ECE-499: Electrical and Computer Engineering Capstone Design Project III

# The AFμS (Autonomous Flocking μ-Sub) Project

Technical Design Report

Written by Jacob Karaul and Xavier Quinn

Advisors: Walter Dixon and Professor Hedrick

March 26, 2020

# *Report Summary*

Underwater data collection is a field that concerns the tracking, monitoring, or gathering of underwater data and has significant financial, social, and environmental implications. Technologies used to attain this data are vast, but suffer from key shortcomings that prevent their adoption at mass in society. Autonomous Underwater Vehicles (AUV's) provide a cheap and autonomous option, however current industry products suffer either from a high price point or lack in ease of use (i.e. meters long and several tons heavy). Recently, AUV's have been emerging on the micro-scale (a few feet in length), easily operable by one person.

This project is the Autonomous Flocking μ-Sub project, or AFμS project. The goal of this project is to develop a micro-scale AUV at a cost of no more than $500, thus both handlable and affordable by a much larger audience. Further specifications include a communication system with a BER less than 10%, data rate of at least 62.5 bps, and transmission distance of at least 1 meter, as well as successful obstacle detection at 1.5 meters away, and finally position holding in a 5 meter cube for at least 5 minutes. Flocking capabilities will increase the applicability of the product, and is achievable by utilizing a low latency optical communication system. Sonar is used for obstacle avoidance, and a combination of inertial measurement and global positioning allows for navigation and exploration. Results of this project show that decent performance in communication, locomotion and obstacle avoidance can still be achieved with off-the-shelf electrical components, however dead-reckoning using an off-the-shelf inertial measurement unit is not feasible due to unbounded error accumulation.

# *Table of Contents*

# *List of Tables and Figures*

## *Tables*



## *Figures*



[1] J. Sticklus, P. A. Hoeher and R. Röttgers, "Optical Underwater Communication: The Potential of Using Converted Green LEDs in Coastal Waters," in IEEE Journal of Oceanic Engineering, vol. 44, no. 2, pp. 535-547, April 2019.
[2] Ainslie M. A., McColm J. G., "A simplified formula for viscous and chemical absorption in sea water", Journal of the Acoustical Society of America, **103**(3), 1671-1672, 1998.

# Introduction

Underwater exploration is a field of study that involves investigating chemical and physical characteristics of large water bodies (typically oceans) and the organisms that reside in them. In total, humanity has explored about 8% of the world's water bodies, and only 5% of ocean basins[3]. Some areas of study in this field are oil spills, habitation patterns for aquatic species, residual radiation and other contaminants, turbidity levels, and general 3D mapping, all of which have significant health or monetary implications. A need for accurate underwater data can be found in the Great Lakes, a source of 21% of the world's freshwater supply[4], which suffers from harmful algal blooms (HAB's) in various locations that render the water undrinkable. More locally, HAB's are also present in Moreau Lake State Park in Saratoga County at such a debilitating quantity that several pets have died and parts of the state park have been shut down[5]. Underwater data collection would be useful in these instances to detect oxygen and toxicity levels where HAB's are present in order to characterize and document them.

Systems that can generate this data are vast; there are large research vessels outfitted with research teams and sensing equipment, stationary underwater sensor networks, and Remotely Operated Vehicles, or ROV's, that are controlled on the surface (just to name a few). All of these systems are limited by the fact that they require active governance by humans, therefore making these technologies for underwater data collection quite slow. This limitation spurred the emergence of Autonomous Underwater Vehicles, or AUV's, which are capable of performing

[3] https://oceanexplorer.noaa.gov/backmatter/whatisexploration.html
[4] https://coast.noaa.gov/states/fast-facts/great-lakes.html
[5] https://www.timesunion.com/7dayarchive/article/Moreau-State-Park-reports-algae-bloom-14341929.php

some decision-making during their journeys. These vehicles can be “dropped” into a water body with mission parameters and, depending on the system, may run for days or weeks at a time with no human contact.

The goal for our project is to develop an autonomous underwater vehicle with reasonable accuracy at a much lower price point than is currently possible by leveraging recent emergences in low-cost Single Board Computers (SBC’s), Inertial Measurement Units (IMU’s), and high performance motors/motor drivers to do so. A single system will be affordable, but will still be accurate enough to be useful for a range of uses. This team also includes two mechanical engineers: Alexander Pradhan and Samuel Veith, who will be designing the hull, thrusters, and renewable charging solution for their senior capstone project. They will also construct and maintain all test systems. Their specific contributions are mentioned below when relevant.

Even with a team of four students, this project is a large endeavor to say the least. Two of the most limited resources are time and money, so it is imperative that both be spent on worthy causes. Therefore, we will be purchasing off-the-shelf components whenever possible and focusing our efforts on implementing systems that require finer control and precision, and are large blocks in the scope of this project. Any non-mandatory systems that take too much time/money to implement will be either stashed away for future work or eliminated entirely. Finally, a clear schedule will provide deadlines from both computer/electrical and mechanical engineering teams to ensure that work continues in parallel. See *Implementation Schedule* for specifics.

Furthermore, with these constrictions, it is not feasible to design and build an AUV that can boast better performance than any of the systems already in existence. The *Design*

*Requirements* section details the specifications that we believe can be accomplished, and are still respectable enough for this product to be useful.

This paper details the background of the field, design requirements and alternatives, overall design and implementation, test results, schedule, and a discussion for the Autonomous Flocking μ-Sub project.

# Background

Though the development of AUVs was not possible until computation units were sufficiently advanced for real-time autonomous navigation, a wide range of AUVs have come out since this point. Based on the necessary size of these computation units since their capabilities reached those necessary for autonomy, the initial generations were extremely large scale. Only in recent years have some AUVs shrunk below the size of human carrying submersibles. Currently, there exists a large variation in the size of available aquatic autonomous vessels based on their desired functionality. The need for highly precise sensor data, locomotion, or long term deployment all result in a bulkier product, and there has even been an increased interest in highly use specific[6] AUV designs with their own physical requirements[7].

As developments in technology have allowed them to do so, an increase in micro-scale AUVs has occurred. These systems are roughly defined as being in the ~1 meter or less range, and inevitably have less precise sensors and reduced capabilities from their full scale counterparts. Examples of such AUVs are the Hydroid REMUS M3V[8], the Hydromea Exray[9]

[6] https://www.hakaimagazine.com/news/rangerbot-programmed-to-kill/
[7] https://spectrum.ieee.org/robotics/humanoids/meet-aquanaut-the-underwater-transformer
[8] https://www.hydroid.com/REMUS-M3V
[9] https://www.hydromea.com/exray-wireless-underwater-drone/

and the Riptide μUUV[10]. All three of these are small and light enough to be hand deployed and retrieved, making them drastically more useful to anyone who does not have access to hoist equipment. All of the companies that manufacture AUVs have attempted to maximize the precision and capability of their products; all manufacturers also maintain a price point high enough to limit who can reasonably afford to use their services.

The majority of the microscale AUVs on the market have been designed for very specific uses, and therefore contain a sensor suite perfectly suited to said task. An example of this is the Hydromea Exray, which is designed with the specific task of exploring and measuring the wall thickness of flooded confined spaces, such as shipping vessel ballast tanks. This means that this particular AUV does not have much use outside of this task, regardless of how well suited the rest of the system is for another application. The Riptide μUUV has the option for a sensor payload bay to be attached in the middle of the AUV that allows for water to flow through it. This is to allow the user to design and build their own completely seperate sensor collection system and place it inside the bay to record data.

Although humanity has explored only a tiny percentage of the world's ocean bodies, we are dependent on oceans for underwater chemical/physical processes such as photosynthesis, safe transportation of cargo ships that carry billions of dollars in merchandise, stable and regenerable food sources, renewable generation of electricity, and countless more examples. The economic effects of cheap-to-collect, widespread submersibles with comprehensive interfaces can allow for not only researchers, but civilians to gather oceanic data in an automated fashion. Combined with open repositories for users to post and view this data, putting our world's oceans on the map can

[10] https://riptideas.com/micro-uuv/

be accomplished in the scale of years as opposed to decades. This will increase financial gains in a variety of industries; for example, cargo ships can identify routes with minimal interference from currents and save fuel, HAB's can be identified as soon as they emerge, thus preventing costly cleanup methods and loss in business (if located in a public park), and more.

As of now there is no AUV that fills the niche of a highly affordable system, with or without the drawbacks expected from less expensive equipment. As not all aquatic data collection needs high precision location or external sensor information, a system affordable enough for individual or university level research would greatly increase the amount of aquatic based monitoring and analysis, which is needed with today's climate conditions now more than ever.

With an exponentially growing population, the amount of waste humans will produce will undoubtedly grow as well; it is therefore vital that future engineering design take environmental sustainability as a demand. This product will be useful in a large number of ongoing sustainability efforts, such as pollution management, biological hotspot detection, and nutrient monitoring. Due to its place in the "sustainability" market, this product will be applicable in society and to individuals for the foreseeable future, especially due to its low price point.

While it may be desirable to make an AUV system as cheap as possible, there are some aspects which cannot be compromised when it comes to leaving something in a water source for extended periods of time. No corners can be cut when it comes to avoiding environmental impact, whether it be from materials leaching chemicals, the equipment interfering with local

wildlife, or the entire system breaking under expected conditions and littering its parts into the water.

Furthermore, there is also a large social implication of widespread, autonomous submersibles. This distinguishes our submarine from those on the market, and the impact they have due to a limited customer base; users will not only include research teams and local governments, but individuals as well. This will increase public education about our world's oceans and water bodies, as well as their limited resources, and promote a greater societal attitude towards environmental consciousness. Widespread submersibles in society also create another mechanism: active monitoring. A submarine (or flock of submarines) can operate autonomously, thus many applications may include constant monitoring of a water body. Any deviations from reasonable water conditions will be discovered quickly, such as a HAB, and can be dealt with before the situation becomes untenable. In situations where public health is at risk, such as increased radiation levels, this active monitoring may mean the difference between a small cleanup and a pandemic. Increased submarines available at the consumer and education level will allow for many issues, often dangerous, to be dealt with quickly and lead to a more proactive society.

# Design Requirements

This system is being designed with the aim of its use within a flock. While aspects of its functionality are chosen specifically to allow for this behavior, the specifications in the scope of this design report represent our expectations of not having a final product with full flocking capabilities within the timeline of this project. To better visualize, a block diagram of

functionalities can be found below, where high level functionalities are at the top (in blue), and low level functionalities are at the bottom (in black).

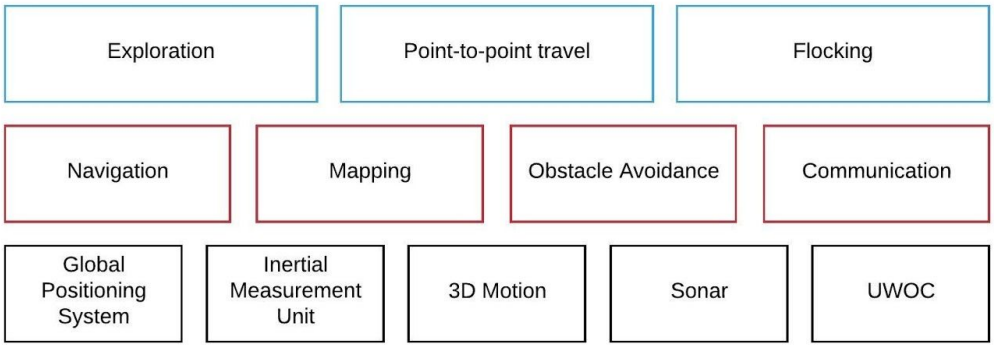



*Fig. 1. Functionality tree*

High level functionalities are marketable, and in this project they include exploration, point-to-point travel, and flocking. These functionalities depend on several low layer functionalities working together; for example, point-to-point travel requires GPS and IMU peripherals, and 3D motion. Due to time and budgetary constraints, several of these high level functionalities may not be tested or fully implemented by the end of this project (i.e. flocking would require many identical subs be manufactured, which is not financially feasible). Therefore, the goal of this project is to demonstrate a functional proof-of-concept for a system that would be able to accomplish these high level functionalities, and for development, low-level functionalities are implemented that these behaviors depend on.

In an attempt to ensure that our project is applicable to the sector it is being designed for, we held an ideation session at the start of the design process. This involved collecting and collating use cases for an AUV, then sorting them as a function of importance to the consumer and how widespread the specific use case is. These results were then analyzed, condensed, and

repeatedly sorted through until a set of design parameters were generated with the goal of maximizing usefulness to the prospective customer. Further information on this process is available in Appendix A. The design requirements detailed in this report only cover the aspects that the computer and electrical team are directly involved in the design of.

The most important requirement of this system is that any environmental impact is mitigated. This means that the AUV must exhibit robustness and durability to avoid breakage and therefore contamination of the local environment. Most of the requirements to fulfill this need falls on the side of the mechanical engineering team, though ensuring that the AUV has the capability to keep itself out of dangerous situations is an interdepartmental effort. To be able to do this, the foremost requirement is that the AUV is able to translate itself in 3D space, which is a boolean requirement; it either can or cannot. A highly related requirement is the ability to translocate to a designated set of coordinates. The selected range for this to be considered successful is within 5 meters of the point. While this may seem like a wide region of error, it must be considered that this device will be acting on the scale of lakes, oceans and rivers. If the device can translocate to a point, it must also be able to hold its position within said range of that point for at least 5 minutes.

Another important factor for avoiding environmental impact is avoiding collisions between the AVU and other objects, not only to protect the AUV, but to protect whatever it has the potential to collide with as well. To avoid objects, the AUV must be able to detect them at a minimum range of 1.5m. As the AUV may not be the object that is moving in this situation, it must be able to avoid an oncoming collision at a relative velocity of 1m/s from the time the AUV was detected. To be able to react quickly in situations like these, the AUV must be able to

accelerate itself to its maximal velocity at at least 1 $m/s^2$ . For it to be able to handle and remove itself from currents, the AUV will need to be able to travel at least at 3*m/s.* If an obstacle is detected and avoided during point to point travel, the AUV must continue to the designated point.

For the AUV to be able to provide useful sensor data, it must be able to traverse a designated area, staying within the region, and passing over all parts of the area evenly e.g. not simply containing its exploration to a corner of the zone.

The chosen method of communication is wireless optical transmission/reception. Pertinent specifications for any communication system include Bit Error Rate (BER), minimum transmission distance, and data rate. Additionally, turbidity is included as optical communication capabilities can largely vary with clarity. These factors are all closely related, thus the requirement will include them all in a single scenario: At a turbidity of 10 NTU, the AUV must be able to communicate optically at a distance of 0.5 meters at a data rate of 62.5 bps with a BER of 10% or less.

A value of 10 NTU was found to the average during low-flow periods in rivers and lakes, where this sub will be tested experimentally[11]. Both BER and distance were estimated from a set of tests run on an underwater optical communication system in various turbidity conditions[12]. Finally, the data rate was calculated from early tests run with optical components, described in the *Preliminary Testing Results* section, and a communication protocol. While the

[11] https://www.fondriest.com/environmental-measurements/parameters/water-quality/turbidity-total-suspended-solids-water-clarity/

[12] M. E. G. Mital *et al*., "Characterization of underwater optical data transmission parameters under varying conditions of turbidity and water movement," *2017 5th International Conference on Information and Communication Technology (ICoIC7)*, Malacca City, 2017, pp. 1-6.

communication modulation technique is not yet set in stone, a clocking pulse-position modulation (PPM) scheme is being seriously considered and is used for this calculation.

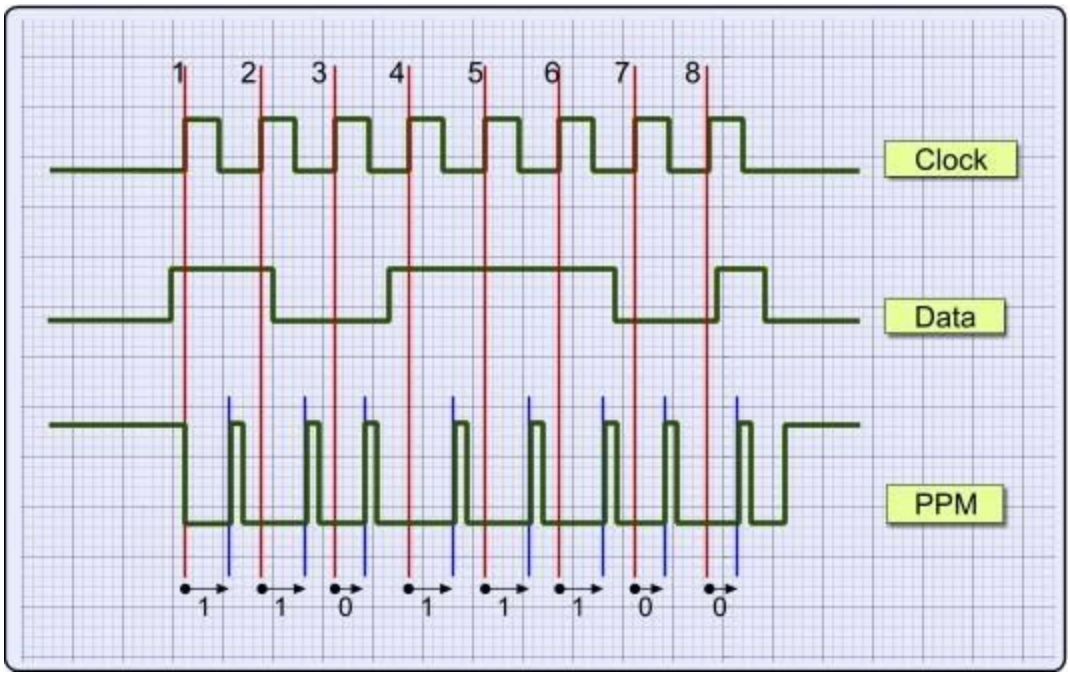



*Fig. 2. Timing diagram of the PPM protocol*

PPM is an encoding technique that modulates the time since the last clock pulse before transmitting a high pulse as a means of communicating data. For example, in a 15 ms period, a delay since the last clock pulse of 5 ms may indicate a ‘0’ and a delay of 10 ms may indicate a ‘1’. This method allows for the detection of errors in a noisy channel by utilizing a fixed set of known locations a pulse must be at any given clock cycle. If pulses are not in the correct position within a clock cycle or are not the correct pulse width, then an error is detected. Except for the unique case in which an error both erases the original pulse and creates a pulse in another allowable pulse location, errors should be easily detectable.

*Fig. 3. Illustration of PPM clocking*

This protocol, however, does not provide a method for synchronization. That is, the clock at the receiving end may be out of phase, faster or slower than the clock at the transmitting end and thus the actual time since the last clock pulse is compromised. To account for this, clocking can be implemented by sending a pulse at the beginning of every clock cycle, as illustrated above. Blue pulses represent data, green dashed lines represent the start of clock cycles, and red pulses represent the pulse at the beginning of every clock cycle. The receiving end can use this periodic clocking pulse to establish the beginning and end of every cycle and stay synchronized with the transmitter.

Returning back to the data rate, a single clock cycle of PPM consists of four block times for encoding a single bit per cycle: the time for a clock pulse, an intermediate period between pulses, the time for a data pulse indicating a bit of 1, and another intermediate period between pulses. In Fig. 3 above, there is space for two data pulses and thus two bits can be transmitted every clock cycle. The block time used here is 4 ms, the minimum pulse period found for an early prototyping for a white light optical system as described in *Preliminary Testing Results*.

Using a minimum pulse period and intermediary period between pulses of 4 ms, the data rate $R$ can be computed as $R = (1\ bit)/(4\ blocks * 4 * 10^{-3}\ seconds) = 62.5\ bps$.

While many of the aspects of this requirement are not necessarily numerically quantifiable, one of the largest design requirements is that the AFμS system is as consumer facing as possible. This means making sure its features suit those that could be needed by the people of whom the specific functionality of the system is useful. These requirements are the addition of the modular sensor bay, ensuring that the final design is ergonomic enough to be easily deployed and retrieved, the ability to passively recharge itself, and that its price point is affordable for individual researchers, with the explicit goal of a sub-$500 product.

# Design Alternatives

## Communication

Besides waterproofing and pressurization, communication has historically been one of the largest hurdles to overcome when it comes to underwater vehicles. A variety of solutions have been implemented through the years in which active underwater exploration has been in vogue, though each has its own set of advantages and disadvantages.

The simplest of these solutions is running a physical wire between the transmitter and receiver. This has the benefit of being an extremely reliable method for data communication, as there is a provided transmission medium with favorable characteristics. The drawbacks of this system are due to having a physical connection between the transmitter and receiver. Not only

does this limit the distance they can be apart, but it also adds significant mass to the system, and measures to ensure that the line does not get caught or tangled need to be taken.

Though radio frequencies are one of the most prominent methods of communication in the 21st century, these signals attenuate extremely rapidly in water as a factor of their frequency. Several AUV systems, such as the Hydromea Exray[13], have implemented radio communication systems. To accommodate the attenuation factor, these AUVs need to utilize high powered transmitters and high quality receivers to get a relatively low range signal (~10m). Further ranges have been achieved by using extremely low frequency signals, though this requires very large antennas and results in a drastically slowed bitrate compared to higher frequencies, as can be inferred via the relationship between frequency, period and wavelength in a known medium.

As sound travels better in water than it does in air, it is one of the more widely used aquatic communication methods, such that the U.S. Navy has a communication standard called JANUS[14]. This method by far has the best range characteristics as a ratio of form factor, though it comes at the cost of extremely low bitrates. The frequencies that do best in water are below 100kHz, and effective range only increases as the frequency goes down. This limitation means that even if the data being transmitted had no encoding, the highest possible data rate in this frequency range is 50kbps. Another factor is that because sound travels so well in water, the background level of noise is higher for this across a wider range of water than any other communication method. This means that it is often the case that complex encoding schemes or purposeful information redundancy is required to account for higher BERs (Bit Error Rates) in this communication channel, slowing the bit rate even further.

[13] https://www.hydromea.com/exray-wireless-underwater-drone/
[14] http://www.januswiki.com/tiki-index.php?page=About+Janus

Similar to radio, light attenuates quite quickly in water, however, it has the advantages of not requiring hardware anywhere near as large or power intensive as radio transmitters to get a similar range. With the ubiquity of LEDs during this age, this is also a very frugal communication implementation. However, clear line of sight is a lot more important for optics than it is for radio, and impairments such as highly murky water or seaweed can affect light propagation to a much higher degree than radio waves. Similar to acoustics, there is a large amount of background noise in this medium, though the deeper the AUV goes, the less light from the surface there will be. Unlike acoustics, light travels very quickly, and the wavelength is nominally small, so very high bit rates can be achieved, albeit only over small distances.

As affordability is a key design requirement, and as communication is not vital for the AUVs navigation or base functionality, we decided to utilize optics, limiting our required communication range to 1.5m, but allowing for rapid transfer of information.

The "optimal" wavelength for this application should be established. Keeping in mind that the proposed audience for this product will be small research groups, local governments and colleges/universities, it is imperative that its widespread applicability be maintained. While lower wavelengths (~500nm, blue) attenuate less in ocean bodies, slightly higher wavelengths (~600nm) show less attenuation in more turbid conditions found in coastal waters[15], where this sub will be tested initially, and still boast relatively low attenuation is ocean bodies. The wavelength chosen for this application is 567nm (lime). Hence, range may be reduced but usability in a variety of environmental conditions is preserved.

[15] Johnson, L. J., Jasman, F., Green, R. J., & Leeson, M. S. (2014). Recent advances in underwater optical wireless communications. Underwater Technology: International Journal of the Society for Underwater, 32(3), 167–175. doi: 10.3723/ut.32.167

As with any communication system, transmitting and receiving elements are both needed. For transmission, viable options include laser diodes and high power LEDs. As discussed in the *Design Requirements* section, omni-directional optical emission is desirable, as well as low power draw. Laser diodes provide high switching speeds, however they are sharply limited in range, temperature stability, and require specialized circuitry to compensate for environmental factors[16]. While LEDs cannot switch as fast as laser diodes, they offer greater beam divergence, resilience, and as mentioned before, come at a much lower cost. Therefore LEDs were chosen as the component for transmitting optical signals.

For optical reception, the choice is not so linear. Viable options include photoresistors, phototransistors, and photodiodes. Beginning with the most commonly known component, photoresistors exhibit different resistances over different light levels in a fairly linear relationship. However, photoresistors can take anywhere from a few milliseconds to a few seconds to return back to a dark state after being exposed to light, which is not reasonable for a communication link. Phototransistors are essentially typical bipolar or field-effect transistors with their base/gate exposed to the light source. Therefore they exhibit a current gain and collector-emitter voltage proportional to the light level they receive. Phototransistors offer high robustness in the presence of noise, and can switch at a moderately fast rate ( =< 250 kHz). Photodiodes are components that convert light energy into electrical current. These components are very fast, capable of switching in the MHz region, relatively hardy to ambient noise, and are very affordable. However, as this component produces current, extra circuitry is required to

[16] Brundage, H. (2010). Designing a wireless underwater optical communication system. Mechanical Engineering - Master's Degree. Retrieved from http://hdl.handle.net/1721.1/57699

convert that current into a voltage level, such as a transimpedance amplifier, so it can be processed by the SBC.

A phototransistor was chosen for this application. While photodiodes display more desirable traits pertaining to performance, they are also capable of receiving optical signals from similarly powered LEDs over distances of 10 meters or more, which is beyond the requirements for this application. The design, testing and refinement of a transimpedance amplifier may also take a large amount of time to complete, which is not preferable when considering the time constraint of this project, and complicates the hardware requirements. Finally, as the expected data rate as described in the *Design Requirements* section is in the range of Hz as opposed to kHz, the good noise robustness and far simpler circuitry which the phototransistor offers makes it a suitable choice for this application.

Methods of signal modulation include using an ADC and extracting the data through software, and using a comparator circuit. Based on what ADCs exist within our price point, the selection may place another limit on the maximum data rate achievable, equal to half its maximum sampling rate, while delays in a comparator circuit are on the order of nanoseconds and are thus negligible. A comparator circuit will be implemented and tested first, followed by alternatives if necessary.

## Power

The energy reserve of this system will physically vary in size according to the available space in the vessel. The battery chemistry is selected based on which chemistry type offers the greatest energy density. Lithium-ion (Li-ion) and Lithium-polymer (Li-Po) batteries lead in this

respect, compared to options such as Nickel-Cadmium (Ni-Cd) and lead acid[17]. Due to widespread adoption of portable, rechargeable technological devices over the past several decades, these batteries are also available for a low price point.

The other consideration to incorporate is the number of cells to use in series and parallel. Increasing the number of batteries in a parallel configuration increases the maximum current draw of the array and the overall capacity. Again, this will be limited to the available space in the vessel. The number of cells for the series configuration modifies the efficiency of the buck-boost voltage regulators, and the speed that the motors can spin at as they are powered directly from the battery. Based on early PID tests described in the *Preliminary Testing Results*, a low RPM value is desirable for finer control and the greatest source of potential inefficiency in voltage regulation lies with the component that will be regulated continuously: the SBC. As SBC's are typically powered off of 5 V, the series configuration chosen was 2S, or a nominal voltage of 7.4 V.

# Processing

For the computational requirements of this system, the final performance has yet to be seen. The ATS Mk. 1 incorporates the Raspberry Pi 3B, and this SBC will be used until the computation requirements exceed its specifications. This SBC has 1GB of RAM and a quad-core processor with clock speeds up to 1.2 GHz. As more real-time systems are developed and integrated, CPU performance tests will be run to ensure delays are not beyond reasonability. Raspberry Pi's have the benefit of low cost, a massive community base, a plethora of third-party

[17] https://circuitdigest.com/article/different-types-of-batteries

software libraries, and backward-compatibility. These features will allow us to implement functionalities with existing libraries rather than creating them ourselves, which is highly useful considering the time constraints of this project. Hence, if the computing requirements of the 3B is not sufficient at greater offered load, the newly minted Raspberry Pi 4B (up to 4GB DDR4 RAM/quad-core processor up to 1.5 GHz)[18] may be a viable option, and if not then an SBC with superior computational characteristics such as the ASUS TinkerBoard S (2GB DDR3 RAM/quad-core processor up to 1.8 GHz/integrated graphics processor up to 650 MHz)[19] will also be considered.

# Sensing

## Movement

The most common method of motion detection for autonomy applications is GPS. GPS will provide the devices specific coordinates via satellite triangulation as long as the device is not in a region where there is enough attenuation that communication with the required number of satellites is impossible. Due to this stipulation, GPS does not work particularly well under water, and can in fact only function directly below the surface or else the attenuation factor is too great to provide accurate data.

Another method of motion detection, which has recently started to gain traction and practicality, is called optical flow. This uses a camera to determine velocity based on the rate at

[18]Raspberry Pi 4 Model B specifications – Raspberry Pi. (n.d.). Retrieved from https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/.
[19]Tinker Board S: Single Board Computer. (n.d.). Retrieved from https://www.asus.com/Single-Board-Computer/Tinker-Board-S/.

which pixels travel across the optical sensor. This method only works if there is a non-monochrome surface for it to be pointed at, and enough light for the sensor to be able to pick up the surface. While this would likely work in very shallow clear water, it would provide little functionality in water deep enough that the optical sensor cannot detect the floor.

A sensor called an IMU, or Inertial Measurement Unit, can be used to get motion data as well. This sensor reads in acceleration and orientation data, which can be utilized to calculate the velocity and displacement of the device. Unfortunately, all but the extremely expensive IMUs[20] have intrinsic noise and error in their readings that accumulate with the calculations of velocity and displacement, making this a very imprecise solution.

A method especially suited for underwater use is having a set base station which can be used to reference distance from the station to the device. This usually works by having the base station emit a signal that the AUV can use to calculate its distance and position in relation to the station[21]. This adds the need for a station to be placed in every location the AUV will be utilized in, limiting effective range and use cases significantly.

A sensor called a DVL, or Doppler Velocity Log, is commonly[22] used on human scale vessels, though modern technology has allowed the size of these sensors to decrease over time, making it possible to have them on smaller systems. This sensor uses three or more transducers to generate acoustic waves while under motion, and measures the doppler shift in frequency upon return of the signal. These sensors remain quite large and expensive relative to the other options at the moment.

---

[20] https://aerospace.honeywell.com/en/learn/products/sensors/hg1700-inertial-measurement-unit

[21] S. M. Smith and D. Kronen, "Experimental results of an inexpensive short baseline acoustic positioning system for AUV navigation," *Oceans '97. MTS/IEEE Conference Proceedings*, Halifax, NS, Canada, 1997, pp. 714-720 vol.1.

[22] J. Snyder, "Doppler Velocity Log (DVL) navigation for observation-class ROVs," *OCEANS 2010 MTS/IEEE SEATTLE*, Seattle, WA, 2010, pp. 1-9. doi: 10.1109/OCEANS.2010.5664561

For this project, we determined to attempt to utilize both an IMU and GPS module. The GPS module will function while the AUV is on the surface, and once the AUV submerges, it will utilize the less accurate method until it recalibrates on the surface.

The modern GPS modules available on the market in the <$75 price range, such as the SAM-M8Q, MT3339 and NEO-6M/V, all have similar sensitivity (~-162dBm) as well as similar power characteristics. The SAM-M8Q has the ability to connect to multiple constellations, and the Matek SAM-M8Q GPS Module breakout keeps a very low form factor and provides UART connection capabilities. This breakout has been selected to be the one used in our AUV.

## Water Sensors

As there is a large variety in the type of information that researchers are interested in collecting[23] [24] [25], it was determined that the most accommodating solution would be to simply provide a bay in which water sensors to suit the individual researcher's needs could be used.

## Local Awareness

There are several options for detecting objects in the local of a device. The simplest to implement is a physical sensor. In the case of an AUV, this would be a system of switches or pressure sensors around the exterior of the hull, which would alert the AUV of any direct contact with external objects. This is very low level, and therefore has the advantage of being able to use minimal hardware and processing to determine collisions.

---

[23] https://www.ysi.com/products/aquaculture-process-monitors-and-sensors

[24] http://news.mit.edu/2018/fundamental-equations-guide-marine-robots-optimal-sampling-sites-0510#separator-comments

[25] https://peerj.com/articles/1770/

Acoustic rangefinders are a common method of not only detecting an object, but detecting how far away it is. These systems are often used both on land and in water, though the specifics of the hardware differ based on medium. For this system to work, the object being detected must reflect the acoustic signal instead of absorbing it.

A similar functionality exists with the use of infrared. IR range finders can detect how far away an object is from itself, though only as long as the object reflects infrared. Due to the attenuation of this wavelength in ocean water, as can be seen in Fig. 4, this would not be the ideal solution for underwater purposes (Infrared falls in the ≥700nm range).

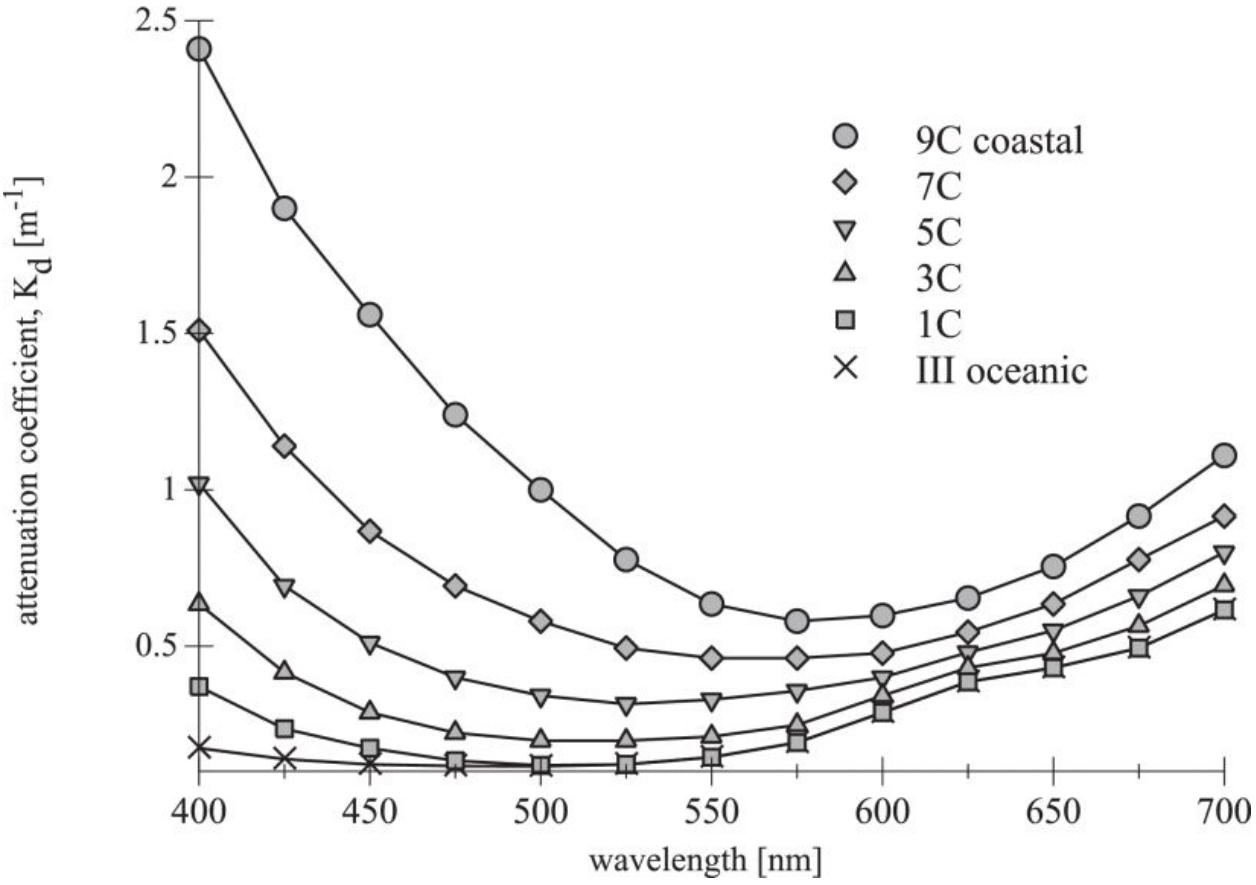



*Fig. 4. Relationship between attenuation and wavelength in oceans.*[26]

[26] J. Sticklus, P. A. Hoeher and R. Röttgers, "Optical Underwater Communication: The Potential of Using Converted Green LEDs in Coastal Waters," in IEEE Journal of Oceanic Engineering, vol. 44, no. 2, pp. 535-547, April 2019.

As minimization of environmental impact is one of the most important requirements on this project, any solution which involves direct contact with detected objects is not conducive to our specifications. Due to the attenuation factor in infrared sensing, acoustic sensing was determined to be the best option.

To accomplish this task, there are several ways to approach the problem. The first being the usage of completely off the shelf components, which has the advantage of minimal implementation time, and higher performance than could likely be achieved from scratch in the alloted time period. Unfortunately, all commercially available solutions have significantly greater functionality than required, and have price points and physical sizes to match. An alternative would be to purchase an off the shelf part designed to function in air, which would be significantly more available in the range of functionality price and size that we are looking for, and modify it to work in water. Though, due the physical differences in water and air, a large amount of modification would be required, and there is very little information available on the effectiveness of this. Eliminating the previously mentioned methods leaves the options based around the design and construction suited to our needs. For this, the options are to either purchase transducers designed for aquatic usage, or to construct a custom transducer to suit the design requirements. As the commercial availability of transducers that match the needs of this system is not prolific, it was determined that the feasibility of making custom equipment would be explored only to the point it was deemed impractical to pursue.

For acoustic signal generation, a transducer is required, which is a component that converts electrical signals to physical motion. Due to how well they are suited to the task,

piezoelectrics[27] seem to be the only standard method of making acoustic transducers. The method for unidirectional sonar purposes uses flat piezo elements sandwiched between two layers. One layer is the impedance matching layer, which should have an acoustic impedance half way between the piezoelectric's and transmission medium's impedances. The optimal thickness of this layer is debated to either be one quarter wavelength of the resonant frequency of the piezo[28], or half the thickness of the piezo itself[29]. Tests are planned to compare the performance of each of these methods. The other layer is required to be "immovable". That is, when the piezo is run at a frequency, making it contract and expand, the impedance matching layer should be producing the maximal amount of motion, while the other side is as relatively stationary as possible. As previously noted, acoustic waves travel furthest in water under values of around 100kHz, with the lower the frequency the better. A graph of this phenomena can be seen in Fig

[27] https://en.wikipedia.org/wiki/Piezoelectricity
[28] (n.d.). Retrieved from https://www.nde-ed.org/EducationResources/CommunityCollege/Ultrasonics/EquipmentTrans/characteristicspt.htm.
[29] Butler, J. L., & Sherman, C. H. (2018). Transducers and Arrays for Underwater Sound. Cham: Springer International Publishing.

5.

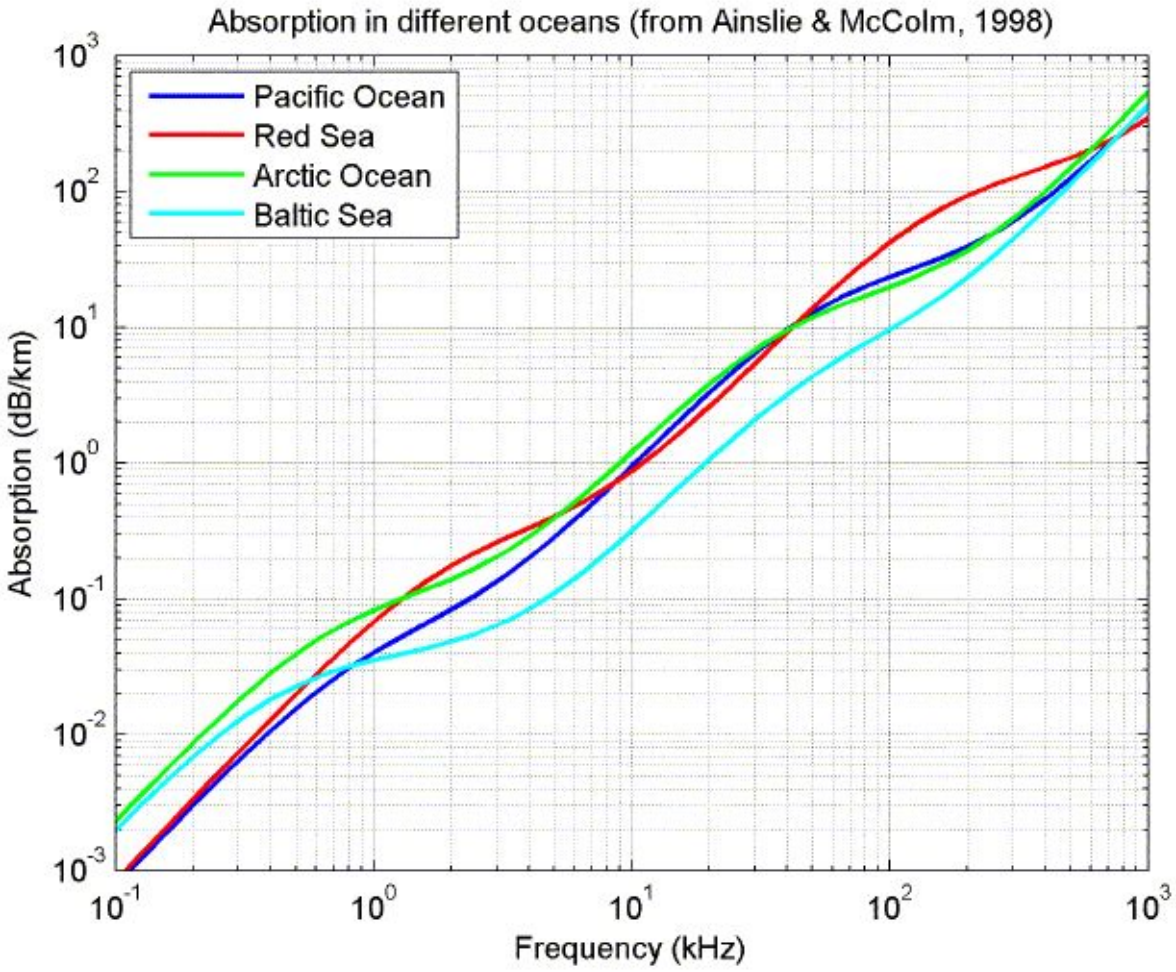



*Fig. 5. Relationship between attenuation and frequency in oceans.*[30]

This provides a wide range of options for the frequency selection of the AUVs transducers**.** Based on availability and price point, a selection of waterproofed transducers were ordered, with their resonant frequencies falling between 2.8kHz and 5kHz. These values are on the lower end of the usable range, with the initial intention of having a high range that could be applied to the potential of acoustic communication. The main drawback of using said range is the length of a

[30] Ainslie M. A., McColm J. G., "A simplified formula for viscous and chemical absorption in sea water", Journal of the Acoustical Society of America, **103**(3), 1671-1672, 1998.

single period in the transmission medium, as the longer this is the more it could overlap with other data, though the extent of this as an issue will be explored through testing

To transmit acoustic signals, the piezo devices produce higher amplitude waveforms at higher voltages. As the sensitivity of the receiving methodology plays a large factor in the required amplitude of the received signal, the requirements for the driving voltage will be dynamically determined through testing.

There are several options for processing the received acoustic signals. The raw data will be analog AC voltage, which contains all vibrations imparted on the piezoelectric, not just the ones which contain information from the other AUVs. The first option is to run the signal through an ADC and perform the processing on the SBC. This has the disadvantage of increasing SBC load, but it also means that there is minimal hardware required to read in this data. An alternative to this would be to implement a daughterboard to the SBC which contains analog read in capabilities, and would perform all of the processing before passing the final data to the SBC.

If the data is to be processed beforehand, a filter is required. Though the signal may experience some frequency distortion, it will still approximately be the same as the piezo's resonant frequency. To remove background noise, a bandpass filter, which allows a specific frequency range to pass through, will be required. This signal, which would now ideally only contain the amplitudes of signals received in the exact frequency range of the transmitted waves, could then be fed through an ADC to the SBC for further processing. This method has the advantage of removing a portion of the processing requirements while maintaining information about how strong the received signal was. However, in the situation where there is noise within

this frequency range, there would still be a good amount of processing required to detect and extract the features of the actual data.

An alternative to this would be to replace the ADC in the previous configuration with a comparator, such as the LM311P, which would have a set threshold, and if the received signal is above said threshold, a digital 1 would be output to the SBC. This would effectively directly convert from the received AC signal to digital data in hardware, removing all significant processing from the SBC. This has the drawback of losing amplitude information, which could be used to extrapolate location information or to algorithmically reject certain types of data based on its characteristics.

There are a lot of ways that sound waves can be distorted in water, and a primary concern is multipath, where the signal ends up bouncing off of various surfaces, concluding in the same transmission reaching the receiver across a range of times. The signal can also bend, shift frequency, or heavily attenuate, and we have yet to perform the testing which will inform us of how these problems will need to be approached, which guide how to optimally decide the hardware configuration.

# Movement

Several propulsion methods were explored by the mechanical engineering team, and as the control of thrusters is an interdepartmental effort, the choice of methodology affected the electrical design.

The options were determined to fall within two categories; Biomimicry or propellor based systems. Biomimicry, or the copying of methods used by organisms, has been a subject of

interest in the development of AUVs[31] [32] . A large amount of this research shows that these propulsion methods are highly efficient, which suits the needs of this project. However, these methods require a large number of complex moving parts, which reduces the robustness of the system. The thrust vectors produced by biomimicry options are also significantly less controlled than traditional propellor based propulsion methods, and due to these limitations, the mechanical engineering team chose to pursue the propellor based path in the interest of having a functional product within the designated time frame.

Traditional propellers, with the blades extending outwards from a central axel, have a significant amount of data collected on their characteristics, and therefore designing them to match a performance is relatively simple. However, when it comes to their actual usage for an AUV of this scale, there are a large number of drawbacks that show up. The foremost is the tendency of this prop type to get caught up in seaweed, line, or any other fibrous free floating material. Having the blades externally facing also provides the opportunity for the harm of wildlife[33], which violates one of our design requirements.

To solve this, Alex Pradhan, a member of the mechanical engineering team, designed a hubless rim driven propellor with hydro lubrication. This has the advantage of being significantly more difficult to tangle, and not having potentially harmful elements directly exposed. Alex worked with Xavier to write an optimization algorithm for the blade characteristics, and

[31] Fish, F.E.; Schreiber, C.M.; Moored, K.W.; Liu, G.; Dong, H.; Bart-Smith, H. Hydrodynamic Performance of Aquatic Flapping: Efficiency of Underwater Flight in the Manta. *Aerospace* **2016**, *3*, 20.
[32] Font D, Tresanchez M, Siegentahler C, et al. Design and implementation of a biomimetic turtle hydrofoil for an autonomous underwater vehicle. *Sensors (Basel)*. 2011;11(12):11168–11187. doi:10.3390/s111211168
[33] https://www.nationalgeographic.com/animals/2019/07/north-atlantic-right-whales-mass-mortality/

production and testing of the thruster will be performed in 2020. A CAD model of the design can be seen below in Fig. 6.

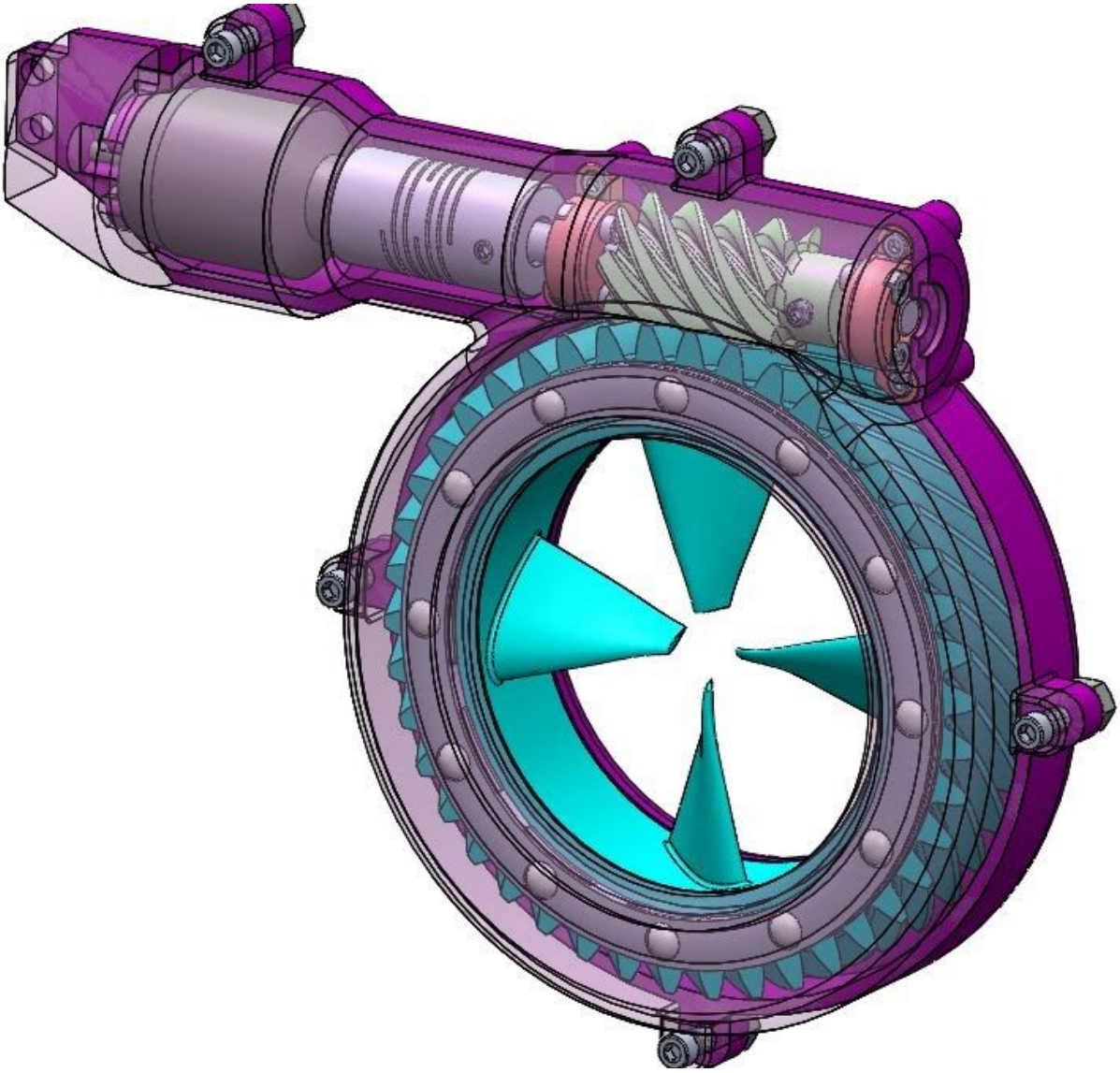


*Fig. 6. CAD model of hubless rim driven thruster*

While the required characteristics of the motor are not yet known, the motor selection can be narrowed down based on suitability for our requirements. The types of motors that are easily accessible within this size range and are able to provide speeds high enough to fall in the predicted range of thruster requirements are DC brushed motors or brushless outrunners.

Brushed motors utilize brushes to generate and alternating current in the rotor, where permanent magnets induce electromagnetic force on the rotor, causing it to rotate at a velocity proportional to the input voltage. Due to the constant contact between the spinning rotor and the brushes, brushed motors are extremely prone to failure due to wear. As the brushes are live contacts, any introduction of water to the interior of the motor would result in immediate termination of its functionality,

Brushless motors utilize 3 coils provided with a 120° phase offset AC voltage to induce a rotational velocity proportional to the input frequency from the magnets positioned on the rotor. As there is no direct contact between the stator and the rotor anywhere besides the bearings, there is very little wear on these motors over time. As the coils are insulated, these motors are also able to function while submerged as long as the connection to the ESC is waterproofed.

Due to their increased lifespan and suitability for aquatic usage, we have determined brushless motors will be used for the final thruster. Depending on the predictability of the thruster characteristics, an encoder may be included with the motor to get direct feedback on rotational velocity to ensure synchronization of thrust. This will be determined based on the results of preliminary thruster testing.

# Preliminary Proposed Design

As there are so many interlinking aspects that will need to function in conjunction for the final system to work as designed, a minimum viable product[34] approach is being used for the development of the system. This means that versions of the system which are able to represent

[34] https://en.wikipedia.org/wiki/Minimum_viable_product

minimal subsets of the final systems' functionality to allow for testing before implementation into the next iteration. The scope of this project has been divided into three different iterations, referred to as the MKI through MKIII. The MKI is designed to exhibit 2D motion in water, and provides a testbed for motion sensing and navigation code. The MKII has the addition of communication and 3D navigation capabilities, allowing for testing of a significantly greater functionality. The final iteration will implement all aspects and components as designed by the mechanical engineering and computer/electrical engineering teams.

The software and hardware aspects of design for this project will be examined in two separate sections.

# Hardware

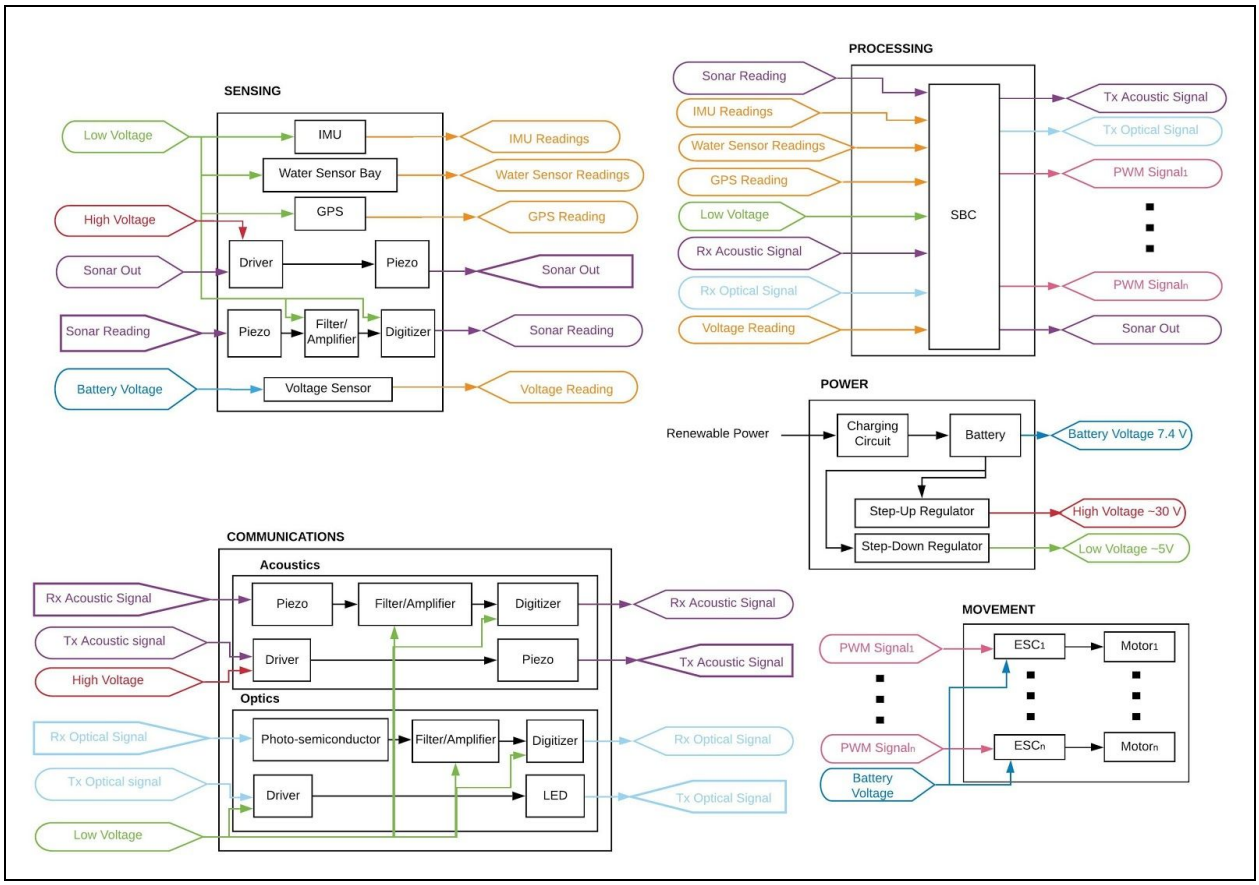



*Fig. 7. Hardware Block IO Diagram*

There are five main blocks in the hardware design: Communications, Movement, Power, Processing, and Sensing. Each of these blocks are described in detail below.

## Communication block

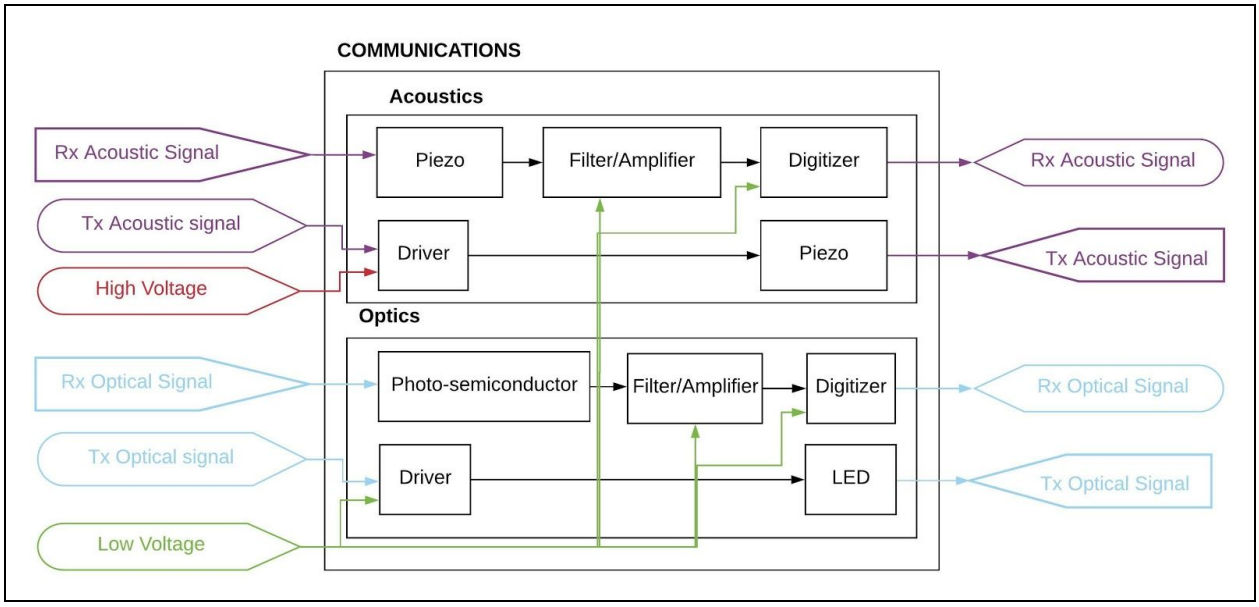



*Fig. 8. Hardware IO diagram for Communications block*

The Communications block is responsible for sending and receiving all inter-sub messages. While this block is not expected to form these messages, and in fact is expected to treat all messages agnostically, it handles all physical transmission and reception of both optical and acoustic signals to the degree that it can be digitally processed by the SBC. This section is divided into two subsections: Acoustics and Optics, which details the internal structure of the two methods for communication.

For transmitting optical pulses, an n-channel MOSFET transistor circuit will be used in a switching application to drive a high power LED. The high power LEDs used here are 3 W Power LEDs with a viewing angle of 120 degrees, a maximum current draw of $I = 1A$ and a forward voltage of $V_f = 3V$ . A thorough justification of this implementation can be found in the *Design Alternatives* section. A diagram for this circuit is shown below.

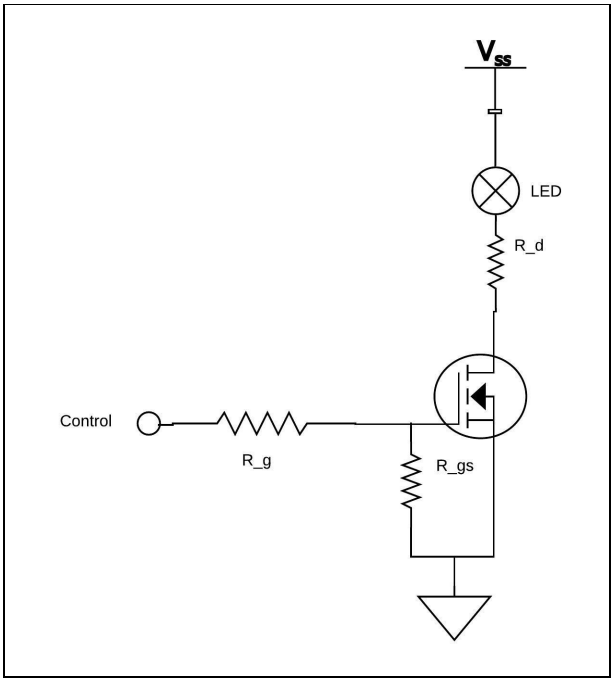



*Fig. 9. Optical emitter transistor switching circuit*

The transistor used in this application will be a power MOSFET with a high continuous drain current, made necessary by the draw characteristics of the LEDs. As mentioned in the *Design Requirements* section, omni-directional optical emission during transmission is desirable, so a single MOSFET with a maximum continuous drain current greater than the current draw of multiple power LEDs running simultaneously would simplify circuitry and keep the price low. As this MOSFET will be driven by a GPIO pin of the SBC (3.3 V), a single MOSFET would also prevent potential draw issues at the gate, as opposed to multiple MOSFET gates connected to a single GPIO pin. The transistor chosen for this application is an IRLB8721PbF MOSFET, which has a continuous drain current $I_D = 62$ A, max drain-to-source voltage $V_{DS} = 30$ V, gate threshold voltage $V_{GS,th} = 1.8$ V, drain-to-source resistance $R_{DS,on} = 8.7$ mΩ, and switching delay times in the tens of nanoseconds.

In *Fig. 9*, R_g is implemented to reduce buzzing at rising/falling edges during switching. The value of R_g will be kept small in order to prevent the consequential RC delay (from R_gate and C_gate-source) from limiting the overall switching speed of the system.

R_gs is implemented to ensure that noise and stray internal capacitances, often known as "Miller Capacitance", do not accidentally turn the gate on. This is especially necessary in high frequency switching applications.

R_d will be used to limit current draw from V_ss. Due to the LEDs low forward voltage of V_f = 3 V, V_ss can be the low voltage of 5 V, as discussed in the Power block section. Therefore, the optimum R_d value for I_d = 1A is $R_d = \frac{5V-3V}{1\,A} - 8.7m\Omega = 1.99\Omega$.

For receiving optical pulses, a 570nm phototransistor will be utilized. A full justification for this choice can be found in the *Design Alternatives* section. A diagram for this circuit is shown below.

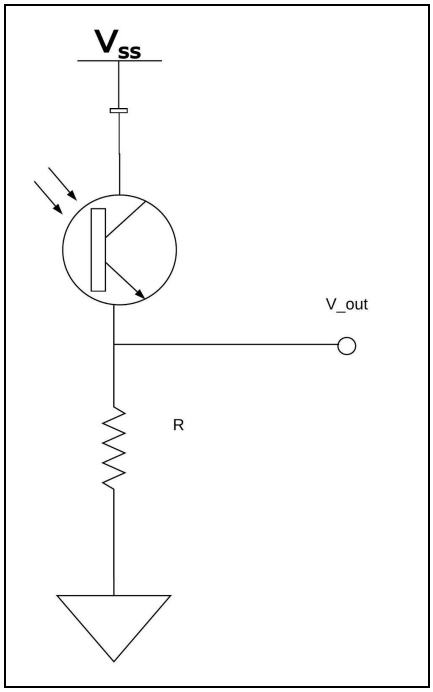



*Fig. 10. Optical receiver circuit*

The common-collector circuit consists of a phototransistor and a resistor in series, with V_out = V_R. When light is present and the phototransistor is saturated, it has a very low voltage across the collector and emitter. When light is absent, it has a higher voltage. V_out will feed into a comparator circuit, which will output a logical 1 for the microcontroller when a strong enough signal is detected by the phototransistor. Through empirical testing, the optimal threshold voltage will be determined and implemented into the comparator circuit

The value of R will be selected according to how sensitive the circuit must be to different light levels. The phototransistor used in this application is an SFH 3310 570nm NPN phototransistor, with V_CE_max = 5.5 V and I_CC_max = 20 mA. With V_ss = 5 V, the minimum resistor value is $R_{min} = \frac{5V}{20mA} = 250\Omega$. However, this condition is only reached at full saturation, which may not ever be reached in this application. Therefore the final value of R will reflect the maximum saturation that can be reached.

## Power block

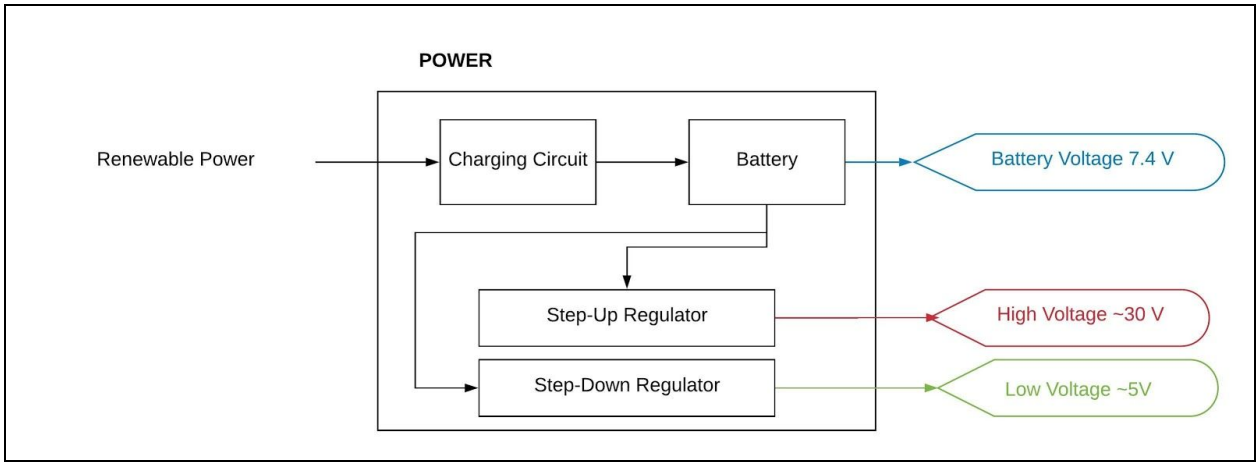



*Fig. 11. Hardware IO Diagram for Power block*

The Power block is responsible for powering all active components in the system, and for generating power renewably while away from a charging station. The power reserve will be some configuration of common off-the-shelf batteries with a single cell voltage of around 3.7 V, therefore requiring that step-up and/or step-down regulators be used to provide at least two distinct voltage levels: one of a relatively low voltage (~5V) for powering the numerous low-power components, and one of a relatively high voltage (~30V) for driving piezoelectric transducers.

The battery will have a nominal voltage of 7.4 V, also known as a 2S configuration. The SBC will need to be continually powered during any given mission, so selecting a nominal battery voltage close to the actively regulating voltage will minimize power loss. Using a lower nominal voltage will also lessen the power consumption from the motors. The mechanical engineering team working on this project have implemented a helical gear train calculator in their ongoing thruster design with a modifiable gear ratio, meaning that thruster speed and torque can be tuned mechanically. With driving high performance motors, active voltage regulation and periodic high current draw from communicative components, the battery must boast a high maximum continuous current draw and decent capacity. Therefore the battery will include several individual batteries in parallel, creating a 2S*x*P configuration. The final value of *x* will be determined by the available space in a single sub. Please refer to the Movement block section for more information regarding motors and the Communications block section for more information regarding communicative components.

The step-up regulator will provide a high voltage level for the piezoelectric transducers to be driven off of. Generally speaking, AC waves produce greater displacement in a piezoelectric

transducer when higher voltages are utilized, meaning that a high driving voltage is needed to send acoustic waves over large distances. For more information about transducers, please refer to the Communications block section. The implementation for this regulator will be a boost switching converter, ideally with high efficiency ratings and low latency. As this regulator has a single relatively-stable load, large continuous current ratings are not required.

The step-down regulator will provide a low voltage level for the SBC, IMU module, GPS module, optical driving circuit, and all active filters/amplifiers. The regulation for this voltage will be continuous as discussed before, however the load will largely vary; driving circuits will draw power periodically and in bursts, and filters/amplifiers will be similarly pulsed. Therefore, it is important that this regulator has a high current rating and handles under non-linear conditions. The exact current draw will be dependent upon the total number of LEDs used and other particulars of the system, but may very well be up to 10 amps. The implementation for this regulator will be a buck switching converter. This use case, being the conversion of some battery voltage to 5 V, is a common issue in RC circuits. While primary motors are driven at the battery voltage, other components such as servo motors and the receiver are typically powered off of 5 V. In order to avoid needing to use a second battery for these components alone, a Battery Eliminator Circuit, or BEC, is used to step down the battery voltage to 5 V for only these components. Therefore, regulators that can handle nonlinear supply voltages and loads are relatively cheap, tried, and true.

Finally, the charging circuit will provide a means for some renewable power to be harvested and recharge the battery. The implementation of this component will be handled by the

mechanical engineers on this project. The only requirements for this circuit is that it recharge the battery entirely, and does so completely renewably.

In an effort to gauge what the power draw might be, calculations of consumption for each component to be included in this design is performed and provided below. These values are for a worst-case scenario, the details and justifications of which are provided in the following paragraphs.

| **Component** | **Notes** | **Final Power Consumption (Wh)** |
|---|---|---|
| SBC - Raspberry Pi 4B | *Overclocked @ max utilization*[35] | 8 |
| Piezoelectric transducers | With 6 transducers, $Z_{resonant} = 300\Omega$, $V = 30V \rightarrow$ $P = \frac{V^2}{Z_{resonant}} * 6\ transducers = 18W$, assuming 50% on time $\rightarrow$ $E = 9Wh$ | 9 Wh |
| LED driving circuit | With 6 LEDs and FET characteristics of $V_{cc} = 5V$, $I_c = 1A$, $R_{ds(on)} = 8.7m\Omega \rightarrow$ $P = (6\ LEDs * I_c)^2 * R_{ds(on)} = 0.31W$, assuming constant transmission $\rightarrow$ $E = P * \frac{on\ time}{off\ time} = 0.31W * \frac{3}{8} = 0.12Wh$ | 0.12 |
| LEDs | With 6 LEDs, $P = 3W$, assuming constant | 6.75 |

[35] https://www.pidramble.com/wiki/benchmarks/power-consumption

| | | |
|---|---|---|
| | transmission $\rightarrow E = 3W * 6\ LEDs * \frac{3}{8} = 6.75Wh$ | |
| IMU | With IMU characteristics of $V_{DD} = 3.3V,\ I_{DD} = 12.3mA \rightarrow P = 0.041W$, assuming constantly on $\rightarrow E = 0.041Wh$ | 0.041 |
| GPS | With $I_{continuous} = 29mA,\ V = 3V \rightarrow P = 0.087Wh$ | 0.087 |
| Filters/ Amplifiers | With 6 LEDs, 6 transducers, $V = 5V$, $I = 7mA \rightarrow P = 12 * 5V * 7mA = 0.45W$, assuming 50% on time for transducers and constant transmission for LED $\rightarrow$ $E = (\frac{0.45W}{2} * 0.5) + (\frac{0.45W}{2} * \frac{3}{8}) = 0.197Wh$ | 0.197 |
| Digitizers | With 6 LEDs, 6 transducers, $V_{cc} = 3.3V$, $I_{in} = 7.5mA \rightarrow P = 12 * 3.3V * 7.5mA = 0.297W$, assuming 50% on time for transducers and constant transmission for LED $\rightarrow$ $E = (\frac{0.297W}{2} * 0.5) + (\frac{0.297W}{2} * \frac{3}{8}) = 0.13Wh$ | 0.13 |
| ESCs/Motors | With $V_{nominal} = 7.4V$, $I = 15A$, 5 motors, and 50% on time $\rightarrow P = 111W$, assuming 5 motors and 50% on time $\rightarrow E = 111W * 5 * 0.5 = 277.5Wh$ | 277.5 |
| Phototransistor | With phototransistor characteristics of | 0.04 |

| | | |
|---|---|---|
| circuit | $V_{ce,\,max} = 5.5V$ , $I_{c,max} = 0.02A \rightarrow P = 0.11W$ , assuming constant transmission $\rightarrow$ $E = 0.11W * \frac{3}{8} = 0.04Wh$ | |
| Piezoelectric transducer driving circuit | With 6 transducers, $I = \frac{30V}{300\Omega} = 0.1A$ and FET characteristics of $V_{cc} = 30V$ , $R_{ds(on)} = 8.7m\Omega \rightarrow$ $P = (6\ transducers * I)^2 * R_{ds(on)} = 0.003W$ , assuming 50% on time $\rightarrow$ $E = 0.0015Wh$ | 0.0015 |
| | **TOTAL:** | **301.8665 Wh** |

*Table. 1. Power consumption breakdown of all hardware components*

For the piezoelectric transducers, 6 transducers are used in this calculation. This provides sonar readings in all directions of all axes. The possibility of obstacles in all axes is a concern, but as LED placement is also omnidirectional, this will allow a sub to prevent collisions with other subs as local awareness is also omnidirectional. The on-time to off-time ratio used here is 0.5, as described in the Sensing block.

For LED driving circuit, 6 LEDs are used in this calculation. As stated in the Communication block, the power LEDs used in this design have a max current draw of 1 A and a viewing angle of 120 degrees. Therefore, 6 LEDs allow for omnidirectional visibility. For the ratio of on-time to off-time in the same category, the communication protocol as described in the design requirements section (PPM) describes the ratio of on-time to off-time for sending a bit '0' or a bit '1'. For '0', the clock pulse is the only on-time, so this ratio is ¼. For '1' both the clock pulse and the data pulse make up the on-time, so this ratio is ½. Therefore, the average on-time

to off-time ratio is ⅜. The FET used here is the one selected for this design as described in the Communication block.

For filters/amplifiers, the exact part selection has yet to be completed for reasons described in *Design Alternatives*. For calculations performed above, a universal active filter similar in functionality to our use case, the UAF42, is used for calculations with specifications listed in the above table.

For digitizers, a differential comparator as described in *Design Alternatives* is used for calculations. This comparator is used for digitization of both optical signals and sonar readings due to its wide operating range.

For ESCs/motors, the voltage figure is the nominal voltage for the battery configuration this design will utilize and the current draw is based on the maximum current rating for the bidirectional ESCs currently used. The mechanical engineering team was consulted for the greatest number of motors that may be incorporated in the design, for which the answer was 4. In this configuration, 2 motors would be used to propel the AUV forward as a tank-drive propulsion system, and 2 motors would be used to surface and dive the AUV. The average on time assumes that two motors on the AUV will always be spinning; either the two motors for forward motion, or the two motors for vertical motion.

## Processing block

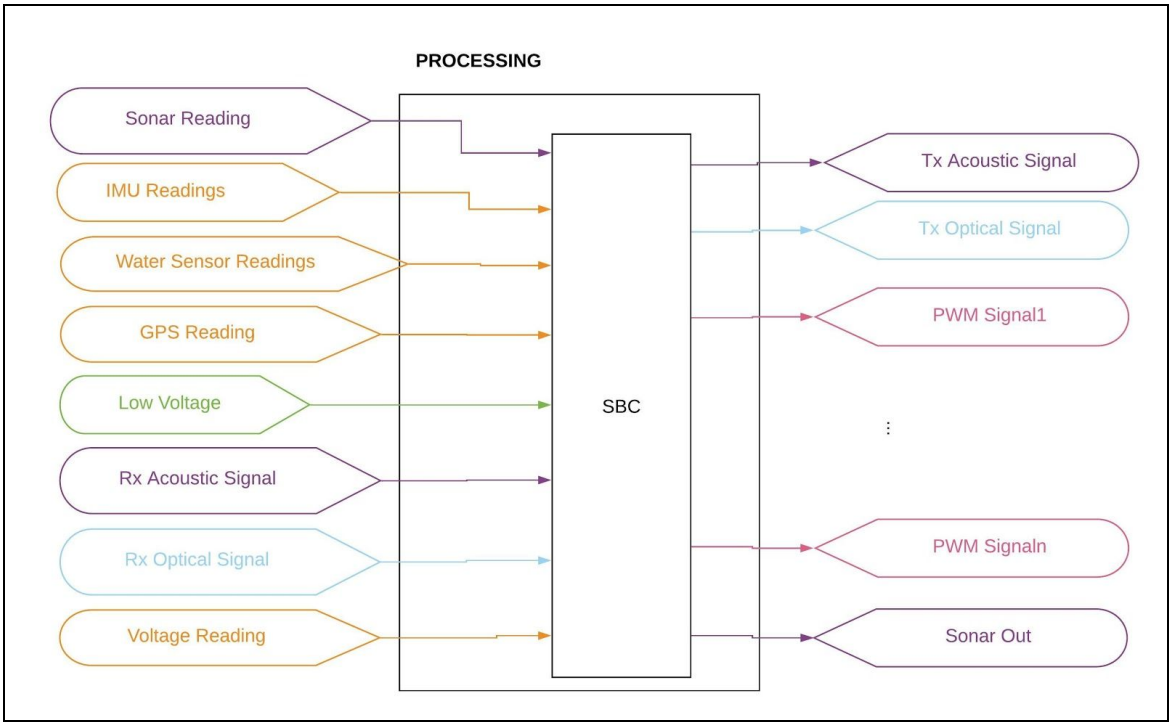



*Fig. 12. Hardware IO Diagram for Processing block*

The Processing block is responsible for all computational requirements. Therefore its inputs are all readings from the Sensing block, all digitized incoming communication signals, and low voltage for powering. It's outputs are digitized outgoing communication signals and PWM signals for all motors/control surfaces. For more information about the communication signals, please refer to the Communications block.

The processing block itself does not have any internal hardware implementations aside from the SBC. The SBC essentially contains the entire software design, making for a short conversation in a hardware IO breakdown. A full examination of the IO requirements for this SBC is provided below, along with a list of all hardware components that the SBC will interface

with and their IO. These requirements are based on a worst-case scenario that has been established in the Power block.

| Block | Component | I/O Required |
|---|---|---|
| Sensing | Piezoelectric transducer driver circuit | GPIO pin |
| Sensing | Piezoelectric transducer digitizer | GPIO pin |
| Communication | LED driver circuit | GPIO pin |
| Communication | Phototransistor circuit | GPIO pin |
| Sensing | Battery voltage sensor | I2C |
| Sensing | IMU | I2C/UART |
| Sensing | GPS | UART |
| Sensing | Sonar out | GPIO pin |
| Sensing | Sonar reading | GPIO pin |
| Sensing | Water sensor bay | I2C |
| Movement | Motor drivers | GPIO pin*5 |

*Table. 2. IO breakdown of all hardware components that interface with the SBC*

As can be seen above, the overall IO requirements for the SBC include: 11 GPIO pins, two I2C ports, a UART port, and one more I2C/UART port. As described in the *Design*

*Alternatives* section, the current SBC being used is the Raspberry Pi 3B. This SBC has 2x I2C, 2x UART, 2x SPI, and 14x dedicated GPIO pins (that is, not also for I2C/SPI/UART). It should be mentioned that one of UARTs on the Raspberry Pi is tied to its wireless/bluetooth module which is used for interfacing with the pi during tests, effectively making this 1x UART. As I2C can be performed by chaining multiple devices, the water sensor bay I2C channel will be chained to one of the two I2C ports and properly addressed to ensure that there is no address collision between I2C peripherals. Therefore, this SBC still has the necessary GPIO to house all peripherals stated in this design.

For process-based design, please refer to the Software section for more information.

## Sensing block

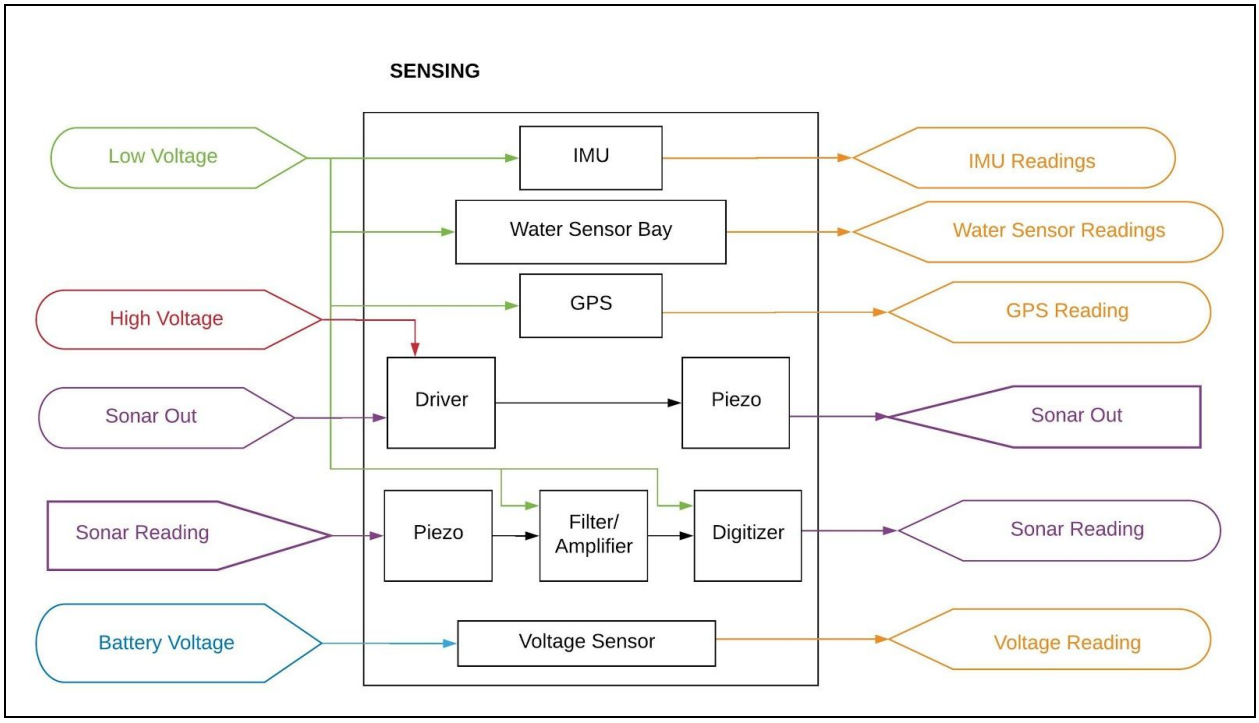



*Fig. 13. Hardware IO diagram for Sensing block*

The Sensing block is responsible for handling all sensory input. This does not include communication between individual systems. There are five sensing modules involved in this block: an IMU module, a GPS module, a water sensor module (a bay containing multiple water sensors), a sonar module, and a battery voltage sensor module. The first four sensing modules will require power at a low voltage, and the last sensor module will require battery voltage.

The purpose of an IMU module is to provide estimations for velocity and displacement underwater, necessary for dead-reckoning. A desirable IMU for this application is a 9-DOF sensor, which includes an accelerometer, capable of providing instantaneous acceleration readings, a gyroscope, capable of providing orientation and angular velocity, and a magnetometer, capable of gauging magnetic fields (most commonly, Earth's magnetic field). This reports absolute orientation in 3-space and motion estimations in all axes.

Calculating displacement from instantaneous acceleration results in unbounded error accumulation, spurring the need for some post-processing and filtering on the collected data. This work is explored under *Preliminary Testing Results*. After comparing similarly priced IMU's, the BNO055 absolute orientation sensor was chosen due to its relatively high accuracy, configurable sampling rates and onboard sensor fusion and processing capabilities. The second point was especially poignant, as sensor fusion is a complicated field that would take significant time to develop and refine (potentially a senior capstone project on its own), and onboard processing means those computations can be offloaded on the SBC- a desirable trait because these readings will happen continuously during a mission and the possibility of software delays due to absolute orientation calculation is abated.

The purpose of the GPS module is to reference the sub to a global coordinate frame and realign all estimated positions from dead-reckoning. This allows for course correction if the sub accidentally strays from its projected path. For point-to-point travel, GPS will also provide a means to definitively start and conclude a mission. The SAM-M8Q chipset, selected based on the criteria outlined in *Design Alternatives*, was released in 2017.

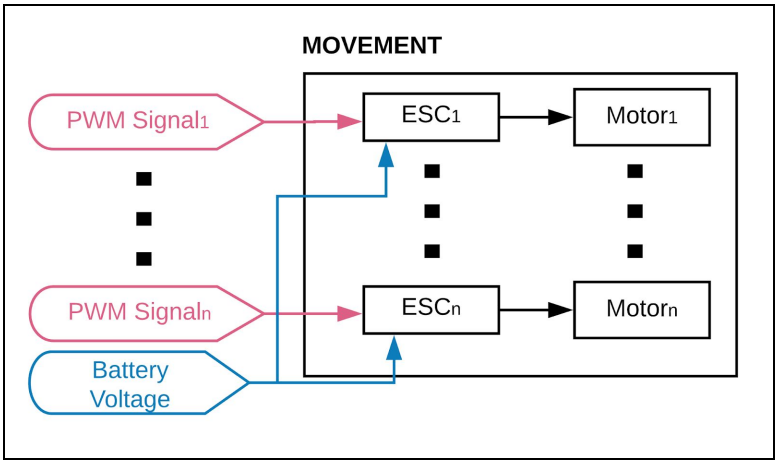
The purpose of the water sensor module is to generate the underwater data that the user is interested in, allowing them to choose what sensors best suit their purpose. With the water sensors that the user decides to equip their sub with, readings will be taken periodically and timestamped for later post-processing. The water sensor bay will be modular, enabling any water sensor with I2C capabilities to be attached to the bay. This will allow the user to decide what factors they care about and implement sensors accordingly. Additionally, this keeps the overall price lower because expensive sensors won't be implemented as a default option which might not be needed at all. In the instance in which a user wants to execute point-to-point travel, they may choose to omit water sensors altogether.

The purpose of the sonar module is to provide basic information about local surroundings to the sub. This implementation will be as selected in design alternatives, though the specifics of the hardware will not be confirmed until preliminary tests are performed. Sonar is a necessary implementation to ensure that the sub does not contact any walls, floors, or ceilings while underwater because position estimations may not be fully accurate. To match our requirement of being able to detect obstacles at a distance of 1.5 meters, the transmission power will be selected based on the results of our initial acoustics testing. It takes sound $\frac{3m}{330m/s} = 0.009s$ to travel a distance of 3m. To meet the requirement of being able to avoid an obstacle with a relative

velocity of 1m/s, the AUV would have $2s - 0.009s = 1.991$s of time to negate its relative velocity. Based on the design requirement of being able to exhibit at least 1m/s of acceleration, this gives the AUV an overhead of 0.991s to achieve this, as it would reach a negating velocity after 1s. Sensors will be positioned top, bottom, fore, starboard and port, with the potential for multiple sensors dependent on the final length of the AUV as determined by the mechanical engineering team.

The purpose of the battery voltage sensor module is to detect when the battery must be recharged. This is strictly vital, as a battery voltage below the operating range of the step-down regulator will result in shutdown of the SBC, at which point the sub will be lost. Once a low battery voltage is detected, with a factor of safety integrated to ensure the sub can resurface before shutting off, the sub will do so and recharge. This will be accomplished with the 2S fuel gauge MAX17044 with the sparkfun breakout board[36] which uses I2C to interface. When each cell reaches the suggested minimum threshold for the selected battery chemistry.

## Movement block




[36] https://www.sparkfun.com/products/10617

*Fig. 14. Hardware IO diagram for Movement block*

The movement block will simply consist of the selected motors and a reversible ESC for each motor. Each ESC will require a single PWM channel. The thruster design and layout, along with the motor selection and number of motors, falls under the jurisdiction of the mechanical engineers. Once the parameters of the motors have been chosen to match the relevant requirements, ESCs will be selected to match these characteristics. Thruster flow testing and iteration through rapid prototyping will occur at the beginning of 2020, and once the speed to thrust relationship is determined, the motor requirements will be finalized.

# Software

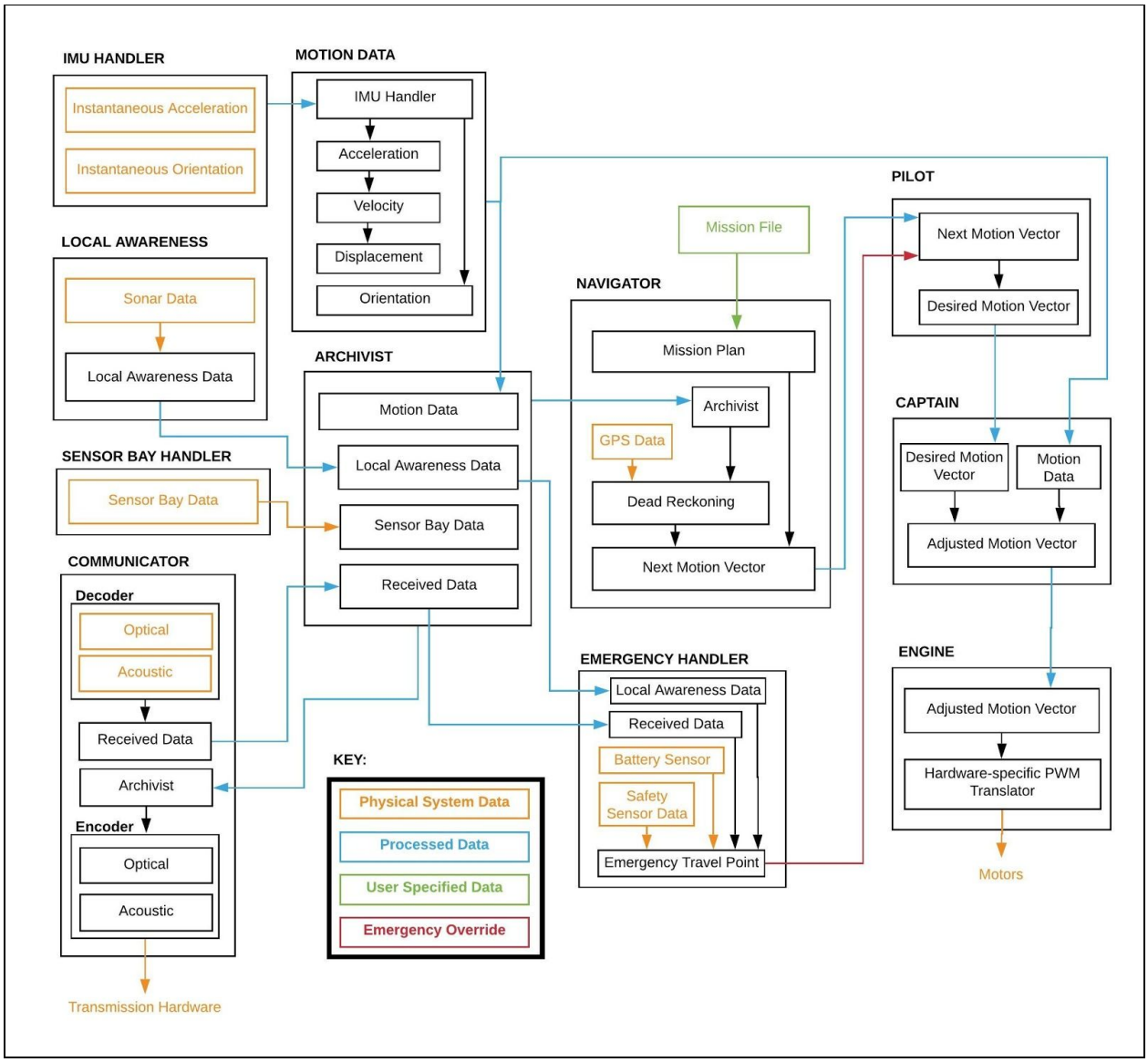



*Fig. 15. Software Diagram*

There are eleven main sections of code for the system, each of which will be described below in detail. Fig. 15 contains a representation of the connections between these systems.

## IMU Handler

The IMU Handler is responsible for initializing, calibrating and starting the IMU's measurement. It has a function that returns the most recent acceleration and orientation data when called. The BNO055 interfaces using I2C, and is being accessed via the adafruit unified sensor library. When this is initialized, calibration data is loaded in from the previous successful calibration, and continues running until all values have reached their maximum calibration level. The IMU Handler reads in the linear acceleration vector and the euler vector via the library.

## Local Awareness

Depending on the finalized results from the acoustic testing, the precise handling of incoming data will be different based on the finalized selection of hardware, however, the main functionality of this block remains the same. The local awareness block will periodically instruct the acoustic system, described in the *Hardware* section of *Design* to broadcast a signal, and then listen for the same signal to return. The time gap between sending and receiving will be utilized in conjunction with the speed of sound in water to calculate how far away the object that the signal bounced off of was. The transmission hardware should be perfectly functional as receiving hardware, though testing to confirm that this is the case remains to be done. At the required awareness range of 1.5m, sending and receiving a pulse would take 4.54ms to respond. The encoding is necessary to ensure that the received signal was not broadcast from another sonar sensor. The distances detected for each transducer can then be logged by Archivist.

## Sensor Bay Handler

The Sensor Bay Handler will simply interface with the modular sensor bay connections. A protocol will be designated that these sensors will be required to use, and therefore any sensor connected in accordance with this will be able to have its data read. This data will be collected at a user specified period, and the resulting data will be collected by Archivist.

## Communicator

The Communicator will be the interface with the communication hardware. Transmission will entail determining the transmission method based on content, encoding the data to be transmitted, waiting for its turn to transmit, and transmitting said data.

The communicator will also consistently listen for incoming transmissions. Upon reception, and processing depending on the state of the received data, the information will be decoded and ready to be archived.

## Motion Data

Motion Data starts a daemon process which queries IMU Handler for the instantaneous acceleration and orientation values. Acceleration and orientation values are collected and added to logs as they are measured. Once the number of new acceleration values reaches a predetermined range, R, the most recent R values are passed into the filter function. Because the data is being filtered in relation to frequency, the filter is affected by the size of R. The double integration  drastically amplifies even the smallest of errors, so a lot of filtering and processing is

required to ensure that the displacement data even resembles reality. The R values of filtered acceleration are appended to its own log, and from it the integrated velocity and displacement ranges are generated and appended to their own logs. Further development and testing is required to determine if this method is capable of permitting our design requirements without a functional GPS connection, and the current progress of this testing is available in *Preliminary Test Results.* It is important to note that the functionality of the dead-reckoning system is not a direct indicator of the AUVs translocational capabilities, as it will only serve as an input to the navigation system.

## Archivist

Archivist is the section of code responsible for collecting, logging and providing all important information. It takes in the collected information from Local Awareness, the Sensor Bay Handler, Motion Data, and Communication, which contains the information received from other AUVs. Archivist then ensures that all data that is frequently used by other modules is held in memory, while all else is stored in a database and removed from memory.

## Navigator

The Navigator is a high-level mission control block that performs several functions. First, it generates a mission plan from a user-provided mission file. This includes the mission type (i.e. exploration) and coordinates of the pick-up location at a minimum. Second, it retrieves the most recent motion data from the Archivist and uses this data to estimate its position via dead-reckoning. This estimated position coupled with the waypoint in the mission plan produces

a next motion vector, which is sent to the Pilot. When GPS is used (i.e. during recharging and initial deployment), actual coordinates will be received and the estimated position at that time will be discarded in favor of the newly attained reference position. GPS will be periodically used to prevent unbounded error accumulation from getting to an unreasonable degree.

## Emergency Handler

The Emergency Handler is responsible for detecting unsafe states and signalling the pilot accordingly. Emergencies can fall under one of three types: Collision, Man-down, and Low battery.

Collision emergencies occur when the sub is at risk of colliding with some physical object. At this point, the sub will immediately stop if need be and take steps to ensure its own safety. This type of emergency is found from local awareness data.

Low battery emergencies occur when the sub has detected a low battery level and must resurface and recharge. This type of emergency is found from battery sensor data.

## Pilot

The Pilot is responsible for passing high level motion instructions to the Captain that factor in high priority safety requirements. Thus, it acts as more of an intermediary class between the Navigator and the Captain, but with a vital function. The Pilot will be blind to any physical implementation of the system and only give instructions pertaining to desired motion vectors.

These motion instructions can include a heading change, distance to move, a combination of these two, an emergency stop or an emergency redirect. Heading and distance changes are

routine and should happen repeatedly, but emergency overrides will only occur during situations where the wellness/safety of a sub is compromised, and is discussed in the Emergency Handler section.

## Captain

The Captain is responsible for housing logic to exhibit repeatable and modular movement, such as heading changes and linear distance changes. Internally, the Captain adjusts the motion vector instruction received by the Pilot by any deviation produced from the environment, which comes from the most recent motion data. These deviations can include small heading shifts and overall displacements from waves. Continuous motion data is also required to provide feedback for closed-loop control systems.

For the ATS Mk. 1, heading changes are performed by a PID controller. Work describing the implementation and testing of the controller can be found under *Performance estimates and results*. This section will describe how a PID controller works, and how the output of the PID is translated into motor output.

The development of a control system for an autonomous underwater vehicle with no active referencing is a complex task. The traditional method of developing a control system consists of developing a block diagram that contains all electric and mechanical subsystems, deriving mathematical equations to relate these subsystems, and combining all these equations into a single transfer function. This is a fairly trivial task for deterministic and repeatable systems, such as industrial mechanical machinery, but modelling interactions between an underwater vehicle and its environment requires knowledge in the field of fluid dynamics, which

is highly nonlinear and is dependent on the environment (temperature, dimensions, pressure) itself as well as the actuating body; hence, since the environment is non-deterministic, any derived control system is not guaranteed to work in all situations. It is also worth noting that the mechanical engineers on this project were consulted for this task and offered the following justification: In many situations, fluid dynamics problems cannot be solved in a reasonable amount of time and thus lookup tables are often used- therefore any implementation will either be computationally-demanding, if these problems are to be solved, or require lots of memory, if lookup tables are to be used, or some combination of the two (however neither is preferable for a single board computer with limited resources).

There are more complex control system design methods that may allow for more dependable performance, but these methods were not explored for three reasons. The first reason is that development and reiteration of a derived control system is a time-consuming and laborious task. Given the limited time allocated for this project, this path was viewed as a major risk that may hinder development in other fields and was thus retired. The second reason is that this project is marketed towards college/university teams and local governments, as opposed to massive organizations with well-educated researchers. This submarine is modular in both software and hardware layers, and therefore the implemented control system should be abstract enough that the client may substitute in their own control system should they feel inclined. Finally, the third reason stems from our minimum-viable product approach. Multiple test systems with different control surface layouts are used for testing movement, which means that a complex control system must be re-derived and re-implemented for every new test system, adding further development time. Therefore, a control system that is relatively easy to

implement, can be implemented abstractly, and results in decent performance is more desirable than a finely-tuned control system that takes a long while to derive and debug.

A PID controller was chosen for this application, due to the reasons listed above. PID controllers also have been implemented in non-deterministic environments before for differential-thrust drive systems with success; specifically, quadrotors are often controlled using a PID controller[37].

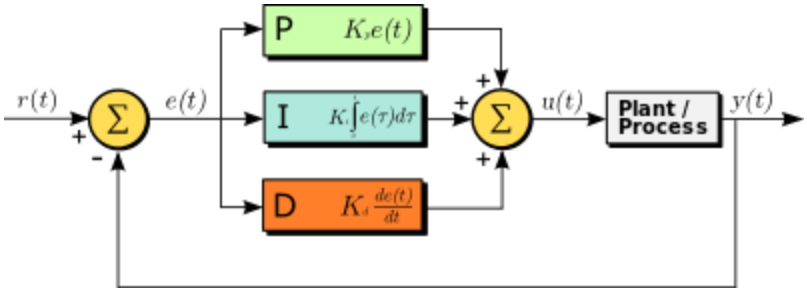



*Fig. 16. PID controller diagram*

PID controllers are relatively simple to understand and easy to implement. The output of the controller is a function of the deviation from a process variable (PV) and a desired setpoint (SP), called $e(t)$ in Fig. 16. Using weighted proportional, integrative, and derivative factors of this function, the output is computed by summing these three respective terms. The output of the PID controller, defined as $u(t)$, is expressed mathematically below.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

*Fig. 17. PID controller output equation*

In Fig. 17 above, $K_p$ represents the weight of the proportional term, $K_i$ represents the weight of the integral term, $K_d$ represents the weight of the derivative term, and $e(t)$ represents

[37] Salih, Atheer L., et al. "Flight PID controller design for a UAV quadrotor." Scientific research and essays 5.23 (2010): 3660-3667.

the error function once again, equal to $SP - PV(t)$. The proportional gain linearly scales the error function, which largely determines how reactive the controller is and how often/severely it overshoots. The integral gain sums the instantaneous error over time, and functionally accelerates motion towards the setpoint. Finally, the derivative gain is computed through the slope of the error function over time and predicts system behavior. Due to unpredictable environmental disturbances such as waves and strong winds, the derivative gain may not be useful in practice.

For our application, there are two types of heading changes; stationary and mobile. While stationary, the goal is to keep a net displacement of zero, and thus motors are spun at equal but opposite speeds. This allows the ATS to modify its heading without compromising it's position drastically. In this case, the PID output $o$ is linearly scaled to a speed change $s_c$ with a programmable PID output ceiling $o_{max}$ and maximum speed change $s_{c,max}$, so that if $o \geq o_{max}$ then $s_c = s_{c,max}$. Otherwise, $s_c = (o/o_{max}) * s_{c,max}$. This speed change is duplicated and sent to the Engine as left/right motor values, polarized according to the intended direction of rotation.

While mobile, the motors are already spinning at some speed $s$, whereas this was assumed to be 0 for the stationary case shown above. As the captain is the only module that can directly control the motors, it is safe to assume that these motors are being driven at the same speed value as the only time when they should not be is when a heading change is underway. The process for converting PID output to changes in motor speed is very similar to that for a stationary heading change: $s_c$, $s_{c,max}$, $o$, and $o_{max}$ terms are still used in the same manner as described above, but the final motor speeds are not sent to the Engine in the same manner. Once $s_c$ is computed, it is subtracted from the current speed $s$ and sent to one of the two motors in

order to slow that motor down. For example, if the sub is moving forward and needs to make a slight left turn, the speed change will be subtracted from the left motor's speed value to yaw the sub to the left. Generally, the updated motor speed can be expressed as $s = s - s_c$.

There is the case where $s_c \geq s$; that is, the speed change is greater than the current motor speed. In this case, $s - s_c$ will either stop the motor or cause it to spin in the opposite direction. This is undesirable in real-time navigation, as although logically this is a speed difference of only a small percentage, spinning motors in opposite directions will cause the rate of heading change to increase very quickly and make stability harder to achieve. Therefore, if $s_c \geq s$, the motor to be slowed down will change its speed to 1%, and the other motor will change its speed to the value of the speed change + 1%. This ensures that both motors keep spinning in the same direction and that the difference between the motor speeds is always equal to $s_c$.

```
/* For yawing to the left */
if PID output >= max PID output; then
    speed change = max speed change
else; then
    speed change = (PID output/max PID output)*max speed change

if  speed > speed change; then
    left motor speed =  speed - speed change
    right motor speed =  speed
else; then
    left motor speed = 1
    right motor speed = speed change + 1
```

*Fig. 18. Pseudocode for PID output to motor change logic*

Examples have been given for forward motion, but the same logic is implemented for backward motion. This can be useful if a sub needs to backtrack after reaching a dead end in a convoluted environment.

A final implementation detail to mention is the translation of heading values. Heading, commonly read on a compass, ranges from 0 degrees (inclusive) to 360 degrees (exclusive). Consider the situation in which the target heading $h_t$ is equal to 355 degrees and the current heading $h$ is equal to 2 degrees. In this instance, the PID will see a subtraction of 353 degrees as the only way to get to $h_t$; however, yawing in the other direction would only require a 7 degree heading change. Another example if $h_t = 220$ degrees and $h = 40$ degrees. In this case, the heading difference is 180 degrees whether the sub moves clockwise or counter-clockwise. This caps the maximum necessary turn radius to 180 degrees, which is both power- and time-efficient. Therefore, the following logic is implemented: If the initial heading $h_0$ is within 180 degrees of $h_t$, do nothing. If $h_t - h_0 > 180$ degrees, then $h_t = h_t - 360$ degrees. If $h_0 - h_t < 180$ degrees, then $h_t = h_t + 360$ degrees. While the target heading isn't between 0 and 360 anymore, the controller will now see the shorter path between $h_t$ and $h_0$. Proper framing is performed to ensure that $h$ is always within the proper reference window for the PID controller.

## Engine

The Engine is responsible for driving all motors and control surfaces. This is layout-dependent, however motor drivers and servo motors alike will use PWM as a modulation scheme so a uniform interface for controlling these actuators will be implemented, called a PWM translator.

The interface for driving a motor is a single integer that corresponds to a speed and direction. Specifically, this integer is in the range of [-100, 100]. Sending a value of +100 spins the motor at full speed in one direction (corresponding to forward motion), sending a value of 0

stops the motor, and sending a value of -100 spins the motor at full speed in the other direction (backward). The ATS Mk. 1 has only two motors, so spinning clockwise (for example) simply consists of sending a positive left motor value and a negative right motor value, and vice-versa for counter-clockwise.

In implementation, the Electronic Speed Controllers (ESCs) used here are bidirectional and map out the range of motor values to the standard range of duty cycles for servo motors and ESCs. This duty cycle range is 5% - 10% of a 50 Hz signal, meaning that the pulse width ranges from 1ms to 2ms. Therefore a motor value of -100 corresponds to a duty cycle of 5%, 0 to 7.5%, 50 to 8.75%, and so on.

With the assistance of the mechanical engineering team, a tachometer was used to ensure a common minimum speed (i.e. speed = 1 or -1) across all motors. This motor value was hardcoded into the PWM translator to ensure that at base speeds, motors spin at the same rate. Undesirable outcomes that this prevented include passive drift over time when movement is supposed to be linear.

# Design Changes

Over the course of this project several aspects have been refactored to more realistically represent what would be feasible to complete in the scope of this research. A significant change in the original planned functionality stemmed from the limitations discovered in the functionality of dead reckoning based on IMUs within our price range. As a result of this information, the planned method of navigation shifted to one that does not require knowledge about precise

location. However this section of development has yet to be reached, so this has yet to affect any implementation.

# Current Design and Implementation

## Movement (performed by Jacob)

The primary change between our preliminary design and our current design in terms of movement is the implementation of a hardware PWM board over a software PWM library. As can be found below in the Performance estimates and results section, Movement subsection, utilizing a software PWM library results in pulse jitter in the microsecond range. Brief testing with a hardware PWM board shows pulse jitter in the tens of nanosecond range, effectively 1/100th of the inaccuracy.

This board was implemented for a few reasons: firstly, the price of high resolution PWM breakout boards is very low due to hobbyist demand. Second, due to the limited speed resolution of the Mk. I, pulse jitter over time may cause errors in navigation, which is especially problematic since the navigation approach has shifted to reaction-based as opposed to point-to-point. The implementation of a more stable PWM generator retires this potential risk. Finally, the resolution for a single PWM channel is higher for the hardware PWM board over the software library, allowing for more precise control.

# Dead Reckoning (performed by Xavier)

As previously noted, generating accurate displacement values from an IMU is very difficult due to the fact that any source of error is exponentially increased during the calculation. Not only does this mean that accuracy in determining how far the AUV has moved if not processed correctly, but it could also falsely generate information saying it has moved when it has been effectively stationary. With the selected BNO055 IMU sensor, these are the steps which have been implemented in an attempt to get usable displacement values.

To have a starting place for this process, information about the measured values from the IMU needed to be collected. To make sure these values were as close to possible as what would be measured in one of the AUV iterations, the internal skeleton of the AFμS MK1, was 3D printed and all tests were performed with the IMU mounted on the center axis. This had the added benefit of encapsulating the Raspberry Pi, which became useful further down the line in testing. The skeleton was moved by hand in a variety of directions at a rate predicted to fall within the range of likely motion. As the goal of this test was to get a broad indication of the values that will be experienced, precise motion was not necessary.

A fourier transform of the acceleration data was taken, which shows the frequencies experienced along with their amplitudes.

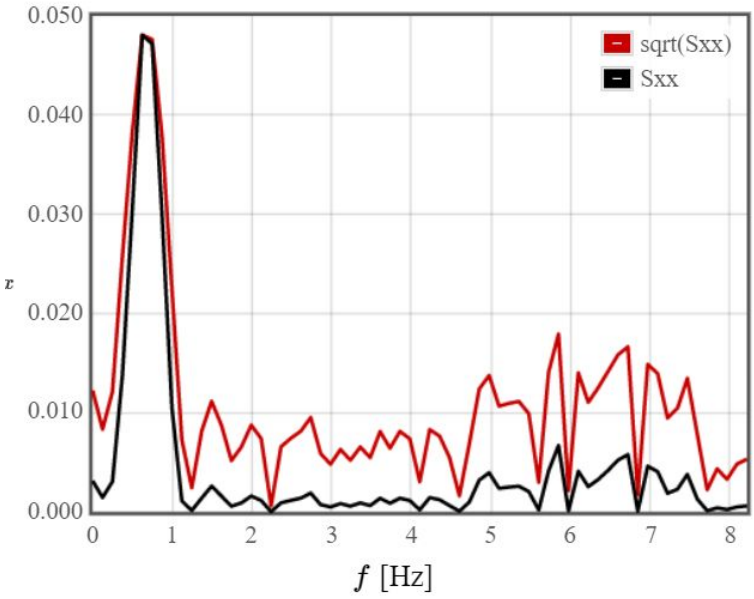



*Fig. 19. Fourier transform of predicted motion*

As can be seen in Fig. 19 above, the purposefully generated motion falls in the range of 0-~2Hz, while the lower amplitude data falls into higher frequencies.

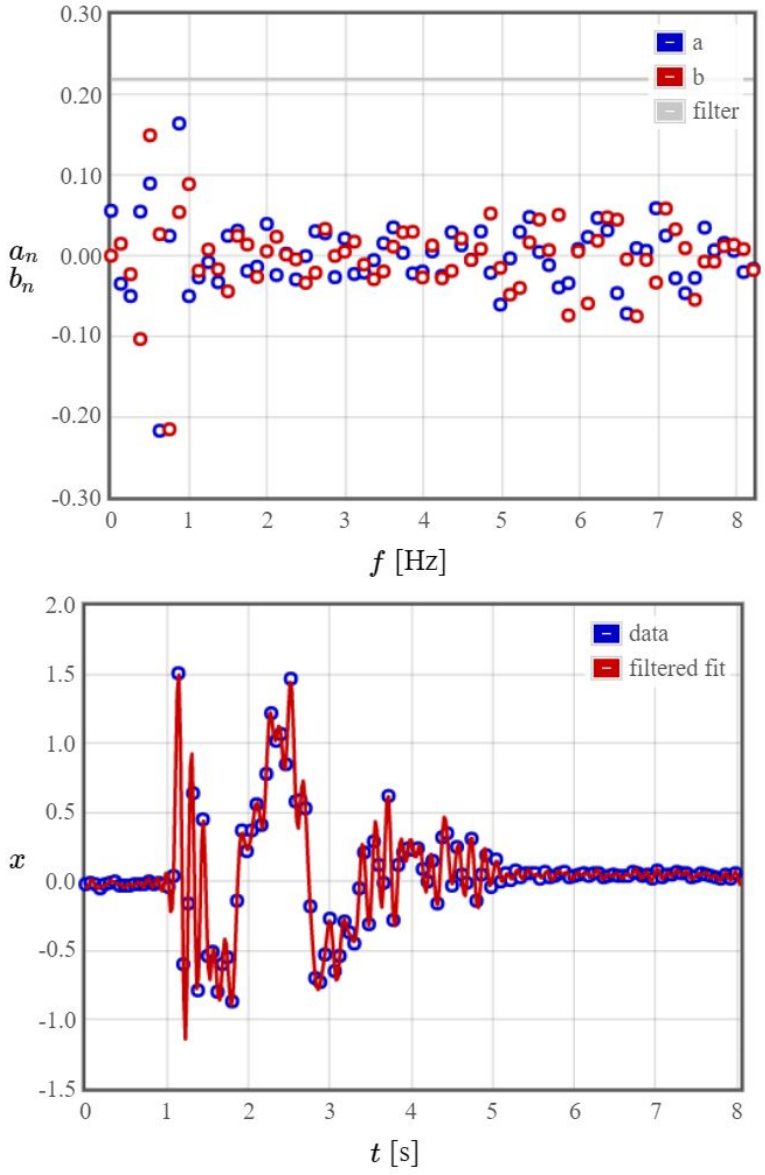



*Fig. 20. Unfiltered predicted motion data*

Fig. 17 shows the collected data run through a filter with no parameters. Knowing the frequency range of collected motion from the initial fourier transform, the data was passed through this function again with a lowpass filter of 1.5Hz.

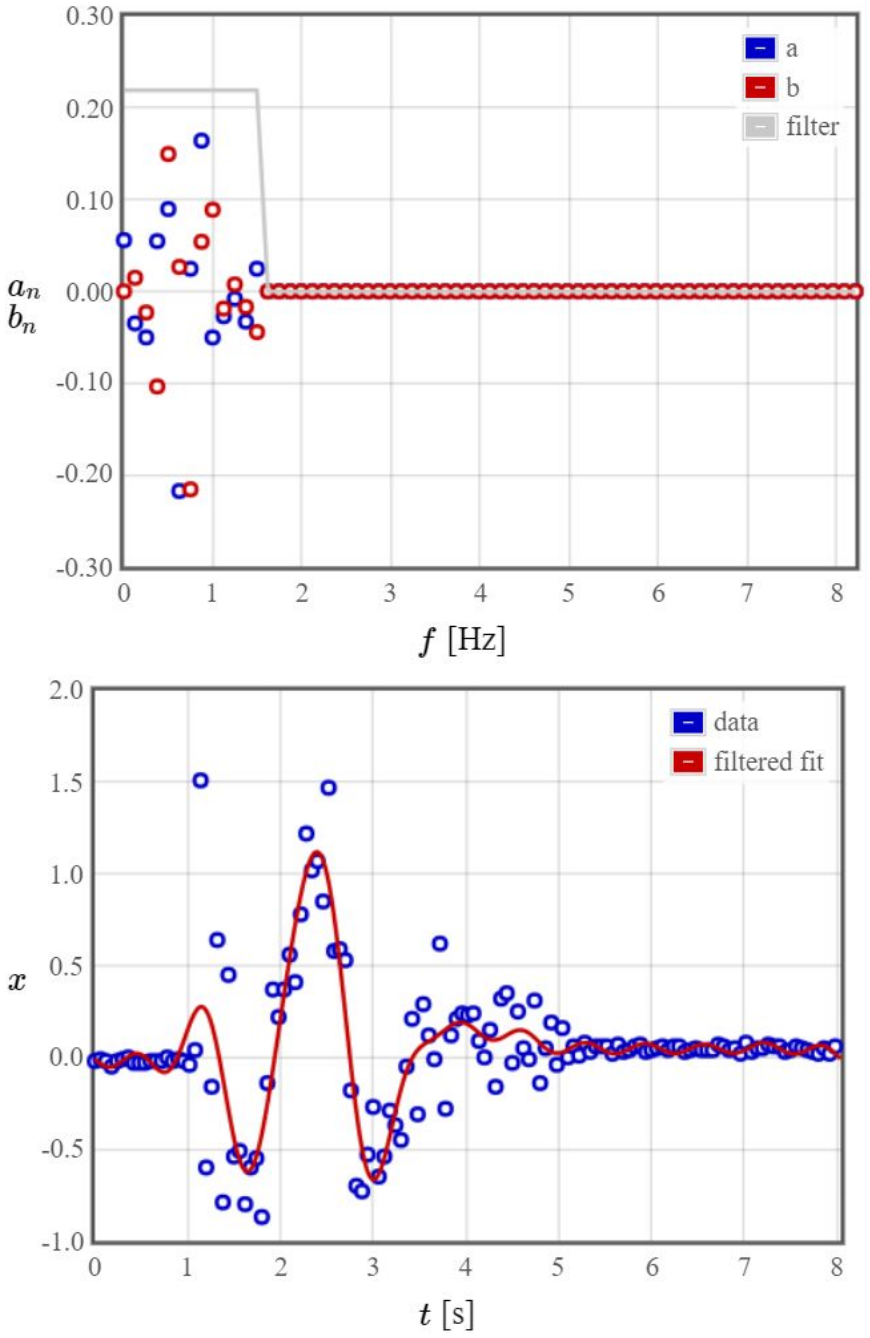



*Fig. 21. Filtered predicted motion data*

Fig. 21 shows the same data with the filter applied to it. A comparison of this to figure 20 shows a significantly cleaner signal, which is at least visually indicative of expected motion.

While a cutoff value of 1.5 certainly produces better data than no filter, this value will be tuned as more data is collected.

The results of this test were implemented into filter functions using the scipy.filtfilt()[38] function, which results in a phase offset of 0.

The next step was to verify that the IMU was reading acceleration data correctly, or at least a range approaching correctly. To do this, a simple drop test was performed, and the acceleration data was recorded.

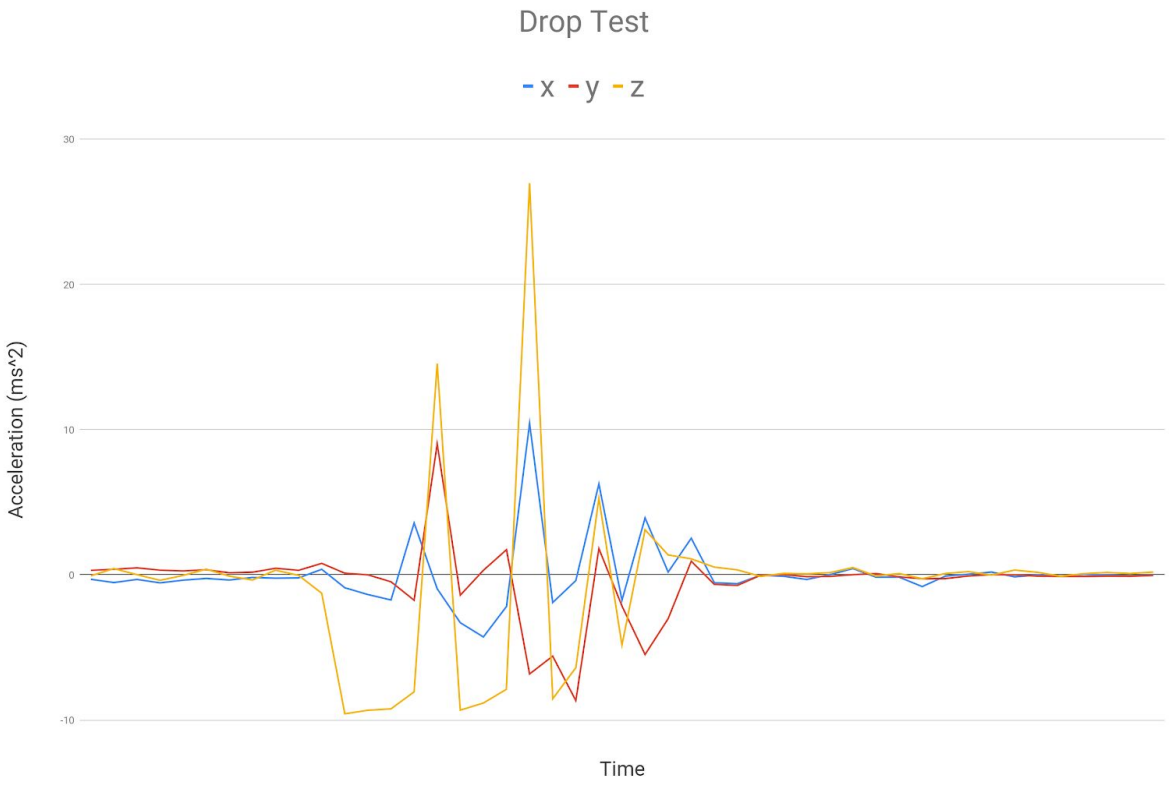



*Fig. 22. Drop test acceleration data*

[38] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html

The peak negative value shown in Fig. 19 is -9.57 $m/s^2$, which is close enough to earth's gravity of 9.8 $m/s^2$ that the discrepancy can be explained by the orientation not remaining completely flat during the fall. Now that the values read from the IMU are verified to fall within reasonable bounds of reality for large measurements, edge case information needs to be gathered. This test entails gathering stationary acceleration data to see what affect the application of the lowpass filter has on the background acceleration noise across a range of cutoffs. While stationary noise will be easy to filter out using a threshold, this noise will be present in all measurements, and therefore analysing it in isolated conditions is important for the overall design of the filter system.

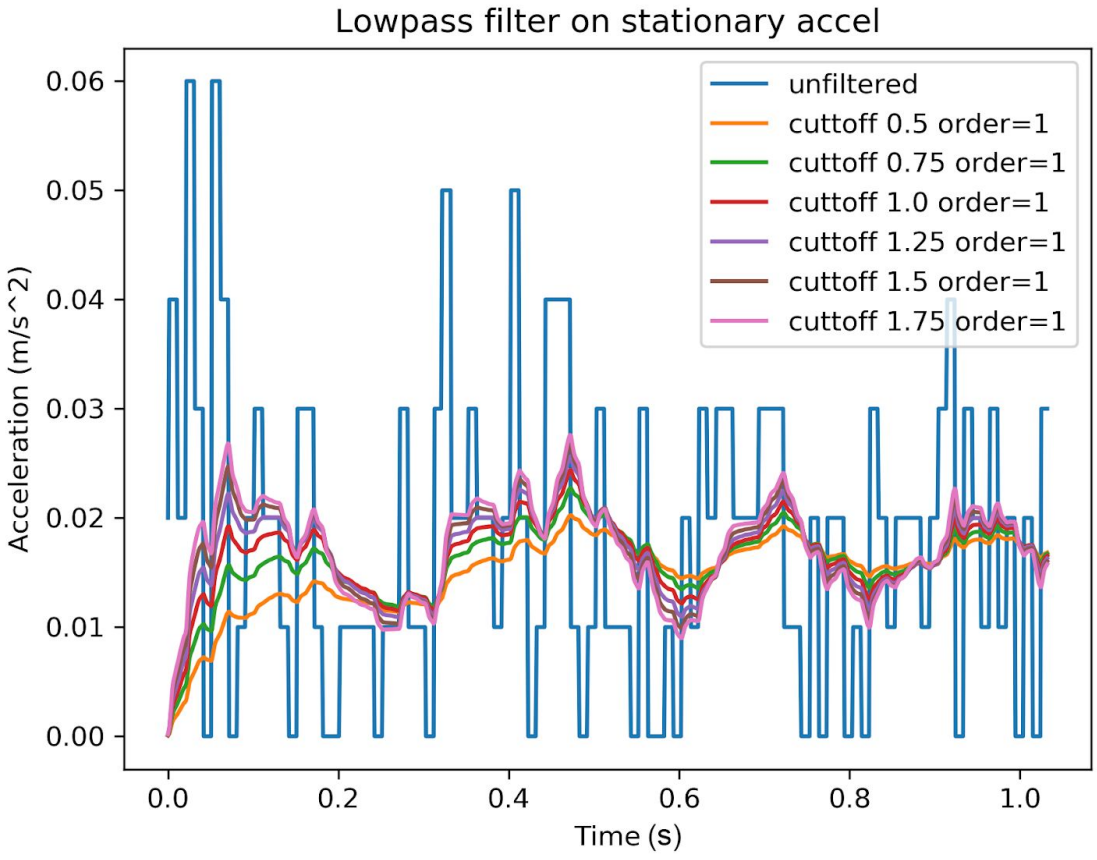


*Fig. 23. Stationary Acceleration data with a range of filters applied, order 1*

Though there is variance in the results shown in Fig. 20, none of it is drastic. This test was repeated for a higher order, and a range of orders.

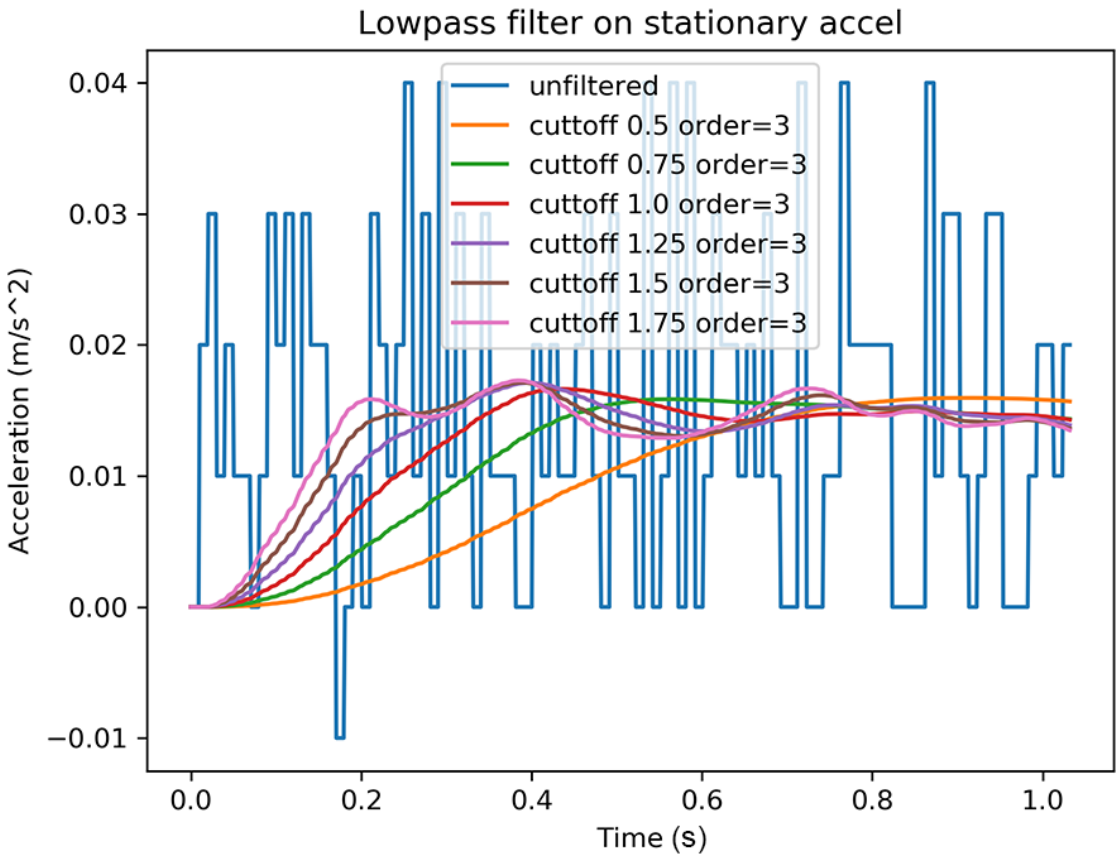



*Fig. 24. Stationary Acceleration data with a range of filters applied, order 3*

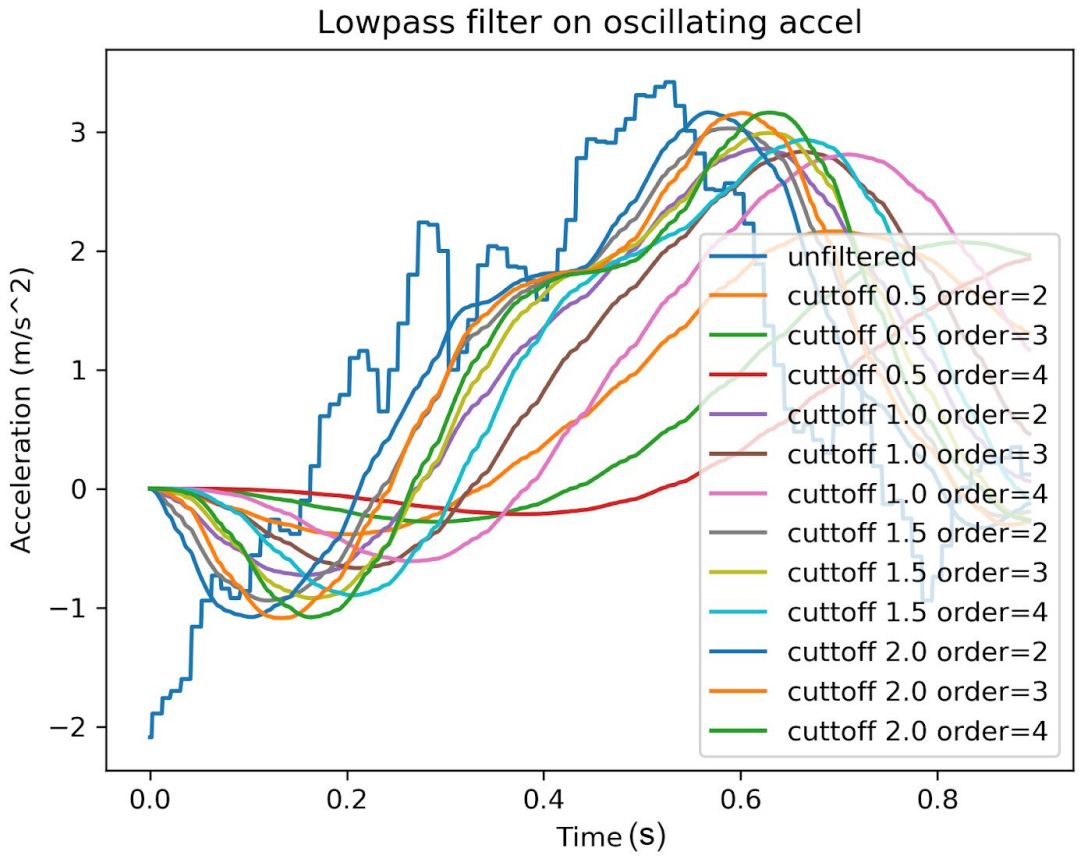



*Fig. 25. Acceleration data with a range of filters applied, under motion*

From the data shown in Figs. 23 through 25, the relationship between order, delay, and signal smoothness comes through. In Fig. 25, the acceleration data comes from the system under motion, which brings to light the phase offset resultant of higher order filters.

With this information gathered, the next step was to develop the code to collect and generate this data in the manner that will be used in the AUV itself. As the output of the IMU is instantaneous acceleration, and because integration over a single value is not possible, the calculation of displacement has to be handled in chunks. Hypothetically, integration could be performed over the entire range of acceleration every time there is a new value added, though

this would be very computationally inefficient and result in unneeded time resolution. The final system is as described in the design section under *Motion Data*.

To test this system, simulated sinusoidal acceleration values were fed into Motion Data as though they were coming from the IMU handler.

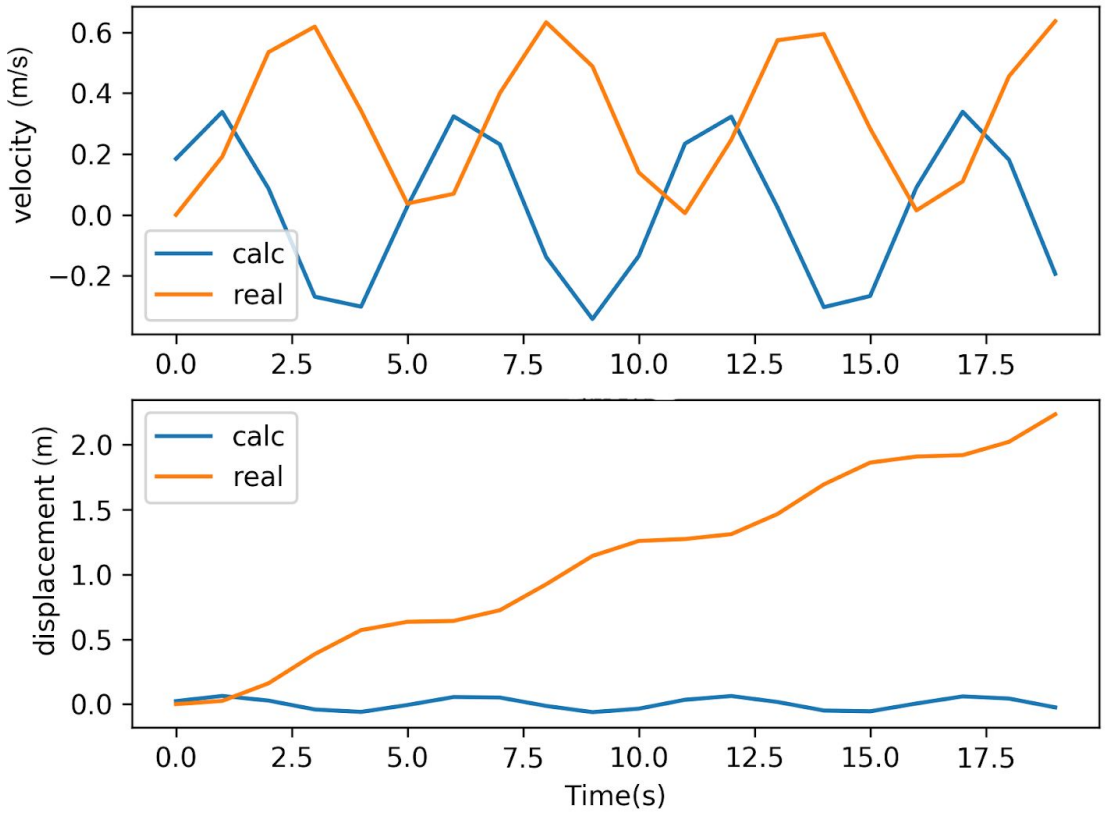



*Fig. 26 Velocity and displacement from simulated acceleration data*

As can clearly be seen in Fig. 26, the first results of this system were incorrect. After significant debugging and comparison of results with the "real" values, a much improved output was achieved, which can be seen below in Fig. 27.

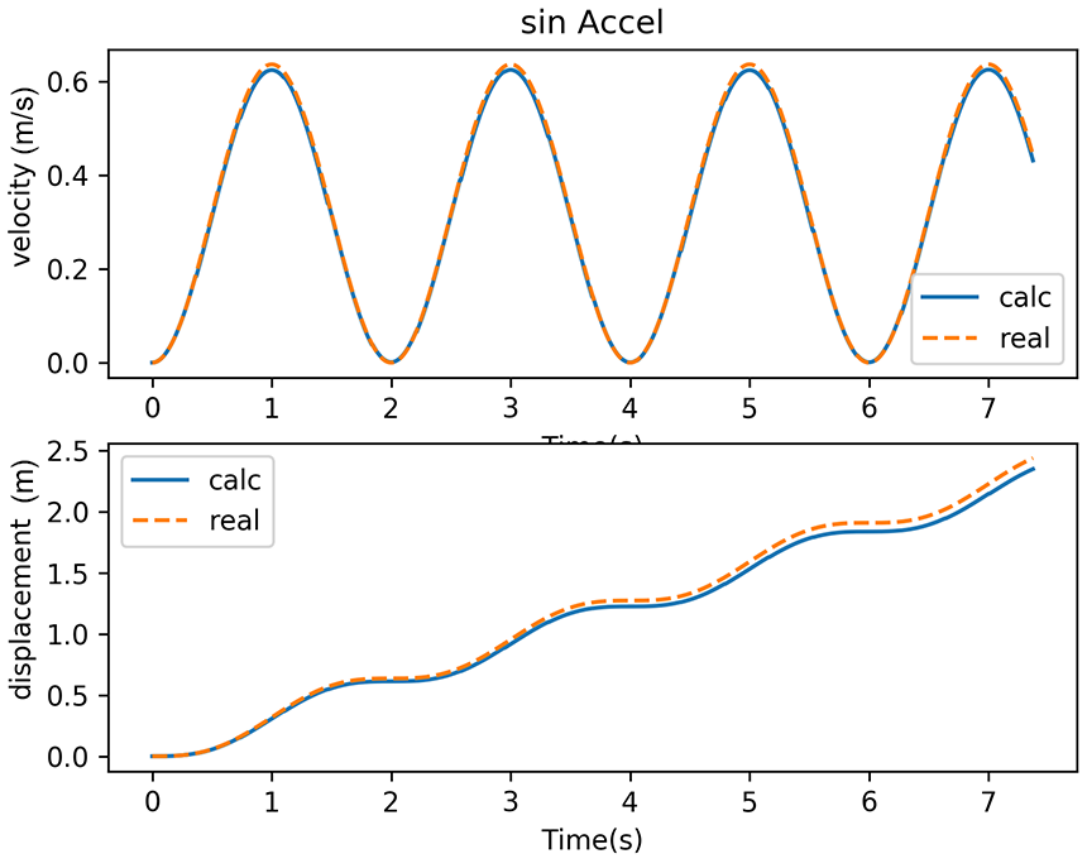



*Fig. 27. Velocity and displacement from simulated acceleration data*

This initially seemed correct, but close inspection shows that the values stray further from each other as time progresses. The source of this error was difficult to track down, but it turned out to be because acceleration needs to be passed along with the immediately most recent value to have velocity align with the previous calculation.

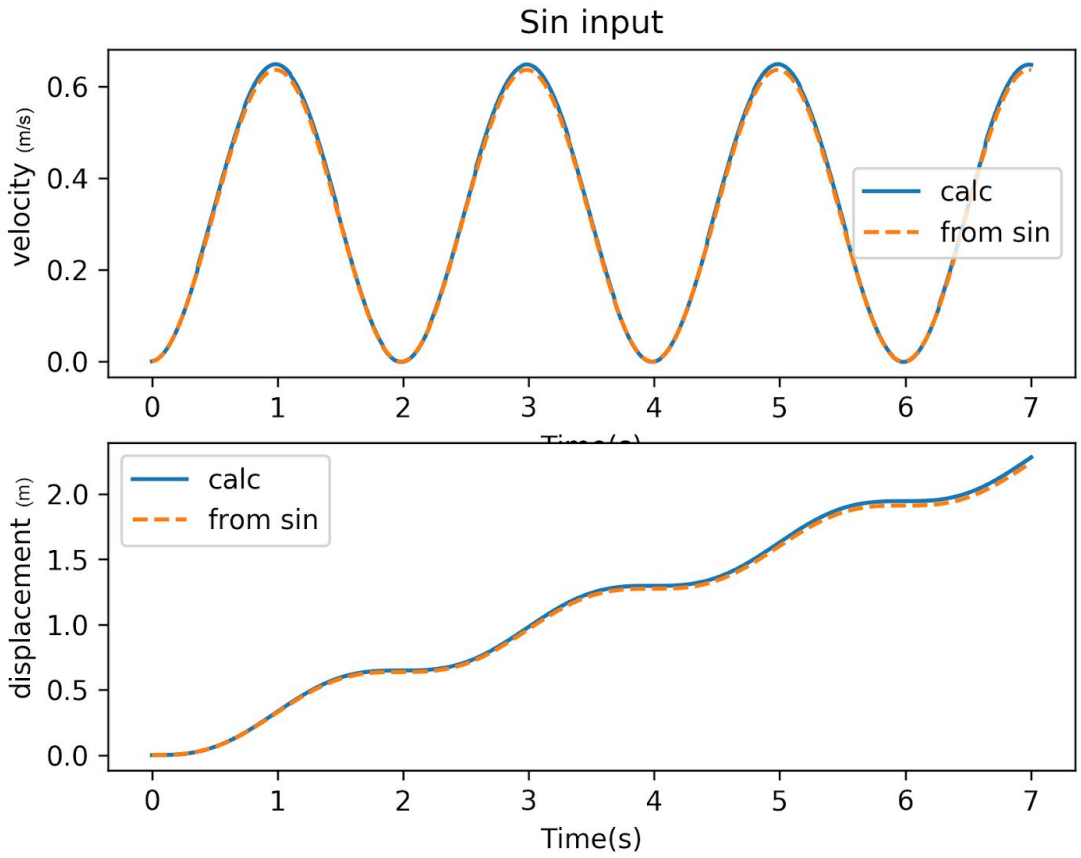



*Fig. 28. Velocity and displacement from simulated acceleration data*

Though the calculated values in Fig. 28 do not exactly match the base values, further inspection showed that this discrepancy is due to the difference in integration methods used for the integration calculation.

Now that the algorithm was confirmed to be functioning correctly, it was time to work with real data. The simplest way to gather data was to move the system by hand while calculating the input, then graphing out the resultant displacement to compare with the

performed motion. As this data is coming from the IMU, it is in 3 dimensions, therefore the results of these tests are in 3D graphs.

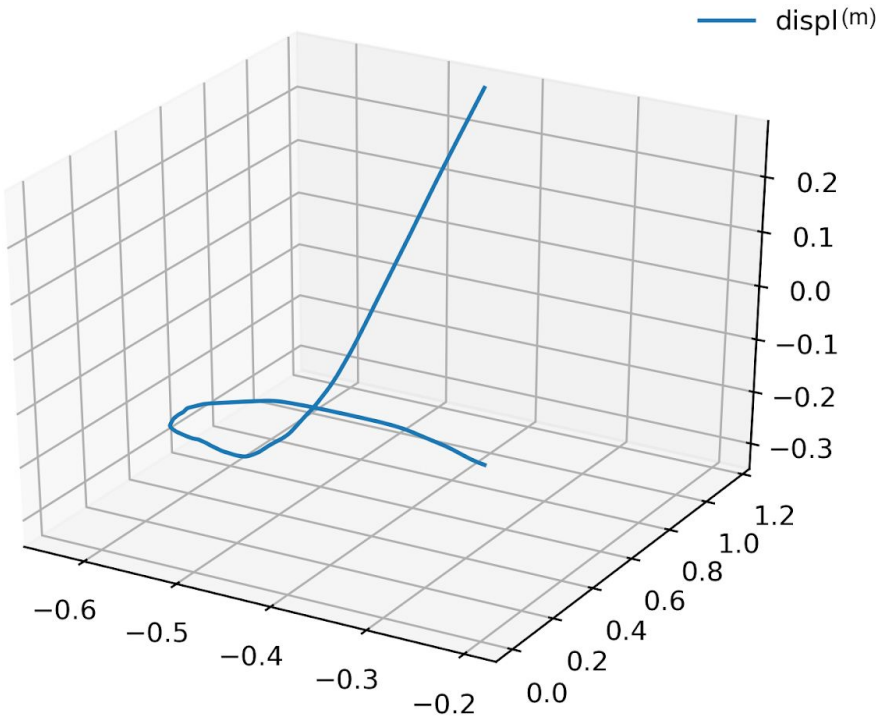



*Fig. 29. Displacement XYZ (cm), vertical motion test*

The test displayed in Fig. 29 is the resulting data from directly moving the system upwards and then terminating the test. Though this does display direction indicative of this motion, observation of the axis shows that the displayed figure is misleading, and little to no aspect of the real motion was recorded. Due to limitations of the 3D graphing suite, mainly the inability to have defined axis dimensions, pursuit of this presentation method was rapidly discontinued,

though after going through several hand controlled motions, it was quickly determined that significant work on the filter would be required.

To evaluate how well the filter is functioning in order to improve it, repeatable known motion of the system was required. To enable this, the mechanical engineering team was asked to design and build a test rig, which would hold and spin the IMU+Raspberry Pi system smoothly, controlled by a continuous rotation servo.

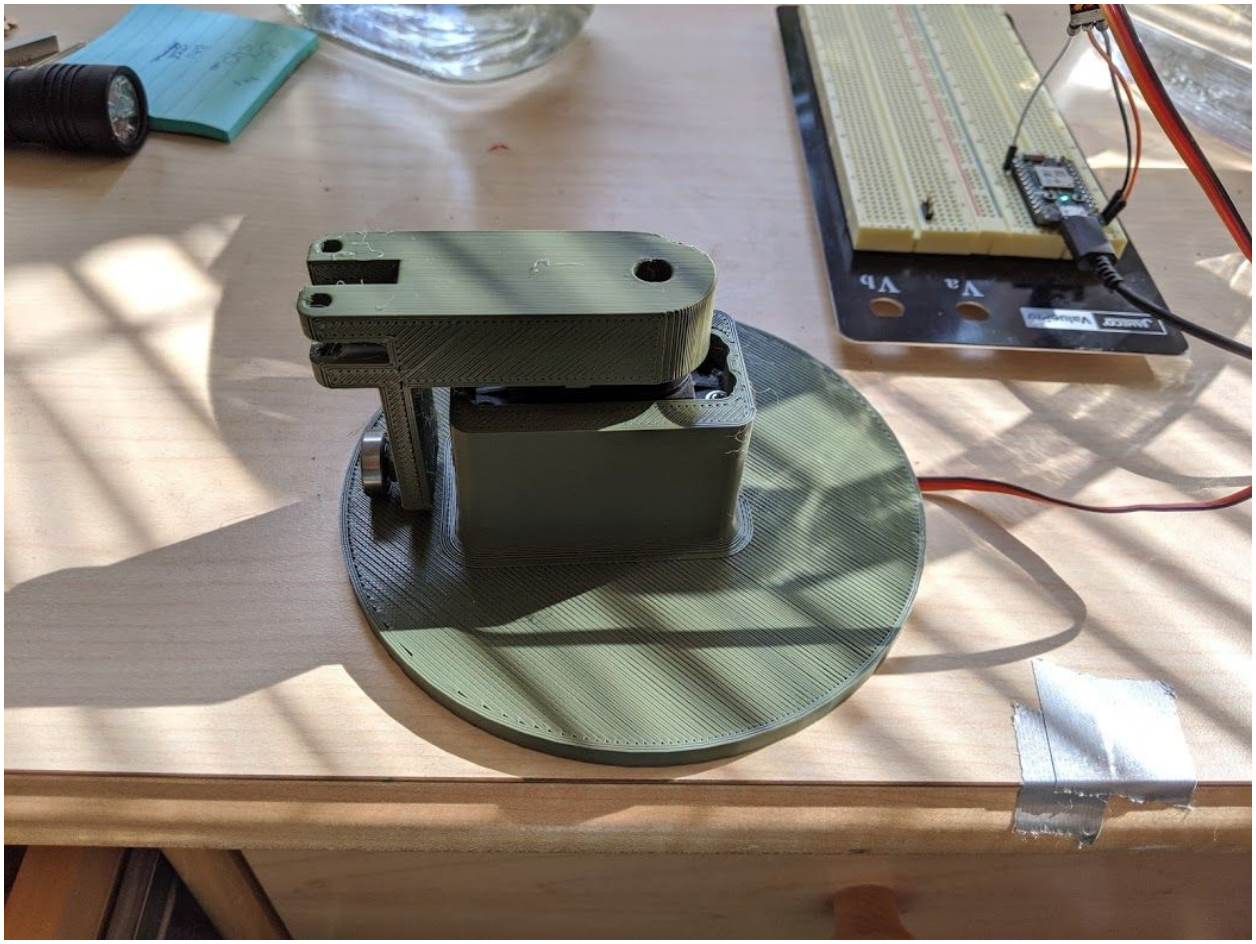


*Fig. 30. Motion generating test rig*

With the continuous rotation servo set at a known PWM value, the RPMs were measured, and from this the tangential velocity of 0.713m/s was calculated. From $A_n = \frac{Vt}{r}$ a normal acceleration of 5.09 $m/s^2$ was determined. This is displayed in the IMUs measurements as well.

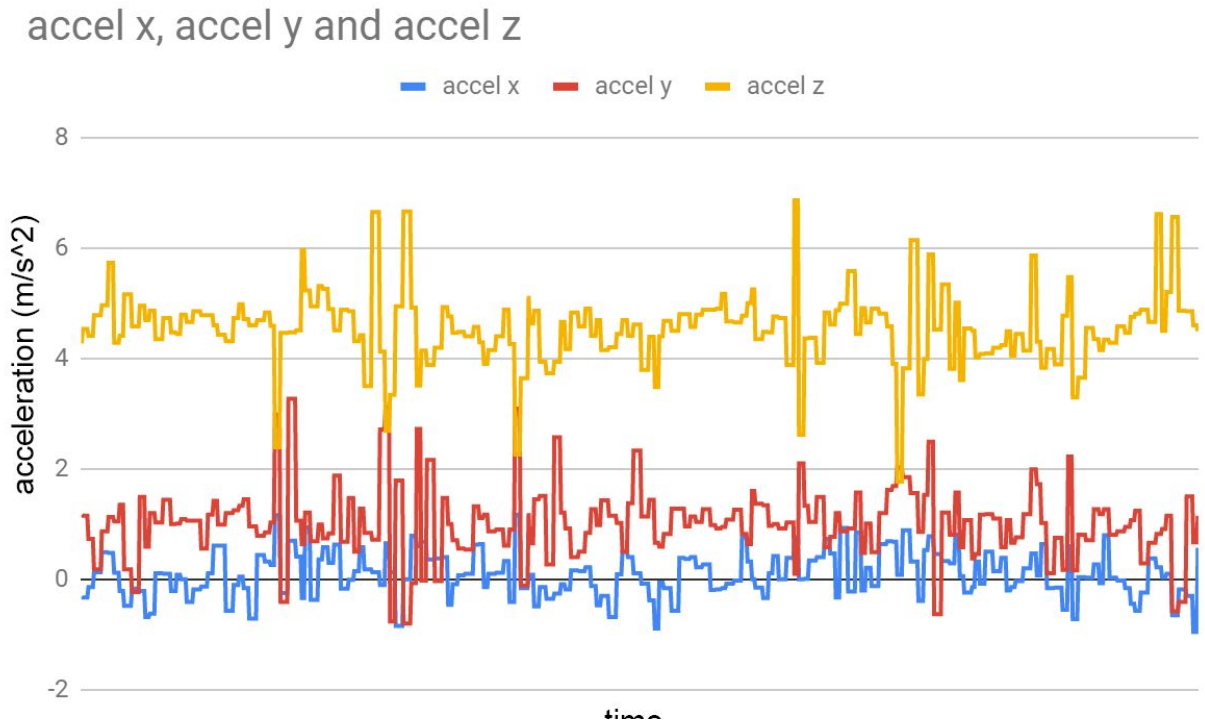



*Fig. 31. Spin test acceleration values*

Now that there was a method for generating values to compare against which included the IMUs errors, the next step was to use this comparison to optimize the filter parameters. It was decided to use regression algorithms to hone in on the ideal parameters. The way this works is the regression algorithm is provided with a function to pass parameters to, and which will return a score. The algorithm attempts to find the input parameters which produce the lowest returned score from within a designated range. The function that this algorithm is given initializes Motion Data with the parameters it has been given, and gathers data for a set period of time while under

motion in the test rig. Once data has been collected for said period of time, it sums the RMSE[39] score over each value in the gathered and real datasets. This summation is then returned to the regression function. The parameters that this regression function is tuning are range, cutoff frequency, and filter order. The range that the algorithm is allowed to vary over was informed by the initial filter testing, and narrowed down over time based on the results of previous regression runs.

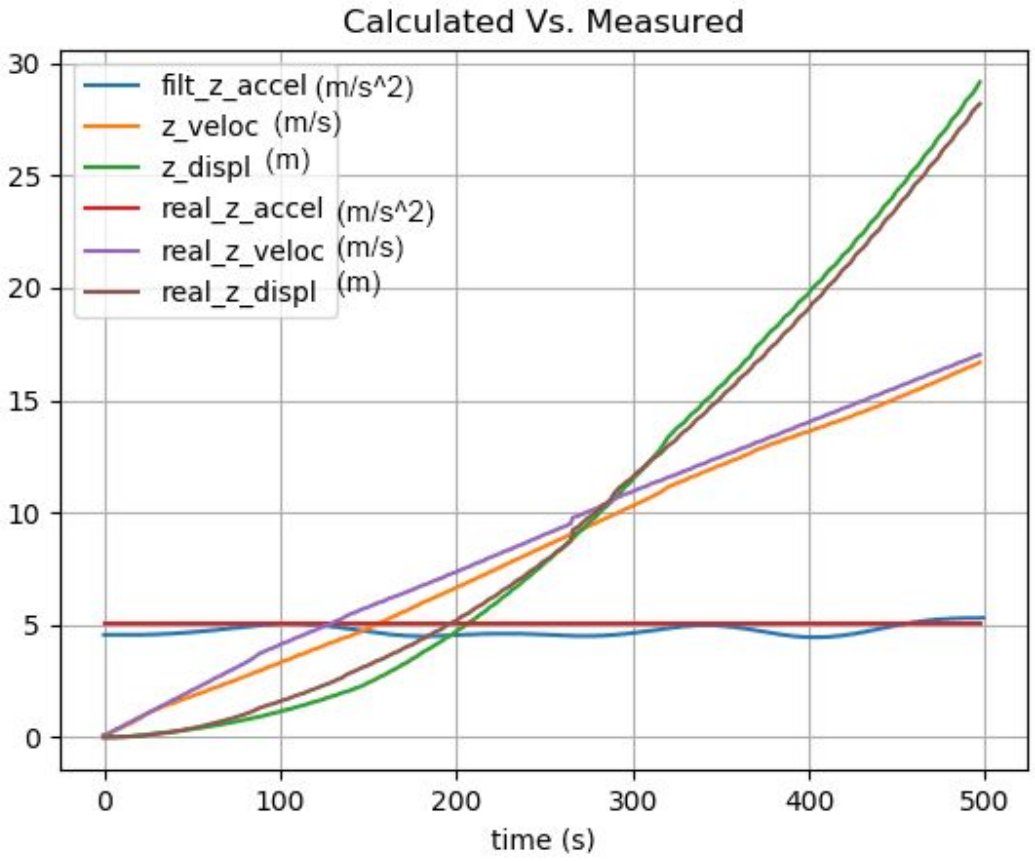



*Fig. 32. Spin regression results*

[39] https://www.statisticshowto.datasciencecentral.com/rmse/

Fig. 32 shows the gathered data, run through a filter with the regressed parameters, graphed on top of the real data. These results appear very promising, however this is for linear acceleration, which is significantly simpler than nonlinear.

The test rig was next modified to generate a normal acceleration, which upon integration would produce $sin(t) + 1$ . This is analogous to velocity, though because it is generated solely from the normal component of acceleration, it is not a real velocity, even though the IMU will not be able to differentiate. This function will be referred to as velocity to simplify matters, and from it, virtual displacement is calculated, which will just be referred to as displacement.

"Real" values for each of these were generated, and can be seen in the following three Figs..

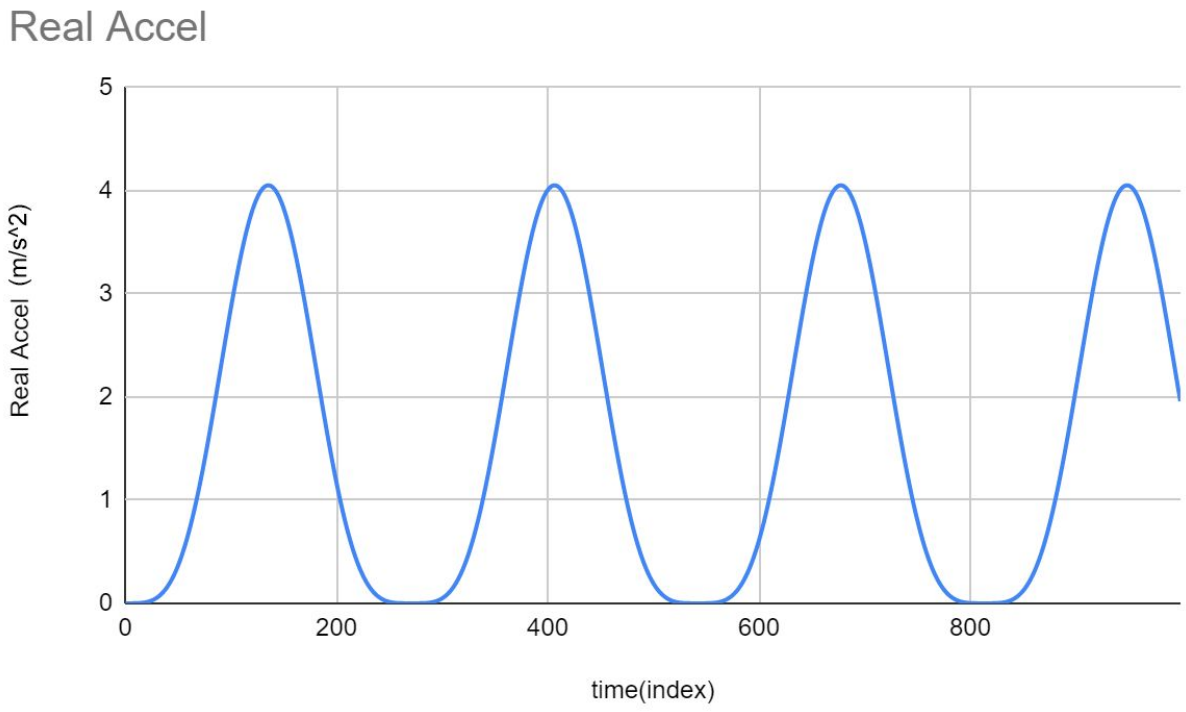



*Fig. 33. Calculated acceleration value for scoring*

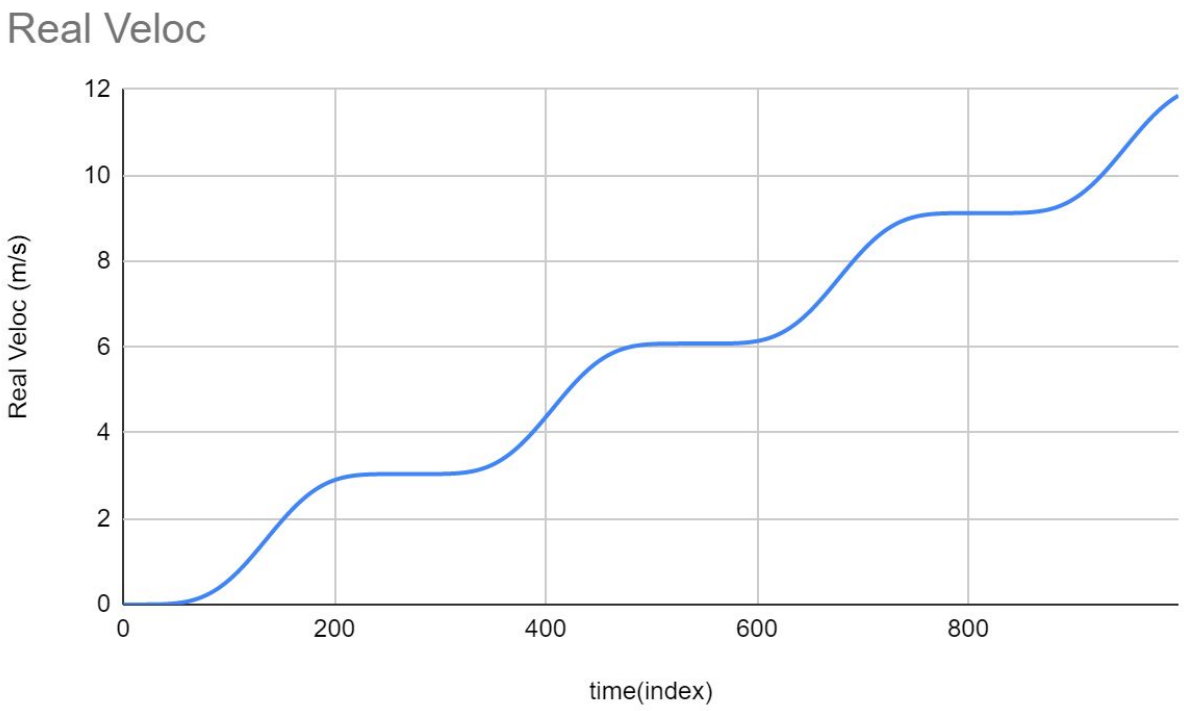



*Fig. 34. Calculated velocity value for scoring*

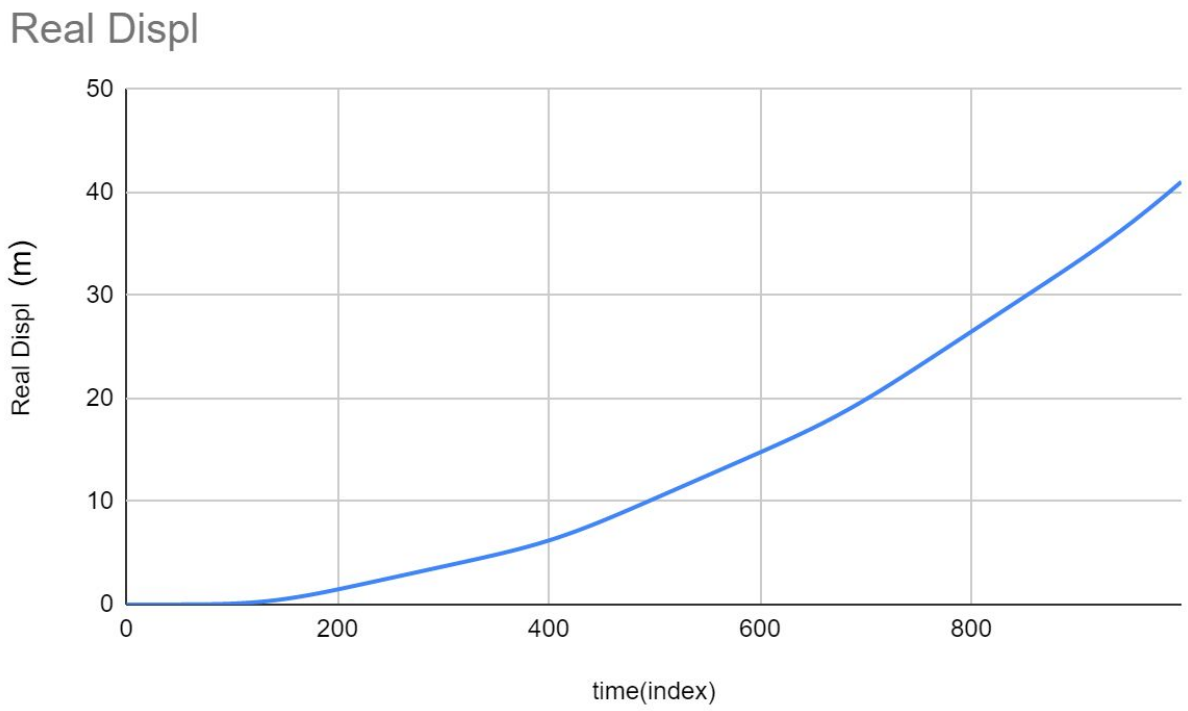



*Fig. 35. Calculated displacement value for scoring*

Using the filter parameters provided from the linear acceleration regression, an initial graph of measured and filtered acceleration was generated.

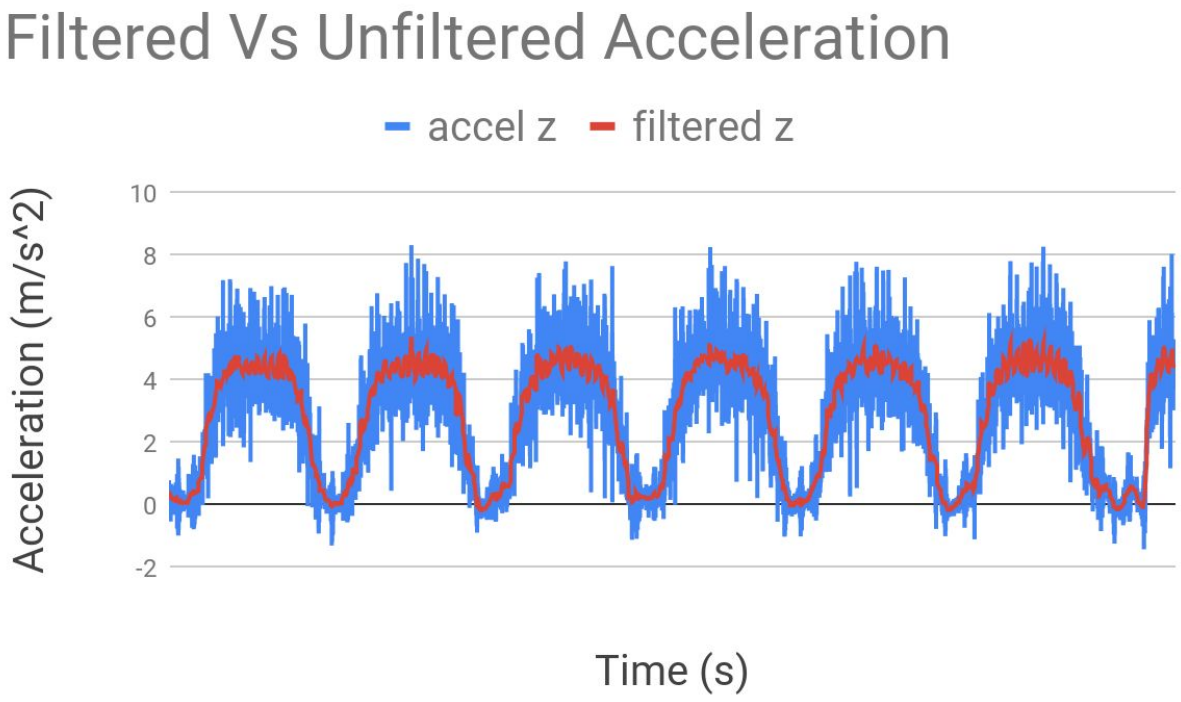



*Fig. 36. Nonlinear spin acceleration*

While the filtered acceleration in Fig. 36 is certainly not an ideal representation of what the acceleration data should look like, it is a lot better than the unfiltered values.

The regression code was modified to generate a graph for each epoch, and code was added to start the nonlinear spinning at a specific time to synchronize the measured values with the starting point of the calculated values. The graphs allow for human analysis of the results of each test, regardless of the score, and insights made from this were used to tune the range of parameters the algorithm optimized over. For this human analysis it was much easier to compare

values using index as the x axis, so each x value corresponds to 0.007s. Initially results appeared promising.

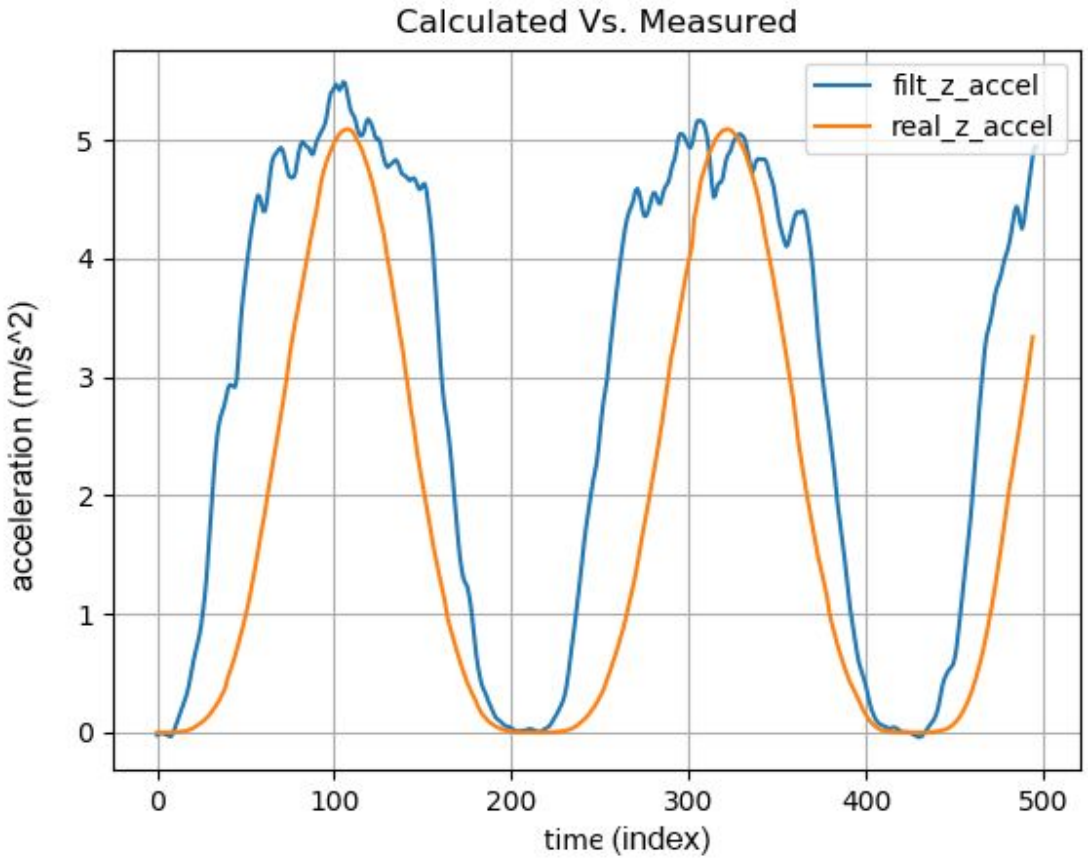



*Fig. 37. Calculated Vs. measured & filtered acceleration*

However, as testing progressed, issues with synchronization began to show up.

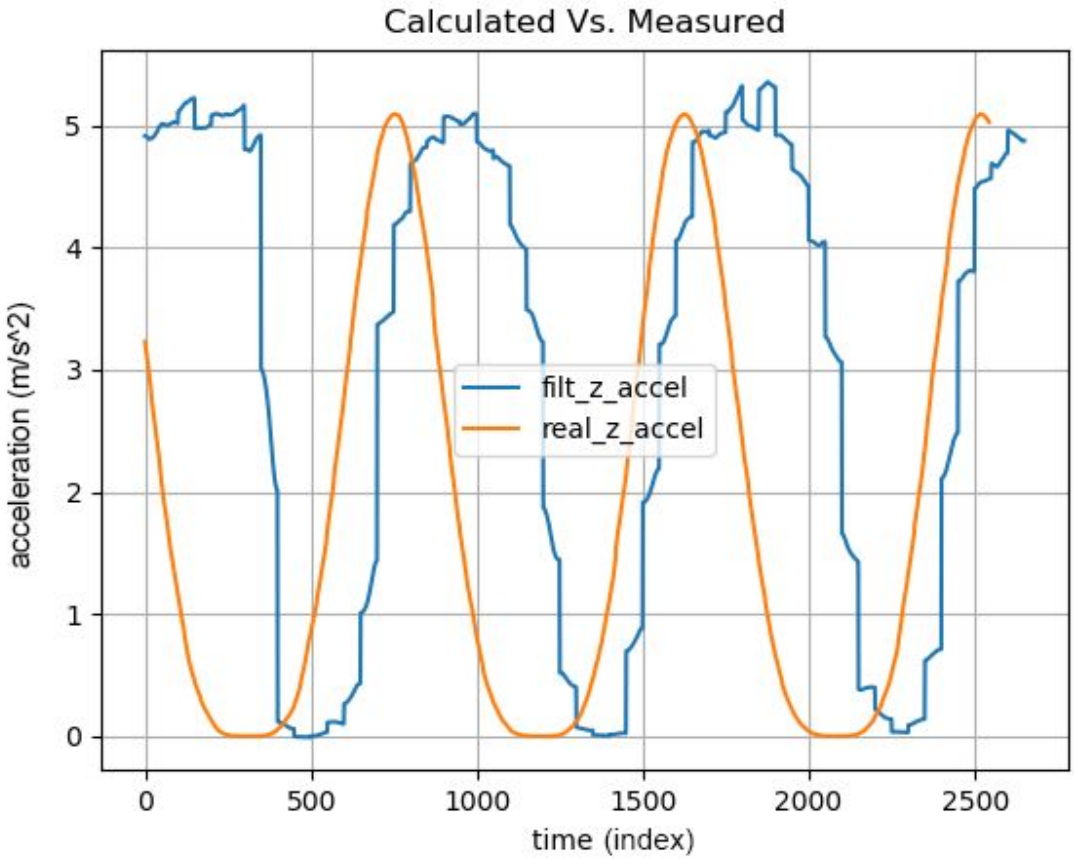



*Fig. 38. Calculated and measured acceleration, synchronization issue*

While the synchronization distance in Fig. 38 may not seem particularly large, due to the scoring method used, slight differences compound greatly. Further testing and investigation lead to the conclusion that a lagging network connection was causing discrepancies in the start time of the test rig.

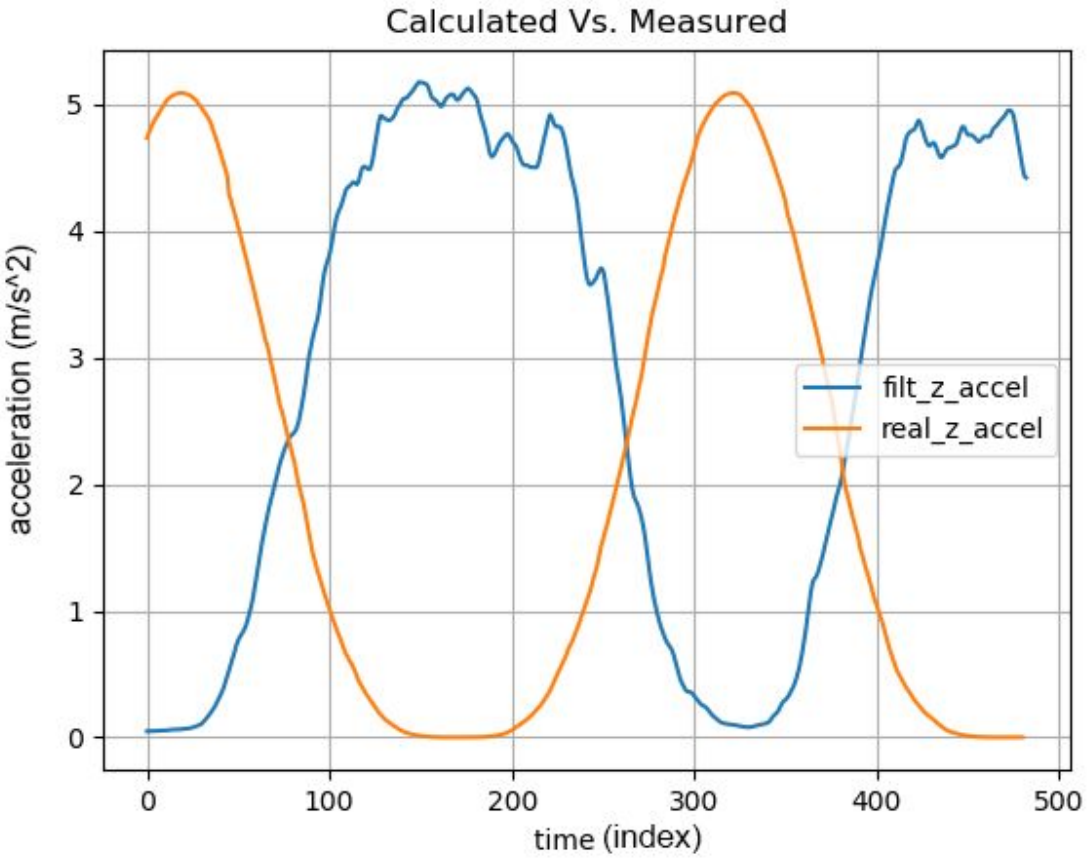



*Fig. 39. Calculated and measured acceleration, significant synchronization issue*

Said lag was sometimes great enough to put the signals completely out of phase with each other, as can be seen in Fig. 39. This produces a horrible score, but as the algorithm is only aware of its parameters, it ruins the regression. As the start signal requires the connection to an external server, and direct interfacing between the Pi and servo controller is impossible due to the Pis spinning, there was no way to concretely eliminate the lag itself.

The solution found for this issue was to make the servo function continuous, so there was no initialization signal, and have some onboard method of synchronizing the signals after they are collected, but before they are scored.

To do this, a function to detect the valleys in the filtered acceleration was implemented, and used to sync to the valleys of the real acceleration.

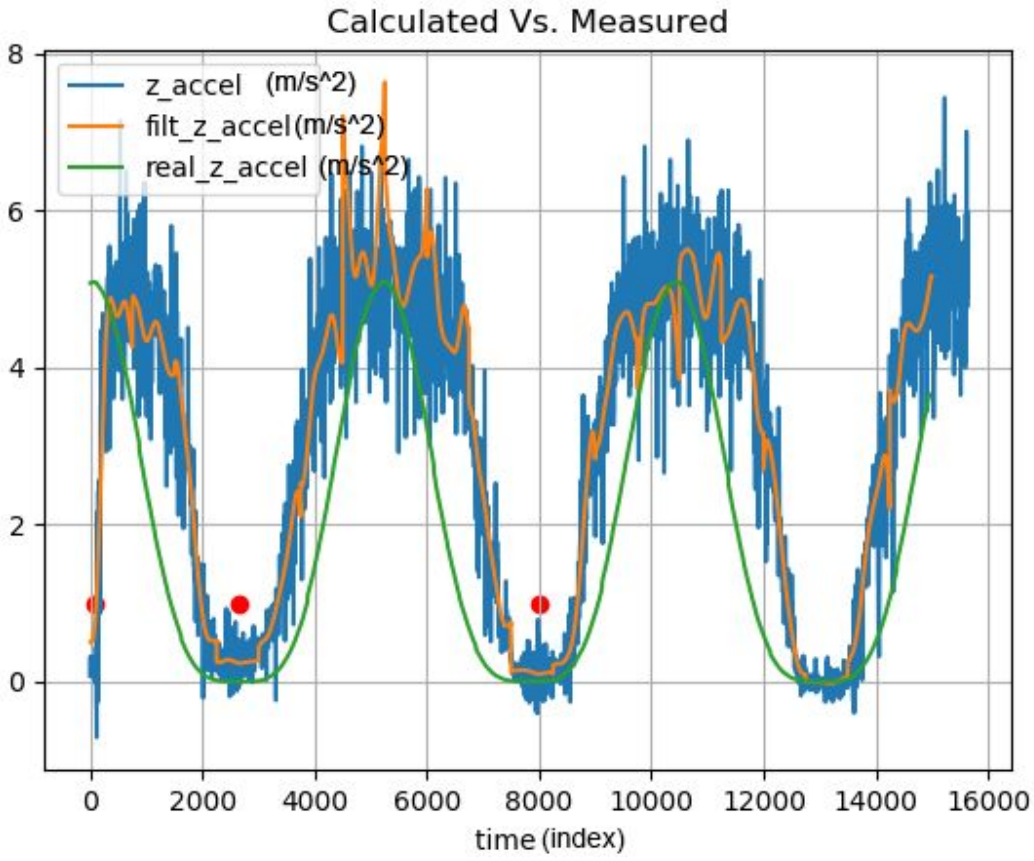



*Fig. 40. Calculated, measured and filtered acceleration, valley based synchronization*

The red dots in Fig. 40 represent the location of the located valleys, and it can be seen that the real acceleration signal is adequately synchronized with the filtered one. This figure also displays large spikes in the filtered data, which signifies that a cutoff value of 1.5 is too high, allowing

insight which can be applied to the parameter ranges. Unfortunately, the valley detection algorithm was not as robust as desired, and resulted in situations where no valleys were detected for no discernable reason.

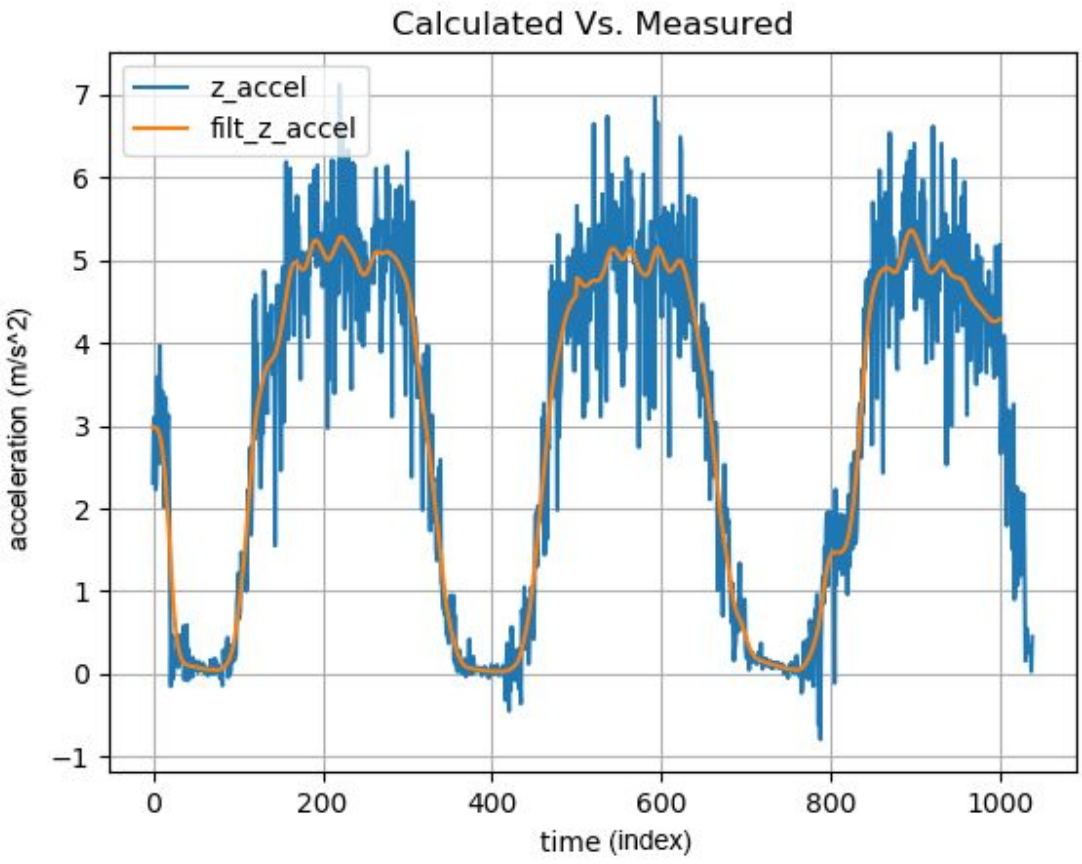



*Fig. 41. Calculated and measured acceleration, valley detection issue*

No valleys were detected in the filtered acceleration signal shown in Fig. 41, even though the data appears to be clean enough for the detection to work flawlessly. This error was too difficult

to reproduce for it to be efficient to try to directly debug it, so in the case where no valleys were found, the smallest acceleration value would be assumed to be a valley, and then used to synchronize the calculated real acceleration.

After running the optimization algorithm and honing of the parameter ranges over the course of a week, the resulting filtered values began to consistently display desirable results.

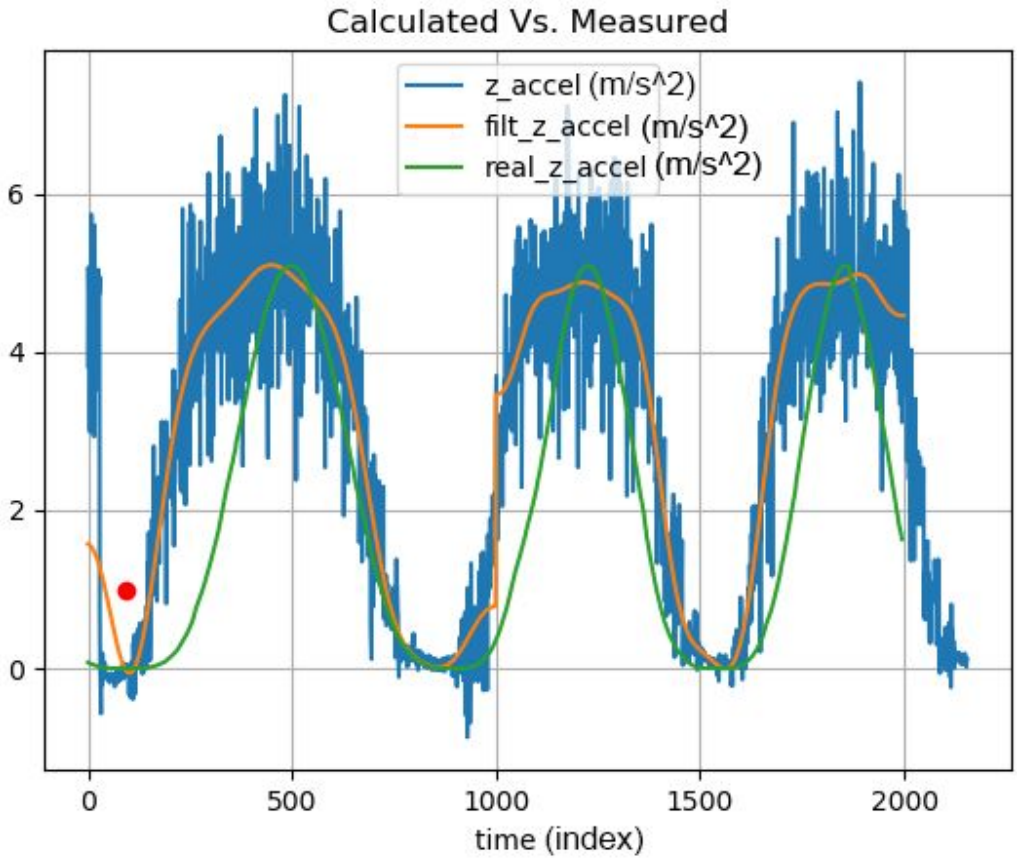



*Fig. 42. Calculated, measured and filtered acceleration, good looking results*

Once the selected parameters were constantly coming up with low scores, the velocity and displacement values were added to the graphs.

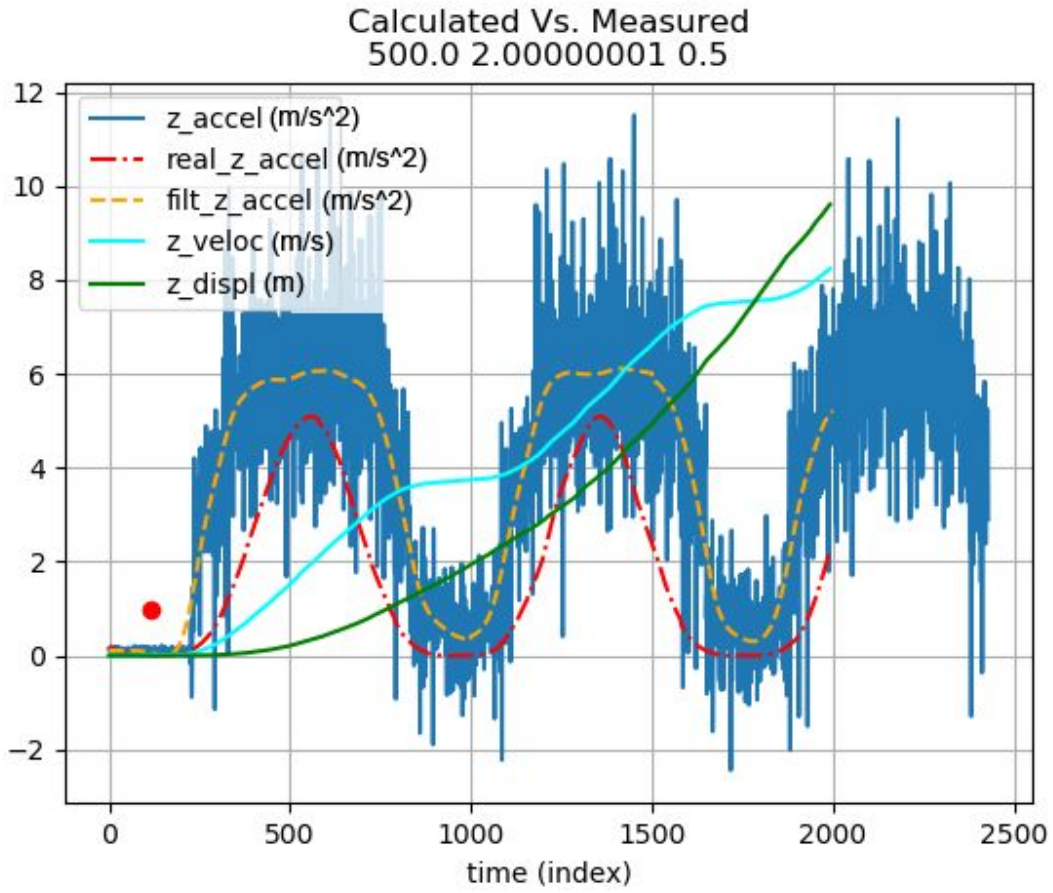



*Fig. 43. Measured, real and calculated values from filter*

If you compare the calculated values in Fig. 40 with Figs. 30, 31 and 32, they all appear to match fairly well, which was taken as an extremely promising sign. The filter parameters were set to range=500, order=2, and cutoff=0.5, and the system was set to be stationary while it gathered data.

# Communication (performed by Jacob)

The proposed design of the communication system in terms of encoding technique remains unchanged from the preliminary design. However, there is a small change in the hardware design of the receiver (phototransistor + comparator). Instead of setting the non-inverting input of the comparator to a constant voltage value, a DAC is implemented instead. This allows for the sub to dynamically change the sensitivity of the receiver in response to the perceived “noise floor” (minimum level so that the output isn’t driven from the noise in the environment alone), thus adding robustness to the design. DAC’s are also cheap and easy to implement, making it a prudent choice for this application.

The primary implementation from this term in terms of communication was a test rig to gauge the BER of the communication system This test rig is implemented as a 55-gallon water tank with a 1 meter maximum distance. A photo of this tank can be found below.

*Fig. 44. UWOC tank, empty*

An optical communication system, like any communication system, is affected by the noise in the transmission channel. In this case noise is ambient light, and since this is unguided wireless transmission, there is little to be done to reliably eliminate the noise completely; the communication system must instead be able to perform in a variety of noise conditions to be useful (i.e. a variety of ambient light levels).

Therefore, ambient light level must be controlled in order to determine how well this communication system performs in a given environment. Firstly, the tank was fully light sealed using a combination of multiple coats of black spray paint and Gaffer's tape, and then by using an ambient light sensor inside the tank to measure the light level when no transmission was occuring, and verifying the illuminance in the tank was 0 lux. To produce an ambient light level, an array of high power full spectrum LEDs were attached to the underside of the lid and controlled via an NPN transistor, which acted as a current amplifier. The UWOC tank lid is shown below.

*Fig. 45. Underside of UWOC tank lid with LEDs*

Once the environment was set up, the optical communication peripherals (LED and phototransistor, in this case) had to be installed. As the transmission distance might vary from test to test, it was imperative that at least one of these peripherals be repositionable in the tank. As ambient light levels were recorded at the phototransistors' location, and even ambient lighting in this environment is not guaranteed, the phototransistor was held fixed while the LED was made repositionable. Extruded aluminum was used for this application as it is very cheap, was easy to source, and has high oxidation resistance. Placing these electrical components on stands also allows for easy replacement and maintenance, if required.

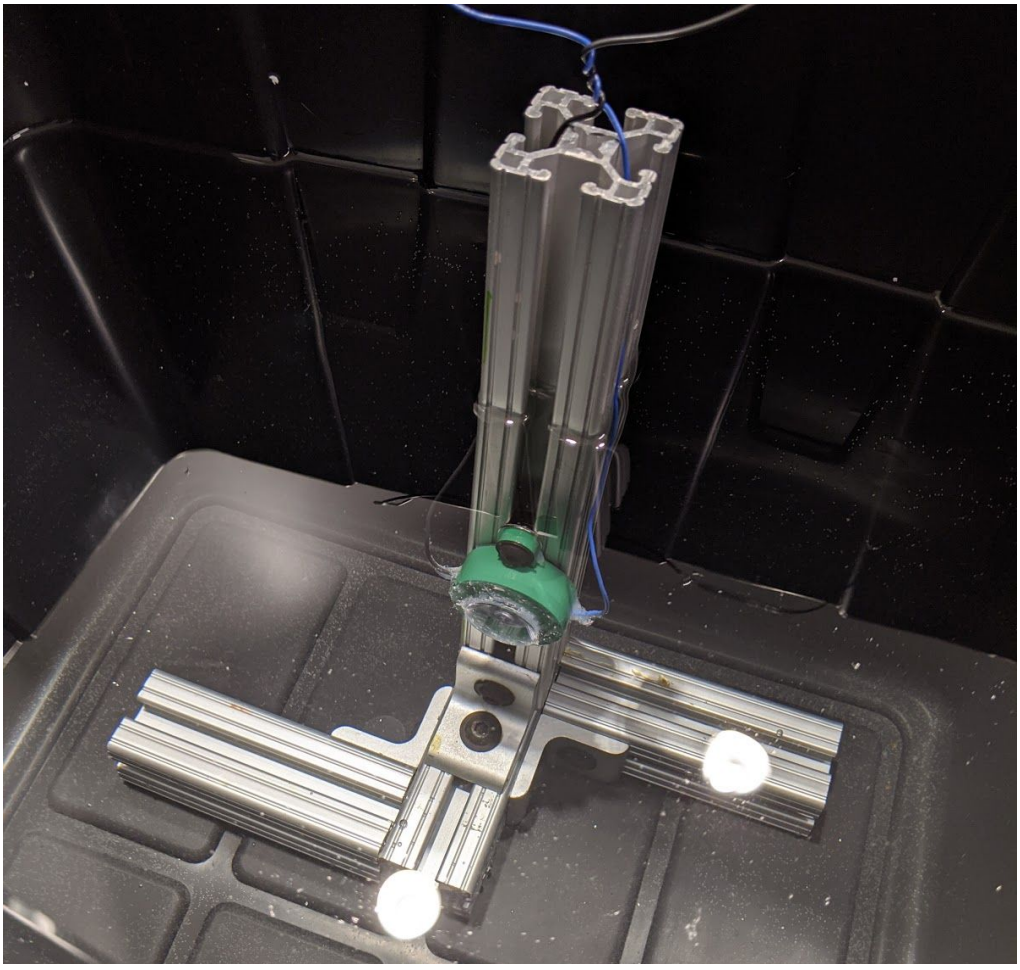


*Fig. 46. Completed LED stand for UWOC tank*

As for the waterproofing of the electrical components, parts were developed by the mechanical engineering team based on specifications provided by the computer engineering team. These specifications were that for the LED, nothing may obscure the view of the lens, and for the phototransistor, nothing may obscure its view to the half-angle, which is 75 degrees in this case. These parts were 3D-printed in ABS and smoothed using acetone vapor, then epoxied together using marine epoxy. Photos of these parts for the LED can be found below.

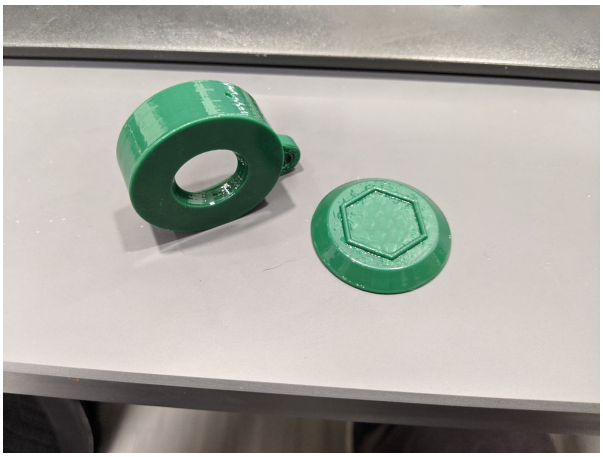


*Fig. 47. LED test rig part, disassembled*

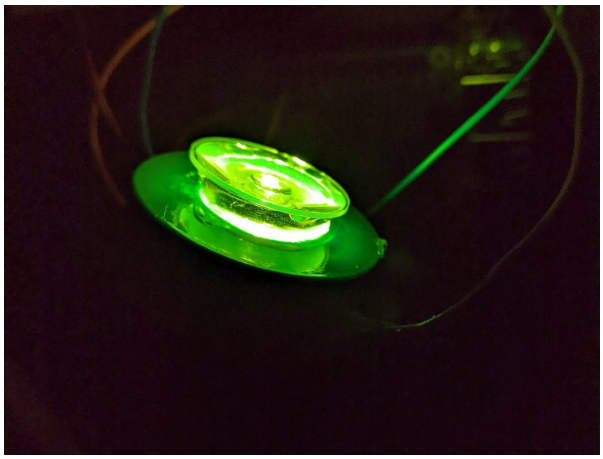


*Fig. 48. LED test rig part, assembled*

All of these parts make up the UWOC test rig that the communication system was tested in. Results of these tests can be found below in the Performance estimates and results section.

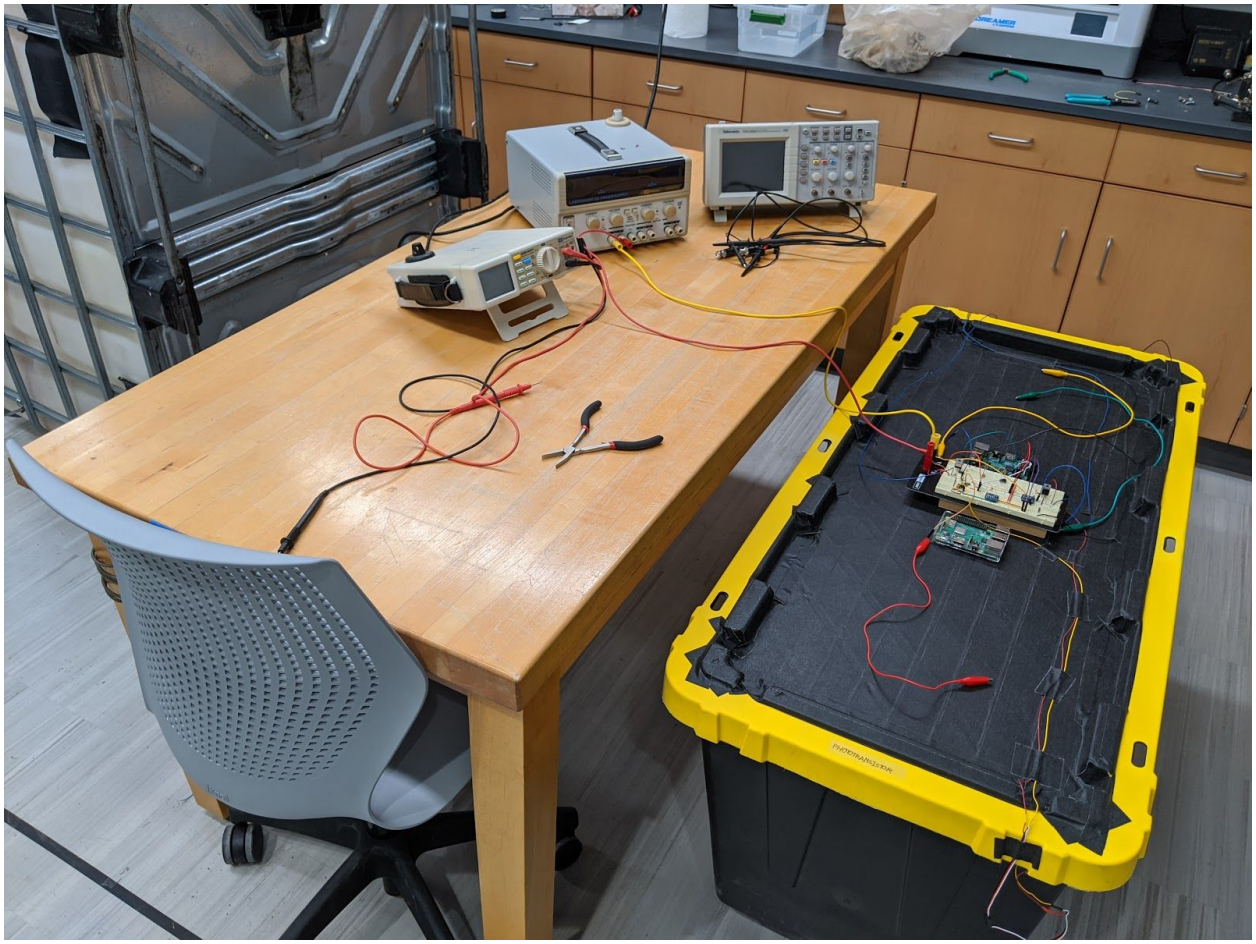


*Fig. 49. UWOC tank, completed*

# Obstacle Avoidance (performed by Xavier)

## Hardware

Initial sonar development began during the first term of this project, with experimentation and research based around the construction of custom waterproof transducers and hydrophones. Custom construction would allow for greater control over the price point vs the precision of this system, which is desirable for affordability. The physical characteristics of the design were

researched and implemented by Sam Veith from the mechanical engineering team, and focused mainly on the layer that interfaces between the piezoelectric element and the transmission medium. Methodologies researched were from a variety of sources, ranging from DIY projects[40] [41][42] to graduate level research papers[43][44], on both transducers and hydrophones. While the computer engineering focus was directed at motion and displacement measurement, the mechanical team developed several prototype transducers. These were tested, and while they displayed desirable functionality, concerns over durability and size on a large scale of production determined that off the shelf transducers and hydrophones would be utilized.

For initial prototyping and testing the CPE-267 was selected, as it operates in a convenient voltage range (6-14v), has the highest theoretical range out of the selected transducers, and is self driven, reducing the number of places at which errors can occur. The only hydrophone selected was the CMC-6027-42L100, which is an electret condenser microphone and can easily cover the range of frequencies that may be encountered.

The transducer was wired to the SBC via an NPN transistor, allowing it to be driven off of 14v from the SBCs 3.3v GPIO. The hydrophone, without amplification, has a relatively low

[40] *Homebuilt Side Scan Sonar*. (2010). MBT Electronics. https://www.mbtelectronics.com/side-scan-sonar.php

[41] Joy, K., Hamilton, J., Jewell, M., & Babb, I. (2016). *Simple Hydrophone Design*. Simple Hydrophone Design. https://www.nurtec.uconn.edu/wp-content/uploads/sites/287/2016/08/COSEE-TEK-Simple-Hydrophone-Material-List-Fabrication-Instructions-V4.2-7-7-2016.pdf

[42] Grzinich, J. (2016, December 9). do-it-yourself hydrophones. john grzinich. https://maaheli.ee/main/d-i-y-hydrophones/

[43] B. Benson *et al*., "Design of a low-cost, underwater acoustic modem for short-range sensor networks," *OCEANS'10 IEEE SYDNEY*, Sydney, NSW, 2010, pp. 1-9.

[44] Won TH, Park SJ. Design and implementation of an omni-directional underwater acoustic micro-modem based on a low-power micro-controller unit. *Sensors (Basel)*. 2012;12(2):2309–2323. doi:10.3390/s120202309

output. The signal from the transducer read directly from the hydrophone produces the output as shown below in Fig. 50**.**

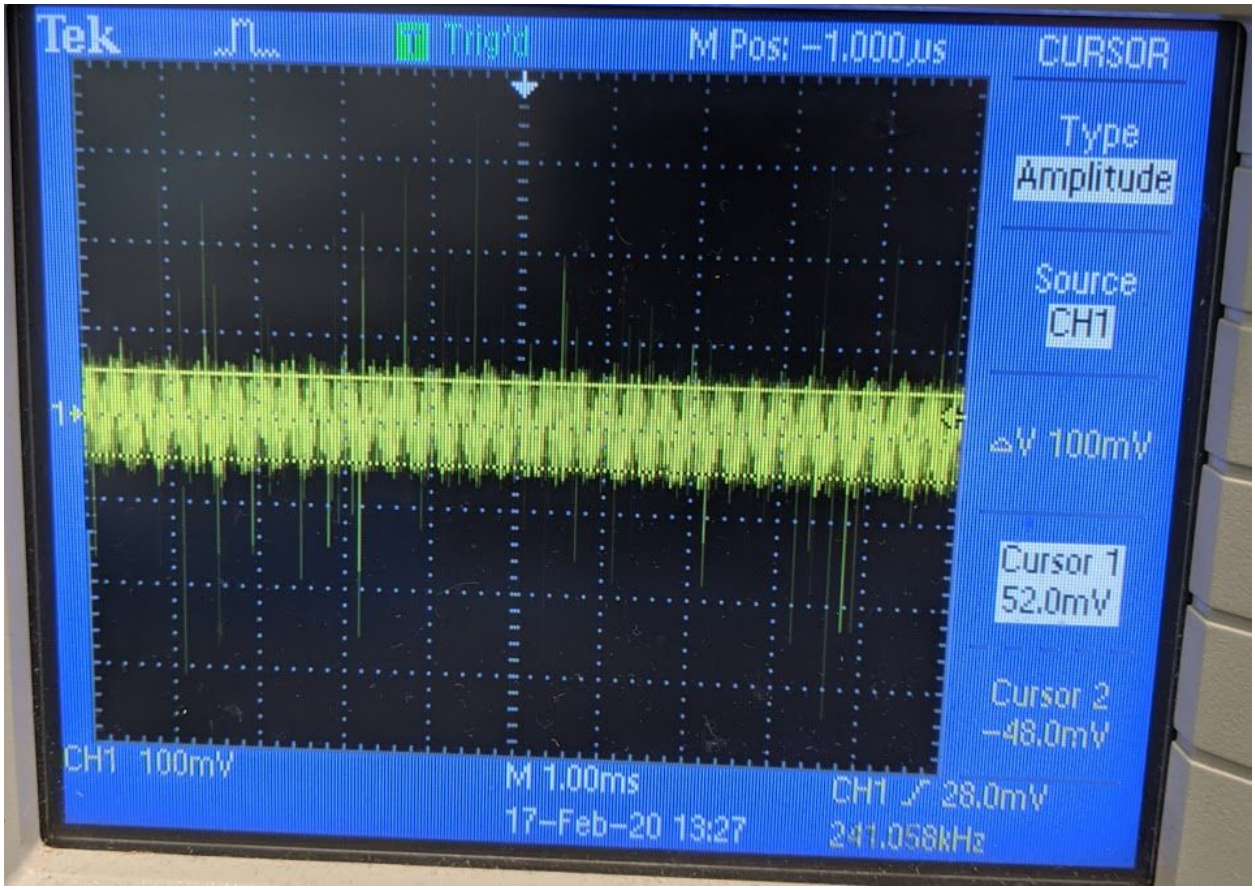



*Fig. 50: Unamplified read in signal from undriven hydrophone to oscilloscope*

The transducer signal is clearly there, but there is a lot of noise present as well, which was expected. The hydrophone is undriven at this point as it was only being used with the oscilloscope, but to have it function in a circuit a simple driver is required.

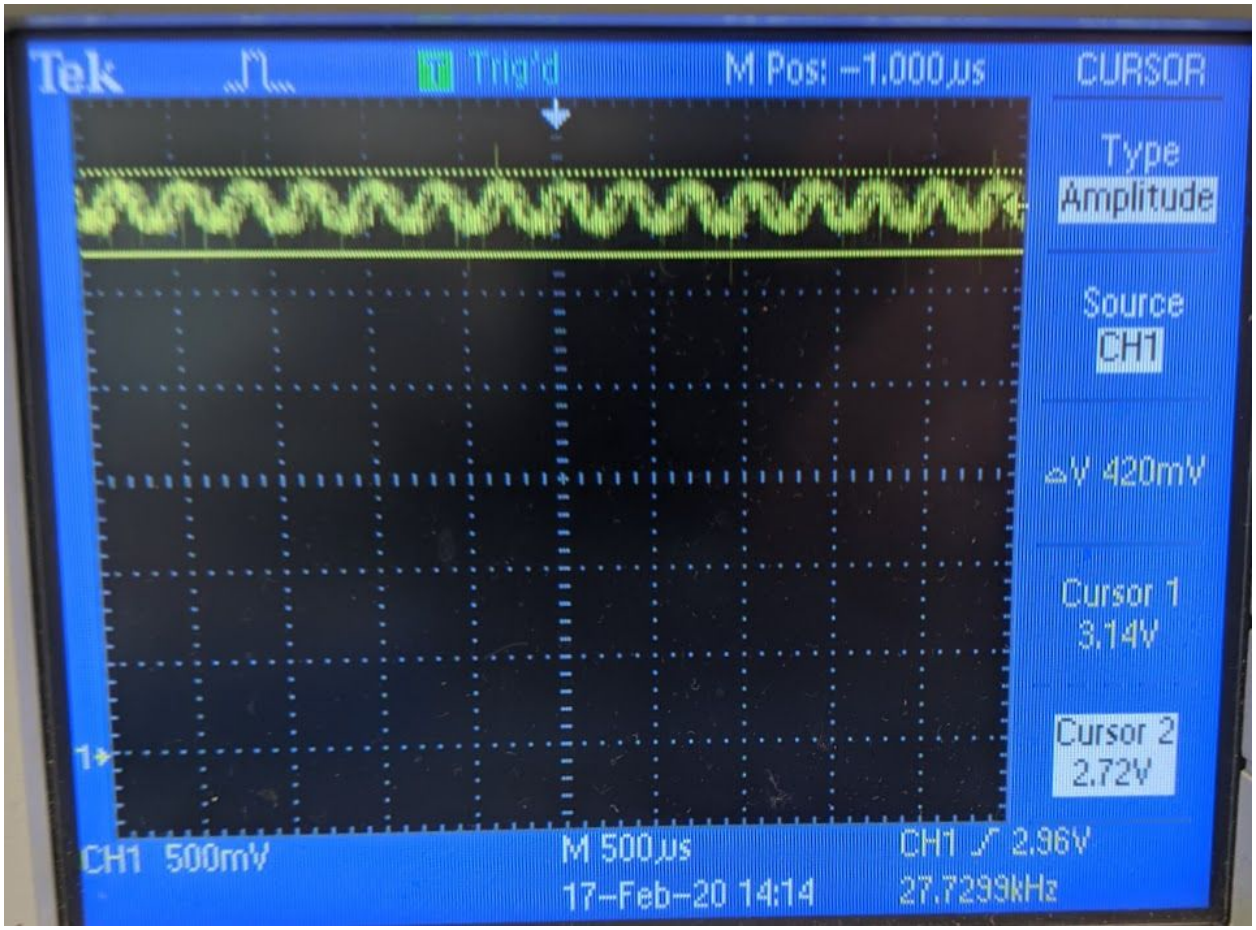



*Fig. 51: Driven hydrophone output*

Driving the hydrophone adds a DC offset, but the output amplitude is now ~4x larger, which makes it a lot easier to process. The offset was filtered out with a capacitor, which both reduced the noise slightly and centered the signal around 0v.

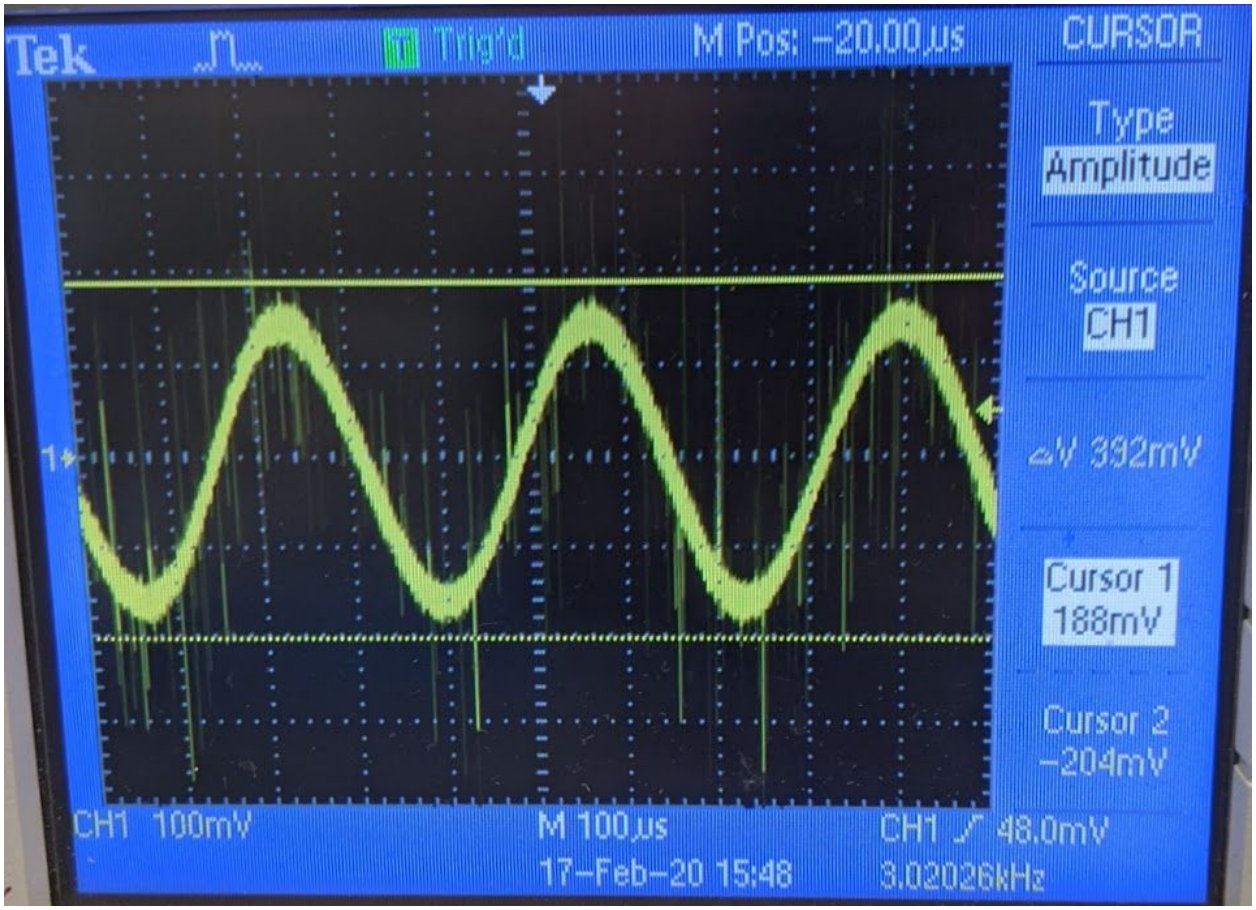



*Fig. 52: DC filtered hydrophone output.*

A passive bandpass filter was designed around the transducer frequency, however this was experimentally determined to be ineffective. A large number of ICs with bandpass capabilities were compared, and based on the performance/pricepoint ratio, the MAX274 chip was selected, which allows for the construction of 4 2nd order filters. The filter design software for this chip had to be spun up in a DOS emulator, though the resulting functionality provided resistor values to generate the desired filter. This ended up being a 4th order butterworth bandpass filter centered at 2.9kHz with a moderate Q factor to allow for frequency shifts due to both medium and potential doppler effect while under motion. Said resistor values were replaced with the closest standard resistor values.

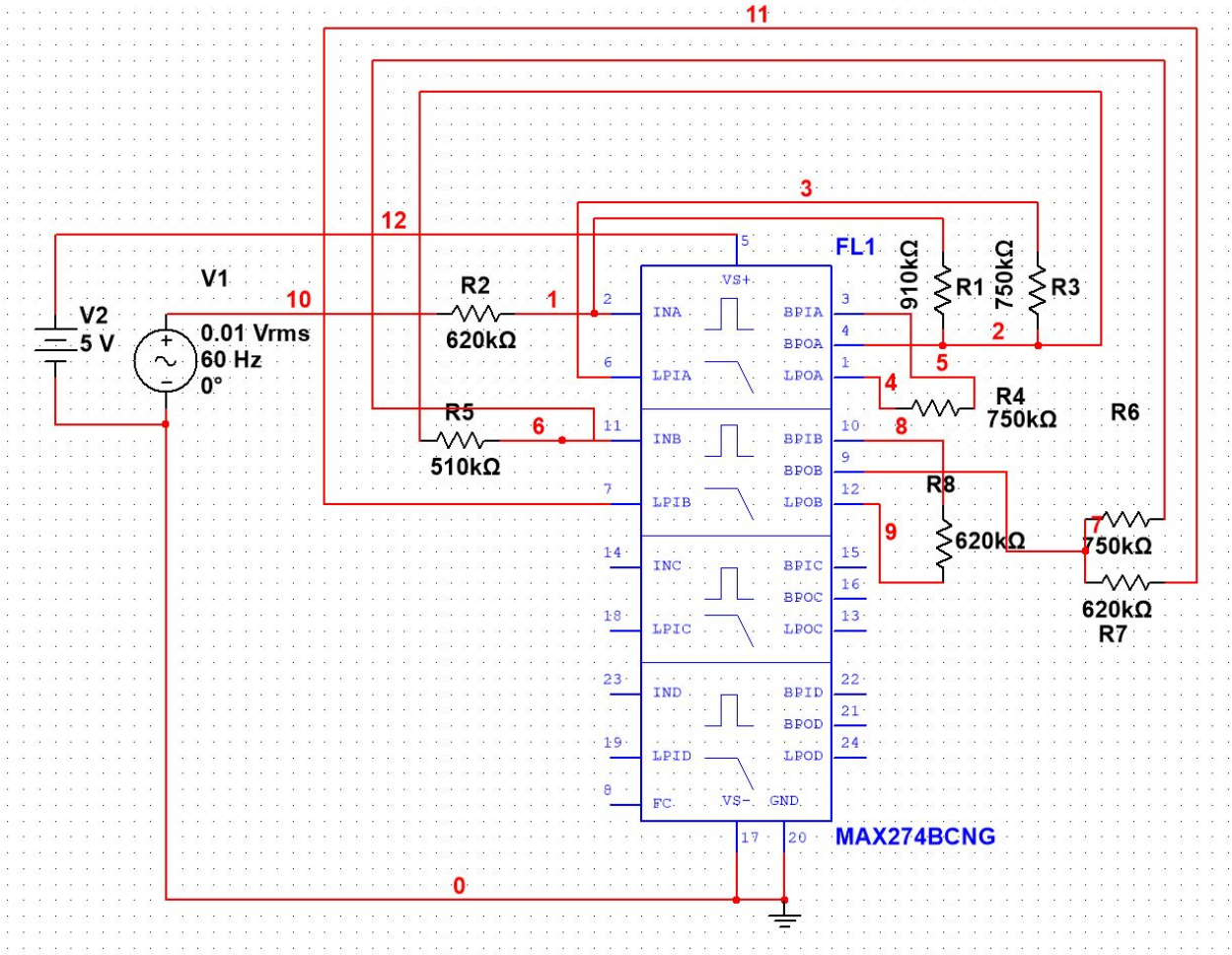



*Fig. 53: 4th order butterworth filter*

The filter shown in Fig. 53 was simulated across a range of frequencies, and its response is shown below in Fig. 54.

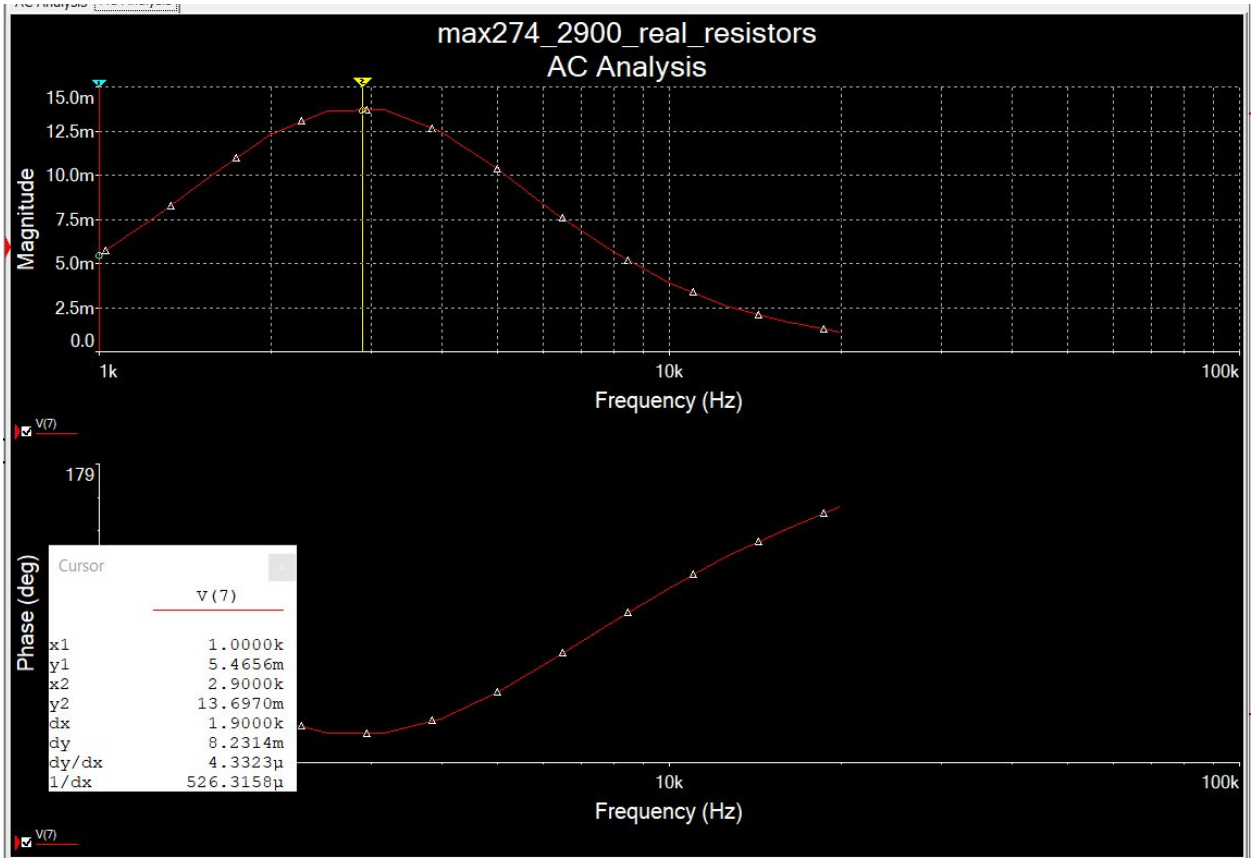



*Fig. 54: Response of 4th order butterworth filter.*

As can be seen there is a reasonably sharp dropoff as the frequency extends to either side of the selected band.

The output of the DC filtered hydrophone needed to be amplified to match the range of the bandpass filter, so the signal is passed through a simple op-amp circuit built from a TL072 chip before reaching the bandpass stage.

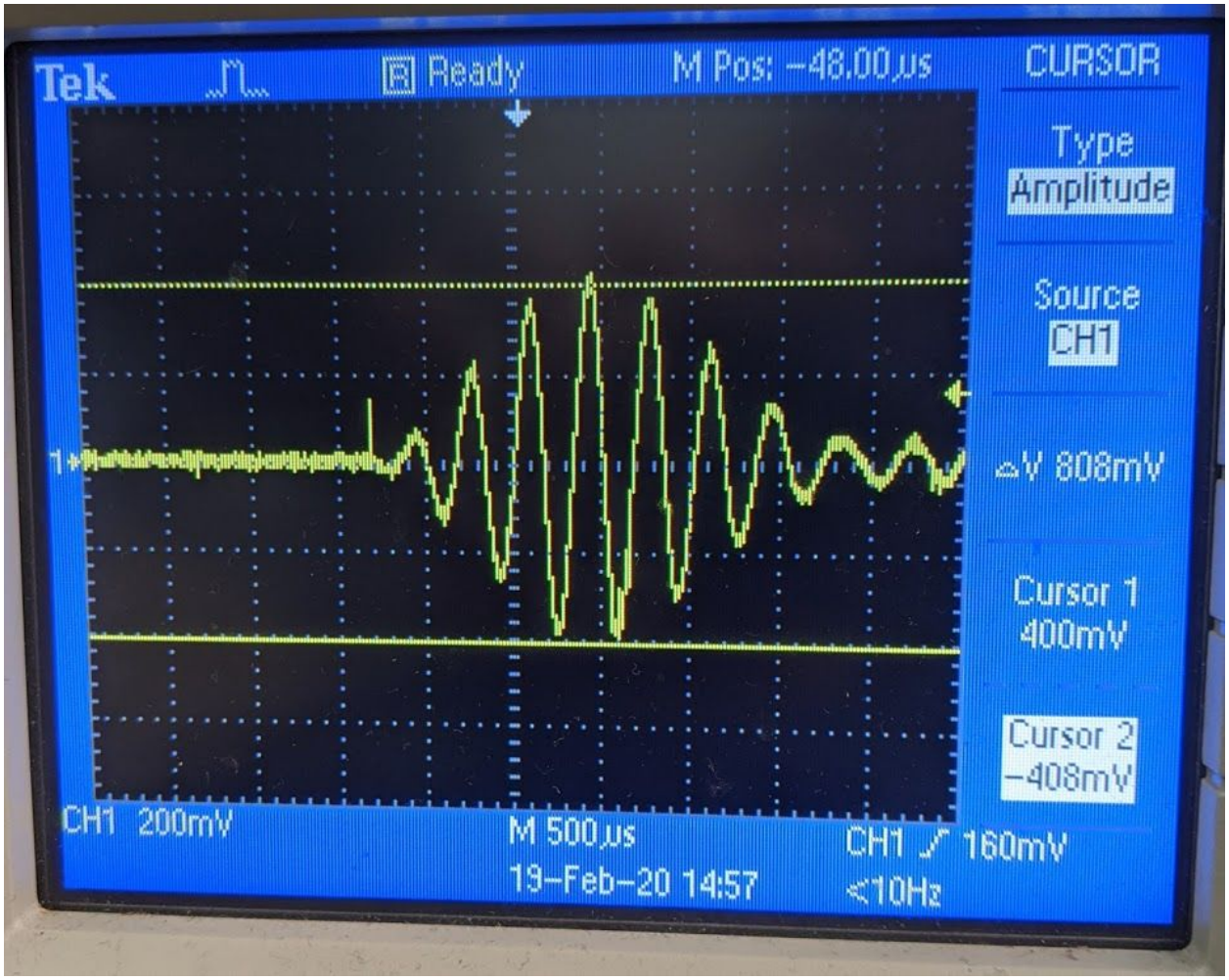



*Fig. 55: Filtered received ping signal.*

Fig. 55 displays the signal produced by driving the transducer for 2ms, read in by the hydrophone and passed through the bandpass filter. It very clearly is a much cleaner signal than the unfiltered tests, and exactly matches what is desired for this stage.

The next problem is reading in a ping to the SBC. As the selected SBC does not have analog inputs, the read in values need to be digital values. To do this, a LM311p comparator was used. A comparator is a device which has a setabel threshold, and when a read in value is above said threshold, outputs a digital 1, and outputs 0 otherwise. When a received and filtered ping is fed into the comparator, the output is as shown in Fig. 56.

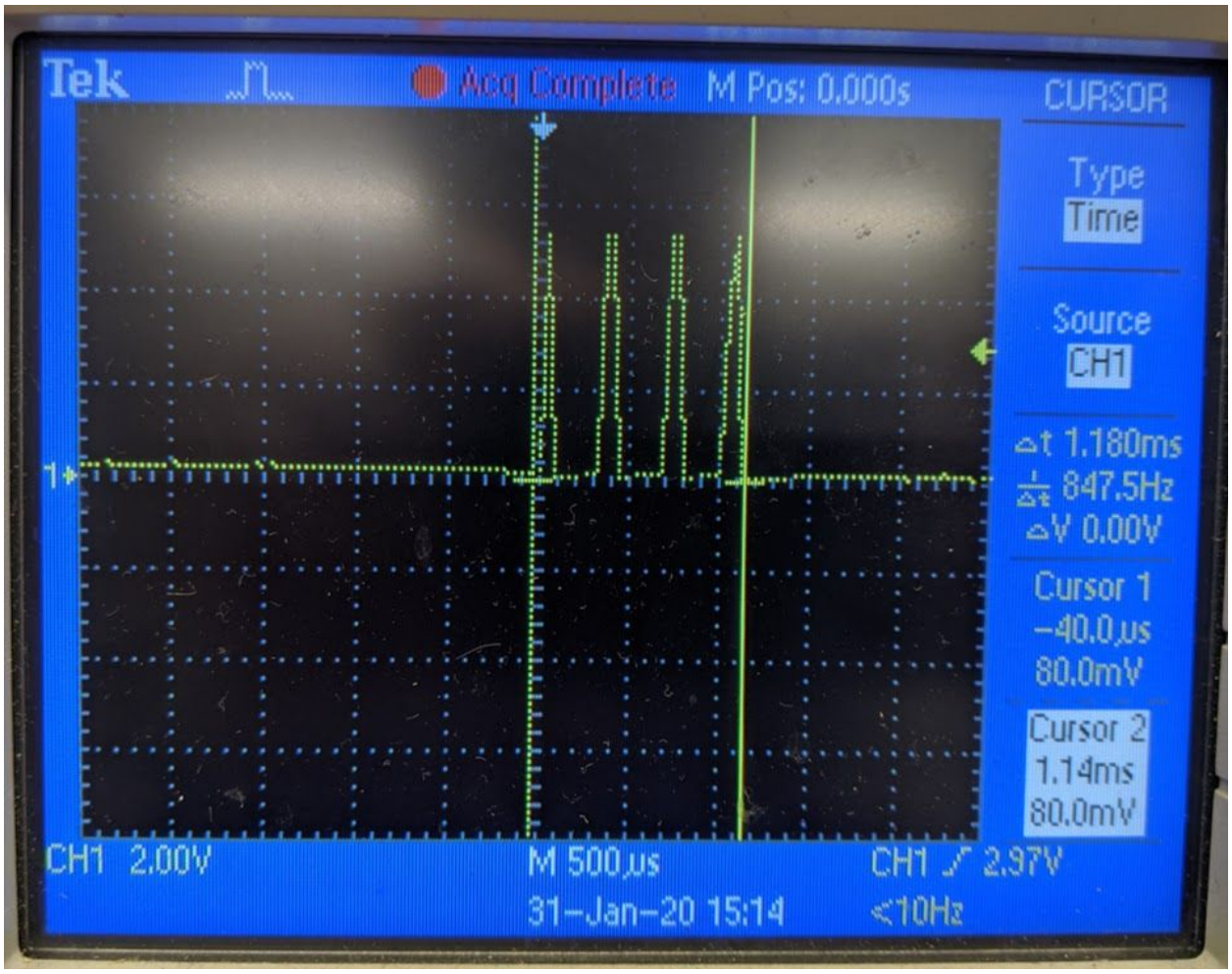



*Fig. 56: Comparator result from a ping signal*

Each of the spikes corresponds to one of the signals pulses crossing the threshold value. From preliminary software tests, it was determined that this level of signal abstraction was too great to easily process, as the spikes from the sent signal and the received echo are difficult to differentiate with so many values per ping. To solve this, a circuit was designed to convert a ping into a single signal pulse, to be inserted between the filter and the comparator. The first stage rectifies the signal so that only the positive part of the signal remains, as can be seen in fig. 57. The second part is a smoothing capacitor, which converts the series of positive pulses into a

single pulse, which can be seen in fig. 58,  to be fed into the comparator, the output of which can be seen in fig. 59.

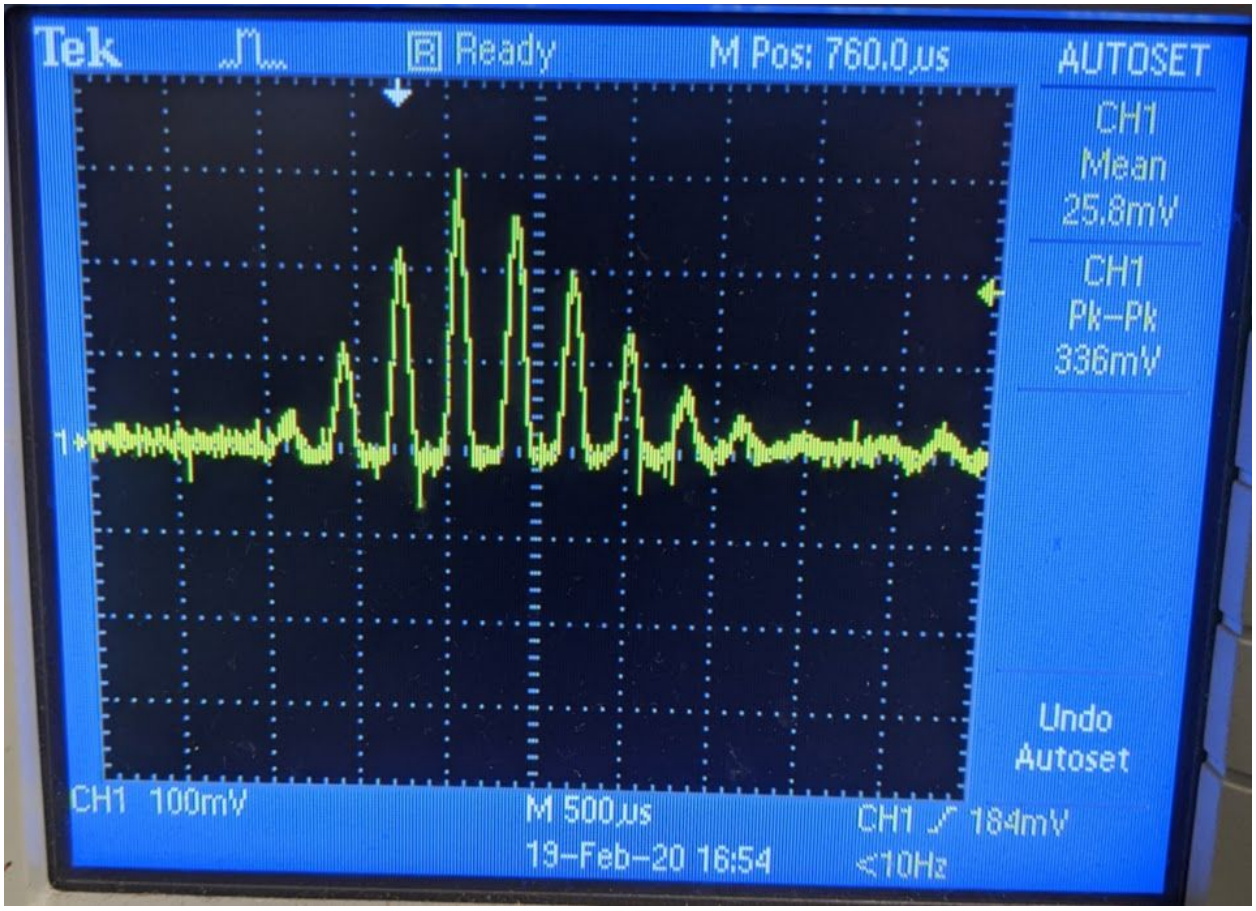



*Fig. 57: Rectified signal*

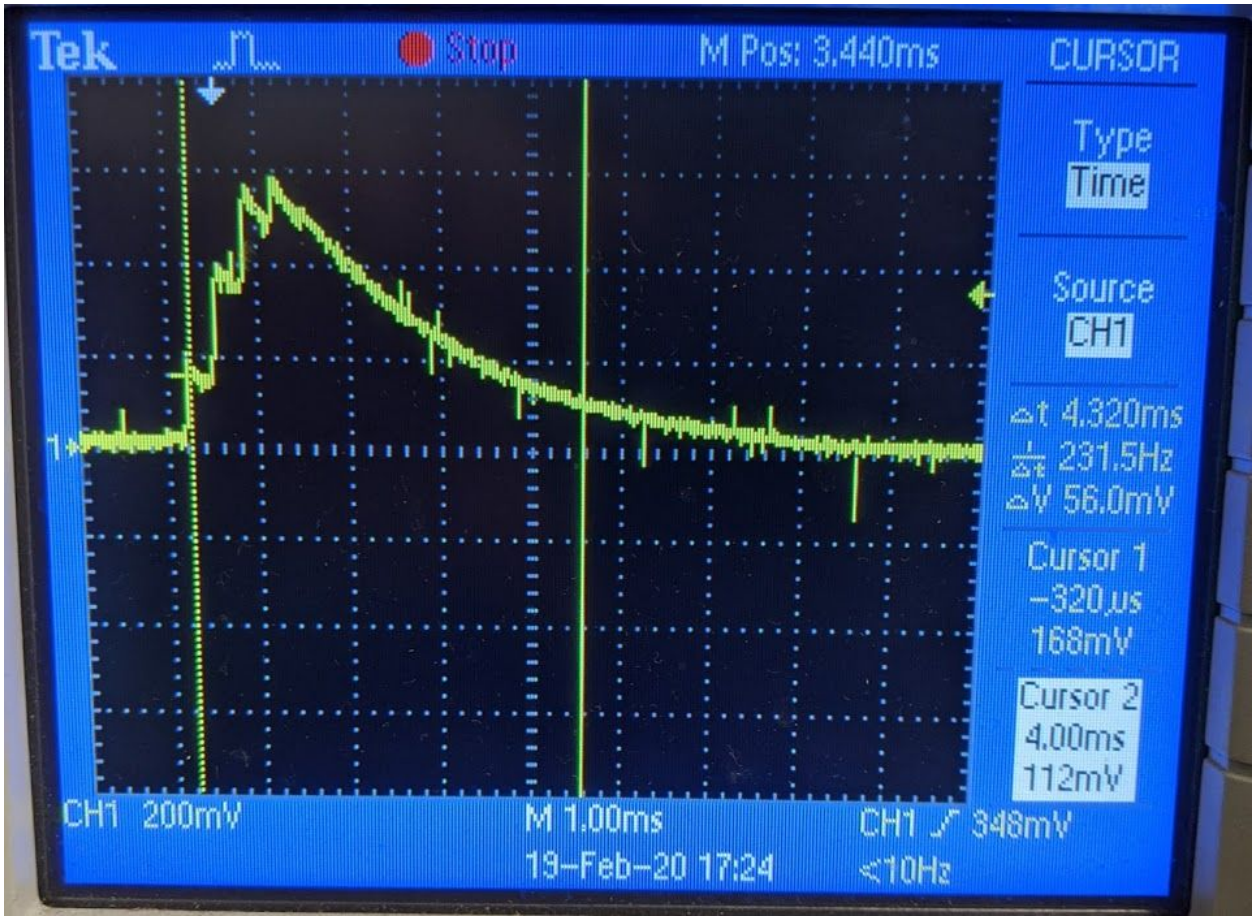



*Fig. 58: Smoothed rectified signal*

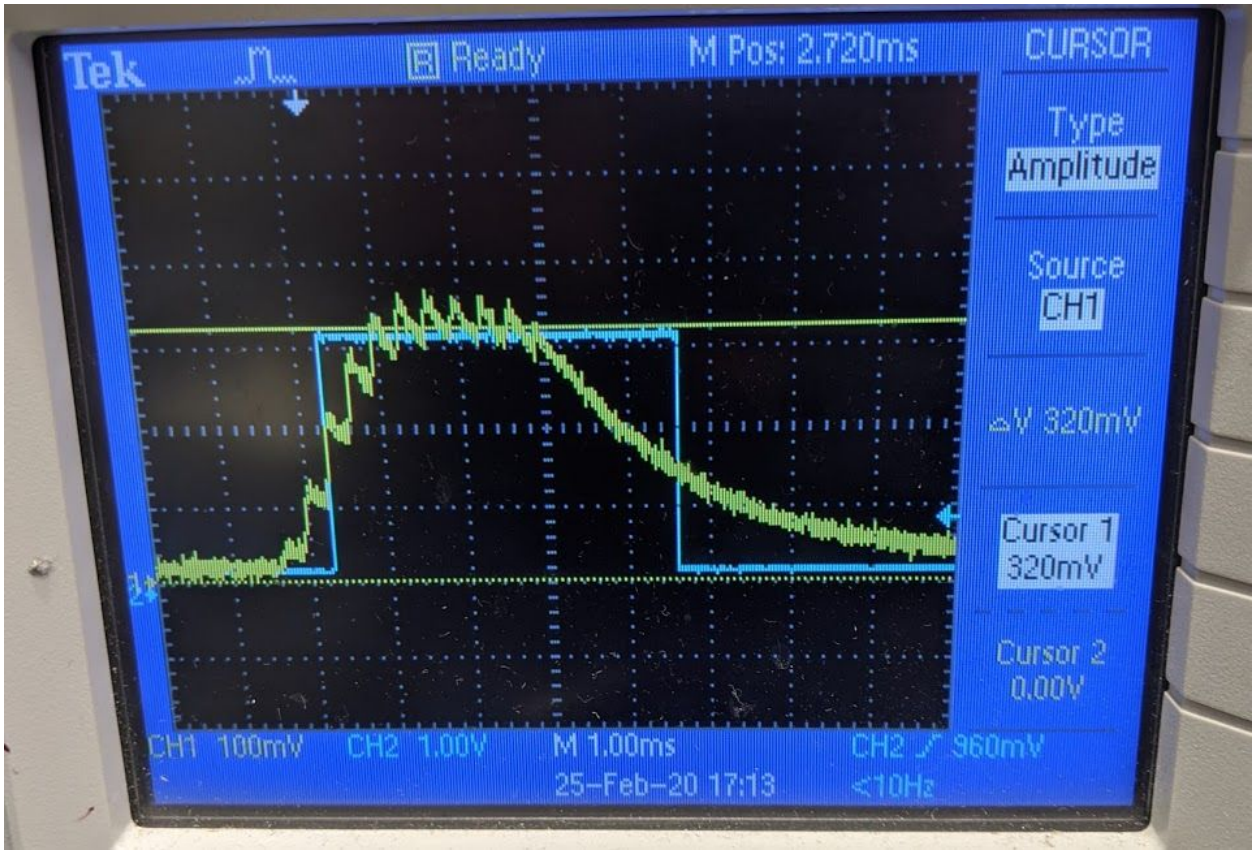



*Fig. 59: Smooth signal and associated comparator output.*

The entire constructed circuit with all of its stages allows a pulse to be consistently read in digitally by the SBC. The diagram and physical hardware layout of the full circuit can be seen below in figs. 60 and 61.

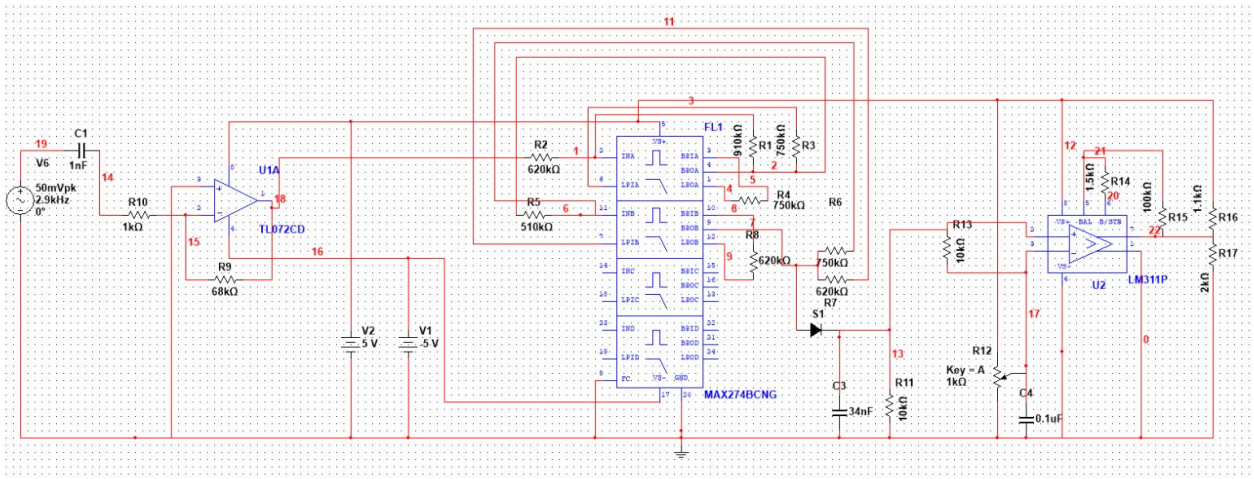



*Fig. 60: Sonar receiver circuit diagram*

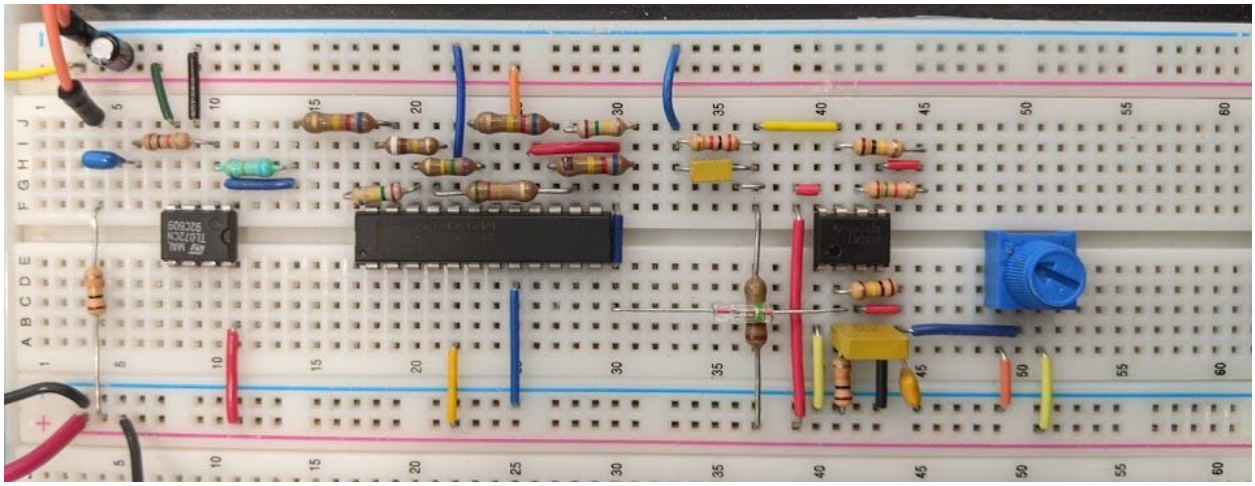


*Fig. 61: Sonar receiver circuit.*

## Software

As the hardware stage of this module is relatively complex, the software side of things is equivalently simple. The sonar module is encapsulated in a class, which is initialized with the associated pins and the transmission medium [air or water]. A measure_distance() function

handles everything needed. It sends out a ping, then waits until the return value is populated before returning the calculated distance. When the sonar class is initialized, it starts a callback function on the receive pin, which when called calculates and stores the time difference between transmission and reception of an echo. Once this time value is recorded, the measure_dist() function uses the speed of sound in the given transmission medium to return the measured distance. This setup allows for constant asynchronous distance measurements to be taken so that the most recent distance measured is readily available, as well as being able to read a new distance on command. Having this much control over the system is critical as obstacle avoidance is the largest active safety factor in the AUV.

# Performance estimates and results

## Movement (performed by Jacob)

### Software PWM Library under load

For an early implementation of PWM in the *Design* section, Engine block, a software PWM library was used to drive all ESCs. To ensure that the PWM library would be able to keep up with real time duty cycle changes and not oscillate uncontrollably, a load test was performed to determine the performance of this library.

PWM was performed on five GPIO pins. One of the five pins was randomly selected to act as the reference pin. The reference pin was sent a 5% duty cycle at 50 Hz (as would be

experienced experimentally) and the maximum jitter of the signal was recorded before any other pins were turned on. Here, "jitter" refers to the deviation of the pulse period from the expected value. For a 5% duty cycle at 50 Hz, the expected pulse period is $0.05 * (1/50\ Hz) = 1ms$. However, deviations (or jitters) of up to 3 μs were observed, or 0.33% of the expected pulse width. The other four pins were then randomly and continuously assigned duty cycles between 5 and 10% at 50 Hz while the reference pin's output signal was examined. It was found that the maximum jitter was approximately 3.2 μs. This is a negligible difference and thus it can be concluded that the PWM library is capable of performing PWM on multiple pins simultaneously without issue.

## Heading Change PID Controller

Traditional second-order system behavior indicators (e.g. %OS error, peak/settling times) are not useful for approximating the behavior of this submarine, as these approximations assume a deterministic environment. Therefore, literature was consulted to provide some estimation for performance- in this case, settling time is the most important performance indicator- and a settling time of 5 to 10 seconds was found to be likely for stationary heading change.

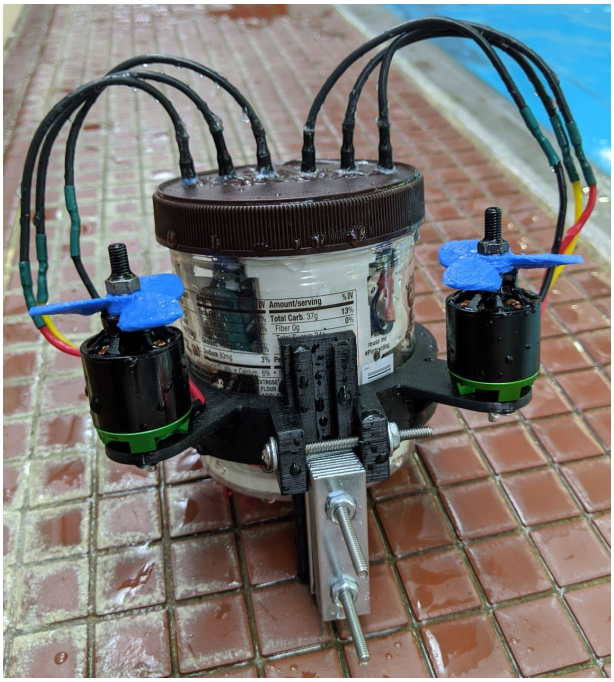


*Fig. 62. Photo of the ATS Mk. 1*

One of the most basic behaviors this sub must be able exhibit is changing it's heading to a target heading. Therefore, some control system must be capable of deciding what PWM values to send to the actuators in the system to achieve this heading. As described in the *Design* section, Engine subsection, each motor is driven from a value in the range of [-100, 100]. It should also be mentioned that the motors used for this application are high performance, high RPM motors that can draw up to 35 A each. This is overpowered for testing with the ATS Mk. 1 which is a small and light submersible, but were the only available motors at the time. For the stationary tests examined in this section, motor value/speed range was constricted to [-2, 2] (i.e. 2% speed) for general stability reasons as a speed over 2% caused overshooting in all cases. This sharply limits the range of speeds that the motor can be driven at by the PID controller, and future tests and test systems will include hand-selected motors that are appropriately powered.

Once the PID was implemented and confirmed as working correctly, the next step was to tune the controller. Several heuristics exist for tuning PID controllers; manual tuning and the Ziegler-Nichols method are two prominent choices. Manual tuning follows a set experimental plan[45]: Set the I and D terms to 0. Increase P until the system oscillates, then halve that value- this is you $K_p$. Increase the I term until the system corrects itself within a reasonable period of time, however not too much as that will call instability- this is you $K_i$. Finally, increase the D term until the system corrects itself within a reasonable time frame- this is you $K_d$. This method guarantees a result that is suitable for the use case, however can be quite time consuming. The Ziegler-Nichols method produces $K_p$, $K_I$, and $K_d$ by using the proportional gain term at the point of oscillation $K_c$ and the period of that oscillation $T_c$ as variables for the three respective terms. For a PID controller (methods exist for P and PI controllers as well), these equations are $K_p = 0.6K_c$, $K_i = 2.0/T_c$, and $K_d = T_c/8.0$[46]. This method is quick to implement, however has several limitations. The loop is tuned for quarter-amplitude damping, meaning that absolute oscillations above the setpoint have amplitudes that are ¼ of the previous amplitude. Inherently this means the loop is built to oscillate, which is not desirable. Instability is another issue, especially in lag-dominant processes such as this one (that is, slow reaction time and drift due to momentum on turns). Therefore, as the loop time for this process is fairly short, manual tuning was used.

[45]Bucz, Š., & Kozáková, A. (2018). Advanced Methods of PID Controller Tuning for Specified Performance. PID Control for Industrial Processes. doi: 10.5772/intechopen.76069

[46]Ziegler, J. G., & Nichols, N. B. (1993). Optimum Settings for Automatic Controllers. Journal of Dynamic Systems, Measurement, and Control, 115(2B), 220–222. doi: 10.1115/1.2899060

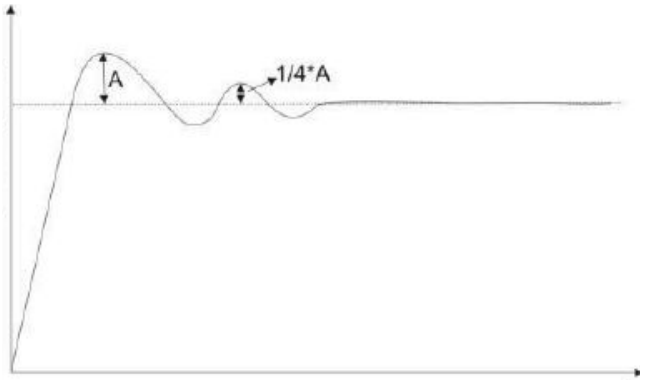



*Fig. 63. Illustration of quarter amplitude decay*

Early testing results are shown below, performed in the Union College swimming pool. While one of the main goals of these tests were to determine the optimal $K_p$, $K_i$, and $K_d$ terms, the other was to gain some insight into how large of an impact waves and other transient motions in a body of water would impact the stability of the system. Specifically, how much did waves/buoyancy impact heading vs. the PID controller. Therefore, qualitative observations are provided at various phases of testing to offer a potential explanation for any unidentified behaviors, which are obtained by watching footage obtained of every test.

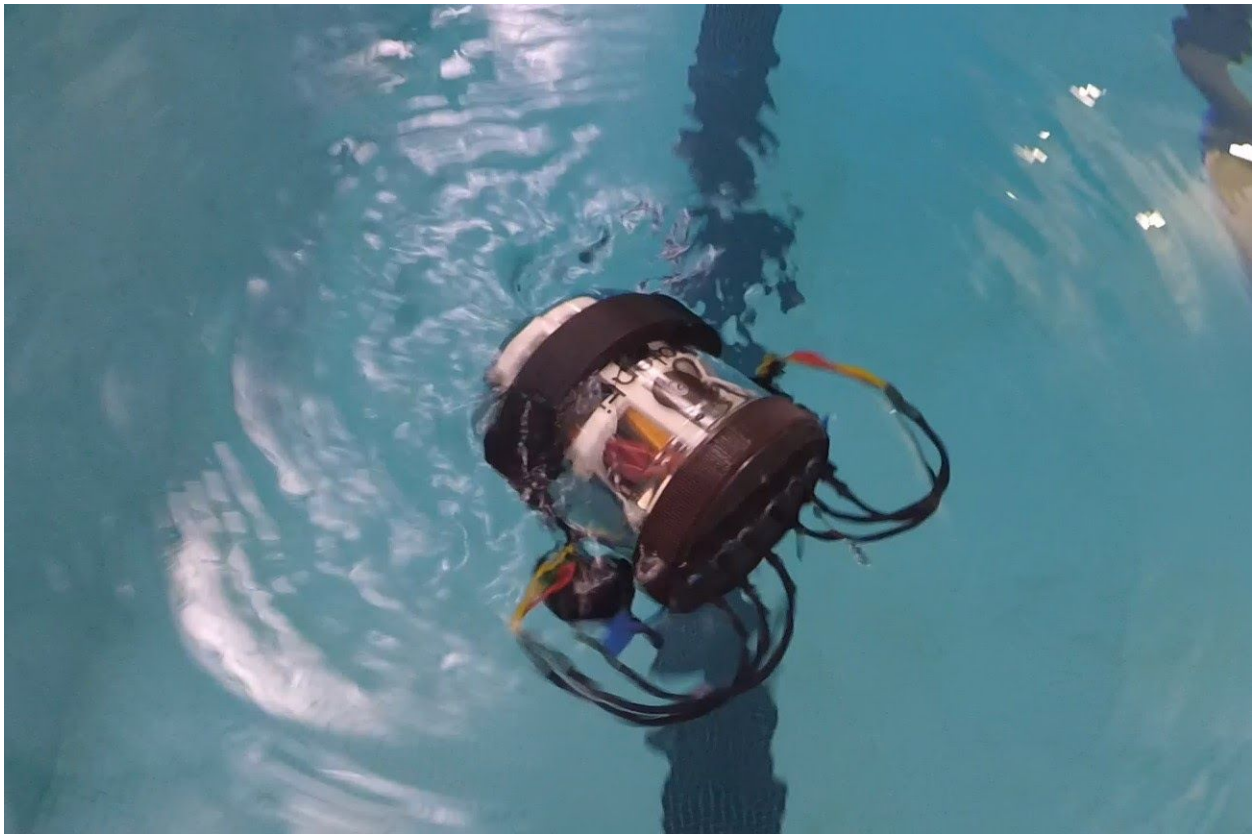
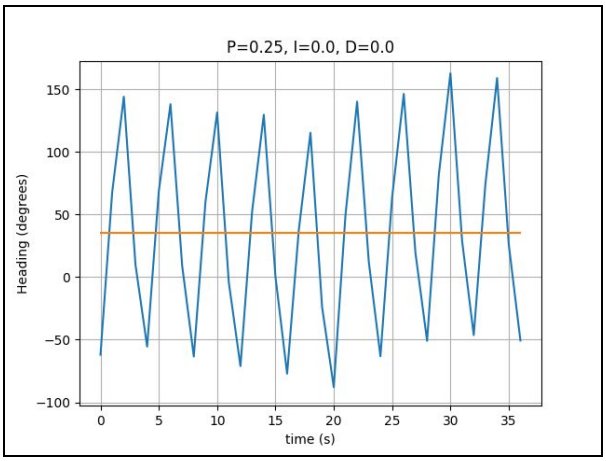


*Fig. 64. ATS Mk. 1 in action*

Once the system was placed in water and confirmed as ready, the I and D terms were set to 0, the P term was increased until oscillations formed. Below is an example of severe oscillation with P=0.25.



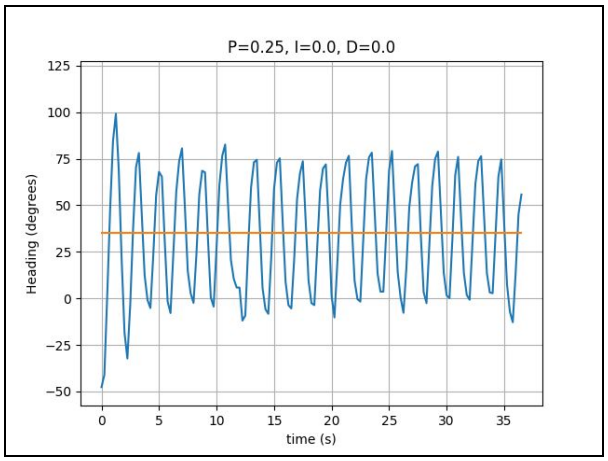
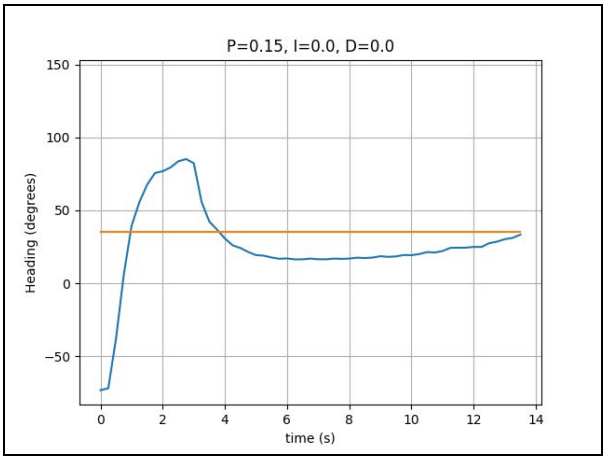



*Fig. 65. PID oscillation at P=0.25, low frequency*

*Fig. 66. PID oscillation at P=0.25, high frequency*

During these tests, almost all motion was generated by the motors on the sub and waves played a very small role. Although both tests were run with the same PID parameters, the period of the oscillation is significantly different between the two, along with amplitude. This points to severe overshooting in both directions and appears to create an endless loop; a large heading difference spurs a strong motor response, which overshoots the setpoint largely creating another large heading difference, hence a strong motor response, and so on. The P term was scaled back to 0.15.



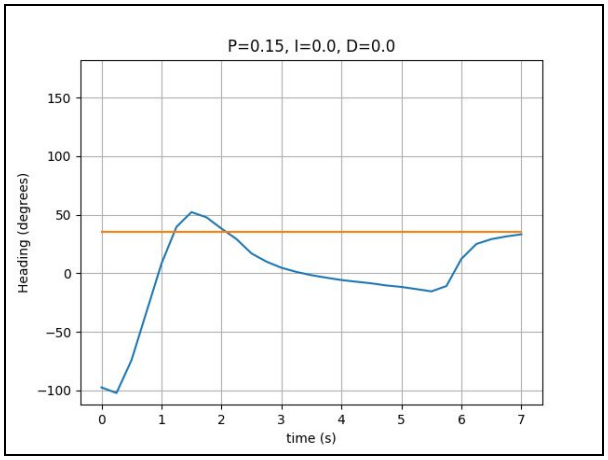




*Fig. 67. PID performance at P=0.15, t=14s*

*Fig. 68. PID performance at P=0.15, t=7s*

To ensure that the oscillations from the previous tests (P=0.25) were addressed, the sub was positioned at the same starting heading, roughly -50 degrees. As can be seen above, the oscillation is completely gone, which is an overcorrection as by definition the $K_c$ term requires fixed oscillation. Hence, $K_c = 0.2$ was determined.

For the left figure above, waves placed a large role in the controller's reaching of the setpoint. There was a counter-clockwise motion induced by the motors at $t = 3\ s$ but all motion afterwards was purely due to forces in the environment acting on the sub. As any motor pulse typically creates more disturbance than in corrects, reaching a space generally close to the setpoint is understood as a large factor for continuous stability.

The next step was to divide this value by two and set it as the $K_p$ term, then start on determining the optimal I term. Several I terms were used, with some of the best results being at I=0.0 and I=0.05.

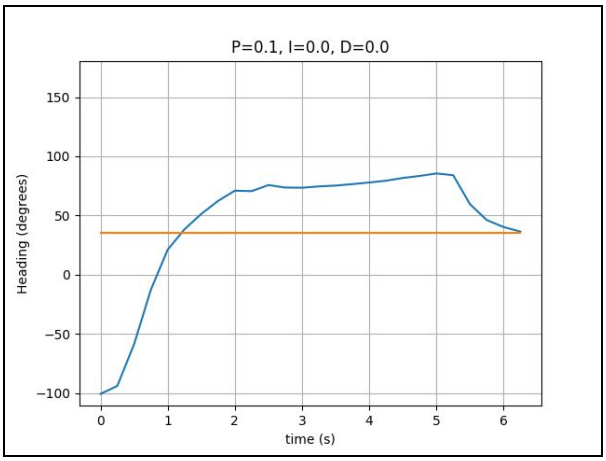



*Fig. 69. PID performance at P=0.1, I=0.0*

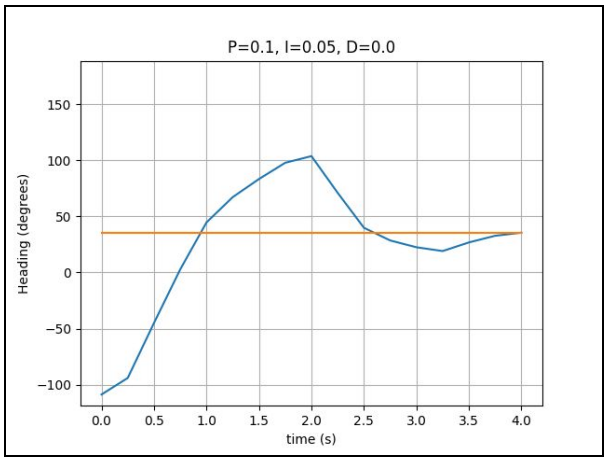



*Fig. 70. PID performance at P=0.15, I=0.05*

Results begin to look promising, and at the very least the controller closes in on the setpoint within a more reasonable time period. The right figure above points to better stability and loop control, as the slope's tangent line towards the final headings approaches is closer to the horizontal. Once again, for this figure, environmental forces caused the inflection at $t = 3.25\ s$, and the only motor actions were at times $t = 0\ s$ and $t = 2\ s$.

Increasing the I term to 0.075 caused oscillation, as seen below and as expected at some point when taking the integral term. Therefore, $K_i = 0.05$ was determined.

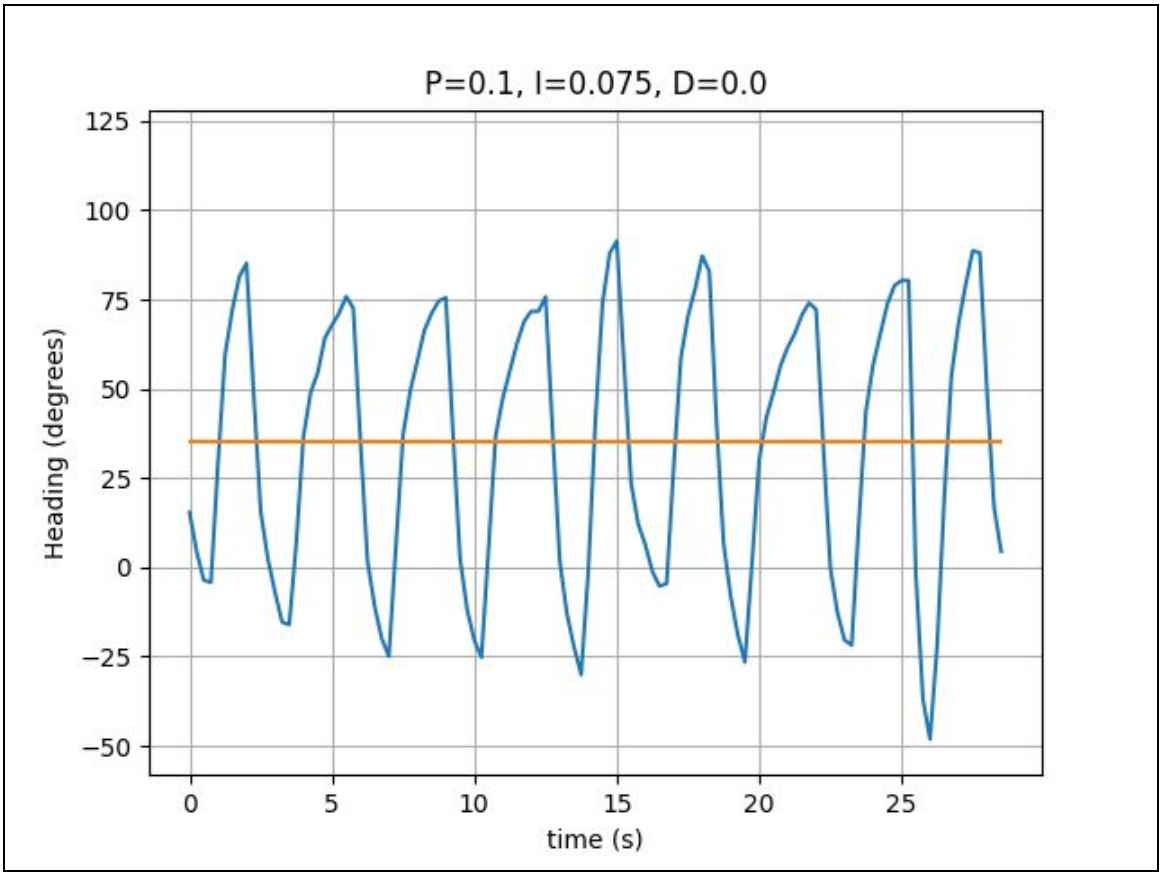



*Fig. 71. PID performance at P=0.1, I=0.075*

Finally, the D term was examined. As mentioned before, the D term may not be useful in this application, especially when factoring in how disturbing spinning motors can be to the relatively tranquil state of the environment. A good visual example of this hindrance is displayed below, when the D term was set to 0.1.

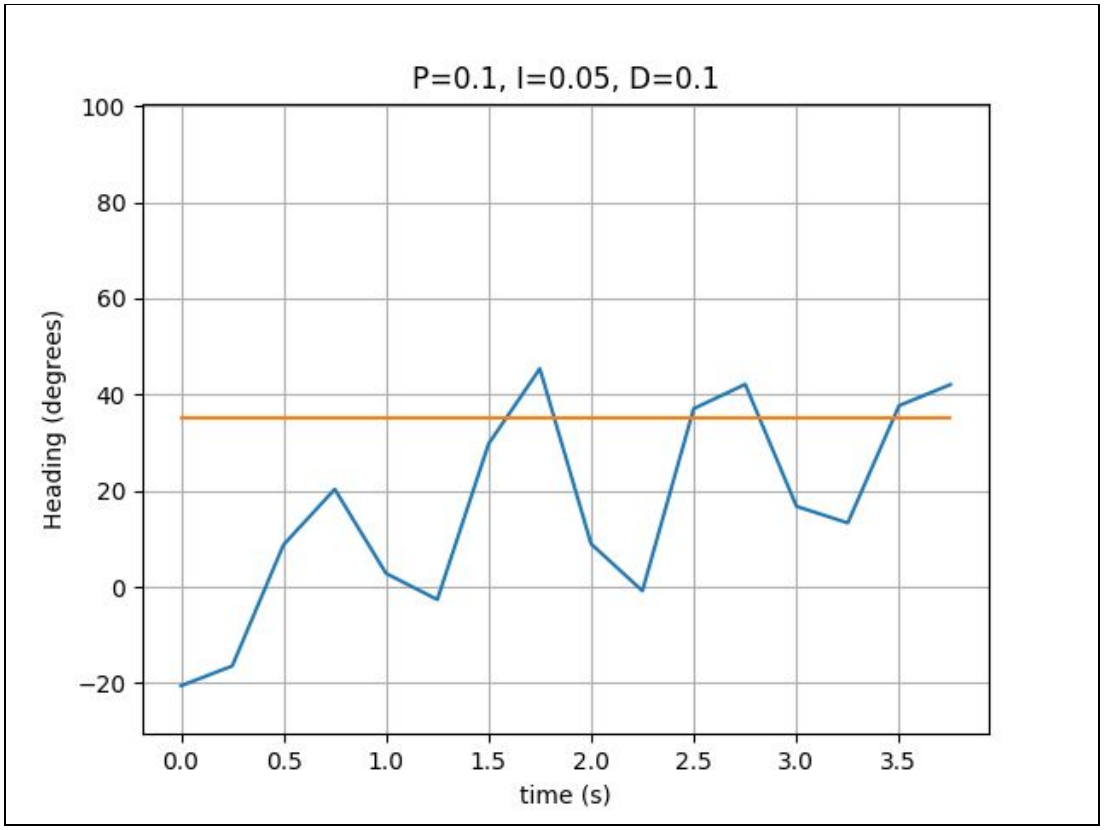



*Fig. 72. PID performance at P=0.1, I=0.05, and D=0.1*

While the controller was able to close in on the setpoint fairly quickly, the rapid motion exhibited by the sub caused drift and a relatively steep tangent line at the finishing point. There is also an example of preemptive action at $t = 0.75\ s$, where the sub backed away from the setpoint line as it was heading towards it. Therefore while more motion may allow the PID to reach its target quicker, and can even be orchestrated in a way that doesn't cause infinite oscillation, stability is still problematic.

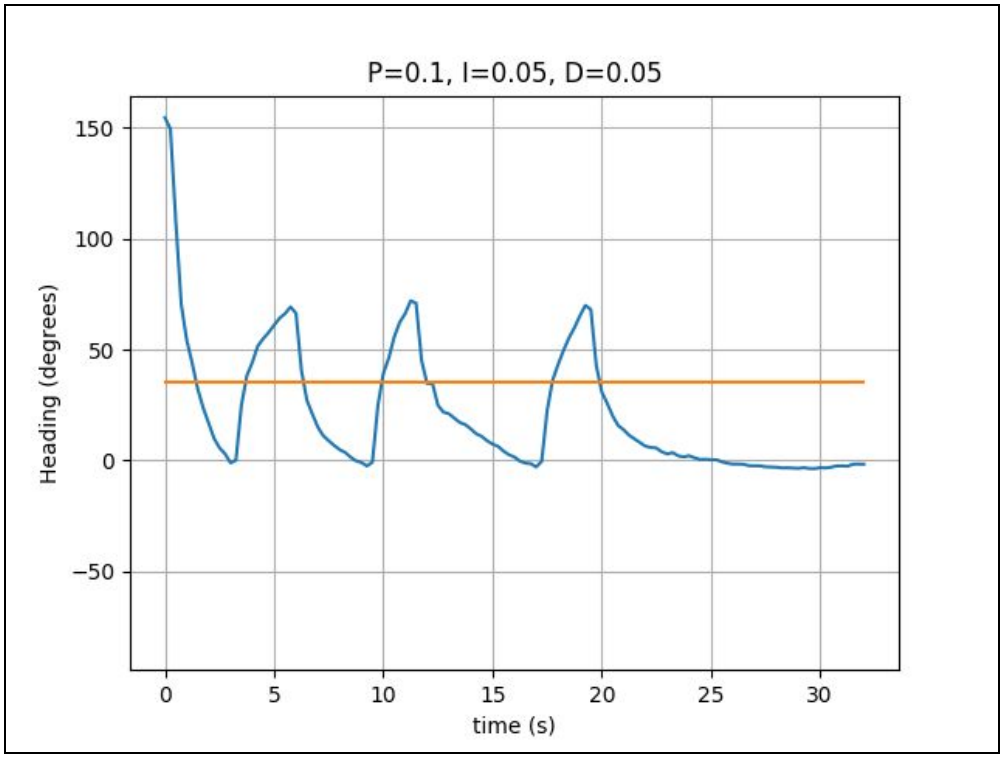



*Fig. 73. PID performance at P=0.1, I=0.05, D=0.05*

Using D=0.05 similarly produced undesirable results. Oscillations occurred once more, but at higher amplitudes and significantly longer periods than tests with D=0.075. After the third oscillation, the controller stops pulsing the motors entirely (effectively deciding that even a speed of 1 is too great), thus allowing drift and environmental forces to move the sub. This test also began with a heading difference of more than 100 degrees, whereas the previous test had a heading difference of only 50 degrees or so.

D was reduced to 0.025. Results for these parameters are shown below.

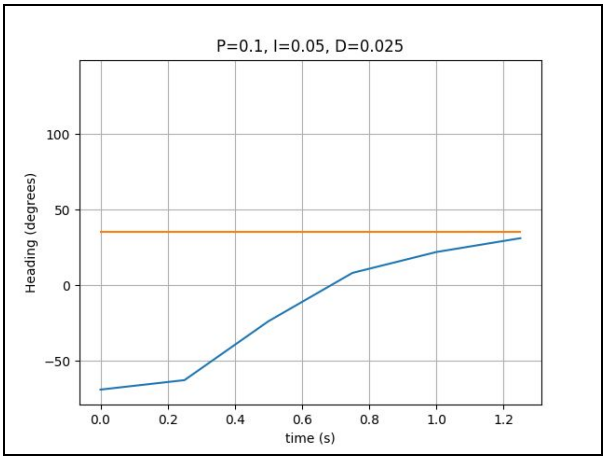



*Fig. 74. PID performance at P=0.1, I=0.05, D=0.025, test #1*

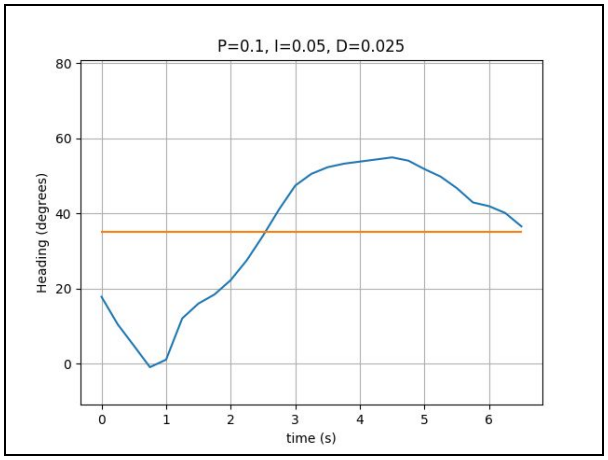



*Fig. 75. PID performance at P=0.1, I=0.05, D=0.025, test #2*

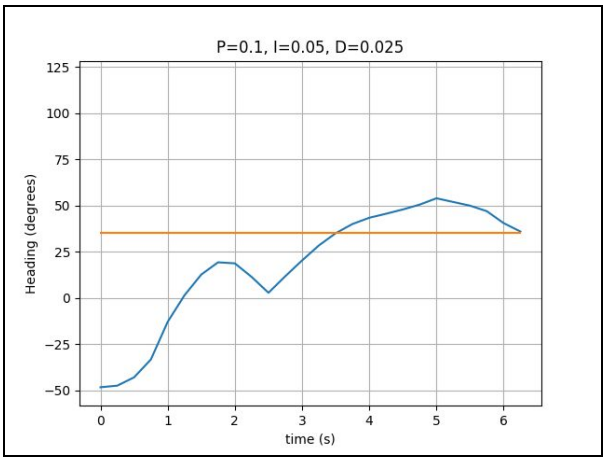



*Fig. 76. PID performance at P=0.1, I=0.05, D=0.025, test #3*

| **Test #** | 1 | 2 | 3 | Avg. |
|---|---|---|---|---|
| **Completion time (s)** | 1.3 | 6.5 | 6.2 | 4.667 |

*Table. 3. Average completion times with P=0.1, I=0.05, D=0.025*

For all tests, the controller needs no more than a single motor pulse and the final tangent line is relatively close to horizontal. Additionally, average completion time took less than 5 seconds overall, as can be seen in the above table. This provides the best parameters encountered thus far for achieving system stability. The top left figure above is especially encouraging, as the

heading difference was roughly 100 degrees and a single motor pulse was sufficient to achieve stability in a small period of time.

This is not an indication that these parameters are optimal, however they are a good starting point for future tests. Due to limited time to run tests after waiting several weeks to get access to the pool, the number of tests run thus far is small. The ATS Mk. 1 was able to hone in on target headings in many cases, but the class of motors used makes actuation rather blunt at this point- future tests may demand slower motors.

Future testing will include testing more PI configurations as opposed to PID, to see if a derivative term is needed. Beyond that, tuning the controller will include more intermediate values, instead of blocks of 0.025 used to increment/decrement parameters. Finally, testing of Ziegler-Nichols method will also be performed using $K_c = 0.2$ and an averaged $T_c$ value at many different starting headings, as it has been shown that the oscillation period can vary greatly.

Testing performed thus far has also been solely for stationary heading changes, although the majority of heading changes in a mission will be mobile. This grants more granular control of yaw change over time and thus should allow for an increase of allowable speed differences between motors.

## Discussion

Tests performed in this section utilize unideal electrical components for this application. These tests are also fairly limited, as the goal for such was to ensure correct PID implementation

and tuning- this is why stationary heading change as opposed to mobile is utilized here, as linear movement adds further complexity.

Even with these issues, our system has shown the ability to reach its setpoint with an average settling time of less than 5 seconds, which is better than our initial expectations.

Movement testing is fairly limited; as the Mk. I is limited in its abilities and has limitations as a test system for motion, other functionalities such as UWOC and Obstacle Avoidance were focused on while the mechanical engineers designed and manufactured the Mk. II. This unfortunately was not completed before winter term was concluded.

The Mk. II has a much larger footprint than the Mk. I, which results in greater drag. Coupled with a gear ratio to reduce thrust, the Mk. II should be far less reactive than the Mk. I, and concerns may arise regarding its ability to reach a target heading in a reasonable amount of time. As the gear ratio for the drivetrain is modifiable, the thrust output is customizable and can be adjusted to ensure that the Mk. II has the reactivity for this requirement.

The Mk. II also physically has all the necessary control surfaces for 3D movement- in this case, 2D movement with the ability to dive/resurface. This is due to the fact that in addition to the tank-drive propulsion system that allows for linear 2D motion and yawing, elevons are installed to allow for pitch control.

Therefore, the Mk. II contains all the control surfaces required to meet our design specifications. Based on the Mk. I's performance given its electrical design, the fact that tests were run in a fairly noisy environment (a member of our team was in the pool to document its performance which undoubtedly created some current), and the robustness of the Mk. II, meeting design specifications with this next test system should be possible.

# Dead Reckoning (performed by Xavier)

Results of the dead reckoning system unfortunately determined that this approach would not be feasible to pursue.

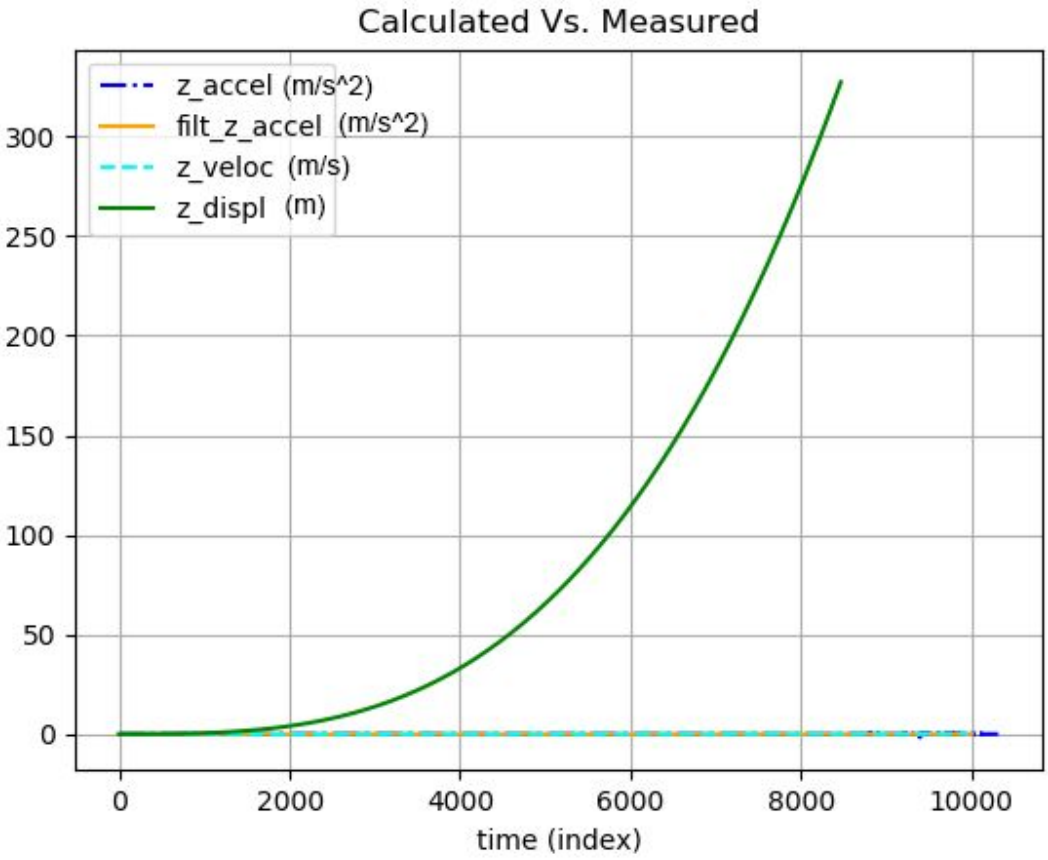



*Fig. 77. Measured, real and calculated values, stationary test*

As can be seen in Fig. 77, the displacement value increases exponentially fast, which is not desirable behavior. This implies that the driftrate/noise level in the IMU is high enough to cause this level of imprecision. To alleviate this problem, the maximum acceleration value from

completely stationary data collection, 1 $m/s^2$ , was implemented as a threshold, so any acceleration value below that would not be registered.

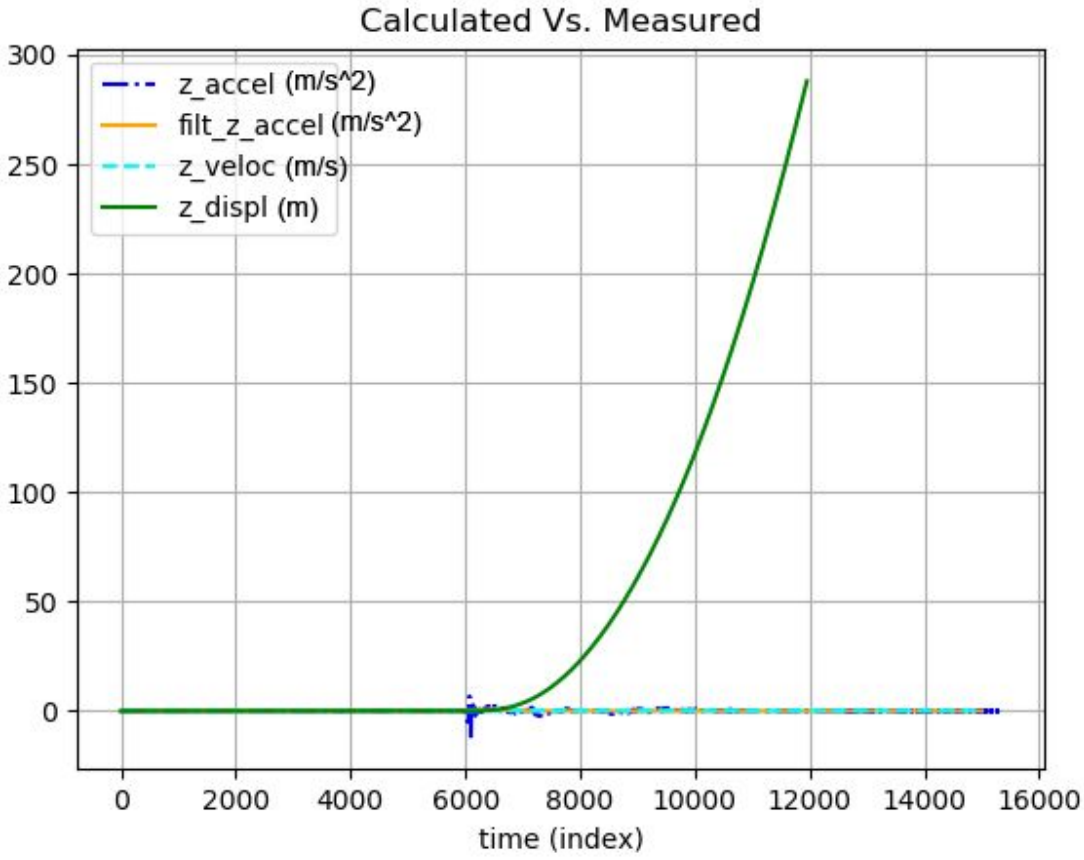



*Fig. 78. Measured, real and calculated values. Stationary, then moved, test*

Fig. 78 shows a test where the system was left static for a period of time, then picked up. As can be seen, there is no drift until the system is moved, at which point the displacement values begin to exponentially increase.

Further testing showed that the exponential incrementation could be influenced, at least temporarily, by initiating motion in one direction, and then changing to the opposite direction, but instead of returning to 0 or representing a reasonable displacement value, the exponential change continues. This effect can be seen in Fig. 79 below.

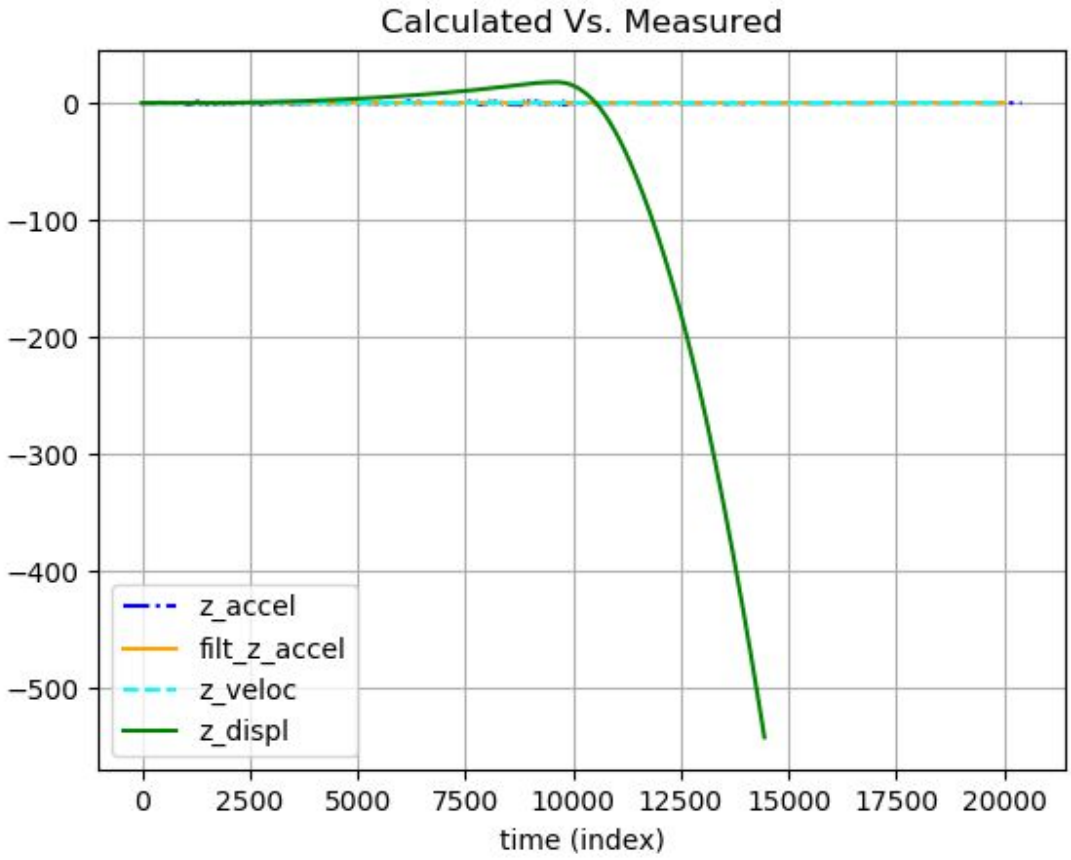



*Fig. 79. Measured, real and calculated values. Reversed motion test*

This behavior indicates sensor drift, which, according to preliminary research, is nearly impossible to correct without a baseline to reset to.

Further testing and development on this will continue for the duration of 2019, and if the results are promising, into 2020. However, if it is deemed not feasible to implement this system with the timeframe of this project, alternative solutions will be implemented.

# Communication (performed by Jacob)

## Early UWOC Component Testing (white light)

This subsection details early work performed in gauging performance of optical communication components. Specifically an SFH 310 900nm phototransistor and a 3W white power LED are used, as described in the *Design* section, Communication block. These components do not operate in the wavelength necessary for underwater wireless optical communication, but were used for general prototyping. Neither of the datasheets for these products include an average rise/fall time, so the maximum pulse rate at which the phototransistor can still receive full saturation could not be computed mathematically. Therefore, a small test was run using a signal generator, oscilloscope, DC power supply, an LED driving circuit and a phototransistor circuit (see the *Design* section for a schematic of these circuits). LEDs were pulsed at various frequencies and the voltage across the phototransistor was read out.

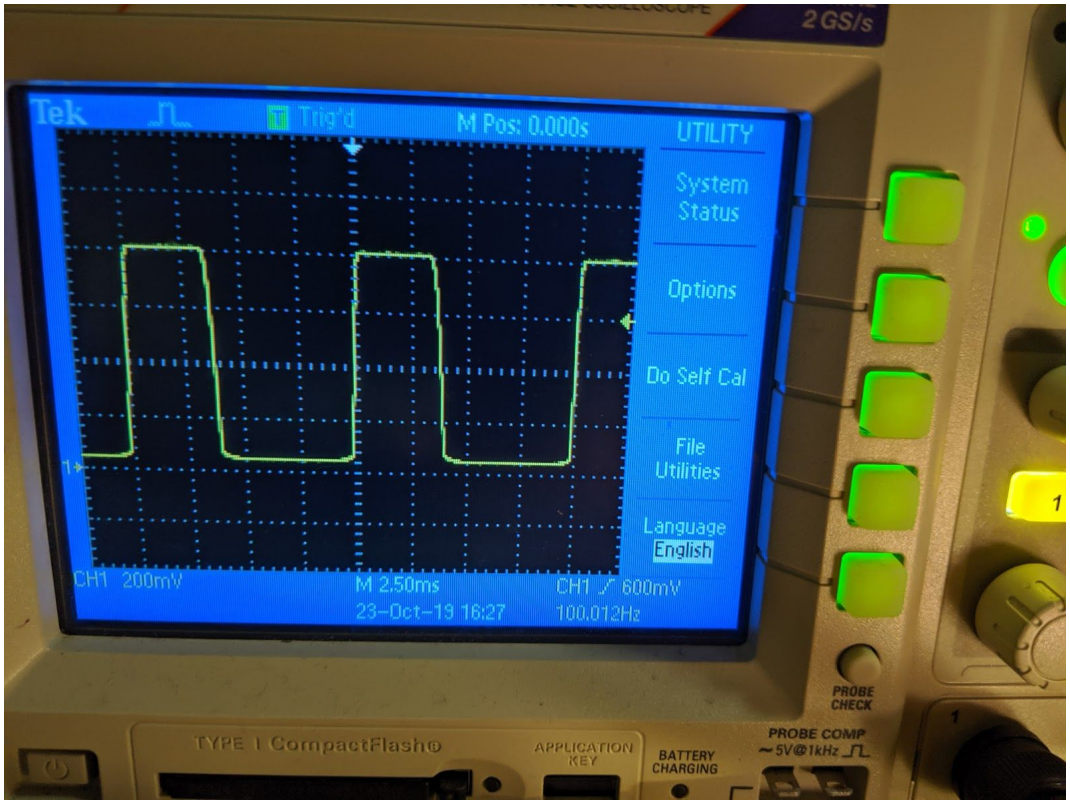



*Fig. 80. Optical pulsing at 100 Hz*

At 100 Hz, the square wave is read out without issue and the waveform is relatively rectangular. Rounding appears to be more of an issue on the falling edge, which indicates that the LED fall time may be the hindering factor as LEDs typically have quicker rise times than fall times[47].

[47] High-speed switching of IR-LEDs — Background and data sheet definition

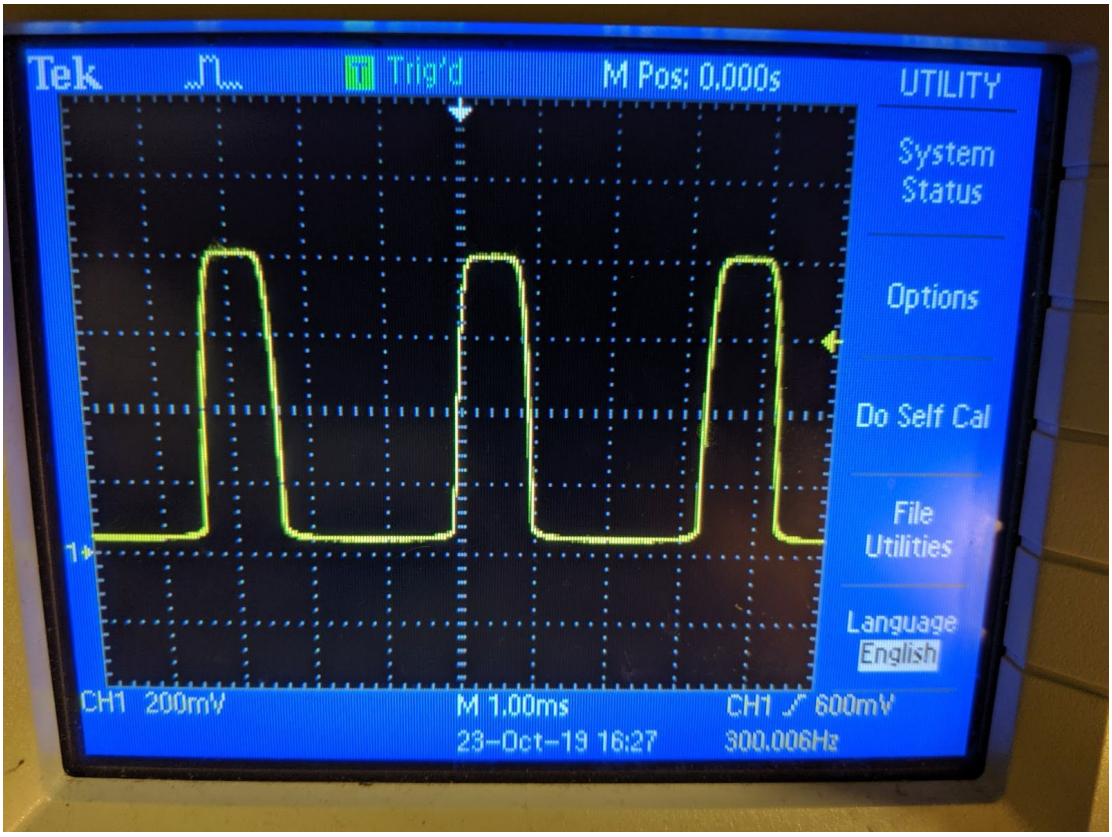



*Fig. 81. Optical pulsing at 300 Hz*

At 300 Hz, rounding is noticeable on both rising and falling edges, and the time delay is roughly even for both (0.4 µs). Potentially, the phototransistors' rise time is becoming more visible.

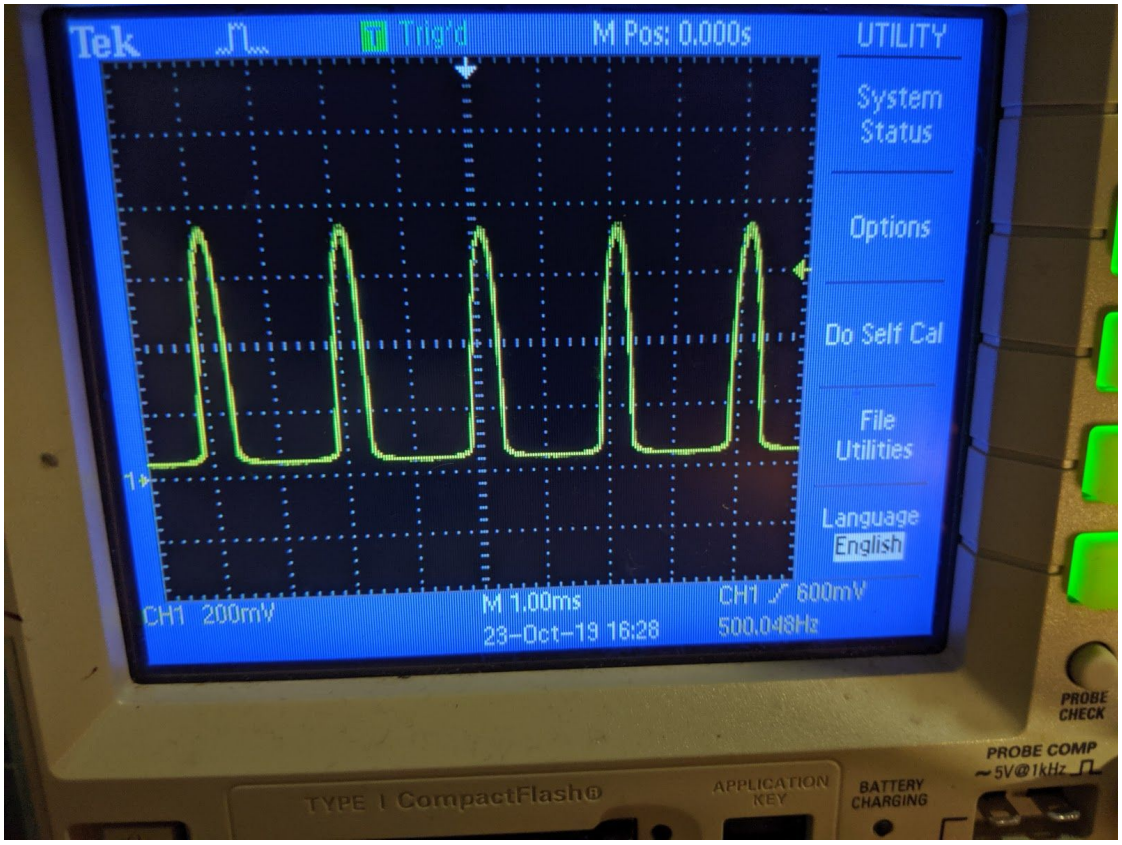



*Fig. 82. Optical pulsing at 500 Hz*

At 500 Hz, the pulses are almost completely rounded at maximum voltage. After 500 Hz, the saturation voltage cannot be achieved by the phototransistor. While this is not strictly necessary for communication, the high level of optical signal attenuation makes it especially important to keep the maximum voltage achievable at the receiving end as high as possible. Therefore, for the white light system, a minimum period between pulses of 4 ms is defined.

Tests were also performed to determine what the phototransistor voltage was at various distances to the LED. This is shown below.

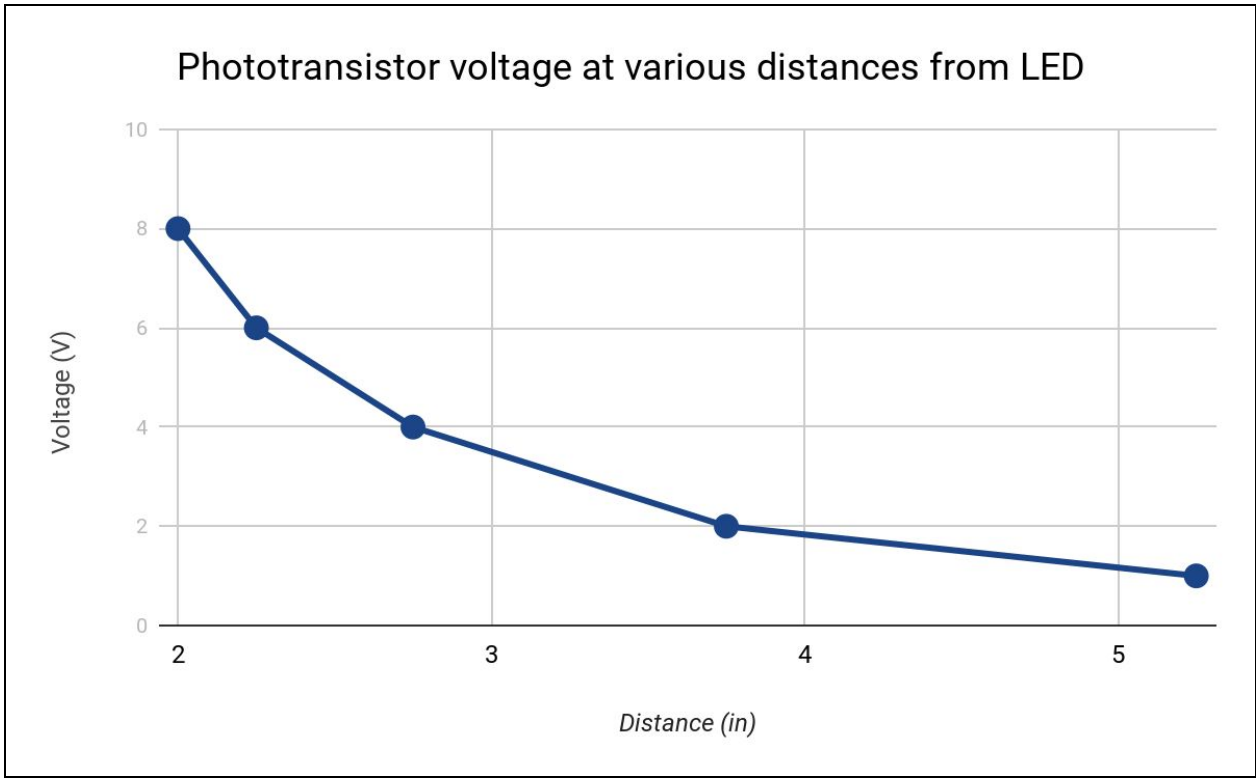



*Fig. 83. Phototransistor voltages vs. distance for white light system*

As can be seen, the signal strength is roughly 1/8th of maximum at a little of 5 inches away. It must be mentioned that the wavelengths of the LED and phototransistor are not very compatible; the peak wavelength of sensitivity of the phototransistor corresponds to about 0.6 at maximum for the relative spectral power distribution at the LED, and is largely scattered. For the components selected for optical communication, this number is at least 0.95 with a much smaller bandwidth and the LED draws approximately four times as much current.

These components have arrived but have yet to be extensively tested. Early indications show that a minimum pulse width of 2 ms is achievable, half of that for the white light system.

## Final UWOC Component In-air testing

Once the selected components for UWOC arrived, an early set of tests were run to ensure their functionality and to roughly gauge and characterize performance. A stationary receiver (phototransistor + comparator circuit) was positioned at one end of the testing environment and the non-inverting input voltage (V_compare) was set. A mobile transmitter (high power LED + driver) was positioned at the furthest distance from the receiver possible such that the comparator output was still high (i.e. a signal can still be successfully received/digitized). This was repeated for a range of V_compare values, and in both light and dark conditions.

It should be addressed that “light” and “dark” conditions can be arbitrarily defined as “lights on” and “lights off” in our testing environment. The proper method of defining these environments would be to determine the illuminance (lux) level at the phototransistor in these two conditions using an ambient light sensor. As will be shown below in the UWOC Tank Results subsection, a light sensor has been used for testing in this project, however this sensor arrived long after in-air testing was completed and thus only later tests include SI units. Earlier in-air tests were going to be re-setup for the purpose of measuring the illuminance in both lighting conditions, however events near the end of the capstone term prevented these tests from happening.

These results of these in-air tests can be found below, where the red line represents bright condition results and the blue line represents dark condition results.

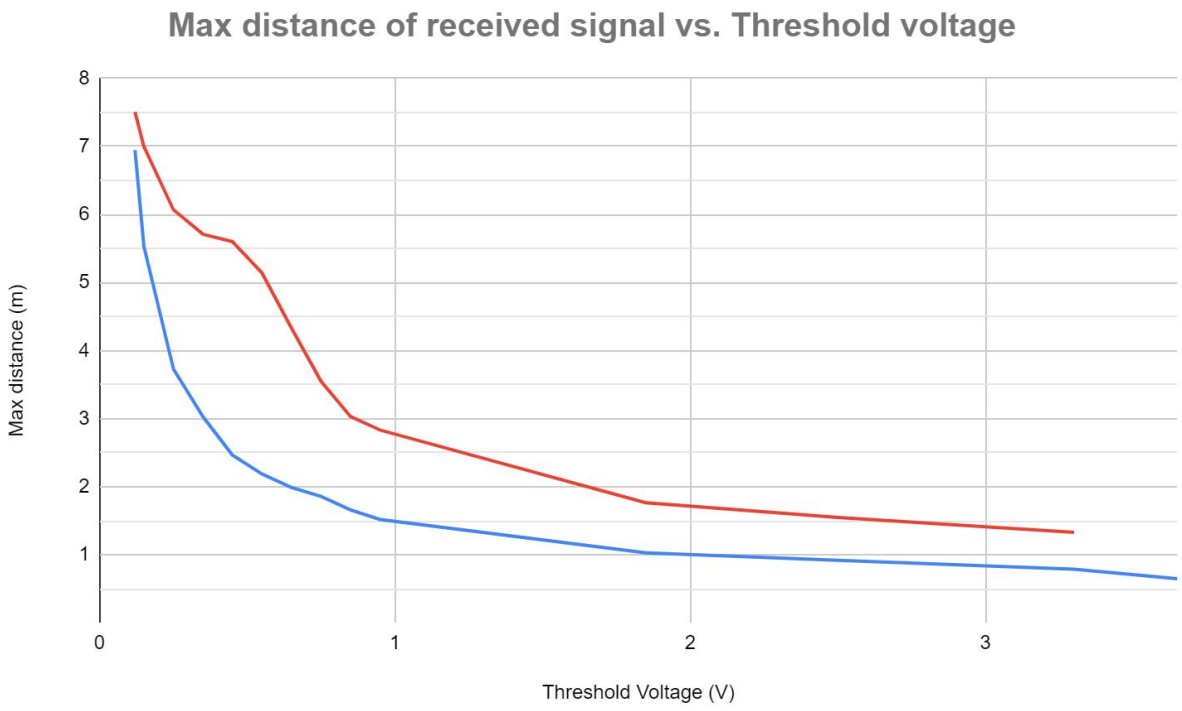



*Fig. 84. Distance vs. V_compare voltage for bright/dark lighting, in-air*

As can be seen above, results vary in different lighting conditions. This points to nonlinear regions of sensitivity in the phototransistor- specifically, ambient lighting allows for greater maximum transmission distances compared to no ambient lighting. It should be briefly mentioned that the maximum distance reached during light conditions is actually the maximum dimension of the testing environment; in reality, this number may be far larger than ~7.5 meters.

With this new information, insights into the performance of this communication system in various environments can be gained. For example, if low ambient light levels are detected (i.e. the flock travels under an ice sheet), then a tighter flock configuration will prevent frames from being lost due to range reasons.

## UWOC Tank Results

The UWOC tank is used to gauge communication system performance in a controlled, submerged environment. Controllable parameters include ambient light level, V_compare (or non-inverting input of comparator) voltage, transmission distance, and all aspects of the data frame itself.

The goal of this result set, aside from development/reiteration of the communication system, is to create a lookup table where an illuminance level is input and BER and V_compare are outputs. This will simplify future testing in foreign environments, as a single ambient light level measurement will be taken once and this will provide the V_compare that the sub must set, as well as expected performance.

A brief overview of the test methodology is offered here. An ambient light level is selected and the tank brightens to this light level. V_compare is then set so that the comparator is not driven high due to ambient lighting, however pulses from the transmitter can still be received- V_compare should be as low as possible while still satisfying these conditions. Next, the maximum transmission distance is determined by setting the transmitting LED high and increasing the transmission distance until the receiver cannot receive the signal or the absolute maximum transmission distance (1 meter) is reached (note: for these tests, all transmission distances are 1 meter). Frames of random length (up to 100 bytes) are then sent with random "break" periods (up to 1 second) in between transmissions, until 100 frames have been sent. The bit error rate is then calculated and stored, and the process repeats for another ambient light level.

Results from tank tests are shown below.

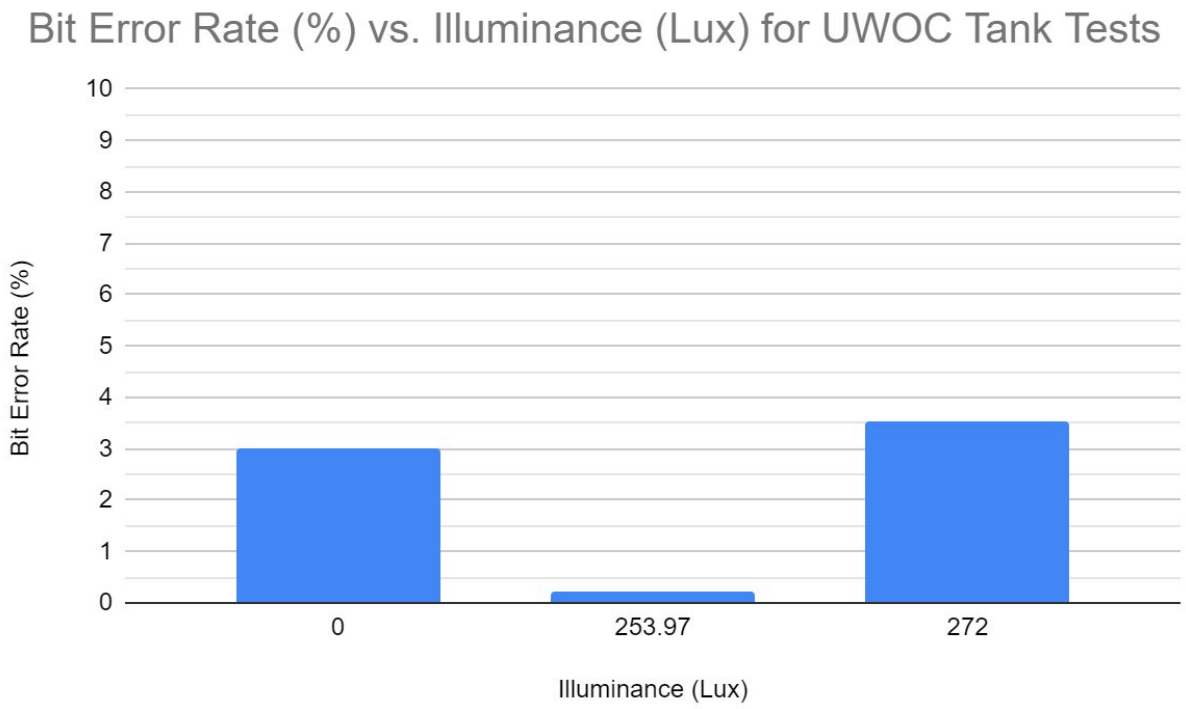



*Fig. 85. Bit Error Rate vs. Illuminance for UWOC tank tests*

As a brief reminder, the original specifications for our communication system are: less than 10% BER, transmission distance of at least 0.5 meters, and data rate of at least 62.5 bps. For our system, a data rate of 250 bps is utilized and results captured above were taken with a transmission distance of 1 meter with an average BER of 2.25%. Therefore, this communication system exceeded design requirements by a large margin.

It should be noted that for these tests, there is no mechanism to test various distances or V_compare voltages. The goal of these tests are to determine the average BER for this communication system, and thus all controllable parameters are fixed. Future tests may include maximum transmission distance w.r.t V_compare level in a submerged environment, similar to what was done in the Final UWOC Component In-air testing section.

## Discussion

UWOC tank tests have exceeded expectations in BER, transmission distance and data rate, however the original specification includes turbidity as an environmental factor, which is omitted here. Due to budgetary constraints in acquiring a turbidity meter, time constraints in finding a solution and quantity to produce a known turbidity in water, and logistical constraints with our submarine testing environment being a campus-wide swimming pool that we are not allowed to make turbid, this environmental factor was ignored for these tests. The National Sanitation Foundation sets a maximum turbidity for swimming pools of 0.5 NTU[48], which is 1/20th of the original specification's turbidity level (10 NTU). Therefore, this factor remains unknown and may be a large issue for this communication system. The wavelength chosen for transmission was selected due to its resilience to varying levels of attenuation, and the results presented in the UWOC Tank Results subsection display far better performance than imposed by the original design specifications, all of which allow room for adjustment if the current test parameters and configuration cannot perform adequately in a turbidity of 10 NTU.

# Obstacle Avoidance (performed by Xavier)

The sonar module with the final circuitry only made it to the air testing stage. Due to the significantly greater speed of sound in water than in air (~1480m/s vs ~344m/s respectively), a significantly larger testing area is required for in water tests to avoid the transmission signal overlapping the received signal. This issue puts a low end measurement limitation based upon

[48] http://www.nsf.org/media/enews/documents/nsf_50_150715.pdf

the speed of sound in a particular transmission medium. Due to this constraint, it was determined to be prudent to perform in air testing before moving onto a medium with more variables.

The in air test setup involved a rapidly prototyped waveguide for both the transducer and hydrophone. This provides significantly more for directionality than amplification.

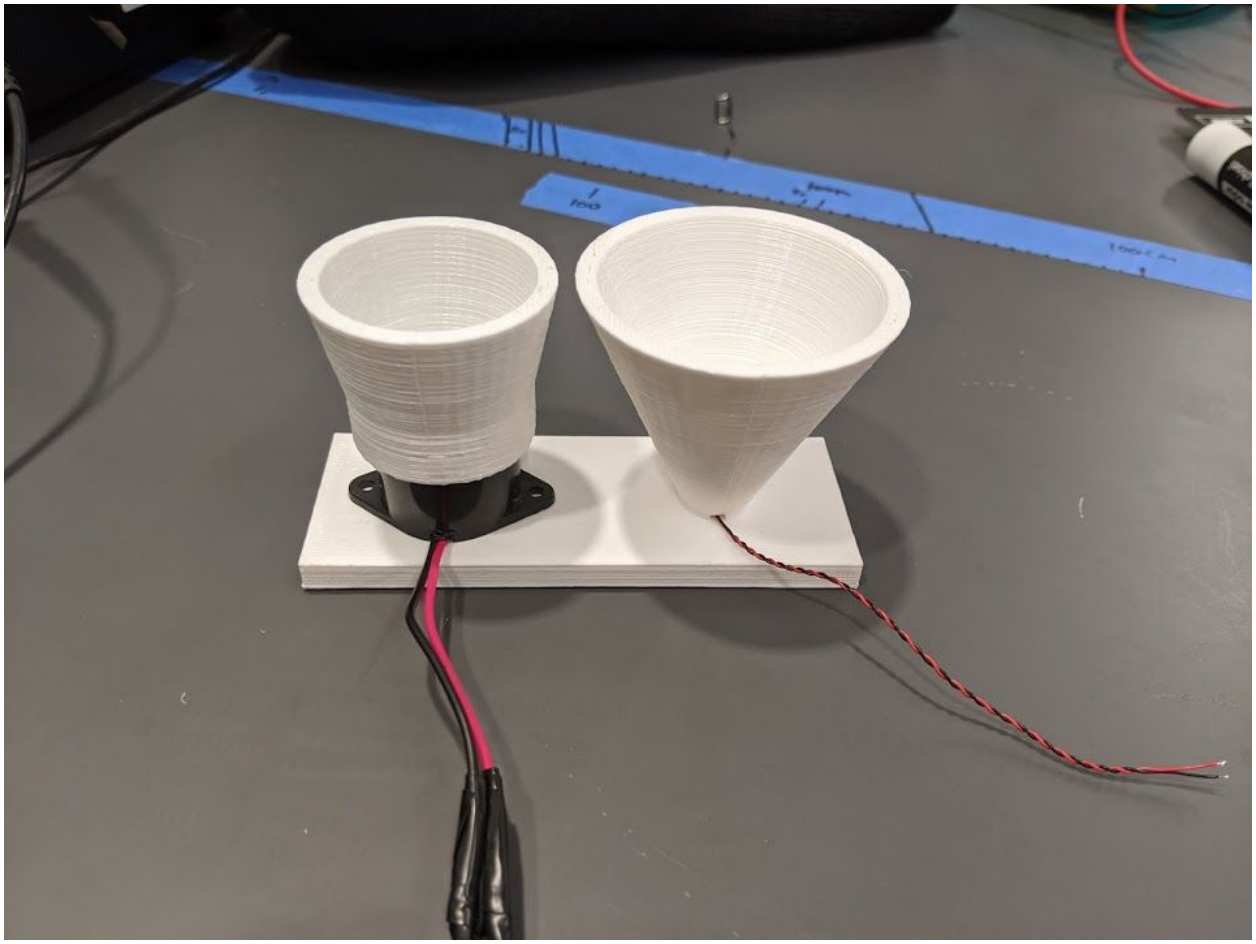


*Fig. 86: 3D printed sonar module test rig with waveguide*

The module depicted in Fig. 86 was mounted to a linear slide, allowing precise vertical positioning from the ceiling. This allows for repeated measurement at known distances.


*Fig. 87: Received echo. Yellow is the trigger for the transducer, blue is the received and filtered signal, and purple is the comparator output.*

In Fig. 87 a successfully received echo can be seen with a time difference of 16ms between the sending of the signal and the receiving of the signal. This test was performed at a distance of 2.2m from the ceiling, and upon plugging in the transmission speed (16ms/1000ms)*(344m/s)=5.5m, or just about twice the distance to the ceiling. This system successfully detects obstacles at greater than 1.5m, matching the design specification. Unfortunately, as the system approaches the lower limit of its range, the pulse from the echo overlaps the pulse from the transmission enough that the comparator is unable to differentiate the

signals. As can be seen in Fig. 88 when the round trip distance is brought to less than ~5m, the results begin being non-linear.

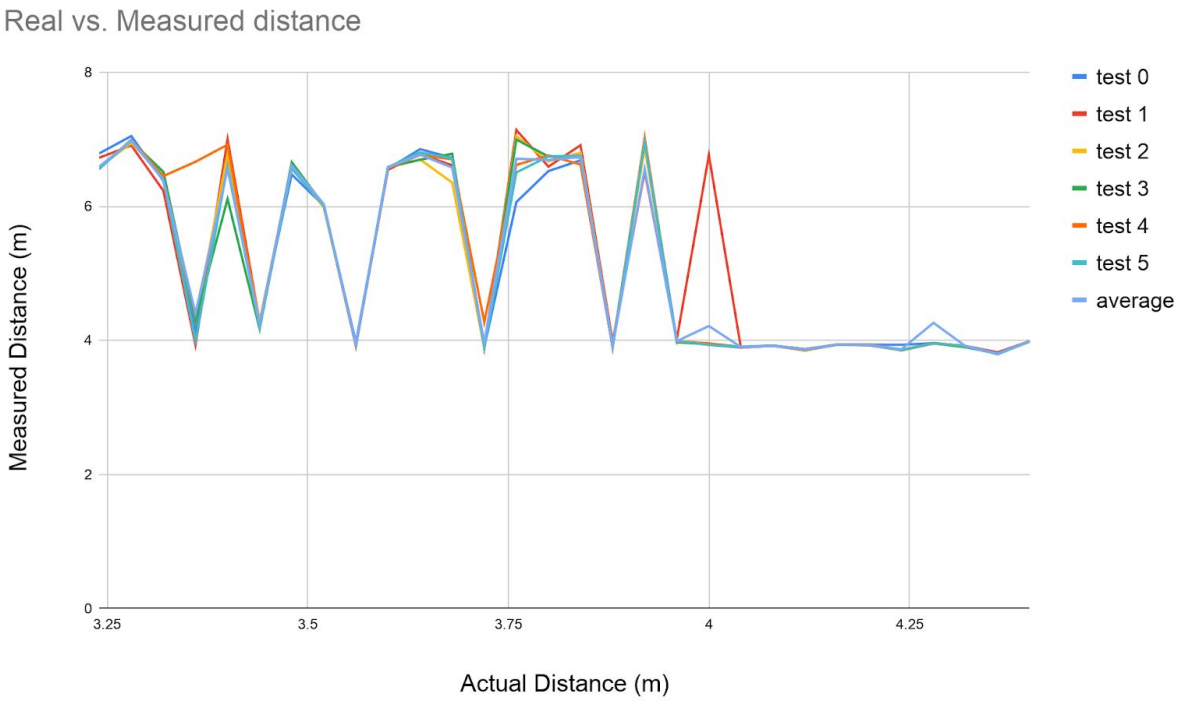



*Fig. 88: results of too close sonar module.*

This issue scales in water, and a large enough body was unable to be acquired before the end of this section of the project. Further testing will be completed in a large body of water, but solutions to the echo overlap will be researched as well. Possible solutions include active suppression based on feedback from the transmission, higher frequencies, which would allow for decreased ping durations and therefore less feedback.

# Production Schedule

## MKI phase

Testing relating to the further development of Motion Data and Captain will be performed in the remaining weeks of 2019, before the completion of the MK II. Simultaneously, testing of the optical communication system and the local awareness sensors will be performed on their own test beds to ensure they are ready for implementation into the MKII. During this phase, the evaluation of using an IMU based displacement sensor will be finalized, and alternative designs will be selected if deemed unfeasible.

## MKII phase

The completion of the MKII, scheduled for the first two weeks of 2020 bar any mechanical setbacks, will include 3-space navigation capabilities as well as communication and local awareness hardware. Testing, integration and additional development of the code designed in the previous phase will be completed over a couple weeks following the systems completion. The remainder of the time before the completion of the MKIII will be dedicated to building up the environment in which all code sections communicate and evaluation of the final hardware component selection.

The completion of the MKII, initially scheduled for the first two weeks of 2020, will include 3-space navigation capabilities as well as communication and local awareness hardware. This iteration will be as modular as possible given the restrictions, which will be completed by having removable wire connectors to the outside of the hull, allowing for rapid replacement of the external systems to test, such as sonar and optical communication modules and new iterations of the thruster design. Testing, integration and additional development of the code designed in the previous phase will be completed over a couple weeks following the system's completion. The remainder of the time before the completion of the MKIII will be dedicated to building up the environment in which all code sections communicate and evaluation of the final hardware component selection. Unfortunately there were setbacks on the mechanical engineering aspect of this, and not enough aspects of the system were finalized by the end of winter term to allow for this to happen. During this time, the computer engineering team worked on completing as much of the other modules as could be without a hull for testing.

## MKIII phase

The MKIII, which should exhibit all described design aspects, will be constructed with durability over modularity, as by this point all external components should be finalized. As the MKII will have the same motion and sensor capabilities, minimal refactoring will be required for this transition. Testing, modification, verification and documentation will occur during this time period. Once the hardware has been finalized, at least one other AUV would be constructed to work on the flocking aspect of the AFμS project.

Due to the outbreak of COVID-19 and the rapid disbandment of the team for this project, it is at present unknown how close to this goal it will be possible to realize.

## Future work

At the time of the submission of this paper, the team members are planning on working remotely on all of the sections that they can, with the aim of completing as much as possible without being in the same location as an active prototype.

# Conclusions

Upon the genesis of this project, all involved team members were fully aware that the goals set greatly exceeded the scope of a capstone project, and therefore a completed product would be very unlikely to be finalized during the given timeline. To this end, all team members agreed early on to continue this project for an extra term with the aim of completing more than would have been possible otherwise. Knowing that the whole system would not be completed, significant effort was invested early on in attempts to achieve as much as possible. The first instance of this was a team ideation session with the aims of ensuring a clear design to work towards that everyone agreed upon.

Another decision based on the scale of this project was the implementation of a minimal viable product strategy, meaning that there would always be a prototype to develop on. This approach is conducive to having a proof of concept of the system regardless of how much of the overall system could be achieved in the alloted timeline. This methodology ended up being

beneficial to the cross-departmental teamwork effort, as having periodic cross-team goals ensured all members stayed on the same timeline, and allowed for members to work on development of other systems if aspects for the next prototype were falling behind. This allowed for improved overall efficiency of time usage, however even with all of these efforts, the amount of time for completing certain aspects were grossly underestimated. Due to this, prototypes which were scheduled to be completed within the first two terms were only being finished up at the end of the third term. While, retrospectively, it does not seem like these could have been completed significantly earlier, our timeline estimates could have been significantly more accurate.

The production of the final AUV iteration was planned to be completed during the fourth term of this project, however with the disbandment of the team due to the global pandemic, it is unsure at this point what will be able to be completed with the available resources.

# References

M. E. G. Mital *et al*., "Characterization of underwater optical data transmission parameters under varying conditions of turbidity and water movement," *2017 5th International Conference on Information and Communication Technology (ICoIC7)*, Malacca City, 2017, pp. 1-6.

Johnson, L. J., Jasman, F., Green, R. J., & Leeson, M. S. (2014). Recent advances in underwater optical wireless communications. Underwater Technology: International Journal of the Society for Underwater, 32(3), 167–175. doi: 10.3723/ut.32.167

Brundage, H. (2010). Designing a wireless underwater optical communication system. Mechanical Engineering - Master's Degree. Retrieved from http://hdl.handle.net/1721.1/57699

Tinker Board S: Single Board Computer. (n.d.). Retrieved from https://www.asus.com/Single-Board-Computer/Tinker-Board-S/.

S. M. Smith and D. Kronen, "Experimental results of an inexpensive short baseline acoustic positioning system for AUV navigation," Oceans '97. MTS/IEEE Conference Proceedings, Halifax, NS, Canada, 1997, pp. 714-720 vol.1.

J. Snyder, "Doppler Velocity Log (DVL) navigation for observation-class ROVs," OCEANS 2010 MTS/IEEE SEATTLE, Seattle, WA, 2010, pp. 1-9. doi: 10.1109/OCEANS.2010.5664561

J. Sticklus, P. A. Hoeher and R. Röttgers, "Optical Underwater Communication: The Potential of Using Converted Green LEDs in Coastal Waters," in IEEE Journal of Oceanic Engineering, vol. 44, no. 2, pp. 535-547, April 2019.

(n.d.). Retrieved from https://www.nde-ed.org/EducationResources/CommunityCollege/Ultrasonics/EquipmentTrans/characteristicspt.htm.

Butler, J. L., & Sherman, C. H. (2018). Transducers and Arrays for Underwater Sound. Cham: Springer International Publishing.

Ainslie M. A., McColm J. G., "A simplified formula for viscous and chemical absorption in sea water", Journal of the Acoustical Society of America, 103(3), 1671-1672, 1998.

Fish, F.E.; Schreiber, C.M.; Moored, K.W.; Liu, G.; Dong, H.; Bart-Smith, H. Hydrodynamic Performance of Aquatic Flapping: Efficiency of Underwater Flight in the Manta. Aerospace 2016, 3, 20.

Font D, Tresanchez M, Siegentahler C, et al. Design and implementation of a biomimetic turtle hydrofoil for an autonomous underwater vehicle. Sensors (Basel). 2011;11(12):11168–11187. doi:10.3390/s111211168

Bucz, Š., & Kozáková, A. (2018). Advanced Methods of PID Controller Tuning for Specified Performance. PID Control for Industrial Processes. doi: 10.5772/intechopen.76069

Ziegler, J. G., & Nichols, N. B. (1993). Optimum Settings for Automatic Controllers. Journal of Dynamic Systems, Measurement, and Control, 115(2B), 220–222. doi: 10.1115/1.2899060

Salih, Atheer L., et al. "Flight PID controller design for a UAV quadrotor." Scientific research and essays 5.23 (2010): 3660-3667.

Homebuilt Side Scan Sonar. (2010). MBT Electronics. https://www.mbtelectronics.com/side-scan-sonar.php

Joy, K., Hamilton, J., Jewell, M., & Babb, I. (2016). Simple Hydrophone Design. Simple Hydrophone Design. https://www.nurtec.uconn.edu/wp-content/uploads/sites/287/2016/08/COSEE-TEK-Simple-Hydrophone-Material-List-Fabrication-Instructions-V4.2-7-7-2016.pdf

Grzinich, J. (2016, December 9). do-it-yourself hydrophones. john grzinich. https://maaheli.ee/main/d-i-y-hydrophones/

B. Benson et al., "Design of a low-cost, underwater acoustic modem for short-range sensor networks," OCEANS'10 IEEE SYDNEY, Sydney, NSW, 2010, pp. 1-9.

Won TH, Park SJ. Design and implementation of an omni-directional underwater acoustic micro-modem based on a low-power micro-controller unit. Sensors (Basel). 2012;12(2):2309–2323. doi:10.3390/s120202309

Søreide, Fredrik (2011-04-28). Ships from the Depths. ISBN 9781603442183

# Appendices

## A) Team dynamics

While the parallelization of work is a significant advantage that stems from working in a team, a great deal of effort must be exerted to ensure that the team can function as an efficient unit. As the scope of the course is not designed to accommodate or teach methodologies conducive to this situation, significant research was performed at the start of the project with the aim of proactively avoiding future problems. As all members of the teams are students, it is not feasible for any individual to spend the majority of their time on this project, unlike the majority of workplace environments. This meant that all research methodologies needed to be adapted to our situation, and certain aspects would need to be developed or adapted to match the environment.

The first implementation of research methodologies was a group ideation session where the critical design features were determined by analysis of current aquatic research efforts, currently available AUV solutions, and aspects judged to be important to customers. Many of the relevant methodologies from here[49] and here[50] were adapted to apply towards the domain of this project. This had the dual benefit of resulting in a well thought out project in regards to what is available in the field, as well as ensuring that all team members were on the same page regarding

[49] https://medium.com/the-creative-founder/needfinding-for-disruptive-innovation-71d8532f2cf3
[50] https://medium.com/the-creative-founder/ideation-sprints-for-new-products-services-74f925190b4f

the project and its core features, which would remain an important factor for the duration of its development. Photos of this session are available in *Appendix B*.

To keep everyone on the same page, we decided to adopt the idea of weekly sprint meetings where we would inform each other of what we have accomplished since the previous meeting, and what we planned to accomplish before the next meeting. To further this goal, it was mandated that all research and documentation is stored in a shared cloud based environment, allowing all members to quickly catch themselves up on the progress of other team members, and the easy sharing of documents and ideas.

Having two distinctive groups from different departments provided more challenges than initially anticipated. The most evident of these problems is that this effectively creates two separate projects that need to be meticulously synchronized in their development. To assist in this, a shared document was generated where each team could designate a project for the other team to work, and then fill out needs and wishes. There is then a section where the other team can leave notes on each of these aspects, either to reference or to request clarification. This was to supplement rather than replace discussion during the weekly meetings.

The unforeseen issues stemmed from the way members of each department have been trained to think. This has shown to be beneficial in some cases, for example there have been design issues that one team was struggling to solve and the other was able to come up with a solution rapidly due to a different perspective. However, it has been the case multiple times that there was a disconnect between teams due to assumptions of common knowledge. For example initial issues on the MKI and the spinning test rig were due to the mechanical team overlooking the physical dimensions of wires, something that the other team took for granted as a constraint.

All members have learned to be careful when communicating with the other team to ensure that optimal comprehension is achieved.

To further the efficiency of working in groups, we have made sure that as many aspects of the AUVs components can be duplicated, allowing parallel development and testing by different members of the team. An example of this is that the MKI will be duplicated, in full or in part, so both members of the Computer/Electrical team can continue work over winter break.

# B) Media

## Uncurated live updated photo gallery of work to date:

https://photos.app.goo.gl/6BxMWJuVAo57jb7g8

## Media from pool tests:

https://drive.google.com/drive/folders/18NZGOvMQ7kgKeahAzZrMWY-nTwBu4vXU?usp=sharing

# C) Meeting Log

An attempt was made to document all team meetings, though inevitably there were several situations where documentation was forgotten. Below is an informal meeting log.

| When | What | Who | Notes |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 5/14/2019 | Ideation Session | Alex, Jacob, Sam, Xavier | See Photos |
| 9/12/2019 | Initial Piezo Testing | Jacob, Xavier | See Photos |
| 9/13/2019 | Piezo testing with driver and hydrophone prototypes | Jacob, Xavier | See Photos |
| 9/18/2019 | Group progress and planning meeting | Alex, Jacob, Sam, Xavier | |
| 9/19/2019 | CpE specification meeting | Jacob, Xavier | Discovered transducer housing development issues |
| 9/19/2019 | Call with Scott about Transducers | Jacob, Sam, (Scott), Xavier | Relevant document here |
| 9/20/2019 | Setting up a pi environment with needed software and file management. Connecting the BNO055 IMU to test base functionality | Jacob, Xavier | Pi chosen, raspbian installed, scripts added to path, IMU packages installed. There may be |

| | | | |
|---|---|---|---|
| | | | communication limitations on the pi with I2C, so we shall see |
| 9/22/2019 | Discussing timeline, Developed Gantt Chart | Alex, Jacob, Xavier | Sam was unable to make it, which resulted in some drawbacks in our overview of planning |
| 9/24/2019 | SRG writing meeting | Alex, Jacob, Sam, Xavier | Also debriefed on Walt Dixons insights on the Gantt chart and the rest of the project |
| 10/1/2019 | Meeting with ME professors | Alex and Sam | |
| 10/4/2019 | Full team meeting | Alex, Jacob, Sam, Xavier | Caught up all team members on cross |

| | | | |
|---|---|---|---|
| | | | departmental needs |
| 10/6/2019 | IO and specifications meeting | Jacob, Xavier | |
| 10/7/2019 | Discussion of optical receiver requirements and potential component selection | Jacob, Xavier | |
| 10/15/2019 | Team meeting, green grant proposal | Alex, Jacon, Sam, Xavier | |
| 10/26/2019 | Sonar, delegation, general procedure meeting | Alex, Sam, Xavier | |
| 10/27/2019 | Sched_deadline and interrupt meeting | Jacob, Xavier | |
| 10/30/2019 | Prop optimization meeting | | |
| 10/31/2019 | Poster planning meeting | Jacob, Xavier | |
| 11/5/2019 | Total power draw estimation meeting | Jacob, Xavier | |
| 11/7/2019 | Team meeting, planning out the rest of the term, | Alex, Jacob, Sam, Xavier | |

| | | | |
|---|---|---|---|
| | ensuring MK2 will have required features | | |
| 11/12/2019 | First pool test | Jacob, Xavier | |
| 11/13/2019 | Meeting to discuss the results and issues encountered during the first pool test | Alex, Jacob, Sam, Xavier | |
| 11/18/2019 | Second pool test | Jacob, Sam, Xavier | |
| 11/20/2019 | Third pool test | Alex, Jacob, Xavier | |
| 11/22/2019 | Fourth pool test | Alex, Jacob | |
| 11/23/2019 | Design report meeting | Jacob, Xavier | |
| 1/18/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |
| 1/25/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |
| 2/1/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |
| 2/8/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |
| 2/15/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |
| 2/22/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |
| 2/29/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |
| 3/7/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |

| | | | |
|---|---|---|---|
| 3/14/2020 | Full team lab meeting | Alex, Jacob, Sam, Xavier | |

# D) Code

## Movement

### ats_captain.py

```
import logging
import time
import PID
from collections import deque
from math import sqrt

from captain.engine.ats_engine import AtsDualThrusterEngine


"""
Class responsible for taking in motion instructions and creating
motion.
"""
class AtsDualThrusterCaptain:

    def __init__(self, motion_analyzer, l_motor_gpio:int,
r_motor_gpio:int, speed:int=10):
        # constants
        self.motor_factor = 0.25  # scale of motor power
        self.dist_tolerance = 0.5  # in meters, distance
        self.heading_tolerance = 3.6  # in degrees
        # instantiated at runtime
        self.motion_analyzer = motion_analyzer
        self.engine = AtsDualThrusterEngine(l_motor_gpio,
r_motor_gpio, self.motor_factor)
        self.l_motor = l_motor_gpio
        self.r_motor = r_motor_gpio
        self.stop_motors = {l_motor_gpio: self.engine.min,
r_motor_gpio:self.engine.min}
```

```
        self.speed = speed if (speed <= self.engine.max and
speed > 0) else 50
        self.heading_PID = None
        self.dist = 0  # desired forward distance
        self.heading = None  # desired heading
        self.is_active = False
        self.stop = False

        logging.info("motor_factor = %s" %
str(self.motor_factor))
        logging.info("dist_tolerance = %s" %
str(self.dist_tolerance))
        logging.info("heading_tolerance = %s" %
str(self.heading_tolerance))
        logging.info("speed = %s" % str(self.speed))

    """
    Returns the direction that the sub is moving in. Based on
the sign of
    self.speed.
    ===
    Returns:
    - 1 if moving forward
    - 0 if stationary
    - (-1) if moving backward
    """
    def _get_direction(self):
        if self.speed > 0:
            return 1
        if self.speed < 0:
            return -1
        else:
            return 0

    """
    Converts output of PID into motor values. Output of PID is
the DIFFERENCE
    in heading to strive for, not the absolute heading to
attempt. Values
    should get smaller as the sub nears its target heading.
    ===
    Inputs:
    - heading_diff: Adjustment to be made for heading (PID
output).
      - (+) positive value means sub must yaw right
```

```
      - (-) negative value means sub must yaw left
    - stationary: Whether sub is changing heading while
stationary.
    Returns:
    - Tuple of adjusted motor values. Format: l_motor, r_motor
    """
    def _pid_output_to_heading_adj(self, heading_diff:float,
stationary:bool=False):
        # Max drive if heading diff is more than ceiling
        heading_diff_ceiling = 10
        # max motor val difference allowable
        motor_diff_ceiling = 4

        if stationary is True:
            if heading_diff <= -heading_diff_ceiling:
                # needs to yaw left
                return -motor_diff_ceiling/2,
motor_diff_ceiling/2
            elif heading_diff >= heading_diff_ceiling:
                # needs to yaw right
                return motor_diff_ceiling/2,
-motor_diff_ceiling/2
            else:
                drive_val =
(heading_diff/heading_diff_ceiling)*(motor_diff_ceiling/2)
                return drive_val, -drive_val
        else:
            # if chain determines if sub is moving forward or
backward
            if self.speed > 0:  # Case 1: moving forward
                if heading_diff < 0:  # Case 1A: yaw left
                    if heading_diff <= -heading_diff_ceiling:  #
Case 1Aa: yaw left, full speed
                        motor_diff = motor_diff_ceiling
                    else:  # Case 1Ab: yaw left, intermediate
speed
                        motor_diff =
(-heading_diff/heading_diff_ceiling)*motor_diff_ceiling
                    # Yaw left execution
                    if self.speed > motor_diff:
                        return -motor_diff, 0
                    else:
                        r_val = motor_diff - self.speed
                        return -self.speed+1, r_val+1
                elif heading_diff > 0:  # Case 1B: yaw right
```

```
                    if heading_diff >= heading_diff_ceiling:  #
Case 1Ba: yaw right, full speed
                        motor_diff = motor_diff_ceiling
                    else:  # Case 1Bb: yaw right, intermediate
speed
                        motor_diff =
(heading_diff/heading_diff_ceiling)*motor_diff_ceiling
                    # Yaw right execution
                    if self.speed > motor_diff:
                        return 0, -motor_diff
                    else:
                        l_val = motor_diff - self.speed
                        return l_val+1, -self.speed+1
                else:  # Case 1C: don't yaw
                    return 0, 0
            elif self.speed < 0:  # Case 2: moving backward
                if heading_diff < 0:  # Case 2A: yaw left
                    if heading_diff <= -heading_diff_ceiling:  #
Case 2Aa: yaw left, full speed
                        motor_diff = motor_diff_ceiling
                    else:  # Case 2Ab: yaw left, intermediate
speed
                        motor_diff =
(-heading_diff/heading_diff_ceiling)*motor_diff_ceiling
                    # Yaw left execution
                    if self.speed > motor_diff:
                        return 0, motor_diff
                    else:
                        l_val = motor_diff - self.speed
                        return -l_val-1, self.speed-1
                elif heading_diff > 0:  # Case 2B: yaw right
                    if heading_diff >= heading_diff_ceiling:  #
Case 2Ba: yaw right, full speed
                        motor_diff = motor_diff_ceiling
                    else:
                        motor_diff =
(heading_diff/heading_diff_ceiling)*motor_diff_ceiling
                    # Yaw right execution
                    if self.speed > motor_diff:
                        return motor_diff, 0
                    else:
                        r_val = motor_diff - self.speed
                        return self.speed-1, -r_val-1
                else:  # Case 2C: don't yaw
                    return 0, 0
```

```
            else:
                # Undefined state; throw error
                logging.error("Non-stationary movement
specified, but dist variable is 0. Exiting")
                return None, None

    """
    Converts adjusted PWM motor values for heading change into
final motor
    values to be sent to the motors.
    ===
    Inputs:
    - heading_adj: Adjusted PWM motor value (int or float).
    Returns:
    - Final motor value to be passed to the engine (int).
      - should be between self.engine.max and -self.engine.max
    """
    def _heading_adj_to_motor_val(self, heading_adj:float):
        direction = self._get_direction()
        motor_val = int(heading_adj + direction*self.speed)
        motor_val = min(self.engine.max, motor_val)
        motor_val = max(-self.engine.max, motor_val)
        return motor_val

    """
    Changes heading.
    ===
    Inputs:
    - target_heading: Heading to change to.
      - must be between 0 (inclusive) and 360 (exclusive).
    - loop_delay: Time to wait before PID updates.
    - num_prev_headings: Number of consecutive headings that
must be within
      self.heading_tolerance before adjustment ends.
      - Note: not dependent on loop_delay.
    Returns:
    - 0 if successful
    - 1 if fails
    """
    LOOP_DELAY = 1.0
    NUM_PREV_HEADINGS = 50
    def change_heading(self, target_heading:float,
loop_delay:float=LOOP_DELAY,
num_prev_headings:int=NUM_PREV_HEADINGS,
is_stationary:bool=False):
```

```
        self.is_active = True
        # target_heading check
        if target_heading < 0.0 or target_heading >= 360.0:
            logging.error("target_heading must be between 0
(inclusive) and 360 (exclusive), got %s. Exiting" %
str(target_heading))
            self.is_active = False
            return 1
        old_heading = self.heading
        self.heading = target_heading
        logging.info("target_heading = %s" %
str(target_heading))

        # loop_delay check
        if loop_delay < 0.0:
            logging.warning("loop_delay cannot be negative.
Using default loop_delay=%s." % str(self.LOOP_DELAY))
            loop_delay = self.LOOP_DELAY
        logging.info("loop_delay = %s" % str(loop_delay))

        # num_prev_headings check
        if num_prev_headings <= 0:
            logging.error("num_prev_headings must be positive
integer, got %s. Using default num_prev_headings=%s" %
(str(num_prev_headings), str(self.NUM_PREV_HEADINGS)))
            num_prev_headings = self.NUM_PREV_HEADINGS
        logging.info("num_prev_headings = %s" %
str(num_prev_headings))

        # create PID, initialize variables
        self.heading_PID =
AtsHeadingPidController(target_heading, loop_delay)
        heading = self.motion_analyzer.get_last_heading()
        self.heading_PID.set_initial_heading(heading)
        prev_headings = deque([heading])
        prev_headings_is_full = False

        # main loop, change heading and hold to make sure it's
stable
        is_stable = False
        while not is_stable:
            # exit if stopped externally
            if self.stop is True:
                self.engine.drive(self.stop_motors)
                self.heading = old_heading
```

```
                self.is_active = False
                return 1
            # get last heading
            heading = self.motion_analyzer.get_last_heading()
            # update PID, get feedback according to loop_delay
            if self.heading_PID.is_time_to_update():
                # frame current heading for PID
                adj_heading =
self.heading_PID.frame_heading(heading)
                # get PID output
                heading_change =
self.heading_PID.update_heading(adj_heading)
                # translate PID output to PWM motor values
                l_adj, r_adj =
self._pid_output_to_heading_adj(heading_change, is_stationary)
                # ensure adjusted values are valid
                if l_adj is None or r_adj is None:
                    continue
                else:
                    # get motor values within bounds
                    l_motor_val =
self._heading_adj_to_motor_val(l_adj)
                    r_motor_val =
self._heading_adj_to_motor_val(r_adj)
                    # drive motors
                    motor_drivers = dict()
                    motor_drivers[self.l_motor] = l_motor_val
                    motor_drivers[self.r_motor] = r_motor_val
                    self.engine.drive(motor_drivers)

            # store previous N headings
            if heading != prev_headings[-1]:
                prev_headings.append(heading)
                if prev_headings_is_full:
                    prev_headings.popleft()
            prev_headings_is_full = len(prev_headings) ==
num_prev_headings

            # stable if previous N headings are within tolerance
            if not prev_headings_is_full:
                is_stable = False
            else:
                avg_heading =
sum(prev_headings)/num_prev_headings
```

```
                is_stable1 = abs(avg_heading-target_heading) <=
self.heading_tolerance
                is_stable2 = abs(heading-target_heading) <=
self.heading_tolerance
                is_stable = is_stable1 and is_stable2  # AND
gate

        # finishes successfully
        logging.info("Successfully changed heading to %s" %
str(self.heading))
        if is_stationary is True:
            self.engine.drive(self.stop_motors)
        else:
            motor_drivers = {self.l_motor: self.speed,
self.r_motor: self.speed}
            self.engine.drive(motor_drivers)
        self.is_active = False
        return 0

    """
    Moves sub a linear distance.
    ===
    Inputs:
    - distance: Distance to move (in meters).
      - can be positive (+) or negative (-).
    Returns:
    - 0 if successful
    - 1 if fails
    """
    def move_distance(self, distance:float):
        self.is_active = True

        # distance check
        if distance == 0:
            logging.error("Distance must be non-zero. Exiting")
            self.is_active = False
            return 1
        logging.info("distance = %s" % str(distance))

        # get reference, one time variable setting`
        old_speed = self.speed
        old_dist = self.dist
        self.dist = distance
        x_axis = self.motion_analyzer.x
        y_axis = self.motion_analyzer.y
```

```
        x0 = self.motion_analyzer.get_displ(x_axis)
        y0 = self.motion_analyzer.get_displ(y_axis)
        logging.info("reference position = (%s, %s)" % (str(x0),
str(y0)))
        x_prev = x0
        y_prev = y0
        x = x0
        y = y0

        # spin motors
        direction = self._get_direction()
        motor_drivers = dict()
        motor_drivers[self.l_motor] = direction*self.speed
        motor_drivers[self.r_motor] = direction*self.speed
        self.engine.drive(motor_drivers)

        # main loop
        moved = 0
        abs_dist = abs(distance) - self.dist_tolerance
        while moved < abs_dist:
            # stopped externally
            if self.stop is True:
                self.engine.drive(self.stop_motors)
                self.dist = old_dist
                self.is_active = False
                return 1
            # distance changed externally
            if self.dist != distance:
                if distance == 0:
                    logging.error("Distance must be non-zero.
Exiting")
                    self.is_active = False
                    return 1
                logging.info("Distance changed to %s" %
str(self.dist))
                distance = self.dist
                abs_dist = abs(distance) - self.dist_tolerance
            # speed changed externally
            if self.speed != old_speed:
                direction = self._get_direction()
                motor_drivers = dict()
                motor_drivers[self.l_motor] =
direction*self.speed
                motor_drivers[self.r_motor] =
direction*self.speed
```

```
                self.engine.drive(motor_drivers)
                old_speed = self.speed
            # get current readings, get overall distance moved
            x = self.motion_analyzer.get_displ(x_axis)
            y = self.motion_analyzer.get_displ(y_axis)
            if x_prev != x and y_prev != y:  # don't calculate
if no change
                x_prev = x
                y_prev = y
                moved = sqrt((x-x0)**2 + (y-y0)**2)/100
                logging.info("Total distance moved = %s" %
str(moved))
        # finishes successfully
        self.engine.drive(self.stop_motors)
        self.is_active = False
        return 0


"""
Class that encapsulates the underlying PID controller for
heading adjustment.
"""
class AtsHeadingPidController:

    P = 0.1
    I = 0.05
    D = 0.025

    def __init__(self, setpoint:float, loop_delay:float):
        self.PID = PID.PID(self.P, self.I, self.D)
        self.PID.SetPoint = setpoint
        self.loop_delay = loop_delay
        self.initial_heading = None
        self.last_heading = None
        self.last_update_time = -1.0

        logging.info("P, I, D = %s, %s, %s" % (str(self.P),
str(self.I), str(self.D)))
        logging.info("PID.SetPoint = %s" %
str(self.PID.SetPoint))
        logging.info("loop_delay = %s" % str(self.PID.SetPoint))

    def is_time_to_update(self):
        return (time.time() - self.last_update_time) >=
self.loop_delay
```

```
    def set_initial_heading(self, heading:float):
        self.last_update_time = time.time()
        self.initial_heading = heading
        self.last_heading = heading
        if heading - self.PID.SetPoint < -180:
            # desired heading is closer if it goes down rather
than up
            self.PID.SetPoint -= 360
        elif self.PID.SetPoint - heading < -180:
            # desired heading is closer if it goes up rather
than down
            self.PID.SetPoint += 360
        logging.info("initial_heading = %s" % str(heading))

    def update_heading(self, heading:float):
        self.PID.update(heading)
        self.last_update_time = time.time()
        self.last_heading = heading
        out = self.PID.output
        logging.info("Updated heading at %s" %
str(self.last_update_time))
        logging.info("PID in %s, out %s" % (str(heading),
str(out)))
        return out

    def frame_heading(self, heading:float):
        # get heading within 180 degrees of last (most likely
direction change)
        while abs(self.last_heading - heading) > 180:
            if heading < self.last_heading:
                heading += 360
            else:
                heading -= 360
        return heading
```

ats_engine.py

```
import time
import logging

from captain.engine.base_engine import BaseEngine
```

```
class AtsDualThrusterEngine(BaseEngine):

    def __init__(self, l_motor_gpio:int, r_motor_gpio:int,
motor_factor:float=1.0, do_esc_init=True, use_i2c=False):
        self.l_motor = l_motor_gpio
        self.r_motor = r_motor_gpio
        self.motors = [self.l_motor, self.r_motor]
        self.motor_factor = 1.0 if ((motor_factor <= 0.0) or
(motor_factor > 1.0)) else motor_factor
        self.use_i2c = use_i2c
        self.max, self.min = 100, 0
        # internal PWM generation value setting
        if use_i2c is False:  # using pigpio software PWM
library
            # constants
            self.pwm_range = 4000
            self.dc_full_a = int(0.1*self.pwm_range)  # 10% duty
cycle
            self.dc_full_b = int(0.05*self.pwm_range)  # 5% duty
cycle
            self.dc_stop = int(0.075*self.pwm_range)  # 7.5%
duty cycle
            self.dc_half_a = self.dc_stop + (self.dc_full_a -
self.dc_stop)/2  # 8.75% duty cycle
            self.dc_half_b = self.dc_stop + (self.dc_full_b -
self.dc_stop)/2  # 6.25% duty cycle
            self.std_dc_range = self.dc_full_a - self.dc_full_b
            # internal cutoffs determined by motors
            self.l_zero_cutoff_h = 4  # left motor, zero cutoff
in positive direction
            self.l_zero_cutoff_l = -10  # left motor, zero
cutoff in negative direction
            self.l_max_cutoff_h = 100  # left motor, max cutoff
in positive direction
            self.l_max_cutoff_l = -100  # left motor, max cutoff
in negative direction
            self.r_zero_cutoff_h = 7  # right motor, zero cutoff
in positive direction
            self.r_zero_cutoff_l = -11  # right motor, zero
cutoff in negative direction
            self.r_max_cutoff_l = -100  # right motor, max
cutoff in positive direction
            self.r_max_cutoff_h = 100  # right motor, max cutoff
in megative direction
```

```
            self.l_cutoff_map_h =
(self.l_max_cutoff_h-self.l_zero_cutoff_h)/(self.max-1)  # left
map for positive direction
            self.l_cutoff_map_l =
(self.l_zero_cutoff_l-self.l_max_cutoff_l)/(self.max-1)  # left
map for negative direction
            self.r_cutoff_map_h =
(self.r_max_cutoff_h-self.r_zero_cutoff_h)/(self.max-1)  # right
map for positive direction
            self.r_cutoff_map_l =
(self.r_zero_cutoff_l-self.r_max_cutoff_l)/(self.max-1)  # right
map for negative direction
            # PWM driver setup
            import pigpio
            self.driver = pigpio.pi()
            self.driver.set_mode(self.l_motor, pigpio.OUTPUT)
            self.driver.set_mode(self.r_motor, pigpio.OUTPUT)
            self.driver.set_PWM_frequency(self.l_motor,
BaseEngine.pwm_freq)
            self.driver.set_PWM_frequency(self.r_motor,
BaseEngine.pwm_freq)
            self.driver.set_PWM_range(self.l_motor,
self.pwm_range)
            self.driver.set_PWM_range(self.r_motor,
self.pwm_range)
            # run ESC initialization
            if do_esc_init:
                self.driver.set_PWM_dutycycle(self.l_motor,
self.dc_full_a)
                self.driver.set_PWM_dutycycle(self.r_motor,
self.dc_full_a)
                time.sleep(2)
                self.driver.set_PWM_dutycycle(self.l_motor,
self.dc_stop)
                self.driver.set_PWM_dutycycle(self.r_motor,
self.dc_stop)
                time.sleep(2)
        else:  # using PCA9685 I2C 16-channel 12-bit PWM board
            # constants
            self.pwm_range = 4096
            self.dc_full_a = int(0.1*self.pwm_range)  # 10% duty
cycle
            self.dc_full_b = int(0.05*self.pwm_range)  # 5% duty
cycle
```

```
            self.dc_stop = int(0.075*self.pwm_range)  # 7.5%
duty cycle
            self.dc_half_a = self.dc_stop + (self.dc_full_a -
self.dc_stop)/2  # 8.75% duty cycle
            self.dc_half_b = self.dc_stop + (self.dc_full_b -
self.dc_stop)/2  # 6.25% duty cycle
            self.std_dc_range = self.dc_full_a - self.dc_full_b
            # internal cutoffs determined by motors TODO
            self.l_zero_cutoff_h = 0  # left motor, zero cutoff
in positive direction
            self.l_zero_cutoff_l = 0  # left motor, zero cutoff
in negative direction
            self.l_max_cutoff_h = 100  # left motor, max cutoff
in positive direction
            self.l_max_cutoff_l = -100  # left motor, max cutoff
in negative direction
            self.r_zero_cutoff_h = 0  # right motor, zero cutoff
in positive direction
            self.r_zero_cutoff_l = 0  # right motor, zero cutoff
in negative direction
            self.r_max_cutoff_l = -100  # right motor, max
cutoff in positive direction
            self.r_max_cutoff_h = 100  # right motor, max cutoff
in megative direction
            self.l_cutoff_map_h =
(self.l_max_cutoff_h-self.l_zero_cutoff_h)/(self.max-1)  # left
map for positive direction
            self.l_cutoff_map_l =
(self.l_zero_cutoff_l-self.l_max_cutoff_l)/(self.max-1)  # left
map for negative direction
            self.r_cutoff_map_h =
(self.r_max_cutoff_h-self.r_zero_cutoff_h)/(self.max-1)  # right
map for positive direction
            self.r_cutoff_map_l =
(self.r_zero_cutoff_l-self.r_max_cutoff_l)/(self.max-1)  # right
map for negative direction
            # PWM driver setup
            import Adafruit_PCA9685
            self.driver = Adafruit_PCA9685.PCA9685()
            self.driver.set_pwm_freq(BaseEngine.pwm_freq)
            # run ESC initialization
            if do_esc_init:
                self.driver.set_pwm(self.l_motor, 0,
self.dc_full_a)
```

```
                self.driver.set_pwm(self.r_motor, 0,
self.dc_full_a)
                time.sleep(2)
                self.driver.set_pwm(self.l_motor, 0,
self.dc_stop)
                self.driver.set_pwm(self.r_motor, 0,
self.dc_stop)
                time.sleep(2)

        logging.info("motor_factor = %s" % str(motor_factor))
        logging.info("l_motor = %s" % str(self.l_motor))
        logging.info("r_motor = %s" % str(self.r_motor))
        logging.info("use_i2c = %s" % str(use_i2c))

    """
    Converts a supplied motor speed factor to dutycycle value.
    ===
    Inputs:
    - motor_val: motor speed factor. Must be between -self.max
and self.max.
    Returns:
    - dc: int, dutycycle value to be sent to PWM generator.
    """
    def _motor_to_dc(self, motor:int, motor_val:int):
        # scale motor value to extended dc value range for I2C
board
        if self.use_i2c is True:
            scale_factor =
(self.dc_full_a-self.dc_full_b)/(self.std_dc_range)
            motor_val = scale_factor*motor_val
        dc = None
        # motor ID check
        if motor == self.l_motor:
            if motor_val == self.min:  # stationary
                return self.dc_stop
            elif motor_val > self.min:  # positive direction
                dc = self.dc_stop + self.l_zero_cutoff_h +
(motor_val-1)*self.l_cutoff_map_h
            else:  # negative direction
                dc = self.dc_stop + self.l_zero_cutoff_l +
(motor_val+1)*self.l_cutoff_map_l
        elif motor == self.r_motor:
            if motor_val == self.min:  # stationary
                return self.dc_stop
            elif motor_val > self.min:  # positive direction
```

```
                dc = self.dc_stop + self.r_zero_cutoff_h +
(motor_val-1)*self.r_cutoff_map_h
            else:  # negative direction
                dc = self.dc_stop + self.r_zero_cutoff_l +
(motor_val+1)*self.r_cutoff_map_l
        return int(dc)  # Will be None if it doesn't recognize
the motor ID

    """
    Drives motors at a given speed factor.
    ===
    Inputs:
    - drive_values: motor ID's to speed factor.
      - Format: {motor ID: speed factor}
      - speed factor must be between -self.max and self.max
    """
    def drive(self, drive_values:dict):
        # go through all motor values
        for motor, motor_val in drive_values.items():
            # motor val within range check
            if abs(motor_val) > self.max:
                logging.error("Max motor val is %s, got %s from
motor %s. Skipping" % (str(self.max), str(abs(motor_val)),
str(motor)))
                continue
            # adjust motor val and drive
            dc = self._motor_to_dc(motor, motor_val)
            if dc is not None:
                if self.use_i2c is False:
                    self.driver.set_PWM_dutycycle(motor, dc)
                else:
                    self.driver.set_pwm(motor, 0, dc)
            else:
                logging.error("Motor at pin %s isn't recognized.
Skipping" % str(motor))
                continue
            # set motor values for referencing
            if motor == self.l_motor:
                self.l_val = dc
            elif motor == self.r_motor:
                self.r_val = dc
```

base_engine.py

```
class BaseEngine:

    pwm_freq = 50
```

## Dead Reckoning

IMU_handler.py

```
# Written by Xavier quinn to interface with the BNO055 IMU for
dead reckoning
import logging
from Adafruit_BNO055 import BNO055
import pickle
import time

class IMU_handler(object):

     def __init__(self):
          self.IMU= BNO055.BNO055(i2c=3,rst=18)
          self.IMU.begin()
          self.IMU.set_mode(0x0C) #Puts in FMC mode (auto
calibrate)
          # Print out an error if system status is in error
mode.
          status, self_test, error =
self.IMU.get_system_status()
          if status == 0x01:
               logging.CRITICAL('IMU error: {0}'.format(error))
               logging.CRITICAL('IMU failed to initialize')
          prev_string=""

          self.load_cal() #loads saved calibration between runs

          #Allows calibration readout before starting the main
loop
          try:
               while True:
```

```
                    # print("calibrating")
                    syscal, gyro, accel, mag =
self.IMU.get_calibration_status()
                    cal_string='Sys_cal={0} Gyro_cal={1}
Accel_cal={2} Mag_cal={3}'.format(syscal, gyro, accel, mag)
                    if prev_string!=cal_string :
                         print(cal_string)
                    prev_string=cal_string
                    if(syscal+ gyro+ accel+ mag>=9 and accel==3)
:
                         print("calibrated")
                         self.save_cal() #saves calibration if
its good.
                         break
          except KeyboardInterrupt:
               pass


     def collect_data(self) :
          start_time=time.time()
          mag_vector=self.IMU.read_linear_acceleration() #gets
acceleration data
          end_time=time.time()
          collect_time=(end_time-start_time)/2 + start_time #may
be negligible, but should be a slightly more accurate estimation
of when it was measured.
          euler_vals=self.IMU.read_euler()
          return mag_vector, euler_vals,collect_time

     def load_cal(self) :
          try:
               with open('.calibration_data', 'rb') as file:
                    data=pickle.load(file)
                    self.IMU.set_calibration(data)
          except Exception as e:
               print("No calculation data to load {}".format(e))
               pass


     def save_cal(self):
          cal_data=self.IMU.get_calibration()
          print("saving cal {}".format(cal_data))
          with open('.calibration_data', 'wb') as file:
               pickle.dump(cal_data, file)
```

motion_data.py

```
#Written by Xavier Quinn to generate dead reckoning data in a
usable format
import sys
import time
import os
import numpy as np
import scipy
import matplotlib.pyplot as plt
import copy
import scipy.integrate as it
import threading
from scipy.signal import butter, lfilter, freqz, lfilter_zi
import scipy.signal as signal
import csv

try:
     import IMU_handler
except:
     from motion_analyzer import IMU_handler
     pass

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ##Methods:
# #General
#Format print
# VERBOSE=False
VERBOSE=True

def fprint(label, value) :
     try: VERBOSE
     except NameError: print("{0}: {1}\n".format(label, value))
     else:
          if VERBOSE :
               print("{0}: {1}\n".format(label, value))


def lprint(label,toPrint) :
     print(label)
     for data in toPrint :
```

```
          # for variableName in data :
          fprint("entry",data)
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

class motion_data :

     def __init__(self, range=1000, order=5, cutoff=1.2):

          self.x = 0
          self.y = 1
          self.z = 2

          self.accel_log=[[],[],[]]
          self.filt_accel_log=[[],[],[]]
          self.veloc_log=[[],[],[]]
          self.displ_log=[[],[],[]]
          self.time_log=[]

          self.heading_log=[]
          self.roll_log=[]
          self.pitch_log=[]

          self.integral_range=int(range)
          self.filter_order=int(order)
          self.filter_cutoff=cutoff


          self.IMU=IMU_handler.IMU_handler()

          # self.fs_rate=568 #0.002349853516s (average time
diff) -> 568.715999872Hz x2->~1200
          # self.filter_order=4
          # self.filter_cutoff=2.5

          print("About to thread")
          thread = threading.Thread(target=self.generate_data)
          thread.daemon = True                                #
Daemonize thread
          thread.start() #this will continually update the data.

     #takes in an accel list with the previous 2 values, time
list and the most recent displ and veloc values
     def accel_breakdown(self,accel_list, timing_list,
last_veloc, last_displ) :
```

```
          #this integrates acceleration, including the two
previous points
          veloc_list=self.integrate(accel_list, timing_list,
last_veloc)

          displ_list=self.integrate(veloc_list, timing_list[1:],
last_displ)


          return list(veloc_list)[1:], list(displ_list)


     ##~~~~~~~~~~~~~~~~~Setters~~~~~~~~~~~~~~~~~~~~~~~~
     def add_accel(self,accel, axis) :
          self.accel_log[axis].append(accel)

     def add_filt_accel(self,accel, axis) :
          self.filt_accel_log[axis].extend(accel)

     def add_veloc(self,veloc, axis) :
          self.veloc_log[axis].extend(veloc)

     def add_displ(self,displ, axis) :
          self.displ_log[axis].extend(displ)

     def add_time(self,time) :
          self.time_log.append(time)

     def add_euler(self,euler) :
          self.heading_log.append(euler[0])
          self.roll_log.append(euler[1])
          self.pitch_log.append(euler[2])
     ##~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

     def filter_data(self,to_filter, timing) :
          rate=self.calculate_fs(timing)
          order=self.filter_order
          cutoff=self.filter_cutoff

filtered=self.butter_filter(cutoff,rate,order,to_filter)
          return filtered

     #butter filter
     def butter_filter(self,cutoff, fs_rate, order, data):
          # return data
```

```
          b, a = self.butter_lowpass(cutoff, fs_rate,
order=order)
          zi = lfilter_zi(b, a)
          # y = lfilter(b, a, data , zi=zi*data[0])
          y = signal.filtfilt(b, a, data, method="gust") #this
method should give improved transition smoothing.
          return y


     def butter_lowpass(self,cutoff, fs, order=5):
          nyq = 0.5 * fs #TODO: does this make sense?
          normal_cutoff = cutoff / nyq
          b, a = butter(order, normal_cutoff, btype='low',
analog=False)
          return b, a

     def calculate_fs(self,timing_list) :
          diff_list=[]
          for i in range(1,len(timing_list)) :
               diff_list.append(timing_list[i]-timing_list[i-1])
          average_diff=sum(diff_list)/len(timing_list)
          freq=1/average_diff
          return freq

     def generate_data(self) :
          while True :
               self.pull_IMU_data()
               if(self.get_accel_len()%self.integral_range==0) :
                     #if there are enough new accel points
                    self.update_data() #this populates values
for XYZ AVD

     ##~~~~~~~~~~~~~~~~Getters~~~~~~~~~~~~~~~~~~~~~~

     #gets the n most recent values
     def get_accel_range(self,num_points, axis) :
          return self.accel_log[axis][-num_points:]

     def get_filt_accel_range(self,num_points, axis) :
          return self.filt_accel_log[axis][-num_points:]

     #gets the n most recent values
     def get_veloc_range(self,num_points, axis) :
          return self.veloc_log[axis][-num_points:]
```

```
#gets the n most recent values
def get_displ_range(self,num_points, axis) :
     return self.displ_log[axis][-num_points:]

def get_last_displ(self, axis) :
     try :
          return self.displ_log[axis][-1]
     except : #if not populated yet
          return 0

def get_last_veloc(self, axis) :
     try :
          return self.veloc_log[axis][-1]
     except : #if not populated yet
          return 0

#gets the n most recent values
def get_full_displ(self,axis) :
     return self.displ_log[axis]

#gets the n most recent values
def get_full_veloc(self,axis) :
     return self.veloc_log[axis]

#gets the n most recent values
def get_full_accel(self,axis) :
     return self.accel_log[axis]

#gets the n most recent values
def get_full_filt_accel(self,axis) :
     return self.filt_accel_log[axis]

#gets the n most recent values
def get_time_range(self,num_points) :

     return self.time_log[-num_points:]

def get_accel(self,axis) : #returns the most recent value
     try:
          return self.accel_log[axis][-1]
     except :
          raise ValueError("Array is not populated")

def get_accel_len(self) : #returns length of accel list
     try:
```

```
                return len(self.accel_log[0])
            except :
                raise ValueError("Accel log has no length?")

        def get_veloc(self,axis) : #returns the most recent value
            try:
                return self.veloc_log[axis][-1]
            except :
                return 0 #If it has yet to move, it is at 0

        def get_displ(self,axis) : #returns net displacement
            try:
                return sum(self.displ_log[axis])
            except :
                return 0 #If it has yet to move, it is at 0

        def get_time(self) : #returns the entire time Array
            try :
                return self.time_log
            except :
                raise ValueError("Something must have gone very
wrong.")

        def get_last_heading(self) :
            try:
                return self.heading_log[-1]
            except :
                return 0

        def get_last_roll(self) :
            return self.roll_log[-1]

        def get_last_pitch(self) :
            return self.pitch_log[-1]

        ##~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        #this integrates a single list and returns an integral
list.
        def integrate(self,to_integrate, timing, start_val) :
            integral=it.cumtrapz(to_integrate, x=timing)

            integral=[x+start_val for x in integral] #This adds
the start val to everything but the val that is the start val
            return integral
```

```
    def dump_data(self, file) :
        to_save=[self.get_full_filt_accel(0),
                 self.get_full_filt_accel(1),
                 self.get_full_filt_accel(2),
                 self.get_full_accel(0),
                 self.get_full_accel(1),
                 self.get_full_accel(2),
                 self.get_full_veloc(0),
                 self.get_full_veloc(1),
                 self.get_full_veloc(2),
                 self.get_full_displ(0),
                 self.get_full_displ(1),
                 self.get_full_displ(2),
                 self.get_time()
        ]
        self._save_to_csv(to_save, file)



    #Saves all passed lists to a csv which can be easily read
into sheets
    #to_save is of the form [list_0, list_1, list_n]
    def _save_to_csv(self, to_save, file) :
        zipped=zip(*to_save)
        with open('{}.csv'.format(file), 'w',
encoding="ISO-8859-1", newline='') as myfile:
                wr = csv.writer(myfile)
                wr.writerows(list(zipped))
        myfile.close()

    # Pulls IMU data
    def pull_IMU_data(self) :
        acceleration, euler_vals,time_stamp
=self.IMU.collect_data()
        self.add_euler(euler_vals)
        for i in range(3) :

            self.add_accel(acceleration[i], i)
        self.add_time(time_stamp)

# Updates Data
    def update_data(self) :
```

```
            timing_list=self.get_time_range(self.integral_range)
            for i in range(3) : #there are 3 axis, XYZ


accel_list=self.get_accel_range(self.integral_range, i)
                filtered_accel=self.filter_data(accel_list,
timing_list)

                last_veloc=self.get_last_veloc(i)
                last_displ=self.get_last_displ(i)

                #generates new displ values
                new_veloc, new_displ
=self.accel_breakdown(filtered_accel, timing_list, last_veloc,
last_displ)

                #adds the values to their storage
                self.add_filt_accel(filtered_accel, i)
                self.add_veloc(new_veloc, i)
                self.add_displ(new_displ, i)
```

filter_minimization.py

```
# Written by Xavier Quinn
# The goal of this is to run multiple acceleration tests on the
test rig while gathering data and performing minimization on
filter parameters to generate the best possible settings for
this system.

import motion_data
import matplotlib.pyplot as plt
import time as tm
import logging
import csv
import requests
import scipy.optimize as optimize
import scipy.integrate as it
import pickle
import copy
import math
import numpy as np
from scipy.signal import find_peaks
import time as tm
```

```
logging_fmt = '[%(asctime)s] %(filename)s [%(levelname)s]
%(message)s'
logging.basicConfig(filename='test.log', filemode='w',
format=logging_fmt, level=logging.INFO)
logging.getLogger().setLevel(logging.INFO)



def control_servo(speed) :
     access_token = "5e6258fe38fd74782b03e54885850847c40517d7"
     address="https://api.particle.io/v1/devices/events"
     data = {'access_token': access_token,
'name':"chtapodiPWM",'data': '{}'.format(speed)}
     r = requests.post(address, data=data)
     logging.info(r.text)


#taken from https://stackoverflow.com/a/1969274 because google
is faster than memory
#scales from one range of values to another
def translate(value, leftMin, leftMax, rightMin, rightMax):
     # Figure out how 'wide' each range is
     leftSpan = leftMax - leftMin
     rightSpan = rightMax - rightMin

     # Convert the left range into a 0-1 range (float)
     valueScaled = float(value - leftMin) / float(leftSpan)
     # Convert the 0-1 range into a value in the right range.
     return rightMin + (valueScaled * rightSpan)

#generates acceleration data based on known test rig behavior
def gen_accel(time_val, period) :
     accel_val=(gen_veloc(time_val, period)**2)/.1
     mapped_val=translate(accel_val, 0,40, 0,5.09168459433)
     return mapped_val


#generates velocity data based on known test rig behavior
def gen_veloc(time_val, period) :
     value=-math.cos(time_val*((2*np.pi)/period))+1 #this is the
integral of sin(x), with variable period.
     return value

#Sends the correct signals to the photon to generated the
correct motion signal on the test rig
```

```
def rot_handle(time_val, period) :
     veloc_val=gen_veloc(time_val, period)
     mapped_val=translate(veloc_val, 0,2, 90,120) #maps between
pwm values which result in an omega of 0->~71rpm
     control_servo(mapped_val)


#collects and returns relevant data
def run_data_colletion(period, range, order, cutoff) :
     logging.info("starting")
     md=motion_data.motion_data(range=range, order=order,
cutoff=cutoff)
     logging.info("initialized")
     start_time=tm.time()
     while(md.get_accel_len()<range) : #this should handle the
case where at this point filtered data has not been generated.
Was hard-coded to 500
          print(md.get_accel_len())
          tm.sleep(period*3)
     print("the wait is over")


accel=[md.get_full_accel(0),md.get_full_accel(1),md.get_full_acc
el(2)]

filt_accel=[md.get_full_filt_accel(0),md.get_full_filt_accel(1),
md.get_full_filt_accel(2)]


veloc=[md.get_full_veloc(0),md.get_full_veloc(1),md.get_full_vel
oc(2)]

     displ=[md.get_full_displ(0),md.get_full_displ(1),
md.get_full_displ(2)]

     time=md.get_time_range(len(displ[0]))
     logging.info("collected")
     return accel, filt_accel, veloc, displ, time

#Caclulates real values
def calc_real_vals(period, input_time) :
     time=copy.deepcopy(input_time)

     accel_list=[]
     for time_inst in input_time :
```

```
          accel_list.append(gen_accel(time_inst, period))

     return accel_list


#Performs RMSE eval for the scoring of the minimization
def rmse_eval(actual,predicted) :
     result=0
     for i in range(min(len(predicted), len(actual))) :
          result=result+(predicted[i]-actual[i])**2

     average=result/len(actual)
     return average

#Saves all passed lists to a csv which can be easily read into
sheets
#to_save is of the form [list_0, list_1, list_n]
def save_to_csv(to_save, file) :
     zipped=zip(*to_save)
     with open('{}.csv'.format(file), 'w',
encoding="ISO-8859-1", newline='') as myfile:
            wr = csv.writer(myfile)
            wr.writerows(list(zipped))
     myfile.close()

#The function to be minimized, which uses filtfilt(), a form of
butter
def butter_minimize(param) :
     tm.sleep(10)
     rot_veloc=7.1356041057
     period=10
     range=param[0]
     order=param[1]
     cutoff=param[2]
     print("range {}".format(range))
     print("order {}".format(order))
     print("cutoff {}".format(cutoff))


     accel,filt_accel, veloc, displ,
time=run_data_colletion(period, range, order, cutoff)
     z_accel=accel[2]
     zerod_list=find_zero_point(z_accel)
     z_veloc=veloc[2]
     z_displ=displ[2]
```

```
     filt_z_accel=filt_accel[2]
     try:
          if len(zerod_list)>1 :
               first_zero=time[int(zerod_list[1])] #first entry
in zerod list is the index of the first detected zero point,
which will match with the time at the same index
          else:
               first_zero=time[int(zerod_list[0])] #first entry
in zerod list is the index of the first detected zero point,
which will match with the time at the same index
     except Exception as e:
          print("no peaks found?: ", e)
          print('len {}'.format(len(zerod_list)))
          print("XXXXXXXXXXXXXXXXXXXXXXXXX\n")
          fig, ax = plt.subplots()
          ax.plot(z_accel, label="z_accel")
          ax.plot(filt_z_accel, label="filt_z_accel")
          for point in zerod_list :
               ax.scatter([point], [1], c="r")
          ax.legend()
          ax.set(xlabel='time (index)',
                 title='Calculated Vs. Measured')
          ax.grid()
          #Saves plot with relevant title
          file="data/ERROR {0}-{1}-{2}:{3}".format(range, order,
cutoff, tm.time())

          fig.savefig("{}.png".format(file))
          plt.close()
          return 20 # I think this large a value was confusig
the gradient, and I know 20 is above the score I care about
9223372036854775807 #maxint -- returns the worst value if
something is wrong enough to make it so no peaks are detected.
If I was only running one iteration of this code, this would be
a foolish way to handle this, but as I will be optimizing the
optimization ranges, this should be inconsequential to the
results.


     zerod_time=[entry-first_zero for entry in time]



     real_z=calc_real_vals(period,zerod_time)
```

```
     real_z_accel=copy.deepcopy(real_z) #at the moment I am not
using the other 2


     #Runs scoring calculation
     accel_score=rmse_eval(real_z_accel,filt_z_accel)
     logging.info("accel score {}".format(accel_score))
     score=accel_score


     print("combined score {}".format(score))
     print("iteration done\n------------------------------\n")

     #Plots out data specific to epoch
     fig, ax = plt.subplots()
     ax.plot(z_accel, label="z_accel")

     ax.plot(real_z_accel, label="real_z_accel", linestyle='-.',
c='red')
     ax.plot(filt_z_accel, label="filt_z_accel", linestyle='--',
c='orange')
     ax.plot(z_veloc, label="z_veloc", linestyle='-', c='cyan')
     ax.plot(z_displ, label="z_displ", linestyle='-', c='green')
     for point in zerod_list :
          ax.scatter([point], [1], c="r")
     ax.legend()
     ax.set(xlabel='time (index)',
            title='Calculated Vs. Measured\n{0} {1}
{2}'.format(range, order, cutoff))
     ax.grid()
     #Saves plot with relevant title
     file="data/{3}:{0}-{1}-{2}".format(range, order, cutoff,
score)
     fig.savefig("{}.png".format(file))
     plt.close()
     save_to_csv([z_accel,filt_z_accel,real_z_accel], file)

     return score

# Locates minima to align real with gathered data
def find_zero_point(y) :

     y_inv=[0-i for i in y]
     found = find_peaks(y_inv, height=-.5)
     peaks=found[0]
```

```
     # this way the x-axis corresponds to the index of x
     peak_list=[]

     #This groups by valley
     prev_peak=peaks[0]
     peak_range=[]
     for point in peaks[1:] :
          if y[point]>-.5 : #this should constrain peak
selection to actual valleys
               diff=point-prev_peak
               if diff<500 :
                    peak_range.append(point)
               else :
                    peak_list.append(copy.deepcopy(peak_range))
                    peak_range=[]
               prev_peak=point
     #this just averages the group, I could weight the average
by how prominant the peak is, but based on the level of
resultion being used, I do not think this will make a
difference.
     minima_list=[]
     if (len(peak_list)>0) :
          for range in peak_list :
               if(len(range)>0) :
                    minima=sum(range)/len(range)
                    minima_list.append(int(minima))
     else:
          print("synchronization point highly estimated")
          minima_list.append(y.index(min(y)))
     print("minima_list ", minima_list)
     return minima_list


#pretty much just encapsultes the minimization function
def minimize(params, bounds): #[range, order, cutoff]
     #minimization
     res = optimize.minimize(butter_minimize, params,
bounds=bounds)
     return res


#initialize values
range_range=(100,500)
order_range=(2,5)
cutoff_range=(.5,1.2)
```

```
params=[500,3,.75]
bounds=[range_range,order_range,cutoff_range]

print("beginning")
# Begin minimization
results=minimize(params, bounds)
print(results)

#Saves the results
with open('minimization_results', 'wb') as file:
     pickle.dump(results, file)
```

## Sonar

### sonar.py

```
# Written by Xavier Quinn
# The sonar class encapsulates all functions to run and test the
sonar module for the AFuS project

import time
import pigpio
import pickle
import matplotlib.pyplot as plt



class sonar :
     def __init__(self, ping_duration, transmit_pin,
receive_pin, medium="water"):
          self.SPEED_OF_SOUND_IN_WATER=1480
          self.SPEED_OF_SOUND_IN_AIR=344

          if medium=="air" :
               self.sound_velocity=self.SPEED_OF_SOUND_IN_AIR
          elif medium=="water" :
               self.sound_velocity=self.SPEED_OF_SOUND_IN_WATER

          self.transmit_pin=transmit_pin
          self.receive_pin=receive_pin
          self.ping_duration=ping_duration
          self.pio=pigpio.pi()
          self.send_tick=0
```

```
            self.travel_duration=0
            self.callback=self.init_listen()
            self.received_list=[]
            self.has_pinged=False;
            self.detected_transmit=False;

            self.init_transmit()


      #initializes reception and callback
      def init_listen(self) :
            self.pio.set_mode(self.receive_pin, pigpio.INPUT)
            return self.pio.callback(self.receive_pin,
pigpio.RISING_EDGE, self.heard_response)

      #initialized transmission
      def init_transmit(self) :
            self.pio.set_mode(self.transmit_pin, pigpio.OUTPUT)


      #Sends out a ping for the configured duration, handles
logic for preparing distance calculation
      def ping(self) :
            #updates the tick at which the ping was sent.
            self.received_list=[]
            pre_tick=self.pio.get_current_tick()
            self.pio.write(self.transmit_pin,1)
            time.sleep(self.ping_duration)
            self.pio.write(self.transmit_pin, 0)
            post_tick=self.pio.get_current_tick()
            tick_diff=post_tick-pre_tick
            # print("diff", tick_diff)
            self.send_tick= pre_tick#int(tick_diff/2)+pre_tick
#This is using the middle time as the send time.

            self.travel_duration=0 #This is so an incorrect value
cannot accidentally be acquired while waiting for the callback

            self.has_pinged=True;

            return self.send_tick

      #gets the most recent distance value.
      def get_dist(self) :
            timeout=.5
```

```
          start_time=time.time()
          while(self.travel_duration==0) : #this should act to
catch unhandled pings
               time.sleep(.01)
               # print("waiting")
               if (time.time()-start_time)>timeout :
                    print("UNRECEIVED PING")
                    return 0

          dur_ms=self.travel_duration/1000000
          return self.sound_velocity*dur_ms



     #callback function which is triggered when a received ping
is detected.
     def heard_response(self, gpio, level, tick) :
          self.received_list.append(tick)
          # print("CALLB")


          if (self.has_pinged and not self.detected_transmit) :
               # print("detected transmit")
               self.detected_transmit=True

          elif (self.detected_transmit) :
               # print("detected echo")
               self.detected_transmit=True
               has_pinged=False

self.travel_duration=self.received_list[-1]-self.received_list[0
]
               # print("duration ", self.travel_duration)

     #encapsulates pinging and waiting
     def measure_dist(self) :
          dist=0
          while dist ==0 :
               self.ping()
               dist=self.get_dist()
          return dist

# returns the average value of a list of values
def take_average(list) :
     # print("list ", list)
```

```
     average=sum(list)/len(list)
     # print("average ", average)
     return average

def main():
     transmit_pin=18
     receive_pin=17
     duration=.0035 #~4 periods



     yell=sonar(duration, transmit_pin, receive_pin, "air")
     print("setup")


     #Code that generates and evaluates data by running a series
of pings at known distances and comparing the average of 5
trials to the real value.
     DEFAULT_HEIGHT=2.2
     DEFAULT_STEP=.02
     #gets start height
     start_height=float(input("please input starting height(m)
(default: {}(m)): ".format(DEFAULT_HEIGHT))or DEFAULT_HEIGHT)
#2.2m is the start height of the stand
     print("confirmed: ", start_height,"(m)")
     height_step=float(input("please input step size(m)
(default: {}(m)): ".format(DEFAULT_STEP))or DEFAULT_STEP) #
     print("confirmed: ", height_step,"(m)")

     curr_height=start_height
     test_list=[]

     num_trials=5
     print("\nstarting tests at {0}(m), default step of {1} and
{2} trials\n\n".format(start_height, height_step, num_trials))
     try:
          while (True) :
               #take input height, take measurement, take step,
loop

               trial_data=[]
               for i in range(num_trials) :
                    try:
                         trial_data.append(yell.measure_dist())
                    except KeyboardInterrupt:
```

```
                        trial_data.append(-1)
                        print("ERROR")
                        pass
                    time.sleep(.1) #delay between tests
               av_data=take_average(trial_data)
               trial_data.append(av_data) #last element is the
average
               test_list.append([curr_height*2, trial_data])

print("actual:{0:.2f}\tmeasured:{1:.2f}".format(curr_height*2,
trial_data[-1]))

               curr_height+=height_step
               curr_height=float(input("new height(m) (default:
{:.2f}(m)): ".format(curr_height))or curr_height) #2.2m is the
start height of the stand
               print("confirmed: ", curr_height,"(m)")
     except KeyboardInterrupt:
          pass


     for_sheet="real, "
     for i in range(len(test_list[0][1])-1) :
          for_sheet+="test {}, ".format(i)

     for_sheet+="average\n"

     for test in test_list :
          line_str=str(test[0])
          for val in test[1] :
               line_str+=", {}".format(val)
          line_str+="\n"
          for_sheet+=line_str


print("actual:{0:.2f}\tmeasured:{1:.2f}".format(test[0],
test[1][-1]))

     file_name=input("\nPlease input save name: " or
time.time())
     pickle.dump(test_list,open("{}.p".format(file_name),"wb"))


     print(for_sheet)
```

```
if __name__== "__main__":
  main()
```

## Communication

### decoder.py

```
import logging
import struct
import pigpio
# dac packages
import busio
import board
import adafruit_mcp4725

DEFAULT_FREQ = 125  # in Hz
DEFAULT_VAL_LENGTH = 32  # in bits
DEFAULT_TOLERANCE = 499  # in microseconds

DAC_RESOLUTION = 4096  # 12-bits
DEFAULT_V_COMPARE = 2.5  # in volts
V_MAX = 5.0  # in volts
V_MIN = 0.0  # in volts


class AtsMk2Decoder:

    def __init__(self, pi, pin, val_length=DEFAULT_VAL_LENGTH,
freq=DEFAULT_FREQ, tolerance=DEFAULT_TOLERANCE,
v_compare=DEFAULT_V_COMPARE):
        # receiver
        self.rx = AtsMk2Receiver(pi, pin, val_length, freq,
tolerance, v_compare)

    def is_busy(self):
        return self.rx.is_busy()

    def int(self, bits):
        return int(bits)

    def float(self, bits):
```

```
        float_val = None
        try:
            float_val = struct.unpack('!f',struct.pack('!I',
int(bits, 2)))[0]
        except TypeError:
            float_val = struct.unpack('!f',struct.pack('!I',
int(bits)))[0]
        return float_val

    def char(self, bits):
        return chr(bits)

    def get_raw_data(self):
        return self.rx.get_raw_data()

    def get_bit_string(self):
        return self.rx.get_bit_string()

    def get_v_compare(self):
        return self.rx.get_v_compare()

    def get_dataf(self):
        return [self.float(i) for i in self.rx.get_raw_data()]

    def clear(self):
        return self.rx.clear()

    def shutdown(self):
        return self.rx.shutdown()


class AtsMk2Receiver:

    def __init__(self, pi, pin, val_length=DEFAULT_VAL_LENGTH,
freq=DEFAULT_FREQ, tolerance=DEFAULT_TOLERANCE,
v_compare=DEFAULT_V_COMPARE):
        # hardware interface
        self.pi = pi
        self.pin = pin
        self.pi.set_mode(self.pin, pigpio.INPUT)
        self._i2c = busio.I2C(board.SCL, board.SDA)
        self._dac = adafruit_mcp4725.MCP4725(self._i2c,
address=0x62)
        self.set_v_compare(v_compare)
        # square wave signal characteristics
```

```
        self.freq = freq
        self.period = 1/freq
        self.period_us = self.period*1000000
        self.half_period_us = self.period_us/2
        # data
        self.val_length = val_length
        self.tolerance = tolerance
        self._ticks = list()
        self._data = list()
        self._bit_data = str()
        # callback function
        self._cb = pi.callback(pin, pigpio.RISING_EDGE,
self._cb_func)

        # log everything
        logging.info("pin = %s" % str(self.pin))
        logging.info("v_compare = %s" % str(v_compare))
        logging.info("freq = %s" % str(self.freq))
        logging.info("period = %s" % str(self.period))
        logging.info("period_us = %s" % str(self.period_us))
        logging.info("half_period_us = %s" %
str(self.half_period_us))
        logging.info("val_length = %s" % str(self.val_length))

    def set_v_compare(self, voltage):
        if voltage > V_MAX or voltage < V_MIN:
            logging.error("Value must be between %s and %s
(inclusive), got %s" % (str(V_MIN), str(V_MAX), str(voltage)))
            return 1
        self._dac.raw_value =
int((voltage/V_MAX)*(DAC_RESOLUTION-1))
        return 0

    def get_v_compare(self):
        return (self._dac.raw_value/(DAC_RESOLUTION-1))*V_MAX

    def _cb_func(self, pin, level, tick):
        #print("cb")
        self._ticks.append(tick)

    def is_busy(self):
        if len(self._ticks) == 0:
            return True
        elif self.pi.get_current_tick()-self._ticks[-1] <
3*self.period_us:
```

```
            return True
        else:
            return False

    def _process(self):
        # data is present check
        if len(self._ticks) < 1:
            return 1
        # loop variables
        skip_next = False
        last_val = 0
        bit_counter = 0
        val_counter = 0
        data = 0b0
        # loop over ticks from callback function, extract 1's
and 0's
        for i in range(1, len(self._ticks)):
            diff = self._ticks[i] - self._ticks[i-1]
            if skip_next is True:  # two concurrent pulses if a
bit is 1
                skip_next = False
                continue
            elif self._is_full_period(diff):  # full-period/bit
0
                bit_counter += 1
                data |= 0<<(self.val_length-bit_counter)
                self._bit_data += "0"
            elif self._is_half_period(diff):  # half-period/bit
1
                bit_counter += 1
                data |= 1<<(self.val_length-bit_counter)
                self._bit_data += "1"
                skip_next = True
            else:
                logging.error("Error on value %s, bit %s: diff
is %s" % (str(val_counter), str(bit_counter), str(diff)))
            # finished value check
            if bit_counter == self.val_length:
                self._data.append(data)
                val_counter += 1
                bit_counter = 0
                data = 0b0
        # successful finish
        logging.info("Received %d values: %s" % (val_counter,
str(self._data)))
```

```
        return 0

    def _is_half_period(self, num):
        return self.half_period_us+self.tolerance >= num >=
self.half_period_us-self.tolerance

    def _is_full_period(self, num):
        return self.period_us+self.tolerance >= num >=
self.period_us-self.tolerance

    def get_raw_data(self):
        if len(self._data) == 0:
            self._process()
        return self._data

    def get_bit_string(self):
        if len(self._bit_data) == 0:
            self._process()
        return self._bit_data

    def clear(self):
        self._ticks = list()
        self._data = list()
        self._bit_data = str()
        self._cb.reset_tally()
        return 0

    def shutdown(self):
        self.clear()
        self._cb.cancel()
```

encoder.py

```
import logging
import struct
import numpy as np
import pigpio


DEFAULT_FREQ = 125  # in Hz
DEFAULT_VAL_LENGTH = 32  # in bits
DEFAULT_DC = 0.025  # out of 1.00

"""
```

```
Class responsible for encoding values into bit representations
and sending them.
"""
class AtsMk2Encoder:

    def __init__(self, pi, pin, val_length=DEFAULT_VAL_LENGTH,
freq=DEFAULT_FREQ, dc=DEFAULT_DC):
        # data
        self._data = list()
        # transmitter
        self.tx = AtsMk2Transmitter(pi, pin, val_length, freq,
dc)

    """
    Encode integer as string of 0's and 1's, and returns it.
    ===
    Inputs:
    - data: 32-bit integer.
    Returns:
    - Binary representation of 'data' as a String.
      - i.e. input of 5 produces "101"
    """
    def int(self, data):
        return str(bin(data))[2:]

    """
    Encode float as a string of 0's anad 1's, and returns it.
    This method adheres to the IEEE 754 32-bit floating point
number standard.
    ===
    Inputs:
    - data: float (32 or 64 bit).
    Returns:
    - Binary representation of 'data' as a String.
    """
    def float(self, data):
        return format(struct.unpack('!I', struct.pack('!f',
np.float32(data)))[0], '032b')

    """
    Encode char (or string of chars) as a string of 0's and 1's,
and returns it.
    ===
    Inputs:
    - data: char (<=8 bits), or a string of chars.
```

```
    Returns:
    - Binary representation of 'data' as a String.
    """
    def char(self, data):
        return str(bin(int.from_bytes(data.encode(),
'big')))[2:]

    """
    Encode string as a string of 0's and 1's, and returns it.
    ===
    Inputs:
    - data: string of chars (<=8 bits).
    Returns:
    - Binary representation of 'data' as a String.
    """
    def str(self, data):
        return self.char(data)

    """
    Add an encoded value to the queue of items that will be
transmitted.
    Note: Encode items by using any of the above methods (int,
float, char, str).
    ===
    Inputs:
    - encoded_data: Binary string, returned by above methods.
    Returns:
    - 0 if successful
    """
    def add(self, encoded_data):
        self._data.append(encoded_data)
        logging.info("Adding encoded value %s" %
str(encoded_data))
        return 0

    """
    Send all values that have been encoded.
    ===
    Returns:
    - 0 if successful
    - 1 if failure
    """
    def send(self):
        return self.tx.send(self._data)
```

```
    """
    Clear all waveforms and data.
    Returns:
    - 0 if successful
    """
    def clear(self):
        self._data = list()
        return self.tx.clear()


"""
Class responsible for generating/transmitting PPM-encoded
signal.
"""
class AtsMk2Transmitter:

    def __init__(self, pi, pin, val_length=DEFAULT_VAL_LENGTH,
freq=DEFAULT_FREQ, dc=DEFAULT_DC):
        # Hardware interface
        self.pi = pi
        self.pin = pin
        self.pi.set_mode(self.pin, pigpio.OUTPUT)
        # Square wave signal characteristics
        self.freq = freq
        self.period = 1/freq
        self.period_us = self.period*1000000
        # Waveform settings
        self.dc = dc
        self.pw_hi = int(self.dc*self.period_us)
        self.pw_lo = int((self.period_us/2)-self.pw_hi)
        self.pw_zero = int(2*self.pw_lo + self.pw_hi)
        # data
        self.val_length = val_length
        self._pulses = []
        self._id = None

        # log everything
        logging.info("pin = %s" % str(self.pin))
        logging.info("freq = %s" % str(self.freq))
        logging.info("period = %s" % str(self.period))
        logging.info("period_us = %s" % str(self.period_us))
        logging.info("dc = %s" % str(self.dc))
        logging.info("val_length = %s" % str(self.val_length))

    """
```

```
    Convert string of bits to pulses in a waveform.
    ===
    Inputs:
    - bits: Binary string.
    Returns:
    - 0 if successful
    """
    def bits_to_pulses(self, bits):
        # prepend extra zeros to make neat 32-bit chunks
        num_zeros = self.val_length - (len(bits) %
self.val_length)
        if num_zeros != self.val_length:
            bits = "0"*num_zeros + bits
        # encode bit by bit
        for bit in bits:
            self._pulses.append(pigpio.pulse(1<<self.pin, 0,
self.pw_hi))
            if int(bit) == 1:  # bit is 1, send pulse
                self._pulses.append(pigpio.pulse(0, 1<<self.pin,
self.pw_lo))
                self._pulses.append(pigpio.pulse(1<<self.pin, 0,
self.pw_hi))
                self._pulses.append(pigpio.pulse(0, 1<<self.pin,
self.pw_lo))
            else:  # bit is 0, stay low
                self._pulses.append(pigpio.pulse(0, 1<<self.pin,
self.pw_zero))
        return 0

    """
    Transmit a provided waveform.
    ===
    Inputs:
    - data: Waveform to transmit (list of pigpio.pulse()
objects).
    Returns:
    - 0 if successful
    - 1 if failure
    """
    def send(self, data):
        # convert all values into waveforms
        for bits in data:
            # successful encoding check
            if self.bits_to_pulses(bits) != 0:
```

```
                logging.error("Issue encoding val %s. Exiting" %
str(bits))
                return 1
        # prepare and send wave
        return self._send()

    """
    Underlying method to transmit a waveform on a pin.
    ===
    Returns:
    - 0 if successful
    - 1 if failure
    """
    def _send(self):
        # append closing clock pulse
        self._pulses.append(pigpio.pulse(1<<self.pin, 0,
self.pw_hi))
        self._pulses.append(pigpio.pulse(0, 1<<self.pin,
self.pw_lo))
        # clear existing waveforms, create new one
        self.pi.wave_clear()
        self.pi.wave_add_generic(self._pulses)
        self._id = self.pi.wave_create()
        self.pi.wave_send_once(self._id)
        logging.info("Sent wave with ID %s" % str(self._id))
        # clear waveform for next time
        self._data = []
        self._pulses = []
        return 0

    """
    Clear all waveform data.
    ===
    Returns:
    - 0 if successful
    """
    def clear(self):
        self.pi.wave_clear()
        self._pulses = list()
        return 0
```