Electrical and Computer Engineering

Montana State University

EELE489R -- Electrical Engineering Design II

5/4/2023

Senior Capstone Final Project Report

RoboSub Sound Localization

Design Engineers:

Fraser Robertson, Jarred Pickens, Elliott Avery

Project Customer:

Dr. Bradley Whitaker

Mr. Alexander Read

NAVSEA

# Signature Page

Prepared By

Fraser Robertson\_**\_\_\_\_\_\_\_\_\_** 05/07/23

Jarred Pickens\_\_\_**\_\_\_\_\_\_\_\_\_** 05/07/23

Elliott Avery\_\_\_\_\_**\_\_\_\_\_\_\_\_** 05/07/23

Approval:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_

Senior Engineer: {advisor’s name} Date

***I certify that this document is ready to be sent to the project client provided that the requested changes/edits within are made and the course instructors also approve delivery.***

# Technical Summary

Project Title: RoboSub Sound Localization

Team Members: Jarred Pickens, Elliott Avery, Fraser Robertson

Customer: RoboCats

Every year, RoboSub holds a competition for students from around the globe to create Autonomous Underwater Vehicles. Montana State University’s robotics club, RoboCats, wishes to produce a winning submarine for this annual RoboSub Competition. The Montana State University robotics club desires a solution to the challenge of providing their submarine with the location of an underwater pinger. A system must be built that is able to detect a pinger, determine the location of the pinger relative to the submarine, and must integrate with the existing RoboSub. The system will need to transmit the location of the pinger into usable information that the submarine can then use to navigate its way to the pinger.

To solve this problem a system was constructed that records audio through four hydrophones for 2.5 seconds each. These hydrophones are placed at known locations on a test frame. The audio recordings are then filtered for a specified pinger frequency and values are calculated for the time at which the pinger sound arrives at each hydrophone. From those values, the location of the pinger is calculated and returned to the submarine through a MatLab function. The system is wrapped into a python function with an input for the frequency of the pinger that the team wishes to locate.

Ultimately, the project was able to successfully detect pingers of a specified frequency and filter out unwanted data. It was able to calculate the arrival times of sound at specified hydrophones and solve for a location based on those values. Additionally, the system successfully integrated with the existing hardware and software of the RoboSub PC, specifically in Linux. There were, however, some issues with the accuracy of the system. These issues were a result of inconsistency in the time at which audio arrives at the hydrophones.

This project was an excellent learning experience for the team members and team as a whole. The team was able to learn the real-world applications of signal processing, how to integrate different pieces of code, and team management. While the project was not ultimately successful, the team learned the engineering design process and look forward to applying that to the real world.

# Table of Contents

[Signature Page 1](#_Toc5634124)

[Technical Summary 3](#_Toc187050016)

[Table of Contents 4](#_Toc741227879)

[1. Introduction 5](#_Toc652008823)

[1.1. Project Motivation 6](#_Toc533651899)

[1.2. Project Description 6](#_Toc430976731)

[1.3. Project Background 6](#_Toc1503975688)

[1.3.1. Project History 7](#_Toc1091425003)

[1.3.2. Technology Review 10](#_Toc1465693731)

[1.3.3. Applicable Standards 18](#_Toc504317188)

[2. Project Requirements 19](#_Toc1369073226)

[2.1. Objectives 19](#_Toc788169790)

[2.2. Product Requirements Outline 19](#_Toc1339914165)

[2.3. Design Constraints 20](#_Toc123906040)

[2.4. Additional Project Requirements or Deliverables 20](#_Toc1706734924)

[2.5. Team Member and Project Responsibilities 20](#_Toc778132544)

[3. Technical Design Solution 23](#_Toc554531260)

[3.1 Overview of Technical Solution 23](#_Toc734570172)

[3.3 Subsystem 2: Frequency Filter [Jarred Pickens] 25](#_Toc1569712255)

[3.4 Subsystem 3: Sound Location Algorithm [Fraser Robertson] 27](#_Toc311219457)

[3.5 Subsystem 4: Code Interface [Fraser Robertson] 29](#_Toc1226109330)

[3.6 Subsystem 5: Time Difference of Arrival [Elliott Avery & Jarred Pickens] 30](#_Toc575176609)

[3.7 Subsystem 6: Hydrophone Array [All Members] 32](#_Toc842141961)

[3.8 System-Level Integration Solution 34](#_Toc2095446750)

[4. Testing and Verification 36](#_Toc1885005579)

[4.1. Verification of Frequency Filter 37](#_Toc1858683328)

[4.2. Verification of Location Algorithm 50](#_Toc874742775)

[4.2.1. Spec 1.2.2 50](#_Toc976168612)

[4.2.2. Spec 2.1.1 51](#_Toc739901679)

[4.2.3. Spec 2.1.2 51](#_Toc947513092)

[4.2.4. Spec 2.2.1 52](#_Toc310695715)

[4.2.5. Spec 2.2.2 52](#_Toc408206151)

[4.3. Verification of Code Interface 53](#_Toc303072267)

[4.3.1. Requirement 2.2 53](#_Toc704203568)

[4.4. Verification of Time Difference of Arrival 54](#_Toc2095300315)

[4.5. Verification of Hydrophone Array 58](#_Toc1832698705)

[4.5.1. Spec 1.1.1 58](#_Toc1991667570)

[4.5.2. Spec 3.1.1 59](#_Toc2142956036)

[4.5.3. Spec 3.1.2 59](#_Toc350204756)

[4.5.4. Spec 3.1.3 59](#_Toc908108591)

[4.5.5. Spec 3.1.4 59](#_Toc550223398)

[4.5.6. Req 3.1 60](#_Toc1838950156)

[4.5.7. Spec 3.2.1 60](#_Toc2143382752)

[4.6. System-level verification 61](#_Toc1085150378)

[4.6.1. Req 1.1 61](#_Toc88330239)

[4.6.2. Req 1.2 62](#_Toc1730698980)

[4.6.3. Req 3.2 65](#_Toc179353024)

[4.6.4. Obj 1 65](#_Toc763286937)

[4.6.5. Obj 2 and Req 2.1 68](#_Toc1449122360)

[4.6.6. Obj 3 70](#_Toc1794566962)

[4.6.7. Full System Verification 72](#_Toc481833951)

[5. Conclusions 72](#_Toc1606500937)

[5.1. Summary of Project Results 73](#_Toc174446198)

[5.2. Future Work 73](#_Toc1676527324)

[6. Bill of Materials 75](#_Toc1222227981)

[7. Acknowledgements 76](#_Toc338782268)

[References 77](#_Toc766781416)

[Appendix A (Fabrication Plan) 79](#_Toc293957798)

[Appendix B (Verification Plan) 81](#_Toc483987118)

[Appendix C Charter Updates 94](#_Toc1665719359)

[Appendix D Matlab Code Used for System Testing 95](#_Toc683663469)

[Appendix E MatLab Filter Test Code 110](#_Toc1070842903)

[Appendix F MatLab Multilateration Test Code 118](#_Toc345106384)

[Appendix G Python Wrapper Function 121](#_Toc1251834455)

# Introduction

## Project Motivation

Every year, RoboSub holds a competition for students from around the globe to create Autonomous Underwater Vehicles. Montana State University’s robotics club, RoboCats, wishes to produce a winning submarine into this annual RoboSub Competition. The goal of the competition is to produce a new generation of engineers interested in automation. To win, the RoboCats submarine must complete navigation challenges autonomously. An example of one of these navigation challenges is moving to the location of an underwater pinger and completing a task or maneuver at that location. Providing a winning submission to the competition could open the path to more sponsorships with NavSea and could provide opportunities to engineers at Montana State University for careers in the industry after college.

## Project Description

The Montana State University Robotics club wants a solution to the challenge of providing their submarine with an underwater pinger. A system must be built that is able to detect a pinger, determine the location of the pinger relative to the submarine, and must integrate with the existing RoboSub. The system will need to transmit the location of the pinger into usable information that the submarine can then use to navigate its way to the pinger. As this project is a subsystem of the larger submarine, it is important to note the bounds of the project. This project does not include navigation and control of the submarine, or permanent mounting of the final solution. The team completing this project must provide suggested mounting of the final solution and ensure that the solution is compatible with the rest of the submarine. It is important to note that while the team will provide a suggestion of mounting geometry, this could be altered by the RoboCats team as long as the sensors are at least 1 foot away from all other sensors.

## Project Background

In order to satisfy the needs of the RoboCats team, a system must be designed to work with the many subsystems of a submarine. This necessitates research into the current state of the RoboCats submarine and into current technologies that can be utilized.

### Project History

Montana State University has had a robotics club, known as RoboCats, since 2010. The RoboCats team currently has a functioning submarine that requires a sound localization system. The submarine in its current state can be seen in Figure 1. The physical structure of the submarine will not change while the sound localization system is being designed.

An important feature of an autonomous submarine is a navigation algorithm that allows the submarine to move around without input. Such an algorithm has already been developed by the club. This algorithm has been coded in the programming language Python. This code requires the sound localization algorithm to interface with Python, but does not limit what language it is written in. Additionally, it is important to note that other necessary features of a submarine, such as a central computer and power, have already been provided.

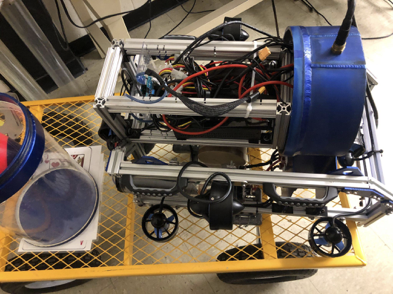
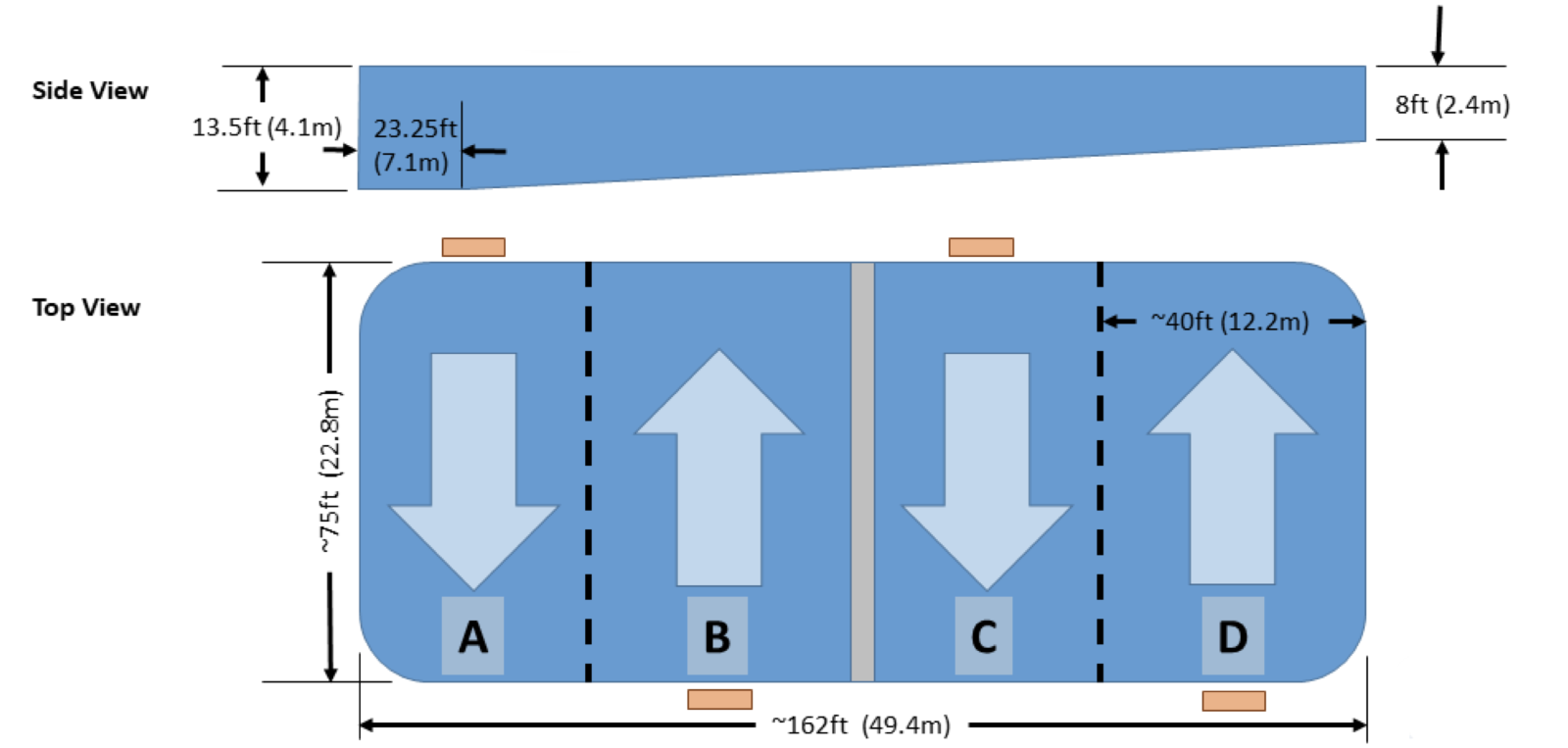


Figure 1: Current RoboCats Submarine

Adapted From [1]

In order to fully define the RoboSub challenge, the general rules of the RoboSub competition must be considered. Each year, the rules and tasks associated with the competition are slightly changed; however, the basic guidelines are constant year over year. Information defining the competition guidelines will be gathered from the 2022 RoboSub Team Handbook [2].

First, it is important to fully illustrate the particular challenge that this project is attempting to address. As has been previously described, the main goal of this project is to calculate the location of an underwater pinger. More specifically, according to the RoboSub handbook [2], during the competition, there will be four separate courses being used simultaneously. Each of these courses will have a pinger operating at a specific frequency. These pingers will be mounted to the bottom of the competition pool. Figure 2 depicts the dimensions of these courses. Each letter in the figure represents a different course; therefore, the maximum dimensions for one course are 22.8 m x 12.2m x 4.1 m. The RoboCats team has dictated that these dimensions can vary slightly from year to year, so the course dimensions were increased slightly for the purposes of this project.

Figure 2: RoboSub Course Dimensions

Adapted From [2]

Each of the above-mentioned courses will have a pinger. These pingers will all have their own frequency from 25 kHz to 40 kHz at integer frequencies. An example of the pinger timing can be seen below in Figure 3. As depicted in the figure, the pingers will be offset from each other. This prevents pingers from multiple courses emitting a frequency at the same time. Additionally, the team will have short notice as to what course they are going to compete on, so the system must be able to quickly adapt the frequency that it is sensitive to.

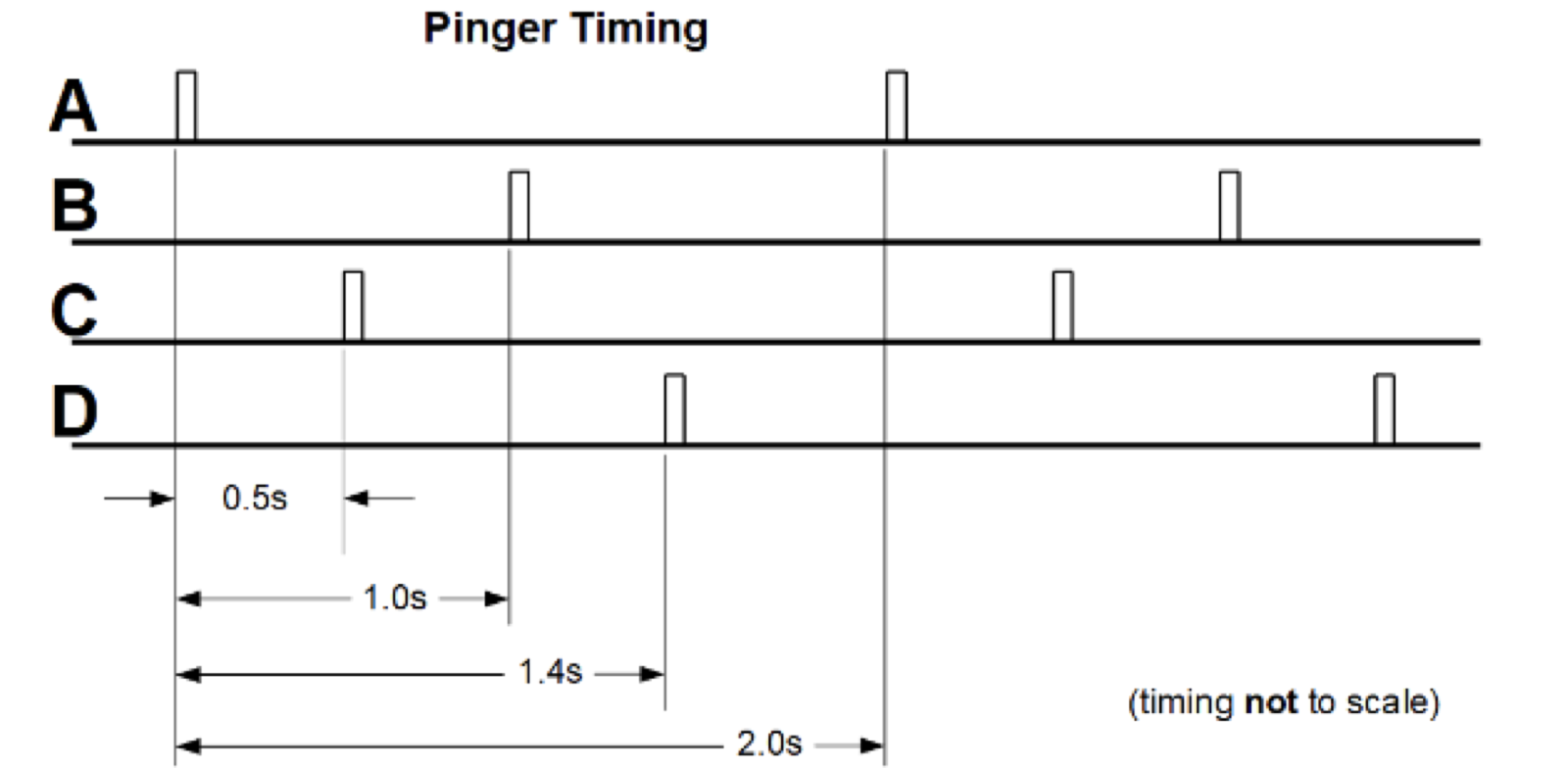
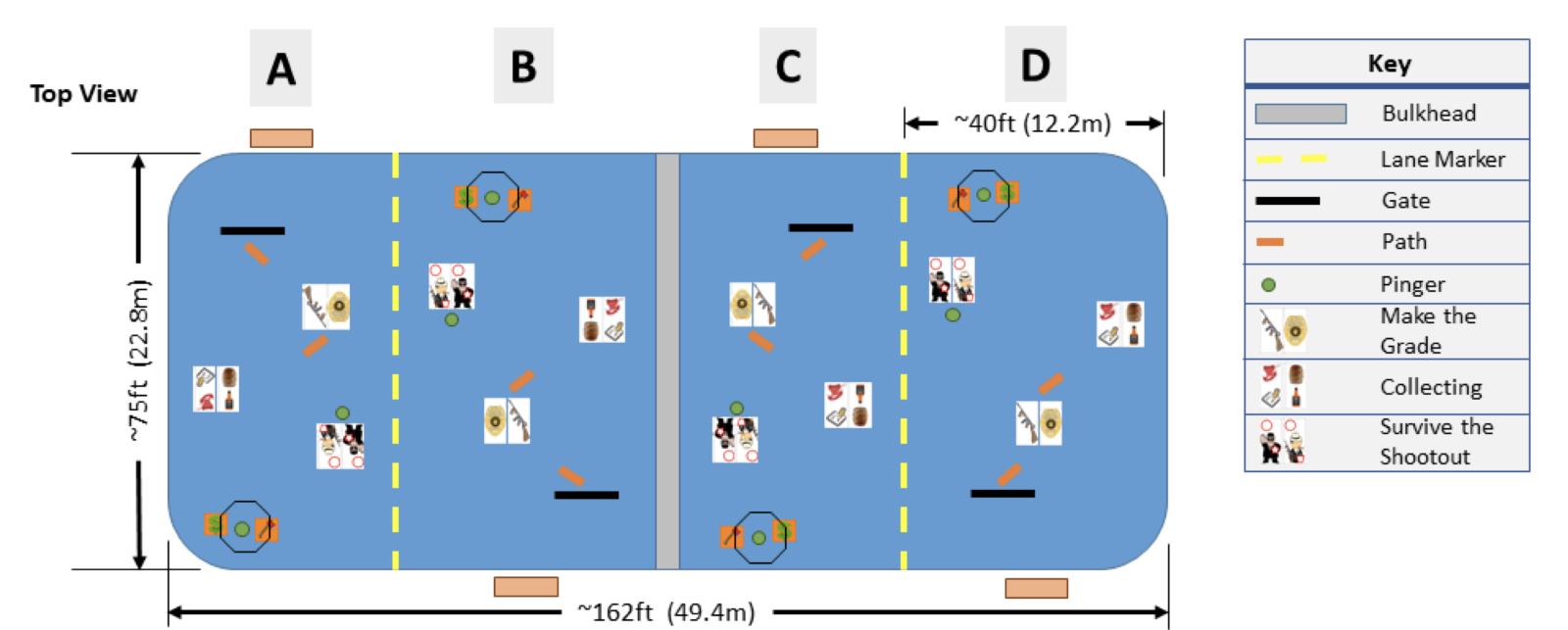


Figure 3. RoboSub Pinger Timing

Adapted From [2]

Once a submarine in the competition has positioned itself at the pinger it will have to complete a specific challenge. This challenge has not yet been defined for the 2023 competition; however, in previous years, the challenge has involved picking up an object. After the task has been completed, the submarine needs to surface inside of a 9-foot octagon hoop on the surface of the pool [2]. This can be seen in Figure 4 which depicts the layout of the course. As seen in the figure, the pinger is centered in the middle of the octagon. Navigation to the pinger will allow for the submarine to surface inside of the octagon.

Figure 4. RoboSub Course Layout

Adapted From [2]

### Technology Review

*Sound Propagation in Water*

Due to the acoustic nature of the RoboSub challenge, it is important to understand how sound travels in water for this project's success. The challenge is based on sound emitted by the pinger, so having a knowledgeable understanding of how that emitted sound travels is essential. To start, sound in water is created from compressions and decompressions in water molecules caused by pressure waves produced from vibrations. Figure 5 highlights what these compressions look like.

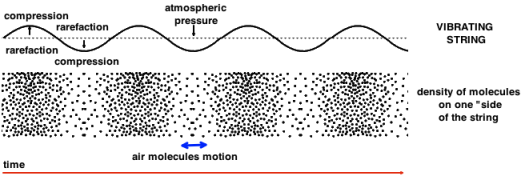


Figure 5: Display of sound compressions

Adapted from [3]

Sound propagating through air uses this same idea and explains why sound radiates outwards the same way in both air and water. However, there are a few differences in how sound propagates between the two, to start, sound is faster in water than in the air. To be exact, sound travels around 1.5 km/s in water whereas in air it travels at 340 m/s. This speed difference is due to the water molecules being in closer contact with each other than the molecules in air, this closeness is due to the nature of water being a liquid rather than a gas. Because sound travels faster in the water, this means that it also has a longer wavelength, which gives it the ability to travel further distances than in air. [4]

A sound’s speed and ability to travel long distances in water is impacted by temperature, with warmer water increasing speed and colder water decreasing speed. With changing temperatures during propagation, the sound no longer moves in a linear manner. Lowering temperatures will cause the sound wave to refract downwards, and pressure increases with a constant temperature will refract the sound upwards. Because of this, sound waves in the ocean can travel great distances with minimal loss. This occurrence is called channeling and consists of the sound wave refracting up and down over and over while traveling through the ocean as depicted in Figure 6 by “sound channel”. [5]

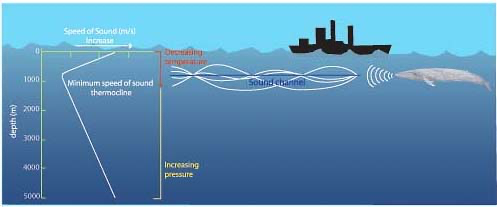


Figure 6: Underwater sound travel

Adapted from [4]

The scope of this project does not include the ocean; however, it is applicable if the project were to be upscaled into a submarine that operates in the ocean if NavSea wanted to do so. Something that is important to note is that sound in water has a different reference pressure than in air. Reference pressure is important when calculating sound pressure levels and may affect how the system is programed to operate. [4]

*Methods of Sound Localization in Robotics*

There has been significant development in sound source localization, specifically in robotics. The technology of sound source localization in robotics attempts to emulate the human auditory system. A simplified explanation of the human auditory system serves as a template for the development of a parallel artificial system:

The human auditory system consists of two structurally complex sensory organs, the ears, and the neural structures of the brain which process the sound and provide the human with sound source location information inducing a sensation of direction, elevation, and distance of the sound source. The path sound follows through the ears begins as it is funneled toward the inner ear by the outer ear structure and terminates at the hair cells which transduce changes in pressure to an electrical signal which is then interpreted by numerous structures in the brain [6].

The analog to the human ear for the purposes of sound source localization in robotics, is the acoustic sensor. The analog to the human’s cognitive faculties which process sound and derive relevant information from it, is computer hardware which houses the computational ability for signal processing.

Research surveyed for this technology review detail methods of sound source localization, all of which make use of acoustic sensor arrays; therefore, it is the assumption of the team that an acoustic sensor array is integral to the solution to this project. Methods of sound source localization employing an acoustic sensor array differ from one another by the physical measurements of sound signals used in signal processing. The three primary physical measurements of sound signal used in sound source localization are time of arrival, direction of arrival, and received signal strength [7].

Time of arrival, also called Time Difference of Arrival (TDOA) as the measurement pertains to multiple acoustic sensors in an array. This technique requires accurate synchronization of acoustic sensors in the array, which can be accomplished in audio processing software or signal processing software using a simple algorithm. This allows for meaningful differences in the time of arrival of the wavefront of a signal to each sensor position. For example, in Figure 7 below, each sensor begins reporting data at time *t = 0*. The orientation of the sensor array and spatial distribution of sensors places sensor *s1* such that the wavefront of a signal emanating from a singular source arrives first to sensor *s1* at *t = t1.* After a short interval, the signal arrives to sensor *s3* at *t = t2*, then to *s2* at *t = t3*, and finally to *s4* at *t = t4*. The time differences can be processed by a software-based algorithm to determine the location of the sound source. It is possible to estimate direction of arrival (DOA) using TDOA data integrated with the spatial coordinates of the sensors in the array [8].

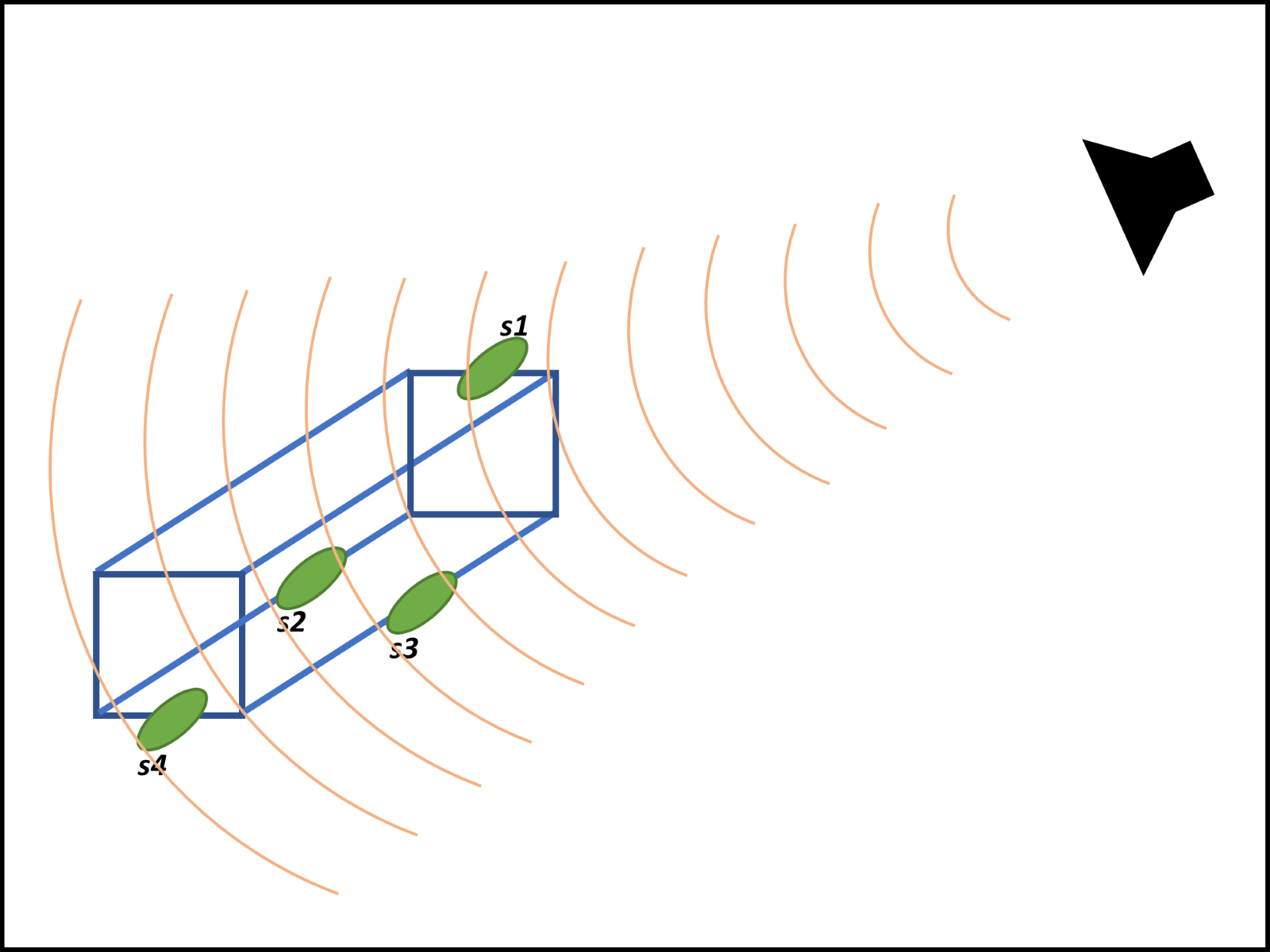


Figure 7: Illustration of TDOA

An important consideration with implications on the computation of TDOA is the geometry of the positions of the sensors within the array. The simplest choice of array geometry is the linear array. As shown in Figure 8 below, it allows for the most basic demonstration of how the distance from the array to the source is affected by TDOA. Sensor M4 is closest to the sound source and thus the sound signal reaches it first. Sensor M3 is a distance *d* from sensor M3 and is reached next by the sound signal, which travels the distance traveled to M4 plus the distance equal to the sine of the angle between the source and the linear array times the distance *d*. The TDOA data, coupled with the knowledge of speed of sound wave propagation in an aqueous medium, allows for the calculation of the angle of the linear array to the sound source [8].



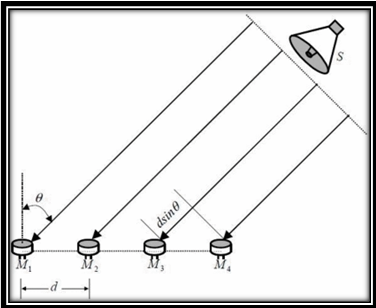


Figure 8: TDOA to Linear Array, adapted from [8]

The methodology detailed above for calculating direction of arrival can be extrapolated to different array configurations by considering the sensors in the array as pairs such that each possible pair within the array provides an opportunity for estimating direction of arrival. Each measurement can be averaged to produce a final estimate for DOA.

*How Hydrophones work*

Hydrophones are one of the most prevalent underwater sensors used in today. They can be considered the aquatic version of a microphone and measure sound in the same manner. Hydrophones detect compressions in the water created from vibrations, it then takes these compressions/pressure differences and converts the acoustic energy from them into electrical energy. Since the hydrophone is like a microphone, it is a passive listening device, meaning it doesn’t emit any sound, making it a dedicated listening device. Hydrophones can measure sounds because of material used in its production called piezoelectric material. Piezoelectric material uses mechanical stress, which, in this situation are compressions and pressure differences, and creates an electrical charge. In the presence of a sound wave, the material flexes to acknowledge the mechanical stress which then gives the data associated with the wave.

Figure 9 depicts the inside of a basic hydrophone. From top to bottom, it depicts the hydrophone circuitry, the hydrophone casing, and the waterproof shell of the hydrophone. As can be seen from the figure, the hydrophone is about 7 inches total in length. Because of the small size, mounting hydrophones onto the RoboCats submarine would be easy.

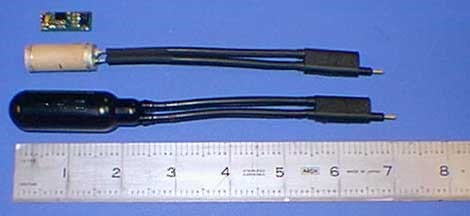


Figure 9: Different parts of basic hydrophone

Adapted from [9]

One factor that can affect the mounting process is which type of hydrophone is used. There are two main types of hydrophones. The first is an omnidirectional hydrophone. Omnidirectional hydrophones measure noises from all directions in an equal manner. One limitation of this is that there is less sensitivity and readability if trying to read sounds coming from a certain direction. The other type of hydrophones are unidirectional hydrophones. Unidirectional hydrophones are best at measuring sounds coming from a certain direction. One downside associated with this is the loss of listening capabilities for incoming noises that aren’t aligned with the unidirectional hydrophone. [9]

Most hydrophones, including both omni and unidirectional ones, make use of ceramic properties to make them operational. The ceramic hydrophones create small voltage signals over a range of frequencies, these signals are what’s measured to obtain the data from the sounds heard by the hydrophone. An example of a hydrophone in use is depicted in Figure 10.

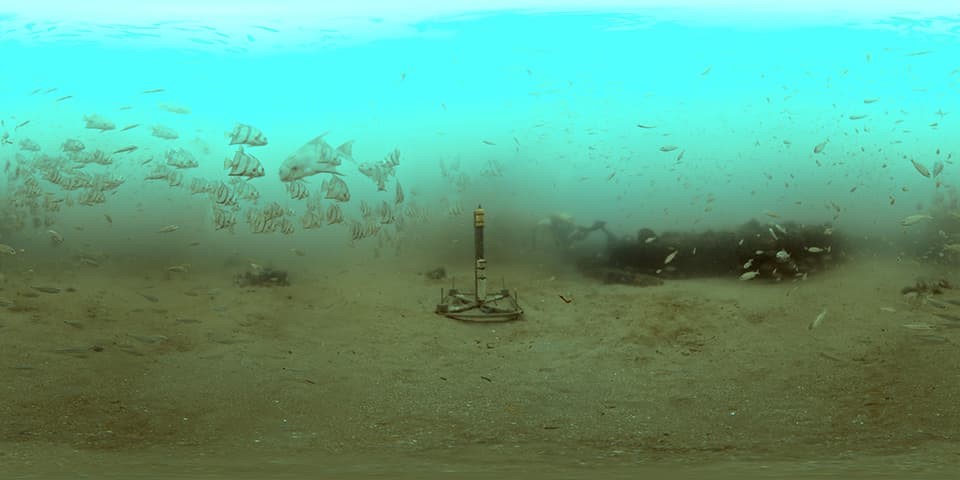


Figure 10: Stationary hydrophone in Gray Reed National Marine Sanctuary, Georgia

Adapted from [10]

This hydrophone is stationary in on the ocean floor and works in tandem with other hydrophones spread miles apart to locate noises and track them. The angle of the sound that comes into contact the hydrophone is very important for the quality of the data and can make readings appear differently due to the different amplitudes that come from different angles as well as frequency. This angle and frequency dependent response of the hydrophones is shown in Figure 11. From this figure as the angle increases away from the hydrophone, the amplitude will reduce by a significant amount. It can also be seen that the higher the frequency, the less amplitude there is. This is important for taking readings from the hydrophone, as well as setting up the hydrophones to make sure they operate in an appropriate manner.

Chart

Description automatically generated  
 Figure 11: Response curve of hydrophones

Adapted from [11]

*Programming Language Integration*

Currently, there are over 700 separate programming languages in the world. Each language has pros and cons that dictate the best language for a specific problem. For instance, some languages may have less built-in functionality, but are quicker to run; while others are slower but have more tools and packages to utilize [12]. In more complex systems, this can create the need for engineers to use more than one language. In these cases, a solution is needed to allow for the integration of multiple languages. RoboCats have made the decision to use Python as the basis for the central navigation system of the submarine. This impacts additional subsystems by necessitating the integration of languages with Python. Specifically, the requested sound localization system must integrate with Python.

There are many documented ways of integrating code bases; the main technique is called wrapping. This technique involves installing a library such as Boost in Python code which allows calling of C functions and methods [13]. Specific libraries such as boost exist for many of the major programming languages such as C, C++, Java, and C#. A list of these libraries can be seen in [13]. These libraries can be used to allow for a central programming language, in this case Python, to implement specific functions programmed in other languages. This allows engineers to break a problem into sections that are best solved using a particular language.

Additionally, there are methods that allow for the integration of environments such as MatLab from a Python code base. This technique is slightly different from the wrapping technique described above. In this case, the MatLab Engine API for Python needs to be imported to a Python script [14]. An instance of a MatLab engine is then created which allows specific MatLab functions to be called. This allows for output arguments to be returned to the Python script for use in the central code. Ultimately, this allows for algorithms written in MatLab to be utilized in Python.

There are many viable techniques for integrating languages with Python. This allows for the sound localization system to be programmed in many different languages despite the need for the final solution to integrate with Python. This results in the need for analysis to determine the appropriate language for the sound localization system.

### Applicable Standards

RoboSub has a variety of regulations for submarine entries that limit a submarine entered in the competition. These regulations state that a submarine cannot be over 125 pounds. Additionally, there are extra points given for entries that are under 84 pounds, and under 48.5 pounds respectively [2]. The RoboCats team has given the sound localization project a weight limit of 5 pounds to allow for the overall submarine to follow these standards. Additionally, RoboSub places restrictions on the overall size of entries. All entries are required to fit within a 6’ x 3’ x 3’ box [15]. The current RoboCats submarine fits within these regulations and thus this project must not cause the submarine to exceed that size.

# Project Requirements

## Objectives

RoboCats requires a sound localization system for a submarine. For this sound localization project to be successful, three critical objectives must be met. The system must be able to detect a pinger, the system must be able to determine the location of pinger relative to the sub, and the system must integrate with the existing RoboSub. Ultimately, completion of these requirements will result in a system that locates the position of an underwater pinger and transmits that position to the central navigation system of the submarine.

## Product Requirements Outline

Obj 1 The system must be able to detect a pinger

Req 1.1 The system must be able to detect sounds of specific frequencies

Spec 1.1.1 Detect in the range of 25 to 40 kHz

Spec 1.1.2 Differentiate between 1 kHz gaps

Spec 1.1.3 System must work in 24M x 13 M x 5 M Course

Req 1.2 The system must be able to detect specified pinger from multiple in pool

Spec 1.2.1 100% accuracy in selecting correct pinger

Spec 1.2.2 Less than 5 minutes for team to setup frequency for system to locate

Obj 2 The system must be able to determine the location of pinger relative to sub

Req 2.1 Pinger location must be determined to within a specified radius

Spec 2.1.1 Within .5-meter radius of pinger when sub within 2 meters

Spec 2.1.2 Correct cardinal direction to pinger when sub is more than 2 meters from pinger

Req 2.2 Pinger location must be communicated to navigation algorithm in x,y,z coordinates

Spec 2.2.1 3-unit vector array returned containing coordinates

Spec 2.2.1 Less than 5 seconds to report location

Obj 3 The system must integrate with existing RoboSub

Req 3.1 Array geometry recommendation must be competition conforming

Spec 3.1.1 Height: Less than 3 feet

Spec 3.1.2 Width: Less than 3 feet

Spec 3.1.3 Length: Less than 6 feet

Spec 3.1.4 Weight: Less than 5 pounds

Req 3.2 Solution must function with various conforming array geometries

Spec 3.2.1 Less than 10 minutes to adapt solution

## Design Constraints

The following is a complete list of constraints for the sound localization project. These constraints have been provided by the project sponsor.

-User interface must call solution with python.

-Navigation algorithm must call solution with python

-Compatible with RoboSub computer.

-Must be tested at an indoor facility in Bozeman.

-Must not cause RoboSub to break competition size and weight regulations

## Additional Project Requirements or Deliverables

The RoboSub Sound Localization System development proved challenging due to its significant signal processing component, necessity of seamlessly interfacing multiple coding languages, and restricted availability of a suitable testing environment. These project characteristics challenged the team members to expand and apply their skills, and be diligent about being prepared for and making efficient use of available testing times.

The amount of signal processing work demanded by the project presented a significant cognitive load. The team felt the strain of many sessions spent defining the problem and developing/implementing the appropriate solutions in code. Additionally, the completed project ran in MatLab, but needed to be contained within a Python wrapper, and run on a Linux operating system.

The limited availability of a suitable testing environment required that the team be diligent in preparing appropriately for short windows of time during which the project could be tested. If something did not go as planned during testing, the team had limited time to troubleshoot issues using collected data before the next time the testing environment became available.

The team did a fantastic job of working within the project constraints, and seeking counsel from project advisors when needed.

## Team Member and Project Responsibilities

Development of a RoboSub Sound Source Localization system can be broken down into three main tasks. The first of these tasks is to develop a subsystem which can receive signals emanating from the pingers. The second task is the development of a method which can filter for that pinger which is relevant to the current competition challenge. The third task is to develop a method for computing the location of the pinger relative to the submarine. As there are three main tasks and three team members responsible for the RoboSub Sound Localization project, it follows that one team member will assume majority responsibility for one task. Each team member will have equal responsibility for documentation as all three tasks will need to be documented equally.

*Responsibilities: [Robertson]*

Team member Fraser Robertson will assume responsibility for the location algorithm portion of the localization system. As this algorithm will likely be complex and central to the success of the project, only 60% responsibility will be assigned. Fraser additionally has 10% responsibility for the sensor array and pinger selection as these systems will interface heavily with the location algorithm. Finally, Fraser is taking 50% responsibility for Team Management and will keep track of team assignments and meeting minutes unless delegated to another member.

*Responsibilities: [Pickens]*

Team member Jarred Pickens will assume responsibility for the subsystem which selects for the pinger relevant to the current competition challenge. The pinger selection subsystem both receives input and delivers output to neighboring subsystems, and as such Jarred will be working closely with Fraser and Elliott to ensure seamless integration of subsystems. Jarred has been responsible for scheduling times & locations for team meetings and presentations and will continue to do so unless conflicts arise. In the later stages of the project, Jarred has also assumed responsibility for ensuring audio data synchronization with a new audio processing hardware configuration, and for developing improvements to the TDOA subsystem.

*Responsibilities: [Avery]*

Team member Elliott Avery will assume responsibility for the sensor array which mounts onto the submarine. Since the sensor array makes up the whole physical aspect of the project, it’s important to work alongside the other team members for smooth integration which is why 10% of the responsibility went to team members Jarred and Fraser. Elliott is the team treasurer, which was decided to be 15% of the team management.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Location Algorithm | Sensor Array | Pinger Selection | Documentation | Team Management |
| Fraser | 60% | 10% | 10% | 33.33% | 50% |
| Jarred | 20% | 10% | 80% | 33.33% | 35% |
| Elliott | 20% | 80% | 10% | 33.33% | 15% |
| Total | 100% | 100% | 100% | 100% | 100% |

Table 1: Percent responsibility for each team member on project

# Technical Design Solution

## 3.1 Overview of Technical Solution

There are six main subsystems that go into creating our system. Firstly, the sensor array is what is used to collect audio and is the physical aspect of the project. Next is the frequency filter subsystem which is represented as “frequency select” and “Filter Function” in the block diagram shown below in Figure 11. This subsystem is responsible for filtering for the desired frequency. Third is the Time Difference of Arrival subsystem which is represented as “TDOA Function” in the block diagram, this subsystem calculates the time delay between each hydrophone when the noise from the pinger is recorded/detected. Lastly, is the Multilateration Algorithim which is responsible for taking the values of the TDOA Function and using them to compete the location of the pinger in relation to the sub.

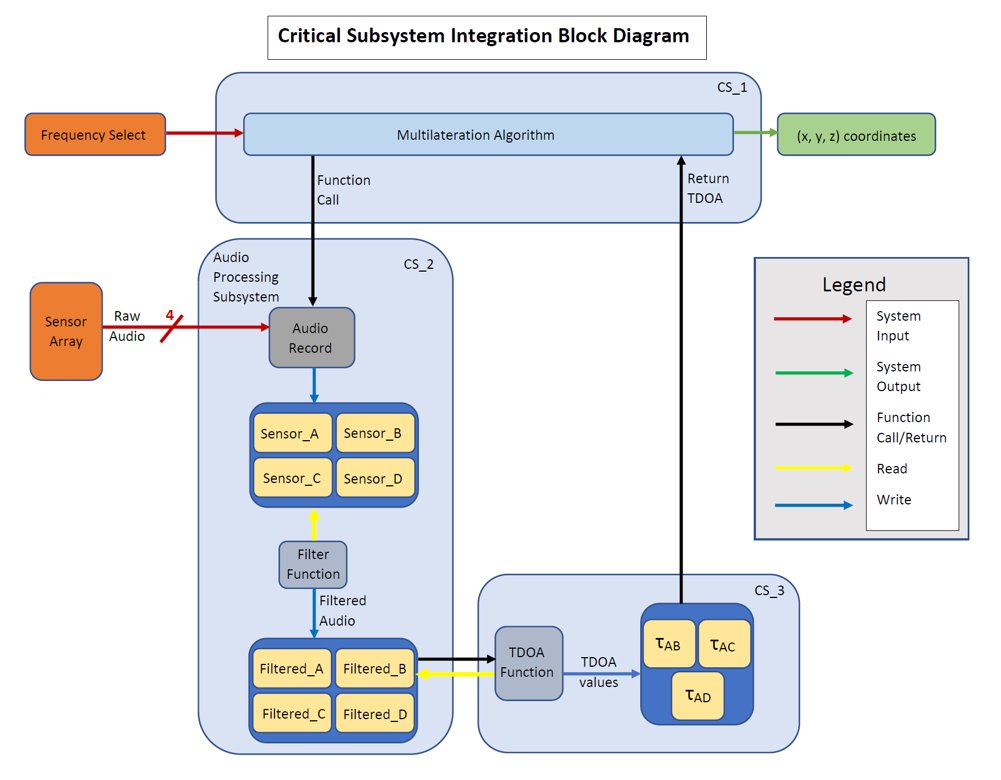


Figure 12: Conceptual block diagram [12]

*3.2 Subsystem 1: Audio Record [Jarred Pickens]*

The RoboSub Sound Localization System relies on the acquisition of raw audio data, which is then processed to determine the location of the sound source. The acquisition of the raw audio data is handled by the Audio Record subsystem. The functionality of the Audio Record subsystem is detailed below in figure 13.

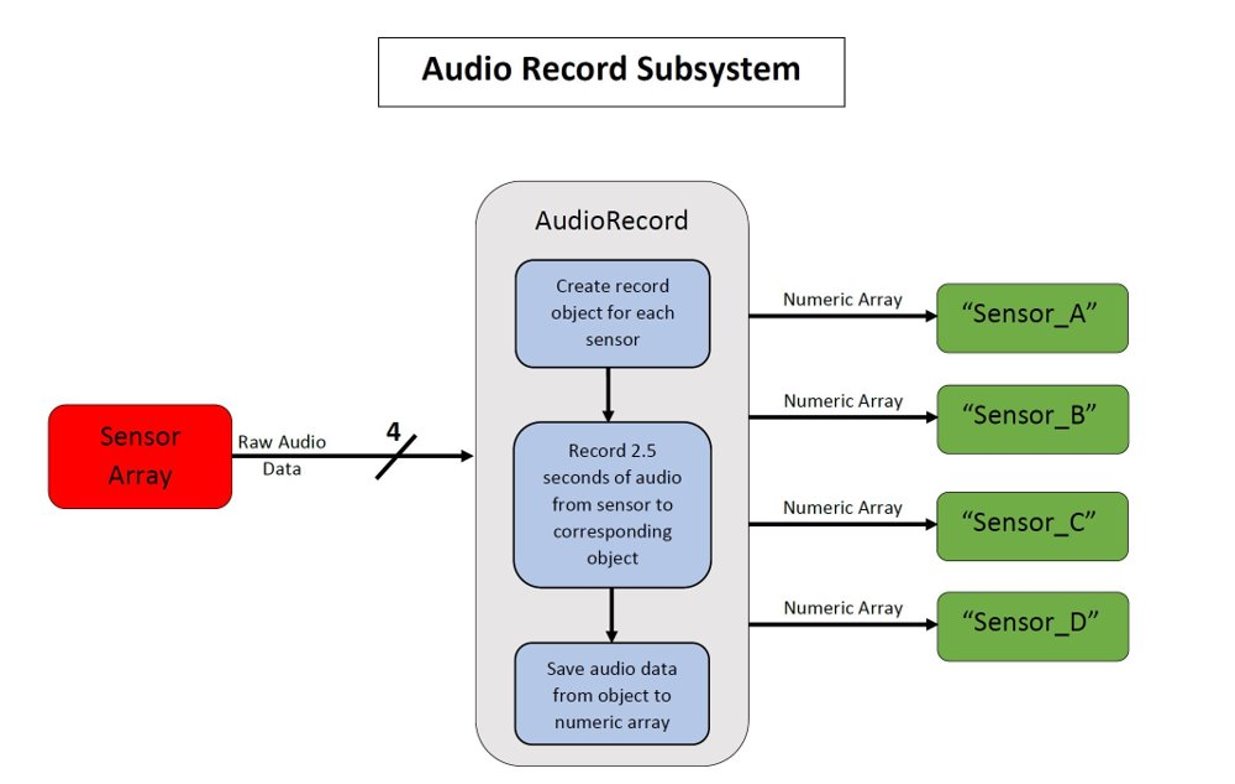


Figure 13: Audio Record [13]

The crucial piece of hardware used in this subsystem is the Behringer UMC404HD USB audio interface device. The device enables writing data audio to four synchronized channels simultaneously. This specific device was chosen for several reasons:

1. It accepts 4 XLR inputs. This is necessary because the hydrophones used in this system require the use of in-line amplifiers which have XLR terminations.
2. The device supplies 48 volts of phantom power, which is necessary to properly power the in-line amplifiers.
3. The device accepts “line level” input, which is required due to the amplified hydrophone signals.
4. The device sample rate is 192 kHz. This is the minimum usable sample rate for finding the peaks in the waveforms of the ultrasonic signals emitted by the competition beacons. -The maximum frequency that will be used in competition is 40 kHz. The Nyquist Rate for this frequency is 80 kHz, but the TDOA subsystem uses the absolute values of these waveforms. This essentially doubles the Nyquist Rate requirement for the signal.
5. The Device form factor accommodates the existing RoboSub’s watertight capsule.

As illustrated above in figure 13, the audio record subsystem accepts raw audio input from the four hydrophones in the sensor array and writes 2.5 seconds of audio to four numeric arrays each time it is called.

## 3.3 Subsystem 2: Frequency Filter [Jarred Pickens]

The pinger selection functionality is carried out by the audio processing subsystem block shown above in Figure 12, and functions as shown in figure 14 below. The audio processing subsystem consists of a Behringer UMC404HD four channel USB audio interface device and MatLab code. The MatLab code will instantiate the device, record synchronized audio data from each of the four hydrophones in the array and filter the raw audio data for a specified frequency. The raw and filtered audio is provided to the TDOA subsystem in numeric array format. The specified filtration frequency is that of the pinger which is relevant to the current stage of the NavSea RoboSub competition.

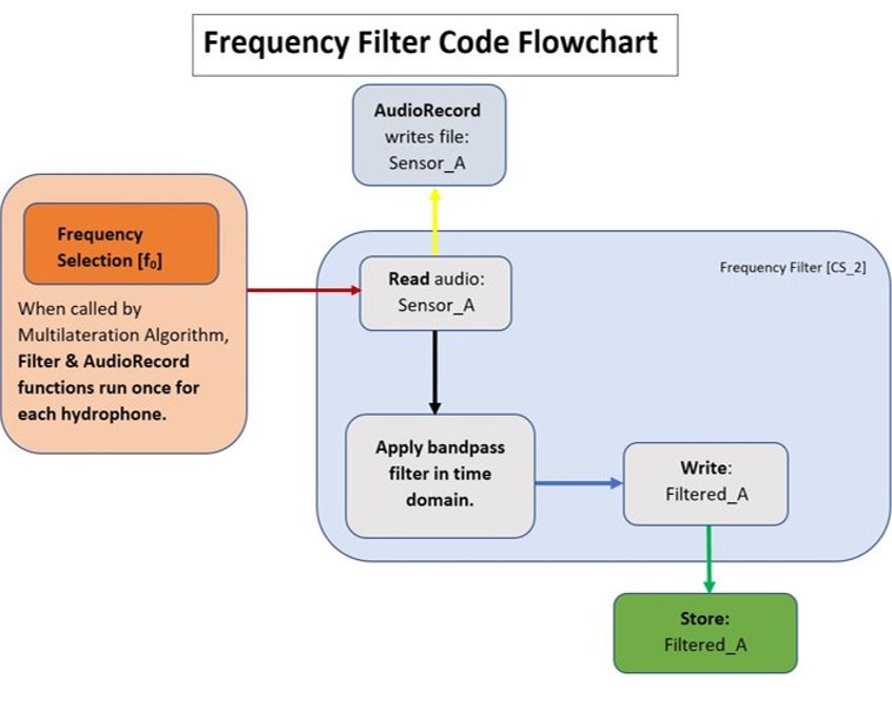


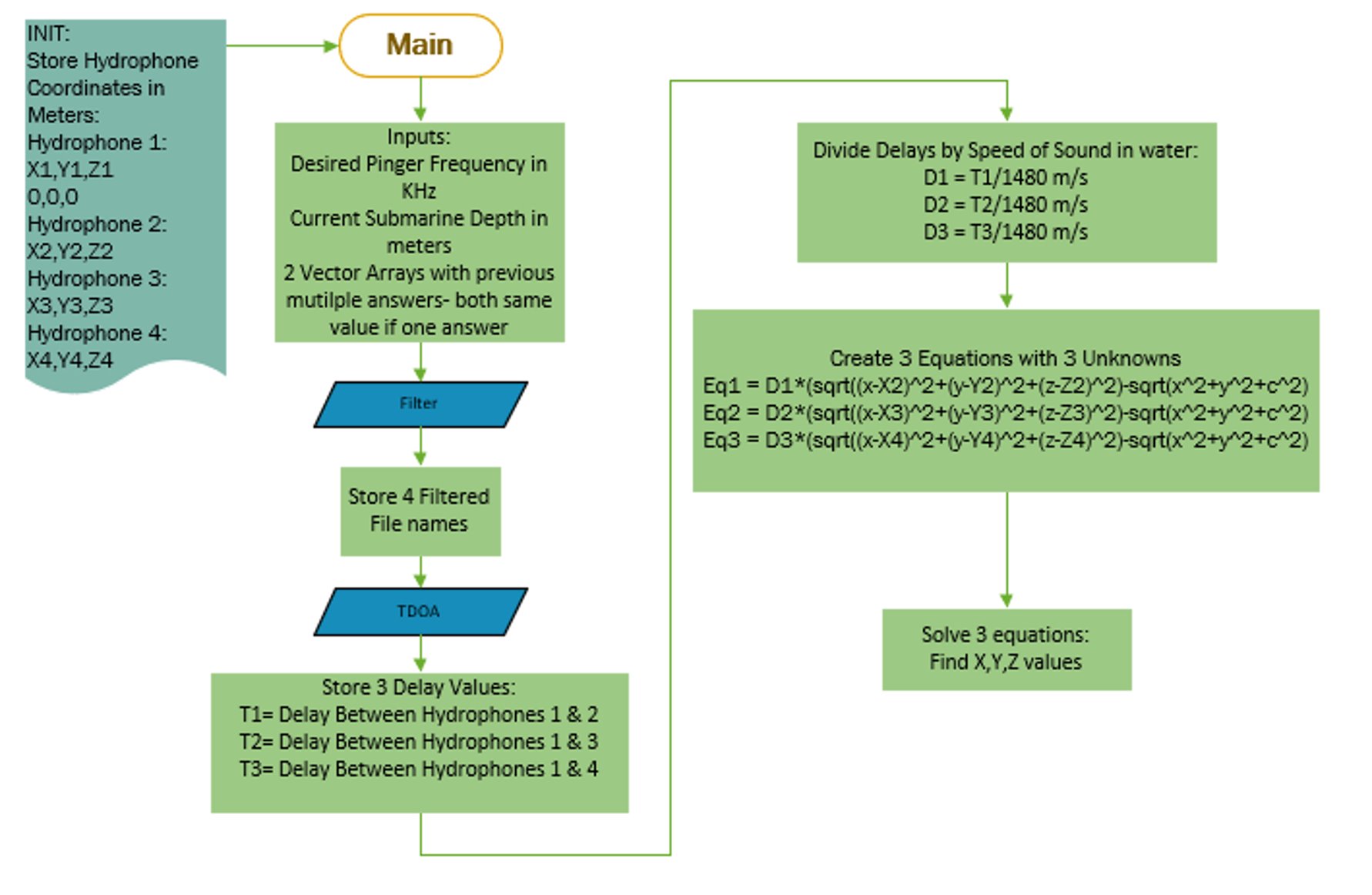
Figure 14: Frequency Filter Flowchart [14]

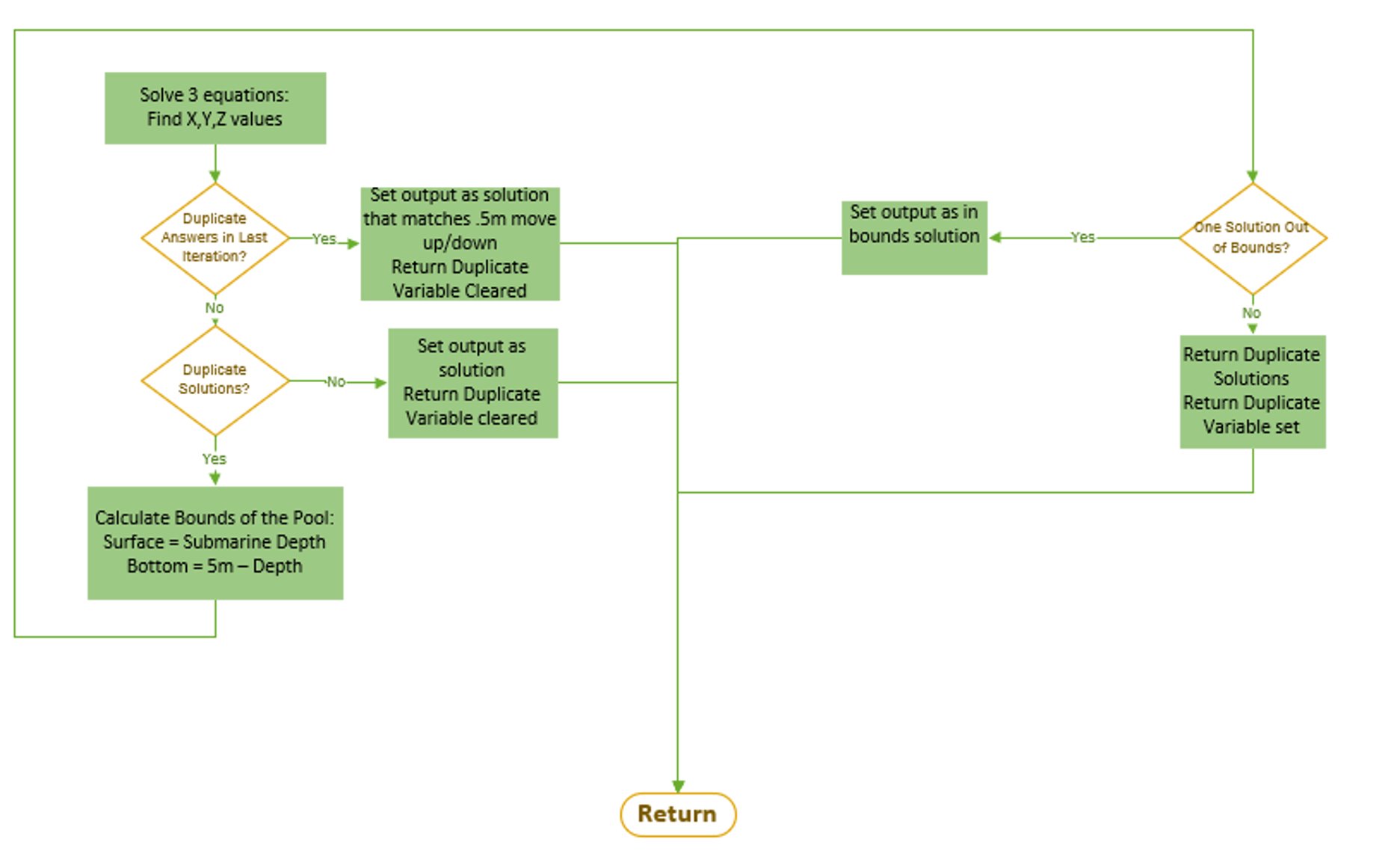
Raw audio data is read from the four hydrophones into their assigned channels on the UMC404HD USB audio interface, and a four-channel numeric array is produced. The four-channel numeric array is split into four single-channel numeric arrays representing their corresponding hydrophone’s raw audio data. A bandpass filter is applied to each single-channel numeric array which preserves the specified frequency data (specified frequencies are between 25 kHz and 40 kHz) and filters out all other frequency data. The TDOA subsystem will make use of both the filtered and raw audio data to determine TDOA values that will be used by the multilateration algorithm.

## 3.4 Subsystem 3: Sound Location Algorithm [Fraser Robertson]

An essential part of determining the location of the pinger at the RoboSub Competition is the algorithm used to determine the location of the pinger. This subsystem is represented by the multilateration block in Figure 12. The multilateration subsystem is responsible for a variety of specifications and requirements of our system. Mainly, the system is responsible for the specs under Req 2.1, pinger location must be determined to within a specified radius, and under Req 2.2, Pinger location must be communicated to navigation algorithm in x,y,z coordinates. The RoboCats team requested location information for the pinger in terms of x,y, and z coordinates. These constraints led to the decision to use a multilateration algorithm as the final solution for the design.

Multilateration involves using four hydrophones to calculate the x,y,z coordinates of the source of a sound. The x,y,z coordinates of each hydrophone are measured and stored as variables in the code. From there, these hydrophone locations and time at which the “ping” arrives at each hydrophone can be used to calculate the location of the underwater beacon. Additionally, oftentimes there are multiple solutions to the multilateration equations. In this case, a duplicate variable will be set in the code and tell the submarine to recalculate the location of the pinger. This can be done by moving the sub in a known direction and tracking which solution changes in accordance with that shift. The full code flow can be seen in the figure below. The essential inputs and outputs of the system are as follows: The function takes the desired frequency of the pinger the submarine is locating and returns a four-value array. Values for x, y, and x, and an additional variable that is set to one if there are duplicate answers or the system was not able to calculate an answer. This variable is set to zero if there is only one solution to the algorithm.

Figure 15: Multilateration Flowchart [15]

Figure 16: Multilateration Flowchart [16]

An essential portion of a functional multilateration algorithm is calculating the time values of when the ping arrives at each hydrophone. This calculation is often referred to as Time Difference of Arrival. The solution to this problem is talked about in the next section.

When fabricating this system, it was quickly determined that MatLab was the best software for the audio processing and recording necessary for our project. This decision for programming language dictated the implementation of the multilateration algorithm. MatLab has a symbolic math toolbox that allows users to write code that solves equations based on other variables. This allowed the solution to dynamically solve the above multilateration equations based on variables for TDOA values and for hydrophone locations. The code for the multilateration subsystem can be seen in the appendix.

## 3.5 Subsystem 4: Code Interface [Fraser Robertson]

A major portion of the system integration of this project was the way in which the code interacts with the existing RoboSub navigation system. The current system that the team uses utilizes a Linux computer with various python scripts and functions. In the overall system block diagram, this subsystem occupies an overlapping place with the previous subsystem, the location algorithm. More specifically however, this subsystem relates to the input of frequency selection and the output of the pinger location back to the navigation system. Due to the fact that this system overlaps significantly with the previous subsystem a new block diagram was not created; however, it is important enough to the overall system that its own fabrication section was warranted. This subsystem mainly relates to Req 2.2, Pinger location must be communicated to navigation algorithm in x,y,z coordinates.

As MatLab was the chosen programming language due to its strengths in audio recording and signal processing, a method of integrating the MatLab program with the python navigation algorithm was required. Python has a MatLab engine package that allows users to call MatLab functions from python code as though they were python functions. This allows the passing of input variables to those functions and for those functions to return values, which is essential for the system. As mentioned above, the MatLab solution takes in the desired pinger frequency in kHz and returns a four unit array back to the navigation algorithm. These types of values are easily passed and returned through the MatLab engine. To complete system integration, the MatLab engine was installed onto the RoboSub computer. Then a python script was written to set up the MatLab engine and call the pinger location system. This code and steps to download the MatLab engine can be found in the appendix.

## 3.6 Subsystem 5: Time Difference of Arrival [Elliott Avery & Jarred Pickens]

Time difference of arrival (TDOA) calculation is essential for the operation of the multilateration algorithm. The more accurate the TDOA calculations are, the more accurate the location of the pinger calculated by the multilateration will be. Accuracy of the system pertains to Req 2.1 and the subsequent specs 2.1.1 and 2.1.2 which all require the calculated location to be accurate to a certain degree. As a reminder, the TDOA subsystem/calculations take place after filtering, and the outputted TDOA values are returned to the multilateration algorithm, the subsystem in the conceptual block diagram can be seen below in figure 13:

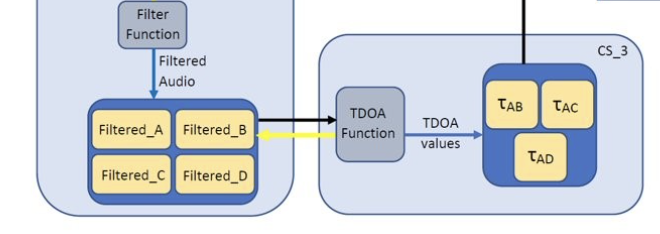


Figure 17: TDOA subsystem in conceptual block diagram [17]

Design of the TDOA subsystem is made up of two parts: an algorithm which attempts to make the best choice of amplitude peaks from each hydrophone to be compared to produce TDOA values, and a cross correlation function which will fine tune those choices for greater accuracy. The bandpass filter used to filter the raw audio for the specified pinger frequency removes high frequency data, which includes the precise moment of arrival of the pinger signal to the hydrophones. The TDOA algorithm whose functionality is illustrated below in Figure 14 is able to make use of the important data removed by the bandpass filter using the process described in the next paragraph.

It is important to note that the filtered and unfiltered audio arrays are the same length. First, the filtered audio array is searched for the moment that the filtered audio sinusoid first appears, and the index of the moment recorded. The unfiltered audio array is then trimmed to a length which includes the 210 samples preceding the index recorded from the filtered audio, and the moment the unfiltered audio arrives at the hydrophones is captured. The new unfiltered audio array is then searched for the instance that the signal amplitude is greater than 15% of its maximum value, and the index recorded for each hydrophone. These indices recorded are then compared for hydrophone pairs A&B, A&C, and A&D, and converted to the time difference of arrival values which will be further tuned for greater accuracy by the cross-correlation function that follows the TDOA algorithm block in figure 14 below.

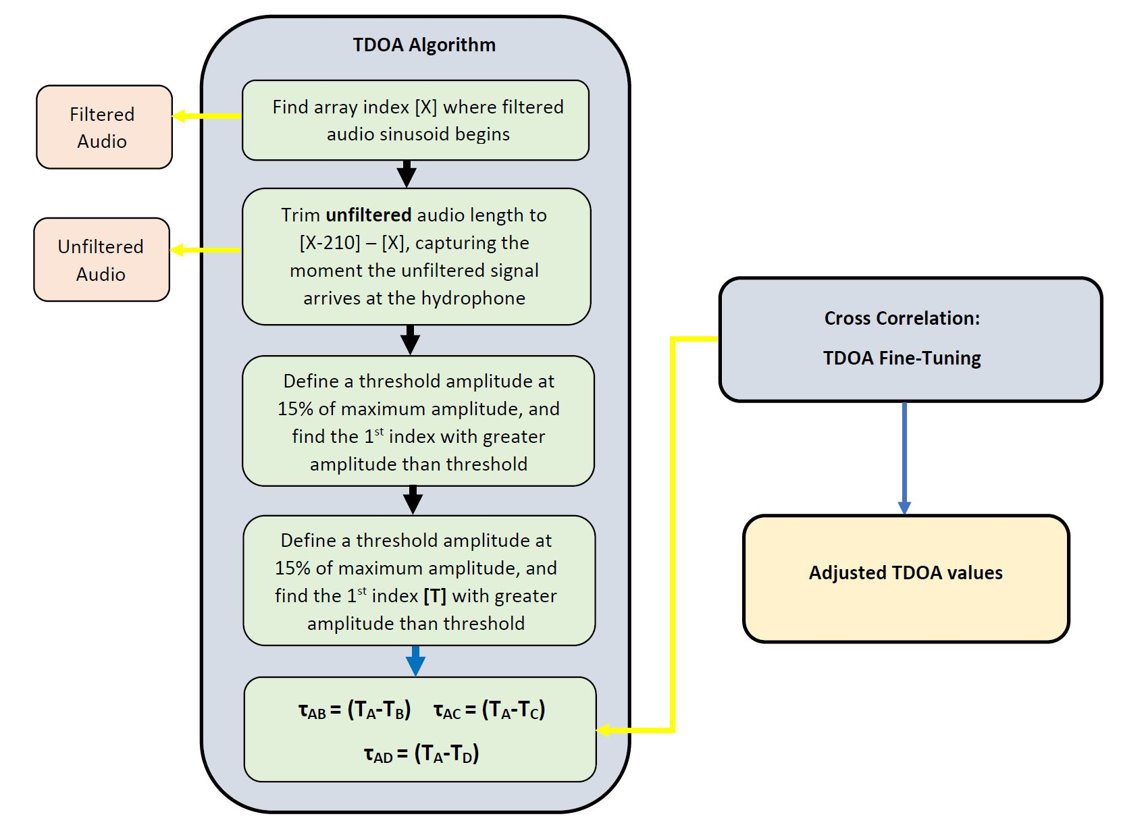


Figure 18: TDOA subsystem functionality [18]

Design of the TDOA algorithm uses the cross-correlation function which is native to MatLab. The function compares the sinusoids of the filtered audios of the hydrophone pairs mentioned before (A&B, A&C, and A&D). Using the sinusoidal peaks, the function finds at what time the peaks are maximally correlated, as in it moves the sinusoids on top of one another until the peaks line up. The shift from where the peaks are when the audio is received, to where they are maximally correlated is the time difference of arrival between the hydrophone pairs A&B, A&C, and A&D. The cross-correlation function in MatLab also has the ability of entering an argument which defines what the max lag can be; the max lag values for each hydrophone pair are based on the geometry of the hydrophone array and were calculated by finding the distance between each hydrophone pair and then finding the time it would take sound to travel that distance. The TDOA algorithm also uses the best choice TDOA’s from before to defined window of where the function should search for the correct TDOA values. This eliminates the problem of the function trying to line up the audio signals as a whole, and instead only searches within the first few sinusoidal peaks, giving us the most accurate answer. Figure 15 shows the flow chart of the cross-correlation part of the TDOA algorithm below.

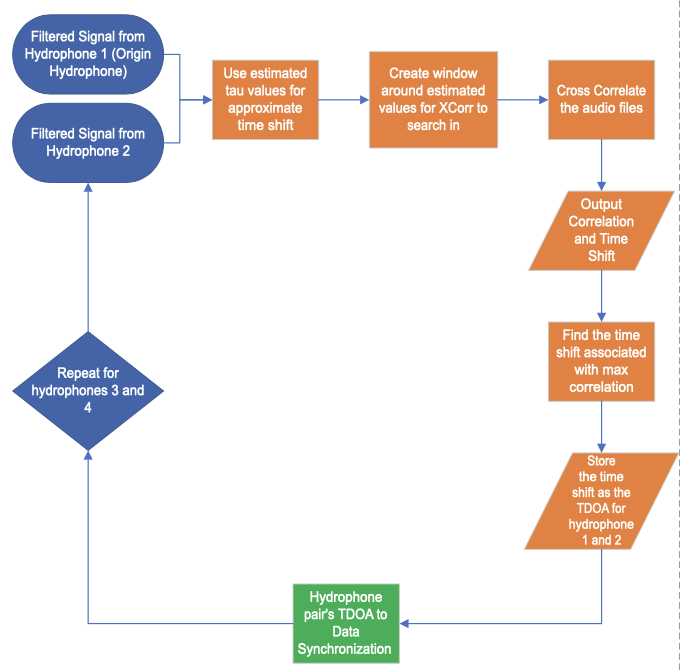


Figure 19: TDOA subsystem cross correlation flow chart [19]

## 3.7 Subsystem 6: Hydrophone Array [All Members]

Setup and design of the hydrophone array is critical to maximizing the accuracy of the data received and is also essential for recording the audios properly and reducing processing errors. The array is made up of a few parts which are: the physical frame of the array, the hydrophones, the inline amplifiers, the USB audio interface, and the RoboSub PC.

Starting with the first main part of the array, the physical array structure is made from aluminum and has been designed to maximize the distance between the origin hydrophone and the other three, which gives the best TDOA accuracy. The hydrophones are attached to the physical array and are used to record sounds underwater enabling use to observe the sound emitted from the pinger. Next the hydrophones are plugged into inline amplifiers which amplify sounds detected by the hydrophones and allow us to detect sounds from further away. These amplifiers are then plugged into the USB audio interface which enables us to use the amplifiers properly and gets rid of timing delays between the signals due to processing making the TDOA values even more accurate. Finally, the USB interface is plugged into the RoboSub PC which runs all the code that makes up the multilateration and audio recording system. Figure 16 below shows each of these subsystem parts and the connection sequence for each.

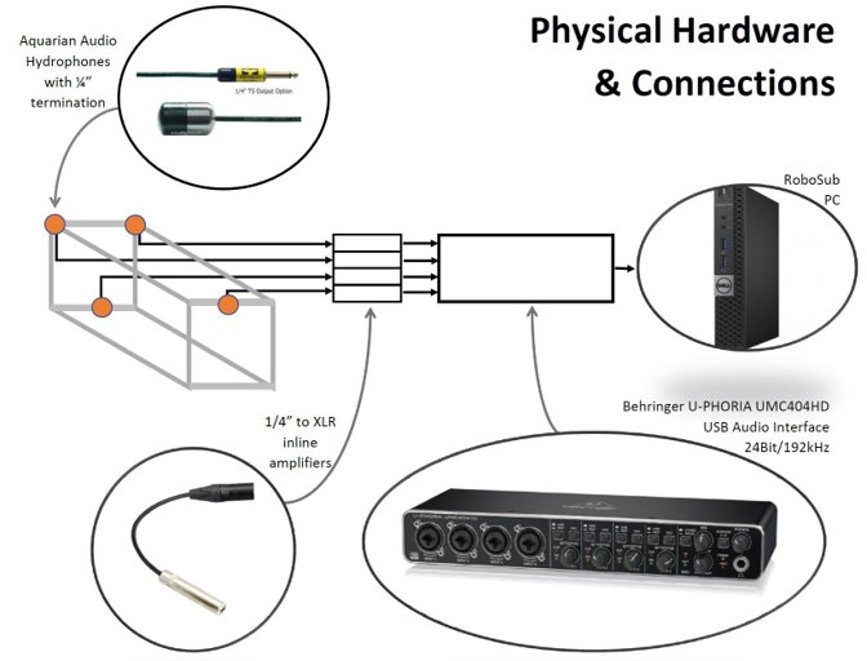


Figure 20: Physical hardware of hydrophone array subsystem [20]

For the purpose of testing the entire system, it was necessary to select hydrophone locations for pool testing. As simulations were run testing the accuracy of the multilateration algorithm, it was noted that accuracy was dependent on the physical hydrophone locations. Due to the speed of sound in water, oftentimes the TDOA values between hydrophone pairs are on the order of 10^-4 seconds. With a sampling rate of 192 kHz, it was necessary to maximize the distance between hydrophones to get the largest resolution for those values. This was achieved by maximizing the distance between hydrophones while placing hydrophones at realistic locations for the current submarine. The hydrophones were placed at the following locations in meters:

Hydrophone A (0,0,0), Hydrophone B (.838, -.813, .254), Hydrophone C (-.838, -.406, .813) Hydrophone D (-.838, -.406, .419).

## 3.8 System-Level Integration Solution

System level integration for the project mostly consists of code being compiled together while the physical aspects are all encapsulated into the hydrophone array subsystem. Focusing on the code of the system, the first subsystem that comes into play is frequency filtering. The frequency filtering subsystem uses the recorded audio from the hydrophones to filter for a frequency of the user’s choosing, removing any noise, and giving the wanted waveforms of the pinger on all 4 channels. After filtering, the TDOA algorithm code is run next which uses the unfiltered, and filtered audio as described before to calculate the TDOA’s of hydrophone pairs A&B, A&C, and A&D. After each TDOA pair has been calculated, the values are used by the multilateration algorithm to calculate the location of the pinger which is sent to the RoboSub PC, that is the end of the code. On the outside is all of the physical hardware which the configuration of has already been explained in section 4.5, Figure 17 shows the flow of the code in a more complex manner and shows how the physical and digital aspects of the system interact.

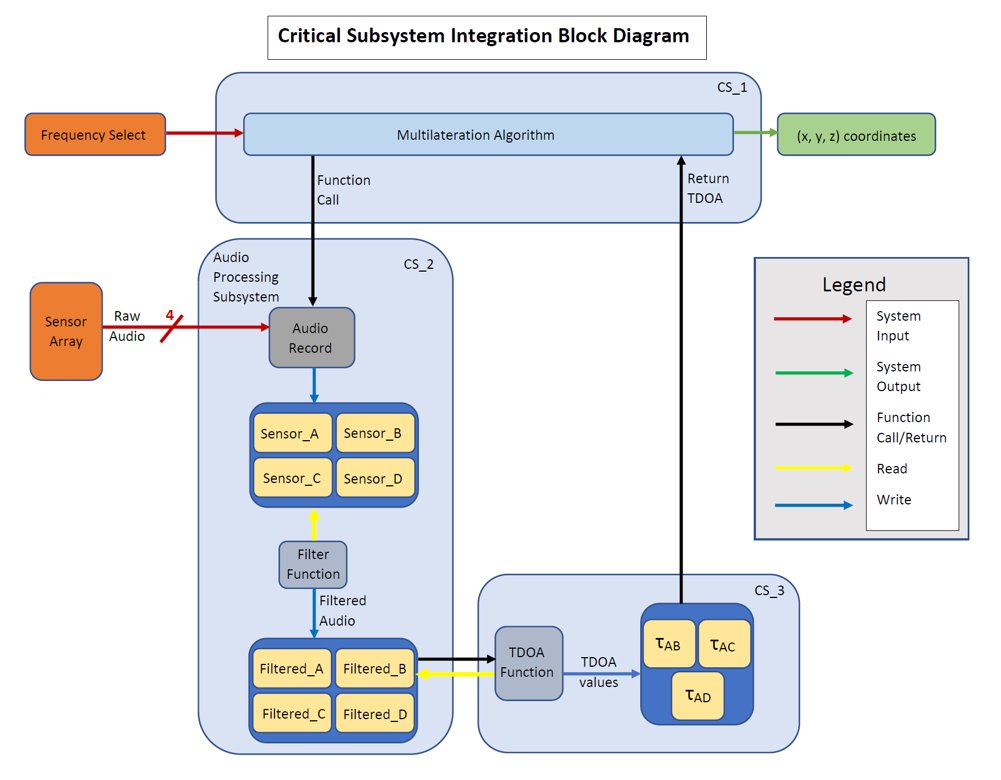


Figure 21: System Level Integration [21]

# Testing and Verification

For the next section of verifying the subsystems the order will be the same as before. The sound location algorithm and the Time Difference of Arrival subsystems will share some req and spec verifications because the sound location algorithm is dependent on the time difference of arrival calculations since that is where the accuracy of the systems output is decided. The table below shows an overview of the testing of the entire system. The following sections will go into the details of the testing process.

|  |  |  |  |
| --- | --- | --- | --- |
| **Num** | **Name** | **Test description** | **Results** |
| 1.1.1 | Frequency Range | Set pinger for variety of frequencies from 25 KHz to 40 KHz to ensure hydrophone works in required bounds | Passed |
| 1.1.2 | Differentiation of 1 kHz gaps | Creating MatLab waveform at different frequencies between 25 to 40KHz and verifying filter can differentiate 1 KHz gaps | Passed |
| 1.1.3 | Work in course dimensions | Place pinger an increasing distance from hydrophone in the pool and verify that waveform can be detected in audio file | Passed |
| 1.1 | System must be able to detect sounds of specific frequencies | A random pinger frequency that is within competition ranges will be set and the system will attempt to detect and record it | Passed |
| 1.2.1 | Filter Selection | Set waveform in MatLab at different frequencies from 25 to 40 KHz and verifying that waveform only shows when correct frequency is filtered for | Passed |
| 1.2.2 | Frequency Setup Time | RoboSub team members will be given an overview of the solution, then will be timed when setting the desired frequency of the system | Passed |
| 1.2 | System must be able to detect specified pinger from multiple in pool | Multiple pingers will be set at different arbitrary frequencies and the system will attempt to filter and record just one of them | Passed |
| 2.1.1 | Under 2 Meter Accuracy | Artificial TDOA's mimicking perfect TDOA's for the situation will be fed into the multilateration algorithim | Passed |
| 2.1.2 | Over 2 Meter Accuracy | Artificial TDOA's mimicking perfect TDOA's for the situation will be fed into the multilateration algorithim | Passed |
| 2.1 | Pinger location must be determined within a specified radius | Array will be placed various locations in the pool both over and under 2 meters from the pinger at a specific frequency, location will be computed | Failed |
| 2.2.1 | Return Type | The multilateration algorithim will be ran with arbitrary numbers and the output will be moitored | Passed |
| 2.2.2 | Algorithm Speed | The system will be tested as a whole and the speed it completes and reports the location calculations starting from when the signal is first received by the hydrophones will be recorded | Passed |
| 2.2 | Location Communication | The software of the system will be tested as a whole and the output from the system will be monitored | Passed |
| 3.1.1 | Array Height | When inspecting build plans for hydrophone array, calculate the height of the array | Passed |
| 3.1.2 | Array Width | When inspecting build plans for hydrophone array, calculate the width of the array | Passed |
| 3.1.3 | Array Length | When inspecting build plans for hydrophone array, calculate the height of the array | Passed |
| 3.1.4 | Array Weight | When inspecting build plans for hydrophone array, calculate the weight of the array without aluminum | Failed |
| 3.1 | Array geometry | Once hydrophone array has been constructed, verify that array fits size and weight specs. | Passed |
| 3.2.1 | Solution Adpatation | The RoboSub team will be tasked with altering the hydrophone locations and changing the code to reflect the new locations | Passed |
| 3.2 | Solution Must Function with various conforming array geometries | The hydrophones of the system must be able to be in different various placements without destroying the functionality of the system | Passed |
| Obj 1 | System must be able to detect a pinger | Pinger of arbitrary frequency will be placed and the system will detect it | Passed |
| Obj 2 | System must be able to determine the location of a pinger | Pinger of arbitrary frequency will be placed at an arbitrary distance from the system and the system will detect it and calculate its location in relation to the system, submarine will be moved relative to calulcted location | Failed |
| Obj 3 | System must integrate with existing RoboSub | Entire system will be placed in the pool on frame replicating RoboSub chassis connected to computer via one usb port. Pinger location will be computed | Passed |

Table 2

## Verification of Frequency Filter

Experiments were run using MatLab software and simulated data in to verify specifications 1.1.2 & 1.2.1. A separate experiment was designed for specification 1.2.2. Verification of specifications 1.1.2 & 1.1.3 required that the project be ready for system level testing, since the project team had very limited access to a pool and saved that time for system level testing.

*4.1.1 Spec 1.1.2*

A test was devised in which MatLab is used to generate a 2.5 second long signal composed of a randomly chosen series of three integer ultrasonic frequencies 1 kHz apart between 25 and 48 kHz. The sub-signals are of random amplitudes between zero and 11 andhave randomly chosen lengths and start times so that each “chirp” fits within the 2.5 second listening window. A bandpass filter is applied to the signal and filters out the highest and lowest frequencies. This shows that the filter can reliably distinguish between 1 kHz gaps.

The trial examples below in figures 22, 23, and 24 are of three different randomly generated signals where the filter distinguished between a series of three integer ultrasonic frequencies.

**Trial example 1:**

**Frequencies Used: 43kHz, 44kHz, 45kHz**

0.8 
0.6 
0.4 
0.2 
-0.2 
-0.4 
-0.6 
-0.8 
1.5 
0.5 
-0.5 
-1.5 
0.5 
0.5 
Generated signal in the time domain, before filtering 
1.5 
Time (seconds) 
Generated signal in the time domain, after filtering 
1.5 
Time (seconds) 
2.5 
2.5 
x 104 
2.5 
1.5 
0.5 
x 10 
2.5 
1.5 
0.5 
Generated signal in the frequency domain, before filtering 
Frequency (10 
Generated signal in the frequency domain, after filtering 
Frequency (10 
x 104 
x 104 Figure 22: Spec 1.1.2 Trial Example 1 [22]

**Trial example 2:**

**Frequencies Used: 30kHz, 31kHz, 32kHz**

1.5 
0.5 
-0.5 
-1.5 
0.5 
1.5 
0.5 
-0.5 
-1.5 
0.5 
Generated signal in the time domain, before filtering 
1.5 
Time (seconds) 
Generated signal in the time domain, after filtering 
1.5 
Time (seconds) 
2.5 
2.5 
x 104 
2.5 
1.5 
0.5 
x 104 
2.5 
1.5 
0.5 
Generated signal in the frequency domain, before filtering 
Frequency (10 
Generated signal in the frequency domain, after filtering 
Frequency (10 
x 104 
x 104 Figure 23: Spec 1.1.2 Trial Example 2 [23]

**Trial example 3:**

**Frequencies Used: 40kHz, 41kHz, 42kHz**

1.5 
0.5 
-0.5 
-1.5 
0.5 
Generated signal in the time domain, before filtering 
1.5 
Time (seconds) 
Generated signal in the time domain, after filtering 
1.5 
0.5 
-0.5 
-1.5 
0.5 
Time (seconds) 
2.5 
2.5 
x 104 
1.8 
1.6 
1.4 
1.2 
0.8 
0.6 
0.4 
0.2 
x 10 
1.8 
1.6 
1.4 
1.2 
0.8 
0.6 
0.4 
0.2 
1.5 
Generated signal in the frequency domain, before filtering 
Frequency (10 
Generated signal in the frequency domain, after filtering 
Frequency (10 
x 104 
x 104 Figure 24: Spec 1.1.2 Trial Example 3 [24]

The test was run in three 1000-trial batches. If the filter were to fail, the code would hang, and that test would fail. Each 1000-trial batch was successful and produced the following histograms. The histograms in figures 25, 26, and 27 show the occurrence rate of input ratios throughout the test. Here, the input ratio is defined as the amplitude of the desired frequency data divided by the highest amplitude frequency data present in the randomly generated signal. The histograms illustrate that the filter runs successfully, even when the input ratio approaches zero, i.e., the amplitude of the desired frequency is much lower than the amplitudes at adjacent integer ultrasonic frequencies.

**1000 Trial Batch 1:**

450 
400 
350 
300 
250 
200 
150 
100 
0.2 
0.6 
0.7 
0.8 
0.9 

Figure 25: Spec 1.1.2 1000-Trial Batch 1 [25]

**1000 Trial Batch 2:**

450 
400 
350 
300 
250 
200 
150 
100 

Figure 26: Spec 1.1.2 1000-Trial Batch 2 [26]

**1000 Trial Batch 3:**

Q Q 
400 
350 
300 
250 
200 
150 
100 Figure 27: Spec 1.1.2 1000-Trial Batch 3 [27]

The data collected from this verification process is convincing evidence that the system’s digital filter solution can reliably differentiate between 1 kHz gaps. The histograms routinely show that most input ratios are close to 1. This means that the amplitude of the frequency being filtered for has close to, if not the highest amplitude component of the generated signal. This is okay, because the histogram also shows that the filter is also successful when the input ratio approaches zero, meaning that the amplitude of the desired frequency component is 10% or less than that of the highest amplitude frequency component. It is important that the filter be successful despite what the amplitudes of other signal frequency components may be. Across three batches of 1000 trials, the filter was able to isolate the desired frequency data 100% of the time, despite the conditions. Although these tests were conducted using perfect simulated data, they provided the necessary data to conclude that the filter would perform to its specification during system-level testing.

*4.1.2 Spec 1.2.1*

A test was devised which used MatLab to generate a 2.5 second long signal that emulates the conditions of NavSea’s annual RoboSub competition. The signal includes the “chirps” from various beacons in the pool, each set to an integer ultrasonic frequency between 25 kHz and 40 kHz. The chirps from these beacons have random amplitudes between zero and one, and have random lengths and start times which will fit within the 2.5 second long listening window. The signal also includes a chosen “random” element: an explosion sound recording which may be a decent substitute for unpredictable noises occurring in the competition pool. Due to the way the signal is generated, the various components of the signal may or may not overlap with one another in the time domain, which creates adversity which will test the robustness of the digital filter. The code for this test is structured such that the bandpass filter will always attempt to preserve the frequency data of the first in the list of randomly generated beacon frequencies.

Figures 28 through 31 show a trial example plotted in the time and frequency domains, before and after filtering. The signal frequencies and amplitudes, and the frequency being filtered for, are clearly indicated.

**Trial Example 1:**

Trial Example 1: Pre-filter, Time Domain 
In this example, the randomly generated frequencies are 26, 45, 35, 48, 44, and 47 kHz. 
The frequency being filtered for is 26 kHz. 
GeneraEd signal in the time domain, before filtering 
1.5 
1 
0.5 
-0.5 
-1.5 
0.5 
1.5 
2 
2.5 
2 €000 
4 sooo 
35000 
48000 
44000 
47000 
>> amp 
0.4840 
0.2408 
0.8466 
o. 9324 
o. 3214 
Time (seconds) Figure 28: Spec 1.2.1 Trial Example 1, Time Domain, Pre-Filter [28]

Trial Example 1: Pre-filter, Frequency Domain 
In this example, the randomly generated frequencies are 26, 45, 35, 48, 44, and 47 kHz. 
The frequency being filtered for is 26 kHz. 
signal in the frequency domain, before filtering 
12000 
10000 
8000 
6000 
4000 
2000 
Frequency (104 Hz) 
x 10 Figure 29: Spec 1.2.1 Trial Example 1, Frequency Domain, Pre-Filter [29]

Trial Example 1: Post Filter, Frequency Domain 
In this example, the randomly generated frequencies are 26, 45, 35, 48, 44, and 47 kHz. 
The frequency being filtered for is 26 kHz. 
Generated signal in the frequency domain, after filtering 
2500 
2000 
0 1500 
1000 
500 
Frequency (104 Hz) 
x 10 Figure 30: Spec 1.2.1 Trial Example 1, Frequency Domain, Post-Filter [30]

Trial Example 1: Post Filter, Time Domain 
In this example, the randomly generated frequencies are 26, 45, 35, 48, 44, and 47 kHz. 
The frequency being filtered for is 26 kHz. 
Generated signal in the time domain, after filtering 
1.5 
0.5 
-0.5 
0.5 
1.5 
Time (seconds) 
2.5 Figure 31: Spec 1.2.1 Trial Example 1, Time Domain, Post-Filter [31]

Two more trial examples from this test are shown below in figures 32 & 33. They include the randomly generated signal in the time domain before and after filtering. Again, the signal frequencies and amplitudes, and the frequency being filtered for, are clearly indicated.

**Trial Example 2:**

Trial Example 2 
GeneraEd signal in the time domain, before filtering 
0.5 
-0.5 
0.5 
Frequencies 
[kHz] 
39 
27 
29 
1.5 
Time (seconds) 
Generated signal in the time domain, after filtering 
1.5 
0.5 
-0.5 
0.5 
Tlme (seconds) 
2.5 
37 
46 
28 
Input Ratio: 
2.5 
.0960 
Amplitudes 
.0931 
.6549 
.7324 
.9701 
.3251 
.6418 
1.5 Figure 32: Spec 1.2.1 Trial Example 2, Time Domain, Pre & Post-Filter [32]

**Trial Example 3:**

Trial Example 3 
signal in the time domain, before filtering 
0.5 
-0.5 
0.5 
1.5 
Time (seconds) 
Generated signal in the time domain, after filtering 
1.5 
0.5 
-0.5 
0.5 
Time (seconds) 
2.5 
2.5 
Frequencies 
[kHz] 
38 
41 
31 
27 
30 
45 
Input Ratio: 
.1392 
Amplitudes 
.1094 
.1451 
.7387 
.0851 
.3297 
.7858 
1.5 Figure 33: Spec 1.2.1 Trial Example 2, Time Domain, Pre & Post-Filter [33]

This test was run in 1000-trial batches. If the filter failed on any trial, the test was a failure. In three 1000-trial batches, the filter was successful for every trial, even when input ratios were close to zero. Here, as in the test for spec 1.1.2, the input ratio is defined as the amplitude of the desired frequency data divided by the maximum amplitude present in the signal. The histograms shown below in figures 34, 35, and 36 show the rate of occurrence of input ratios for each 1000-trial test.

**Batch 1 (1000 Trials):**

Filter Reliability Test 1 of 3 
Input Ratio Frequency (1000 trials) 
U- 
300 
250 
200 
150 
100 
50 
o 
There were zero 
instances of 
failure in 1000 
trials, even 
when input 
ratios were close 
to zero. 
0.1 
0.2 
0.3 
0.4 
0.5 
0.6 
Input Ratio 
0.7 
0.8 
0.9 
1 

Figure 34: Spec 1.2.1 1000-Trial Batch 1 [34]

**Batch 2 (1000 Trials):**

300 
250 
200 
150 
100 
50 
o 
Filter Reliability Test 2 of 3 
Input Ratio Frequency (1000 Trials) 
There were zero 
instances of 
failure in 1000 
trials, even 
with input ratios 
were close to 
zero. 
0.1 
0.2 
0.3 
0.4 0.5 
0.6 
Input Ratio 
0.7 
0.8 
0.9 Figure 35: Spec 1.2.1 1000-Trial Batch 2 [35]

**Batch 3 (1000 Trials):**

250 
200 
150 
U- 
100 
50 
o 
Filter Reliability Test 3 of 3 
Input Ratio Frequency (1000 Trials) 
There were zero 
instances of 
failure in 1000 
trials, even 
when input 
ratios were close 
to zero. 
0.1 
0.2 
0.3 
0.4 0.5 0.6 
Input Ratio 
0.7 
0.8 
0.9 
1 Figure 36: Spec 1.2.1 1000-Trial Batch 3 [36]

## Verification of Location Algorithm

### Spec 1.2.2

A critical portion of the location system is the speed at which members of the RoboSub team are able to select the pinger frequency that the submarine is detecting. At the competition, the team will have many adjustments to make to their system. Thus, the team asked that each setup task be as simple as possible. In order to test the speed at which the team was able to change the desired frequency, 3 members of the team who were most likely to be at the competition were tested. The team members were given the entire system code. After familiarizing themselves with the code, team members were given a frequency to filter for. Each member was timed how long it took for them to correctly alter the frequency. The subjects were asked to do this 3 times each and their times were averaged. These results can be seen in the table below.

|  |  |
| --- | --- |
| Team Member | Time to adapt solution (seconds) |
| 1 | 6.99 |
| 2 | 5.41 |
| 3 | 7.01 |

Table 2

In order to pass this spec, the team must be able to change the frequency in under 5 minutes. As seen from the results, all three members were able to change the frequency consistently under 10 seconds. Even under the additional stress of the competition, there is confidence that the team will be able to correctly alter the frequency in under 5 minutes. Therefore, this spec is passed.

### Spec 2.1.1

To verify the accuracy of the multilateration system a MatLab simulation was run. This simulation generated 100 random pinger locations within 2 meters of the hydrophone array. Expected TDOA values were calculated and rounded to what could be expected with the 192 kHz sampling rate of the system. From there, the TDOA values were plugged into the multilateration algorithm, and the error was calculated for each simulation. The simulation code can be seen in the appendix. The table below contains the accuracy results of the 100 runs.

|  |  |
| --- | --- |
|  | Error (Meters) |
| Min | .0025 |
| Max | .4177 |
| Mean | .1195 |

Table 3

This table highlights the accuracy of the multilateration algorithm. The closest the algorithm was able to determine the actual location of the pinger was .0025 meters away. The furthest the algorithm was able to calculate the pinger was .4177 meters away. This allows the algorithm to pass the spec as the requirement is that the system is able to determine pinger location within .5 meters. Additionally, the mean sits at .1195 meters which means that the majority of the runs were well under the required .5 meter error margin.

### Spec 2.1.2

To verify the accuracy of the multilateration system a MatLab simulation was run. This simulation generated 100 random pinger locations within the entire course bounds. These course bounds Expected TDOA values were calculated and rounded to what could be expected with the 192 kHz sampling rate. From there, the TDOA values were plugged into the multilateration algorithm and the error was calculated for each simulation. The simulation code can be seen in the appendix. The table below contains the accuracy results of the 100 runs.

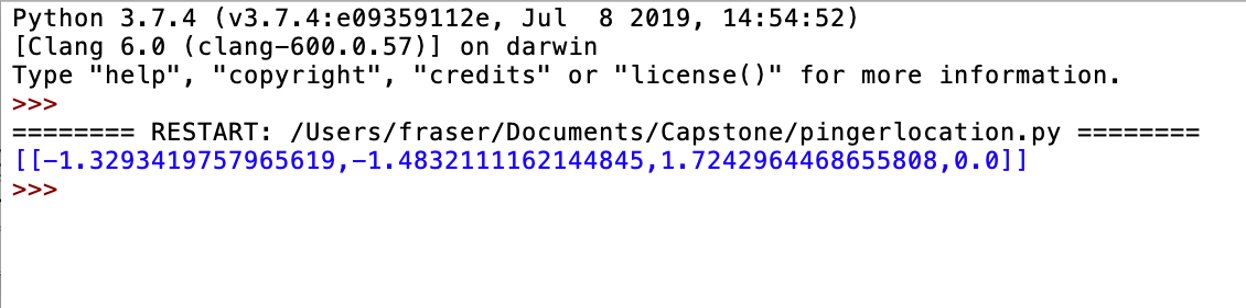
|  |  |
| --- | --- |
|  | Error (Meters |
| Min | .0082 |
| Max | 6.1385 |
| Mean | .7975 |

Table 4

This table highlights the accuracy of the mutlilateration algorithm. The closest the algorithm was able to determine the actual location of the pinger was .0082 meters away. The furthest the algorithm was able to calculate the pinger was 6.1385 meters away. In each of the 100 runs of the code, the algorithm was able to determine the correct direction. This test verifies that the spec is passed.

### Spec 2.2.1

The next specification requires that the return type of the multilateration algorithm be a three-unit vector array. Each unit corresponds to an x, y, and z location. This verification follows a similar process to the two previous verifications. A MatLab simulation was run. This simulation generated 100 random pinger locations within the entire course bounds. These course bounds Expected TDOA values were calculated and rounded to what could be expected with the 192 kHz sampling rate. From there, the TDOA values were plugged into the multilateration algorithm. For each one of these runs, it was verified that the system returned an x, y, and z coordinate. 100 trials were run and each one of runs returned a solution like the one seen in the figure below, meaning this spec was successfully passed.

Figure 37: Multilateration output as a three-unit vector array [37]

### Spec 2.2.2

This specification relates to the speed at which the system can return the solution to the RoboSub navigation algorithm. In order to verify this spec, it was necessary to compile the entire code of the system together. MatLab’s built in timing functions were used to clock the speed at which the algorithm could return a solution on the RoboSub computer. This test was repeated 6 times and the speeds can be seen in the table below.

|  |  |  |
| --- | --- | --- |
| Trial | Entire System Speed (Secs) | Algorithm Speed (Secs) |
| 1 | 4.94 | .93 |
| 2 | 4.89 | .87 |
| 3 | 4.88 | .88 |
| 4 | 4.88 | .86 |
| 5 | 4.89 | .87 |
| 6 | 4.89 | .89 |

Table 5

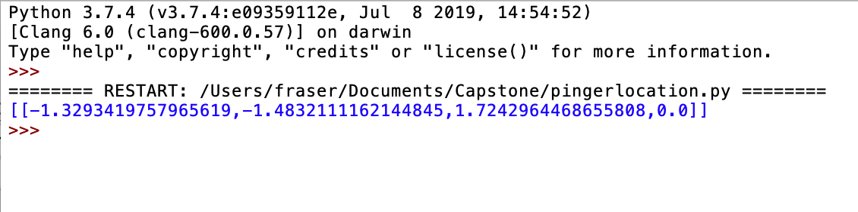
As can be seen in the table above, the system is repeatedly able to return a solution of the pinger location in under 5 seconds. As 5 seconds was the maximum time allowed for the solution to work, this spec is passed. Additionally, as can be seen in the third column, the physical algorithm of filtering, calculating TDOA values, and solving the multilateration equations can be done consistently under 1 second. The rest of the time that is being recorded relates to setting up the audio interface and recording audio. If the team desires an even faster system, there is a potential to write another script for setting up the audio device that only needs to be run one time. This would cut the system speed down significantly.

## Verification of Code Interface

The code interface is a relatively small portion of the project. This interface works closely with the multilateration algorithm from above, but only has one requirement associated with it. The verification of the working of this interface was relatively simple, but essential for the full system verification of the project.

### Requirement 2.2

To verify that the location of the pinger computed by the system is communicated correctly, it was essential to run a multitude of lab simulations to verify that the system was able to consistently have the correct python return type. These simulations were run with spoof TDOA data and run from the command line. In each case, it was verified that a 3 unit vector array was returned in addition to an additional variable to account for duplicate answers. In each of the 10 runs, the system was able to correctly display a solution in addition to the duplicate variable. These solutions were then compared to simply running the same simulation in MatLab to verify that the system was performing as expected. An example of the output of the system from the command line can be seen in the figure below. Due to these results, the system was able to pass requirement 2.2.

Figure 38: Output of code in 3-unit vector array form [38]

## Verification of Time Difference of Arrival

Verification of the TDOA algorithm does not have any specifications or requirements directly associated with it because the accuracy of the subsystem ties directly into the accuracy of the multilateraion algorithm. This means that the verification of the TDOA subsystem is verifying that the that it is working properly and to the best accuracy that the system can attain. To verify this, it was necessary to make sure that the best choice algorithm was choosing the most appropriate peaks on the waveforms. This was done by looking at the waveforms and manually choosing which peaks would be best and then comparing those to the peaks that the algorithm chooses. These comparisons for the 4 hydrophone pairs are shown below in figures 39-44 below.

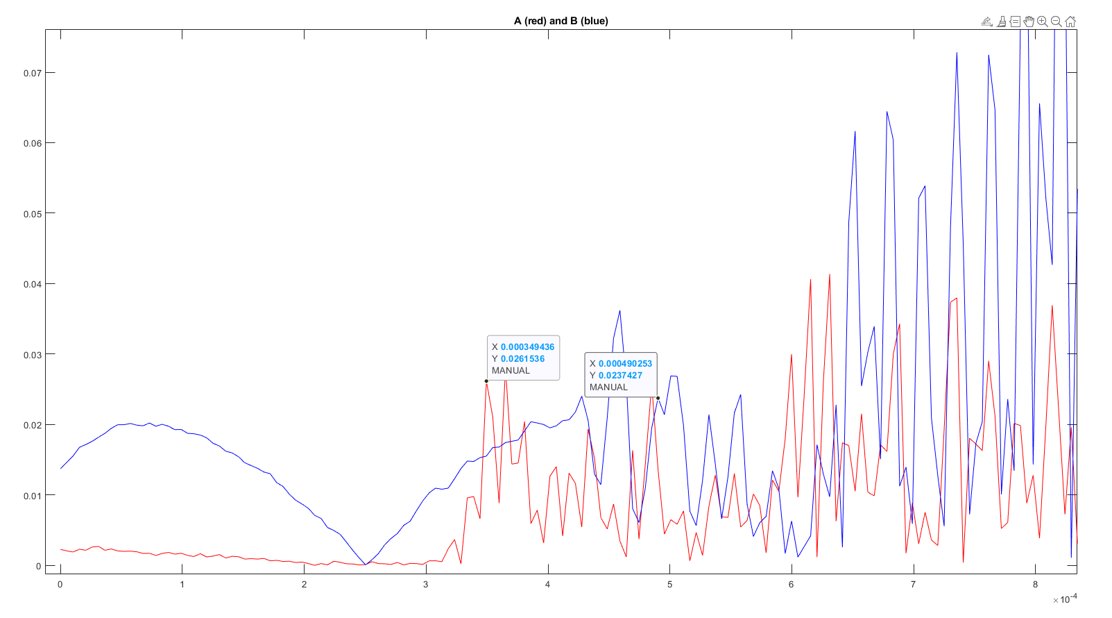


Figure 39: Manually chosen peaks for hydrophone pair A&B [39]

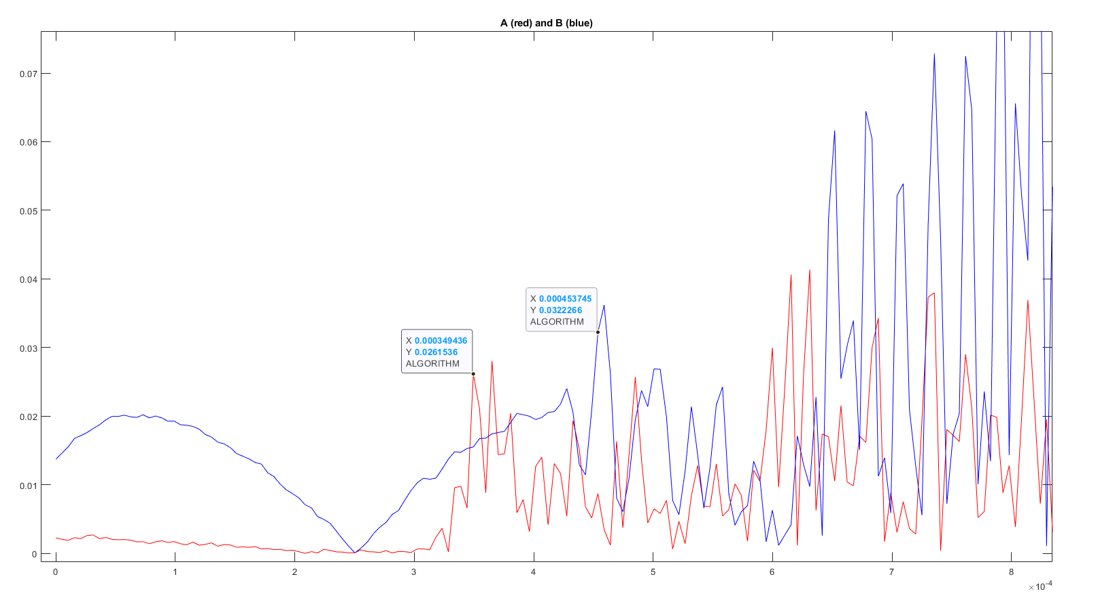


Figure 40: Algorithm chosen peaks for hydrophone pair A&B [40]

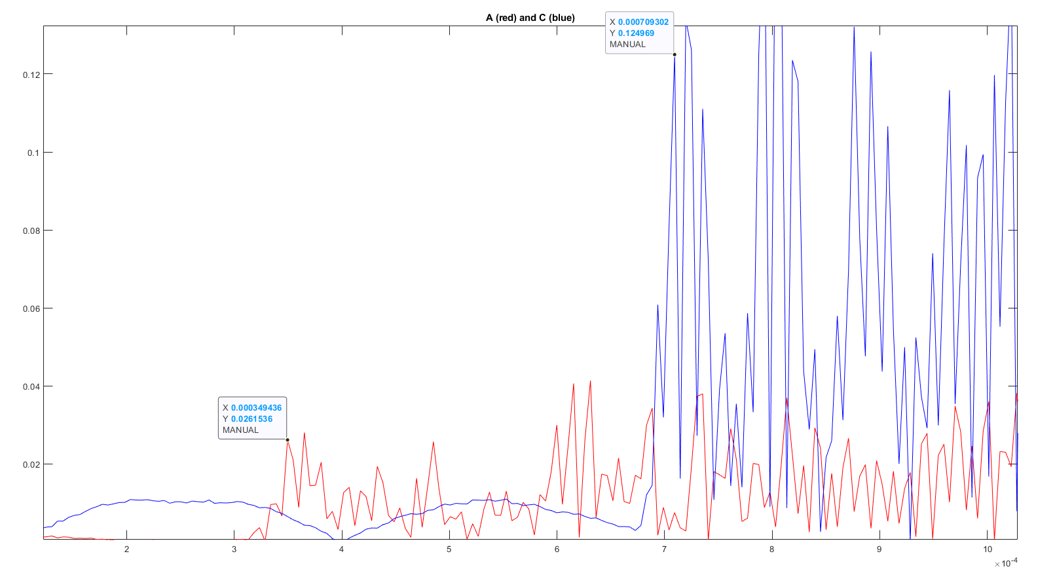


Figure 41: Manually chosen peaks for hydrophone pair A&C [41]

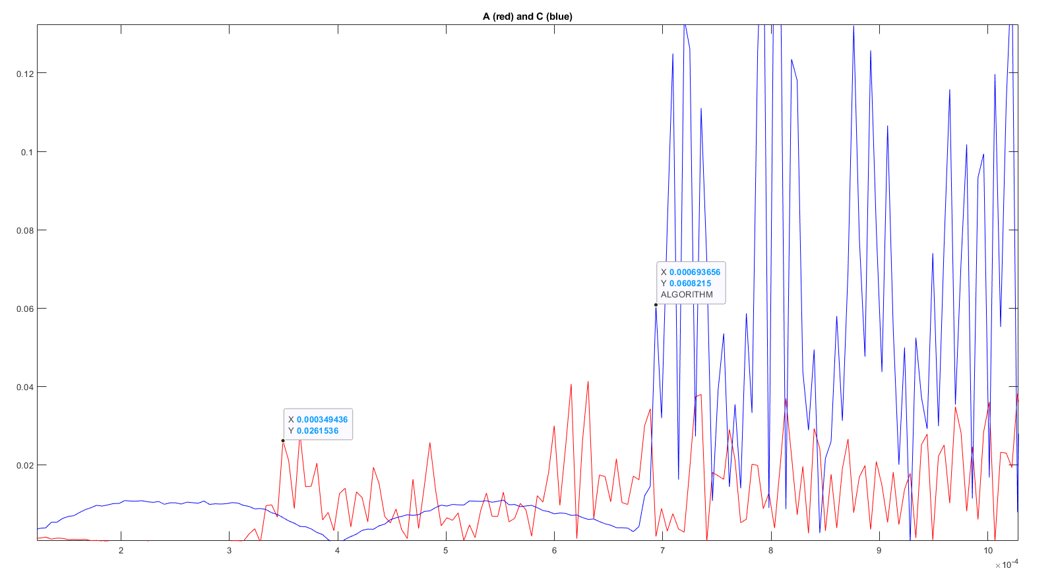


Figure 42: Algorithm chosen peaks for hydrophone pair A&C [42]

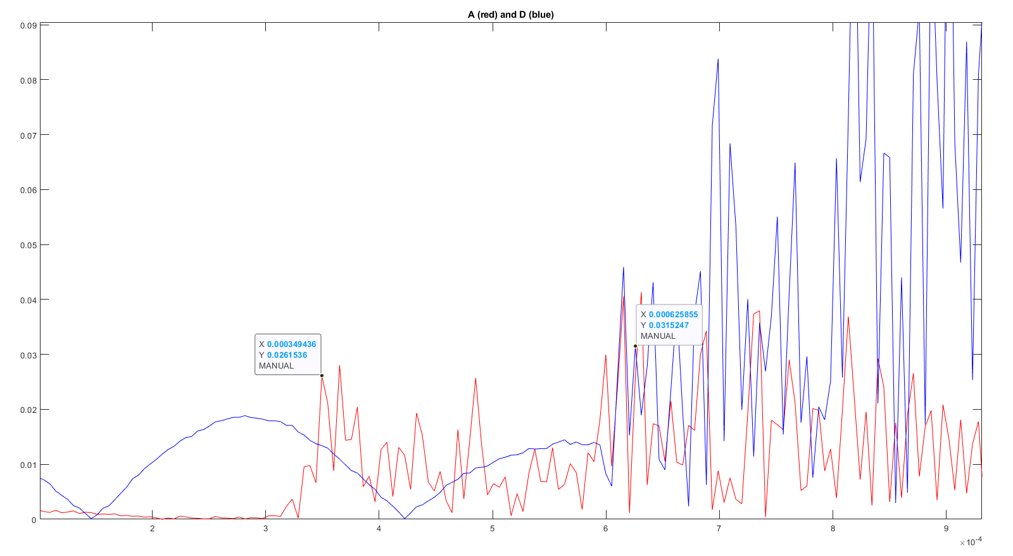


Figure 43: Manually chosen peaks for hydrophone pair A&D [43]

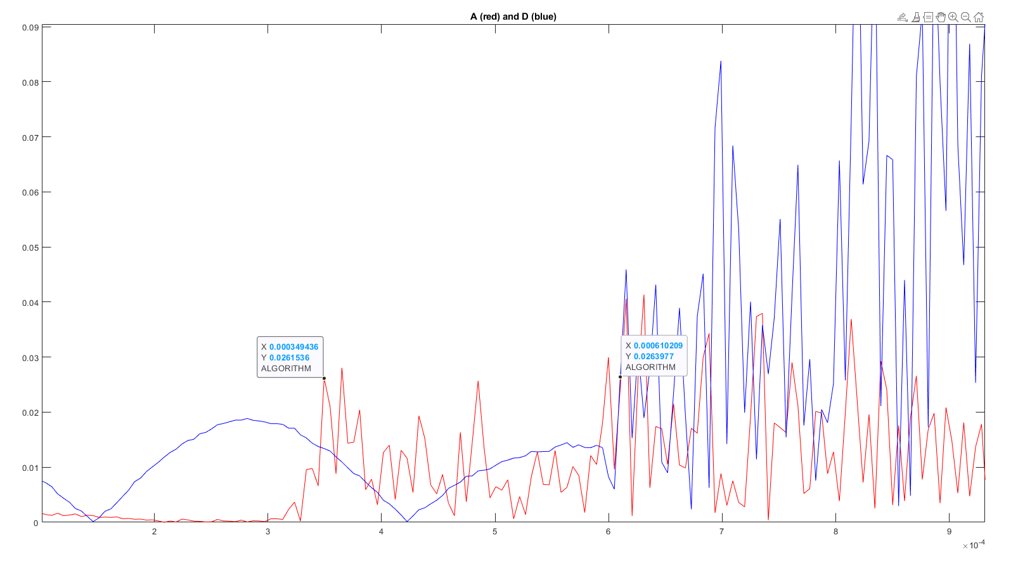


Figure 44: Algorithm chosen peaks for hydrophone pair A&D [44]

It can be seen from the above comparisons that the difference between the algorithm picked and manually picked peaks are at most about 10ish microseconds. It should be noted that this difference is due to a 1 peak shift which is very accurate to what would be manually picked, meaning that the algorithm is operating efficiently, and no more can be expected from it with the waveforms we can obtain. Looking further at the cross-correlation part of the TDOA algorithm, the window for the algorithm was defined to look in as accurately as possible and cannot expect any more accuracy given the technology. This is because constricting the window further will cut off the correct peaks that the algorithm should be finding. Looking at the efficiency of the best choice algorithm, and the accuracy constraint of the cross-correlation algorithm, it can be seen that the TDOA subsystem is working as intended, meaning that the subsystem has been verified.

## Verification of Hydrophone Array

### Spec 1.1.1

To verify that the hydrophones can detect sound frequencies in the 25 – 40 kHz range a pinger was set to 25 kHz for one test and 40 kHz for another in the water with the hydrophone array in the water as well and then recorded the audio. Then to verify that the hydrophones recorded the audio, the audio waveform graphs were looked at the audio waveform graphs in the time and frequency domain. The frequency domain shows the frequency spikes at 40 kHz in the unfiltered and filtered waveforms seen in the figures below, while the time domain waveforms show that the pinger noise is being recorded Figure 45 shows the time domain and frequency domain unfiltered audio, and figure 46 shows the filtered frequency spikes at 25 kHz

Sensor B, Time Domain 
0.2 
-0.2 
0.5 
1.5 
100 
50 
Sensor B, Frequency Domain 
x 24966.7 
Y 80.2312 
2 
2.5 
-0.8 
-0.6 
-0.4 
-0.2 
o 
0.2 
0.4 
0.6 
0.8 
*105 Figure 45: Unfiltered time (left) and frequency (right) waveforms with pinger at 25 kHz [45]

а.ДЕеааа 
B:frequency domain, after filteril 
100 
-0.8 
-0.6 
-0.4 
-0.2 
Frequency (10 
0.2 
0.4 
0.6 
0.8 
х 105 Figure 46: Filtered frequency waveform showing 25 kHz spikes [46]

Figure 47 below shows the time and frequency domain unfiltered waveforms with the pinger set at 40 kHz and Figure 48 shows the filtered frequency domain spikes at 40 kHz.

Sensor B, Frequency Domain 
0.5 
Sensor B, Time Domain 
1.5 
2.5 
200 
-0.8 
x 40106.8 
Y 336.527 
-0.6 
-0.4 
-0.2 
0.2 
0.4 
0.6 
0.8 
x 10 Figure 47: Unfiltered time (left) and frequency (right) waveforms with pinger at 40 kHz [47]

B:frequency domain, after filtering 
200 
-08 
-0.6 
-02 
Frequency (l O 
0.2 
0.4 
0.6 
0.8 
x 105 Figure 48: Filtered frequency waveform showing 40 kHz spikes [48]

The graphs show that the hydrophones are able to record the pinger audio at 25 and 40 kHz meaning that they are, in fact, able to detect sound waves within the required range.

### Spec 3.1.1

Verification of the dimensions of the built aluminum frame (specs 3.1.1, 3.1.2 and 3.1.3) requires simple length measurements of all the dimensions which was done with a tape measure. Spec 3.1.1 pertains to the height of the array which was measured to be 2.75 feet which is below the required height of 3 feet.

### Spec 3.1.2

Spec 3.1.2 requires the width of the frame to be less than 3 feet which, when measured, came to be 2.67 feet which is within the required length.

### Spec 3.1.3

Lastly, spec 3.1.3 requires the length of the frame to be within 6 feet which, when measured, came to be 3.0 feet, verifying the length of the frame is within 6 feet

### Spec 3.1.4

To verify the weight of the physical hardware (not the aluminum frame) which will be used on the RoboSub is below 5 pounds, the hydrophones, pre-amplifiers, and USB audio interface were all placed onto a scale to get the weight measurement which is shown below in figure 49.



Figure 49: Physical hardware being weighed on scale (hydrophones, pre-amps, and usb audio interface) [49]

The weight of the physical hardware was above the limit of 5 pounds which technically fails spec 3.1.4, however, after discussing with the RoboCats team this is not a problem and does not jeopardize the project. In the competition, having lower weight means the team gets more points, and since in comparison to the weight of the actual RoboSub the physical hardware is not nearly as much, this result is acceptable to the team. The team also has the option in the future to cut the length of the hydrophone wires to be shorter which will reduce the length and will most likely be done to make the RoboSub not have extra cord causing more clutter.

### Req 3.1

Req 3.1 is verification of the previous 4 specs (3.1.1 through 3.1.4), making sure that the recommended array for the team is competition conforming. Since specs 3.1.1 through 3.1.3 were verified, and 3.1.4 was passed by the team, this means that req 3.1 was also passed.

### Spec 3.2.1

To verify that the solution will take less than 10 minutes to adapt (change the defined locations of the hydrophones in the code) in the competition, the 3 RoboSub team members that will be participating in the competition were challenged with adapting the solution in under 10 minutes. After giving a walkthrough of the code and how to adapt the solution, each of the team members were timed and their recorded times are shown in the table below.

|  |  |
| --- | --- |
| Team Member | Time to adapt solution (mins : secs) |
| 1 | 1:04 |
| 2 | 0:38 |
| 3 | 0:42 |

Table 6

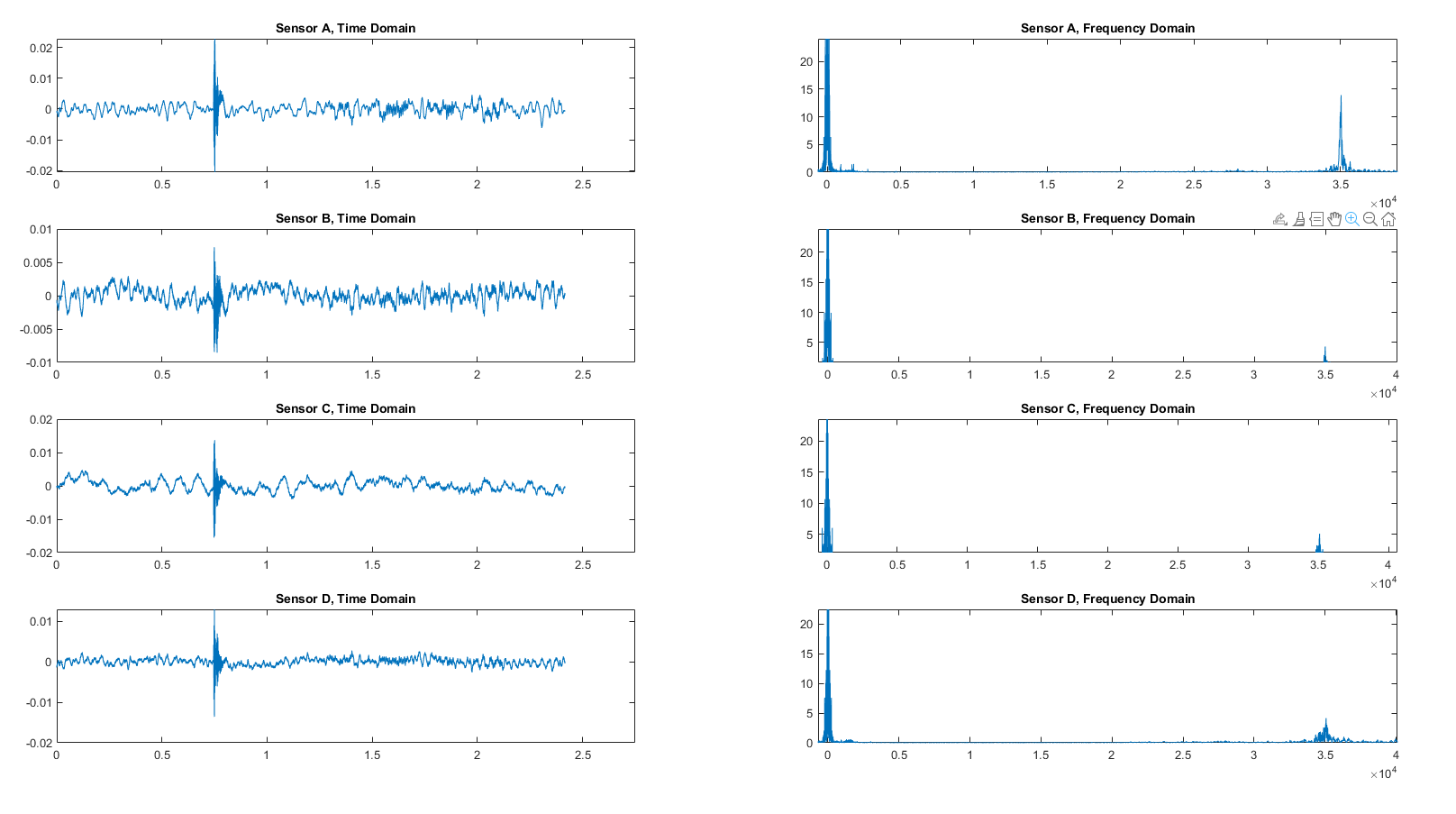
From the table shown it can be seen that all of the team members were able to adapt the solution in a much shorter amount of time than 10 minutes. While 3 samples/runs is not a lot, the times each team member takes would only decrease more and more as they became more adept at adapting the solution so spec 3.2.1 is passed.

## System-level verification

System level verification was carried out over the course of five consecutive Saturdays during the months of March and April in the spring of 2023 at Bozeman’s Swim Center. During all system-level testing, the team attached the hydrophone array to an aluminum frame. The frame dimensions accommodated hydrophone spacing which allowed for the largest possible TDOA values and sound source localization accuracy. The test frame with hydrophones attached was submerged, as were the ultrasonic beacons used for testing. The hydrophones were connected by wire and XLR termination to the Behringer UMC404HD USB audio interface, which in turn was connected to a laptop PC running MatLab code that carried out the system function. To test the system’s sound source localization accuracy and frequency filtration functionality, a three-dimensional coordinate system was established with one hydrophone at the origin and the ultrasonic beacons held at a known coordinates. The ultrasonic beacon's location was varied throughout so a range of directionality and distance could be tested. For each test, the hydrophone array recorded 2.5 seconds of raw audio data, which was then read into MatLab as a four-channel numeric array for processing. The 2.5-second-long listening window was chosen because it captures at minimum one and at most two chirps from an ultrasonic beacon, which is known to chirp one time every two seconds during the competition scenario.

### Req 1.1

Verification of this requirement was made by visual representations of data, i.e., plots of a 2.5-second-long hydrophone recording of an ultrasonic beacon in the time and frequency domains. Each testing event verified this requirement, as an amplitude spike could be clearly seen in the time domain plot. A corresponding amplitude spike was registered in the frequency domain plot both before and after filtering, at the frequency for which the beacon is known to be configured. Figure 50 below clearly shows the 35 kHz beacon amplitude spike registered in the frequency domain, and the correlating spike in the time domain.

Figure 50: 35 kHz detection, time & frequency domains [50]

### Req 1.2

Two ultrasonic beacons, one configured for 35 kHz and the other configured for 51 kHz, were used for the verification of this requirement. Time domain plots for this test registered one amplitude spike for the 35 kHz beacon, and four smaller amplitude spikes from the 51 kHz beacon. It should be noted here that a 2nd competition beacon for use in testing was prohibitively expensive given the project budget, so a cheaper alternative was acquired. The alternative beacon was found to reliably chirp four times in two seconds, and its frequency domain amplitude spike was found to occupy the five to six integer ultrasonic frequencies surrounding the frequency for which it was configured.

The data shown below in figures 51 through 53 illustrates that the digital filter reliably filtered out not only the data in frequencies occupied by the 2nd beacon used in testing, but all other sources of noise as well.

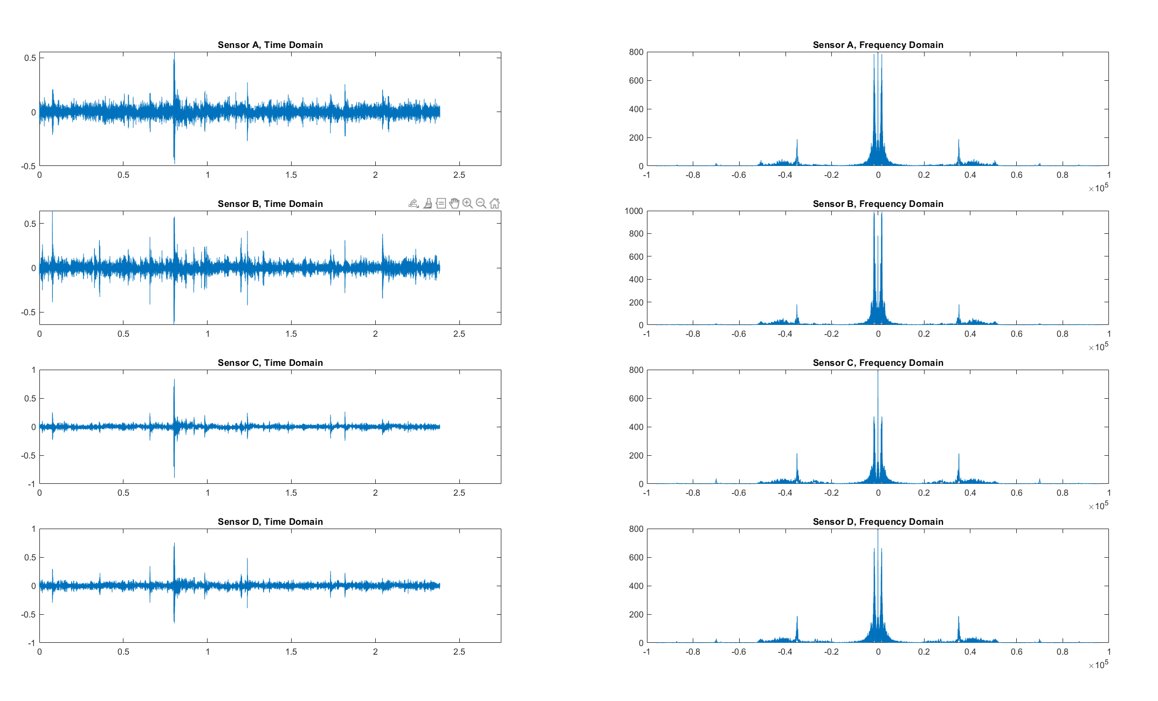
Figure 51: Detection of 35 kHz & 51 kHz beacons, time & frequency domains, pre-filter [51]

Figure 51 shows a zoomed in view of the frequency domain plots from each sensor, so that the 35 kHz and 51 kHz beacon amplitude spikes, as well as noise from an unknown source, can be seen more clearly.

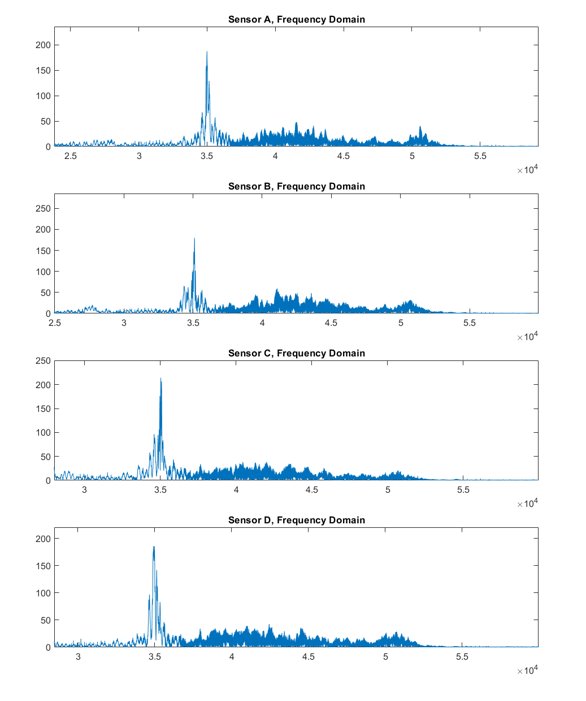
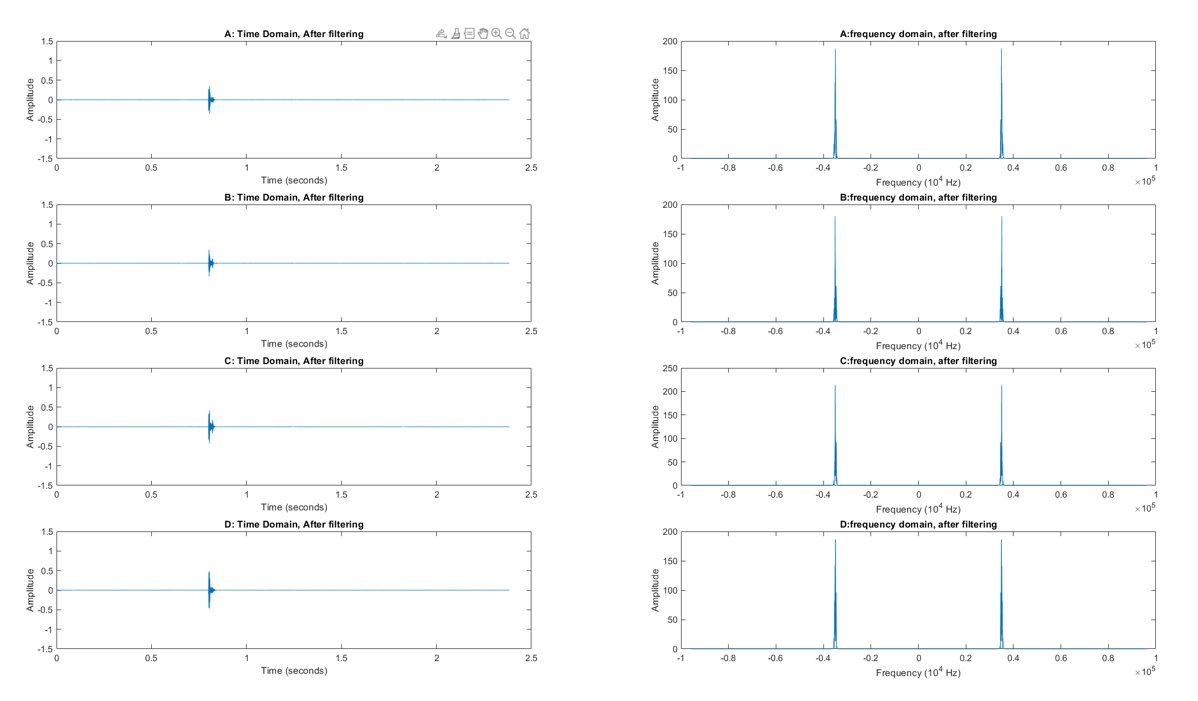


Figure 52: Detection of 35 kHz & 51 kHz beacons, frequency domain close-up, pre-filter [52]

Figure 53 shows the same plots as from figure 51, but after application of a bandpass filter at the 35 kHz frequency. The 35 kHz beacon amplitude spike is all that remains in the time and frequency domain plots.

Figure 53: 35 kHz & 51 kHz beacons, time & frequency domains, post-filter [53]

### Req 3.2

To verify that the hydrophones can be placed in different configurations without destroying the functionality of the system, the placement of the hydrophones were flipped along the y-z axis and the result would be observed. Req 3.2 was not able to be verified because the multilateration algorithm couldn’t work more than about 50% of the time. Without a known pattern to follow/detect, there is no way of changing the hydrophone placements and knowing that the behavior of the system is the same or similar. This problem is also related to the verification of objective 2 and has been diligently worked on for weeks to no avail, it is believed this to be a limitation placed by the hardware. Due to this req 3.2 is not verified and requires more testing.

### Obj 1

Objective 1 is stated as, “The system must be able to detect a pinger.” Verification of objective 1 required was conducted at the Bozeman Swim Center. The system’s sensor array was mounted onto the aluminum test frame and submerged. Beacons emitting ultrasonic frequency signals were submerged at known locations. When the system software was run, a 2.5-second-long recording was read into MatLab for processing. For testing and verification purposes, plots showing the time and frequency data of the recording were also produced. As stated above in the descriptions for verification of requirements 1.1 and 1.2, the requirements, and therefore the objective, are passed when the system can both register the signal emitted by a beacon, and filter for one beacon when multiple are active.

Figures 54 through 56 below use the data collected from a single test to show that the system is able to register the signals of multiple competition beacons. Figure 54 clearly shows time and frequency data from each beacon, and the system’s ability to detect sounds of relevant frequencies and satisfy requirement 1.1 is demonstrated. Figure 55 shows a closer view of the frequency domain amplitude spikes of both active beacons at 35 kHz and 51 kHz, as well as some unwanted noise. Figure 56 shows the effectiveness of the digital filter in removing unwanted frequency data from the recording, demonstrating the system’s ability to select one pinger from multiple and satisfy requirement 1.2

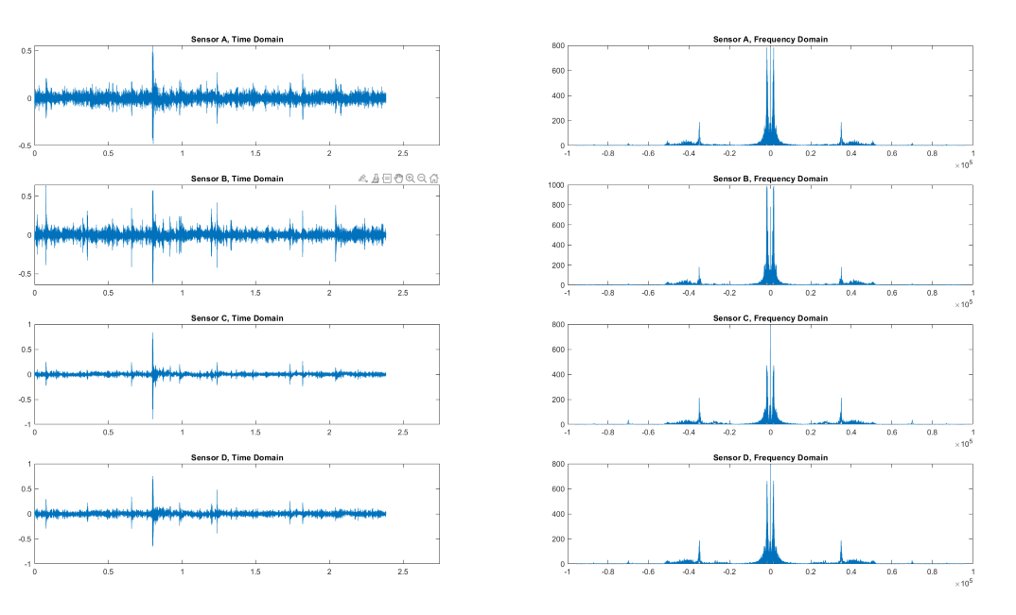


Figure 54: Raw time and frequency domain waveforms recorded [54]

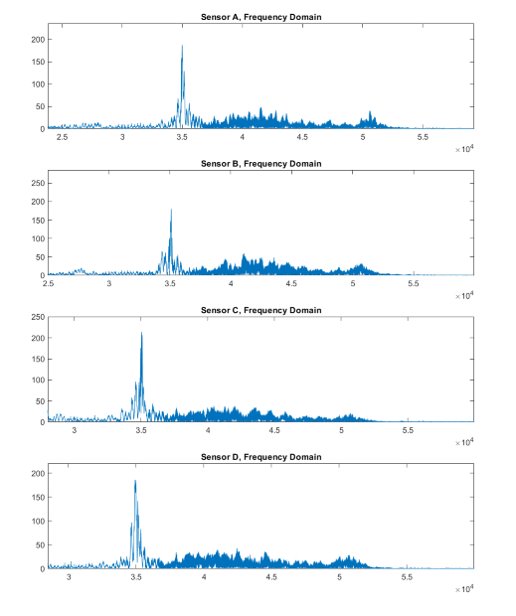


Figure 55: Raw frequency data [55]

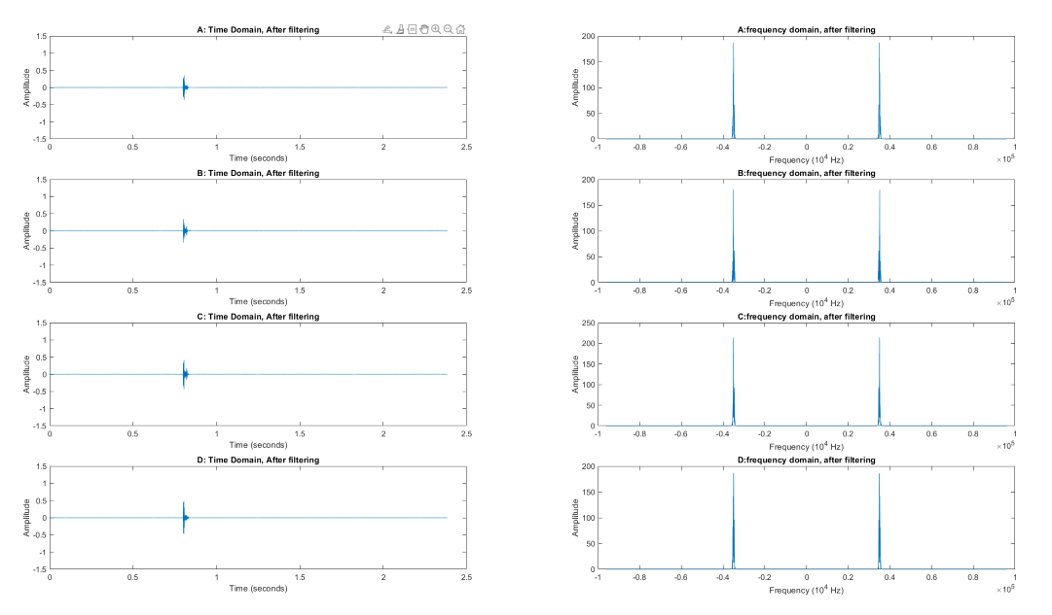


Figure 56: Filtered time and frequency domain data [56]

These results are convincing evidence that the system is capable of carrying out objective 1. With objective 1 satisfied, the system can deliver critical information to the next critical subsystem in the system’s functional chain.

### Obj 2 and Req 2.1

Due to the application of the sound localization system, water tests were an essential part of system testing. However, due to the lack of pools in the Bozeman area, it was necessary to verify that the system was working prior to putting it into the water. In order to do this, an in-air test was performed. During this test, a sound was emitted a known distance from the hydrophone array in the lab and the location system was run. The average results over 5 in air tests can be seen in the table below. As can be seen all of the tau values are very close to what was expected, and the solution provided is reasonable. Due to these in-air tests, it was determined that the system was functional and ready to be tested in water.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pinger Location | Expected AB | Actual AB | Expected AC | Actual AC | Expected AD | Actual AD | Calculated Location |
| (-1.677,-1.626,0) | 0.0034 | 0.00299 | 0.0029 | 0.00237 | 0.0027 | 0.00219 | -0.6482 -1.3994 0.6160 |

Table 7

To verify the accuracy of the system in water, a series of pool tests were conducted. The pinger was placed into the pool at various known locations emitted a sound at 35 kHz. The system was run and calculated location of the pinger was recorded. Additionally, the audio files recorded from each hydrophone were saved to further aid in debugging and verification.

In running these pool tests, the team encountered significant variation in the accuracy of the system. In around half of the tests run, the correct direction of the pinger was calculated with varying accuracy. However, in the remaining half, the directionality of the solution was not correct in either the y direction, x direction, or both. This resulted in the team’s inability to verify the accuracy spec of Req 2.1 and the subsequent Objective 2.

After analysis, it was determined that the root of these inconsistencies was noise in the TDOA calculations. In order to come to this conclusion, expected TDOA values were calculated for each hydrophone pair and compared to what was calculated from the pool test. It was from this comparison that large differences between expected TDOA values and what was calculated were found. This noise can be seen in the table below. As can be seen in some cases, the expected tau values are close to what can be expected; however, in others the values can be off by an order of magnitude or even have the wrong sign. This results in the appearance that the incorrect hydrophone is closer to the pinger. A substantial portion of time was spent debugging this issue.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Pinger Location** | **Expected AB** | **Actual AB** | **Expected AC** | **Actual AC** | **Expected AD** | **Actual AD** |
| (-.61, -1.37,0) | -7.55E-04 | 6.25E-04 | -6.21E-04 | -1.04E-05 | -5.53E-04 | 1.56E-05 |
| (-6.4, -5.89, 0) | -7.80E-04 | -1.04E-04 | -6.19E-04 | -3.44E-04 | -6.02E-04 | -2.60E-04 |
| (-2.12,-3.9,-1.01) | -7.54E-04 | -9.90E-05 | -5.66E-04 | -1.46E-04 | -5.35E-04 | -1.56E-04 |

Table 8

Upon investigation, it was determined that the TODA system was incorrectly setting the window size by creating a different time window for each Indvidual hydrophone. This bug was causing the hydrophone raw audio to become unsynchronized, causing the incorrect TDOA values. The results of our TDOA calculations following this fix can be seen in the table below. As can be seen in the table, the fixed TDOA values are much closer to the correct values and there were no cases of having the incorrect sign.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** |  |  |  | **Expected** |  |  |
| TDOA (AB) | TDOA (AC) | TDOA (AD) |  | TDOA (AB) | TDOA (AC) | TDOA (AD) |
| -7.61E-04 | -5.32E-04 | -4.90E-04 |  | -7.53E-04 | -5.64E-04 | -5.33E-04 |
| -9.02E-04 | -5.11E-04 | -4.53E-04 |  | -7.53E-04 | -5.64E-04 | -5.33E-04 |
| -8.81E-04 | -5.42E-04 | -5.01E-04 |  | -7.81E-04 | -6.32E-04 | -6.08E-04 |
| -7.98E-04 | -5.74E-04 | 5.16E-04 |  | -7.81E-04 | -6.32E-04 | -6.08E-04 |
| -7.46E-04 | -5.89E-04 | -5.32E-04 |  | -7.81E-04 | -6.32E-04 | -6.08E-04 |
| -7.56E-04 | -5.84E-04 | -5.42E-04 |  | -7.81E-04 | -6.32E-04 | -6.08E-04 |
| -7.56E-04 | -6.00E-04 | -5.48E-04 |  | -7.81E-04 | -6.32E-04 | -6.08E-04 |
| -7.51E-04 | -6.26E-04 | -5.74E-04 |  | -7.64E-04 | -6.44E-04 | -6.25E-04 |
| -7.04E-04 | -6.10E-04 | -5.22E-04 |  | -7.64E-04 | -6.44E-04 | -6.25E-04 |
| -7.25E-04 | -5.84E-04 | -5.22E-04 |  | -7.80E-04 | -6.20E-04 | -6.03E-04 |
| -8.45E-04 | -5.79E-04 | -5.22E-04 |  | -7.80E-04 | -6.20E-04 | -6.03E-04 |
| -7.25E-04 | -6.94E-04 | -5.79E-04 |  | -7.58E-04 | -7.16E-04 | -6.46E-04 |
| -7.56E-04 | -7.15E-04 | -5.84E-04 |  | -7.58E-04 | -7.16E-04 | -6.46E-04 |
| -7.51E-04 | -7.35E-04 | -5.95E-04 |  | -7.58E-04 | -7.16E-04 | -6.46E-04 |
| -8.19E-04 | -6.10E-04 | -4.95E-04 |  | -7.53E-04 | -5.64E-04 | -5.33E-04 |
| -7.25E-04 | -5.53E-04 | -4.64E-04 |  | -7.53E-04 | -5.64E-04 | -5.33E-04 |
| -7.15E-04 | -4.95E-04 | -4.17E-04 |  | -7.53E-04 | -5.64E-04 | -5.33E-04 |
| -8.29E-04 | -1.30E-04 | -2.81E-04 |  | -7.81E-04 | -6.32E-04 | -6.08E-04 |

Table 9

In spite of these solutions to the issue, Objective 2 was still not able to be verified. While the calculated tau values are much closer to what can be expected using physics, there is still some error in the calculations. Due to the speed of sound in water compared to air, this error results in most runs, (35 out of 39 runs), not being able to calculate a beacon location. Due to the initial errors in the system, the team was not able to devise a solution to this problem.

### Obj 3

To verify that the system integrates with the existing RoboSub the output of the code was confirmed to be given in the python data type, and the physical hardware was confirmed to fit within the capsule of the RoboSub. To confirm the data type of the output, the code was simply run with two situations where one had a duplicate answer and the other a real answer. These two outputs are shown in figures 57 and 58 below.

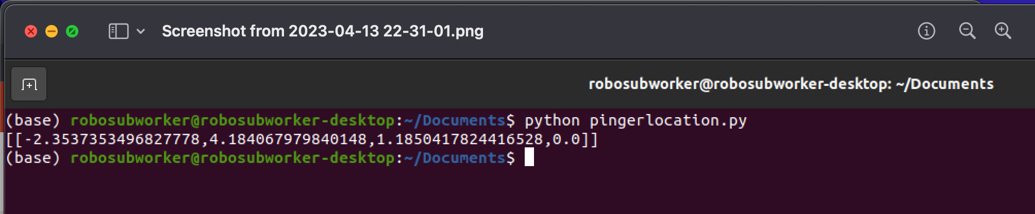


Figure 57: Output in python data type for real solution

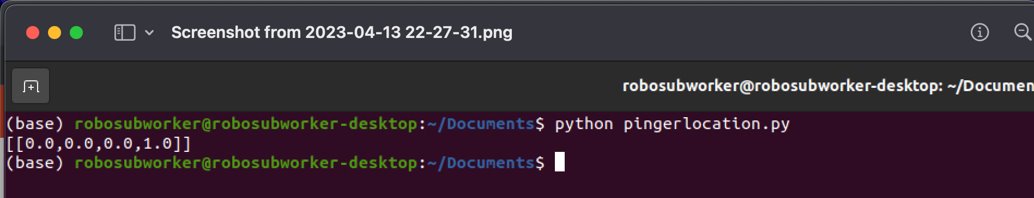


Figure 58: Output in python data type for duplicate answer

From the figures above it can be seen that the output of the code is indeed a python data type, meaning that the RoboSub PC can use the answer provided by our system. Next, the physical hardware was all placed inside of the RoboSub capsule which can be seen in Figures 59 and 60 below.

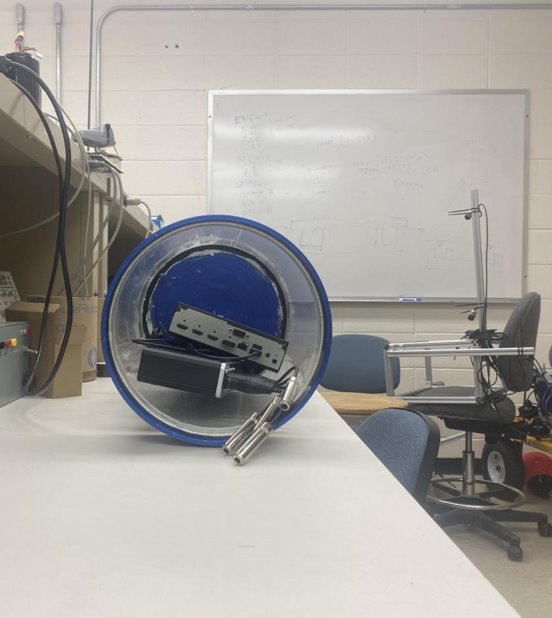


Figure 59: Physical hardware inside of RoboSub capsule

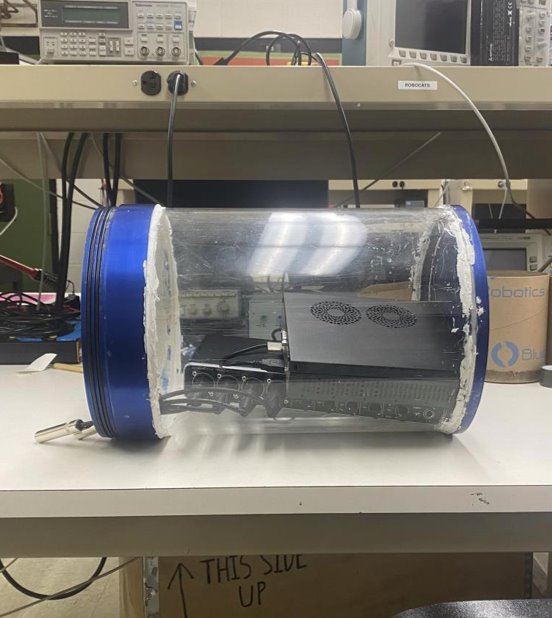


Figure 60: Physical hardware inside of RoboSub capsule

From Figures 59 and 60, it has been demonstrated that the physical hardware is able to fit inside of the RoboSub capsule. With the code outputting the correct python data type, and the physical hardware fitting inside of the capsule, objective 3 has been successfully verified.

### Full System Verification

Unfortunately, the full system could not be verified, as Objective 2 could not be passed. While the project successes include verification of Objectives 1 & 3, the project was ultimately a failure due to not being able to reliably output accurate location data. The project’s partial success includes its audio recording functionality, pinger detection, frequency filtration, and existing RoboSub integration. Some suggestions for addressing the project failures are listed below in section 5.2 titled Future Work.

# Conclusions

## Summary of Project Results

Overall, a lot of effort was put into this project by the team, and while not all objectives and expectations were met, there is still a system that is partially operational and only has few discrepancies that need to be addressed for a fully working system. In total, two out of the three objectives created were met, and the third is fairly close to being operational. The Objectives that worked (1 and 3) are related to filtering for the correct pinger out of multiple options and being integratable with the RoboSub. The objective that failed (2) had to do with the locating of the pinger, and while it was technically failed there was still merit to the result. It may not have been perfect, but the system could compute the correct direction of the pinger approximately 50% of the time, meaning that with a few tweaks or new hardware, there would be a success rate above 50% which could then give lead to finding patterns and make the system operatable for the RoboSub team. However, the reality is that this objective failed, which makes the project a technical failure because the most important aspect was not accomplished.

The impact of the work on the project should not be understated. While the current system is not consistently successful, RoboCats could improve on it. They are already two thirds of the way there and may find the problem to be something subtle with an easy fix, leading to a fully operational system. Currently there is a lack of tools, knowledge, and time required to find this issue.

What was accomplished here is notable and has paved a path for the RoboCats team to follow and hopefully lead them to the success that was not achieved. Looking at this project as a total failure would be an unfair disservice to the work that was put in, and results achieved. For the time frame given and having a technical project with intricacies unseen to us, what was achieved is admirable. The system can filter for the correct pinger frequency and integrates with RoboCats’ existing setup. Although inconsistent, there is still a pinger location output by the system that will give the correct location of the pinger. Some aspects of this project were successful and provide RoboCats a solid foundation for solving their sound source localization problem in the future.

## Future Work

The RoboSub Sound Localization team has recommendations for any who would like to continue the work done on this project. The current system can record audio from a hydrophone array and USB audio interface. It can filter the raw audio for specific frequency data and derive TDOA values from the processed audio. The system functionality breaks down when it attempts to use TDOA values to solve for the sound source location using multilateration. Given that the multilateration algorithm can solve sound source locations using simulated TDOA values, the team suspects that the error lies in the system’s ability to derive TDOA values accurately. The project team suspects the following potential sources of the TDOA inconsistencies:

* The sensitivity of the hydrophones is not consistent across each unit. A more precise and rigorously standardized hardware product might be necessary for consistent system performance.
* The speed of sound in water is high enough that a sampling rate of 192 kHz is insufficient for providing the necessary resolution for accurately deriving TDOA values from ultrasonic beacons. Though prohibitively expensive, an audio recording solution which can sample at 384 kHz might be worth exploring.
* The system’s TDOA algorithm cannot consistently trigger on the correct peaks of the signal waveform. Application of more advanced audio processing techniques may improve TDOA accuracy and allow the multilateration algorithm to consistently solve sound source locations.

The team recommends that any future work on this project be directed to the above-listed potential sources of error. This is an exciting project to work on and a fantastic opportunity for engineering students to practice, apply, and expand on the skills learned in the Electrical Engineering curriculum. Continuing this project and bringing the system to a fully operational state would surely be rewarding.

# Bill of Materials

|  |  |  |
| --- | --- | --- |
| **Component Name** | **Link to order** | **Cost** |
| Audio Interface | Berhinger USB Audio Interface | **$250** |
| Hydrophones | **Aquarian Audio Hydrophones** | **$318.52** |
| Array Aluminum | **T Slot 2020 Aluminum** | **$219.52** |
| Array Connectors | **Aluminum Fastener T-Solt Nuts and Bolts** | **$20** |
| Amplifiers | **Hydrophone Buffer Amp** | **$158.16** |

# Acknowledgements

The team would like to thank our advisor, Dr. Whitaker, for all the time spent with each of the team members working on getting the system operational, . Even though we were unsuccessful in the end, the journey with him as our instructor was worth the experience and we are very appreciative of his time and effort.

The team would also like to thank Fiona of RoboCats for showing up to our weekly meetings. Her presence and positivity were appreciated. RoboCats was extremely helpful in the process, and we thank them for sponsoring this project.

# References

[1] Kiehn B, Bonner G, Geroge N, Smith H. “RoboSub Senior Capstone Final Report”, Electrical Engineering Design II, 2022

[2] “2022 Team Handbook (V2),” *RoboSub*, 2022. [Online]. Available:

<https://robonation.org/app/uploads/sites/4/2022/05/2022-RoboSub_Team-> Handbook\_v2.0.pdf. [Accessed: 29-Sep-2022].

[3] “Waveform introduction,” *Introduction - Waveform Introduction*. [Online]. Available: <https://support.ircam.fr/docs/AudioSculpt/3.0/co/Acoustic%20Notions_1.html>. [Accessed: 27-Sep-2022].

[4] Oceanexplorer.noaa.gov. 2022. *NOAA Ocean Explorer: Sounds in the Sea 2001: diagram of how sound travels underwater*. [online] Available at: <https://oceanexplorer.noaa.gov/explorations/sound01/background/acoustics/media/sofar.html> [Accessed 27 September 2022].

**[5]** **Oceanservice.noaa.gov. 2022.***How far does sound travel in the ocean?*. [online] Available at: <https://oceanservice.noaa.gov/facts/sound.html> [Accessed 27 September 2022].

[6] Li X, Shen M, Wang W, Liu H. “Real-Time Sound Source Localization for a Mobile Robot Based on the Guided Spectral-Temporal Position Method,” International Journal of Advanced Robotic Systems. 2012;9(3). [Accessed 28-Sep- 2022].

[7] Meng, Wei, and Wendong Xiao. "Energy-based acoustic source localization methods: a survey." *Sensors* 17.2 (2017): 376. [Accessed 28-Sep-2022].

[8] C. Junu Jahana, M. Sinith and P. Lalu, "Direction Of Arrival Estimation using Microphone Array," *2021 Fourth International Conference on Microelectronics,*  *Signals & Systems (ICMSS)*, 2021, pp. 1-6. [Accessed 28-Sep-2022].

[9] Discovery of Sound in the Sea. 2022. *Hydrophone/Receiver*. [online] Available at: <https://dosits.org/galleries/technology-gallery/basic-technology/hydrophonereceiver/> [Accessed 27 September 2022].

[10] Oceanservice.noaa.gov. 2022. *What is a hydrophone?*. [online] Available at: <https://oceanservice.noaa.gov/facts/hydrophone.html> [Accessed 27 September 2022].

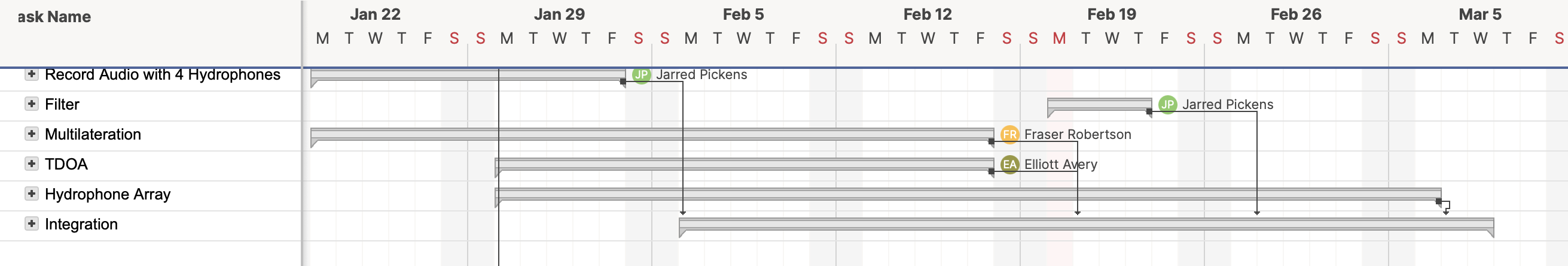
[11] Williams, E., 2022. *The Inverse Problem: Cylindrical NAH*. [online] Science Direct. Available at: <https://www.sciencedirect.com/topics/engineering/hydrophones> [Accessed 27 September 2022].

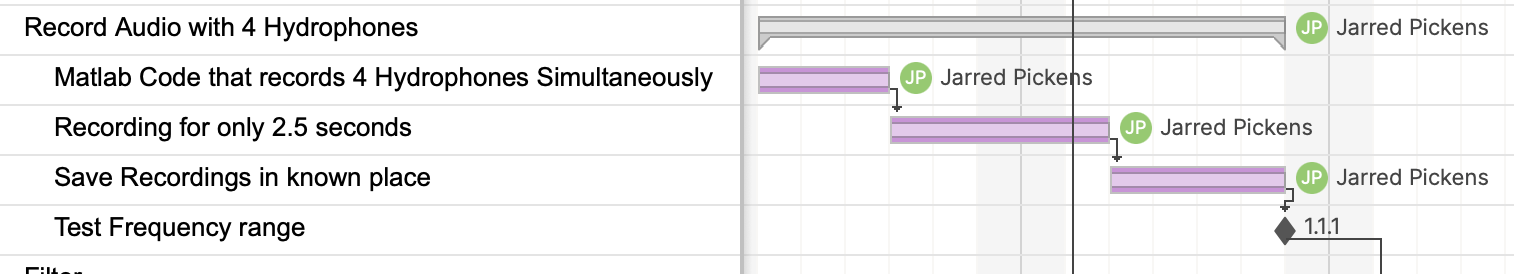
[12] “Guide to Computer Programming & Coding languages,” *Code a New Career |*  *ComputerScience.org*, 19-Sep-2022. [Online]. Available: <https://www.computerscience.org/resources/computer-programming-languages/>. [Accessed: 29-Sep-2022].

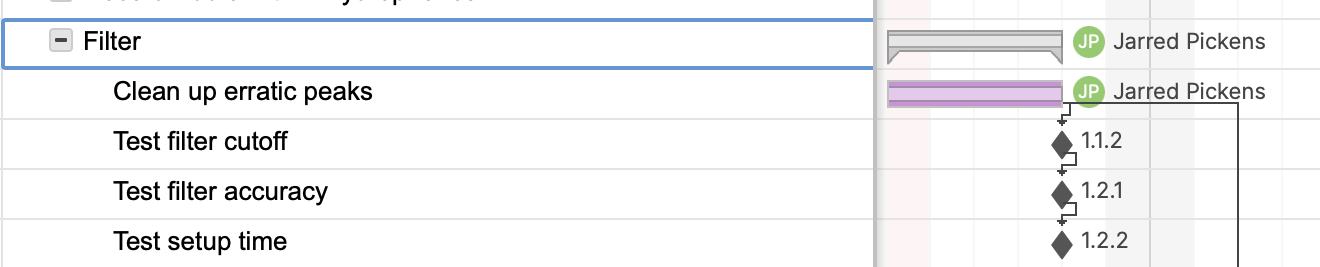
[13] “Integrating Python With Other Languages,” *Integrating Python With Other Languages - Python Wiki*, 2020. [Online]. Available: <https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages>. [Accessed: 29-Sep-2022].

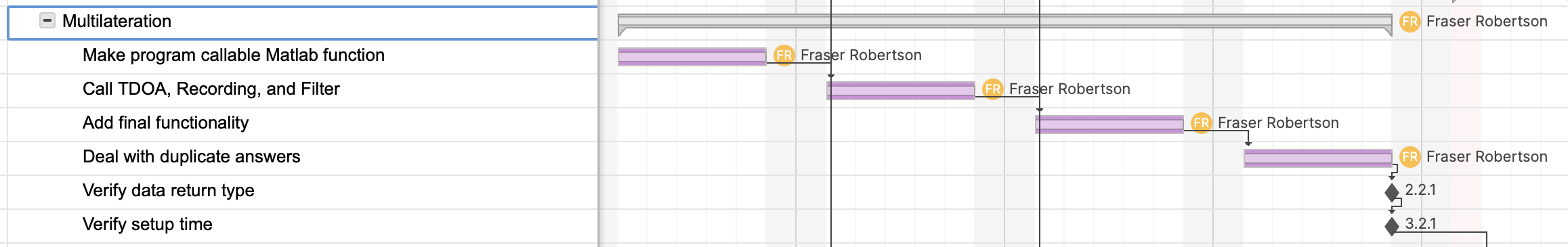
[14] “Matlab.engine.matlabengine,” *Call MATLAB Functions from Python - MATLAB & Simulink*, 2022. [Online]. Available: <https://www.mathworks.com/help/matlab/matlab_external/call-matlab-functions-from-python.html>. [Accessed: 29-Sep-2022].

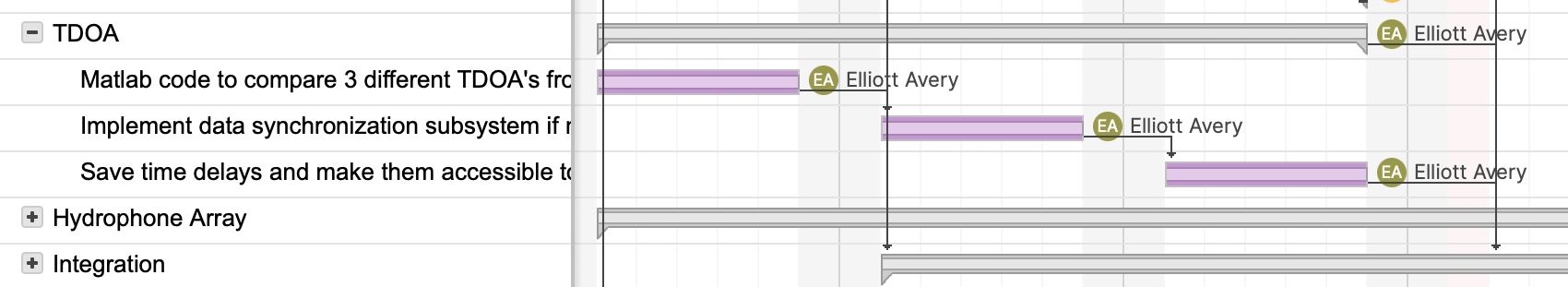
# Appendix A (Fabrication Plan)

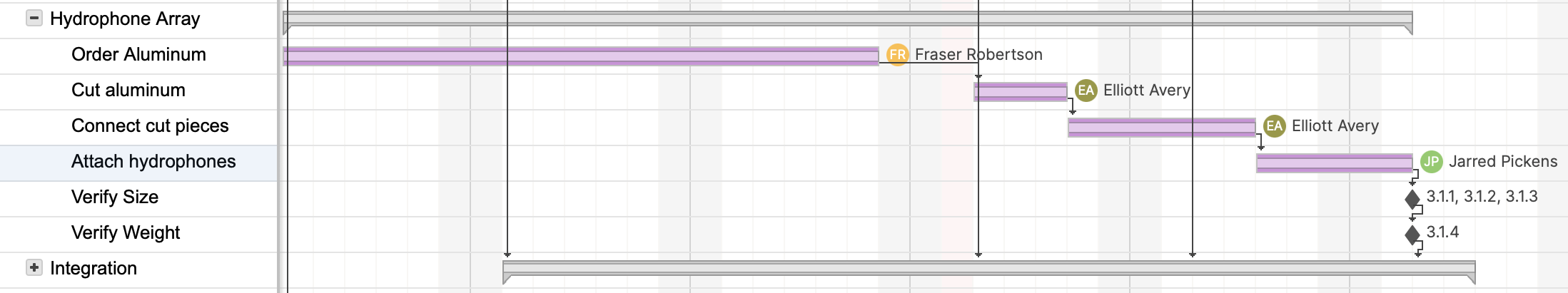


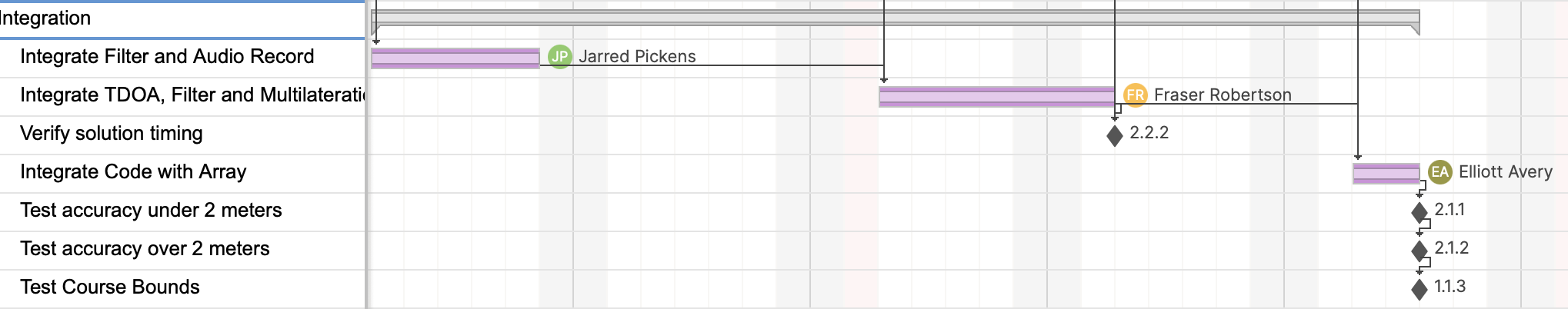












# Appendix B (Verification Plan)

Test Plan: Objective 1

**Objective 1- The system must be able to detect a specified pinger**

Individual in charge of test- Fraser Robertson

Date of Test: 03/24/2023

Experiment Description:

Two beacons of different frequencies between 25 and 40 kHz will be placed in the pool at a random distance from the hydrophone array. The system will then be calibrated for the frequency of one of the beacons.

Equipment:

Hydrophone array, two pingers, computer with appropriate software-full code system, swimming pool, usb hub with adapters

Procedure:

1. Randomly select desired frequency of two pingers from 25 to 40 kHz
2. Randomly select location for each pinger
3. Verify that pingers are not buzzing at the same time
4. Set software to filter for one of set pinger frequencies
5. Place pinger and full hydrophone array in pool
6. Run full multilateration software- including audio record, filtering, TDOA and multilateration

Repeatability:

Repeat test with 10 random combinations of pinger frequencies, and pinger locations

Pingers should be placed in all directions from hydrophone array

Data Collection:

Filtered .wav files will be saved from each hydrophone and time and frequency domain plots of those signals will be created. Each hydrophone will be separately verified for the correct pinger signal

Pass Criteria:

Only correct frequency visible in every hydrophone .wav file for each test

**Requirement 1.1 - The system must be able to detect sounds of specific frequencies**

Individual in charge of test- Elliot Avery

Date of test: 03/23/2023

Experiment Description: A random pinger frequency that is within competition ranges will be set and the system will attempt to detect and record it

Equipment:

Hydrophone, pinger, computer with appropriate software- filtering and audio record software, swimming pool.

Procedure:

1. Randomly select desired frequency of the pinger from 25 to 40 kHz
2. Place pinger a known random distance from hydrophone array
3. Set pinger frequency
4. Place pinger and hydrophone in pool
5. Record .wav file for 2.5 seconds
6. Run filtering software

Repeatability:

Repeat test with 10 random combinations of frequency and distance from hydrophone array

Data Collection:

Time and frequency domain data showing detection of signals of each frequency:

Time & frequency domain plots of detected signals before and after filtering. Data will be processed to determine an adequate input ratio is achieved.

Pass Criteria:

Correct pinger frequency seen in every filtered .wav file- 100% Success

**Spec 1.1.1 - Detect in the range of 25 kHz to 40 kHz.**

Individual in charge of test- Jarred Pickens

Date of test: 02/10/23

Experiment Description: A pinger will be set for a variety of frequencies from 25 kHz to 40 kHz to ensure hydrophones work in required bounds.

Equipment:

Hydrophone, usb hub, pinger, and computer with audio record software.

Procedure:

1. Place pinger a known distance from hydrophone
2. Set pinger to 25 kHz
3. Record audio file for 2.5 seconds
4. Save audio file

Repeatability:

Repeat test from 25 kHz to 40 kHz at integer frequencies

Data Collection:

Frequency peak in audio file should be seen for each frequency. This can be done by taking Fourier transform of audio file and plotting

Pass Criteria:

Correct pinger frequency seen in every audio file- 100% Success

**Spec 1.1.2 - Differentiate between 1kHz gaps.**

Individual in charge of test- Jarred Pickens

Date of test: 2/23/23

Experiment Description: A Matlab waveform will be created at different frequencies between 25 to 40 kHz. From this we will verify the filter can differentiate 1 kHz gaps.

Equipment:

1. Computer with appropriate software (MatLab)
2. MatLab code to generate a signal consisting of superimposed sinusoids of each kHz integer frequency between 25 and 40 kHz.
3. MatLab code to plot the generated signal in the time and frequency domains.
4. MatLab code that filters the generated signal for each integer kHz frequency between 25 kHz and 40 kHz.
5. MatLab code that plots each filtered signal in the time and frequency domains.

Procedure:

1. Generate 2.5 second signal consisting of each integer kHz frequency between 25 & 40 kHz and plot in time and frequency domains.
2. Filter the generated signal for each integer kHz frequency between 25 kHz and 40 kHz.
3. Plot each filtered instance of the generated signal in the time and frequency domains.

Repeatability:

Repeat 1000 times

Data Collection:

Plots of the generated signal pre and post filtering in the time and frequency domains

A histogram showing the success of the filter for each frequency

Pass Criteria:

Only correct pinger frequency seen in every filtered numeric audio array.

100% filter success rate

**Spec 1.1.3 - System must work in 24M x 12 M x 5 M Course.**

Individual in charge of test- Jarred Pickens

Date of test: 03/08/23

Experiment Description: A pinger will be placed an increasing distance from hydrophone in the pool and we will verify that waveform can be detected in audio file.

Equipment:

Hydrophone, pinger, computer with audio record software, pool

Procedure:

1. Set pinger frequency to 30 kHz
2. Set desired frequency in code as 30 kHz
3. Place pinger 1 meter from hydrophone array in pool
4. Record .wav file for 2.5 seconds

Repeatability:

Repeat test with pinger 2 meters, 5 meters and 10 meters from array or to the largest distance we can test in pool

This test should be repeated in all directions from hydrophone

Data Collection:

Time and frequency data for each test with signal to noise ratio of desired frequency at each distance, then plotting following the trend to determine if system will work along super diagonal of the course

Pass Criteria:

Plotted trend of accuracy shows that there will be discernible beep within bounds of course

Req 1.2 The system must be able to detect specified pinger from multiple in pool

Individual in charge of test- Jarred Pickens

Date of test: 03/03/2023

Experiment Description: Multiple pingers will be set at different arbitrary frequencies and the system will attempt to filter and record just one of them

Equipment:

Hydrophone, two pingers, computer with appropriate software, swimming pool.

Procedure:

1. Select two random pinger frequencies from 25-40 kHz
2. Verify that pingers are not buzzing at the same time
3. Select one of the random frequencies to filter for
4. Place pingers in random areas of the pool
5. Place hydrophone in pool
6. Record .wav file for 2.5 seconds
7. Run filtering software

Repeatability:

Repeat test 10 times with continuing random frequencies and distances

Pinger should be placed in all directions from hydrophone

Data Collection:

Time & frequency domain plots of detected signals before and after filtering. Data will be processed to determine an adequate signal to noise ratio is achieved. In the filtered plots, verify that only the correct frequency can be seen

Pass Criteria:

Only correct frequency visible in every hydrophone .wav file for each test

Spec 1.2.1 100% accuracy in selecting correct pinger

Individual in charge of test- Jarred Pickens

Date of test: 02/23/23

Experiment Description: Generate a signal in Matlab at different frequencies from 25 to 40 kHz. The frequency data survives if the appropriate filter is applied.

Equipment:

1. Computer with appropriate software (MatLab)
2. MatLab code to generate a signal consisting of a sinusoid at a kHz integer frequency between 25 and 40 kHz with background noise.
3. MatLab code to plot the generated signal in the time and frequency domains.
4. MatLab code that filters the generated signal for each integer kHz frequency between 25 kHz and 40 kHz.
5. MatLab code that plots each filtered signal in the time and frequency domains.

Procedure:

1. Generate 2.5 second signal consisting of a integer kHz frequency between 25 & 40 kHz and plot in time and frequency domains.
2. Filter the generated signal for the correct integer kHz frequency between 25 kHz and 40 kHz.
3. Plot each filtered instance of the generated signal in the time and frequency domains.

Repeatability:

Repeat for each integer frequency between 25 and 40 kHz times

Data Collection:

Plots of the generated signal pre and post filtering in the time and frequency domains

A histogram showing the success of the filter for each frequency

Pass Criteria:

Only correct pinger frequency seen in every filtered numeric audio array.

100% filter success rate

Spec 1.2.2 Less than 5 minutes for team to setup frequency for system to locate pinger

Individual in charge of test- Fraser Robertson

Date of Test: 2/23/23

Experiment Description: RoboSub team members will be given an overview of our solution, then will be timed when setting the desired frequency of the system

Equipment:

Computer with multilateration software, random test subject, pinger

Procedure:

1. Select random frequency
2. Give random subject computer with software
3. Tell subject correct frequency to set software to
4. Start timer
5. Tell subject to setup code to filter for correct frequency
6. Stop timer when subject is done
7. Verify that frequency has been selected appropriately

Repeatability:

Repeat test with 3 different random subjects

Data Collection:

Collect timing information for all subjects. Verify that all subjects are able to change frequency in under 5 minutes

Pass Criteria:

Each test subject is able to correctly change frequency in under 5 minutes

Ideally well under 5 minutes

# Appendix C Charter Updates

The only update needed to our charter was editing Spec 2.1.2. Previously, the spec stated that the mulitalateration algorithm needed to be accurate to within a meter of the pinger location when the submarine was over 2 meters from the pinger. Upon prototyping and testing of the algorithm, it was discovered that due to the possible sampling rates for our system, this accuracy was not possible. However, it was noted that regardless of accuracy, the algorithm can always tell the submarine to correctly move up or down, forward or backward, and left or right. After discussing this with our project advisor and sponsor, this accuracy of the algorithm is acceptable and will allow our project to be successful. Thus, the spec was updated to: Correct cardinal direction to pinger when sub is more than 2 meters from pinger.

# Appendix D Matlab Code Used for System Testing

clc; clear all; close all;

freq = 35;

sensorArray = audioDeviceReader("Device","IN 1-4 (2- BEHRINGER UMC 404HD 192k)", ...

"Driver","DirectSound","NumChannels",4,"SampleRate",192000,"BitDepth",'16-bit integer')

% audioFromDevice=sensorArray() % gets one frame of data from device.

% samples per frame can be specified, is

% the "buffer" which can be defined and is

% the device record start time latency.

setup(sensorArray)

fileWriter = dsp.AudioFileWriter('hydrophones.wav','FileFormat','WAV',"SampleRate",192000);

disp("Record start")

tic

while toc<2.5

acquiredAudio=sensorArray();

fileWriter(acquiredAudio);

end

disp("Record end")

release(sensorArray)

release(fileWriter)

[data, Fs]=audioread('hydrophones.wav');

t = 0:1/Fs:(length(data)-1)/Fs;

% figure(1);

% for i = 1:4

% plot(t,data(:,i)); hold on;

% end

% hold off; legend({'Channel 1','Channel 2','Channel 3','Channel 4'});

sensor\_A = data(:,1);

sensor\_B = data(:,2);

sensor\_C = data(:,3);

sensor\_D = data(:,4);

fs = 192000 % Set sampling rate to variable "Fs"

figure()

subplot(4,1,1);

plot(sensor\_A); xlim([0 3\*fs]);

title("Sensor A, Time Domain (Samples)");

subplot(4,1,2);

plot(sensor\_B); xlim([0 3\*fs]);

title("Sensor B, Time Domain (Samples)");

subplot(4,1,3);

plot(sensor\_C); xlim([0 3\*fs]);

title("Sensor C, Time Domain (Samples)");

subplot(4,1,4);

plot(sensor\_D); xlim([0 3\*fs]);

title("Sensor D, Time Domain (Samples)");

tA = linspace(0,length(sensor\_A)/fs,length(sensor\_A));

tB = linspace(0,length(sensor\_B)/fs,length(sensor\_B));

tC = linspace(0,length(sensor\_C)/fs,length(sensor\_C));

tD = linspace(0,length(sensor\_D)/fs,length(sensor\_D));

figure()

subplot(4,2,1);

plot(tA,sensor\_A); xlim([0 2.75]);

title("Sensor A, Time Domain (Seconds)");

subplot(4,2,3);

plot(tB,sensor\_B); xlim([0 2.75]);

title("Sensor B, Time Domain (Seconds)");

subplot(4,2,5);

plot(tC,sensor\_C); xlim([0 2.75]);

title("Sensor C, Time Domain (Seconds)");

subplot(4,2,7);

plot(tD,sensor\_D); xlim([0 2.75]);

title("Sensor D, Time Domain (Seconds)");

F\_sensor\_A = fft(sensor\_A); % FFT on "sensor\_A"

F\_sensor\_B = fft(sensor\_B);

F\_sensor\_C = fft(sensor\_C);

F\_sensor\_D = fft(sensor\_D);

fvec\_sensor\_A = linspace(-fs/2,fs/2,length(sensor\_A)); % frequency vector

fvec\_sensor\_B = linspace(-fs/2,fs/2,length(sensor\_B));

fvec\_sensor\_C = linspace(-fs/2,fs/2,length(sensor\_C));

fvec\_sensor\_D = linspace(-fs/2,fs/2,length(sensor\_D));

subplot(4,2,2);

plot(fvec\_sensor\_A,fftshift(abs(F\_sensor\_A))); % plot "sensor\_A" in frequency domain

title("Sensor A, Frequency Domain");

subplot(4,2,4);

plot(fvec\_sensor\_B,fftshift(abs(F\_sensor\_B)));

title("Sensor B, Frequency Domain");

subplot(4,2,6);

plot(fvec\_sensor\_C,fftshift(abs(F\_sensor\_C)));

title("Sensor C, Frequency Domain");

subplot(4,2,8);

plot(fvec\_sensor\_D,fftshift(abs(F\_sensor\_D)));

title("Sensor D, Frequency Domain");

% Filter audio, Remove errant peaks, create plots.

% Filtered & cleaned audio from each sensor A is:

% "clean\_filtered\_A"

% "clean\_filtered\_B"

% "clean\_filtered\_C"

% "clean\_filtered\_D"

% Filter & Plots for sensor A

[b,a] = butter(5,[freq\*1000-750, freq\*1000+750]/(Fs/2),'bandpass');

filtered\_A = filter(b,a,sensor\_A);

FFT\_filtered\_A = fftshift(abs(fft(filtered\_A)));

clean\_filtered\_A = filtered\_A;

thresh = .005;

count = 0;

W = 3000; % Window width

for i = W+1:length(clean\_filtered\_A)-W

% for i = 175700:175740 Manually specified test window

if abs(clean\_filtered\_A(i))>thresh

count=sum(abs(clean\_filtered\_A(i-W:i+W))>thresh);

if count<0.2\*W

out2(i)=0;

end

end

end

figure();

subplot(3,1,1);

plot(t,filtered\_A);

axis([0,2.5, -1.5,1.5]);

title("A: After filtering");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(3,1,2);

plot(t,clean\_filtered\_A);

axis([0,2.5, -1.5,1.5]);

title("A: After peak removal");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(3,1,3);

plot(fvec\_sensor\_A,FFT\_filtered\_A);

title("A:frequency domain, after filtering");

ylabel("Amplitude");

xlabel("Frequency (10^4 Hz)");

% Filter & Plots for sensor B

[b,a] = butter(5,[freq\*1000-750, freq\*1000+750]/(Fs/2),'bandpass');

filtered\_B = filter(b,a,sensor\_B);

FFT\_filtered\_B = fftshift(abs(fft(filtered\_B)));

clean\_filtered\_B = filtered\_B;

thresh = .005;

count = 0;

for i = W+1:length(clean\_filtered\_B)-W

% for i = 175700:175740

if abs(clean\_filtered\_B(i))>thresh

count=sum(abs(clean\_filtered\_B(i-W:i+W))>thresh);

if count<0.2\*W

out2(i)=0;

end

end

end

figure();

subplot(3,1,1);

plot(t,filtered\_B);

axis([0,2.5, -1.5,1.5]);

title("B: After filtering");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(3,1,2);

plot(t,clean\_filtered\_B);

axis([0,2.5, -1.5,1.5]);

title("B: After peak removal");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(3,1,3);

plot(fvec\_sensor\_B,FFT\_filtered\_B);

title("B:frequency domain, after filtering");

ylabel("Amplitude");

xlabel("Frequency (10^4 Hz)");

% Filter & Plots for sensor C

[b,a] = butter(5,[freq\*1000-750, freq\*1000+750]/(Fs/2),'bandpass');

filtered\_C = filter(b,a,sensor\_C);

FFT\_filtered\_C = fftshift(abs(fft(filtered\_C)));

clean\_filtered\_C = filtered\_C;

thresh = .005;

count = 0;

for i = W+1:length(clean\_filtered\_C)-W

% for i = 175700:175740

if abs(clean\_filtered\_C(i))>thresh

count=sum(abs(clean\_filtered\_C(i-W:i+W))>thresh);

if count<0.2\*W

out2(i)=0;

end

end

end

figure();

subplot(3,1,1);

plot(t,filtered\_C);

axis([0,2.5, -1.5,1.5]);

title("C: After filtering");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(3,1,2);

plot(t,clean\_filtered\_C);

axis([0,2.5, -1.5,1.5]);

title("C: After peak removal");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(3,1,3);

plot(fvec\_sensor\_C,FFT\_filtered\_C);

title("C:frequency domain, after filtering");

ylabel("Amplitude");

xlabel("Frequency (10^4 Hz)");

% Filter & Plots for sensor D

[b,a] = butter(5,[freq\*1000-750, freq\*1000+750]/(Fs/2),'bandpass');

filtered\_D = filter(b,a,sensor\_D);

FFT\_filtered\_D = fftshift(abs(fft(filtered\_D)));

clean\_filtered\_D = filtered\_D;

thresh = .005;

count = 0;

for i = W+1:length(clean\_filtered\_B)-W

% for i = 175700:175740

if abs(clean\_filtered\_D(i))>thresh

count=sum(abs(clean\_filtered\_B(i-W:i+W))>thresh);

if count<0.2\*W

out2(i)=0;

end

end

end

figure();

subplot(3,1,1);

plot(t,filtered\_D);

axis([0,2.5, -1.5,1.5]);

title("D: After filtering");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(3,1,2);

plot(t,clean\_filtered\_D);

axis([0,2.5, -1.5,1.5]);

title("D: After peak removal");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(3,1,3);

plot(fvec\_sensor\_D,FFT\_filtered\_D);

title("D:frequency domain, after filtering");

ylabel("Amplitude");

xlabel("Frequency (10^4 Hz)");

figure() % Time domain pairs, pre and post filter

subplot(4,2,1);

plot(tA,sensor\_A); xlim([0 2.75]);

title("Sensor A, Time Domain");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(4,2,2);

plot(t,filtered\_A);

axis([0,2.5, -.5,.5]);

title("A: After filtering");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(4,2,3);

plot(tB,sensor\_B); xlim([0 2.75]);

title("Sensor B, Time Domain");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(4,2,4);

plot(t,filtered\_B);

axis([0,2.5, -.5,.5]);

title("B: After filtering");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(4,2,5);

plot(tC,sensor\_C); xlim([0 2.75]);

title("Sensor C, Time Domain");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(4,2,6);

plot(t,filtered\_C);

axis([0,2.5, -.5,.5]);

title("C: After filtering");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(4,2,7);

plot(tD,sensor\_D); xlim([0 2.75]);

title("Sensor D, Time Domain");

ylabel("Amplitude");

xlabel("Time (seconds)");

subplot(4,2,8);

plot(t,filtered\_D);

axis([0,2.5, -.5,.5]);

title("D: After filtering");

ylabel("Amplitude");

xlabel("Time (seconds)");

% TDOA algorithm that uses high frequency data from unfiltered audio:

% Sensor Pair A & B:

% We initially guess that unfiltered audio begins to show amplitude 100 us

% before the clean filtered audio.

%

% There are .192 samples per microsecond. 100 us = 521 samples.

%

% Sensor\_A (unfiltered audio) should begin to show a high frequency amplitude spike in the window of

% Sensor\_A(tempA-521:tempA+521)

%

% We now know that Sensor\_A(tempA-521:tempA+521) includes too much low

% frequency noise, and will use TDOAsensorA =

% abs(sensor\_A(tempA-TDOA\_leftedge:tempA+TDOA\_rightedge)).

tempA = find(clean\_filtered\_A >.001, 1); % Find the array index of clean\_filtered\_A

% where amplitude first begins to climb.

TDOA\_leftedge = 210;

TDOA\_rightedge = 521;

TDOAsensorA = abs(sensor\_A(tempA-TDOA\_leftedge:tempA+TDOA\_rightedge));

% Repeat for sensor\_B

TDOAsensorB = abs(sensor\_B(tempA-TDOA\_leftedge:tempA+TDOA\_rightedge));

% Plot TDOAsensorA & TDOAsensorB together:

t\_TDOApair = linspace(0,length(TDOAsensorA)/192000,length(TDOAsensorA)); % time vector for new shortened arrays

figure()

plot(t\_TDOApair,TDOAsensorA,'r'); hold on; plot(t\_TDOApair,TDOAsensorB,'b'); hold off; title('A (red) and B (blue)');

% Can use plot to define an amplitude threshold and find the indices and time values

% when both A & B cross that threshold.

DiffThreshA = .15\*(max(TDOAsensorA));

DiffThreshB = .15\*(max(TDOAsensorB));

TDOAindex\_A = find(TDOAsensorA>DiffThreshA,1);

TDOAindex\_B = find(TDOAsensorB>DiffThreshB,1);

TDOA\_time\_A = t\_TDOApair(TDOAindex\_A);

TDOA\_time\_B = t\_TDOApair(TDOAindex\_B);

TDOA\_time\_AB = TDOA\_time\_A-TDOA\_time\_B;

tau\_AB = round((TDOA\_time\_A-TDOA\_time\_B)\*fs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sensor pair A & C:

% Repeat for sensor\_C

TDOAsensorC = abs(sensor\_C(tempA-TDOA\_leftedge:tempA+TDOA\_rightedge));

% Plot TDOAsensorA & TDOAsensorC together:

figure()

plot(t\_TDOApair,TDOAsensorA,'r'); hold on; plot(t\_TDOApair,TDOAsensorC,'b'); hold off; title('A (red) and C (blue)');

% Can use plot to define an amplitude threshold and find the indices and time values

% when both A & B cross that threshold.

DiffThreshA = .15\*(max(TDOAsensorA));

DiffThreshC = .15\*(max(TDOAsensorC));

TDOAindex\_A = find(TDOAsensorA>DiffThreshA,1);

TDOAindex\_C = find(TDOAsensorC>DiffThreshC,1);

TDOA\_time\_A = t\_TDOApair(TDOAindex\_A);

TDOA\_time\_C = t\_TDOApair(TDOAindex\_C);

TDOA\_time\_AC = TDOA\_time\_A-TDOA\_time\_C;

tau\_AC = round((TDOA\_time\_A-TDOA\_time\_C)\*fs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Sensor pair A & D

% Repeat for sensor\_D

TDOAsensorD = abs(sensor\_D(tempA-TDOA\_leftedge:tempA+TDOA\_rightedge));

% Plot TDOAsensorA & TDOAsensorD together:

figure()

plot(t\_TDOApair,TDOAsensorA,'r'); hold on; plot(t\_TDOApair,TDOAsensorD,'b'); hold off; title('A (red) and D (blue)');

% Can use plot to define an amplitude threshold and find the indices and time values

% when both A & B cross that threshold.-

DiffThreshA = .15\*(max(TDOAsensorA));

DiffThreshD = .15\*(max(TDOAsensorD));

TDOAindex\_A = find(TDOAsensorA>DiffThreshA,1);

TDOAindex\_D = find(TDOAsensorD>DiffThreshD,1);

TDOA\_time\_A = t\_TDOApair(TDOAindex\_A);

TDOA\_time\_D = t\_TDOApair(TDOAindex\_D);

TDOA\_time\_AD = TDOA\_time\_A-TDOA\_time\_D;

tau\_AD = round((TDOA\_time\_A-TDOA\_time\_D)\*fs);

% End unfiltered audio time-lag component

% Begin fine tuning TDOA values with Cross Correlation

sensor\_A = data(:,1);

sensor\_B = data(:,2);

sensor\_C = data(:,3);

sensor\_D = data(:,4);

w=clean\_filtered\_A;

x=clean\_filtered\_B;

y=clean\_filtered\_C;

z=clean\_filtered\_D;

lA = length(w);

lB = length(x);

lC = length(y);

lD = length(z);

samples1 = 1:min(lA,lB);

samples2 = 1:min(lA,lC);

samples3 = 1:min(lA,lD);

[C1, lag1] = xcorr(w(samples1), x(samples1), 155);

[C2, lag2] = xcorr(w(samples2), y(samples2), 155);

[C3, lag3] = xcorr(w(samples3), z(samples3), 135);

%normalizing XCorr

C1 = C1/max(C1);

C2 = C2/max(C2);

C3 = C3/max(C3);

WINDOW = ceil(fs/(2\*1000\*freq)\*1.5);

n1 = ceil(length(C1)/2);

n2 = ceil(length(C2)/2);

n3 = ceil(length(C3)/2);

C1sub2 = C1((tau\_AB + n1 - WINDOW):(tau\_AB + n1 + WINDOW));

idx\_AB2 = find(C1sub2==max(C1sub2))+tau\_AB+n1-WINDOW;

C2sub2 = C2((tau\_AC + n2 - WINDOW):(tau\_AC + n2 + WINDOW));

idx\_AC2 = find(C2sub2==max(C2sub2))+tau\_AC+n2-WINDOW;

C3sub2 = C3((tau\_AD + n3 - WINDOW):(tau\_AD + n3 + WINDOW));

idx\_AD2 = find(C3sub2==max(C3sub2))+tau\_AD+n3-WINDOW;

tau\_AB2 = lag1(idx\_AB2);

tau\_AC2 = lag1(idx\_AC2);

tau\_AD2 = lag1(idx\_AD2);

C1sub = C1((tau\_AB2 + n1 - WINDOW):(tau\_AB2 + n1 + WINDOW));

idx\_AB = find(C1sub==max(C1sub))+tau\_AB2+n1-WINDOW;

C2sub = C2((tau\_AC2 + n2 - WINDOW):(tau\_AC2 + n2 + WINDOW));

idx\_AC = find(C2sub==max(C2sub))+tau\_AC2+n2-WINDOW;

C3sub = C3((tau\_AD2 + n3 - WINDOW):(tau\_AD2 + n3 + WINDOW));

idx\_AD = find(C3sub==max(C3sub))+tau\_AD2+n3-WINDOW;

adjusted\_tdoa\_seconds\_A\_B = lag1(idx\_AB)/fs

adjusted\_tdoa\_seconds\_A\_C = lag1(idx\_AC)/fs

adjusted\_tdoa\_seconds\_A\_D = lag1(idx\_AD)/fs

% Multilateration now needs to take the above three

% adjusted\_tdoa\_seconds\_A\_X" variables as inputs

%% Speed of sound in water

c = 1500; % m/s

%% Location of the hydrophones

% Sensor A is at the origin. Do not change this.

xA = 0;

yA = 0;

zA = 0;

xB = -.838;

yB = -.813;

zB = .254;

xC = -.838;

yC = -.406;

zC = .813;

xD = -.838;

yD = -.406;

zD = .419;

% Solve Equations

Eq1: tauB = (1/c) \* ( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) )

Eq2: tauC = (1/c) \* ( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) )

Eq3: tauD = (1/c) \* ( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) )

Unknown 1: x

Unknown 2: y

Unknown 3: z

syms x y z

eq1 = -round(adjusted\_tdoa\_seconds\_A\_B, 5) == (1/c) \* ( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq2 = -round(adjusted\_tdoa\_seconds\_A\_C, 5) == (1/c) \* ( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq3 = -round(adjusted\_tdoa\_seconds\_A\_D, 5) == (1/c) \* ( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) );

sol = solve([eq1, eq2, eq3], [x, y, z]);

xSol = real(double(sol.x));

ySol = real(double(sol.y));

zSol = real(double(sol.z));

if length(xSol) == 1

solutions = [xSol,ySol,zSol,xSol,ySol,zSol]

recalculate = 0;

elseif isempty(xSol)

solutions = [0,0,0,0,0,0]; %% Solution using xCorr-adjusted TDOA

recalculate = 1;

else

solutions = [xSol(1) ySol(1) zSol(1) xSol(2) ySol(2) zSol(2)]

recalculate = 1;

end

syms x y z

eq1 = -round(TDOA\_time\_AB, 5) == (1/c) \* ( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq2 = -round(TDOA\_time\_AC, 5) == (1/c) \* ( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq3 = -round(TDOA\_time\_AD, 5) == (1/c) \* ( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) );

sol = solve([eq1, eq2, eq3], [x, y, z]);

xSol = double(sol.x);

ySol = double(sol.y);

zSol = double(sol.z);

if length(xSol) == 1

solutions2 = [xSol,ySol,zSol,xSol,ySol,zSol]

recalculate = 0;

elseif isempty(xSol)

solutions2 = [0,0,0,0,0,0]; %% Solution using orignal TDOA, no XCorr

recalculate = 1;

else

solutions2 = [xSol(1) ySol(1) zSol(1) xSol(2) ySol(2) zSol(2)]

recalculate = 1;

end

# Appendix E MatLab Filter Test Code

**Filter reliability**

The following code was used to test the reliability of the digital bandpass filter. The code runs 1000 trials, and for each trial it generates a signal which emulates the competition environment. The filter is applied at the frequency of the first in the list of randomly generated “beacon chirps”. The code runs 1000 trials and hangs if there is an error. Use command hist(trials) to see a histogram of input ratios for the trial set.

clc, clear all, close all;

N\_Trials = 1000;

trial = zeros(1,N\_Trials);

trial\_ratio = zeros(1,N\_Trials);

for i = 1:N\_Trials

% generate random integers between 25 and 50

int = randi([25 48],6,1);

if length(unique(int)) < 6

int = randi([25,48],6,1);

end

% frequencies

f = [int(1)\*(10^3) int(2)\*10^3 int(3)\*10^3 int(4)\*10^3 int(5)\*10^3 int(6)\*10^3];

% random amplitudes

amp = rand(6,1);

trial\_ratio(i) = amp(1)/max(amp);

% random phase

phi = 2\*pi\*rand(6,1);

%chirps

Fs = 100000;

chirp\_duration = 0.25; % duration of chirp in seconds (<0.5)

t\_chirp = 0:1/Fs:chirp\_duration;

chirp1 = amp(1)\*sin(2\*pi\*f(1)\*t\_chirp+phi(1)) + 0\*randn(size(t\_chirp));

chirp2 = amp(2)\*sin(2\*pi\*f(2)\*t\_chirp+phi(2));

chirp3 = amp(3)\*sin(2\*pi\*f(3)\*t\_chirp+phi(3));

chirp4 = amp(4)\*sin(2\*pi\*f(4)\*t\_chirp+phi(4));

chirp5 = amp(5)\*sin(2\*pi\*f(5)\*t\_chirp+phi(5));

chirp6 = amp(6)\*sin(2\*pi\*f(6)\*t\_chirp+phi(6));

t\_full = 0:1/Fs:2.5;

V = zeros(size(t\_full));

% Read in explosion soundbyte

bomb = audioread("Bomb.mp3");

bomb2 = zeros(29239,2);

bomb2(1:length(bomb2)-1) = bomb(3940:33177);

bomb3 = zeros(1,29239);

bomb3 = bomb2(:,1);

bomb3 = bomb3';

start(1) = ceil(Fs\*2\*rand()); % Generate random start time in units of samples

start(2) = ceil(Fs\*2\*rand());

start(3) = ceil(Fs\*2\*rand());

start(4) = ceil(Fs\*2\*rand());

start(5) = ceil(Fs\*2\*rand());

start(6) = ceil(Fs\*2\*rand());

start(7) = ceil(((Fs\*2)-length(bomb2))\*rand());

V(start(1):start(1)+length(chirp1)-1) = V(start(1):start(1)+length(chirp1)-1)+chirp1;

V(start(2):start(2)+length(chirp2)-1) = V(start(2):start(2)+length(chirp2)-1)+chirp2;

V(start(3):start(3)+length(chirp3)-1) = V(start(3):start(3)+length(chirp3)-1)+chirp3;

V(start(4):start(4)+length(chirp4)-1) = V(start(4):start(4)+length(chirp4)-1)+chirp4;

V(start(5):start(5)+length(chirp5)-1) = V(start(5):start(5)+length(chirp5)-1)+chirp5;

V(start(6):start(6)+length(chirp6)-1) = V(start(6):start(6)+length(chirp6)-1)+chirp6;

V(start(7):start(7)+length(bomb3)-1) = V(start(7):start(7)+length(bomb2)-1)+bomb3;

% plot in time domain

t = linspace(0,2.5,length(V));

% figure(1)

% disp("Frequencies used: ")

% disp(int')

% subplot(2,2,1);

% ylim([-1.5,1.5]);

% plot(t,V);

% title("Generated signal in the time domain, before filtering");

% ylabel("Amplitude");

% xlabel("Time (seconds)");

% hold on

%plot in frequency domain

Fv = fft(V);

fvec = linspace(-Fs/2,Fs/2,length(V));

% subplot(2,2,2);

% plot(fvec,fftshift(abs(Fv)));

% title("Generated signal in the frequency domain, before filtering");

% ylabel("Amplitude");

% xlabel("Frequency (10^4 Hz)");

[b,a] = butter(5,[int(1)\*1000-500, int(1)\*1000+500]/(Fs/2),'bandpass');

out = filter(b,a,V);

FFT\_Out = fftshift(abs(fft(out)));

% subplot(2,2,3);

% plot(t,out);

% ylim([-1.5,1.5]);

% title("Generated signal in the time domain, after filtering");

% ylabel("Amplitude");

% xlabel("Time (seconds)");

% subplot(2,2,4);

% plot(fvec,FFT\_Out);

% title("Generated signal in the frequency domain, after filtering");

% ylabel("Amplitude");

% xlabel("Frequency (10^4 Hz)");

hold off

idx = find(FFT\_Out == max(FFT\_Out));

idx = idx(1);

answer = abs(fvec(idx));

[f(1); answer];

if (abs(f(1)-answer < 250))

trial(i) = 1;

print('Success')

end

end

print('done')

Filter reliability in differentiating between 1 kHz gaps

The following code was used to test the reliability of the digital bandpass filter to differentiate between 1 kHz gaps. The code randomly selects a group of three neighboring integer 1 kHz frequencies, and filters for the one in the center. The code runs 1000 trials and hangs if there is an error. Use command hist(trials) to see a histogram of input ratios for the trial set.

clc, clear all, close all;

N\_Trials = 1000;

trial = zeros(1,N\_Trials);

trial\_ratio = zeros(1,N\_Trials);

for i = 1:N\_Trials

% generate random integer between 25 and 45

lowerBound = randi([25 45],1,1);

% frequencies

f = [lowerBound\*(10^3), (lowerBound+1)\*(10^3), (lowerBound+2)\*(10^3)];

% random amplitudes

amp = rand(3,1);

trial\_ratio(i) = amp(1)/max(amp);

% random phase

phi = 2\*pi\*rand(3,1);

%chirps

Fs = 192000;

chirp\_duration = 0.25; % duration of chirp in seconds (<0.5)

t\_chirp = 0:1/Fs:chirp\_duration;

chirp1 = amp(1)\*sin(2\*pi\*f(1)\*t\_chirp+phi(1)) + 0\*randn(size(t\_chirp));

chirp2 = amp(2)\*sin(2\*pi\*f(2)\*t\_chirp+phi(2));

chirp3 = amp(3)\*sin(2\*pi\*f(3)\*t\_chirp+phi(3));

t\_full = 0:1/Fs:2.5;

V = zeros(size(t\_full));

% Read in explosion soundbyte

bomb = audioread("Bomb.mp3");

bomb2 = zeros(29239,2);

bomb2(1:length(bomb2)-1) = bomb(3940:33177);

bomb3 = zeros(1,29239);

bomb3 = bomb2(:,1);

bomb3 = bomb3';

start(1) = ceil(Fs\*2\*rand()); % Generate random start time in units of samples

start(2) = ceil(Fs\*2\*rand());

start(3) = ceil(Fs\*2\*rand());

start(4) = ceil(((Fs\*2)-length(bomb2))\*rand());

V(start(1):start(1)+length(chirp1)-1) = V(start(1):start(1)+length(chirp1)-1)+chirp1;

V(start(2):start(2)+length(chirp2)-1) = V(start(2):start(2)+length(chirp2)-1)+chirp2;

V(start(3):start(3)+length(chirp3)-1) = V(start(3):start(3)+length(chirp3)-1)+chirp3;

V(start(4):start(4)+length(bomb3)-1) = V(start(4):start(4)+length(bomb2)-1)+bomb3;

% plot in time domain

t = linspace(0,2.5,length(V));

% figure(1)

% disp("Frequencies used: ")

% disp(int')

% subplot(2,2,1);

% ylim([-1.5,1.5]);

% plot(t,V);

% title("Generated signal in the time domain, before filtering");

% ylabel("Amplitude");

% xlabel("Time (seconds)");

% hold on

%plot in frequency domain

Fv = fft(V);

fvec = linspace(-Fs/2,Fs/2,length(V));

% subplot(2,2,2);

% plot(fvec,fftshift(abs(Fv)));

% title("Generated signal in the frequency domain, before filtering");

% ylabel("Amplitude");

% xlabel("Frequency (10^4 Hz)");

[b,a] = butter(5,[f(2)-500, f(2)+500]/(Fs/2),'bandpass');

out = filter(b,a,V);

FFT\_Out = fftshift(abs(fft(out)));

% subplot(2,2,3);

% plot(t,out);

% ylim([-1.5,1.5]);

% title("Generated signal in the time domain, after filtering");

% ylabel("Amplitude");

% xlabel("Time (seconds)");

% subplot(2,2,4);

% plot(fvec,FFT\_Out);

% title("Generated signal in the frequency domain, after filtering");

% ylabel("Amplitude");

% xlabel("Frequency (10^4 Hz)");

hold off

idx = find(FFT\_Out == max(FFT\_Out));

idx = idx(1);

answer = abs(fvec(idx));

[f(1); answer];

if (abs(f(1)-answer < 250))

trial(i) = 1;

print('Success')

end

end

print('done')

# Appendix F MatLab Multilateration Test Code

function solutions = Multilateration(Time)

%% Speed of sound in water

c = 1500; % m/s

%% Location of the hydrophones

% Sensor A is at the origin. Do not change this.

xA = 0;

yA = 0;

zA = 0;

xB = -1.8;

yB = -.9;

zB = -.4;

xC = -1.8;

yC = -.4;

zC = -.9;

xD = -1.8;

yD = -.9;

zD = -.9;

%

% xA = 0;

% yA = 0;

% zA = 0;

%

% xB = -1;

% yB = -1;

% zB = -2;

%

% xC = 2;

% yC = 1;

% zC = 1;

%

% xD = -1;

% yD = 1;

% zD = 1;

% xA = 0;

% yA = 0;

% zA = 0;

%

% xB = -0.2;

% yB = 1.0;

% zB = 0.5;`

%

% xC = -0.5;

% yC = -1.0;

% zC = -0.1;

%

% xD = -1;

% yD = 0;

% zD = 0;

h1 = [xA; yA; zA];

h2 = [xB; yB; zB];

h3 = [xC;yC;zC];

h4 = [xD;yD;zD];

%ping = [22;12;-5];

%ping = [11;6;-3];

% ping = [-1; -.5;-3]

%ping = [-4; 10; -2];

%ping = [2;5;-4];

% ping = [-2.02; 2.01; -1.38]

%ping = [-10;-10;-10];

% ping = [randi([0,22],1,1); randi([0,12],1,1); -randi([0,5],1,1)]

ping = [-9;-8;-8]

d1 = norm(h1 - ping)/c;

d2 = norm(h2 - ping)/c;

d3 = norm(h3 - ping)/c;

d4 = norm(h4 - ping)/c;

d21 = double(d2-d1)

d31 = double(d3-d1)

d41 = double(d4-d1)

%% Solve Equations

%

% Eq1: tauB = (1/c) \* ( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) )

% Eq2: tauC = (1/c) \* ( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) )

% Eq3: tauD = (1/c) \* ( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) )

%

% Unknown 1: x

% Unknown 2: y

% Unknown 3: z

%

syms x y z

eq1 = d21 == (1/c)\*( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq2 = d31 == (1/c)\*( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq3 = d41 == (1/c)\*( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq1 = -.00122 == (1/c)\*( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq2 = -.00121 == (1/c)\*( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) );

eq3 = -.00141 == (1/c)\*( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq1 = .00069 == (1/c)\*( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq2 = -.0013 == (1/c)\*( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq3 = .00043 == (1/c)\*( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq1 = .0006 == (1/c)\*( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq2 = .0001== (1/c)\*( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq3 = .0014 == (1/c)\*( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq1 = .0015 == (1/c)\*( sqrt( (x-xB)^2 + (y-yB)^2 + (z-zB)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq2 = .0015 == (1/c)\*( sqrt( (x-xC)^2 + (y-yC)^2 + (z-zC)^2 ) - sqrt(x^2 + y^2 + z^2) );

% eq3 = .0017 == (1/c)\*( sqrt( (x-xD)^2 + (y-yD)^2 + (z-zD)^2 ) - sqrt(x^2 + y^2 + z^2) );

sol = solve([eq1, eq2, eq3], [x, y, z]);

xSol = double(sol.x);

ySol = double(sol.y);

zSol = double(sol.z);

% norm([2.3126;6.0427;-4.2670] - ping)

if length(xSol) == 1

solutions = [xSol,ySol,zSol,xSol,ySol,zSol];

else

solutions = [xSol(1) ySol(1) zSol(1) xSol(2) ySol(2) zSol(2)];

end

error1 = norm([solutions(1);solutions(2);solutions(3)] - ping)

error2 = norm([solutions(4);solutions(5);solutions(6)] - ping)

# Appendix G Python Wrapper Function

import matlab.engine

import time

eng = matlab.engine.start\_matlab()

# Call to pinger location algorithm

# Input is required pinger frequency in kHz

location = eng.test(35)

print(location)