6-2024

Waypoint Profiler

Jeff Ke

Ricky Schober

Alexander Collins

Kevin Wang

SANTA CLARA UNIVERSITY

Department of Mechanical Engineering

Department of Computer Science and Engineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED
UNDER MY SUPERVISION BY

Jeff Ke, Ricky Schober, Alexander Collins, Kevin Wang

ENTITLED

WAYPOINT PROFILER

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREES OF

BACHELOR OF SCIENCE
IN
MECHANICAL ENGINEERING
AND
BACHELOR OF SCIENCE
IN
COMPUTER SCIENCE AND ENGINEERING

Christopher Kitts — 6/17/2024

Dr. Christopher Kitts, Thesis Advisor — Date

Michael Neumann — 06/12/2024

Dr. Michael Neumann, Thesis Advisor — Date

Michael Taylor — 6/17/24

Dr. Michael Taylor, Department of Mechanical Engineering Chair — Date

6/18/2024

Dr. Silvia Figueira, Department of Computer Science and Engineering Chair — Date

Waypoint Profiler

By

Jeff Ke, Ricky Schober, Alexander Collins, Kevin Wang

SENIOR DESIGN PROJECT REPORT

Submitted to

the Department of Mechanical Engineering

And Department of Computer Science and Engineering

of

SANTA CLARA UNIVERSITY

in Partial Fulfillment of the Requirements for the degrees of

Bachelor of Science in Mechanical Engineering and Bachelor of Science in Computer Science and Engineering

Santa Clara, California

2024

Abstract

Ocean health monitoring is crucial for maintaining the health of the ocean ecosystem. Currently, divers are deployed to collect data manually, which is both time and resource-consuming. Additionally, this process poses significant dangers to the divers. Therefore, a more efficient method for collecting oceanic data is needed. This thesis describes the design of a novel autonomous marine vehicle, the waypoint profiler. Launched from shore with scientific sensors, it autonomously navigates to ocean locations of interest and dives to measure key ocean health markers. The system integrates subsystems for scientific sensing, health monitoring, structural integrity, communications, and navigation/control, tailored to meet stakeholder needs such as the Monterey Bay Aquarium Research Institute (MBARI), the US Army Corps of Engineers, and Occidental College. The Scientific sensing subsystem measures water temperature and captures water samples. The Health subsystem tracks battery levels and detects leaks. The Structural subsystem protects components and supports operation in various conditions. The Navigation and Control subsystem uses GPS and thrusters for precise movement. Extensive testing and validation were conducted to ensure the system's performance and reliability. The results show a partial success of our vehicle's ability to navigate to GPS waypoints and dive vertically to profile water columns. In the future, improvements can be made to the design of an internal charging system, eliminating the need to disassemble the vehicle to remove the batteries for charging. Another area for improvement is the cluster control capabilities, allowing one or more vehicles to be deployed and work collaboratively to complete tasks more efficiently.

Keywords: Autonomous Navigation, Autonomous Underwater Vehicle, Ocean health monitoring, Data Collection

Table of Contents

List of Figures

List of Tables

1. Introduction

The ocean is one of mankind's most vital resources. We owe to our ocean the existence of multinational shipping, fishing, offshore wind, and marine technology industries [1]. The ocean carries more than 90% of internationally traded goods [2]. Three billion people on the planet rely on the ocean as a source of income [2]. Furthermore, our ocean keeps our planet liveable by absorbing 90% of the heat trapped in our atmosphere [2]. In pure dollars and cents, the ocean contributed three trillion USD in 2015 to the global GDP [2]. The ocean is also a source of the wonder and complexity of the natural world: 80% of Earth’s life has its home in the ocean [2].

But we as a civilization are on the path of ruining this precious resource: our carbon emissions acidify the ocean, decimating populations of oysters, clams, mussels, and other marine species that are harmed in acidic water [3]. As our ocean gets hotter from absorbing the heat trapped by greenhouse gasses, the algae that floats in the water grows uncontrollably, clogging up marine ecosystems and depletes the water of nutrients and oxygen [4]. The full damage of these maladies is still being studied by our marine scientists.

Understanding our ocean is key to healing it. In order to ensure that our ocean is still a resource that future generations can rely on, our marine scientists have a diverse lineup of tools that they pull from, such as sampling equipment, remote sensing instruments, and robotic submersibles. Each tool has their purpose, but robotic submersibles offer to marine scientists important advantages in versatility and flexibility. Typical robots that marine scientists deploy include autonomous underwater vehicles (AUVs) [5], unmanned surface vehicles (USVs) [6], remotely operated vehicles [7], and vertical profilers [8]. These kinds of robots are each suited to different tasks: AUVs and ROVs are suited for underwater exploration tasks such as seafloor geological surveys [9]; USVs are suited for ocean-surface tasks such as algal bloom removal [6]; vertical profilers are suited for profiling columns of water [8].

Probing columns of water allow marine scientists to measure important markers of ocean health, such as water temperature, light scatter levels, dissolved oxygen, and turbidity, making vertical profilers essential tools in the marine scientist's toolkit. Vertical profilers typically belong

to one of two classes: tethered [10]-[15] and untethered [16]-[19]. Tethered profilers are typically attached to a buoy or a ship or to a line anchored to the seafloor [15], [19]. In order to dive downwards, these profilers use a winch or the natural motion of the waves. Tethered profilers often transmit the data they collect through the tether. Tethered profilers can operate for days and even months [12], [19], compared to most untethered profilers which have limited operational lifetimes, often up to 24 hours [16], [18]. However, some untethered profilers are low-powered and remain active for many years [32]. In comparison, untethered profilers travel independently, often using thrusters or buoyancy engines. Untethered profilers are typically recovered at the end of their missions in order to retrieve their collected data and samples or are discarded after their mission expires [16], [18].

The Robotic Systems Lab has contributed to the state of the art by developing its own marine robots, particularly ROVs shown in Figure 1 [20] and vertical profilers shown in Figure 2 [8]. The RSL's vertical profiler is designed to be low-cost with less overhead than heavy-duty profilers but more endurance and features than small profilers—ultimately allowing in-the-water research to be conducted more readily and at a lower cost in time and resources. Though not a replacement for large profilers, dives made by a mid-size vertical profiler such as the RSL's might kickstart research by nosing out clues that would justify larger expeditions.

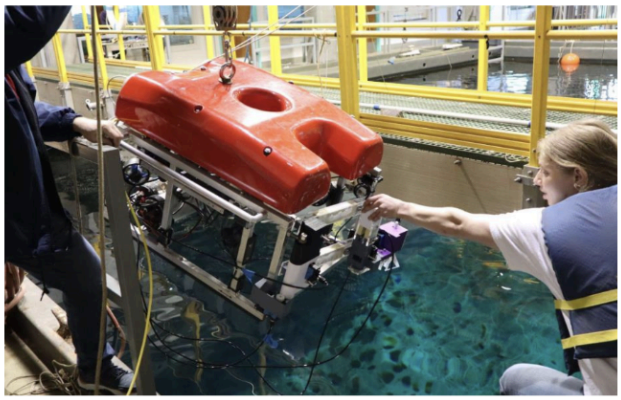


Figure 1: ROV developed by SCU RSL (Nautilus)

Figure 2: Vertical Profiler developed by SCU RSL

Vertical profilers, however, are limited. To profile multiple water columns at specific locations, a marine scientist needs to drive a boat and manually pick up or tow the profiler to each and every location. This added cost and overhead means that marine scientists have less information and therefore less of a grasp of our ocean. We've made it our mission to develop an autonomous marine vehicle that can be deployed from the shore, travel across the water, and dive into the water to collect water-column data using modular scientific equipment.

2. System Overview

2.1 Customer Needs & Requirements

In order to better understand how we could design a vehicle that would help marine scientists the most, we spoke with marine scientists from the Monterey Bay Aquarium Research Institute (MBARI), the US Army Corps of Engineers, and Occidental college and listened to their needs.

The scientists at MBARI wanted a scientific vehicle capable of:

- Diving multiple times in one deployment,
- Operating in rough water and currents,
- Drawing physical water samples,
- Navigating itself to locations of interest,
- Launching from shore,
- Collaborating with other robots to adaptively navigate to locations of interest.

Additionally, the scientists at MBARI wanted a vehicle that was ergonomic, simple to operate, and easy to fix.

The marine scientists at the US Army Corps of Engineers wanted a vehicle that could support their dredging operations. Dredging the seafloor kicks up plumes of sediment into the water. The plumes migrate with the current, spreading the sediment and potentially smothering sensitive marine life. The Corps wanted a vehicle that could:

- Hunt down and pinpoint these migrating clouds of pollutants in real time without exhaustively combing through the area,
- Transmit pollutant concentration data back to the scientists onshore,
- Operate in both shallow and deep water conditions.

The marine researchers at Occidental College are a part of the Palos Verdes Restoration reef project: an endeavor to build an artificial rocky reef off the coast of southern California in

the hopes of restoring lost habitats and shelter for fish and invertebrates. The researchers desired a vehicle that could:

- Launch from shore and autonomously navigate to the artificial reef,
- Monitor and record the health of the artificial reef's burgeoning ecosystem (specifically, temperature, salinity and dissolved oxygen concentration)

2.2 System Requirements

To create a vehicle that satisfies as many needs of our stakeholders as possible, we came up with a matrix of requirements for our system in Table 1.

Table 1: Requirements matrix

Our vehicle shall:	
1. Have an operational range of:	2 km
2. Have a position error less than:	5 m
3. Transmit its location, battery voltage, internal pressure to the user every:	5 s
4. Sense its depth with uncertainty less than:	1 m
5. While profiling 10 meters deep, not drift laterally more than:	5 m
6. Measure temperature at a resolution of:	± 0.01 °C
7. Measure bathymetry at a resolution of:	± 1.0 m
8. Dive to a max depth of:	100 m
9. Have an operating time of:	2 hours
10. Have a weight of:	60 lbs

2.3 Concept of operations

To meet these requirements, we designed our vehicle's concept of operations, as shown in Figure 3.

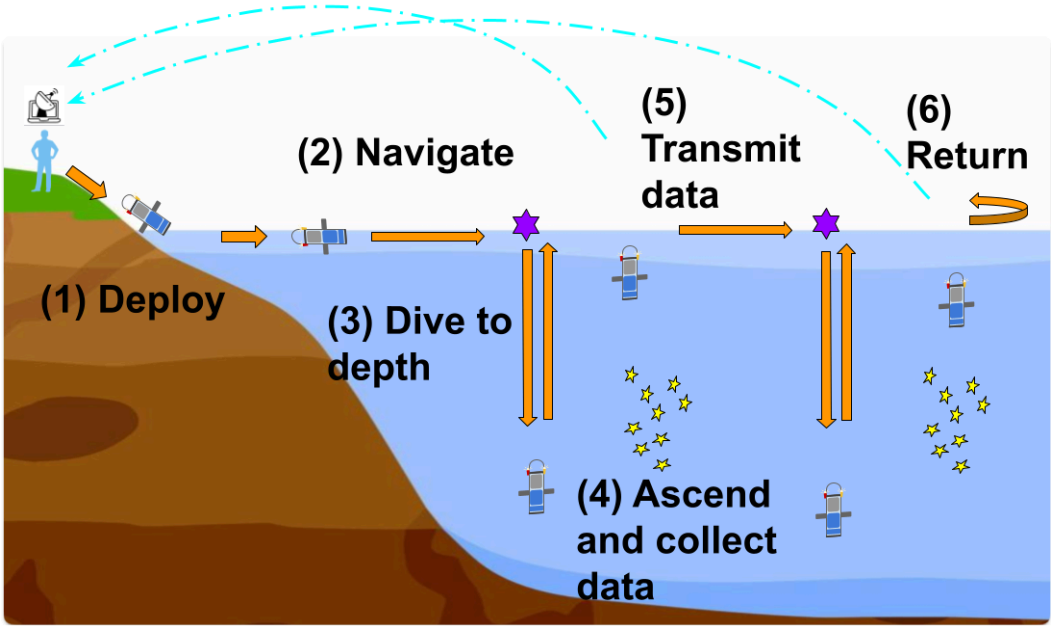


Figure 3: Concept of Operations

Figure 3 demonstrates the process of the vehicle collecting water-column data during deployment. Initially, the craft is deployed either from a boat or from shore. It traverses horizontally across the water's surface to reach the first of an itinerary of waypoints. Upon arrival, it descends to collect water column data, resurfaces, and then proceeds to the subsequent waypoints. This cycle continues until all designated waypoints are completed. Finally, the craft returns to the deployment location.

2.4 Mechanical Configuration

Our craft's design has four thrusters illustrated in Figure 4: two thrusters ("Parallel thrusters") are oriented parallel to the lengthwise axis of the craft in order to propel the craft across the water while horizontal; two other thrusters are oriented perpendicular ("Perpendicular thrusters") to the lengthwise axis of the craft in order to pull the profiler vertically into vertical diving mode. Moreover, with four thrusters the craft now can control yaw and pitch. This design

decision's rationale is detailed in the Navigation and Control chapter. Figure 5 illustrates the internal mechanical configuration of the craft. The battery tray is positioned in the front section, while the electronic tray is located in the back to minimize the influence of the magnetic field on sensitive scientific equipment. Additionally, a nose cone is employed to reduce drag in the water and house the ping sensor.

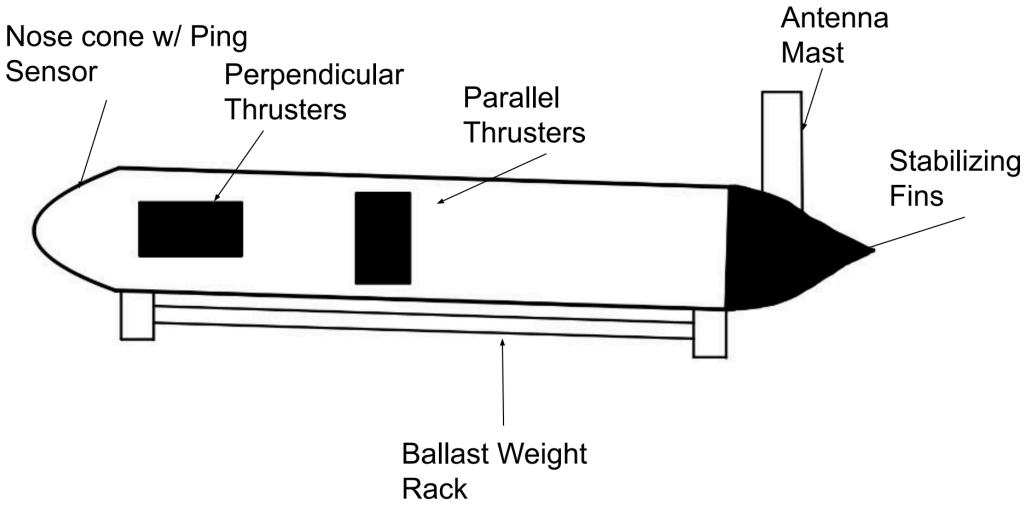


Figure 4: External view of mechanical configuration

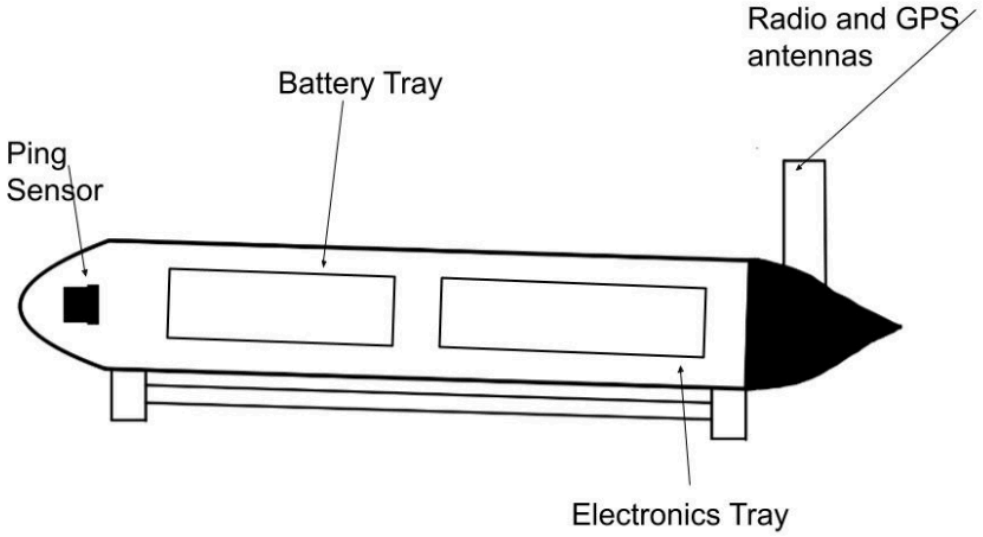


Figure 5: Internal view of mechanical configuration

2.5 Subsystem Breakdown

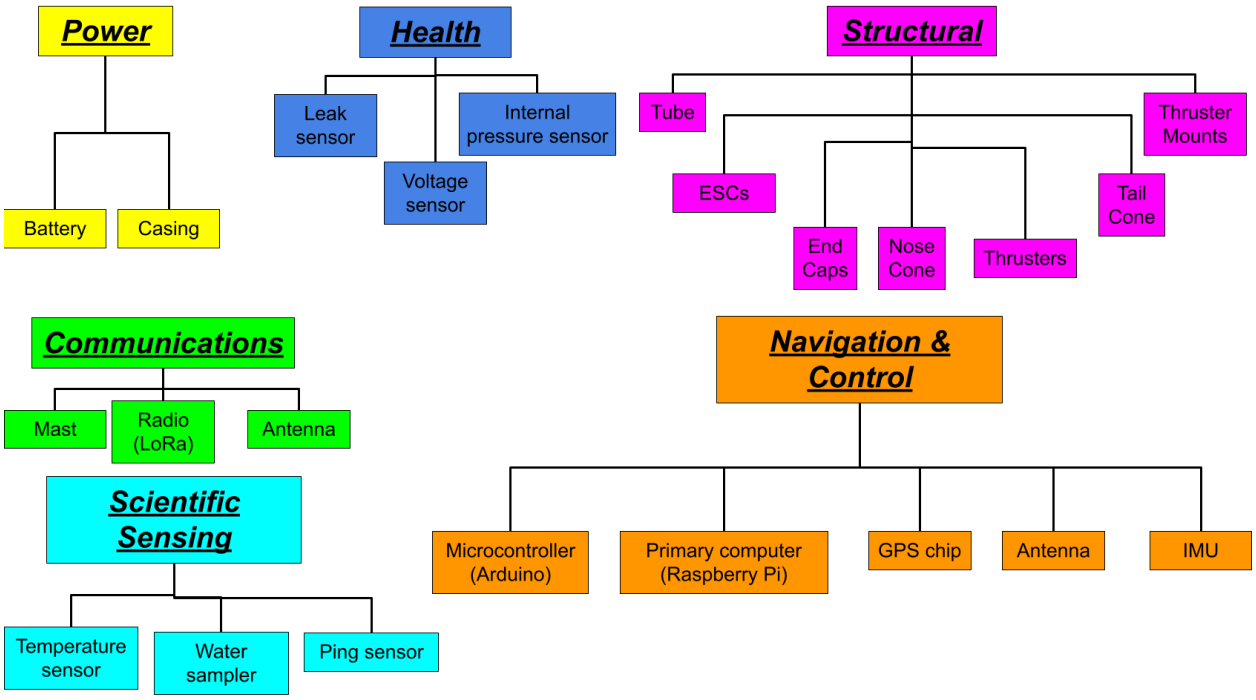


Figure 6: Subsystem diagram

Figure 6 shows the subsystem diagram of the craft. There are six subsystems, each describing different functionalities within the system. The Power subsystem is responsible for the housing and placement of the batteries, which includes the battery casing and the 4 batteries. The Structural subsystem entails the structurecraft, such as the weight rack, the acrylic tube, the end caps, etc. The Health subsystem monitors the vehicle's health during operation. It includes a voltage sensor and an internal pressure sensor. The Communication subsystem includes the radio chip, radio antenna, and the mast that holds the antennas. The Navigation and Control subsystem includes the GPS chip, GPS antenna, the IMU, and the antenna mast that holds the GPS antenna. The Scientific Sensing subsystem includes the temperature sensor, the sonar ping sensor, and the water sampler.

2.6 Component Block Diagram

At the component-level scope, we organized all of components (e.g. sensors, actuators, microcontrollers) according to a blueprint described by our component block diagram in Figure 7:

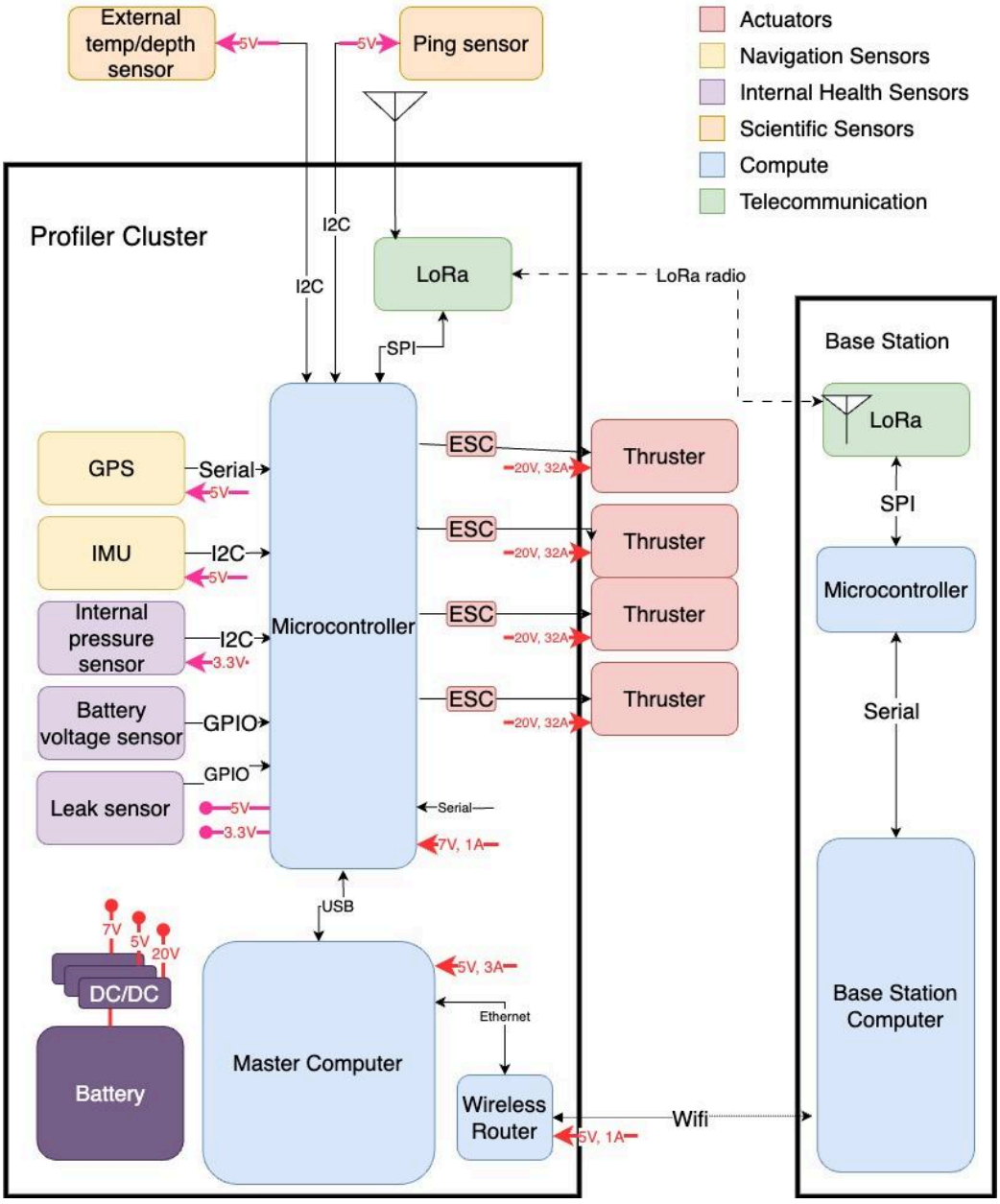


Figure 7: Component block diagram

2.7 Team Management

Because our team is made up of two mechanical engineers and two computer engineers, we had frequent 15-minute meetings to keep the team up-to-date on priorities for our project. At the beginning of the project, we created a Gantt chart to guide our progress, shown in Appendix V. We applied the principles of Agile software development and organized our workflows into biweekly Sprints [33], [34]. Each sprint would have a set of target goals we would aim to complete by the end of the Sprint.

3. Computing Hardware

In this section we discuss a tradeoff analysis we completed to determine what computer should control our vehicle. The important factors that influenced our decision were interoperability, capability, simplicity, power consumption, and affordability. We then discuss how we implemented our chosen computers and the communication protocol between them. Finally, we expound on how we tested this system.

3.1 Alternatives Considered and Trade-off Analysis

When looking at different options for microcontrollers to fulfill this role there are several drivers to consider listed in Table 2.

Table 2: Driver ranking for computing hardware

Driver Weight	Drivers
5	Interoperability: how many and what kind of devices can it connect to
5	Capability: how versatile and powerful it is in terms of functionality and magnitude of performance
4	Simplicity: how simple it is to use and program
2	Power-consumption: how much power does it draw
2	Affordability: how expensive it is

The microcontroller must have the right number and type of ports so it is able to connect to all the other required electronics, which is what makes interoperability so important. This includes USB connections to connect to our computer or other microcontrollers, Ethernet port to connect to the local network, I2C channel, and serial connections in order to connect to our many sensors. This driver is of high importance as it must be able to connect to all the other components for our system to function as a whole.

In terms of Capability this refers to both the microcontroller's computing power and its compatibility with various software packages and libraries that may be of use. This driver is also of highest importance as it determines what we are able to do with the microcontroller.

Simplicity refers to how easy it is to interface with and program. This includes the consideration of what packages and libraries are available on that microcontroller that will reduce some of the work we need to do. This is less important than the first two drivers as it doesn't limit our capabilities, but it does affect the development time of our software which may limit the scope of deliverables.

Power Consumption considers how much power the microcontroller will draw during operation. This is important as our total power draw limits our craft's range of operation before it runs out of battery. But the microcontroller draws less power when compared to our thrusters and thus is not as significant in determining the range of operation.

Lastly we don't have an infinite budget so we must find a solution that's reasonably priced. Based on these drivers we looked at some of the best microcontrollers on the market and arrived at the following conclusion in Table 3.

Table 3: Results of trade-off analysis for computing hardware

Drivers	Weight of each driver	Jetson AGX Xavier	Raspberry Pi	Arduino Mega	*Raspberry Pi AND Arduino Mega*
Interoperability	5	2	2	4	5
Capability	5	5	4	1	5
Simplicity	4	1	3	5	3
Power-consumption	2	1	4	3	3
Affordability	2	1	5	5	4
Total scores **(max: 90)**		43	60	61	**76**

Based on our tradeoff analysis our final design involves a Raspberry Pi which acts as the master and does the majority of computations and state machine management, while the Arduino acts as an interface to the hardware components. Communication between these two microcontrollers will be done on a serial connection with the USB ports. This design overcomes the pitfalls of using either microcontroller on its own as the Pi is much more powerful and able to do parallel tasks, while the Arduino has all the necessary ports and libraries for interfacing with the other electronics. The only significant downside to this design is the added complexity of communication between the two microcontrollers to make sure they are on the same page at all times.

3.2 Detailed Design & Analysis

The code on the Raspberry Pi is implemented using Robotics Operating System (ROS2 Humble). ROS contains a number of nodes which are parallel processes that are each designed to handle a specific task. Communication between these nodes is done by sending messages or requesting services from each other. Further details on our ROS architecture are contained in the Software Architecture chapter. The code is written in a variety of languages that are used for ROS such as C++, and Python.

The Arduino acts as an interface between all the sensors and servos in our system. Its responsibility is to read in requests from the Pi over the serial connection, collect the requested data or command the requested servos and then return a confirmation to the Pi. All of the Arduino code is written in the Arduino version of C++.

In order to maintain this beneficial division of labor between these two computers, we developed a communication protocol that allows the two computers to exchange information without loss or corruption. In the protocol, computers send small messages 10 bytes-wide, and wait until they receive an acknowledgement before sending more messages. This wait-and-send pattern is essential to avoid corrupted or missed messages because serial connection relies on a buffer which stores incoming messages until they are read. This buffer can only store 64 bytes at a time, which means messages must constantly be read as they are received in order to avoid messages overwriting each other. The Arduino checks the buffer at approximately 10 Hz and

responds accordingly to the Pi. This ensures that the serial buffer is never corrupted. Our protocol also involves following a consistent message format which is shown in Table 4. All messages are classified to be a certain type based on the information that it is sending or requesting.

Table 4: Computing hardware message formats

Message Type:	Message Tag:	Message Request:	Message Return:
GPS	G:	G:	G:Lat,Long,Altitude
Thruster	T:	T:PWM1,PWM2,PWM3,PWM4	T:PWM1,PWM2,PWM3,PWM4
Radio	R:	R:Radio message to send	R:ack received/not
IMU	I:	I:	I:X-angle,Y-angle,Z-angle

While this table is not comprehensive, it shows some of the primary message formats used by our system. The message tag is a single capital letter followed by a colon which identifies which type of message was received. This is meant to simplify the parsing of messages as they come as we can immediately identify the message type. Everything following the colon is a comma separated list of arguments or return values. Some message types like GPS do not have any arguments so their request format is simply blank following the colon. A different example is the thruster message which passes its arguments as 4 desired PWM values, one associated with each thruster. Its return type is the actual PWM values it sets to each thruster. Keeping consistent message formats and using the wait and send pattern determined by our communication protocol solved most of the difficulties of communication between our two microcontrollers.

3.3 Verification

In order to test this communication, we simulated we ran various tests throughout the different stages of our development. At the beginning we ran a simple test on the Arduino to see if we could connect to all of our sensors and servos and do basic read-write commands on them. From this test we learned that the thrusters and the radio chip reserve the same internal timer on the Arduino and thus cannot be used at the same time. We were able to get around this issue by

using an alternative software library (ServoTimer2) library which uses the second timer on the Arduino for the thrusters.

We also did tests to ensure communication between the Arduino and Pi is reliable. From these we learned it is important to clear the serial buffer of both microcontrollers before communicating as there may be extraneous data in the buffer. This is done with an initial handshake to confirm a successful message pass. The Pi will send a handshake message at a regular interval constantly reading and clearing its buffer waiting for a confirmation message while the Arduino constantly reads and clears the buffer until it reads the handshake message and then sends its confirmation message. At this point both systems know the other is on and fully initialized and that both buffers are clear.

4. Software Architecture

In this section we discuss the architecture of our codebase. We go into detail on the individual responsibilities of each of the different software modules and their relationships with each other. Including the module containing the state machine and how it affects the behavior of the craft. It is important to note that the code is still in development, and while most of it has been implemented it has not all been fully tested in the field.

4.1 Detailed Design & Analysis

Figure 8 shows a representation of the software architecture of our system.

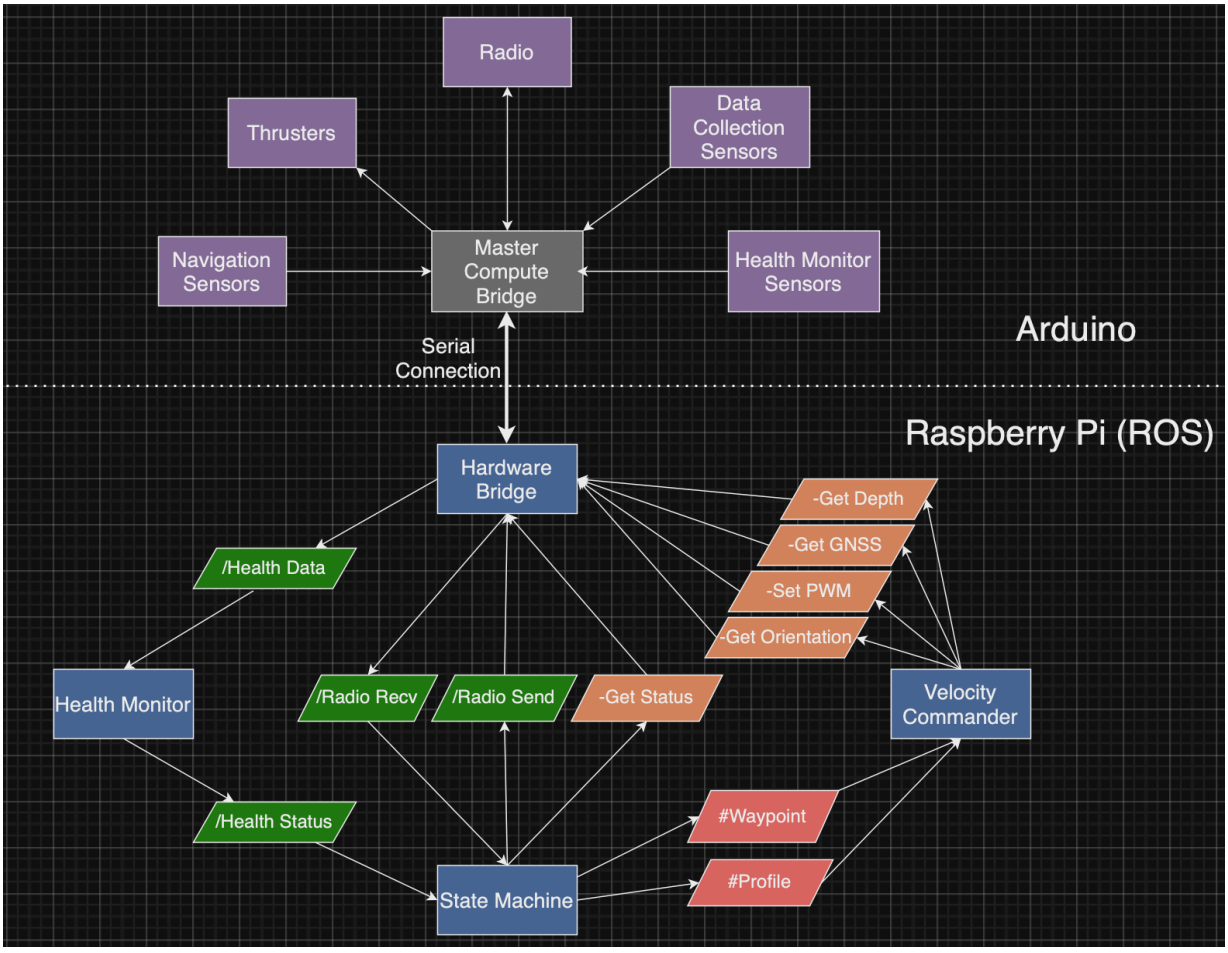


Figure 8: Software architecture diagram

The blue rectangles represent ROS nodes which are the parallel processes that are executing different tasks. The parallelograms connecting the nodes are color coded to represent

the different ways we are passing information between the ROS nodes. The green parallelograms represent ROS topics, where the node pointing to the topic may publish information to it at any time, and all nodes subscribed to that topic will receive the information. ROS services represented by the orange parallelograms are a way for nodes to request another node to do a singular task and return the result of the task. The red parallelograms represent ROS actions, which are requests from one node to another node to complete an continuous task and return the result, which may take awhile to finish. While the node is completing the action it can send back feedback to the node that requested it on the current status of the task. All of these components make up the ROS architecture that is running on the Raspberry Pi. The inner workings of each node is discussed below.

The Hardware Bridge Node manages communication from the Pi's side to the Arduino. This node follows the communication protocol discussed in the previous chapter. Everytime the node receives an incoming message from any of the services or topics, it will add it to its queue of outgoing messages. Whenever the Arduino is available it will send the next message in the queue and wait to receive a response from the Arduino. In the case of services such as Get GNSS or Set PWM, it will parse the response from the Arduino and return the key information as the result of the service. For topics it simply takes the response as a notification that the Arduino is ready to receive the next message (Code snippet in Appendix E).

The Hardware Bridge encodes messages to be sent to the Arduino and parse messages received from the Arduino. In order to distinguish between different types of messages we have a simple system to encode and parse data. The first two characters are capital letters and a colon, with the letter corresponding to the type of message. For example if the Arduino reads the first letter as "T", it knows the following message is a thruster command. Then all of the parameters relevant to that message are in a comma separated list following the colon. Because the messages are expected to be in such a specific format, we have had to write clear documentation on all the expected message types and their parameters.

The State Machine Node manages the state of our system. It does this by tracking key information such as the remaining waypoints and using input from the other nodes to determine the appropriate state. Figure 9 shows the graphical representation of the State Machine. The State

machine will also use the #Waypoint or #Profile actions when the state changes to either of these states (Code snippet in Appendix D).

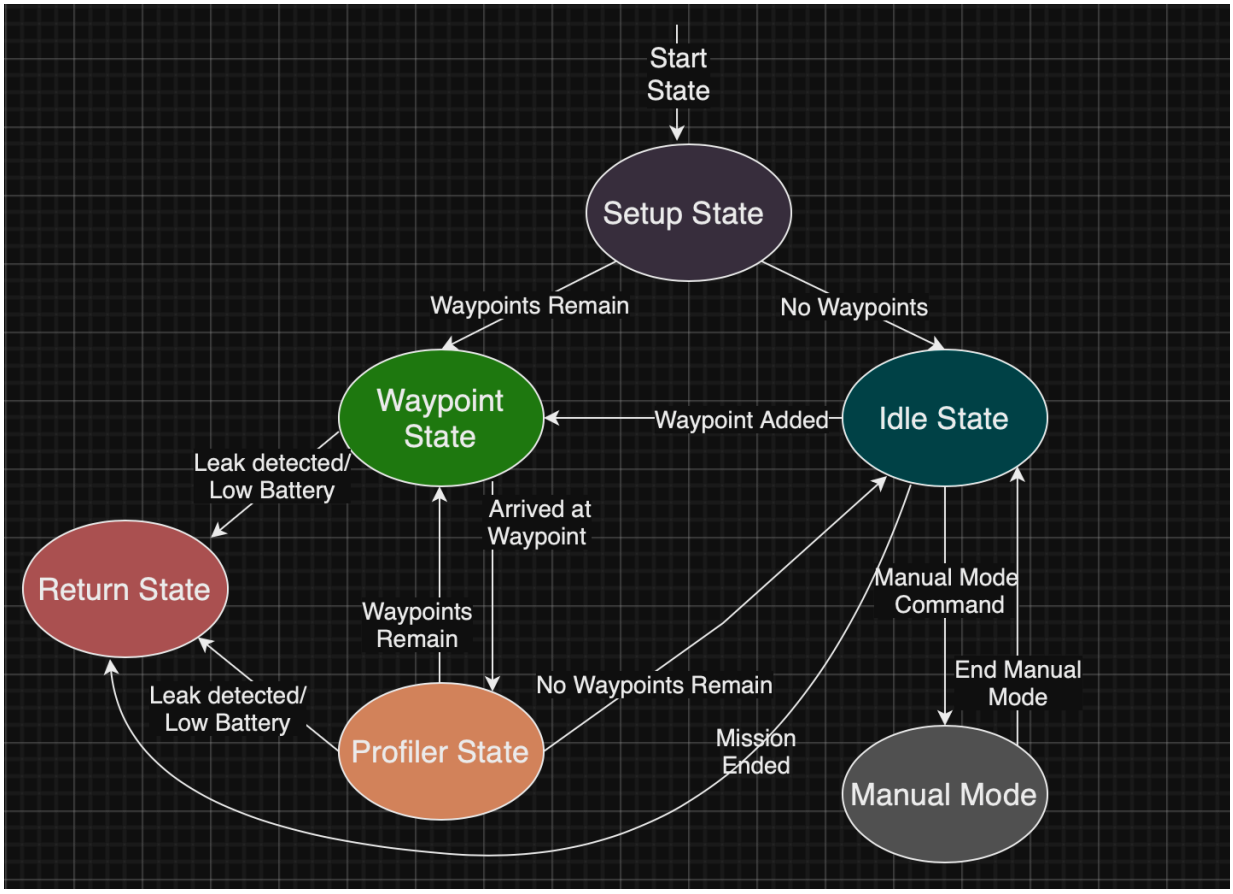


Figure 9: Diagram representation of State Machine Node

The profiler will initially start in the Setup state where time is given for the Arduino to power on and connect to all the hardware components it communicates with. During this time the State Machine Node will use "-Get Status" to request the Arduino's status until it gets an affirmative response that it is ready. Then if there are any remaining waypoints, the profiler will go to the Waypoint State. Immediately after changing to this state it will call the "#Waypoint" action and wait for the Velocity Commander Node to complete the action. After the action is complete, the state machine will transition to the Profiler State and call the "#Profile" action. During the profile, the State Machine Node is receiving messages from the other nodes. For example, if the "/Health Status" topic indicates a leak or a dangerously low voltage, the state machine will cancel the current action and go to the Return state. Otherwise, if there are no

interruptions, the state machine completes the Profiler state. If there are remaining waypoints in its itinerary, the state machine transitions back to the Waypoint state and then to the Profiler state. Once all the waypoints have been reached and profiled, the state machine transitions to the Idle state.

In the Idle state the system simply waits to receive input from the base station over radio. If no message is received for a set timeout, the state machine will go to the Return state. The user may also switch to Manual mode where the vehicle's velocity can be controlled via radio from the base station, although this is only useful if the vehicle is within line of sight.

Lastly the Return state which is much like the Waypoint state, but its coordinates are the home coordinates where the vehicle was launched. These coordinates can be overridden by input from the user. Once the vehicle makes it home, the mission is ended and it will stop all actions.

The Velocity Commander Node determines the appropriate commands to send to the thrusters based on the current action and inputs from sensors. The Node will remain idle until it gets the "#Waypoint" or "#Profile" action request, at which point it will begin running a control algorithm for that task (Code snippet in Appendix B and C). The waypoint algorithm will calculate the heading error and distance error from the profiler's current position to the waypoint coordinates. It will then set velocity of the thrusters to turn the profiler and accelerate it forward proportional to how large the errors are. After every iteration it will use the "-Get GNSS" and "-Get Orientation" services to update the profiler's coordinates and heading, and then call "-Set PWM" to set the thruster velocities. For the profile algorithm it will begin flipping to the vertical position using a PID controller, then it will dive down to the appropriate depth while still using the PID controller to maintain vertical orientation. Once it stays at the target depth long enough to collect all needed data, it shuts off all thrusters and uses the profiler's slight positive buoyancy to return to the surface. Similar to the waypoint algorithm it uses "-Get Orientation" and "-Get Depth" to update its data, and "-Set PWM" to control the thrusters. Once either of these algorithms are complete they will return the result back to notify the State Machine Node. Details on the control algorithm's implementation is expanded in the Navigation and Control Section.

While we have not finished implementing the Health Monitor Node here is its design: The Health Monitor Node will track data from the Health Sensors over time to assess whether there is a leak or low battery. It will track battery voltage over time to ensure that the Profiler can make it home without running out of batteries. The batteries we are using can become permanently damaged if the voltage drops below a certain threshold, so we must monitor it at all times to not push the system beyond this threshold. Since the profiler is filled with sensitive electronic components, even a small leak can interrupt its operation. If the changes in the data coming from the health sensors suggest either of these issues are approaching, the Health Monitor Node will immediately send a negative health status message to inform the State Machine.

4.2 Verification

To test the architecture of our system during development, we tested individual nodes by simulating the response they would receive from other nodes. For example when testing the State Machine node we simulated the response of the Velocity commander by having it automatically return a success message after a delay, rather than running the algorithm. Once we test the nodes in isolation we put the whole system together and simulate a deployment by doing a cart test. In this test we put the system on a cart and set a waypoint nearby to see if the algorithms were running as expected as we wheeled the profiler to the waypoint. This vastly speeds up the testing process as fully assembling the profiler and transporting it to a suitable testing location to put it in the water takes too much time to debug software issues.

5. Power Subsystem

This chapter explores the Power Subsystem of our craft. The subsystem is designed around bus-bars dedicated to power and ground. The craft is powered by high-capacity batteries from BlueRobotics. The operational time of the craft is estimated considering the current draw of each onboard component. The power subsystem's implementation details, including the connection of each thruster's ESC to the bus-bars and the use of buck converters for onboard computing, will be discussed in the following sections.

5.1 Subsystem Functional Overview

The power system of the craft revolves around four bus-bars shown below. There are two bus-bars dedicated to power and another two dedicated to ground. Each pair of bus-bars is rated to 48 V max dc and 150 A dc. We decided on using these specific bus-bars (Figure 10) to prepare for a maximum current draw of 100 A assuming each T200 thruster is sourcing 25 A at max pulse width modulation (PWM).

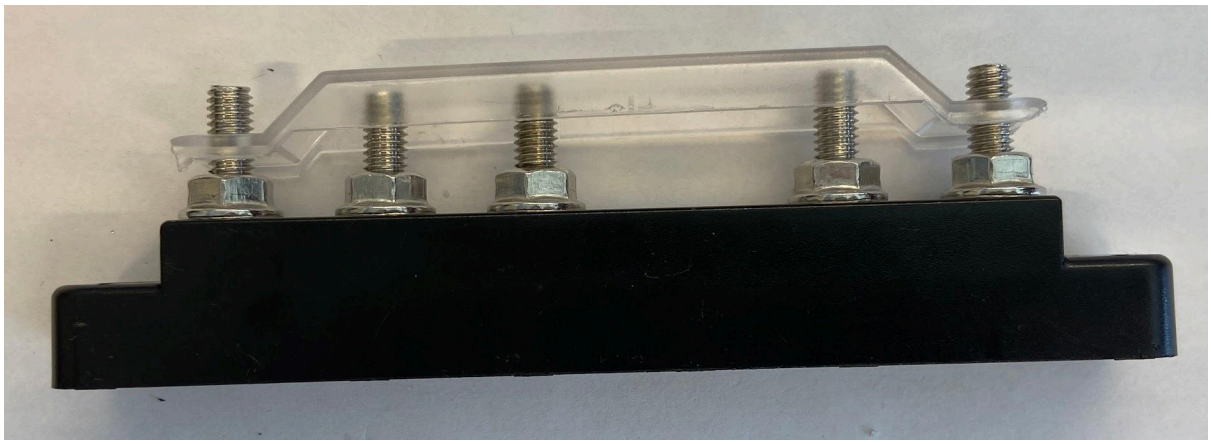


Figure 10: Bus bars

The craft is powered with BlueRobotics high capacity batteries shown in Figure 11. Each battery is rated to 14.8 V with an 18 Ah capacity. We decided on this battery due to our prior experience with BlueRobotics products and the ease of use. Furthermore, the battery is water rated and designed specifically for use with underwater robots. In total, there are four batteries onboard and each are wired in parallel giving us a total capacity of 72 Ah. This is purposely done

to provide use with the maximum range possibility and to provide additional headroom in anticipation for adding sensors in the future. To approximate the operational time of the craft, we used the equation 1 alongside the approximate current draw of each component onboard, assuming that only two thrusters will be active, given the horizontal distances traveled will be much greater than the depths to which our craft dives. The total current drawn by the system is about 50 A. Table 5 shows the current draw of each component.

$$OperationTime = \frac{Capacity(Ah)}{Current(A)} \quad (1)$$

Table 5: Power budget

Component	Current (A)
Pi 4	1
Arduino Mega	0.5
T200 (x2)	24 (48)
Sensors	0.5
Total	50

Figure 11: BlueRobotics battery

5.2 Detailed Design & Analysis

As mentioned earlier in the report, there are four thrusters each with their own ESC. Each ESC has power and ground cables which are connected to their respective bus-bars. To power the onboard computing, we use two buck converters to step down the 14.8 V to 5 V. One buck converter (Figure 12) is dedicated to powering the Raspberry Pi 4 and the other to the Arduino Mega 2560.

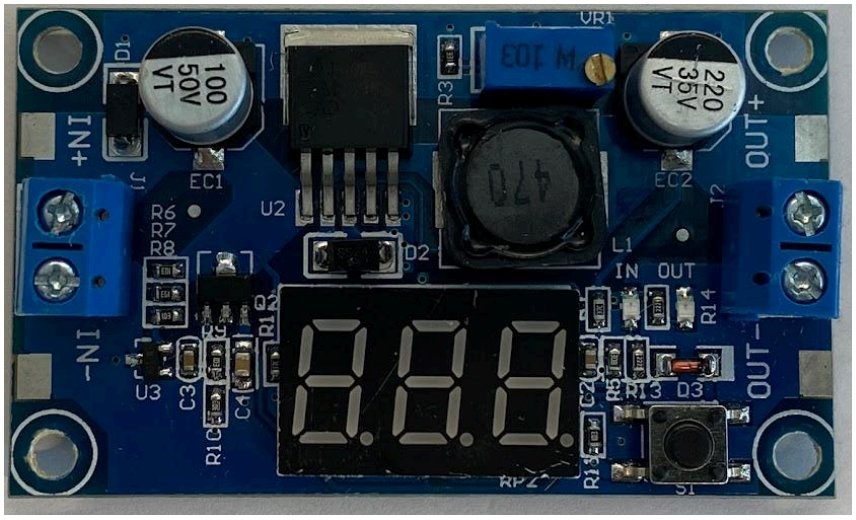


Figure 12: Buck converter

6. Scientific Sensing Subsystem

This chapter focuses on the Scientific Sensing Subsystem that we designed to collect water column data. We'll discuss its functionality, the alternatives considered for sensor selection, and our journey in designing a water sampler. We'll highlight the evolution of the sampler design, its scalability, and its ability to sequentially collect water samples at different locations in the water column. This subsystem is crucial in our quest to understand and preserve marine ecosystems.

6.1 Subsystem Functional Overview

This subsystem involves the actual goal of our project which is the collection of water column data. This data could mean any sort of data from temperature to turbidity or oxygen levels. Ideally the profiler is adaptable to add or remove sensors based on what the specific user is looking for, but for the scope of this project we focused on a few sensors that were most important to the marine researchers we talked to.

6.2 Alternatives Considered and Trade-off Analysis

For our project, there were three main alternatives considered. The first option involves buying sensor penetrators directly from BlueRobotics, as shown in Figure 13, and mounting them onto our end caps.

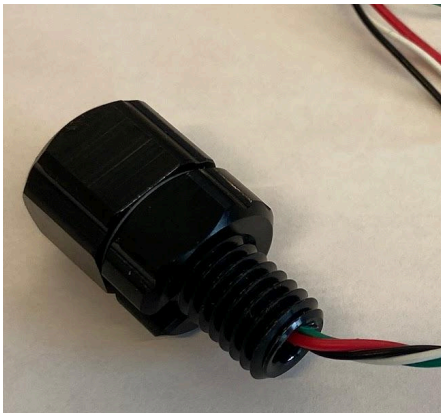


Figure 13: Picture of sensor penetrator

Currently, BlueRobotics sells sensor penetrators to measure temperature and pressure. Moreover, BlueRobotics also sells a ping2 sonar altimeter and echosounder shown in Figure 14.

Figure 14: Sonar altimeter and echosounder

The second option involves buying off the shelf sensors (Figure 15) from instrumentation manufacturers such as Atlas Scientific and integrating them into the computing subsystem. Atlas Scientific offers a wide range of environmental sensors, however, there are a few that meet our user requirements such as those measuring pH, dissolved oxygen, and conductivity.

Figure 15: Scientific sensor

The last option is to develop our own sensor payloads, specifically those that involve sample collection. We undertook this last option when deciding to implement our own water sampler, of which we have two designs, shown in Figure 16 and 18.

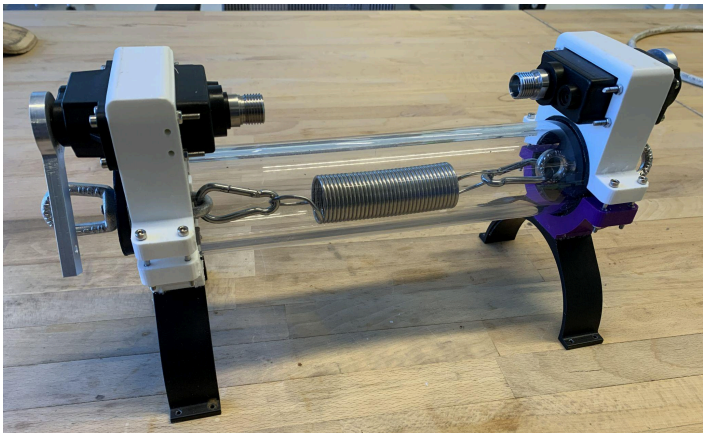


Figure 16: Previous design of water sampler

The previous design relied on using two underwater servos to force the rubber plugs on each end of the acrylic tube to align with the center of the tube. The servo used is shown in Figure 17. Once the plugs and center of the tube are aligned, the stainless steel spring pulls both ends inward. The previous design is deployed with each end of the tube naturally open, making this a flooded design. Both ends would remain open until the robot discovered a point of interest (large temperature gradient, high turbidity, desired depth, etc.) and then capture a sample. We have extensively tested the previous design at the test tank at MBARI and has proven to be reliable, however, it is not scalable due to the amount of servos required to collect one water sample.

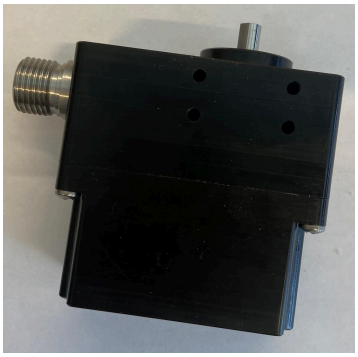


Figure 17: BlueTrail underwater servo

As a part of our senior design project, we further iterated on this design by keeping the spring-loaded mechanism for the individual sample tubes but redesigned the mechanism to close the ends of each sample tube. Furthermore, we wanted to settle on a design that only used one

servo to close up to four of the sample tubes. This ensured that the design would be scalable in the future to collect even more samples. Lastly, our design must keep the same functionality of collecting one sample sequentially. In other words, the water sampler must be capable of collecting water samples at different locations in the water column. Our current design shown in Figure 18.

Figure 18: Current design of water sampler

7. Health Subsystem

This chapter explores the Health Subsystem of our profiler, which uses sensors like voltage and pressure sensors to monitor its health and prevent damage. We discuss our choice of pressure sensor for leak detection and delve into how we use this data. We share our learnings from testing the system, emphasizing the importance of pressure tolerance. This chapter offers insights into the design, implementation, and testing of a key subsystem that monitors the profiler's integrity.

7.1 Subsystem Functional Overview

This subsystem monitors the health of the profiler by tracking a group of sensors to determine whether the profiler is able to complete the current mission or is forced to abort. These sensors include a voltage sensor to track battery life, and a leak and pressure sensor to assess whether there is a leak. This subsystem is vital to avoiding permanent damage to the craft as even an extremely small leak can disrupt the electronic components inside.

7.2 Detailed Design & Analysis

For leak detection we are using a barometric pressure sensor (the MPL3115A2 offered by Adafruit) which is connected to the Arduino. We are using a library provided by the manufacturer to read values of the sensor in units of millimeters mercury. The Arduino uses this pressure sensor to detect any cracks in its watertight seals. To monitor the health of its Power subsystem, our vehicle uses a generic voltage sensor to ensure that the battery voltages never drop to dangerously low levels.

7.3 Subsystem Verification

Doing testing for the Health monitor system was important for understanding when the profiler is safe to deploy. We had to do tests to narrow down what range of values are acceptable from the voltage and pressure sensor. The voltage is fairly simple as we can follow the manufacturer's instructions of 3-4.2 V per cell. For pressure we created a script to calculate the change in pressure over time. When we first deployed, this showed a change of around 1

millimeter mercury every minute, which we thought would be small enough to deploy safely. But in testing the profiler stopped working as intended due to a small leak. From this we learned that we had to be extremely strict with our pressure tolerance, and only accept a few hundredths of a millimeter mercury change every minute. With this stricter tolerance, we ran into no issues with leaking.

8. Structural Subsystem

The structural subsystem chapter covers the key aspects of developing and validating the structural components essential for the system's integrity. It begins with a functional overview, outlining objectives and requirements, followed by a detailed look at design, analysis, and simulation using methods like finite element analysis (FEA). Implementation details cover material selection and manufacturing processes. The chapter concludes with testing, evaluation, and verification to ensure the structural components meet performance and safety standards.

8.1 Subsystem Functional Overview

The structural subsystem is the foundation that supports all the functions of an autonomous marine vehicle (AMV). Much like the framework of a building, its primary goal is to provide stability, durability, and support for the intricate array of components and systems that comprise the AMV. From withstanding immense pressures in deep-sea environments to facilitating efficient maneuverability, the structural subsystem forms the fundamental backbone of the AMV’s functionality and performance. Overall, it needs to have housing for the GPS and LoRa antenna, mount adapters for the thrusters, support for the weight rack along the body of the AMV, and nose and tail cones to reduce drag underwater. Fusion 360 and Solidworks were used to design and simulate all the structural components and the final assembly.

An image of the final assembly is shown in Figure 19 below, with the 7 key components of the AMV labeled for reference. For components undergoing large amounts of stress during operations, such as the thruster mount, the weight rack, the nose cone, and the tail cone, we used PETG material for 3D printing. On the other hand, for parts that don’t require too much strength, such as the light mount and the antenna mast, we 3D printed with PLA material because it’s cheaper and more readily available.

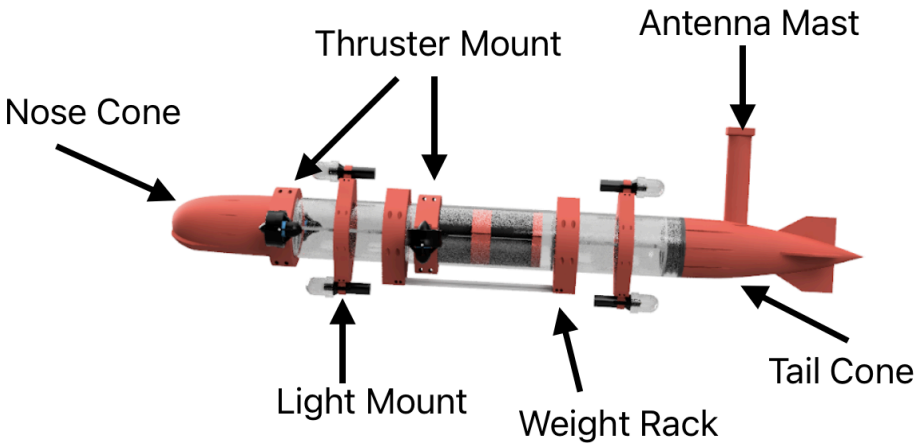


Figure 19: Side view of CAD assembly

8.2 Detailed Design & Analysis

8.2.1 Thruster mounts

The component experiencing the highest stress is the thruster mount. It has to withstand a thruster force of 50 N from the T200 thruster shown in Figure 20. Consequently, Finite Element Analysis (FEA) has been conducted on the thruster mount to guarantee its durability and reliability under normal operating conditions, thereby mitigating the risk of failure.

Figure 20: BlueRobotics T200 thruster

The CAD model of the thruster mount is shown in Figure 21. The thruster mount is used to attach both thrusters propelling water in both horizontal and vertical directions. This means that the part must endure the 50 N max thrust in both directions as well. The magnitude and direction of the 50 N force for the horizontal thruster configuration is shown in Figure 22. Moreover, Figure 23 also illustrates this information but for the vertical thruster configuration.

Figure 21: CAD model of thruster mount

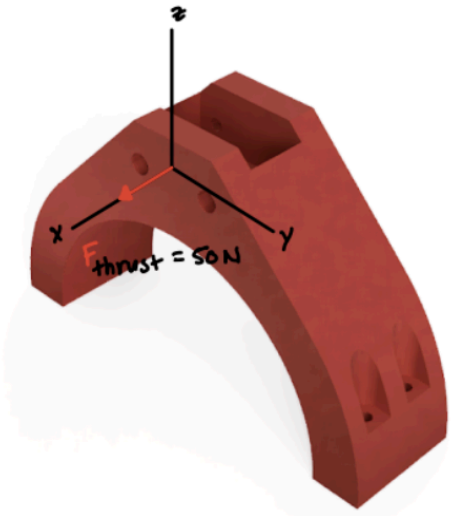


Figure 22: X-direction of T200 thrust in relation to thruster mount

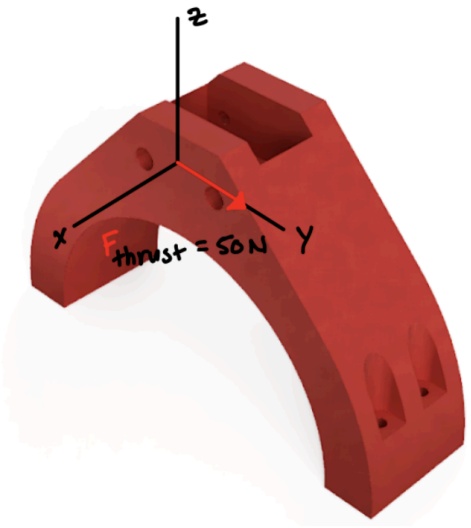


Figure 23: Y-direction of T200 thrust in relation to thruster mount

During standard operational procedures, the thruster's maximum output capacity is 50 N, and the operational limit of the acrylic tube extends to a depth of 20 m. Consequently, testing parameters are established as follows:

Initially, a hydrostatic pressure equivalent to a depth of 20 m is applied, resulting in a pressure of 0.2 MPa. This pressure is calculated using equation 2, wherein ρ denotes the density of water, g represents gravitational acceleration, and h signifies the depth of the AMV.

$$P = \rho \cdot g \cdot h \tag{2}$$

$$P = 1000 \cdot 10 * 20$$

$$P = 0.2\ MPa$$

In addition to the water pressure, a force of 50 N is exerted either along the x or y axis, contingent upon the specific configuration of the mounted thruster.

Regarding boundary conditions, two fixed constraints are imposed on the lower section of the component labeled in Figure 24.

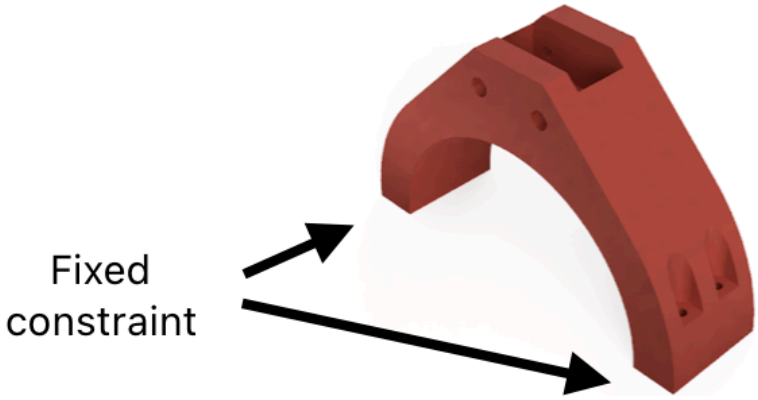


Figure 24: Fixed constraints on thruster mount

Concerning the material composition of the thruster mount, the mounts were fabricated utilizing a 30% infill with a rectilinear pattern and a wall thickness of 11.5 mm, as shown in Figure 25. A visual representation of this pattern is provided in the accompanying image. These details ensure comprehensive testing conditions and robust assessment of the AMV's performance under varied operational scenarios.

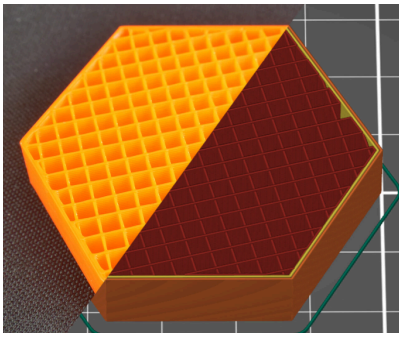


Figure 25: Rectilinear infill pattern with 15% infill

Due to the rectilinear shape of the infill and the enhanced wall thickness, the 3D-printed thruster mount can weigh less with some reduction in strength. Therefore, when interpreting the results from the FEA analysis, a safety factor of 10 should be considered to ensure the actual performance of the part exceeds the performance predicted by the FEA simulation. During operation, it's crucial to ensure that the deformation of the part remains within the elastic region. Therefore, we must confirm that the highest von Mises stress is below the yield strength of the PETG material, which ranges from 4.79 MPa to 5.29 MPa [21], with a safety factor of 10.

Note that, compared to Figures 22 and 23, the directions of x and y are switched in the FEA analysis. Additionally, in Figure 21, it shows an empty gap on top of the thruster mount, but in reality, this space will be filled with a solid block attached to the thruster. To ensure the simulation is realistic, a block of similar size and the same material properties is placed inside the gap in the FEA software.

Case 1:

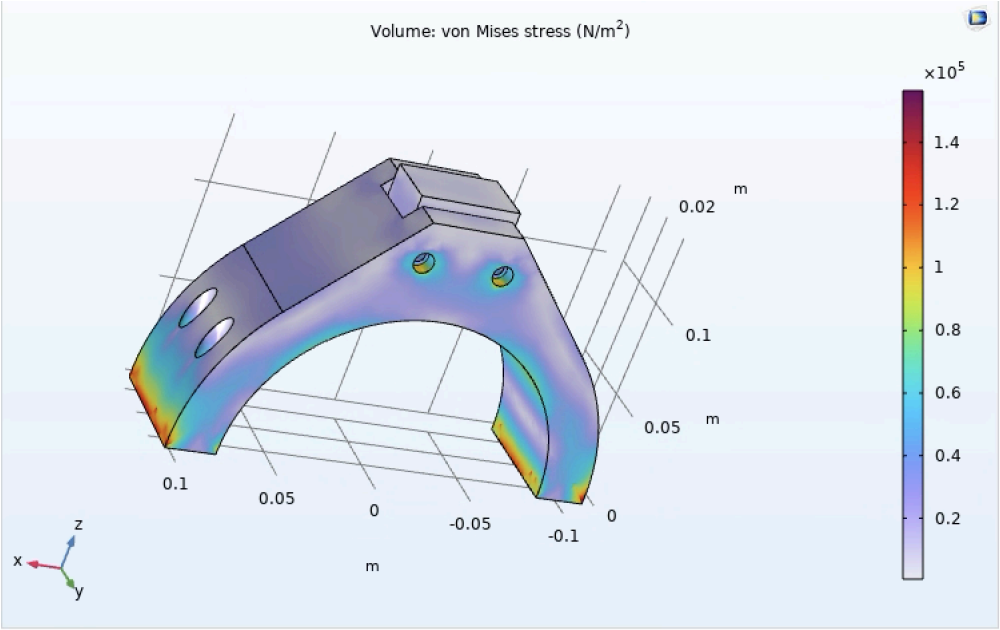


Figure 26: Von Mises stress of the thruster mount in the x-direction

Figure 26 shows the von Mises stress of the thruster mount when the force of the thruster is in the x-direction and under 20 m of water. The bottom of the thruster mount has a high concentration of stress, with the highest value being around 0.15 MPa, which is well below 4.79 MPa.

Figure 27 shows the scaled deformation pattern of the thruster mount under an applied load in the x-direction. The scale factor of the deformation is 2087, as shown in Figure 28. Therefore, the actual deformation will be far less than what appears in Figure 27. Additionally, the direction of the deformation aligns with the direction of the applied load.

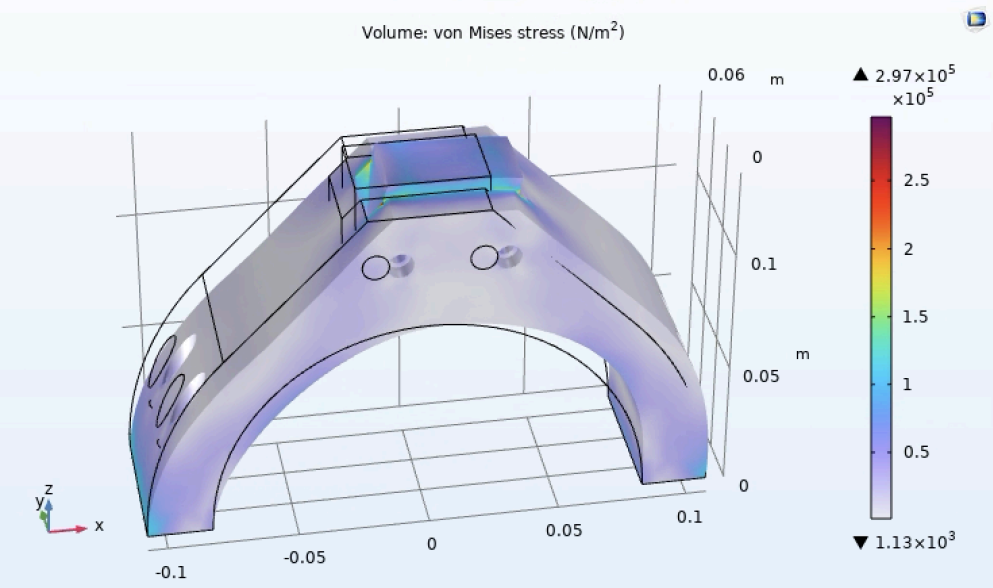


Figure 27: Von Mises stress in x-direction with scaled deformation

Figure 28: Scale factor of the deformation plot in x-direction

To obtain a quantitative understanding of the deformation's magnitude, a displacement magnitude plot is generated, as shown in Figure 29. The largest displacement measures around $7*10^6$ m.

Case 2:

When the thruster load is applied in the y-direction, note that the adapter is extended. This adjustment was made because, during deployment testing, we discovered that the AUV requires more torque on the water's surface to counteract the current. Therefore, we extended the adapter arm to increase the torque. In this configuration, the distribution of stress changes, with most of the stress still concentrating around the bottom of the thruster adapter. The highest von Mises stress measures around 1.63 MPa, which is less than 4.79 MPa, the yield strength, with a safety factor of 10.

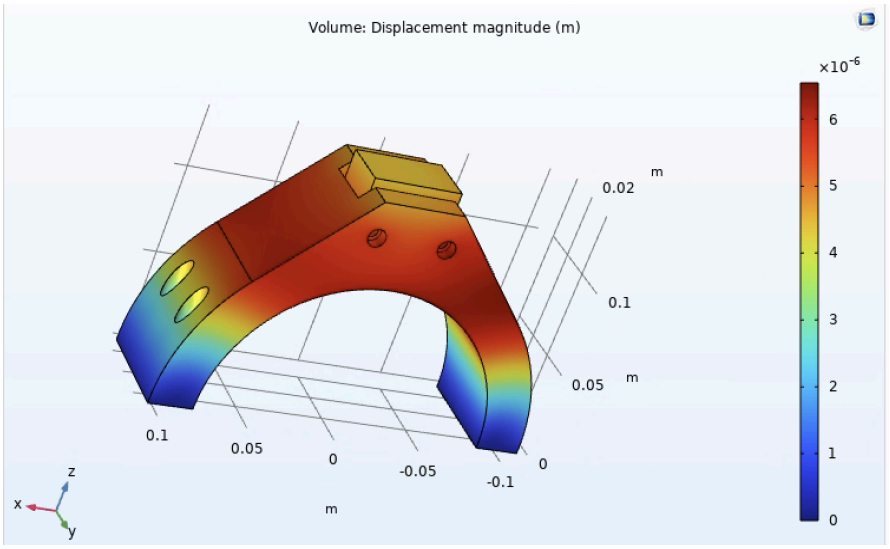


Figure 29: Displacement of the thruster mount in the x-direction

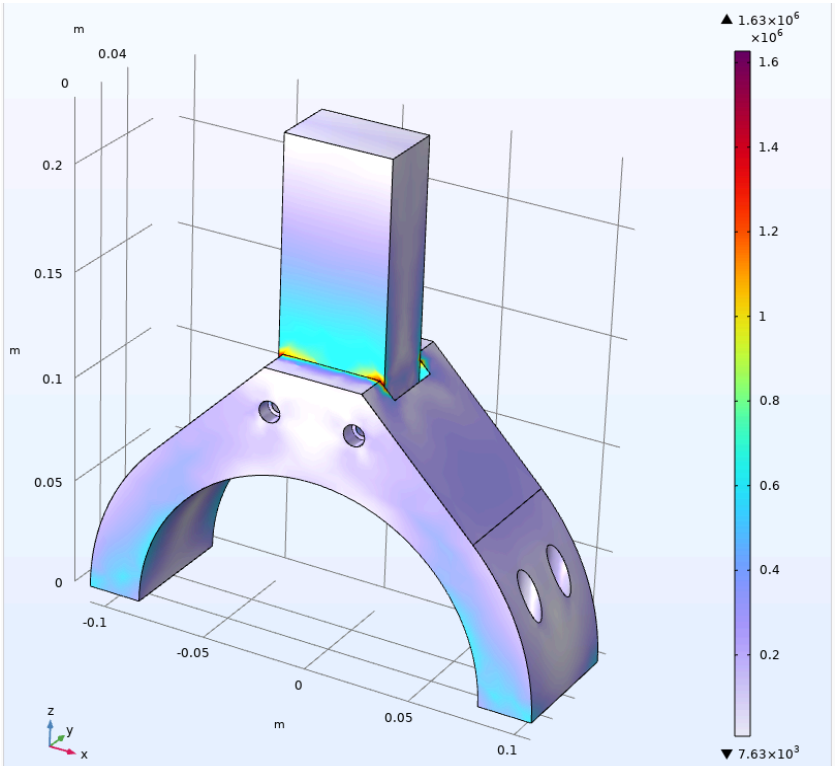


Figure 30: Von Mises stress of the thruster mount in the y-direction

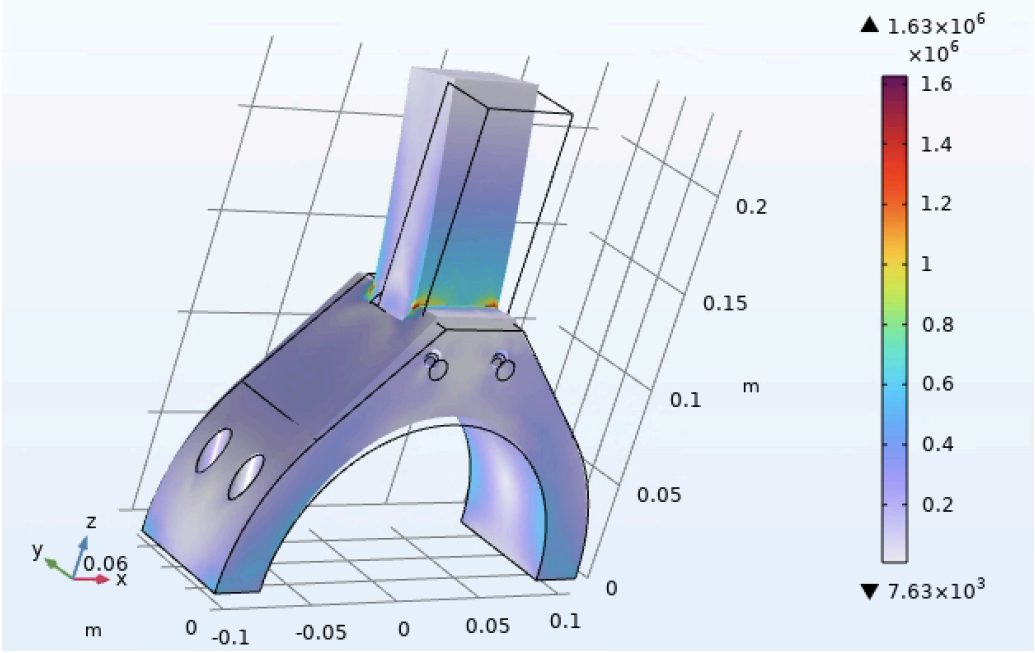


Figure 31: Von Mises stress of the thruster mount in the y-direction with scaled deformation

Figure 32: Scale factor of the deformation plot in y-direction

Again, the deformation pattern of the thruster mount is shown in Figure 31 with the same scale factor of 100 from the simulation software in Figure 32. The direction of the deformation is aligned with the direction of the applied load, which is in the y-direction.

Last but not least, a displacement magnitude plot is shown in Figure 33, and most of the displacement occurs at the top of the part, with the highest displacement value being 0.0269 mm.

In conclusion, the FEA results of the thruster mount in both the x and y directions show that the von Mises stress is far below the yield strength with a safety factor of 10. This implies that the displacement will be elastic. Additionally, the displacement pattern aligns with the direction of the force. As a safety check, the maximum displacement value under both loading conditions is also minimal.

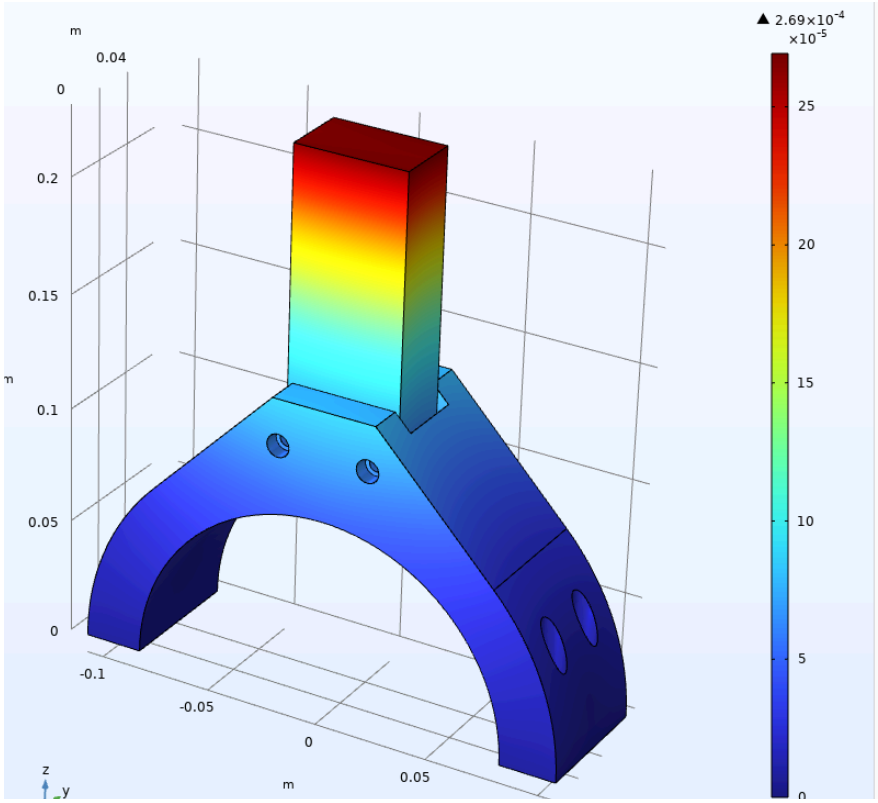


Figure 33: Displacement of the thruster mount in the y-direction

8.2.2 Ballast rack

Throughout the development stage of the AMV, we conducted multiple iterations of the design and 3D printing, each time perfecting not only the functionality but also the ease of assembly and manufacturing. In the later stages, we discovered that the tail cone was very difficult to 3D print and wasted a significant amount of printing material. During the implementation stage, we also found that the tail cone was too heavy and started to pull the end cap off the tube. We deemed it unnecessary and discarded the idea of having a tail cone. Without the tail cone, the antenna mast, which was previously designed to be mounted onto it, was changed to be mounted on top of the weight rack with a newly designed modular mount.

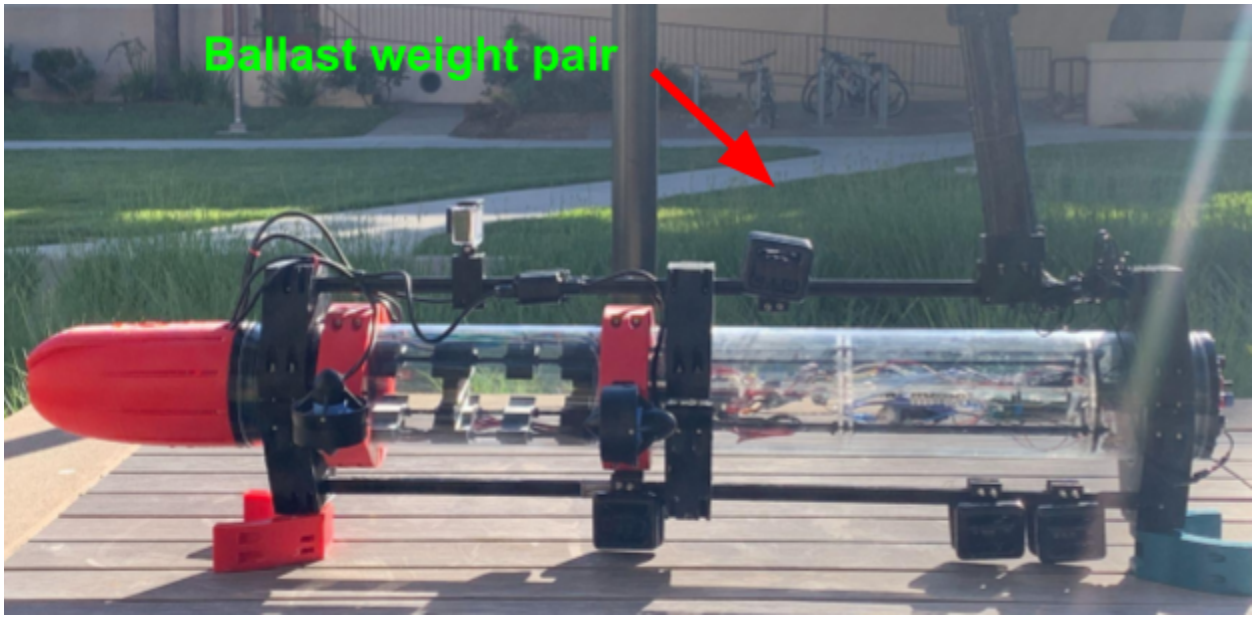


Figure 34: Picture of craft with location of ballast weights indicated

For the weight rack, we discovered that achieving horizontal balance in the AMV requires extending its length. This extension provides additional space for adjusting weights. Furthermore, through the modular attachment of the antenna mast, we identified the potential for the weight rack to serve as a platform for attaching various scientific sensing equipment. This feature enhances the value of the AMV, enabling users to customize the onboard equipment according to different tasks. Consequently, we have opted to extend the weight rack along the entire length of the tube depicted in Figure 34.

As illustrated in Figure 34 above, four pairs of 3.3 lbs ballast weights are affixed to the weight rack. To achieve horizontal balance with the newly devised weight rack system, it is imperative that the center of buoyancy align directly above the center of gravity, as depicted in Figure 35.

Using the green dot on the right side of the rectangle as our origin, the CG was approximated. The final location of both from the origin are shown in the Table 6 using equation 3.

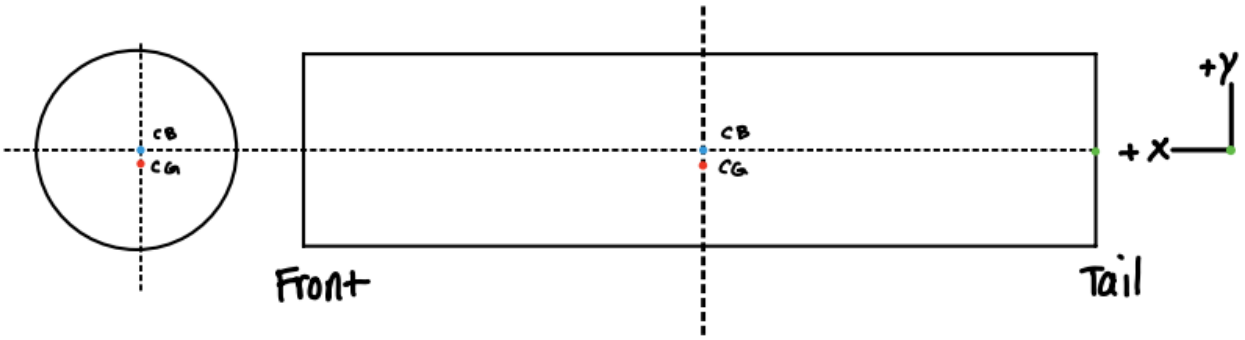


Figure 35: Location of center of gravity and center of buoyancy of the craft

$$X_{CG} = \frac{\Sigma W X}{\Sigma W} \quad (3)$$

Table 6: Approximation of center of gravity and center of buoyancy

	Coordinates (X, Y)
Center of gravity (CG)	(24 in, -3 in)
Center of buoyancy (CB)	(24 in, 0 in)

By adjusting the position of the weights along the weight rack through empirical tuning, we can align the center of gravity and buoyancy on the x-coordinate. One trade-off that had to be made is the horizontal stability while traveling across the surface and the vertical stability while diving. As shown in Figure 34, in the current design, the majority of the weights are placed on the bottom rack. This helps the craft to maintain roll stability while turning or fighting against the current. However, this configuration results in an uneven distribution of weight while the vehicle dives, which motivates the need for pitch control, which is discussed in the Navigation & Control chapter. Therefore, in the future, it is ideal for the craft to achieve both horizontal and vertical stability.

8.3 Subsystem Verification

After thoroughly testing the structural components of the AMV, we conducted multiple deployment tests in different water conditions and different operation modes. All the parts functioned reliably without any signs of deformation.

9. Communications Subsystem

This section discusses the Communication Subsystem which is intended to allow for long distance communication between the profiler and the base station. We did not have time to implement this subsystem or the base station UI but our proposed design is discussed below. We were able to do some preliminary testing to verify that our design makes sense.

9.1 Subsystem Functional Overview

In the future, the vehicle's communications subsystem will allow the vehicle to transmit the data it has collected to a base station on the shore. The communications subsystem will also enable the craft to accept commands from the onshore base station, including new waypoints.

9.2 Detailed Design & Analysis

Internal to the vehicle is an Adafruit RFM95W LoRa Radio Transceiver chip (Figure 36), as seen on the component block diagram. This chip, when attached to our antenna (Great Scott Gadgets ANT700 - 300 MHz to 1100 MHz), is able to transmit messages 2 km with line of sight.

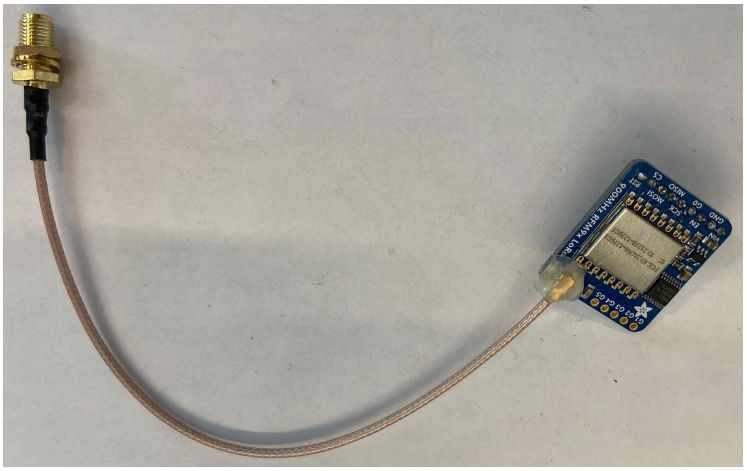


Figure 36: Radio transceiver chip

These messages will be transmitted to an identical chip and antenna connected to an Arduino Mega. The Arduino will be plugged into the base station computer through a USB connection. This setup will allow us to send messages back and forth between the base station

and profiler up to 2 km away. The profiler will primarily be sending messages containing data on the status of the profiler such as its GPS position and current state (e.g. profiling or waypoint navigation). In the future we will be able to use this data to update the user interface on the base station computer so the user is up to date on the profiler's status.

The base station will primarily be sending messages commanding the profiler to do different tasks. This could be adding another waypoint to the list, canceling the current waypoint, going into manual mode, or ending the mission. This will give the user on the fly control over the profiler's movements. Although communication will be limited to when the craft is above water as the radio signals cannot travel through water. This is also why it is so important that we have the antenna mast on the craft to raise the antenna above the waves. When it comes to the communication protocol of our system we plan in the future to implement a simple send and acknowledge protocol. Messages will be tagged with an id the increments with each message. The sender will continuously send the message at a set frequency such as 5 Hz until it receives an acknowledgement message from the receiver with the same id. The receiver will read all incoming messages and store recently received ids. If it receives a new id it will repeatedly send an acknowledgement message back with that id until it receives a message of a different id signifying that the sender has received the acknowledgement and has moved onto the next message. The reason we are sending the messages so frequently is because it is likely that messages may be lost or blocked by the water. So having redundancy gives us a much greater chance that at least one message goes through.

9.3 Subsystem Verification

As discussed we have not implemented this subsystem but we have done some preliminary testing to verify its feasibility. We created a test where a sender sends messages at 5 Hz with incrementing ids, and the receiving sends back an acknowledgement to the message. We then took this test outside to see the message miss rate at different distances and with obstacles in the way. In general the antennas performed quite well, dropping only a few messages even over long distances and behind obstacles. This is quite promising for our design but much more testing would need to be done in an actual marine environment to see how much water affects message transmission.

10. Navigation & Control Subsystem

In this section we discuss how the vehicle's mechanical configuration and software contribute to controlling its movements. First, we discuss the tradeoff analysis we completed on the six candidate mechanical configurations. We then discuss how this subsystem achieves its control objectives: heading control, "flipping", and depth-control while diving.

10.1 Subsystem Functional Overview

The profiler uses an internal GPS receiver and compass to determine its global coordinates and its heading; it uses two pairs of thrusters to steer and propel itself to its goal coordinates and to subsequently dive into the water. The craft is equipped with a depth sensor to be able to control how deep it dives in the water.

10.2 Alternatives Considered and Tradeoff Analysis

To achieve the functionality required of this subsystem, we compared several design alternatives.

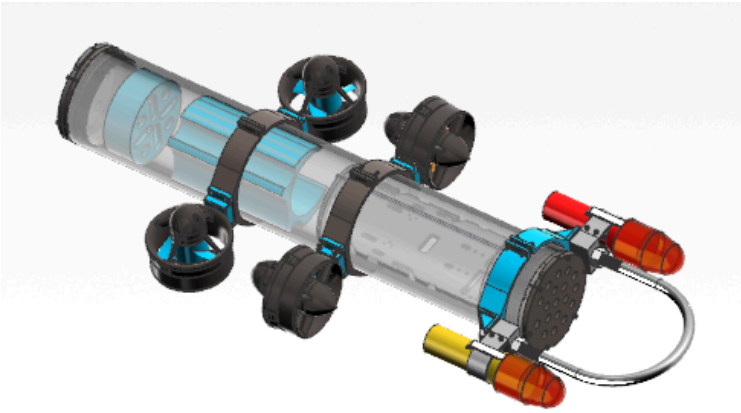


Figure 37: Horizontal and vertical thrusters (configuration 1)

Configuration 1, shown in Figure 37, uses two pairs of thrusters to provide forward thrust along the body of the profiler and also to change the orientation of the profiler between horizontal and vertical orientation. This design achieves multiple degrees of freedom with this configuration. But it's not very power efficient due to drag and load from extra motors.

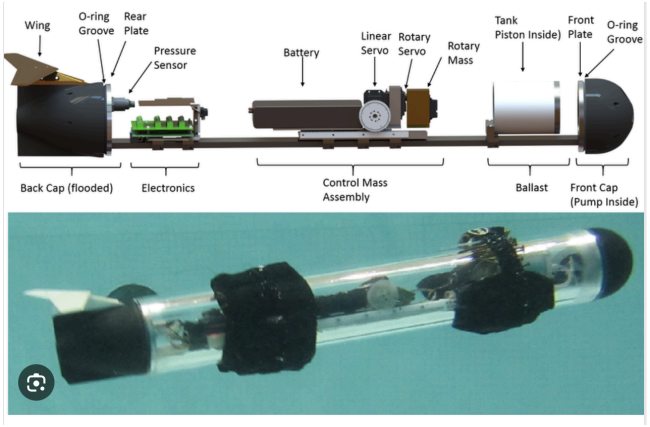


Figure 38: Mass shifter with thrusters (configuration 2) [Used without permission] [22]

Configuration 2, shown in Figure 38, utilizes a mass shifter to shift the battery, adjusting the center of gravity and buoyancy to alter the orientation. It employs thrusters to provide thrust. It's power efficient but it takes a lot of space inside the tube for the mass shifting mechanism.

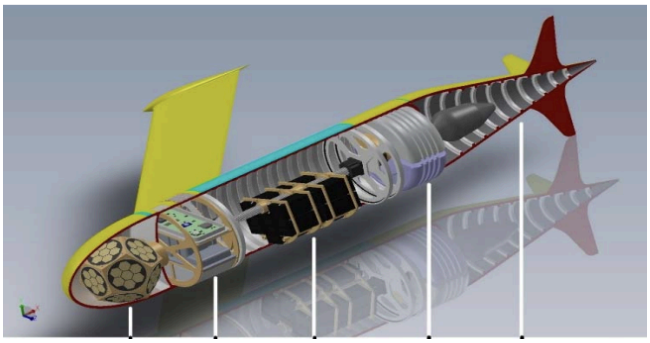


Figure 39: Oil bladder & hydrodynamic fairing (configuration 3) [Used without permission] [23]

Configuration 3, shown in Figure 39, uses an oil bladder to transfer liquid, thereby altering the center of gravity and buoyancy for orientation control. It employs one thruster at the end of the vehicle for power. It’s power efficient because it only uses one thruster. However, the moving of the liquid adds to the complexity of the system, and increases the risk of leaking.

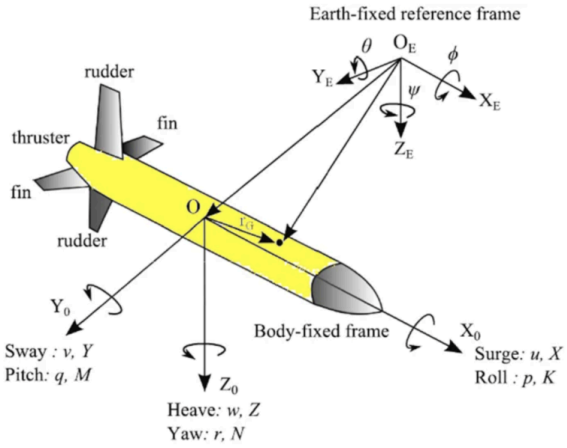


Figure 40: Vertical stabilizer and rudder with thrusters (configuration 4) [Used without permission] [24]

Configuration 4, shown in Figure 40, utilizes a single thruster for thrust and employs a rudder for orientation control. It also uses a fin to stabilize itself. It's power efficient but has limited degrees of freedom.

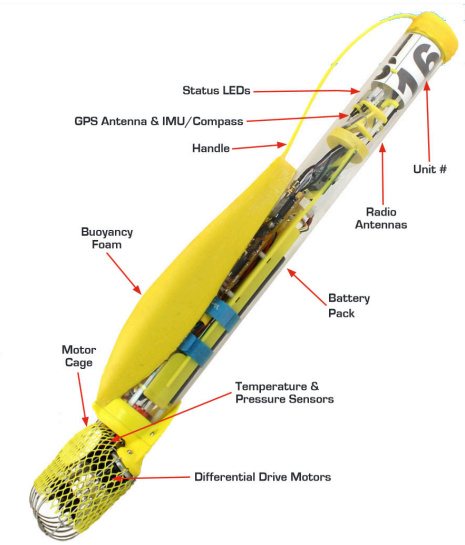


Figure 41: Two angled thrusters with buoyancy foam (configuration 5) [Used without permission] [25]

Configuration 5, shown in Figure 41, uses two thrusters that are offset at an angle to thrust forward and change orientation by speed control. It also uses foam to help with the

balancing of the craft while diving. It's power efficient but requires extremely fine-tuned control of the speed of the thrusters and very precisely tuned ballasting to its environment.

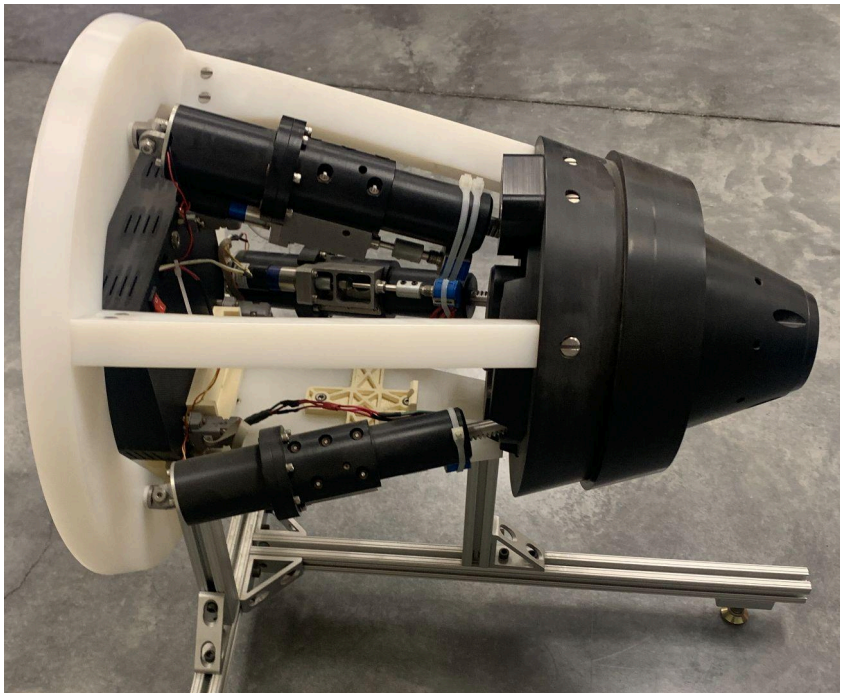


Figure 42: Thrust vectoring (configuration 6)

In Configuration 6, shown in Figure 42, the vehicle thrust is directed using the thrust-vectoring mechanism for power and orientation control. It's power efficient but places significant torque on the thruster joint. It also has a limited range to move in all degrees of freedom.

We identified and ranked five characteristics that were required of our final mechanical configuration, and they are described in Table 7. Each characteristic, called a "Driver," is assigned a qualitative weight that denotes its importance. In Table 8, we assess each of the six possible configurations according to these five drivers, scoring them out of five. By calculating the weighted scores of each of the configurations, we arrived at the conclusion that Configuration 1: Horizontal and Vertical thrusters, scores highest.

Table 7: Driving factors in tradeoff analysis

Ranking	Drivers
5	Versatility: Examines the system's performance in environments including open ocean, surf zones, freshwater lakes.
5	Controllability: Assesses the profiler's ease of navigation through software and operator intuitiveness.
4	Simplicity: Evaluate the design for ease of troubleshooting or future fixes.
3	Power Efficiency: Assesses the system's efficiency by comparing tasks accomplished to power consumption.
2	Affordability: Cost of components needed for implementation.

Table 8: Results of tradeoff analysis

Drivers/design alternatives	Weight of each driver	Mass shifter	Oil Bladder w/ Thrusters	Thrusters w/ Buoyancy Foam(SwarmDiver)	Thrust vectoring	**Horizontal and Vertical Thrusters(4 Thrusters)**	Vertical Stabilizer and Rudder w/ Thrusters	Vertical Stabilizer and Rudder w/ Thrusters
Versatility	5	4	4	5	3	**5**	4	4
Controllability	5	3	3	3	5	**5**	4	4
Simplicity	4	3	2	2	2	**5**	3	3
Power Efficiency	3	5	5	4	4	**3**	4	4
Affordability	2	3	2	5	3	**2**	1	4
Total score		68	62	70	66	**83**	66	10

10.3 Detailed Design & Analysis

10.3.1 Mechanical Configuration

We chose Configuration 1 to be our final design. Two pairs of T200 BlueRobotics thrusters (Figure 43) are affixed to the craft. Each thruster is connected to the craft with a 3D printed adapter made of PETG which is then connected to a mount on the craft. Our analysis of the strength of our adaptors is in the Structural Subsystem chapter. One thruster pair is mounted in the midsection of the craft and is responsible for propelling the craft forwards as well as pivoting when the thrusters are applied differentially. We call these the Parallel thrusters. Moreover, they serve an important role for controlling the descent and ascent of the craft when diving. The second pair of thrusters located in the end of the craft are important for the craft to flip from horizontal to vertical orientation and vice versa. We call these the Perpendicular thrusters. Once activated, these thrusters pull one end of the craft into the water until it stands vertically—after which it is able to dive and descend.

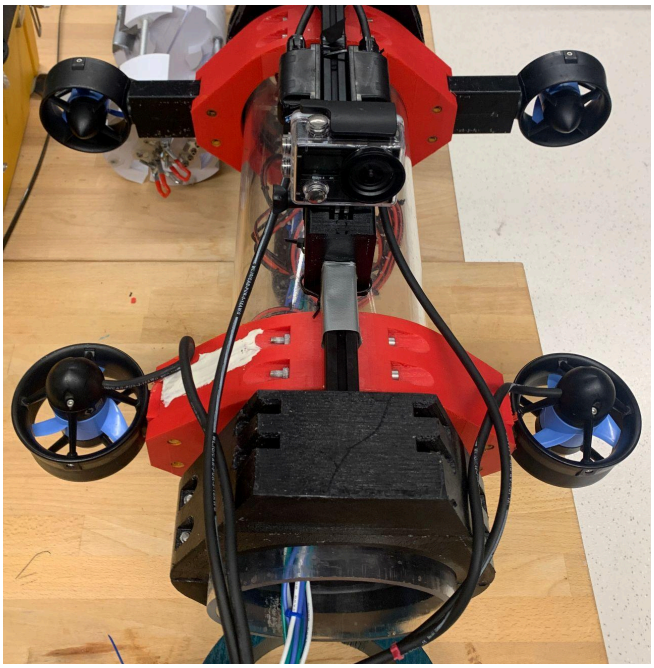


Figure 43: Photo showing Parallel (top) and Perpendicular (bottom) thrusters

10.3.2 Control Law

The vehicle uses a PID control law to achieve its three primary control objectives: heading control, “flipping”, and depth-control while diving. The control block diagram for our PID control law for heading control and flipping is shown in Appendix W, and the control block diagram for our PID control law for depth-control is shown in Appendix X.

Heading Control

The profiler calculates the compass bearing, θ, using equation 4, where (φ1, λ1) is the vehicle’s current location in latitude, longitude and (φ2, λ2) is the vehicle’s target waypoint, and Δλ is the difference in longitude, λ2 - λ1.

$$\theta = \operatorname{atan2}(\sin \Delta\lambda \cdot \cos \varphi 2 , \cos \varphi 1 \cdot \sin \varphi 2 - \sin \varphi 1 \cdot \cos \varphi 2 \cdot \cos \Delta\lambda) \quad (4)$$

The vehicle uses this compass bearing and its sensed heading (via the Adafruit 9-DOF Absolute Orientation IMU Fusion chip) to calculate its heading error. This error drives a PID control law that pivots the vehicle towards its waypoint. The PID gains for this controller were empirically tuned during field tests at Monterey Bay.

“Flipping”

“Flipping” is the term we have coined for when the vehicle transforms from horizontal surface-transit mode to vertical diving mode. At rest, only the gravity force and the buoyant force are present, as shown in Figure 44. When the vehicle wants to flip to vertical diving mode, the Perpendicular thrusters activate, applying a force of F_{app} and causing a torque on the vehicle, as shown in Figure 45. Two primary forces interfere with flipping: the buoyancy force, indicated by F_{buoy}, and the gravity force acting at the vehicle’s center of mass, indicated by W. Due to these disturbances, we elected to implement another closed loop controller to control the vehicle’s flip. The control block diagram for this control law is shown in Appendix W. The PID gains were empirically tuned at the test tank at MBARI.

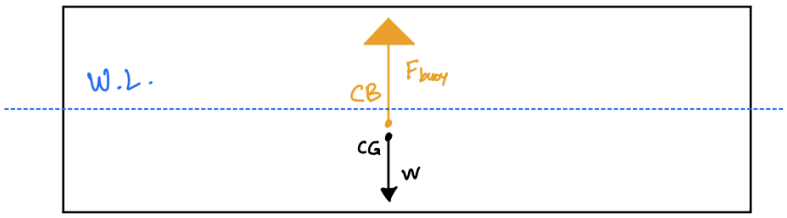


Figure 44: FBD of craft before flipping

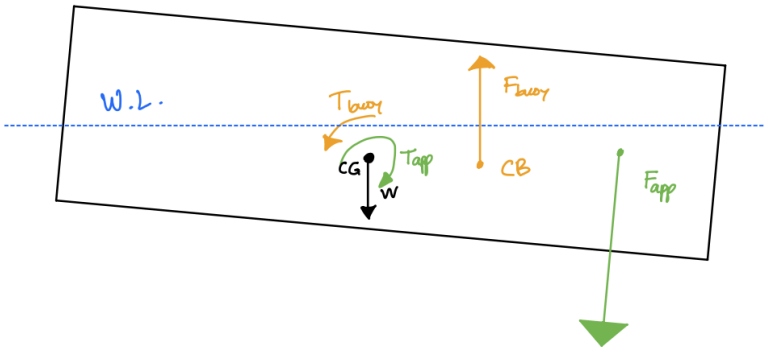


Figure 45: FBD of craft while flipping

Depth-control

We implemented a closed loop controller that enables our vehicle to hover at commanded depths in the water. The control block diagram is shown in Appendix X. Once flipped into vertical diving mode, the vehicle uses its onboard depth sensor (Bar30 High-Resolution 300 m Depth/Pressure Sensor) to measure its depth error. In vertical diving mode, the vehicle uses its Parallel thrusters to pull itself downward to its commanded depth. PID gains were empirically tuned.

10.4 Subsystem Verification

10.4.1 Heading-control & Waypoint Navigation

We field-tested our vehicle's capability to control its heading and navigate to GPS waypoints using a chartered boat in Monterey Bay. Figures 46, 48, 50 display the profiler's trajectory as it steers itself to a commanded waypoint. The vehicle considers itself "arrived" at the waypoint when it detects it is within five meters of the waypoint. We felt this was reasonable

given that our GPS hardware has an accuracy of approximately 1.8 m. Upon arrival, the vehicle was commanded to return back to its home coordinates. Figures 47, 49, 51 display the profiler's controlled heading versus the compass angle of its commanded waypoint.

The graphs show a partially successful demonstration of waypoint navigation. The vehicle successfully arrives at the waypoint. The vehicle's heading control steers the vehicle into alignment with the compass angle. However, *after* the vehicle arrives at its target waypoint and is supposed to start navigating towards its home coordinates, the vehicle instead navigates to the *wrong* set of coordinates. After we analyzed both our software control law and the API provided by the GPS vendor, we observed that the GPS inexplicably loses its connection with the satellites at the exact moment when the profiler stores its home coordinates, causing the vehicle to start navigating to garbage home coordinates. This is shown in the nonsensical trajectory the profiler takes after reaching its waypoint. We have been unable to consistently reproduce this software/hardware issue, and resolving it has been delegated to future work.

10.4.2 Depth-control & Flipping

We field-tested our vehicle's ability to flip and dive and hold a commanded depth at MBARI's test tank. Figure 52 shows the performance of the craft's depth-control when commanded to hold depth at five meters. Figure 53 shows the performance of the craft's ability to flip. While flipping, the vehicle is being commanded to maintain its vertical orientation; only when the vehicle senses it's within 10 degrees of vertical (90 degrees) is the depth-controller actively commanding the vehicle's thrusters to pull the vehicle downwards.

Figures 47, 49, 51 show the vehicle's heading before the nonsensical trajectory. The controller successfully steers the vehicle towards its waypoint's compass bearing with less than 7 degrees of error after settling.

Our results partially demonstrate our vehicle's ability to flip and dive. In Figure 52, the vehicle overshoots its commanded depth. Due to constrained time at MBARI, we were unable to complete tests long enough for the vehicle to settle. In Figure 53, the vehicle remains flipped for the majority of the test, to the credit of the closed-loop controller. Future performance can be improved by further tuning of the PID gains of both the depth-controller and the flip-controller.

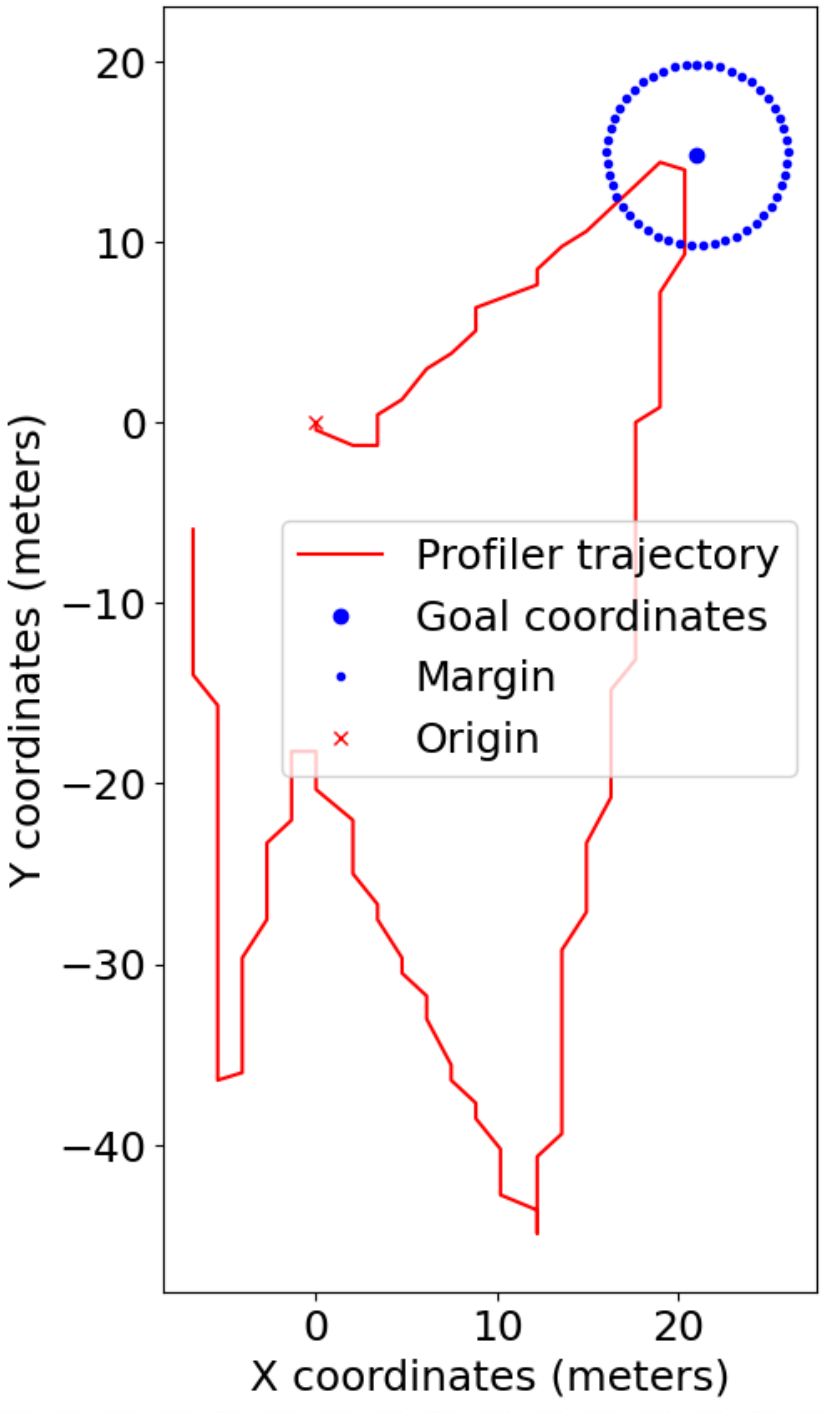


Figure 46: Test 1: Graph of profiler's trajectory when commanded to goal coordinates

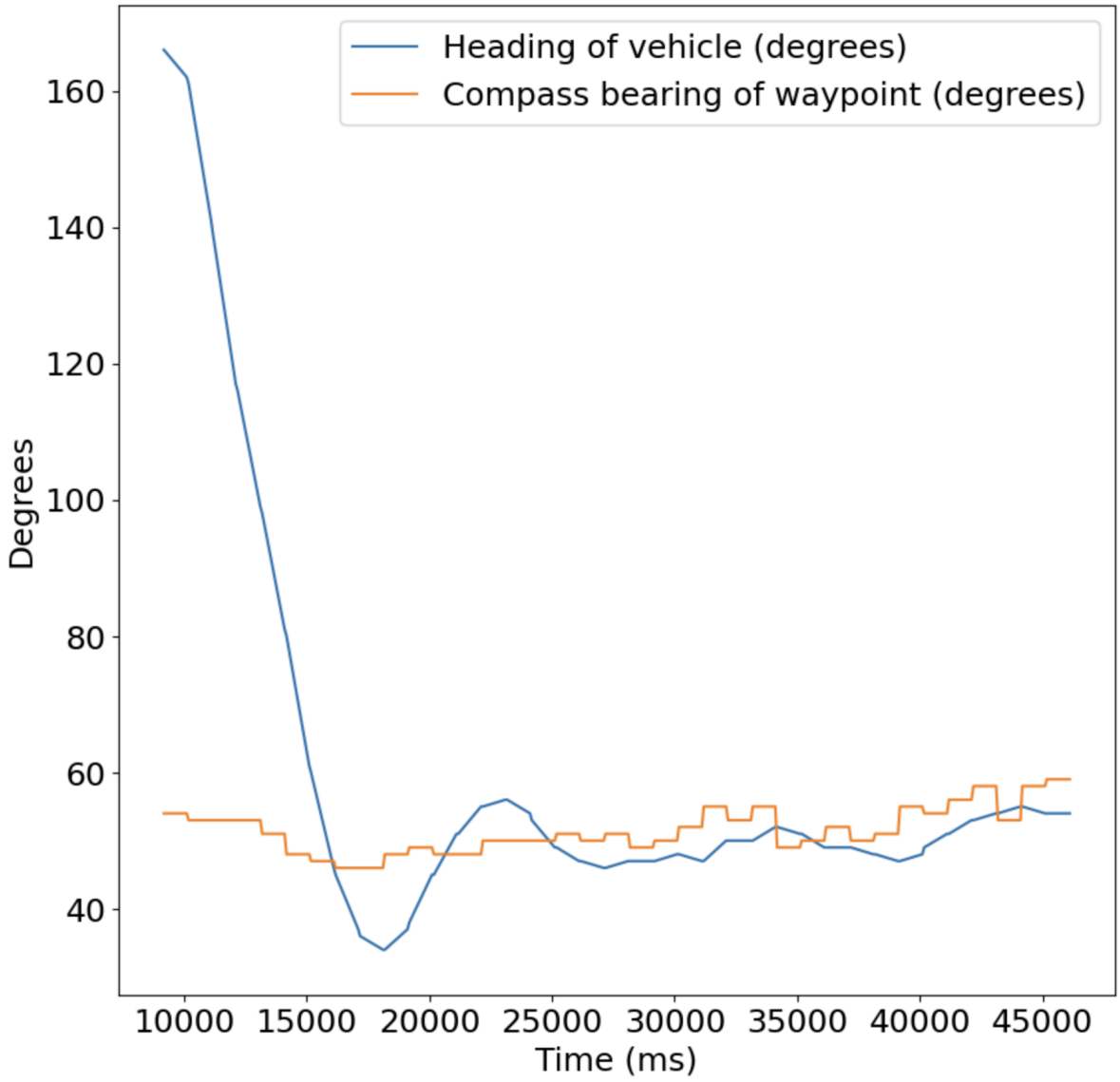


Figure 47: Test 1: Graph of profiler's heading

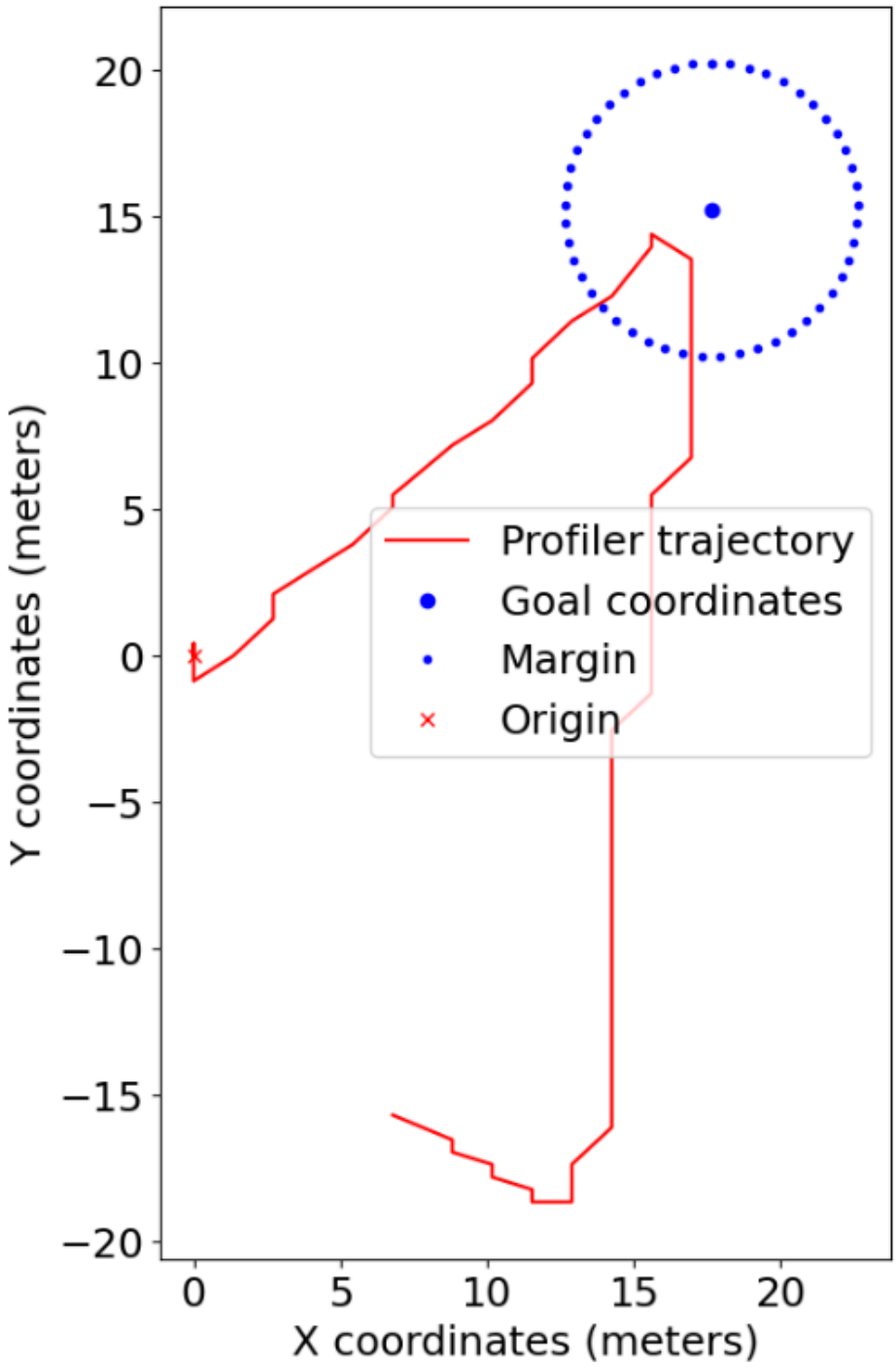


Figure 48: Test 2: Graph of profiler's trajectory when commanded to goal coordinates

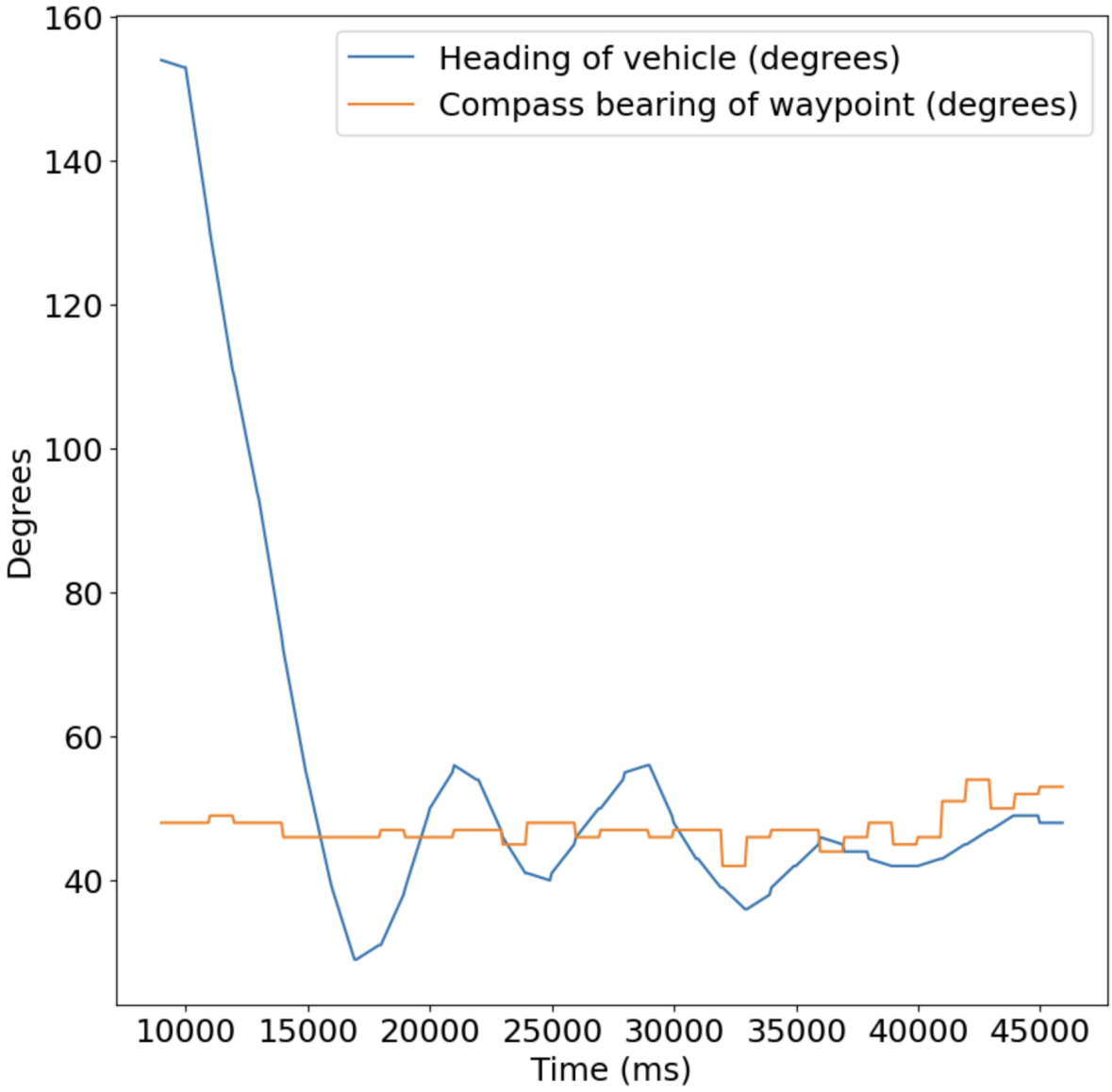


Figure 49: Test 2: Graph of profiler's heading

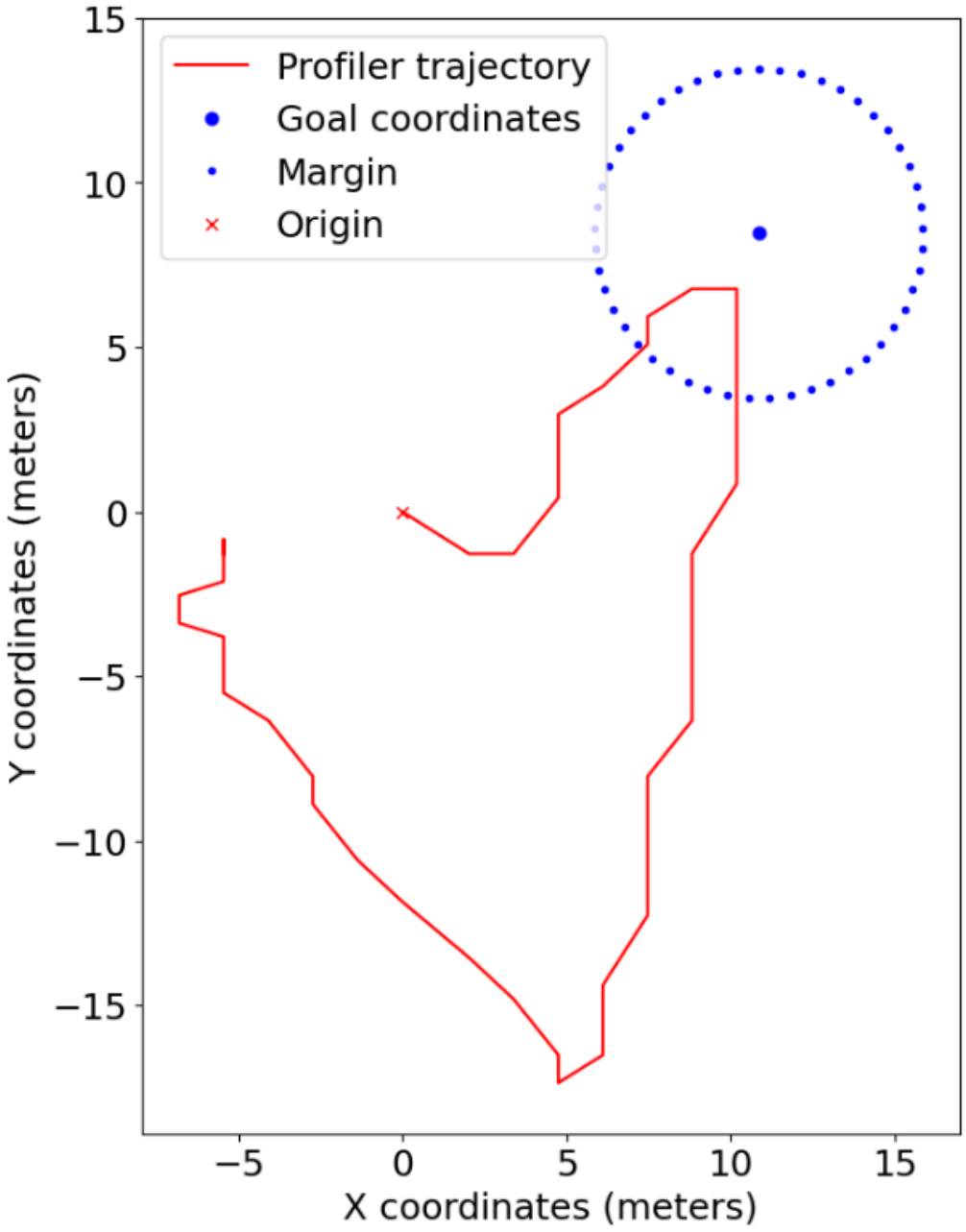


Figure 50: Test 3: Graph of profiler's trajectory when commanded to goal coordinates

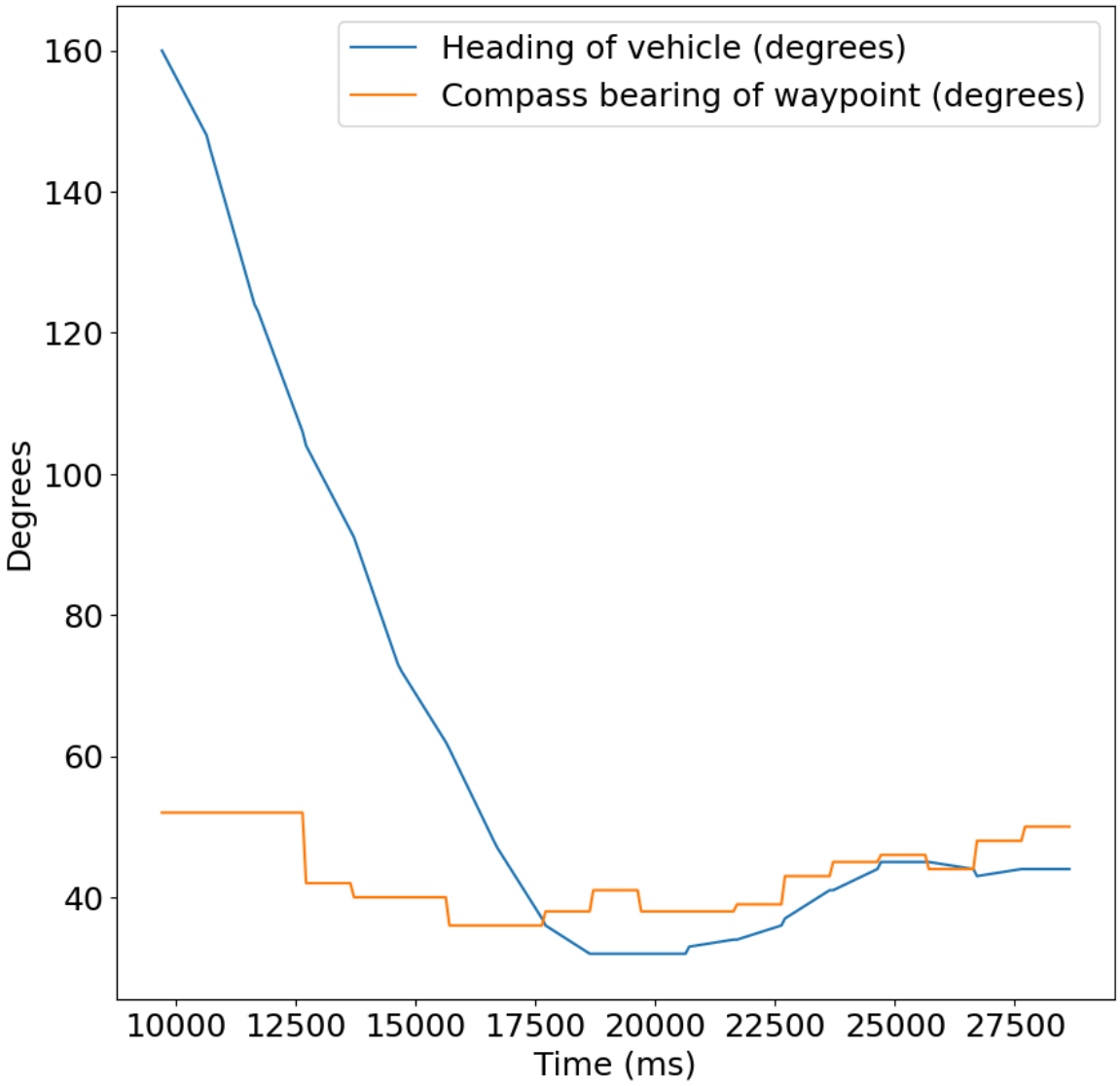


Figure 51: Test 3: Graph of profiler's heading

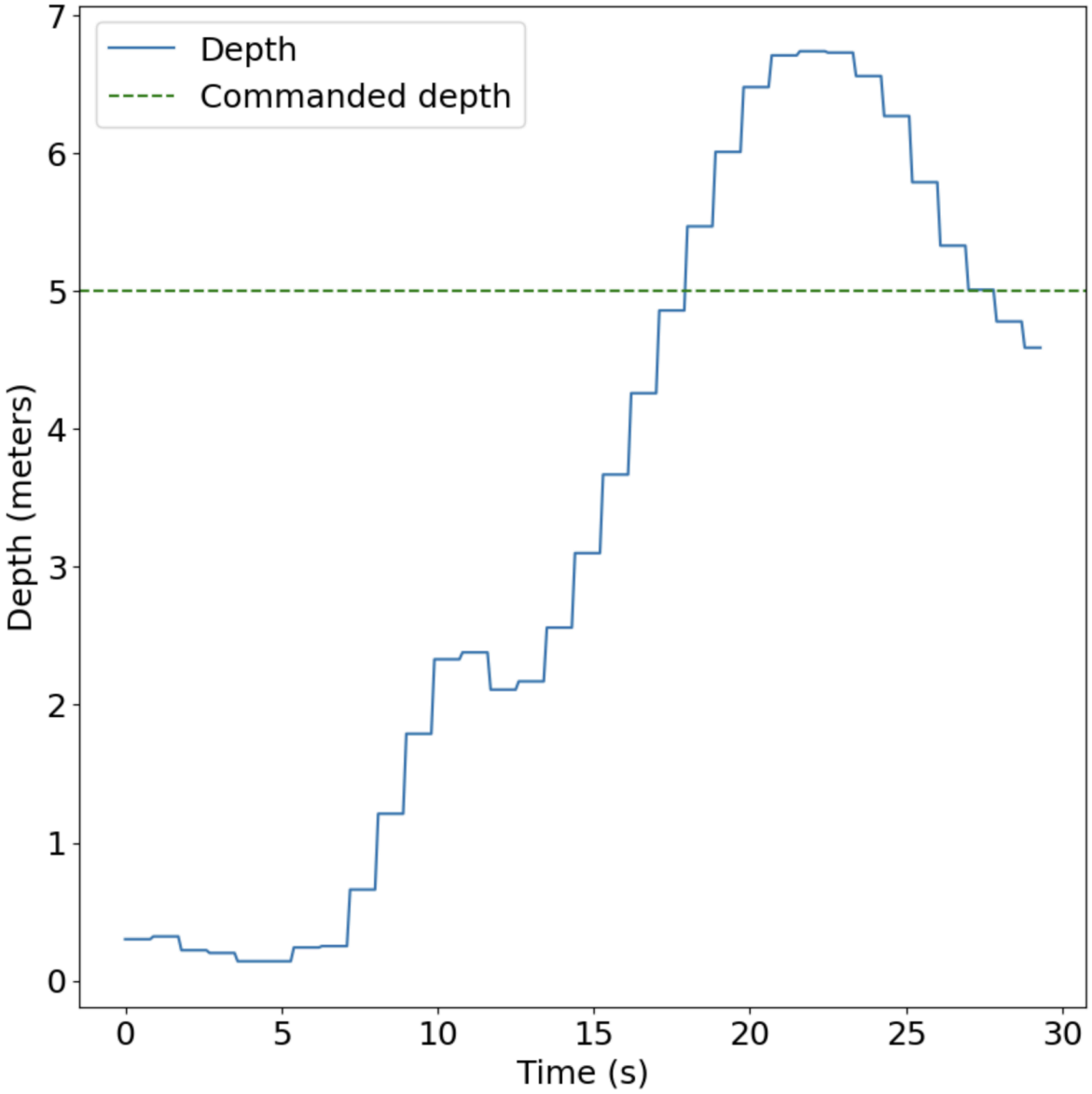


Figure 52: Graph of profiler's position when commanded to maintain depth

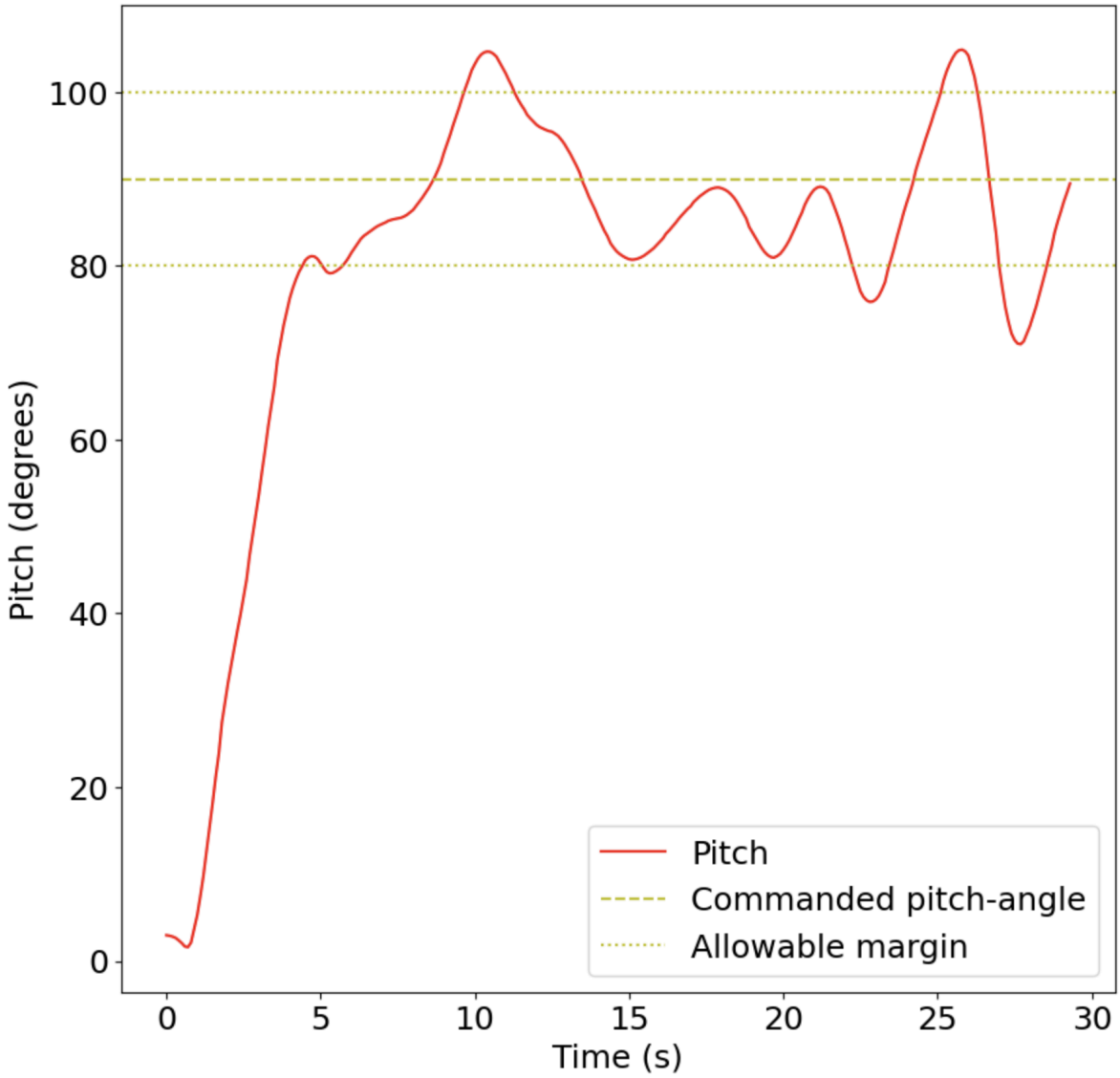


Figure 53: Graph of profiler's pitch-angle when commanded to maintain vertical orientation

11. Constraints & Standards

This chapter covers the constraints that limited what we're able to complete during our project as well as the engineering standards use to guide our design process.

11.1 Constraints

Progress on our project was constrained foremost by the time and resources required to travel to field-testing facilities and locations. Other constraining factors were safety, the availability of 3D printers, and our 3rd party software libraries.

The time and energy required to travel to field-testing facilities bottlenecked our progress. The closest place we could put our craft in the water was the Alviso Marina County Park, a half-hour expedition by SUV. Moreover, the Park itself was limited in that its water was too shallow and murky: preventing us from testing the craft's ability to flip and dive. The only facility where we could test our craft's ability to flip and dive with a measure of veracity was MBARI: a two to three hour round trip that started at 7 AM. Field-testing was crucial to our progress, and it was only by testing our craft in the water did we discover the work still needed to be done. Infrequent field-tests meant delaying the discovery of issues and bugs in our craft—ultimately preventing us from detecting important issues earlier.

Another important constraint was safety, especially in handling our lithium ion batteries. In order to conform to the guidelines set by EHS, we were required to remove the batteries from our craft before and after every deployment and store them in a fireproof box. Moreover, batteries needed to be charged outside of the craft. These safety policies resulted in our craft's watertight seals frequently being broken open to remove the batteries within the craft. In order to maintain the watertight integrity of our craft, we would need to clean and reapply a layer of marine grease to the seals and repressurize the craft onshore at every deployment location—a difficult and sometimes precarious task given the lack of clean and ready working space in the field.

Another significant constraint was the availability of 3D printers. Many mission-critical components were 3D printed. Any delay in printing a part—whether the printer was in-use or out of service—caused a delay in field-testing.

Finally, we were constrained by 3rd party software libraries, particularly the GPS software libraries and the ROS framework. ROS' steep learning curve slowed our progress in implementing the full state machine logic, delaying and ultimately preventing us from field-testing a mature version of the state machine in the field. Furthermore, our inexperience in working with our GPS chip also constrained progress, delaying us from deploying our craft to travel to GPS waypoints. Documentation on our GPS' attached software libraries was often unclear and vague. We instead had to repeatedly experiment with tentative hypotheses about how the GPS hardware and software functioned and made educated guesses on how the API was meant to be used. As a result, we struggled to integrate GPS capabilities into the Navigation & Control subsystem.

11.2 Standards

We conformed to four primary engineering standards in our project, foremost of which being the NASA Systems Engineering handbook [28]. The three other important standards in our project were Agile software development [33], [34] Arduino C++ [35], and ROS2 [36]. We also strove to uphold ethical standards and followed the ACM [26] and IEEE code of ethics [27].

The NASA Systems Engineering handbook prescribes a method in designing complex systems. In accordance with this method, we began designing the waypoint profiler by stating our Mission Statement and assembling a list of Mission Objectives that our craft would need to complete. Our Mission Objectives became quantitative Mission Requirements which formed the basis of the subsystems we identified would be necessary. Only after we had given requirements to the subsystems, we began physically engineering our vehicle. The rationale behind this long period of deliberative engineering rather than physical engineering was to ensure that our design was rooted in the actual reasons we invested ourselves into the project. As we progressed, we updated or reduced those requirements to reflect current circumstances.

The NASA handbook was our big-picture engineering process; Agile software development was our day-to-day routine. We organized our tasks into biweekly “Sprints”, aiming to complete a set of goals by each Sprint’s end. We assigned Owners to each task and we met twice weekly outside of our dedicated meetings with our advisor to update each other on our progress. This style of collaborative workflow allowed us to maintain high momentum in iterating and improving our vehicle, as well as share information on roadblocks to progress and how to effectively overcome them.

Finally, our codebase conformed to the Arduino C++ standards and to the ROS2 code style and language versions guidelines. The Arduino C++ standard was mandatory after we had chosen to use Arduino microcontrollers. On the other hand, we upheld the ROS2 code style and language versions guidelines to create a codebase that was consistent and more easily comprehensible at a glance.

Ethical guidelines also undergirded the project from the beginning. The first canon of the ACM code of ethics: “Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing” [26]. We believe that our project is an exemplification of this standard. Our robotic vehicle serves the mission of the preservation of our ocean. In order to protect our ocean, the study of it is essential—our robotic vehicle takes meaningful steps toward accomplishing this goal.

We also want to ensure our system follows the IEEE standard to not “endanger the public or the environment” [27]. Causing damage to the environment would go against the whole purpose of the profiler in preserving marine environments. We did not design our profiler with any parts or processes that would pollute or litter the ocean.

12. Systems-level Integration and Results

This section discusses the integration of our system, particularly the assembly of all the components as well as the results of our system testing.

12.1 Assembly

We assembled the vehicle one subsystem at a time, roughly following this order:

1. Structural subsystem
2. Power subsystem
3. Navigation & Control subsystem
4. Scientific Sensing subsystem.

The process of piecing together each of the subsystems was difficult. Both the interfaces between the subsystems and the order in which they are assembled needed to be revised over time. For example, initially the Structural subsystem and the Power subsystem interfered with each other physically; as we assembled the full craft for the first time for our first field deployment, we discovered that our batteries from the Power subsystem did not fit in the space for them in the Structural subsystem, leading to multiple needed revisions. Moreover, we discovered that the antenna mast in the Navigation & Control subsystem could not be attached to the weight-rack in the Structural subsystem until all other subsystems were assembled in their places. We recorded the assembly process in a formal procedure checklist, included in Appendix T. Furthermore, the vehicle's microcontrollers are inaccessible once the vehicle is sealed. In order to command the vehicle, we developed a process that would allow us to remotely connect to and start the profiler which is detailed in Appendix Y.

12.2 System Verification

After our subsystems were integrated, we deployed our craft at the Alviso Marina County Park, the Monterey Bay, and in the test tank in MBARI to test the system's performance. We created a pre-deployment procedure, shown in Appendix P, to ensure the safety of the craft. Fully assembled, our vehicle can travel to an itinerary of waypoints. Separately, it can also transform

from horizontal surface navigation-mode to vertical diving-mode and dive down to a commanded depth. Results for these tests are demonstrated and discussed in Chapter 10. Otherwise, we discovered some issues that arose at a system-level.

Firstly, the distribution of mass in the Structural subsystem heavily impacted the performance of the Navigation & Control subsystem. If the craft was ballasted even slightly differently than a previous deployment, the gains for the vehicle's PID control law for flipping would also need to be re-tuned. Because our gains were empirically tuned, we would need to test flipping for every set of candidate PID gains, a very costly and time-consuming process.

Secondly, the craft was very bulky, weighing nearly 80 pounds. Due to the tube's large volume, we needed to add significant ballast to get the craft closer to neutrally buoyant. This complicated transport to field deployments, and made the craft very difficult to lower into the water by hand, especially from a dock or a boat.

Thirdly, the craft struggled to turn against strong currents at Alviso Marina County Park. When trying to return to its home coordinates after completing its itinerary, the craft could not overcome the strong currents pushing it away from the dock. To fix this, we redesigned the thruster mounts for the Parallel Thrusters to have a larger wingspan: giving the craft more torque to turn in the yaw-axis.

Beyond this, we were unable to verify our craft's ability to dive at every waypoint: a capability demanded by our concept of operations. We were constrained by the scarce opportunities to field-test our craft at locations with both water deep enough for our craft to dive and water wide enough for our craft to travel meaningful distances. It is our hope that a future team will be able to complete this test.

13. Professional Development

This section goes over the considerations beyond the sphere of purely engineering to assess or projects ethical and social implications. In addition we discuss how the project impacted our team as aspiring engineers, and its potential viability as a product based on its usability and manufacturability.

13.1 Ethical Reflections

In alignment with the ASME Code of Ethics, it is imperative to prioritize the safety, health, and welfare of the public and the environment in the performance of professional duties [29].

Developing an underwater robot to assist marine scientists in collecting water data and monitoring ocean health is undoubtedly a promising endeavor with potentially significant benefits. However, it also raises several ethical considerations that merit careful reflection.

- Environmental Impact: While the intention behind the project is to monitor and protect the ocean, it's essential to ensure that the deployment and operation of the underwater robot do not inadvertently harm marine life or disrupt delicate ecosystems. Measures should be taken to minimize the robot's environmental footprint, such as selecting non-invasive monitoring techniques and avoiding sensitive habitats.
- Equity and Access: Access to advanced technology, such as underwater robots, can sometimes be limited to well-funded research institutions or organizations. There's a need to ensure that the benefits of this technology are equitably distributed, especially considering that many communities reliant on marine resources may lack access to such tools. Therefore, the majority of the materials we use are 3D printed, which helps to reduce the cost of production. Furthermore, with modularity, researchers can simply purchase the scientific equipment they need, thereby reducing unnecessary costs from extra capabilities. With lower costs and customization, we can make the product more accessible.

- Long-Term Impacts: As with any technological intervention, it's essential to consider the potential long-term impacts of deploying underwater robots in the ocean. This includes not only the direct effects on marine ecosystems but also broader societal implications. Anticipating and mitigating any unintended consequences, such as changes in human behavior or policy responses, is crucial for responsible innovation.
- Ethical Use of Data: The data collected by the underwater robot could have far-reaching implications for marine conservation and management decisions. It's important to consider who has the authority to interpret and act upon this data, as well as the potential for biases or conflicts of interest. Engaging diverse stakeholders, including local communities and indigenous groups, in the decision-making process can help ensure that their perspectives are represented.

Overall, while the development of an underwater robot for ocean monitoring holds great promise for advancing scientific knowledge and conservation efforts, it's essential to approach this work with a strong ethical framework. By prioritizing environmental stewardship, data privacy, equity, and transparency, researchers can maximize the positive impact of their efforts while minimizing potential risks and harms.

13.2 Social

The primary social impact of our product is improving the work of marine scientists and researchers. This is the main consumer base for our project and the primary impact will be increasing the efficiency and safety of their work. Our project will allow them to collect data on demand, year round which will speed up their ability to do research. In a broader context, speeding up marine research may improve our understanding of ocean environments. This may give us greater insights into their importance and preservation, which will affect all coastal communities that benefit from marine environments. Whether it's fishing or tourism, a healthy marine environment is the centerpiece of social activities for coastal communities

13.3 Political

The goal of our project is the study and preservation of marine environments. Caring for the environment as a concept is pretty uncontroversial. But discussions of Climate Change have become politicized in modern times. The ocean is also highly relevant when it comes to research on climate change. This project is not intended to push any political agenda, rather it is meant to enable marine scientists to further their research, which may help inform political decisions one way or the other.

13.4 Economic

During the design development phase of our project we had to consider many cost trade offs. When it comes to our thruster orientation we were considering a design that only used one thruster pair instead of two. This would save us a decent amount of money in buying less thrusters, but we decided it would come at the greater cost of a less functional product. Using only one thruster pair is also a more complicated design, so we may have spent more money on design interactions trying to get it to work.

Speaking of design iterations we realized that we were spending a lot of money iterating on 3d printed parts. To mitigate this we tried to get feedback from more team members on the potential part design to try to foresee potential problems. This helps reduce the number of iterations by eliminating potential problems before we see them firsthand.

Another cost trade off was the size of the tube. A smaller tube would mean most other parts take up less materials and are cheaper. But a larger tube would give us more space to fit more components inside the tube. In the end we chose a larger tube since we needed more space to fit enough batteries to reach the required range of the profiler. This ended up being a good decision because the current design utilizes most of the space in the tube, and fitting it all in a smaller tube would require an immense amount of work redesigning.

13.5 Lifelong Learning

This project did a lot to prepare us for continuing our learning after graduation. Since it was an interdisciplinary project, it required us teaching each other what we know about our own areas of expertise. This gave everyone on the team a more comprehensive understanding of how the profiler works. Each team member got involved with wiring, soldering, assembling, and testing even if it was new to them. By the end of the project we had all gained more confidence with the new skills and tools we had learned over the course of the project. This project provided great practice for our adaptability as engineers to pick up new skills and adjust our ways of thinking.

13.6 Compassion

Our project demonstrates compassion for the planet, and all the life inside of lakes and oceans around the world. It also shows compassion for coastal communities who rely on the wellbeing of local marine environments for economic and recreational benefits. We understand these environments are highly important to our global well being, and are currently under serious risk of decay. In order to help someone who is struggling you must first empathize and understand them. That is exactly what our project aims to do with marine environments. By giving marine researchers greater ability to study these environments, we can improve our understanding and do our best to relieve its suffering.

13.7 Safety Considerations

The main safety considerations for our AMV is solely due to the autonomous nature of the vehicle. Before diving into the safety considerations, we must first clarify what we mean by autonomous navigation. The end goal of our project is to deploy our craft on shore with waypoints already stored on the onboard computing subsystem. The craft will then sequentially collect water column data at all waypoints and then return back to shore. During each deployment, the craft must not only perform its primary function of water column data collection, but must also be capable of safely traversing the water while being mindful of other

craft. This means that in order to safely operate this vehicle, we need to develop redundant and reliable systems that can account for the majority of situations that our craft will encounter.

While testing the current iteration of our craft, we have relied on two methods. The first method involves deploying our craft with a rope attached to the T-slot rack. Our second method involves incorporating a timer into our code. This second method ensures that the robot will naturally turn itself off. However, in the future if the craft is to be deployed fully autonomously without any oversight, a collision detection algorithm will be a priority alongside a watchdog timer. Moreover, an underwater modem could be an additional subsystem implemented to allow simple communication between the craft and the operators on the surface.

13.8 Environmental & Sustainability

Our underwater robot project stands as a testament to our commitment to environmental stewardship and sustainability in marine research. By revolutionizing the methods of data collection and monitoring in the ocean, we aim to mitigate the impact of human activities on marine ecosystems while advancing scientific understanding. This chapter delves into the environmental considerations and sustainability measures integrated into our project.

- Reducing Carbon Emissions and Disturbance to Marine Life: One of the primary objectives of our project is to minimize the carbon footprint associated with marine research activities. Traditionally, researchers rely on chartering boats to deploy divers for data collection, leading to significant emissions. By providing an alternative solution through our underwater robot, we eliminate the need for frequent boat charters. This not only reduces carbon emissions but also minimizes disturbances to marine life, as the operation of our robot is far less intrusive compared to traditional methods.
- Cost Reduction and Increased Accessibility: Our innovative approach not only benefits the environment but also promotes financial sustainability in marine research. By eliminating the recurrent expense of chartering boats, our project significantly reduces the overall cost of data collection. This cost-effectiveness enhances accessibility to marine research, allowing more scientists and organizations to participate in crucial environmental monitoring efforts.

- Battery Safety and Pollution Prevention: Acknowledging the environmental risks posed by lithium-ion batteries, we have implemented stringent measures to mitigate any potential pollution. Our robot is equipped with a lithium-ion battery designed with multiple layers of protection to prevent leakage into the ocean. Furthermore, the slightly positive buoyancy of our craft ensures that in the event of an emergency shutdown, it resurfaces promptly, facilitating easy retrieval and preventing prolonged exposure of the battery to marine environments.
- Proactive Health Monitoring and Emergency Response: Central to our sustainability ethos is the proactive monitoring of the robot's health and performance. Through an integrated health monitoring system, we continuously assess the state of the robot and detect any anomalies or potential failures. In the event of a malfunction or critical issue, the system initiates an immediate shutdown, ensuring minimal impact on the surrounding environment. Additionally, the ability to trace back the robot facilitates swift remedial action, further reducing the risk of environmental harm.

In conclusion, our underwater robot project exemplifies a holistic approach to environmental and sustainability considerations in marine research. By prioritizing the reduction of carbon emissions, minimizing disturbances to marine life, and incorporating sustainable design principles, we aim to pioneer a new era of eco-friendly data collection and monitoring in the ocean. Through continuous innovation and adherence to environmental best practices, we strive to contribute positively to the conservation and preservation of our marine ecosystems for future generations.

13.9 Usability

The general handling of the craft is quite poor. This is mainly due to the length of the tube which makes the maneuvering difficult during transportation and deployment. Moreover, the weight of the craft (~80 lbs) makes carrying the craft and deployment taxing.

Furthermore, in its current state, the user has to reupload code to declare new waypoints, update PID gains, etc. This requires the user to edit the code directly and then upload the new

code over WiFi. The addition of a graphical user interface (GUI) could solve this problem and make the experience more user friendly.

Moreover, in order to deploy the craft, a strict checklist needs to be followed to ensure that the robot is in proper operating condition. Our checklist is shown in Appendix P. A failure to meet any of the checks means that if the craft were to be deployed, the craft will be at high risk of malfunction. One this same point, general troubleshooting and maintenance can be difficult due to a variety of potential sources.

13.10 Manufacturability

Our craft consists of components that can be grouped into two categories, those manufactured in-house and those purchased off-the-shelf. Under the first category, the majority of the components are 3D printed. This includes components mentioned earlier such as thruster mounts, antenna mast, ballast weight rack mounts, nose cone, battery tray, etc. The components on the exterior of the craft are printed out of PETG filament while those on the interior are printed out of PLA+ filament. Other components in this category include laser cut acrylic which form the electronics tray and SLS components such as the handle of the electronics tray. Under the second category, there are several subcategories such as electrical, propulsion, and pressure vessel design. The electrical is the largest subcategory as it consists of all the components used in the computing subsystem and navigation and control subsystems. This includes the Pi 4, Arduino Mega, buck converters, Wifi router, GPS and LoRa chips. The propulsion system mainly includes the four BlueRobotics T200 thrusters to control the craft. The pressure vessel subcategory is the watertight housing of the craft includes components such as the 6” acrylic tube, BlueRobotics watertight enclosure end caps, BlueRobotics penetrators and BlueRobotics cable splice kit. Appendix U has a Bill of Materials listing all of these components.

14. Conclusions

14.1 Summary

Our mission was to develop an autonomous marine vehicle that can be deployed from the shore, travel across the water, and dive into the water to collect water-column data using modular scientific equipment. But our vision was only partially realized:

Firstly, we field-tested our vehicle's ability to navigate to GPS waypoints at Monterey Bay. It was successful in steering itself and reaching its target waypoint, but had issues returning to its home coordinates. Moreover, the vehicle was unable to utilize its radio communications hardware to receive new waypoints to add to its itinerary or transmit its location and measured water-column data back to the user.

Secondly, we field-tested the profiler's ability to flip into vertical diving mode and hold a commanded depth in the water in the test tank at MBARI. It successfully flipped into a vertical orientation and achieved a controlled depth with an error of 2 m.

Thirdly, we designed and prototyped a novel mechanism for capturing water samples that uses half as many servos for twice the amount of sampled water. The design's mechanism properly actuates but leaks water. We tested the mechanism in the in-lab facilities on campus, but were unable to put the sampler on the vehicle for a field deployment due to time constraints.

Most of the critical subsystems of our vehicle were successfully brought into the field and tested, but its performance as a whole standalone system is, due to several constraints discussed earlier, untested. This is primarily due to how a mature version of the state machine software was unable to be completed due to the steep learning curve imposed by the ROS framework. A mature version of the state machine would, as described in the Software Architecture chapter, command the individual subsystems to act as an ensemble to complete the mission as described in the concept of operations. We hope that this vision will be borne out by a future team.

14.2 Future Work

One significant challenge our team faced was enhancing the vehicle's ability to transition into diving mode. Future iterations should focus on utilizing a mechanism to adjust the vehicle's center of mass. This enhancement would significantly increase the reliability of the diving capability, enabling the deployment of fleets of these vehicles for multi-robot scalar field adaptive navigation (SFAN). The ocean presents various parameters such as turbidity and dissolved oxygen in gradient form across its expanse. Locating scientifically significant features within these measurements can be slow and challenging when relying on a single robot to exhaustively search the ocean. In contrast, deploying multiple robots in a coordinated cluster can efficiently locate and navigate to these features. This approach saves valuable time, conserves energy, and optimizes resource utilization in marine exploration missions.

References

[1] OECD, The Ocean Economy in 2030. 2016, p. 252.

[2] M. J. Spalding, “The New Blue Economy: the Future of Sustainability,” *Journal of Ocean and Coastal Economics*, vol. 2, no. 2, doi: 10.15351/2373-8456.1052.

[3] K. J. Kroeker, R. L. Kordas, R. N. Crim, and G. G. Singh, “Meta‐analysis reveals negative yet variable effects of ocean acidification on marine organisms,” *Ecology Letters*, vol. 13, no. 11, pp. 1419–1434, Aug. 2010, doi: 10.1111/j.1461-0248.2010.01518.x.

[4] Q. He and B. R. Silliman, “Climate change, human impacts, and coastal ecosystems in the anthropocene,” *CB/Current Biology*, vol. 29, no. 19, pp. R1021–R1035, Oct. 2019, doi: 10.1016/j.cub.2019.08.042.

[5] D. Li, P. Wang, and L. Du, “Path Planning Technologies for Autonomous Underwater Vehicles-A Review,” *IEEE Access*, vol. 7, pp. 9745–9768, Jan. 2019, doi: 10.1109/access.2018.2888617.

[6] S. Jung, H. Cho, D. Kim, K. Kim, J.-I. Han, and H. Myung, “Development of algal bloom removal system using unmanned aerial vehicle and surface vehicle,” *IEEE Access*, vol. 5, pp. 22166–22176, Jan. 2017, doi: 10.1109/access.2017.2764328.

[7] L. -J. Mu et al., "Underwater topography measurement and observation in Southwest Taiwan using unmanned underwater vehicles," *OCEANS 2014 - TAIPEI*, Taipei, Taiwan, 2014, pp. 1-6, doi: 10.1109/OCEANS-TAIPEI.2014.6964394.

[8] J. Ke et al., "Low-Cost, Minimal-Power Marine Vertical Profilers for Lakes and Coastal Shallows," *OCEANS 2023 - MTS/IEEE U.S. Gulf Coast,* Biloxi, MS, USA, 2023, pp. 1-5, doi: 10.23919/OCEANS52994.2023.10337125.

[9] S. B. Williams, O. Pizarro, D. M. Steinberg, A. Friedman, and M. Bryson, ‘Reflections on a decade of autonomous underwater vehicles operations for marine survey at the Australian Centre for Field Robotics’, *Annual Reviews in Control*, vol. 42, pp. 158–165, 2016.

[10] S. L. Sequeira, E. S. Manish, Rakshith, P. Umesh and K. V. Gangadharan, "Development of Portable Tethered Vertical Profiler for Underwater Monitoring," *2023 International Conference for Advancement in Technology (ICONAT)*, Goa, India, 2023, pp. 1-6, doi: 10.1109/ICONAT57137.2023.10080657.

[11] F. T. Thwaites, R. Krishfield, M. -L. Timmermans, J. M. Toole and A. J. Williams, "Flux measurements from an Ice-tethered profiler: First look," *OCEANS 2011 IEEE - Spain*, Santander, Spain, 2011, pp. 1-6, doi: 10.1109/Oceans-Spain.2011.6003621.

[12] T. McGinnis, N. Michel-Hart, M. Mathewson and T. Shanahan, ”Deep profiler for the ocean observatories initiative Regional Scale Nodes: Rechargable, adaptive, ROV serviceable,” *2013 OCEANS - San Diego*, San Diego, CA, USA, 2013, pp. 1-4, doi:10.23919/OCEANS.2013.6741191.

[13] J. Singleton, B. de Young and R. Bachmayer, ”Design and Development of a Novel Autonomous Moored Profiler,” *OCEANS 2019 MTS/IEEE SEATTLE*, Seattle, WA, USA, 2019, pp. 1-10, doi: 10.23919/OCEANS40490.2019.8962852.

[14] T. J. Osse, C. Meinig, S. Stalin and H. Milburn, "The PRAWLER, a vertical profiler powered by wave energy," *OCEANS 2015 - MTS/IEEE Washington*, Washington, DC, USA, 2015, pp. 1-8, doi: 10.23919/OCEANS.2015.7404354.

[15] O. Loisa, J. -H. Körber, J. Laaksonlaita and J. Kääriä, "High-resolution monitoring of stratification patterns in the Archipelago Sea, Northern Baltic Sea, using an autonomous moored vertical profiling system," *OCEANS 2017 - Anchorage*, Anchorage, AK, USA, 2017, pp. 1-6.

[16] J. A. Breier et al., “Revealing ocean-scale biochemical structure with a deep-diving vertical profiling autonomous vehicle,” *Science Robotics*, vol. 5, no. 48, Nov. 2020, doi: 10.1126/scirobotics.abc7104.

[17] S. C. Riser et al., “Fifteen years of ocean observations with the global Argo array,” *Nature Climate Change*, vol. 6, no. 2, pp. 145–153, Jan. 2016, doi: 10.1038/nclimate2872.

[18] Dartmouth Ocean Technologies Inc. D Profiler Autonomous Vertical Profiler, 2020. Accessed: Apr. 7, 2023. [Online]. Available: https://dartmouthocean.com/sites/default/files/2020-01/BROCHURE-DOT-DProfiler-General-2020-01A-FINAL RGB 2020-01-03.pdf

[19] W. D. Wilson, ”Water quality profiling with a WETLabs Autonomous Moored Profiler (AMP),” *OCEANS’11 MTS/IEEE KONA*, Waikoloa, HI, USA, 2011, pp. 1-10, doi:10.23919/OCEANS.2011.6107245.

[20] D. Stefanides et al., “Nautilus ROV Robot Manipulator,” *Scholar Commons*. https://scholarcommons.scu.edu/idp_senior/92.

[21] “PETG Characteristics,” Dielectric Manufacturing, Jul. 25, 2022. https://dielectricmfg.com/resources/knowledge-base/petg/ (accessed Jun. 06, 2024).

[22] G. A. Ribeiro, A. Pinar, E. Wilkening, S. Ziaeefard and N. Mahmoudian, "A multi-level motion controller for low-cost Underwater Gliders," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, 2015, pp. 1131-1136, doi: 10.1109/ICRA.2015.7139333.

[23] E. Petritoli, F. Leccese and M. Cagnetti, "A High Accuracy Buoyancy System Control for an Underwater Glider," *2018 IEEE International Workshop on Metrology for the Sea*; Learning to Measure Sea Health Parameters (MetroSea), Bari, Italy, 2018, pp. 257-261, doi: 10.1109/MetroSea.2018.8657831.

[24] S. S. Nerkar, P. S. Londhe, and B. M. Patre, “Design of super twisting disturbance observer based control for autonomous underwater vehicle,” *International Journal of Dynamics and Control*, vol. 10, no. 1, pp. 306–322, May 2021, doi: 10.1007/s40435-021-00797-1.

[25] B. H. I. Sutton, “H I Sutton - Covert Shores.” http://www.hisutton.com/SwarmDiver.html (accessed Jun. 12, 2024).

[26] ACM Code 2018 Task Force, “ACM Code of Ethics and Professional Conduct,” Association for Computing Machinery, Jun. 2018. Accessed: May 30, 2024. [Online]. Available: https://www.acm.org/code-of-ethics.

[27] IEEE Board of Directors, “IEEE Code of Ethics,” Institution of Electronics Engineers, Jun, 2020. Accessed May 30, 2024. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html.

[28] N. S. Administration, NASA Systems Engineering Handbook (NASA SP-2016-6105 Rev2). CreateSpace Independent Publishing Platform, 2017.

[29] American Society of Mechanical Engineers, “CODE OF ETHICS OF ENGINEERS,” Oct. 2021. Accessed: May 30, 2024. [Online]. Available: https://www.asme.org/getmedia/3e165b2b-f7e7-4106-a772-5f0586d2268e/p-15-7-ethics.pdf.

[30] M. Woolsey, C. A. Kitts and F. Amend, "A Diving Autonomous Surface Vehicle for Multi-Robot Research and Development," *OCEANS 2019 MTS/IEEE SEATTLE*, Seattle, WA, USA, 2019, pp. 1-7, doi: 10.23919/OCEANS40490.2019.8962547.

[31] D. L. Fougere, A. J. Fougere, J. M. Toole and J. K. O’Brien, "Continued Development and Evaluation of the D-2 Inc. Hybrid CTD Sensor," *OCEANS 2021: San Diego – Porto*, San Diego, CA, USA, 2021, pp. 1-7, doi: 10.23919/OCEANS44145.2021.9706077.

[32] M. Merchel, W. Walczowski, D. Rak, and P. Wieczorek, “The use of Argo floats as virtual moorings for monitoring the South Baltic Sea,” *Oceanologia*, Feb. 2024, doi: 10.1016/j.oceano.2024.01.002.

[33] “Manifesto for Agile software development.” https://agilemanifesto.org/ (accessed Jun. 12, 2024).

[34] Agile Alliance, “What is Agile? | Agile 101 | Agile Alliance,” *Agile Alliance* |, Apr. 26, 2024. https://www.agilealliance.org/agile101/ (accessed Jun. 12, 2024).

[35] “Arduino / C++,” *Arduino Docs*, Jan. 24, 2024. https://docs.arduino.cc/arduino-cloud/guides/arduino-c/ (accessed Jun. 09, 2024).

[36] “Code style and language versions — ROS 2 Documentation: Humble documentation.” https://docs.ros.org/en/humble/The-ROS2-Project/Contributing/Code-Style-Language-Versions.html (accessed Jun. 12, 2024).

Appendices

Appendix A: Master Compute Bridge Class Header File

```
/* This class controls the communication between the arduino and PI.
 It contains all the necessary drivers for the different sensors.
*/
#ifndef MasterComputeBridge_h
#define MasterComputeBridge_h
#include "Arduino.h"
#include "Constants.h"
#include "_GPSDriver.h"
#include "IMUDriver.h"
#include "PingDriver.h"
#include "ThrusterDriver.h"
#include "RadioDriver.h"
class MasterComputeBridge{
 private:
   ThrusterDriver thruster1;
   ThrusterDriver thruster2;
   ThrusterDriver thruster3;
   ThrusterDriver thruster4;
   IMUDriver IMU;
   //PingDriver ping;
   // RadioDriver Lora;
   String functionReturn;
   _GPSDriver GPS;
 public:
   MasterComputeBridge();
   void IMUSetup();
   void thrusterSetup();
   void giveCommand(String command);
   void spinGPS();
   String returnCommand();
};
#endif
```

Appendix B: Waypoint Action Code Snippet

```
        def waypoint_callback(self, goal_handle):
            self.get_logger().info('traveling to waypoint...')
            coords = goal_handle.request.waypoint_coords
            self.wp_lat_ = coords.latitude
            self.wp_lon_ = coords.longitude

            geo = self.get_gnss()
            self.lat_ = geo.latitude
            self.lon_ = geo.longitude
            self.heading_ = geo.altitude

            while(not self.navigator_.atWaypoint(self.lat_, self.lon_, self.wp_lat_, self.wp_lon_)): #while not at waypoint
                pwm = self.navigator_.waypointToPwm(self.lat_, self.lon_,
                                                    self.wp_lat_, self.wp_lon_,
                                                    self.heading_)
                from math import pi
                self.get_logger().info("Bearing {}".format(str(self.navigator_.bearing_)))
                self.get_logger().info("Heading {}".format(str((self.heading_ * 180/pi + 360 + 12.8) % 360)))
                self.get_logger().info("PWM FL{} FR{} DL{} DR{}".format(pwm[0], pwm[1], pwm[2], pwm[3]))
                self.get_logger().info("Distance to waypoint: {}".format(self.navigator_.getDistanceToWaypoint(self.lat_,
self.lon_, self.wp_lat_, self.wp_lon_)))
                self.get_logger().info("Velocity Commands: {}".format(self.navigator_.waypointToVelocity(self.lat_, self.lon_,
self.wp_lat_, self.wp_lon_)))
                self.get_logger().info("Heading Err: {}".format(self.navigator_.getHeadingError(radians(self.lat_),
radians(self.lon_), radians(self.wp_lat_), radians(self.wp_lon_), self.heading_)))

                self.send_pwm(pwm)
                feedback_msg = Waypoint.Feedback()
                feedback_msg.distance_to_waypoint = self.navigator_.getDistanceToWaypoint(self.lat_, self.lon_, self.wp_lat_,
self.wp_lon_)
                goal_handle.publish_feedback(feedback_msg)
                geo = self.get_gnss()
                self.lat_ = geo.latitude
                self.lon_ = geo.longitude
                self.heading_ = geo.altitude

            goal_handle.succeed()
            result = Waypoint.Result()
            result.arrived_at_waypoint = True
            return result
```

Appendix C: Profile Action Code Snippet

```
        def profile_callback(self, goal_handle):
            self.get_logger().info('profiling water column...')
            desiredDepth = goal_handle.request.desired_depth
            self.htovFlip() #flip orientation
            self.depth = self.get_depth()
            while(self.depth>=desiredDepth): #or close to sea floor
                #update depth
                pwm = self.navigator_.descendToPwm(self.depth, desiredDepth, self.heading_)
                self.send_pwm(pwm)
                self.depth = self.get_depth()

            while(self.depth<1.0):
                #get and log sensor data
                pwm = self.navigator_.ascendToPwm(self.depth, desiredDepth, self.heading_)
                self.send_pwm(pwm)
                self.depth = self.get_depth()

            self.vtohFlip() #flip orientation back
            goal_handle.succeed()
            result = Profile.Result()
            result.ending_depth = self.depth
            return result
```

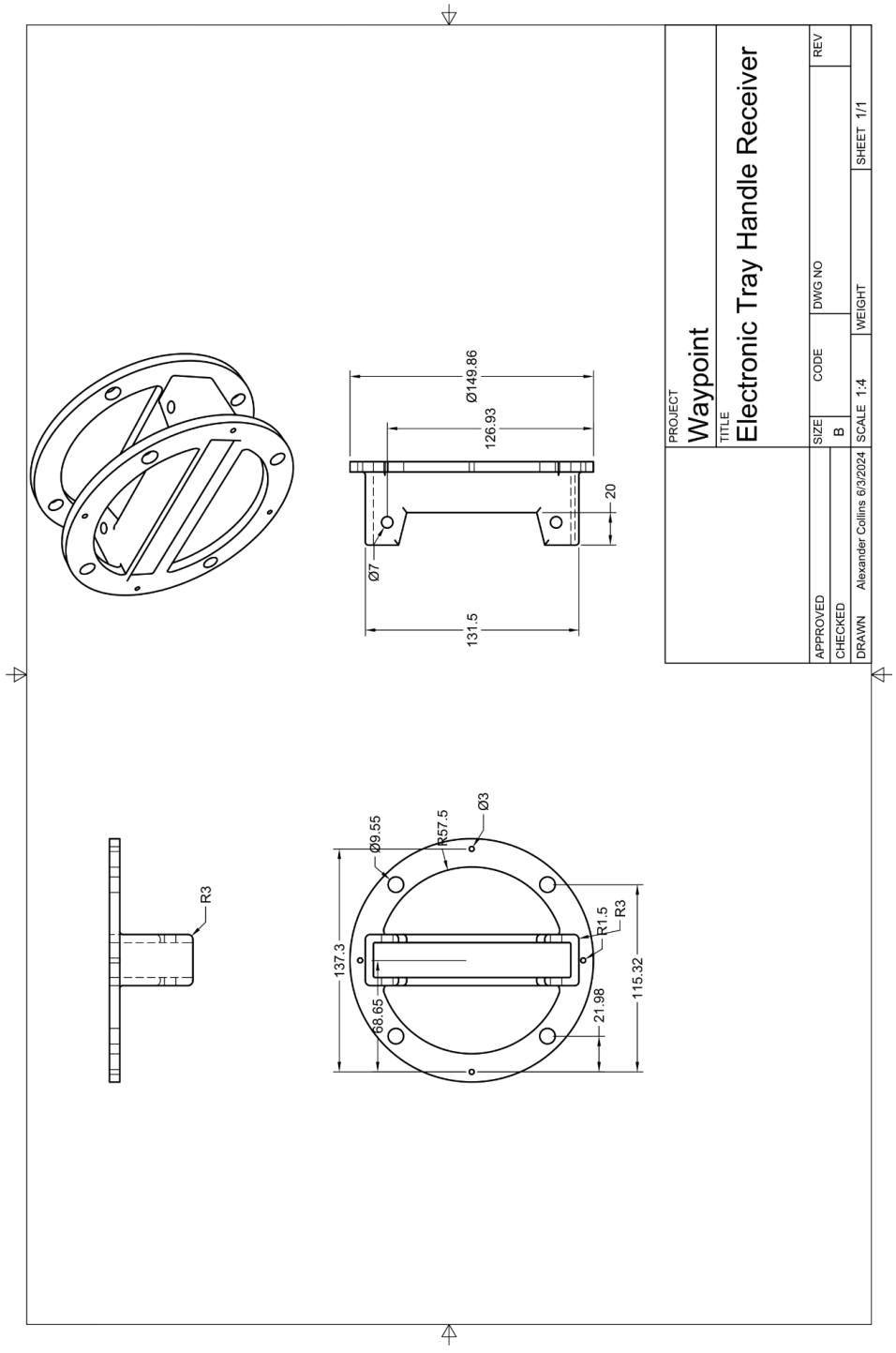
Appendix D: State Machine Code Snippet

```
from geographic_msgs.msg import GeoPoint

class StateMachine:
  def __init__(self):
    #override max depth determined by hardware constraints
    self.MAXDEPTH = 20.0
    #list of geopoints that are lat, long, depth
    self.waypoints_ = []
    #This variable tracks the state as a string default is standby
    self.state_ = "setup" #valid value: setup, idle, waypoint, profile, return, estop

    self.home_ = GeoPoint(latitude=0.0, longitude=0.0, altitude=0.0)#home coords to return to

  def pushBackWP(self, geopoint):
    if(geopoint.altitude>self.MAXDEPTH):
      geopoint.altitude = self.MAXDEPTH
    self.waypoints_.append(geopoint)

  def pushWP(self, geopoint):
    if(geopoint.altitude>self.MAXDEPTH):
      geopoint.altitude = self.MAXDEPTH
    self.waypoints_.insert(0, geopoint)

  def popWP(self):
    return self.waypoints_.pop(0)

  def currentWP(self):
    return self.waypoints_[0]

  def setHome(self, geopoint):
    self.home_ = geopoint

  #after pivotal ROS callback such as waypoint or profile reasses state
  #takes in as input whether the objective of the current state was achieved
  def assessState(self):
    if(self.state_=="setup"):
      if(len(self.waypoints_)>0):
        self.state_ = "waypoint"
        return True
      else:
        self.state_ = "idle"
        return True
    elif(self.state_=="profile"):
      if(len(self.waypoints_)>0):
        self.state_ = "waypoint"
        return True
      else:
        self.state_ = "idle"
        return True
    elif(self.state_=="waypoint"):
      self.state_ = "profile"
      return True
    elif(self.state_=="idle" and len(self.waypoints_)>0):
      self.state_ = "waypoint"
      return True
    return False

  def state(self):
    return self.state_
```

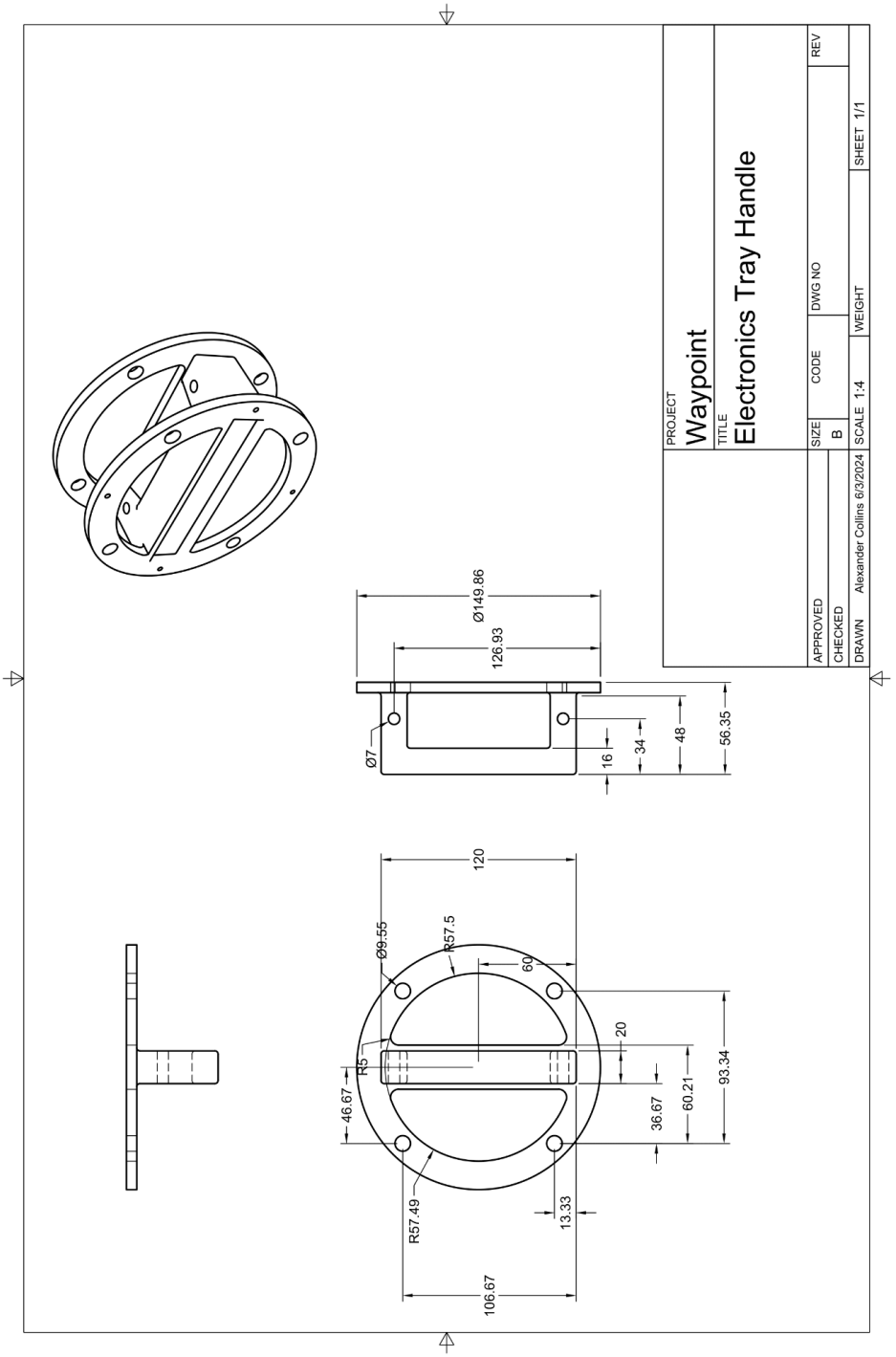
Appendix E: Hardware Bridge Node Code Snippet

```
class HBNode(Node):
  def __init__(self):
    super().__init__("hardware_bridge_node")
    self.create_service(GetDepth, 'get_depth', self.RequestDepth)
    self.create_service(GetGnss, 'get_gnss', self.RequestGnss)
    self.create_service(GetOrientation, 'get_orientation', self.RequestOrientation)
    self.create_service(SendKill, 'send_kill', self.SendKill)
    self.create_service(SendPwm, 'send_pwm', self.SendPwm)
    self.create_service(GetMinionStatus, 'minion_status', self.GetMinionStatus)

    self.bridge_ = HardwareBridge("ACM0", 115200, 5, self.get_logger())

    self.bridge_.WaitForInit()
  def GetMinionStatus(self, request, response):
    from std_msgs.msg import Int16
    result = self.bridge_.AskForStatus(ros_msg_type=Int16)
    success = True
    if result is None:
      result = Int16()
      success = False
    response.alive = True
    response.success = success
    return response

  def SendPwm(self, request, response):
    FL, FR, DL, DR = request.forward_l_pwm, request.forward_r_pwm, request.down_l_pwm, request.down_r_pwm
    self.get_logger().info("Pwm sent:\n\t{}\n\t{}\n\t{}\n\t{}".format(FL, FR, DL, DR))
    result = self.bridge_.SendPWM(FL, FR, DL, DR)
    response.success = True if result is not None else False
    return response

  def SendKill(self, request, response):
    pass

  def RequestOrientation(self, request, response):
    from geometry_msgs.msg import Quaternion
    result = self.bridge_.AskForIMU(ros_msg_type=Quaternion)
    success = True
    if result is None:
      result = Quaternion()
      success = False
    self.get_logger().info("Orientation received: {}, {}, {}, {}".format(result.w, result.x, result.y, result.z))
    response.orientation = result
    response.success = success
    return response


  def RequestGnss(self, request, response):
    from geographic_msgs.msg import GeoPoint
    result = self.bridge_.AskForGps(GeoPoint)
    success = True

    if result is None:
      result = GeoPoint()
      success = False
    self.get_logger().info("GNSS: {}, {}, {}".format(result.latitude, result.longitude, result.altitude))
    response.gnss = result
    response.success = success
    return response
```

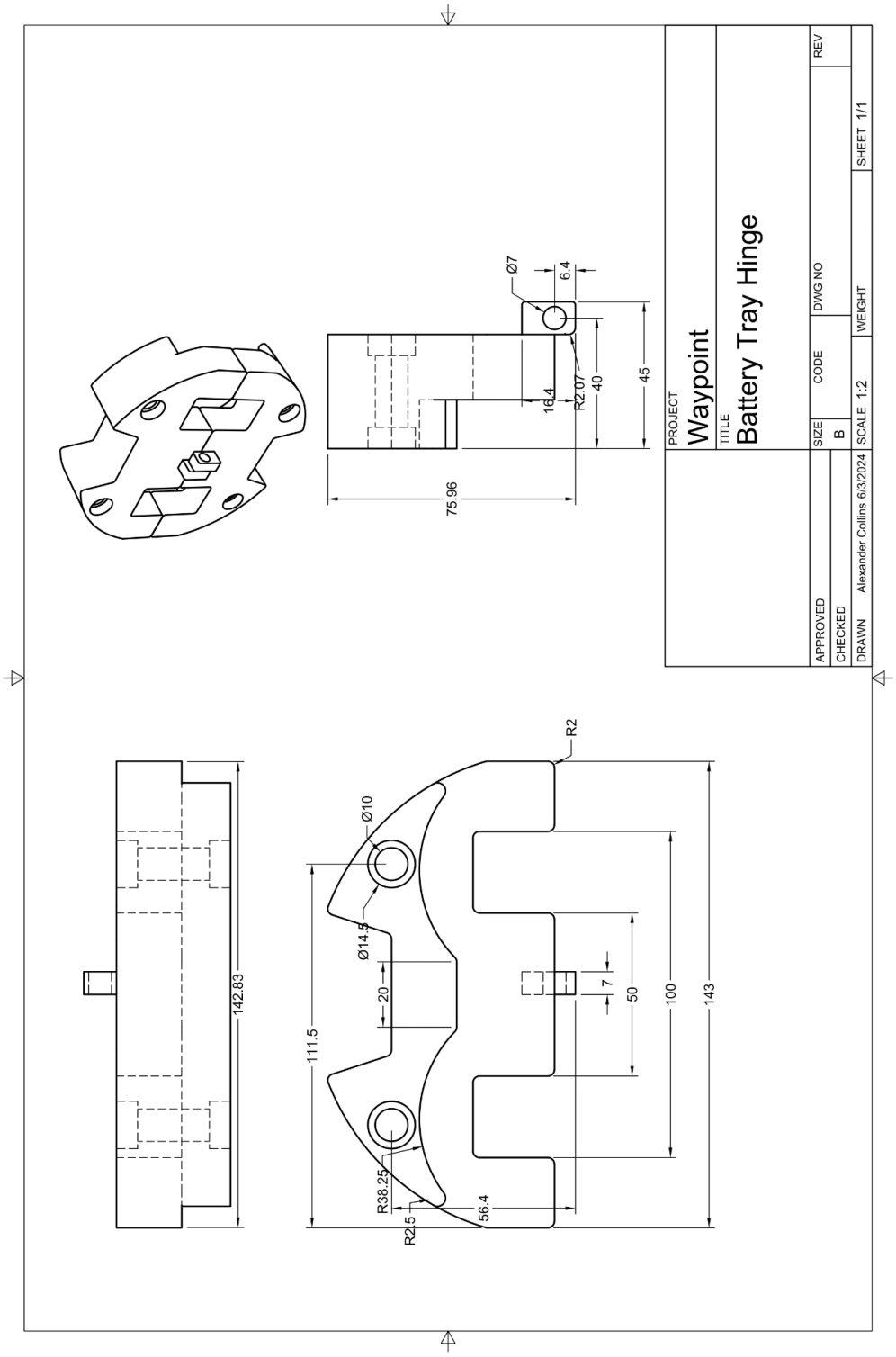
Appendix F: CAD Drawing of Electronics Tray Handle Receiver



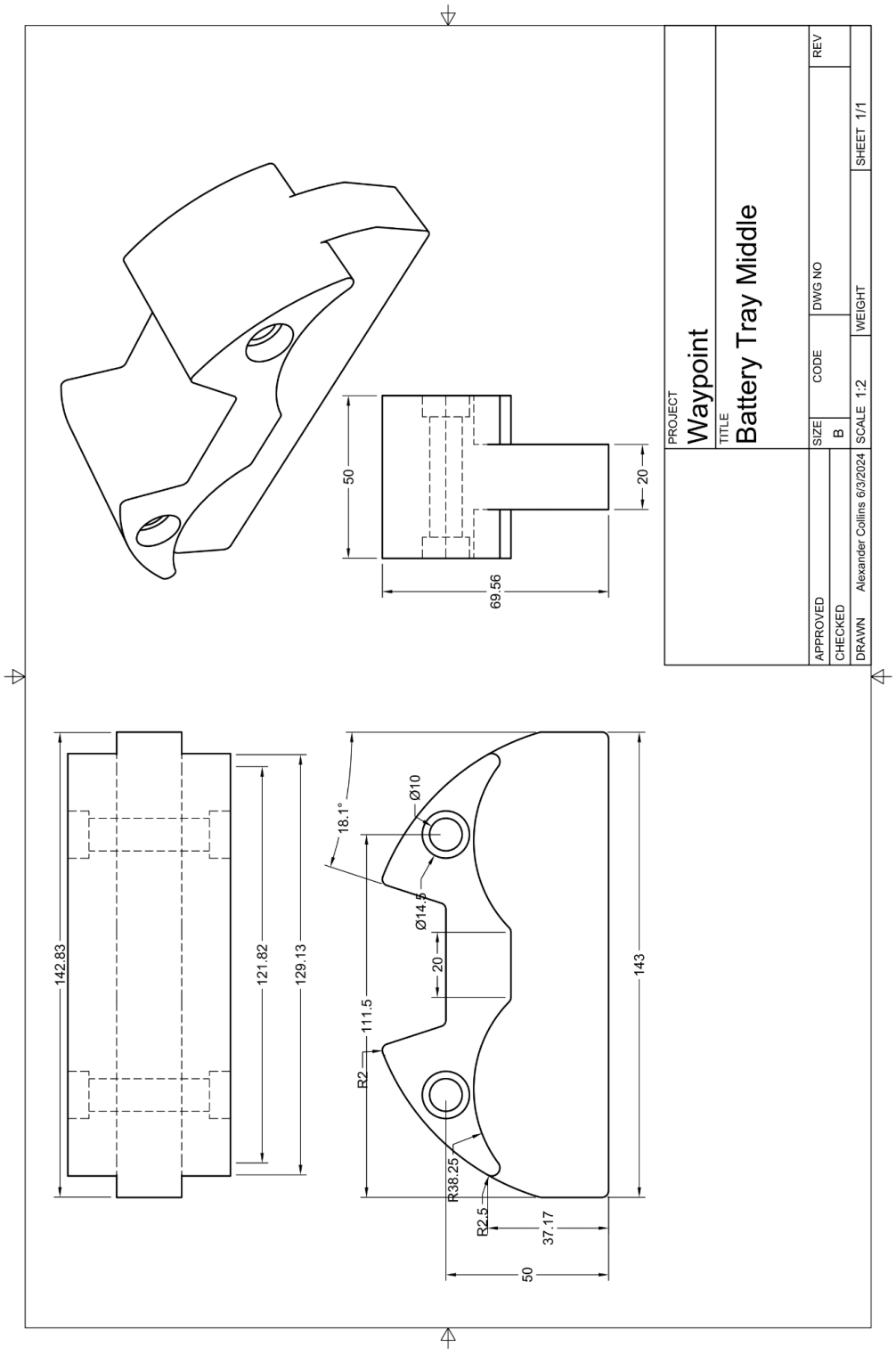
Appendix G: CAD Drawing of Electronics Tray Handle



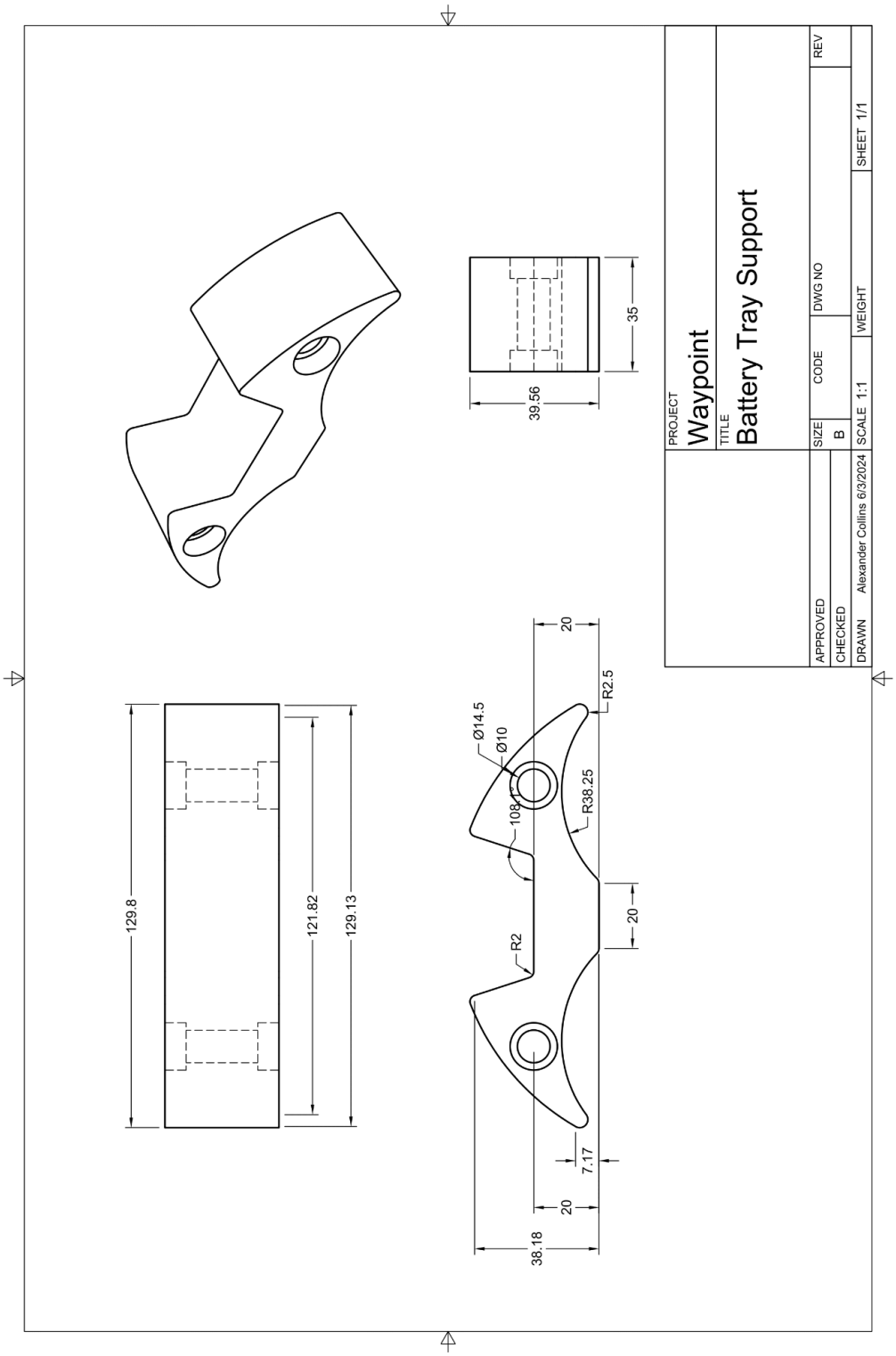
Appendix H: CAD Drawing of Battery Tray Hinge



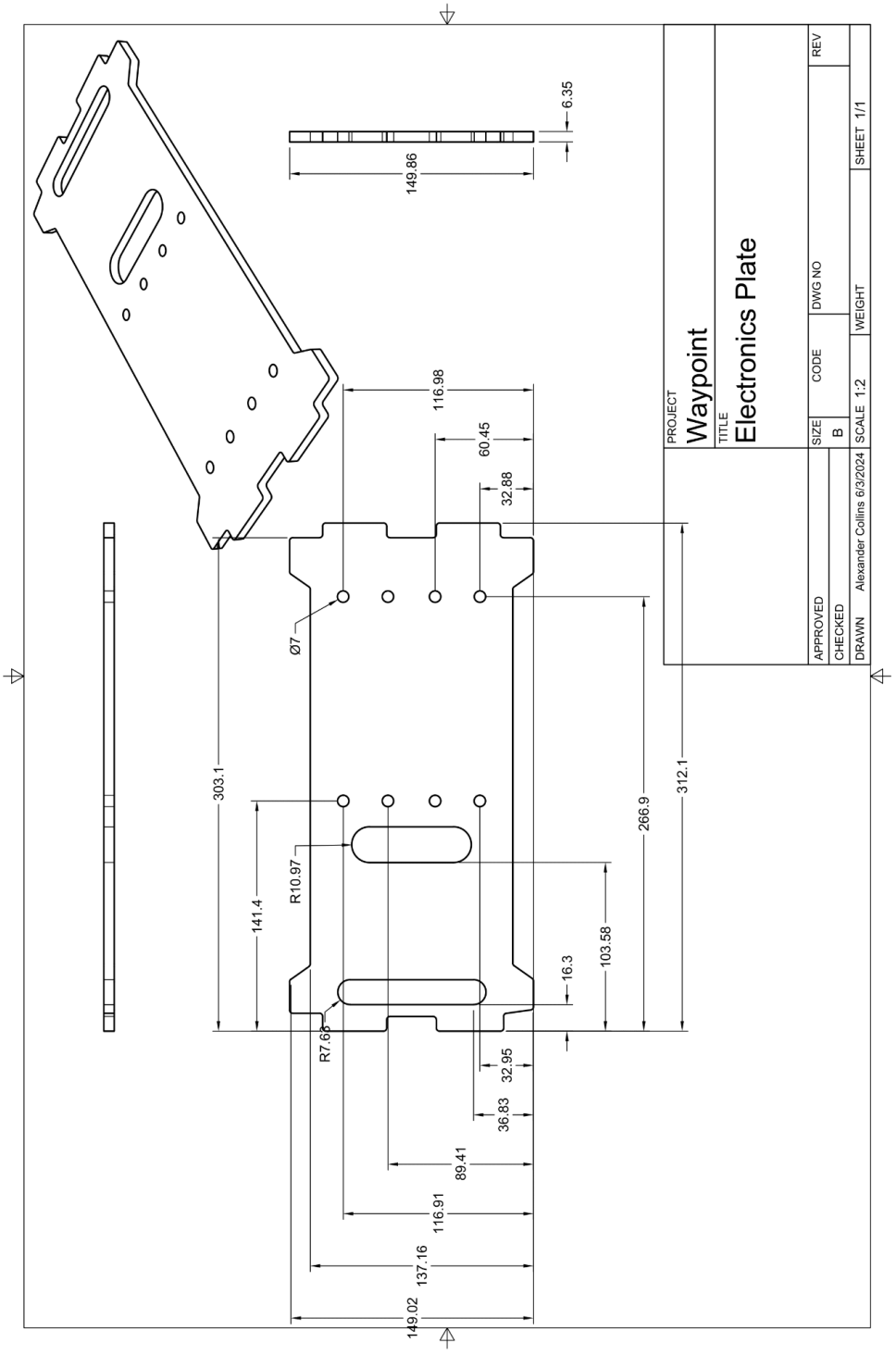
Appendix I: CAD Drawing of Battery Tray Middle



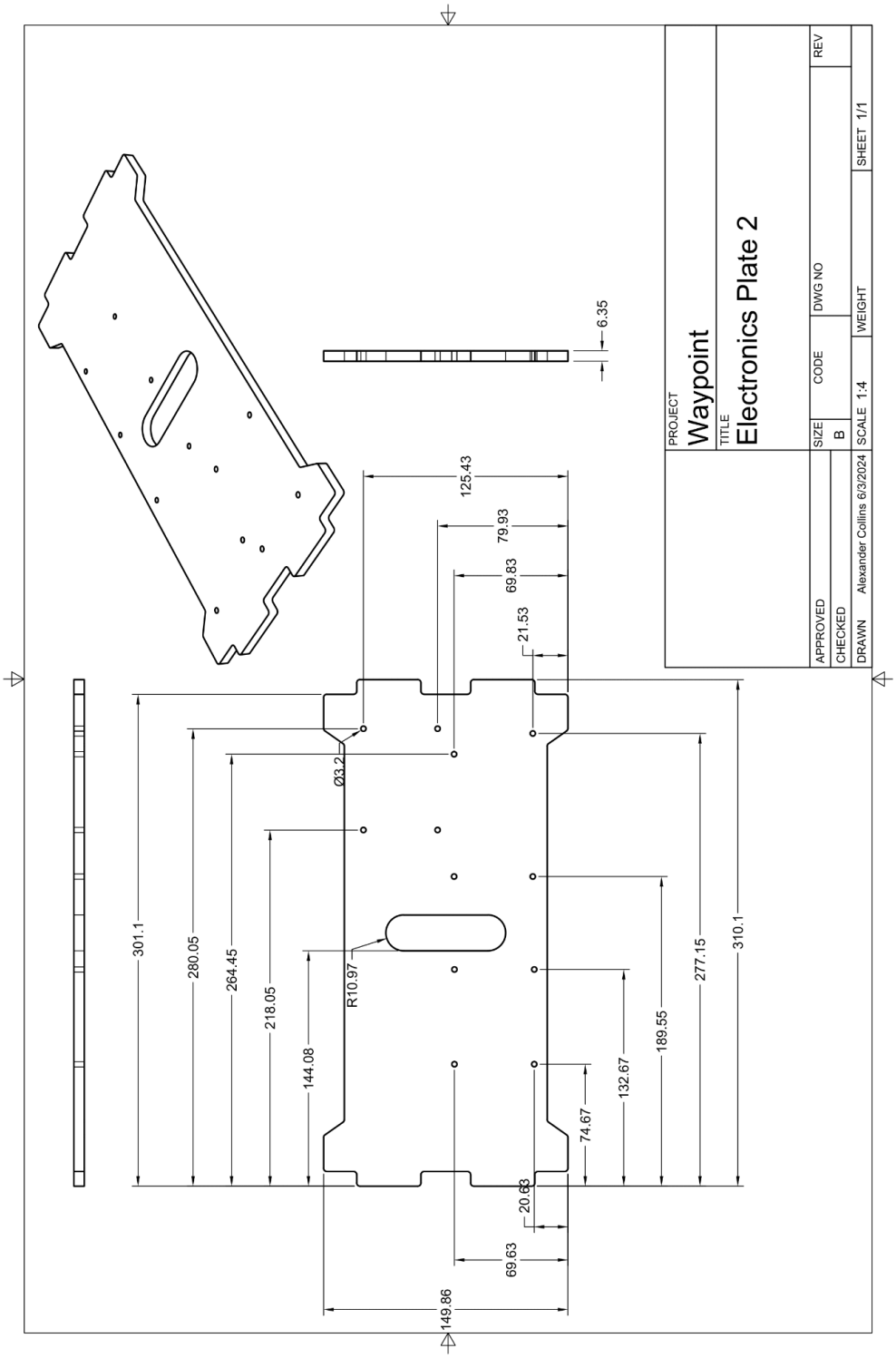
Appendix J: CAD Drawing of Battery Tray Support



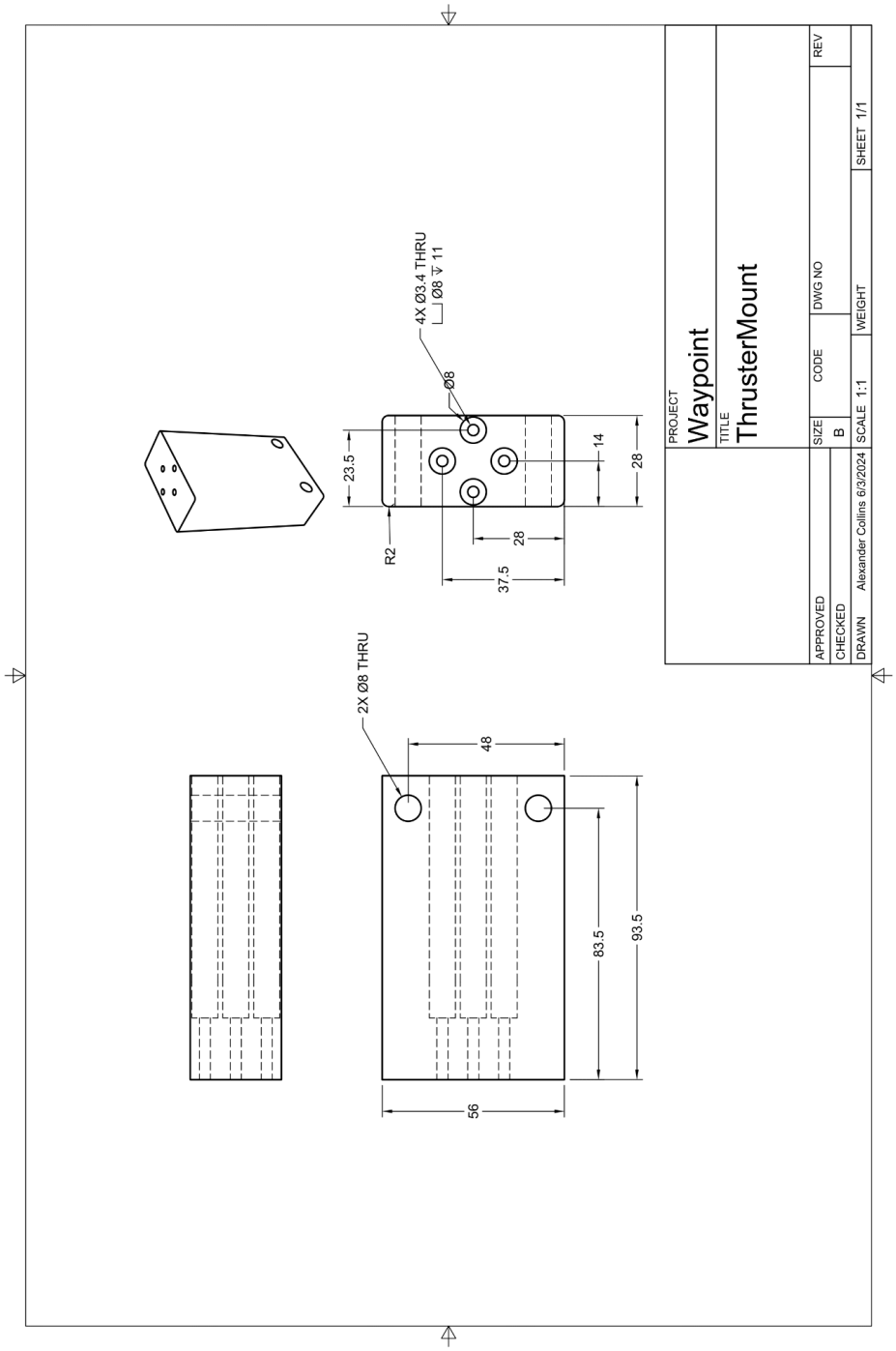
Appendix K: CAD Drawing of Electronics Plate (Bus-bar)



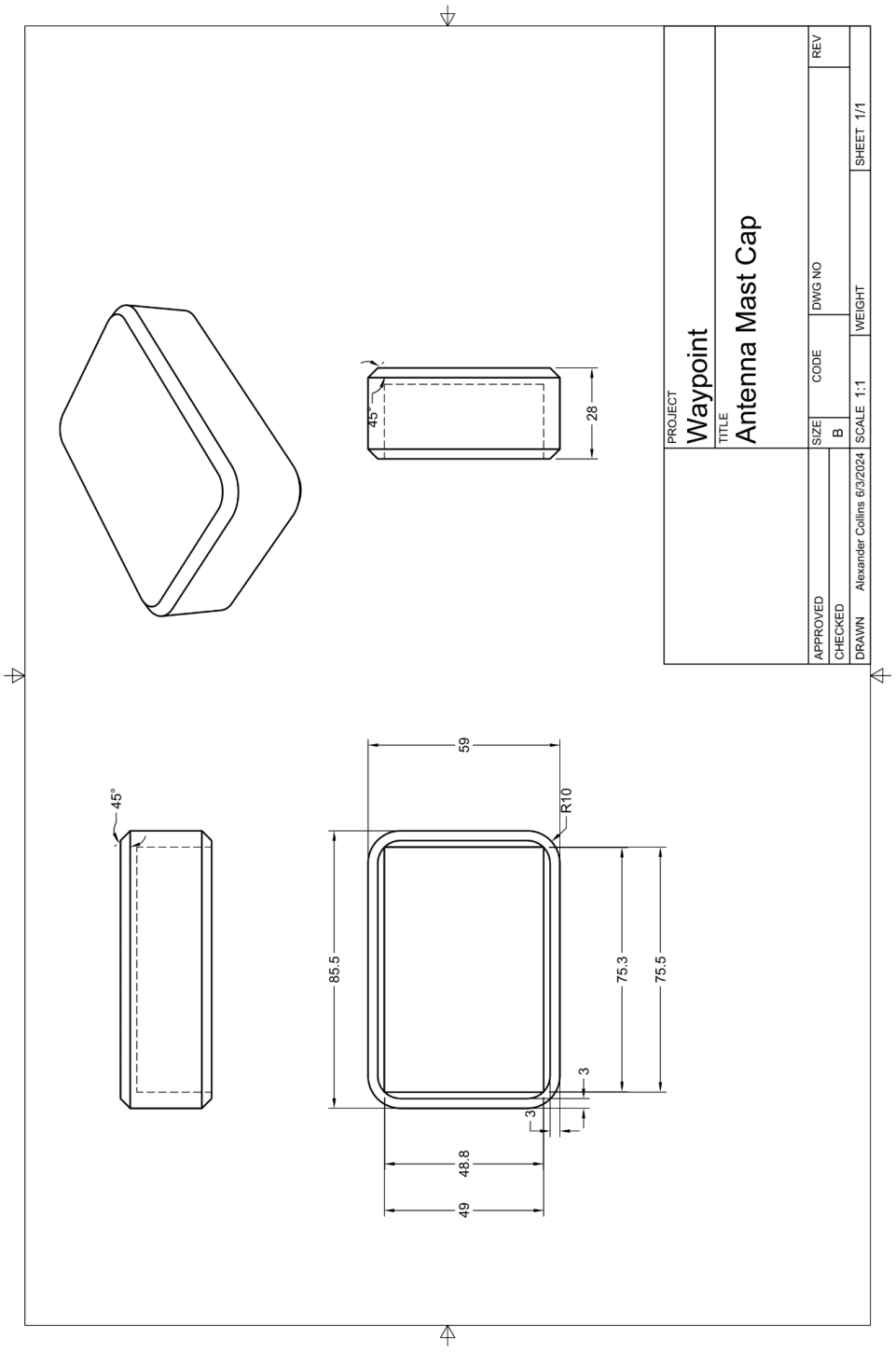
Appendix L: CAD Drawing of Electronics Plate (Pi and Mega)



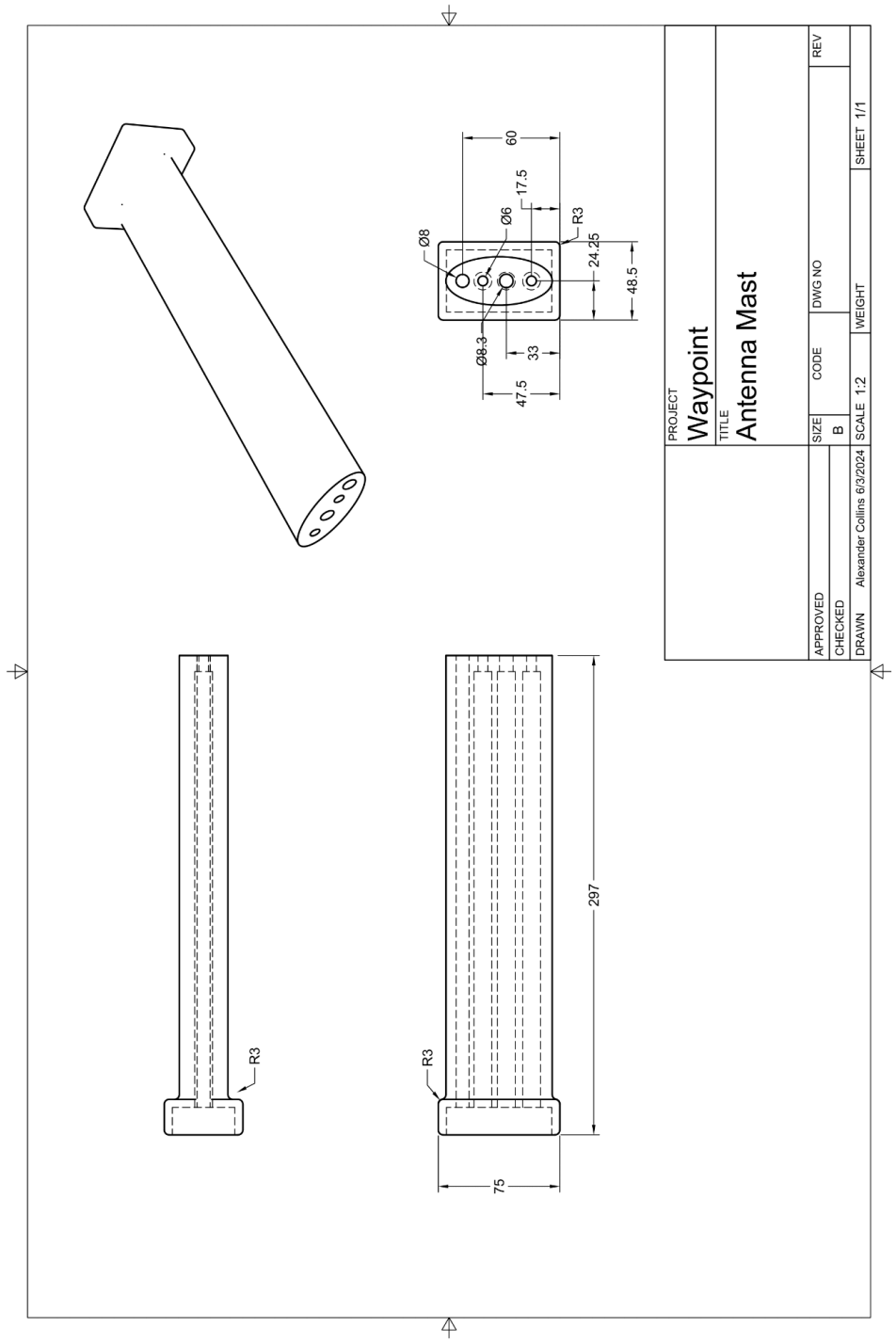
Appendix M: CAD Drawing of Thruster Mount Adapter



Appendix N: CAD Drawing of Antenna Mast Cap



Appendix O: CAD Drawing of Antenna Mast



Appendix P: Waypoint Profiler Deployment Checklist

Waypoint Profiler Pre-Deployment Travel Checklist

(adapted from Vertical Profiler Deployment Checklist written by Jenny Huynh)

1. Pack Waypoint Profiler into travel cases using bubble wrap
2. Check that the 4 batteries are at minimum operational capacity and pack them into ammunition cases with fireproof bags
3. Pack everything included in the Deployment Supplies List
4. Load everything into the SUV

Pre-Deployment Checklist

1. Check Waypoint Profiler and batteries for any visible damage after traveling
2. Check if the O-rings on the end caps for cracks and replace if necessary
3. Check electronics tray for any disconnected connections
4. Secure the rope onto the t slot rail
5. Check battery voltage
6. Plug the battery into the Waypoint Profiler and place it into designated location
7. Open one of the OK vent plugs
8. Close the battery end cap
9. Close the OK vent plug
10. Tighten all of the endcap penetrators
11. Perform pressure test

Test Procedure (repeat for every test)

1. Turn on Waypoint Profiler using the waterproof switch
2. Connect to the Wifi Router
3. Upload Waypoint Coordinates and deploy the profiler

Post-Deployment Checklist

1. Visually inspect Waypoint profiler for any damages
2. Dry off Waypoint profiler
3. Open battery end cap and Unplug batteries
4. Check battery voltage
5. Put battery back into ammunition case
6. Close battery end cap
7. Untie the rope
8. Repack Waypoint profiler into its cases
9. Pack and load up the toolbox and ammunition cases
10. Return to lab
11. Unload everything
12. Rinse down Waypoint Profiler
13. Check battery voltage and charge/discharge to storage voltage; place into yellow battery storage cabinet

Appendix Q: Approximation of Craft Weight Calculations

	Weight Calculations				
Location	Part	Quantity	Weight (g)	Total Weight (g)	Total Weight (lbs)
EXTERNAL	Nose Cone	1	2800	2800	6.172936
	T200 Thruster	4	344	1376	3.033557 12
	Tslot Rail	2	1317	2634	5.806969 08
	Antenna mast	1	460	460	1.014125 2
	End cap (Front)	1	1000	1000	2.20462
	End cap (Rear)	1	1280	1280	2.821913 6
	Thruster Mount	4	440	1760	3.880131 2
	Weight Rack Mount	6	615	3690	8.135047 8
	Acrylic Tube	1	4554	4554	10.03983 948
	Ballast Weights	4	2994	11976	26.40252 912
INTERNAL					
	Electronics Tray	1	3725	3725	8.212209 5
	Battery Tray	1	1355	1355	2.987260 1
				Total Weight of the Craft (lbs)	**80.71113 82**

Appendix R: Approximation of Center of Buoyancy Calculations

Relevant Information	
Center of buoyancy is the centroid of the volume of water displaced	
Observations	
the craft is horizontal when placed in the water, so the center of buoyancy of the craft in the x direction must be located at L/2 from either end	
X-coordinate	
2ft	
Observations	
When the craft is placed in the water, the waterline is up to the top of the tube leaving the top tslot rail above the waterline; therefore the centroid in the y direction is Douter/2	
Y-coordinate (in)	
0	(based on defined coordinate system in the picture above)

Appendix S: Approximation of Center of Gravity Calculations

X - Direction (using coordinate system established above)			
	Weight (lbs)	Distance from Tail End (in)	Weight*distance
battery tray	2.9872601	39	116.5031439
electronics tray	8.2122095	15	123.1831425
ballast #1	6.6	18	118.8
ballast #2	6.6	9	59.4
ballast #3	6.6	12	79.2
ballast #4	6.6	33	217.8
Thruster Flip	3.46	42.5	147.05
Thruster Forward	3.46	28.5	98.61
Antenna mast	1	11	11
nose cone	3.96	52.5	207.9
Total Weight	49.4794696		1179.446286
Centroid X (in)	**23.83708427**		
Y - Direction (using coordinate system established above)			
	Weight (lbs)	Distance from Tail End (in)	Weight*distance
ballast #1	6.6	7	46.2
ballast #2	6.6	-7	-46.2
ballast #3	6.6	-7	-46.2
ballast #4	6.6	-7	-46.2
Antenna mast	1	11	11
Total Weight	27.4		-81.4
Centroid Y (in)	**-2.97080292**	(from the centerline of the craft)	

Appendix T: Profiler Assembly Instructions

Electronics Tray Assembly:

1. Wire all necessary connections on the trays outside of the tube.
2. Make sure the ESC and battery cables are bundled.
3. Connect the all necessary cables to back end cap (Lora,GPS,debug, etc)
4. Slide the back end cap onto the trays rods ensuring the correct orientation of the tray.
5. With at least two people working in unison one holding the tray the other holding the back end cap, slide the entire unit into the tube ensuring the ESC and battery cables are in front of the tray.
6. Secure the end cap on the tube so it doesn't fall off.
7. Pull the ESC and Battery cables out the front of the profiler.
8. Attach the quick connects on the front end cap to the ESC cables.

Battery Tray Assembly:

1. Open the battery tray and put in the batteries all 4 oriented towards the back of the tube.
2. Secure the other side of the battery tray ensuring no cables are smashed and the plugs are accessible.
3. With at least two people one holding the tray the other plugging the batteries in, first plug in the back two batteries (the ones closer to the back of the profiler).
4. Route the other two battery cables through the groove in the top of the tray and plug them in to the front two batteries.
5. Fasten the front two battery cables to the tray using the reusable zip tie.
6. Flip the tray over so those cables are on the bottom.
7. Slide the tray in the tube ensuring the ESC cables are aligned in the top groove of the try.
8. Reattach the front end cap if necessary, pulling a vacuum after pushing it in past the first O-ring.

Full Assembly:

1. Follow Instructions for Electronics Tray Assembly.
2. Follow instructions for Battery Tray Assembly. Omit putting batteries in the tray if preparing for deployment.

Disassembly:

For disassembly simply follow the instructions in reverse order

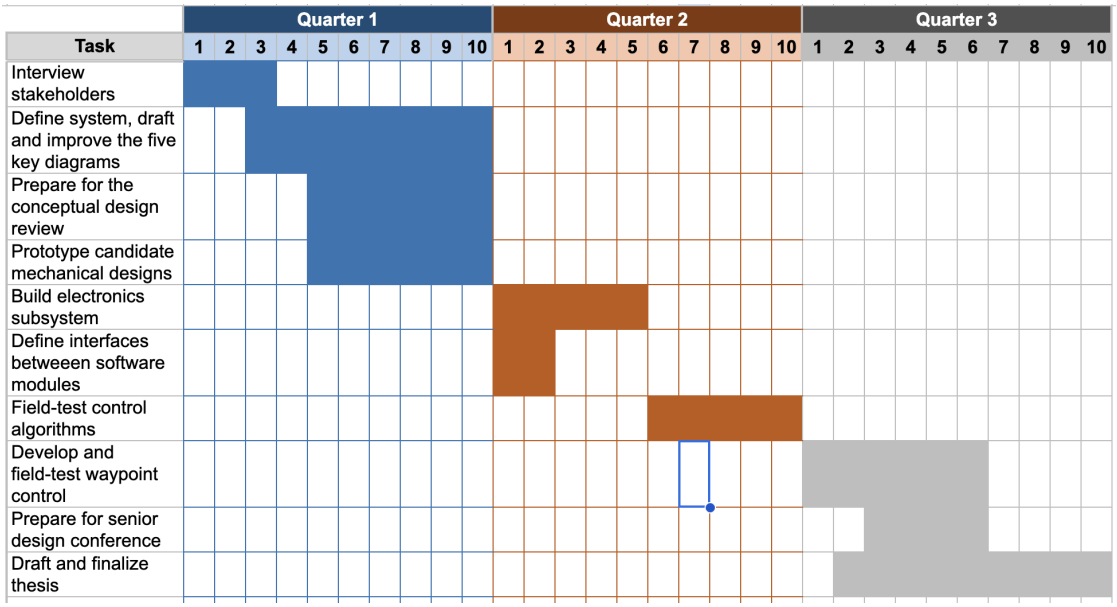
Appendix U: Bill of Materials

Waypoint Profiler			
Component	**Quantity**	**Price**	**Total**
T200 Thruster	4	210	840
Tube	1	540	540
LoRa Antenna	1	28	28
GPS Antenna	1	20	20
Raspberry Pi 4	1	35	35
Arduino Mega	1	50	50
Buck Converter	2	10	20
Bus-bar	2	22	44
Battery Tray Rods	4	14	56
T slot	2	20	40
Blank Penetrators	10	5	50
Vent Plug Penetrator	2	10	20
Pressure Relief Valve Penetrator	2	5	10
Ping Sensor	1	410	410
M5 Bolts	4	10	40
M5 Nuts	4	10	40
M3 Screws	1	5	5
M3 Nuts	1	5	5
M6 Bolts	3	10	30
M6 Nuts	3	10	30
Wifi Router	1	32	32
Cable Splice Kit	4	20	80
WetLink Penetrators	2	50	100
SMB Male to SMB Female	2	10	20
SMB Male to SMA Male	2	8	16
Marine Epoxy	1	150	150

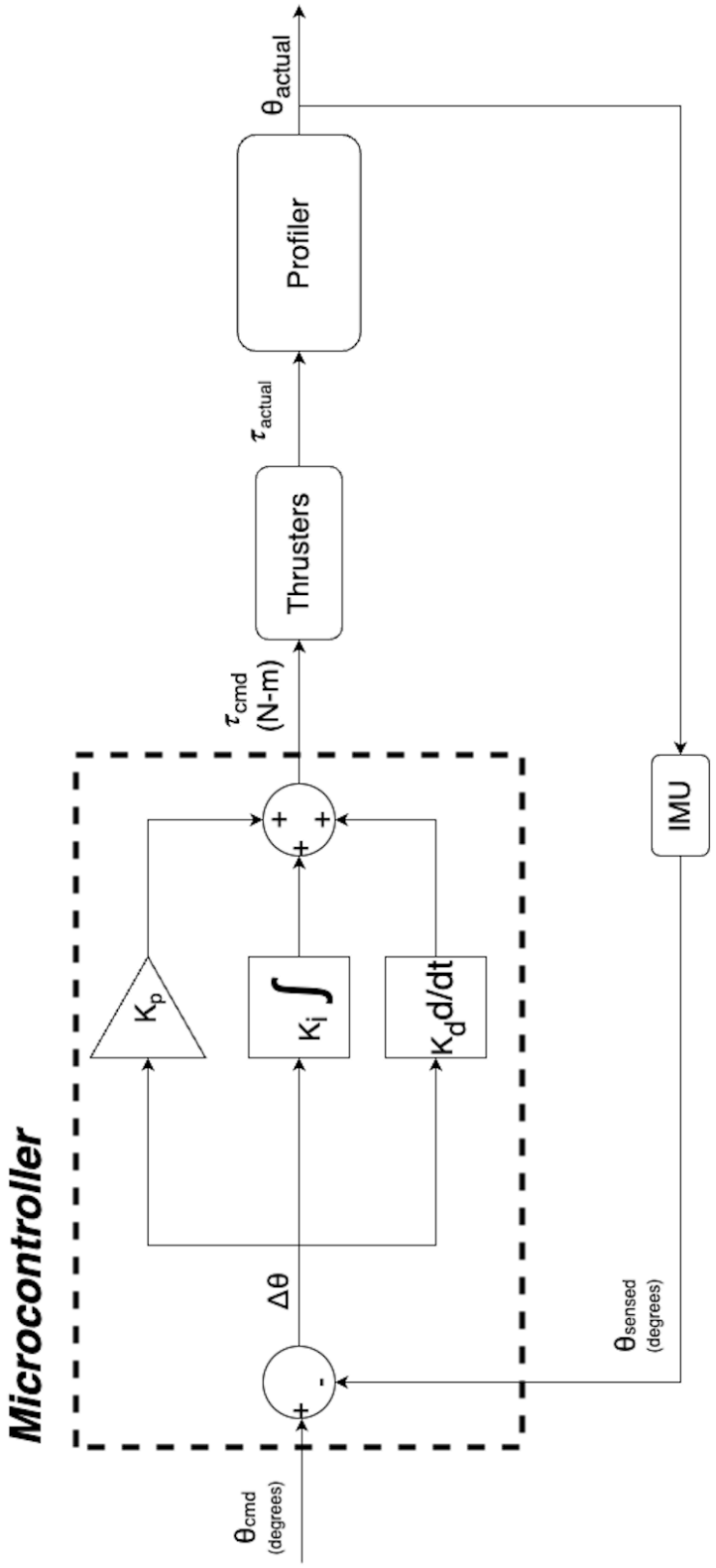
Component	Quantity	Price	Total
Loctite Marine Epoxy	5	38	190
End Cap Flange	2	84	168
End Cap Plate	2	52	104
End Cap O-Rings	2	14	28
Marine Grease	1	12	12
Diving Strobe Lights	4	80	320
Zip Ties	2	22	44
T200 Thruster Cable	1	32	32
PETG Filament	20	10	200
14 Gauge Wire	1	10	10
Ring Connectors	2	12	24
Quick Release Latches	2	15	30
Penetrator O-Rings	2	3	6
Potted Penetrators	4	5	20
1/4" Acrylic Sheets	4	20	80
PLA+ Filament	5	20	100
Carabiners	1	10	10
Wire Connectors	1	12	12
Leak Sensor	5	3	15
GPS Chip	1	30	30
IMU	1	20	20
LoRa Chip	1	20	20
T200 ESC	4	38	152
Barometric Sensor	1	10	10
Desiccant Bags	6	10	60
Thruster Mounting Screws	1	14	14
Rubber Strips	2	12	24
EndCap Switch	1	20	20

Component	Quantity	Price	Total
Potting Kit	2	10	20
Epoxy Mixing Cups	1	14	14
Double Sided Tape	2	12	24
Rod Nuts	1	11	11
Nylon Standoffs	1	10	10
Arduino Cables	2	17	34
Water Sampler			
Tube	1	455	455
Servo	1	395	395
Rubber Plugs	15	8	120
Springs	4	8	32
Rods	4	14	56
Rod Nuts	1	11	11
Bungee Cord	1	22	22
Component	**Quantity**	**Price**	**Total**
Batteries	4	380	1520
		Total	**$7190**

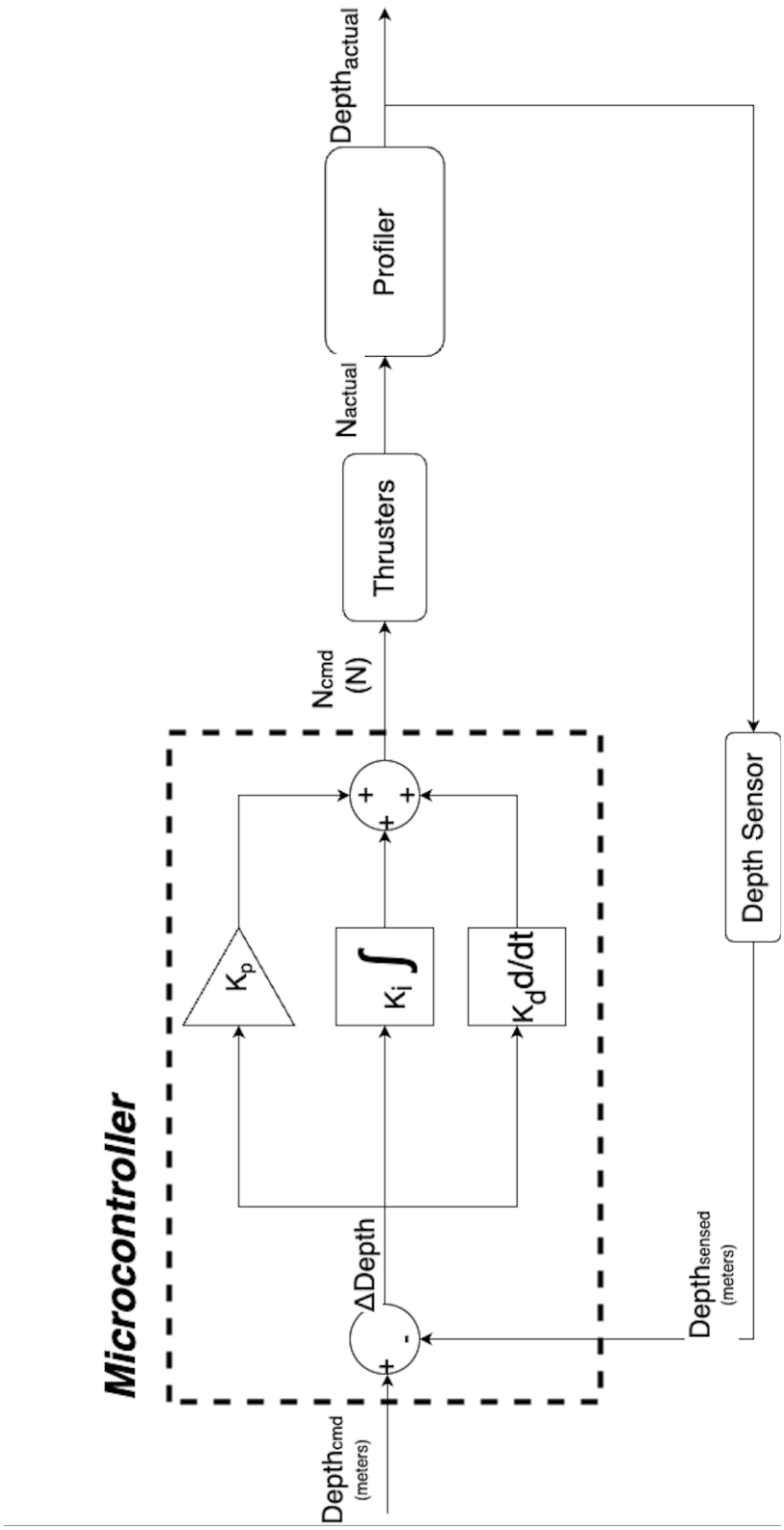
Appendix V: Gantt Chart of High-level Priorities



Appendix W: Control Block Diagram: Flipping & Heading Control



Appendix X: Control Block Diagram: Depth Control



Appendix Y: Profiler Quickstart Guide

Waypoint Profiler Quickstart Guide

Overview:

This document is intended to act as your guide for use and development of the waypoint profiler specifically regarding the software on the system. For details on assembly or deployment refer to either of those checklist documents. This guide will be broken up into three sections which discuss the microcontroller setup and our two primary branches of software: the Subsystem-test code and the System-test code.

The Subsystem-test code predominantly uses the Arduino to run a limited version of the vehicle's state machine control to verify functions such as flipping, diving, and waypoint navigation. The Raspberry Pi in the Subsystem-test code is used only for data-logging and remote communication over WLAN.

The System-test code is our prototype of the full state machine control where the Raspberry Pi acts as the master and the Arduino acts as the minion.

All relevant files are in the corresponding folders. To better understand this document and the system as a whole, please refer to the Software Architecture diagram in the Diagrams folder.

Microcontroller Setup:

Our system is implemented using both a Raspberry Pi4 and an Arduino Mega. So you need to set up and run the code on both microcontrollers.

To setup the Arduino Mega:

1. Download the Arduino IDE on your computer
2. Take the ArduinoMain folder and place it in the Arduino file folder on your computer (usually in Documents)
3. Open the Arduino IDE and open the file with the .ino extension in the ArduinoMain folder
4. Use a USB A to USB B cable to connect your computer to the Arduino Mega.
5. Go Tools in the Arduino IDE and ensure that you have the right board and port.
6. Upload the sketch to the Arduino and unplug it after its done

To setup the Raspberry PI:

1. If the Pi has not been setup follow these instructions to install the OS https://www.raspberrypi.com/documentation/computers/getting-started.html#getting-started-with-your-raspberry-Pi
2. After the Pi finishes booting up, use an ethernet cable to plug both the Pi and your computer into the same router. Alternatively you can connect to the router wirelessly through your computer if the router has a LAN setup.
3. Once you are connected to the router use an application such as Angry IP Scanner to find the IP address of the PI. The hostname should be something like ubuntu.lan

4. Open a terminal window and ssh into the Pi using: ssh ubuntu@<ip address>. The password is profiler.
5. At this point if you are running the ROS version of the code follow these instructions to install ROS2 humble if it has not been installed on this PI. https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html
6. If you have just set up the Pi and need to upload the files, use something like scp or github to remotely transfer the files. Side Note: it is highly recommended that you use github or some other form of version control to remotely store the files in case the Pi dies so you don't lose all your work!.
7. Connect the Pi to the Arduino and the USB A to USB B cable.

At this point you are all set up and ready to run the code. Follow the instructions for which version of the code you want to run and make sure that version of code was the one you uploaded to the Arduino. Also note that while the Arduino is powered on the code runs automatically but should stall until it receives the start command from the PI. If you need to stop it simply cut power by unplugging it or turning the E-Stop. Also note that abruptly powering off the Pi can cause issues, so whenever possible run the command: sudo poweroff before cutting power to the PI. Also when testing in the field you will lose connection in the terminal which is okay as it is not intended to be connected during the test. Simply lift the tail end with the router slightly out of the water after the test and you will reconnect.

Subsystem-test Version:

This version of the code primarily uses the Arduino to do all the computations but still requires the Pi to power it on. This version is intended for testing the core functionalities of the system. After following the Microcontroller Setup instructions you can run this code by navigating to the directory with the Script.py file. Then execute a command in the terminal to run one of the tests. Here are the different command formats:
dive_test -> ./Script dive <Kp> <Ki> <Kd> <duration> <target depth>
flip_test -> ./Script.py flip float <Kp> <Ki> <Kd> <duration>
pressure_test -> ./Script.py pressure <duration> <max_deviation>
waypoint_test -> ./Script.py waypoint <Kp> <Ki> <Kd> <duration> <distance_threshold> <heading_threshold> <goal_lat> <goal_lon>

Everything in <> are the parameters where Kp, Ki, and Kd are the constants for the PID controller. After executing the command, wait for it to fully complete, and then you may run another command.

System-test Version:

This version of the code primarily uses the Raspberry Pi with ROS2 humble installed to do all the computations but still uses the Arduino to connect to the sensors and servos. This will be the finalized version of the code that fully integrates the system but is still in development. After following the Microcontroller Setup instructions you can run this code by navigating to the

root directory of the ROS workspace, likely called src. Build and source your workspace following the instructions:
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html
After that run: ros2 launch launch/launch.py to start the profiler.
For a more in depth understanding of using ROS, be sure to go through the beginner tutorials on the website above.