_HIGH

//  ADDRESS_HIGH = 0x76
//  ADDRESS_LOW  = 0x77

MS5803 sensor(ADDRESS_HIGH);

//Create variables to store results
float temperature_c, temperature_f;
double pressure_abs, pressure_relative, altitude_delta, pressure_baseline;

// Create Variable to store altitude in (m) for calculations;
double base_altitude = 1655.0; // Altitude of SparkFun's HQ in Boulder, CO. in (m)

void setup() {
    Serial.begin(9600);
    //Retrieve calibration constants for conversion math.
    sensor.reset();
    sensor.begin();

    pressure_baseline = sensor.getPressure(ADC_4096);
}

void loop() {
  // To measure to higher degrees of precision use the following sensor settings:
  // ADC_256
  // ADC_512
  // ADC_1024
  // ADC_2048
  // ADC_4096

  // Read pressure from the sensor in mbar.
  pressure_abs = sensor.getPressure(ADC_4096);

  Serial.print("Pressure abs (mbar)= ");
  //  print pressure in kPa, not millibar
  Serial.println(pressure_abs/10);

  Serial.print("Depth (m) = ");
  Serial.println(((pressure_abs/10-99.7)*1000)/(971.8*9.81));

 if(((pressure_abs/10-99.7)*1000)/(971.8*9.81)<0.47){
  Serial.println("THRUST DOWN");
 }else if (((pressure_abs/10-99.7)*1000)/(971.8*9.81)>0.53){
  Serial.println("THRUST UP");
 }else {
  Serial.println("NO THRUST NEEDED");
  }
  delay(500);
  }
```

20.4 Ultrasonic Ranger Test Arduino Code

```
#define TRIGGER 7
//#define TRIGGER_1
#define ECHO 18

uint8_t activeSensor = 0;
volatile bool active = false;
volatile unsigned int pw_time = 0;

void rangerISR(){
  if (digitalRead(ECHO)==HIGH){
    pw_time = micros();
    return;
  }
  pw_time = (unsigned int)micros() - pw_time;
  active = false;
}

void startReading(uint8_t id){
  if (active) return;
  active = true;
  digitalWrite(TRIGGER + id, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER + id, LOW);
}

void setup() {
  // put your setup code here, to run once:
  pinMode(TRIGGER, OUTPUT);
  #ifdef TRIGGER_1
    pinMode(TRIGGER + 1, OUTPUT);
  #endif
  attachInterrupt(digitalPinToInterrupt(ECHO), rangerISR, CHANGE);
}

void loop() {
  // put your main code here, to run repeatedly:
  startReading(0);
  while (active);
  if (pw_time > 30000){
    Serial.println("No Obstacle");
  }
  else {
    Serial.print("Obstacle ");
    Serial.print(pw_time / 58);
    Serial.println("cm away");
  }

  delay(500);
}
```

20 User Manual

18.1 Safety Information

DO NOT COME IN CONTACT WITH WATER WHILE THE VEHICLE IS SUBMERGED

DO NOT PUT FINGERS IN/NEAR PROPELLER CAGES

A LIFEGUARD MUST BE ON DUTY AT ALL TIMES NEAR THE WATER

IN THE EVENT THAT A BYSTANDER GOES IN THE WATER WHILE VEHICLE IS SUBMERGED, DISCONNECT ALL POWER IMMEDIATELY

USING INCORRECT POWER SOURCE COULD RESULT IN DAMAGE TO DEVICE

18.2 Setup and Usage Procedure

Verify that waterproofed joints do not have any cracks and have a good seal.

Connect power leads from the PACT (Power and Communications Tether) to battery

Connect Communications leads either to the hardwired controller (A) or to Raspberry Pi (B).

CASE A: Hardwired Controller

Place Black Box in water

Ensure the cabin is closed on the device, place in water

Hardwired Controller Inputs:

Input	**Output Movement**
Left Stick: Forward	Left Horizontal Motor: Forward
Left Stick: Backward	Left Horizontal Motor: Backward
Right Stick: Forward	Right Horizontal Motor: Forward
Right Stick: Backward	Right Horizontal Motor: Backward
“Down” Button	Vertical Motors Dive
“Up” Button	Vertical Motors Climb

CASE B: Raspberry Pi

Connect power to raspberry pi

Wait for initialization on screen

Connect Joystick to raspberry pi

Turn on Black Box, set to ping, place in water.

Ensure the cabin is closed on the device, place in water

Press the “Autonomous search” Button

Wait several minutes as POLO locates MARCO via sound detectors

When POLO is near MARCO, a notification to switch to manual control will be visible.

To switch to manual control, press the left side button on the joystick.

Joystick controls:

Input	**Output movement**
Forward	Move forward on Y-axis
Back	Move Backward on Y-axis
Twist L/R	Rotate L/R around Z-axis
Throttle up	Climb on Z-axis
Throttle Down	Dive on Z-axis

When near to the Black Box, grab its hoop using the hook connected to the bottom of the AUV.

Climb to the surface. This could take several minutes.

After reaching the surface, move toward the edge of the pool for retrieval.

18.3 Calibration

Make sure sensors are submerged. Power on

The microphones do no require to be calibrated with the power off. However, once you are ready to begin searching look at the ambient noise on the specific frequencies that will be transmitted by the flight data recorder. Take not of the ambient sound level.

18.4 Retrieval

For retrieval, surface the device and maneuver towards the side of the pool. Once its at the side the device should be able to be picked up using the long PVC pipe that is coming out the back of the device. Avoid pulling on the tether cables during the retrieval process as this can cause communication errors with the device in the future. If the device is no surfacing stop all motors and the device will surface due to its buoyancy.

18.5 Disassembly and storage

Once the craft is removed from water,ensure that the kickstand leg on the front of POLO is down and secure before placing POLO on the ground. Next, disconnect the battery wires from the battery to ensure that there is no electricity flowing in the device. Next disconnect the cables from the hand held controller as well as the ARDUINO. Coil up the cables and keep them with the POLO craft.

18.6 Cleaning

Once the SABBLE-Lite platform is removed from the pool the only part that requires servicing and cleaning is the electronics box. Although this supposed to be completely waterproof, some water is expected to enter the box over time through the pressure relief valve after long periods of use. To remove the water from the box first open the box and peel the yellow seel back. Secondly tilt the craft on its side and drain any water out. Lastly, leave the box open to air dry after use.