Project Final Report

BlueFish: Seafloor Imaging Towfish

BlueRobotics

MECH 400 – Group 6

Noah Mar	V00872204
Malaki Vandas	V00844796
Raimund Mullin	V00868338
Nigel Swab	V00871967
Clayton Moxley	V00876017

Instructor: Dr. Curran Crawford

Mechanical Engineering

University of Victoria

April 19th, 2021

Glossary

ADC	Analog-to-Digital Conversion
Angle of Attack	Angle between a reference line (often the chord of an airfoil) and the vector parallel to the fluid flow direction.
AUV	Autonomous Underwater Vehicle
CFD	Computational Fluid Dynamics
Chord	The centerline length of a fin/airfoil from leading to trailing edge.
COTS	Commercial-Off-the-Shelf
CSV	Comma-Separate Values
DFA	Design for Assembly
DFM	Design for Manufacturability
FXTI	Fathom-X Tether Interface
GCS	Ground Control System
GCS	Ground Control Software
GUI	Graphical User Interface
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IO	Input/Output
ISR	Interrupt Service Routine
LED	Light-Emitting Diode
MCU	Microcontroller Unit
NACA	National Advisory Committee for Aeronautics (Superseded by NASA: National Aeronautics and Space Administration)
OS	Operating System

PCA	Printed Circuit Assembly
PID	Proportional-Integral-Derivative
PLM	Product Lifecycle Management
PSM	Power Sense Module
PWM	Pulse Width Modulation
ROV	Remotely Operated Vehicle
SSH	Security Shell
Threading	Threading is a practice in python (and other languages) that allows for concurrent programming with shared data space. This allows multiple processes to run almost in parallel which can speed up programs and allow a GUI to retain responsiveness while logging data at high speeds.
Thru Hole	A hole that continues through all surfaces.

Abstract

This document outlines all the work completed in the BlueFish capstone project, which was undertaken by Group 6 of the Spring 2021 MECH 400 project course for the partial fulfillment of the bachelor of engineering degree at the University of Victoria. The project encompassed the development of an initial prototype for a seafloor imaging towfish for Blue Robotics, a marine robotics company based out of California with a branch in Victoria, BC, which intends to further develop the initial prototype into a marketable product. The project scope was determined to include the design, construction, and testing of a fully functional and physical seaworthy prototype, as well as to produce an image mosaic of the seabed. The project management technique utilized was the Sprint method, in which the progress was governed by the completion criteria of multiple sprints.

The design of the mechanical system began with the drag analysis of existing hydrodynamic profiles that were applicable to the geometry of the BlueFish. CFD simulations were performed and several physical prototypes of the hydrodynamic shell, the BlueFry, were constructed for validation. The main watertight enclosure was chosen to be a Blue Robotics 3" OD cast acrylic tube, which would house the control boards and battery, while also serving as the chassis for the hydrodynamic and control systems. The tow rack assembly and tether were also designed to utilize the Blue Robotics Fathom-X Tether Interface and Fathom Slim Tether for communication between the BlueFish and the end user. The final prototype was designed and modelled in preparation for the construction of the physical prototype and final fluid and structural simulations were performed as a preliminary validation.

Simultaneously, the design of the mechatronic system began with the research of individual components and the development of flow diagrams and wire schematics. The core of the mechatronic system was determined to require an Arduino Uno interfaced with a Raspberry Pi 3B+. The Arduino was used for the control of the motors, the lights, sonar altimeter, pressure sensors, leak sensors, and the IMU sensor. Similarly, the Raspberry Pi was used to display the GUI, send and receive data from the Arduino, operate the onboard camera, and to communicate with the topside end user. The first functional prototype, the BlueFish, was successfully constructed with an integrated mechatronic system. The BlueFish was then tested and evaluated with respect to the project scope, user requirements, and engineering specifications. However, further work and iteration of the BlueFish prototype is still required to produce a fully marketable product.

Overall, the project undertaken by Group 6 of the Sprint 2021 MECH 400 project course of producing an initial prototype of the BlueFish was successful and the prototype will see continued design and iteration of both the mechanical and mechatronic systems in the pursuit of a marketable product.

Table of Contents

List of Figures

List of Tables

1. Introduction

This document presents the all-encompassing work for Group 6 in MECH 400. The primary purpose of this document is to present all technical work and development of the project, while also featuring the project management experience, user requirements, engineering requirements, and all technical documentation and progress over the course of the project.

1.1. Background

This project involves the development of a seafloor imaging towfish prototype, which will be towed from a marine vehicle while maintaining a constant altitude above the seabed to photograph the seafloor. The photos will be spaced evenly to develop a photomosaic. This MECH 400 project is undertaken by Group 6, the Bottom Feeders, in the Spring 2021 term at the University of Victoria. The client is Blue Robotics, a marine robotics company based out of California with a branch in Victoria. The final prototype will likely be used by Blue Robotics to design a marketable product.

Towfish are becoming more common in the hobby, science, and industrial communities. Mounting a camera to a towfish, as is intended for this project, rather than to a surface vessel, can result in higher resolution and quality imaging. This is due to the increased stability of the towfish, which is not affected by wave action on the surface. A towfish also enables varying degrees of resolution of imaging if the altitude or depth is controlled. Thus, they are used for many different purposes, including but not limited to pipeline monitoring, seafloor exploration and mapping, water quality monitoring, national defence, and wildlife monitoring. The goal of the design being developed in this project is to be an open platform for users.

1.2. Project Scope

The project scope encompasses the design, fabrication, and testing of a seafloor imaging towfish, integrated with a BlueBoat from Blue Robotics. This also includes the seafloor image data collection and presentation. The primary goal was to be able to control the depth of the Bluefish to increase the range of depths that the Bluefish can effectively image using control surfaces and a Ping sonar transducer (single beam echosounder). The secondary goal of the project was to enable the BlueBoat to collect clear seafloor images using the Bluefish across the range of sea-states that the BlueBoat can handle. The final deliverables for the project are described below. To add more specificity to these expected final deliverables, a user requirements table and an engineering specifications table were created, which are displayed in Section 4.2 and used to evaluate the final prototype.

1.2.1. Minimum Final Deliverables

The final deliverables of this project will, at minimum, be the following:

- *A physical prototype of the Bluefish, which acts as a proof of concept. It may be a proof of concept that can maintain stability/depth that a high-resolution camera could theoretically mount to.*
- *Solidworks CAD files, related engineering calculations, and a completed bill of materials.*

1.2.2. Optimal Final Deliverables

The final optimal deliverables of this project, in addition to the minimum final deliverables below, were to be the following:

- *Produce a high-resolution image mosaic of the seabed using the Bluefish, towed by the BlueBoat.*

- *A fully functional and seaworthy physical prototype of the Bluefish.*
- *Produce an image mosaic of a specified area (TBD) of the seabed outside of Sidney, BC, Canada.*
- *A retail pricing analysis to prove the retail cost requirements are being met.*

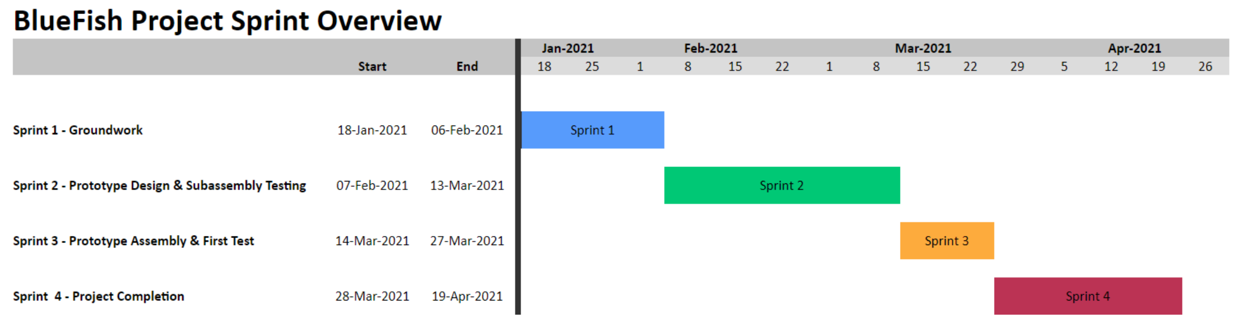
2. Project Management

Given the condensed timeline of this project, the selected project management technique was the sprint method, in which the project work was governed by the completion criteria of each sprint. Using a work breakdown structure, the team worked together to approximate the sprint schedules and identify the completion criteria for each. The result was a three-sprint schedule to design, manufacture, and assemble a first working prototype and complete the project.

The completion criteria of the first sprint, entitled 'Groundwork', includes the construction of hydrodynamic prototypes, the definition of the control system wiring communication, and the ordering of Blue Robotics COTS control components. The completion criteria of the second sprint, entitled 'Prototype Design and Subassembly Testing', included the design of the first working prototype, the design of the testing (towing) apparatus, and the completion of subassembly tests, such as a watertight enclosure test and an initial full-system controls test.

The completion criteria of the third sprint, entitled 'Prototype Assembly, Tests, and Project Completion,' includes the manufacturing and assembly of the first working prototype, testing apparatus, depth control and tuning validation tests, as well as the completion of the final deliverables.

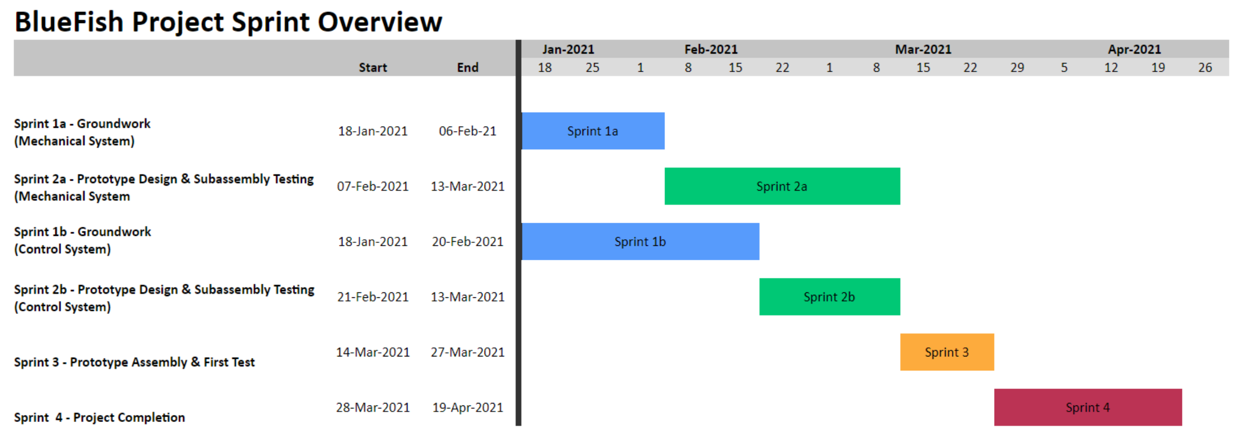
The original project plan allotted approximately seven weeks for the development of BlueFish I, which was to be the first prototype to have an integrated control system. The remaining five weeks were to be reserved for Sprint 3, or full-integration, testing, and project completion. It should be noted that the original plan also included a Sprint 4, but Sprint 3 and Sprint 4 were combined for organizational purposes. Sprint 1 was originally planned to have a two-week duration, ending on February 6th, but it was quickly realized that this deadline only accommodated the mechanical portion and not the control system portion of the project. Thus, the Sprint 1 deadline for the control system was postponed two weeks to February 20th and the Sprint 2 deadline was concurrently postponed to March 13th. Similarly, the mechanical portion of Sprint 2 was also assessed to need extra time and its deadline was also postponed to March 13th. This would have left five weeks for Sprint 3, arguably the most time-consuming portion of the project. However, due to more last-minute requests from Blue Robotics, a large component of the mechanical design had to be redesigned, ultimately pushing the overall deadline for Sprint 2 to March 31st. Thus, due to this second postponing of Sprint 2, Sprint 3 was finally pushed to begin on April 1st. With sprint 3 consisting of final integration, testing, the final report, and final video, the available time remaining resulted in an extremely ambitious push to the finish. With many logistical and technical integration obstacles appearing in this final period (see below), testing was limited to one day, with the prior four days being consumed by troubleshooting and new communication issues. The original, modified, and final sprint overviews are available for comparison in Figure 1 through Figure 3. Additionally, a screenshot of the top-level, finalized, Monday.com Gantt chart for Sprints 1, 2, and 3 can be found in Appendix A. The full detailed view of the Gantt chart can be accessed using the Monday.com link provided in the aforementioned appendix. Moreover, a detailed breakdown of individual sprints and their respective assignees, coassignees, completion status, timeline, and completion criteria can be found in Appendix B or in the Monday.com link provided.



BlueFish Project Sprint Overview

	Start	End
Sprint 1 - Groundwork	18-Jan-2021	06-Feb-2021
Sprint 2 - Prototype Design & Subassembly Testing	07-Feb-2021	13-Mar-2021
Sprint 3 - Prototype Assembly & First Test	14-Mar-2021	27-Mar-2021
Sprint 4 - Project Completion	28-Mar-2021	19-Apr-2021

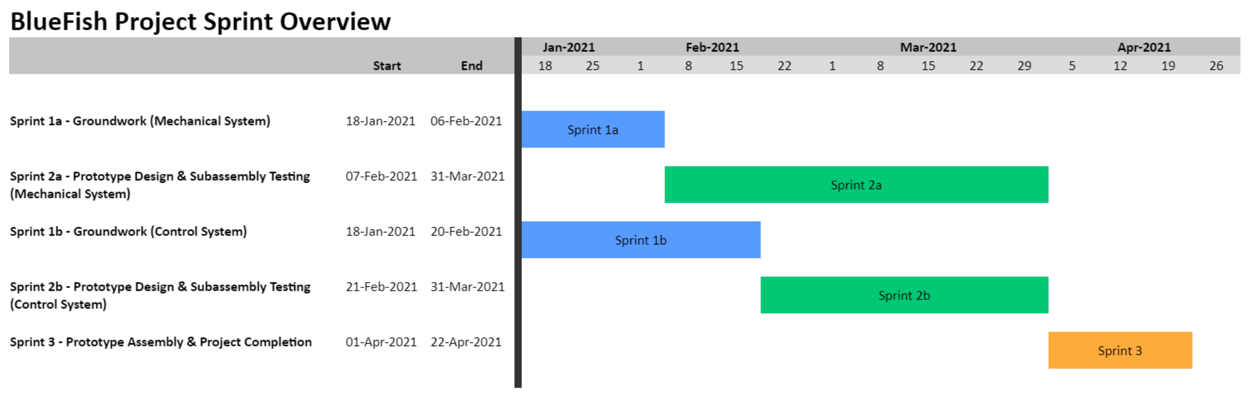
Figure 1: Original BlueFish Project Plan.



BlueFish Project Sprint Overview

	Start	End
Sprint 1a - Groundwork (Mechanical System)	18-Jan-2021	06-Feb-21
Sprint 2a - Prototype Design & Subassembly Testing (Mechanical System	07-Feb-2021	13-Mar-2021
Sprint 1b - Groundwork (Control System)	18-Jan-2021	20-Feb-2021
Sprint 2b - Prototype Design & Subassembly Testing (Control System)	21-Feb-2021	13-Mar-2021
Sprint 3 - Prototype Assembly & First Test	14-Mar-2021	27-Mar-2021
Sprint 4 - Project Completion	28-Mar-2021	19-Apr-2021

Figure 2: First Modified BlueFish Project Plan.



BlueFish Project Sprint Overview

	Start	End
Sprint 1a - Groundwork (Mechanical System)	18-Jan-2021	06-Feb-2021
Sprint 2a - Prototype Design & Subassembly Testing (Mechanical System)	07-Feb-2021	31-Mar-2021
Sprint 1b - Groundwork (Control System)	18-Jan-2021	20-Feb-2021
Sprint 2b - Prototype Design & Subassembly Testing (Control System)	21-Feb-2021	31-Mar-2021
Sprint 3 - Prototype Assembly & Project Completion	01-Apr-2021	22-Apr-2021

Figure 3: Second Modified and Final BlueFish Project Plan

The project management experience came with many technical and logistical challenges. The geographical separation of group members caused the project planning process to be more structured and scheduled, rather than continuous. This led to the necessity of online whiteboard applications, which are sufficient, but inferior to in-person meetings with a physical whiteboard. Additionally, the primarily used project management platform, Subtask, which was selected based on the course suggestion and the accessibility of its free plan, quickly become insufficient for the team's needs due the application's limitations using an unpaid subscription. Thus, the team was forced to migrate and use Monday.com, an easy to use, well-equipped, and visually appealing platform, as its primary project management tool, with an easy workaround to make the free plan marginally satisfactory. However, within the final two weeks of the project, the team was able to acquire a Monday.com sponsored, full-access account to use. After a second migration of the project management system to the full-access version of Monday.com, the team was finally able to produce the required visuals and perfect our project management system.

Additionally, as the team was working with a local chapter of Blue Robotics, logistical and communication challenges proved to be a significant challenge. Although expected, creating an agreed-upon and clear set of user requirements, engineering specifications, and high-level design concepts took longer than initially anticipated, with frequent client meetings and design discussions to facilitate progress in a timely manner. Another aspect was the logistical aspects of working with Blue Robotics and their respective associates. Overall, as this project was not of highest priority for the company, the required Blue Robotics contacts were not always available for assistance for their resources. For instance, ordering and shipping COTS Blue Robotics components took much longer than expected due to internal Blue Robotics shipping miscommunications. This issue proved to be significant, as one of two mechatronics members, located in Calgary, BC, who required the aforementioned components, never received his order, and was forced to use personal components at his disposal and to "digitally share" the apparatus and components of the second member, located in Vancouver, BC.

In addition, another issue arose in the ordering of custom machined components from Kaierwo, a manufacturing company located in Shenzhen, China. As this company was a vendor for Blue Robotics, any orders had to be approved and processed through Blue Robotics. Thus, when custom machined components (with accompanying drawings and documentation) were requested, the approval and payment process between Blue Robotics and Kaierwo was delayed by approximately a week after the pricing quotes from Kaierwo were received. This imposed a large delay in our timeline and consequently, changed the order arrival date from April 5th to the end of the day of April 15th. Despite this, the shipment ultimately arrived on the afternoon of April 19th. As a result, full integration, tuning, testing, and results documentation was limited to two days instead of the initially allotted two-week period prior to the project completion deadline. Thus, emergency contingency components, had to be rapidly modified and prototyped to remove the team's dependency on DHL and to allow us to test if the parts did not arrive in time.

Furthermore, in this time, faulty external components (such as the Blue Robotics tether communication, the Arduino shield, and the interface PCA) and complex communication issues arose, which further decreased available time to test with. Ultimately, a last resort, three-day extension was requested and granted to allow for as much integration of test results into this final report and final video as possible.

In future iterations of this course, there are various recommendations that are suggested that may serve to improve MECH 400 at UVic. First, as the course now has full access to Monday.com, it is highly recommended that all future teams use this platform as their project management system, as it allows for the easy and intuitive development of a clear and functional project management system. Additionally, by incorporating a segment into the project timeline in weekly TA design meetings, it would help to ensure all teams actively use and update their project management systems, rather than simply create it for project requirements and visual purposes. This will also allow the course administration to monitor team progress in a more transparent and in-depth manner should they choose to do so.

Moreover, it is recommended that all teams use GrabCAD, a free CAD management system and open-source model repository, for their CAD management, as updates can be easily and quickly uploaded and downloaded to avoid conflicts and multiple floating CAD files. This allowed us to essentially create a PLM-like system to allow for multiple users to create, modify, and use official parts, drawings, and assemblies. This system also allowed for seamless revision control, ensuring all drawings, components, and assemblies used were up to date. Moreover, it would be beneficial if MECH 400 took place over an eight-month period. Given what was achieved by our group, we would have loved to see what we could have achieved in eight-months. This duration would have also allowed for a reduced workload over a single semester, reducing the time constraint and inrush of ordering components from external sources and manufacturing components (internal and external).

Finally, as a five-member group, an overall average of ~370 hours (~28.5 hours per week) per member was carried out. As each member was concurrently in additional courses, on a co-op work term, or both, a focus on each other's mental health was of utmost importance, while ensuring work was still progressing at a steady pace. Thus, it is suggested that frequent reminders to take breaks, sleep, eat, or to tend to other commitments/responsibilities be sent by course administration to help students avoid mental burnout.

3. Technical Design

Disclosed below are the design processes and technical developments over the duration of the project and by extension, the BlueFish's design cycle.

3.1. Hydrodynamics

The hydrodynamic profile and control surfaces pertain to the exterior profile of the Bluefish, including the nosecone, fishtail, dynamic control surface, keel, and static fins. This "sub-system" is also known as the BlueFry.

3.1.1. Hydrodynamic Profiles

First, several variations were considered for each component and were simulated in ANSYS Fluent (CFD) to determine the best overall profiles to incorporate into the designs of the hydrodynamic features. A full test report can be found in Appendix D.

First, the nosecone, four profiles were simulated: conic, dome/hemispherical, full parabola, and ½ power series. Of these profiles, the full parabolic profile yielded the best relative results (see Table 1). It should be noted that the fishtail features the same curve profile as the nosecone. Next, for the static fins and control surfaces, different leading and trailing edge profiles were simulated. Overall, the filleted leading-edge and outward fillet trailing-edge profiles yielded the best relative results. Tabulated results can be found in Table 1. It should be noted that all control surfaces have a flat, ¼ inch thick cross-sectional profile. Although this is not optimal, all BlueFry prototype fins are manufactured out of laser-cut plywood and was deemed a sufficiently close approximation.

Table 1: Nosecone, Leading/Trailing Edge Drag Coefficients and Force, CFD Analysis Results.

Profile Geometry	Drag Coefficient	Drag Force [N]
Nosecone Profiles		
Blunted Full Parabola	1.667	1.021
$x^{1/2}$ Power	1.710	1.047
Blunted Conic	1.794	1.099
Dome/Hemisphere	1.757	1.076
Leading Edge Profiles		
Straight	1.701	1.042
Chamfer	1.296	0.794
Fillet	1.258	0.771
Trailing Edge Profiles		
Straight	1.701	1.042
Outward Chamfer	1.293	0.792
Inward Chamfer	1.444	0.884
Outward Fillet	1.238	0.759
Inward Fillet	1.416	0.867

From this data, the primary static control surface, the fins, were implemented to help reduce the vessel's tendency to roll, to allow for smooth diving/surfacing, and to counteract the vertical normal forces acting on the vessel relative to the direction of travel. Thus, fins featuring a filleted leading edge and an inward fillet trailing edge were designed, manufactured, and implemented on the BlueFry I and BlueFry II prototypes. It should be noted that the inward fillet trailing edge, not the outward fillet, was used to allow for more space behind the wings for future features to be more easily implemented without the need to redesign the fins.

Next, "dynamic" control surfaces were implemented to allow for pitch and roll control. As there are various methods and designs to allow for such control, many designs were considered. The best method was deemed to be a horizontal aileron design, like the control surfaces found on conventional airplanes, where the ailerons help to control pitch and roll. When the ailerons are moved in the same direction, a change in pitch occurs. This motion would be the primary mechanism for altitude control. Similarly, when the ailerons are moved in opposing directions, a rolling action occurs. This motion would be the primary mechanism for roll control. It should be noted controlling the yaw of the BlueFish is not necessary (see section 4). Therefore, a modified version of the static control fins was designed to feature rear foils that could be offset by angles of +/- 20° or +/- 45° using aluminum angle brackets. These modified fins were manufactured and implemented on the BlueFry I prototype (see Figure 4).

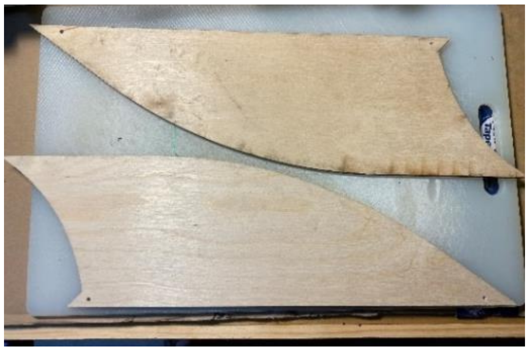
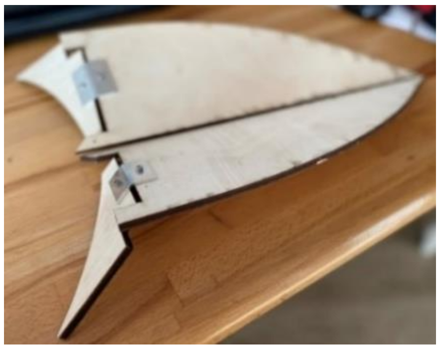



Figure 4: Static Fins (Left) and "Dynamic" Control Surfaces [Modified Static Fins] with 45° Foils (Right), BlueFry I and II.

Furthermore, a secondary static control surface, the keel, was implemented to help reduce the vessel's tendency to roll and to counteract the horizontal normal forces acting on the vessel relative to the direction of travel. Additionally, different keel types have various properties that are important to consider when choosing a type of keel to design (see Table 2). Thus, two keel types, the full and bilge keel, were designed, manufactured, and implemented on the BlueFry I prototype (see Figure 5).

Table 2: Keel Benefit and Deficit Comparison

Keel Type	Benefits	Deficits
Full Keel	• High stability	• High overall drag • Decreased maneuverability • Cannot be beached (sit flat)
Bilge (Twin) Keel	• Low overall drag • High maneuverability • Can be beached (sit flat)	• Reduced stability

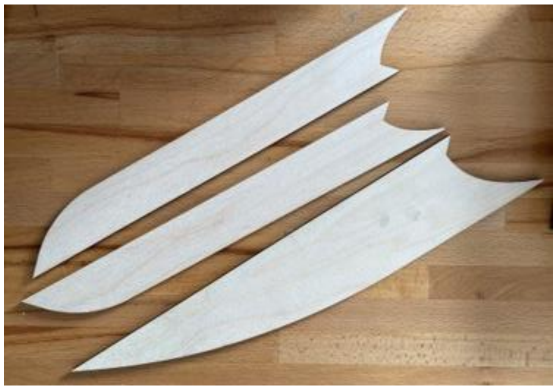


Figure 5: Full Keel (Bottom) and Bilge Keel (Top), Laser Cut Keels, BlueFry I and II.

3.1.2. BlueFry I

The culmination of the CFD analysis, requirements, specifications, and available custom and COTS components is represented by the BlueFry I, the first prototype of the BlueFish (see Figure 6). This first prototype served to provide a baseline for the pitch/roll stability, a baseline for its diving characteristics, justification for the current design that all future prototypes will be based on, and to help identify any unconsidered aspects of the overall hydrodynamic and enclosure design. Using the BlueFry I prototype, the first hydrodynamic test was conducted. A full test report can be found in Appendix E.

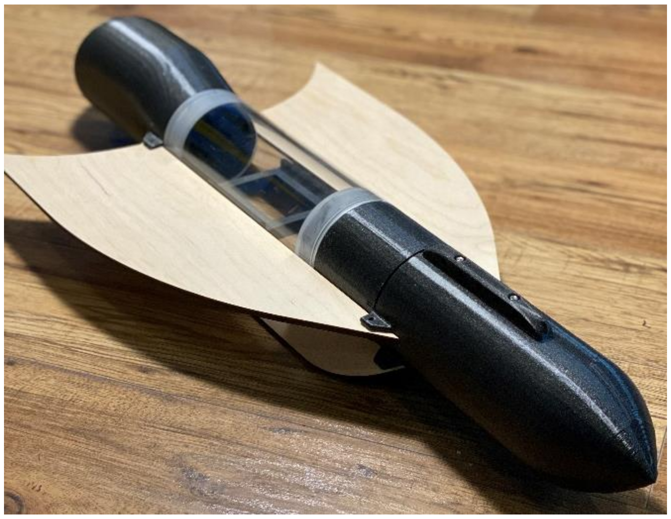


Figure 6: BlueFry I Prototype.

Overall, it was determined that the prototype was sufficiently stable and moved through the water very well, diving and leveling in a smooth and controlled manner. It was also determined that the bilge keel provided the best combination of roll stability and maneuverability when travelling along a straight and/or curved path. Additionally, it was determined that the "dynamic" rear foils were not able to induce diving at their maximum angle and at the maximum forward speed. The potential causes for this result included: a) too much water flowing through the gap between the foil and the fin body, b) the foil was too small to provide enough lift, or c) the static fins were too large, resisting change in pitch as they skimmed the surface.

3.1.3. NACA 0012 Hydrofoil

Consequently, the shortcomings of the BlueFry I provided means to find an alternative solution to induce diving from the surface at the maximum tow speed. Thus, the proposed solution was the use of NACA airfoil profiles, which were to be used as a hydrofoil profile [1]. After researching variations of NACA airfoil profiles, the symmetrical NACA 0012 airfoil profile was chosen (with a 90 mm chord length) and subsequently, a 2D model was created and simulated in ANSYS Fluent (see Figure 7). A transient solution with the foil rotating from a 0° to ~50° angle of attack was created and the stall angle, 2D coefficient(s) of lift, and the 2D coefficient(s) of drag were determined (see Table 3). A full test report can be found in Appendix F.

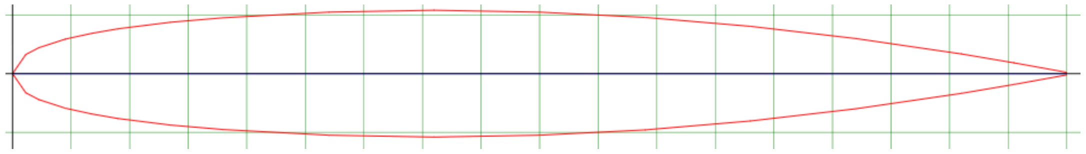


Figure 7: NACA 0012 Airfoil Profile with a 90 mm Chord.

Table 3: NACA 0012 Hydrofoil (with a 90 mm chord) CFD Analysis Results.

Parameter	Value	Units
Stall Angle	+/- ~18	Degrees, °
2D Coefficient of Lift, C_l	At ~18°: 1.25	-
2D Coefficient of Drag, C_d	At ~18°: 0.04	-
Pitching Moment Coefficient, C_M	0.008	-

3.1.4. BlueFry II

Two 3D printed NACA 0012 hydrofoils with a 90 mm chord were manufactured and mounted to the rear of the vessel using dowels. Using the results from the NACA 0012 Airfoil CFD test (see Table 3), the hydrofoils were allowed rotate to angles of 0°, +/- ~18°, and +/- ~45°, representing zero angle, stall angles, and an angle beyond stall, respectively. This was achieved using various thru holes drilled into the REV02 fishtail through which two wooden dowels could pass through (see Figure 8). The lower placement of the NACA hydrofoil should also be noted, for when the foil's angle of attack is greater than its stall angle, the drag alone could still induce a moment to help control pitch. This was also to ensure that at the surface, the NACA hydrofoil would be guaranteed to be submerged. This feature, amongst additional flooding/drainage holes, formed the BlueFry II, a modified version of BlueFry I.

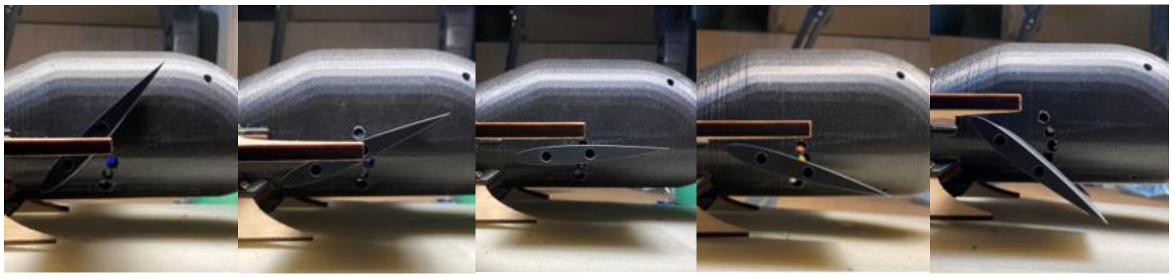


Figure 8: NACA 0012 Hydrofoil at Discrete Angles, (Left to Right: 45°, 18°, 0°, -18°, & -45°), BlueFry II.

Using this prototype, a test was conducted to determine if the new hydrofoils would be an adequate solution to the problem discovered during Test No. 2: Hydrodynamic Test, which was the inability to dive from the surface. Overall, it was determined that the prototype was capable of diving from the surface of the water, while maintaining the stability and diving/leveling characteristics of the BlueFry I. A full test report can be found in Appendix G.

3.1.5. Depth Control Mechanism

To address the shortcomings of the BlueFry I and continued developments from BlueFry II, the depth control mechanism was developed to automate and motorize the roll and pitch control of the BlueFish. Thus, the loading conditions of the NACA 0012 hydrofoils were first calculated from the 2D loading coefficients determined from the prior CFD analysis (see Table 3 and Table 4). Sample calculations of the aforementioned loading conditions can also be found in Appendix P. These were used to verify that the servo motors in the depth control mechanism would be able to provide sufficient torque to rotate the hydrofoils as required (see Appendix S for servo motor datasheet).

Table 4: NACA 0012 Hydrofoil (with a 90 mm chord) Loading Conditions (Per Hydrofoil).

Parameter	Value	Units
Stall Angle	+/- ~18	Degrees, °
3D Coefficient of Lift, C_L	At ~18°: 0.1531	-
3D Coefficient of Drag Induced, $C_{D,i}$	At ~18°: 0.006	-
3D Coefficient of Drag (Skin & Form Friction), $C_{D,0}$	At ~0°: 0.01	-
3D Coefficient of Total Drag, $C_{D,T}$	0.016	-
Dynamic Pressure, q	498.5	Pa
Lift Force, F_L	0.790	N
Drag Force, F_D	0.082	N
Pitching Moment, M	0.041 (5.845)	Nm (oz-in)
Moment due to Gravity, M_g	0.013 (1.791)	Nm (oz-in)
Maximum Moment (Pitch Up)	0.054 (7.636)	Nm (oz-in)
Maximum Torque from Servo Motor	3.981 (141.612)	Nm (oz-in)
Safety Factor, SF	18.545	-

Additionally, raked wingtips were added to the far edge of the NACA 0012 hydrofoils (see Figure 10). When a pressure differential between the fluid beneath the foil and the fluid above the foil (working principle of airfoils/hydrofoils) is present, a by-product is a region of swirling fluid known as vortices that reduce the effective area of the foil (see Figure 9). Thus, to reduce this effect, raked wingtips, one option amongst many others, was used for this purpose.

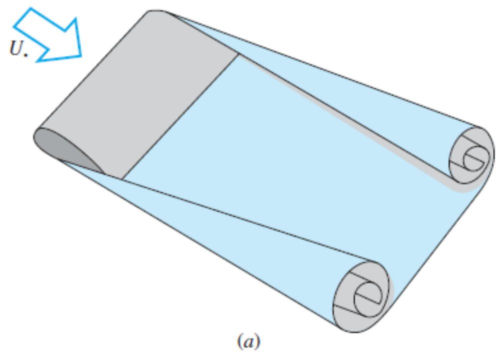


Figure 9: Lifting-Line Theory for a Finite Wing, Trailing-Vortex System behind a Wing [2].

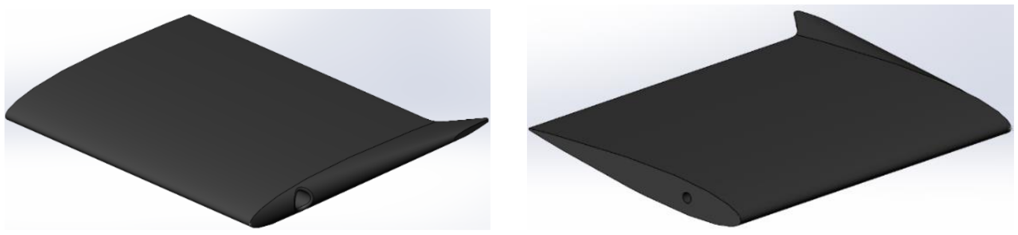


Figure 10: NACA 0012 Hydrofoils, Depth Control Mechanism.

To mount the 3D printed hydrofoils to the servos, a machined aluminum mounting bracket was implemented (see Figure 11). Using two countersunk wood screws, the bracket was fastened to the foil itself. This allowed for the edge of the raked wingtip to be unencumbered by holes for fastening, unlike the screw affixing all the components axially to the servo. This method was used to avoid disturbing the flow around the critical section of the raked wingtip, the tip itself. Additionally, in combination with the axial screw fastened to the servo, the bracket itself is used as a split clamp to grip onto the twenty-five-tooth servo motor spline. Furthermore, as both assemblies are symmetrical, the opposing side of the depth control mechanism uses the same assembly, but mirrored upside down.

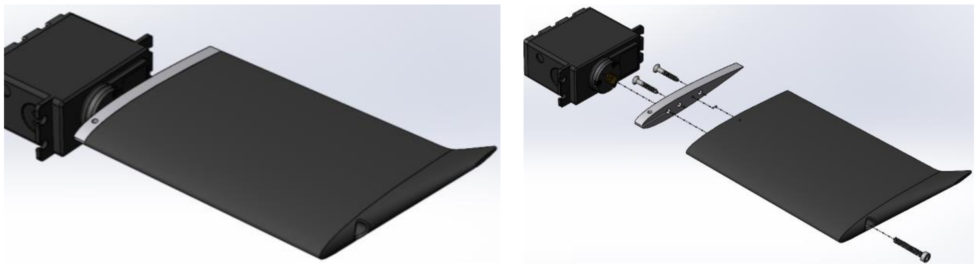


Figure 11: NACA 0012 Hydrofoil Mounting Assembly, Depth Control Mechanism.

3.1.6. Rear Assembly

To affix the depth control mechanism to the main enclosure and to house the servos that control the pitch of the NACA 0012 hydrofoils, the 3D printed fishtail allows for the servos to be inserted from the exterior, allowing for easier assembly and part replacements if necessary (see Figure 12). Featuring two slots to seat the servo motors, four screws and nylon lock washers are used to fasten the servo motors to the fishtail itself. Additionally, as the servos are waterproof, the rear fishtail is not watertight and to allow for consistent and easy drainage, four large drainage holes were built into the fishtail. This feature also allows easy access to the nylon lock washers on the far side of the servo motors nearest the rear nosecone. It should also be noted that the PING sonar transducer is located within the fishtail enclosure to allow for more space within the nosecone enclosure (see section 3.5.4.1). Using a custom machined mounting bracket, the Ping sonar is mounting using its integrated threaded mounting features (see Figure 13).

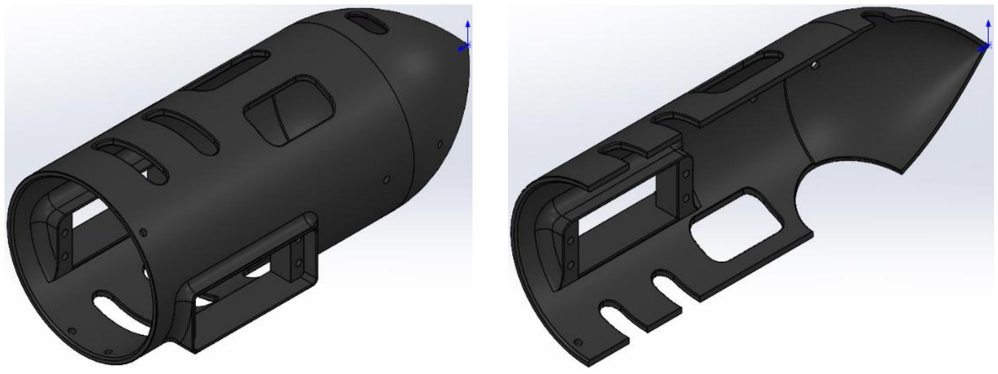


Figure 12: Empty Enclosure (Left) and Cross-Sectional View (Right), Fishtail Enclosure.

To connect the rear assembly to the main enclosure, the rear endcap with watertight penetrators (see section 3.2.1) is used (see Figure 13). As the fishtail fastens to the rear endcap rather than the main enclosure itself, the rear assembly (excluding the endcap) can be removed for maintenance, troubleshooting, or part replacements without need to break the seal of the main enclosure.

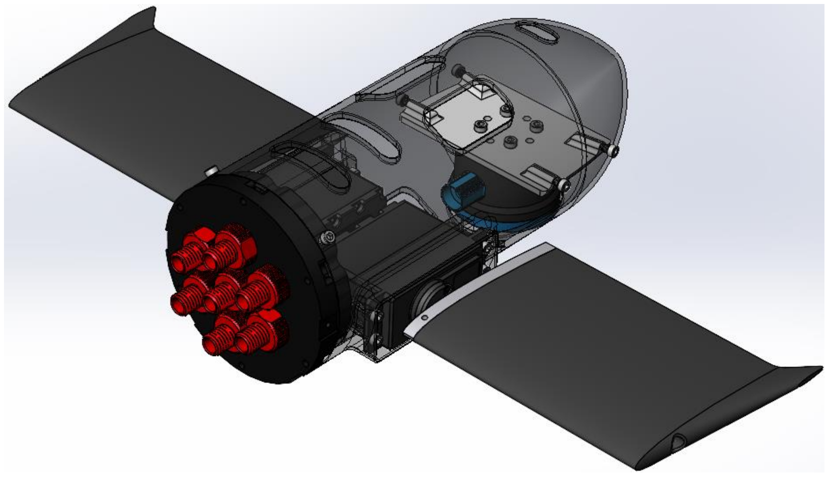


Figure 13: Full Rear Assembly, Depth Control Mechanism.

3.1.7. Front Assembly

Using the same parabolic profile as the BlueFry I and II, the nosecone was designed to house the camera and Lumen lighting modules (see section 3.4.2). The bottom half of the nosecone features two large cut-outs for the camera housing and Lumen lighting module (see Figure 14 and Figure 15).

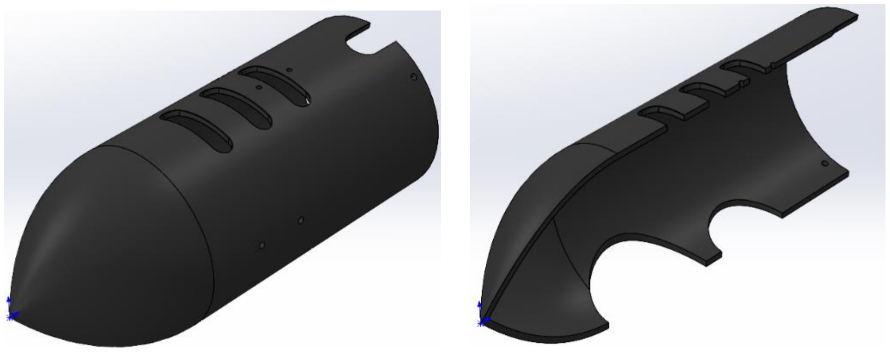


Figure 14: Full View (Left) and Cross-Sectional View (Right), Nosecone Enclosure.

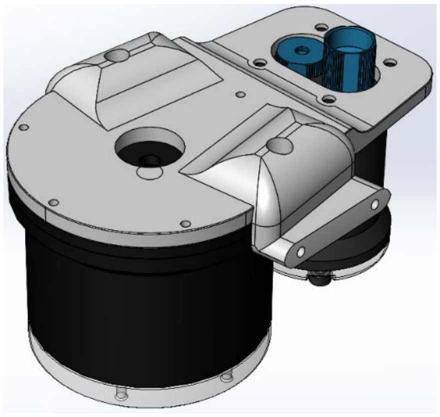


Figure 15: Custom Mounting Bracket, Lumen, and Camera Module, Front Assembly.

Moreover, affixed to the top half of the nosecone assembly, is the tow rack and tow thimble for towing and data transmission purposes (see Figure 16 and section 3.3). Furthermore, like the rear assembly, the front assembly is connected to the main enclosure using the front endcap with watertight penetrators (see section 3.2.1 and Figure 16). As the nosecone halves fasten to the front endcap rather than the main enclosure itself, the front assembly (excluding the front endcap) can be removed for maintenance, troubleshooting, or part replacements without need to break the watertight seal of the main enclosure.

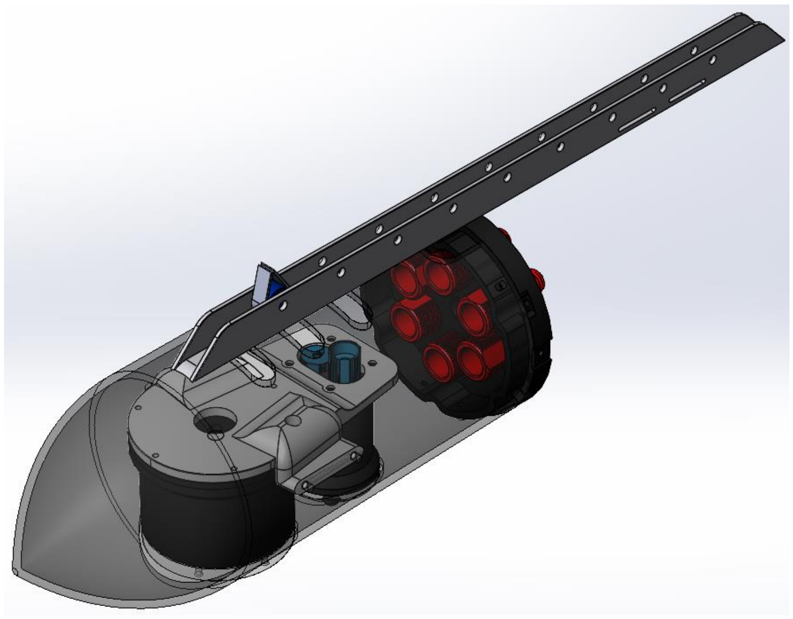


Figure 16: Full Front Assembly.

3.1.8. Static Hydrodynamic Profiles II

Once again using the data gained from the initial hydrodynamic profile CFD analysis (section 3.1.1 and Appendix D), the nosecone, fishtail, bilge keel, and static fins featured the same primary profiles and/or features. An exception to this was static fins and the bilge keel, which for BlueFry III, featured outward fillets for both their leading and trailing edges to reduce the drag caused by the fins and keel. Additionally, the overall lengths of both aforementioned features were increased to accommodate the lengthened circular enclosure for the final prototype iteration.

Additionally, three main features were added to the static fins and keel to allow for a quick-attach method during assembly (see Figure 17 and Figure 18). The first feature was a "hooked" open slot at the front of the static fin and keel that hooks into a mating feature, known as the "wing dovetail retainer" on the front endcap. The second feature is the open clearance slot and M3 clearance hole at the rear of both components. This feature, known as the "wing slider retainer," allows for the static fin to be snapped into place along a J-shaped closed slot affixed to the rear endcap. The third feature is the large hole at the far edge of the static fin. This feature allows the users' fingers to gain leverage and snap the fin into place during assembly more easily. It should also be noted that the span or width of the static fins were increased to protect the larger span of the NACA 0012 hydrofoils featured on the BlueFry III.

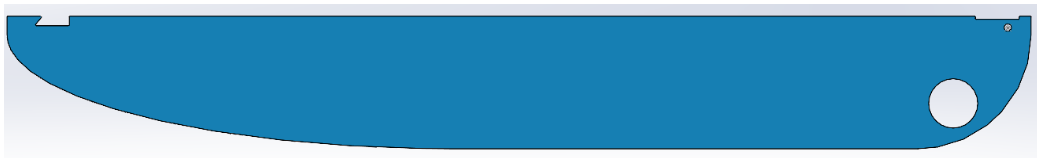


Figure 17: Bilge Keel, BlueFry III.

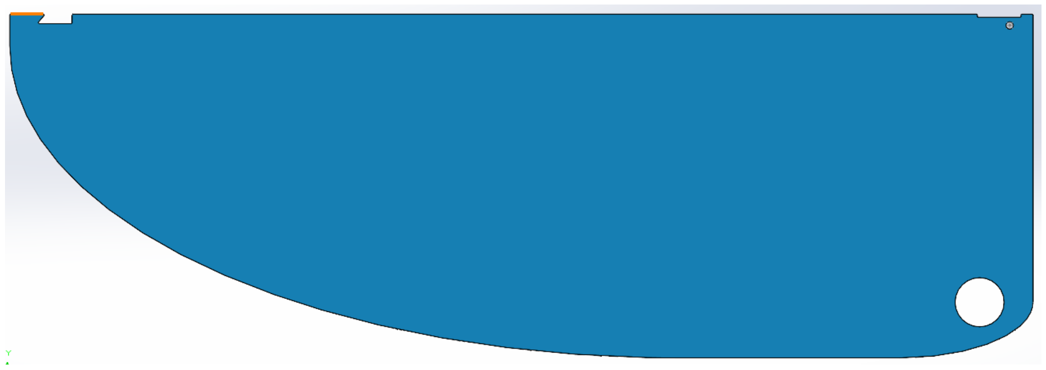


Figure 18: Static Fin, BlueFry III.

3.1.9. BlueFry III

Using the information gathered and lessons learned from the BlueFry I and BlueFry II, the final BlueFry III prototype was developed and represents the pinnacle of the hydrodynamic work over the course of the project and product design cycle. Together the front and rear assemblies, static fins, bilge keel, and main enclosure (see section 3.2) form the final hydrodynamic prototype, the BlueFry III (see Figure 19). This prototype was subjected to a successful waterproof test prior to the integration of the FishGuts sub-system to ensure that there were no manufacturing or design defects. During the aforementioned test, it was noted that during the BlueFry's unassisted descent, its path followed that of a "falling leaf," gliding forwards and backwards in an oscillatory motion (see Figure 20). This was likely due to the static fins large surface area and as designed, helped to resist rapid changes in the vertical axis while aiding in a smooth gliding motion. A full test report can be found in Appendix H.

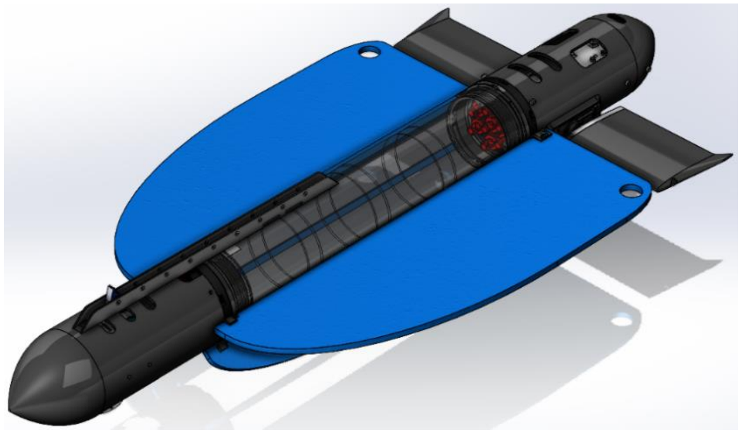


Figure 19: Full Assembly, BlueFry III.

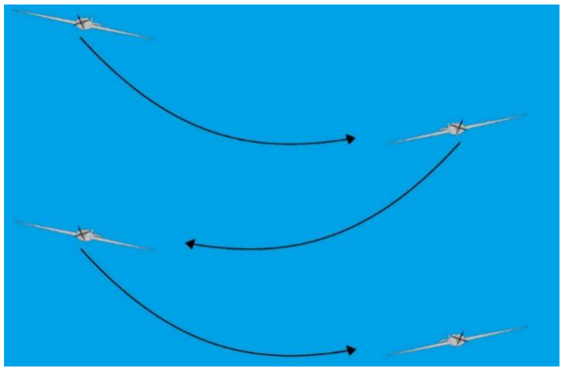


Figure 20: "Falling Leaf" Motion [3] .

3.2. Main Enclosure

The enclosure selected for the BlueFry III/BlueFish I is a prototype Blue Robotics, 400 mm long, 3" diameter, cast acrylic enclosure with machined mating O-ring surfaces. The enclosure mates up against a prototype Blue Robotics O-ring flange, where two, 3" radial O-rings provide a tested seal for all of the electronics. The endcaps, however, were modified such that the endcaps can still retain the bolt pattern that seals an axial O-ring to the locking flange seal, while mounting the E-tray, the fish tail and nosecone, and any external features such as sidescan sonar transducers or future modifications (see Figure 21).

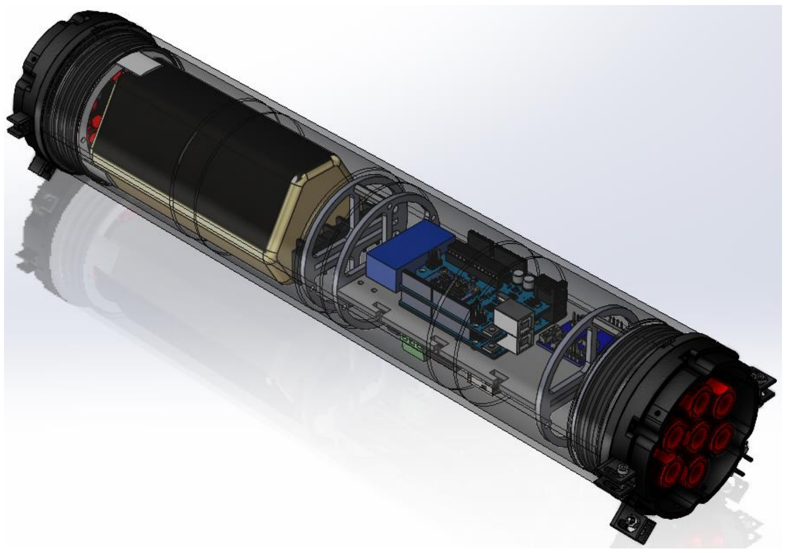


Figure 21: Full Assembly, Main Enclosure.

3.2.1. Locking Flange Seals & End Caps

To create a waterproof enclosure, pre-tested, Blue Robotics O-ring flanges and O-rings were used to create a watertight seal between the Blue Robotics cast acrylic enclosure and the flanges (see Figure 22). Then, custom endcaps were pressed up and sealed against the face O-ring to complete the watertight enclosure (see Figure 21). As the endcap was a modified version of a pre-existing Blue Robotics component, critical O-ring mating features were maintained. It should also be noted that O-ring verification calculations can be found in Appendix Q. Furthermore, to route cables into and out of the watertight enclosure, Blue Robotics penetrators were used (see Figure 23). Like the O-ring flanges, as these components and their mating components (the O-ring flange) were COTS Blue Robotics components, no additional design validation was conducted on these components.

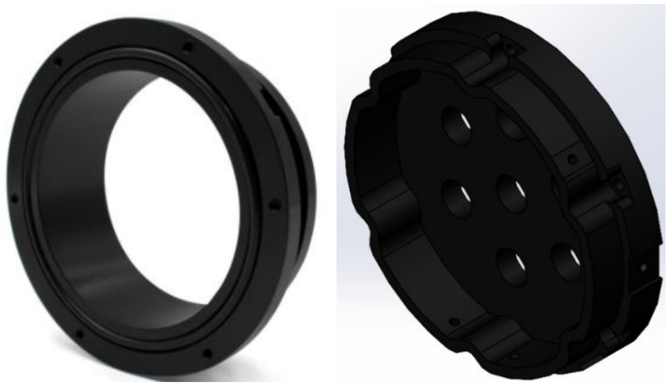


Figure 22: Blue Robotics O-Ring Flange [4] (Left) and Custom Endcap (Right).

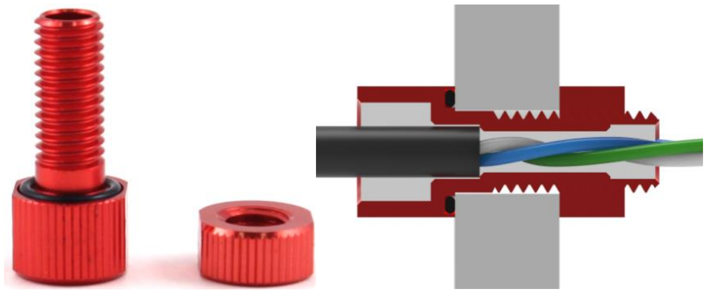


Figure 23: Blue Robotics M10 Cable Penetrator (Left) and Cross-Sectional Diagram (Right) [5].

3.2.2. Static Fin and Keel Mounting Feature

To mount the static fins and bilge keel, mounting features were required. To allow for rapid prototyping, simple mounting features were integrated into the nosecone and fishtail enclosures and used a simple fastener-and-nut system (see Figure 24). However, as learned during the hydrodynamic validation test, this system resulted in a time-consuming and tedious procedure to remove and install the static fins and keel. Thus, affixed to the radial mounting features of the endcaps are the front wing dovetail retainers and the rear wing sider retainers (see Figure 25). As previously mentioned in section 3.1.8, the design allows for the static hydrofoils to be "hooked" to the dovetail retainer and slid backwards (towards the curve of the J-shaped slider retainer) and "snapped" into a locked position, similar to a snap-fit. This reduced the affixing/detaching of the fins from approximately four minutes to under thirty seconds, correlating to an 87.5% reduction in time for affixing/detaching the static fins and keel.

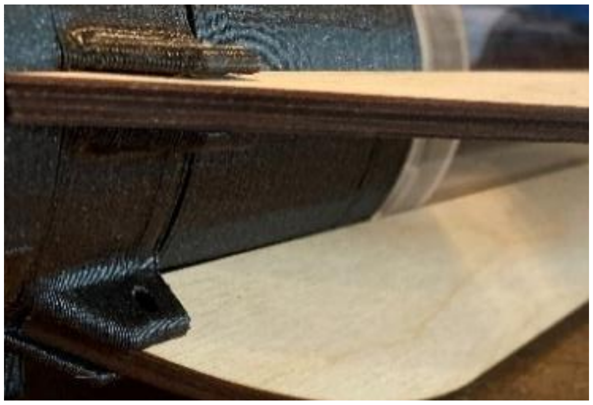


Figure 24: Original Static Fins and Keel Mounting Feature, BlueFry I and II.

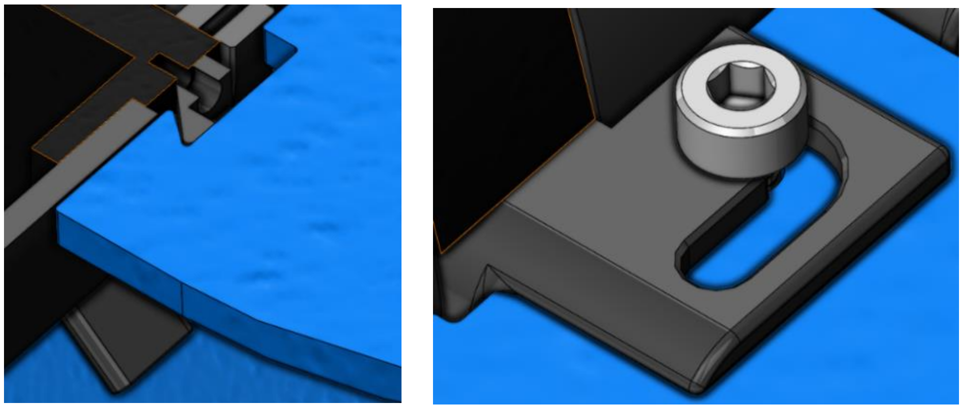


Figure 25: Front Wing Dovetail Retainer (Left) and Rear Wing Slider Retainer (Right) Mounting Features, BlueFry III.

3.2.3. E-Tray

To mount and hold all the electrical components (excluding the Ping sonar transducer, servo motors, lumen, and camera module), a modular and expandable "E-tray" was designed and manufactured out of acetal and press-fit together (see Figure 26). To provide additional structural integrity, aluminum standoffs were used to form a post-and-plate assembly (not all standoffs are shown to display electrical components). This component was designed with ease of assembling in mind, allowing for nearly all the electronics to be assembled and connected outside of the main enclosure, thus when complete, minimal electrical connections would need to be made. Furthermore, this module was designed such that if a component fails or is damaged, a replacement E-tray assembly could be quickly inserted within a matter of minutes. This modularity applies to nearly every component of the BlueFish. It should also be noted that the main components, if not affixed to another electrical component, are fastened to the E-tray using 3D Command Tape strips.

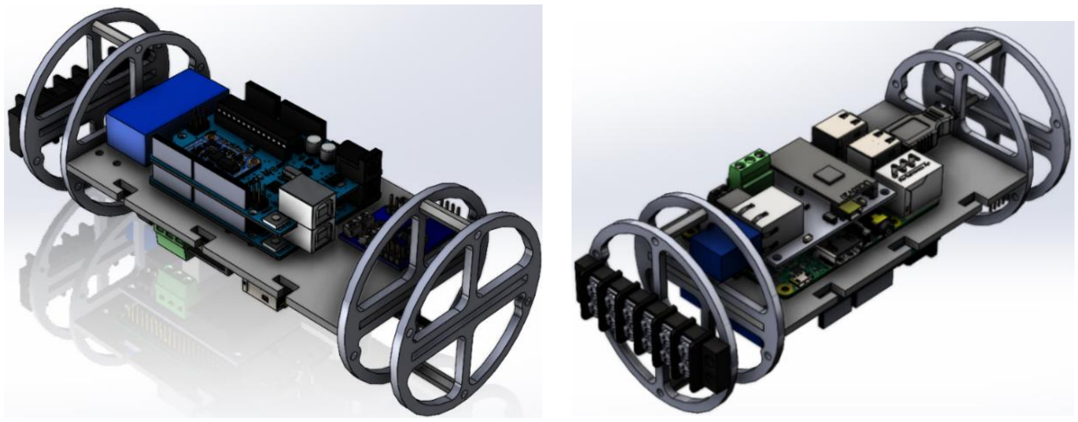


Figure 26: Top Side (Left) and Bottom Side (Right), E-Tray Assembly.

3.2.4. Battery

To power the BlueFish, a 14.8 V, 18 Ah, COTS Blue Robotics Lithium-ion battery pack was selected (see Figure 27). Supplying an approximate seventeen-hour battery life, this well exceeds the ten-hour engineering specification, allowing for an extended single-operation period before the battery would need to be replaced. Moreover, in a similar fashion as the E-tray assembly, the battery pack can easily be removed and replaced. Additionally, as the battery pack is a standard Blue Robotics component, the BlueFish would allow for seamless integration into Blue Robotics' battery power supply system. For battery calculations or battery technical specifications, see Appendix K and Appendix R, respectively.

Figure 27: Blue Robotics Battery Pack (Right) [6].

3.3. Towline

The towline is critical to maintain a real-time connection to the BlueFish while it is collecting data. After discussion with the Blue Robotics, a fixed-length towline length was selected as opposed to a variable length, which is common in some towfish. It should be noted that the tether attachment at the BlueBoat is comprised of a set of anodized aluminum rails affixed to the frame of the BlueBoat, while a swiveling cable clamp allows for control of the tether's length while also providing tether routing to avoid any kinks or sharp bends.

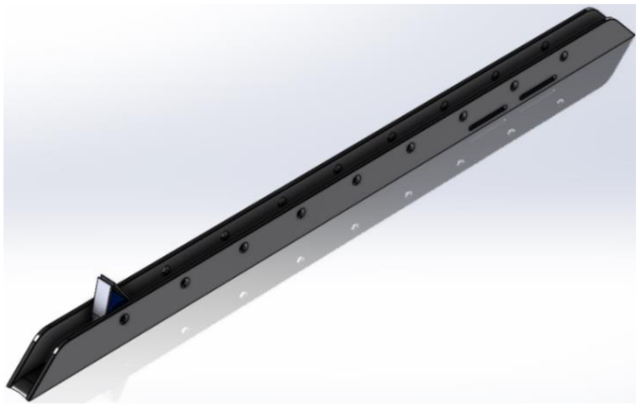


Figure 28: Towline Assembly.

3.3.1. Tow Rack

To affix the tether to the BlueFish, the tow rack acts as the primary structural member to do so. Made of stainless steel, this custom machined component was attached to the nosecone using fasteners, while as a contingency, two slots near the rear end of the tow rack allow for the use of one or more adjustable hose clamps or zip-ties (see Figure 28). Additionally, the tow rack features nine mounting holes to adjust the lateral position of the tow thimble (see section 3.3.2) to rapidly adjust the tether connection point for optimal hydrodynamic performance. It should be noted that the secondary and emergency structural member to maintain a connection to the tether, is the tether's connection to the cable penetrator. Although this feature is not intended to resist tensile loading, it can be used in emergency scenarios for BlueFish retrieval purposes.

3.3.2. Tow Thimble

The primary purpose of the tow thimble is to physically attach the tether to the tow rack. Using a rubber spacer and a shoulder screw, the COTS stainless steel thimble is affixed to the tow rack with the shoulder screw being fastened through one of the tow racks various fastening holes (see Figure 29). A shoulder screw was used to provide smooth rotation of the acetal/Delrin spacer, which sits inside the thimble. This small assembly acts as the primary strain relief for the tether, while also preventing the tether from exceeding its minimum bend radius. The rubber spacer, however, acts as a shock absorber between the tether, the thimble, and the tow rack itself. It should be noted that in the final BlueFish prototype, the tow rack and rubber spacer was not integrated into the final prototype due to various time constraints (refer to section 2). Alternatively, the thimble was connected to the BlueFish via the nosecone drainage holes (see Figure 30).

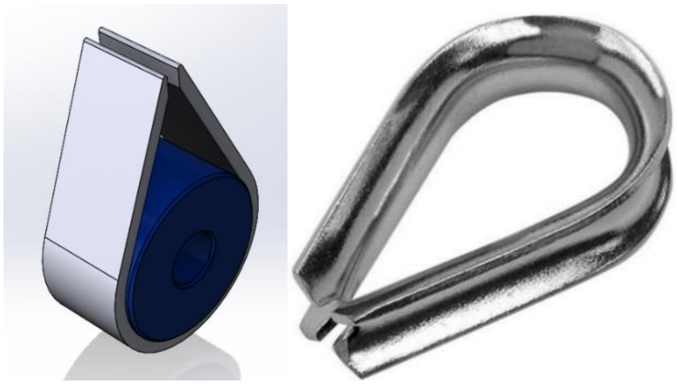


Figure 29: Simplified Model of Thimble with Rubber Spacer (Left) and Actual Rope Thimble (Right) [7].

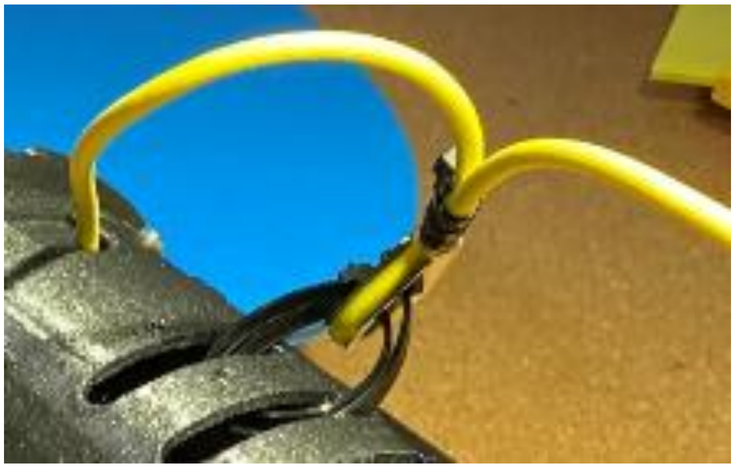


Figure 30: Alternate BlueFish Tether Connection Method.

3.4. Camera and Lighting

To collect the seafloor images, a COTS Blue Robotics camera and lighting module were selected. Although the camera may be sufficient to take seafloor images in shallow waters, in deeper and darker waters, supplementary lighting is required to illuminate the seafloor for imaging. Additionally, as previously mentioned in section 3.1.7, the camera and lighting are mounted in the nosecone enclosure for easy installation, access, and maintenance purposes.

3.4.1. Camera

To collect the images, a COTS Blue Robotics Low-Light HD USB camera module is being used. Featuring a HD 1080P image resolution and low-light imaging capabilities, this camera module was selected (see Figure 31). As Blue Robotics currently uses this camera module on their AUV/ROVs, this camera was deemed sufficient for our purposes. Additionally, by selecting a COTS Blue Robotics component, this allowed for a reduced prototype cost and better integration into Blue Robotics' products. Technical specifications of the camera can be found in Appendix T. However, as the module is not waterproof, a waterproof enclosure had to be designed (see Figure 32). Featuring a custom machined aluminum housing, Blue Robotics penetrator, and flat Blue Robotics acrylic face cover, and an O-ring, this enclosure allows the camera to be housed outside of the watertight main enclosure and prevents and image distortion due to the curved geometry of the main enclosure.

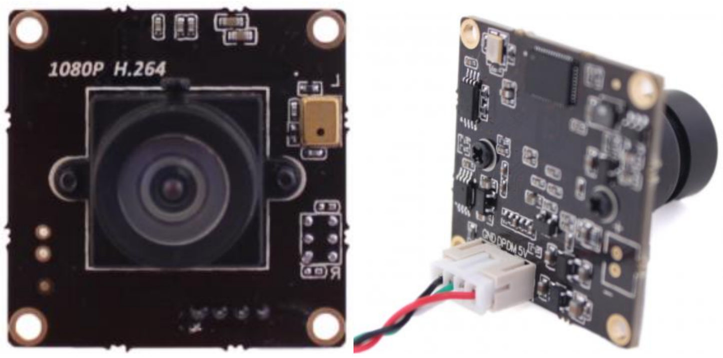


Figure 31: Blue Robotics Low-Light HD USB Camera [8].

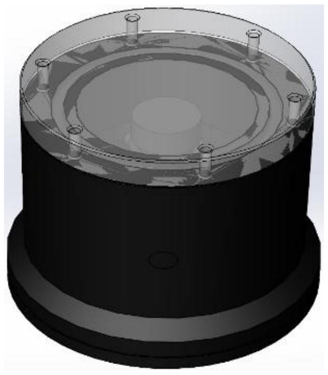


Figure 32: Custom Camera Waterproof Housing.

3.4.2. Lighting

In low-light conditions, a waterproof, a COTS Blue Robotics Lumen R2 Subsea Light is used to illuminate the seafloor such that camera can take good quality and detailed images (see Figure 33). Outputting a maximum of 1500 lumens of cool white (6200 K) light in a 135° wide beam, the brightness can be adjusted with a PWM input as required to yield the best lighting conditions for imaging. Technical specifications can be found in Appendix U.

Figure 33: BlueRobotics Lumen R2 Subsea Light for ROV/AUV [9].

3.4.3. Image Post-Processing

Once the photographing functionality of the BlueFish had been integrated, tested, and finalized, the images were going to be collected and enumerated to later produce a photomosaic of the seafloor. The anticipated method of collating the images was to use Adobe Photoshop, which is capable of automatically recognizing overlap in images to produce a high-resolution photomosaic. This method was never attempted, yet complications were anticipated, such as the ability of Photoshop to recognize overlap despite the fish-eye effect produced by the USB camera. Alternatives were discussed, such as the selection of a different USB camera that did not produce images with the fish-eye effect and the exploration of a different photomosaic-producing software.

3.5. Control System

The control system of the BlueFish encompasses the electronics (FishGuts), along with the firmware and software (FishBrains). It is responsible for communication between the BlueFish and the user, data acquisition and storage, and autonomous control of depth or altitude, pitch, and roll.

3.5.1. Communication and Data Storage

The Raspberry Pi 3 B+, a small single board computer, was selected to control communications, data acquisition, and storage (see Figure 34). It was chosen for its compact size, versatility, compatibility with Blue Robotics architecture; Raspberry Pi 3s are already used in the BlueROV, BlueROV2, and BlueBoat. All data can be stored locally via CSV files on a micro-SD card, and with the use of a FXTI and Fathom Slim Tether from Blue Robotics (see Figure 35), the Raspberry Pi can be accessed with the user's laptop through either the top side BlueBoat that produces a long-range Wi-Fi signal, or a FXTI box on a topside boat. An added benefit to using a Raspberry Pi is that firmware on the MCU can be updated remotely without having to retrieve or disassemble the BlueFish, allowing for dynamic tuning, custom commands, and operational mode control.

Figure 34: Raspberry Pi 3 B+ [10].

Figure 35: Blue Robotics Fathom Slim Tether [11] (Left) and Fathom-X Tether Interface [FXTI] [12] (Right).

3.5.2. Microcontroller

Raspberry Pi computers are not optimized for control systems as they lack the ability to read or write analog signals typically required for motion control or sensor reading, while also adding the complexity of an operating system. For this reason, inclusion of a microcontroller was required for the control system of the BlueFish. Selection of the MCU was based on preliminary research and the decision came down to using either an Arduino or a Pixhawk Flight Controller (see Figure 36).

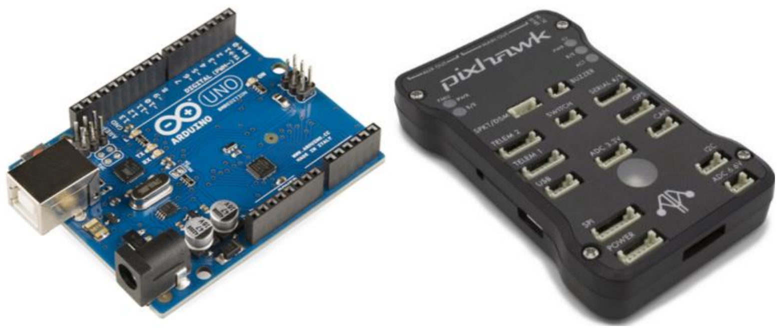


Figure 36: Arduino UNO MCU [13] (Left) and Pixhawk Flight Controller [14] (Right).

Arduino produces a variety of robust MCU boards that are commonly used to introduce hobbyists to electronics and coding. As such, they are one of the most available and well documented MCU's with abundant open-source programming libraries. Based on the required I/O of the BlueFish and space considerations of the enclosure, the Arduino UNO R3 was determined to be the best model as it was small enough to fit inside the 3-inch enclosure and provided sufficient digital, serial, and I2C ports. Since the UNO is similar in many ways to the MCU's used in the University of Victoria's mechatronics course, it also has a distinct advantage over the Pixhawk when it comes programming familiarity. Arduino uses a language that is essentially C++ with additional methods and functions which reduced the teams required learning curve. However, using the

Arduino UNO meant that everything must be built and coded from the ground up, including all the control algorithms, analog-to-digital conversions, and any live telemetry functionality.

On the contrary, the Pixhawk had the advantage of being built specifically for drones, AUVs, and ROVs. The Pixhawk is intended to be used with ArduPilot, an open-source software with specific variants (ArduSub, ArduPlane, etc.) for different vehicle architectures in conjunction with GCS like QGroundControl. This means that it has pre-programmed, live telemetry functionality and control algorithms. They also integrate seamlessly with Blue Robotics' architecture as the Pixhawk is used on the BlueROV, BlueROV2, and BlueBoat. However, given the BlueFish's unique control layout, the team would have needed to learn a messaging protocol named MAVLink to make large changes to the ArduPlane build.

After consultation with the client, it was recommended that within the scope of this project both controllers be used in the BlueFish since the final product will likely rely exclusively on the Pixhawk to maintain continuity with the Blue Robotics existing products. Our prototype initially planned to utilize the Arduino UNO for controlling the BlueFish while the Pixhawk provided integrated live telemetry. Opting to use the Arduino for control instead of the Pixhawk eliminated the steep learning curve necessary for the team to learn MAVLink and create a custom build of ArduPlane for the Pixhawk.

Once CAD mock-ups of the electronics tray were completed with all the required components, it was determined that the Pixhawk could not be included due to the space limitations within the enclosure. Using a larger diameter enclosure could accommodate the Pixhawk but due to project timeline restrictions this change could not be implemented. Exclusion of the Pixhawk from the control system removed the built-in ability to provide live telemetry data plotting and battery power sensing through the QGroundControl software. To provide this functionality without the Pixhawk, changes were made to the mechatronic system wiring, Raspberry Pi firmware, and Arduino firmware. The Raspberry Pi firmware was developed to include a GUI with live data plotting, while the Arduino firmware was changed to include ADCs on analog ports connected to the battery power sense module. These changes are reflected in the variations between FishGuts I and FishGuts II wiring shown in Appendix L to Appendix O.

3.5.3. Inputs and Outputs

In parallel with selecting a MCU, all the control system's inputs and outputs were identified based on the engineering requirements, along with specific components and their communication protocols. This is summarized in Table 5. To minimize cost, Blue Robotics components were used as much as possible.

Table 5: Table of Required Inputs and Outputs.

Input/Output	Parameter	Device	Signal/Protocol
Input	Leak Detection	BR Leak Detector	Digital
Input	Depth	BR Bar30 Pressure and Temperature Sensor	I^2C
Input	Temperature	BR Bar30 Pressure and Temperature Sensor	I^2C
Input	Pitch	BNO055 9-DOF IMU	I^2C
Input	Roll	BNO055 9-DOF IMU	I^2C
Input	Altitude	BR Ping Sonar Sensor	UART
Input	Camera	RPi USB Camera	USB
Input	Target Depth	Fathom X Tether Interface	Ethernet
Input	Target Altitude	Fathom X Tether Interface	Ethernet
Output	Telemetry	Fathom X Tether Interface	Ethernet
Output	Light	BR Lumens Lights	PWM
Output	System Warnings	Blue LED lights	Digital
Output	Actuator 1	BR Underwater Servo Motors	PWM
Output	Actuator 2	BR Underwater Servo Motors	PWM

3.5.4. Sensors

3.5.4.1. Ping Sonar Altimeter and Echosounder

The Blue Robotics Ping Sonar Altimeter and Echosounder provides the BlueFish with the ability to determine its height above the sea floor. The 115 kHz, 30° single-beam echosounder can detect the distance to the seafloor up to 30 m away underwater (see Figure 37) [15]. Due to the use of a simulated "software serial" port on the Arduino, the ping communicates at a maximum baud rate of 9600 bps.

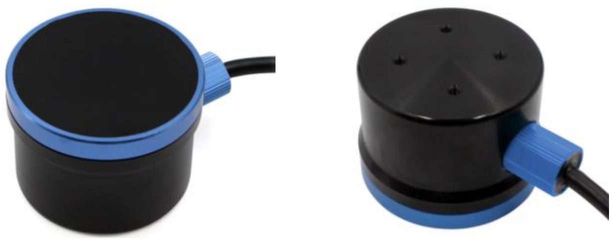


Figure 37: BlueRobotics Ping Sonar Altimeter and Echosounder [15].

As the soundwaves are emitted from the transducer, the Ping awaits the return of the "echo." Using the working principle of sonar, the time elapsed between beam emission and collection is used to calculate the altitude above the seafloor and allows for constant measurements to ensure the BlueFish does not hit the seafloor or any obtrusions from it. Waterproof up to 300 m, as mentioned in section 3.1.7, the sonar is mounted in the non-watertight rear assembly inside the fishtail enclosure. See Appendix V for full technical details.

3.5.4.2. Bar30 Pressure Sensor

The Blue Robotics Bar30 pressure sensor provides the BlueFish with the ability to determine its depth underwater, while also providing temperature measurements. The pressure transducer can measure pressures up to 30 bar (300 m depth) while providing a depth resolution of 2 mm [16]. The temperature sensor included in the Bar30 provides an accuracy in readings of ±1 °C. Although the Bar30 can be powered using 5 V, it requires a 3.3 V I^{2}C communication, necessitating the inclusion of a 3.3 V to 5 V logic level converter between the Arduino and Bar30 (see Figure 38).

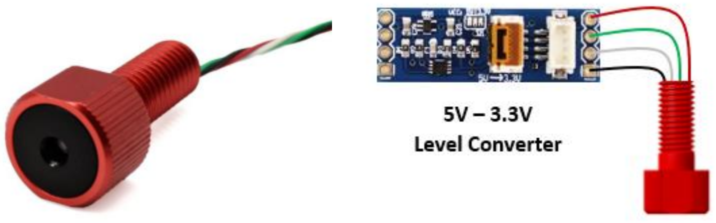


Figure 38. Blue Robotics Bar30 Pressure and Temperature Sensor [16].

The pressure readings from the Bar30 are used to calculate the depth underwater based on the density of the water (fresh or salt) which allow the BlueFish to control its depth and ensure it does not exceed its specified maximum depth. This functionality also allows the BlueFish to determine when it has reached the surface. The Bar30 is waterproof up to 300 m and is mounted in the non-watertight rear assembly inside the fishtail enclosure.

3.5.4.3. BNO055 9-DOF IMU Sensor

The Adafruit BNO055 9-DOF IMU sensor provides the BlueFish with the ability to determine its absolute orientation through use of an internal gyroscope, accelerometer, and magnetometer. The BNO055 converts readings from the internal sensors into a stable absolute orientation in 3D space (based on a 360° sphere) using fusion algorithms [17]. The BNO055 is a 3.3 V device and requires 3.3 V I^{2}C communication. Due to the inclusion of a built-in voltage regulator and logic level shifter on the breakout board (see Figure 39), the BNO055 can also be powered using 5 V with 5 V I^{2}C logic.

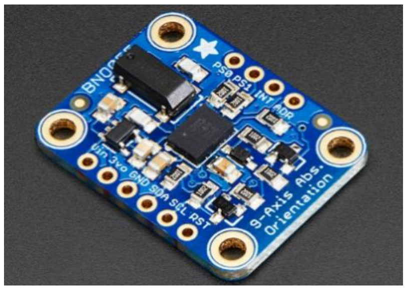


Figure 39. Adafruit BNO055 9-DOF IMU Sensor [17].

The readings from the BNO055 are used to determine its orientation in space and allow the BlueFish to maintain stability through control of its roll, pitch, and yaw, to ensure it does not exceed the specified range of acceptable angles. The BNO055 requires a manual calibration process at start up before accurate readings can be obtained and is mounted to the electronics tray inside the watertight electronics enclosure.

3.5.4.4. SOS Leak Sensor

The Blue Robotics SOS leak sensor provides the BlueFish with a failure detection mechanism in the case of a leak in the sealed electronics enclosure. Sponge-tipped probes are placed along the edges of the enclosure caps to ensure that if any water enter the enclosure due to an improper seal it will be detected (see Figure 40). If water touches the probes, the leak sensor will create a produce a "HIGH" 5 V digital signal which can be read by any of the Arduino's digital pins.

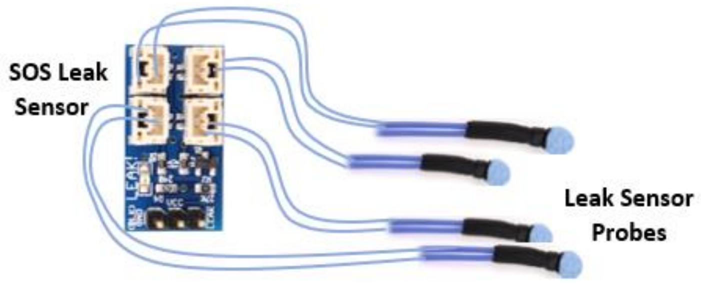


Figure 40. Blue Robotics SOS Leak Sensor and Probes.

3.5.4.5. Power Sense Module

The Blue Robotics PSM provides the BlueFish with the ability to measure the voltage and current draw of the BlueFish batteries during operation. The PSM utilizes a hall-effect sensor to convert the battery voltage and current readings into proportional 0-3.3 V outputs with maximum voltage and current sensing of 25.2 V and 100 A, respectively [18]. The PSM is wired in series with the batteries and the 3.3 V output lines from the PSM are connected to the analog ports of the Arduino (see Figure 41). The PSM voltages are read using ADC in the Arduino firmware and internally converted from the raw ADC values into the battery voltage and current values.

Figure 41. Power Sense Module [18].

3.5.5. Actuators

3.5.5.1. Waterproof Servo Motor

The BlueRobotics waterproof servo motors utilized by the BlueFish are not currently available for consumer purchase as they are currently in development. A nearly identical model is shown below in Figure 42. To supply the servo motors with the required 7.4 V up to 2 A, a variable 3 A voltage regular was included in the mechatronic system. The servo motors are capable of outputting a maximum torque of 40.6 kg-cm at the supplied 7.4 V and have a PWM range of 500 µs to 2500 µs [19]. For full technical details, see Appendix S.

The servo motors control the rotation of the hydrofoils which allow the BlueFish to control its depth, altitude, pitch and roll during motion. The output PWM signal to the servo motors are determined through use of a PID controller implemented in the Arduino firmware. The servo motors are waterproof up to 300 m and mounted in the non-watertight rear assembly inside the fishtail enclosure.

Figure 42. Servo Motor (Non-Waterproof Version) [19].

The servo motors control the rotation of the hydrofoils which allow the BlueFish to control its depth, altitude, pitch and roll during motion. The output PWM signal to the servo motors are determined through use of a PID controller implemented in the Arduino firmware. The servo motors are waterproof up to 300 m and mounted in the non-watertight rear assembly inside the fishtail enclosure.

3.5.5.2. System LED Lights

Three generic blue LED lights are attached to the Arduino expansion shield to a provide visual indication of sensor initialization issues, leak warnings, and calibration status of the BNO055 IMU (see Figure 43). The LEDs are connected to the digital pins of the Arduino and wired in series with 220 Ω resistors to limit the current flow from the MCU.

LEDs that are flashing during the initial setup (before calibration) indicate that one of the sensors has not been initialized and that there could be a hardware connection issue. During calibration, a solid LED turning on will indicate that the corresponding sensor (gyroscope, accelerometer, or magnetometer) has reached its highest level of calibration. All LEDs flashing during operation indicate that a leak has been detected in the electronics enclosure. Explanation of the LED visual indicators are further detailed in Section 3.6.2.

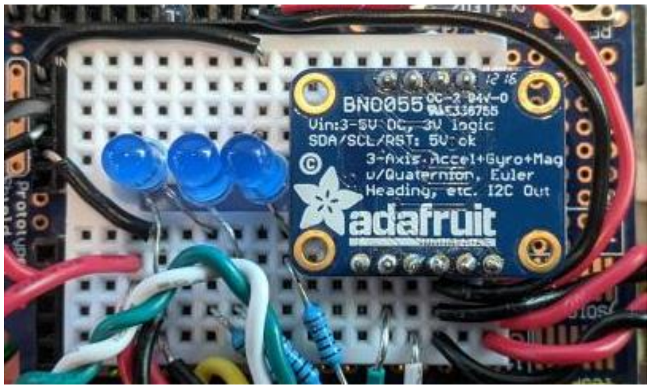


Figure 43. BlueFish System LED Lights Configuration.

3.5.6. *Integration*

With the Raspberry Pi, MCU, and sensors defined, the mechatronics team began working on integrating the components together, detailing any additional interfacing hardware needed for the FishGuts. Such components ranged from sourcing a 14.8 V lithium-ion battery to Fathom-X ethernet boards for communication between the top and bottom side Pi's. Once power and current draw calculations were done (see Appendix K), a preliminary component connection diagram (see Appendix L) and initial wiring schematic (see Appendix N) were created to show the physical connections and pinning of the FishGuts I.

Components were then ordered and received, leading to the creation of FishGuts I and FishGuts II (see Figure 44). FishGuts I represented a flexible prototyping board for component and circuit testing. However, this prototype was large, immobile, and was void of critical components like the Bar30 depth sensor. FishGuts II was created as a step towards a more permanent solution for the BlueFish, implementing an Arduino prototype shield that mounts to the top of UNO. For this prototype, a small breadboard was used and mounted atop the prototype expansion board to allow for easy iteration and flexibility. However, the prototype board can also be used without it. For the time being, connections are still made through pins and a small breadboard, but the shield has soldering pads and bars for more permanent solutions. See appendix M and O for the finalized component connection diagram and wiring schematic, respectively.

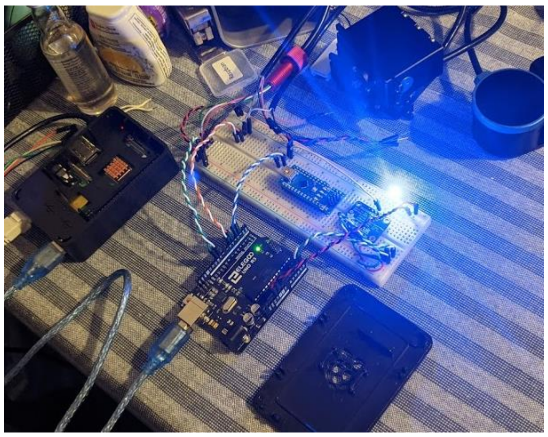
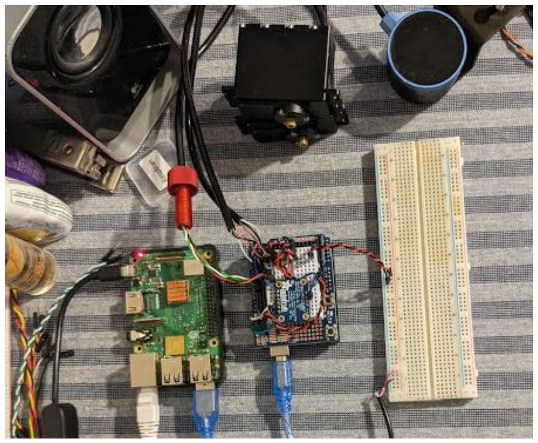



Figure 44: FishGuts I (Left) and FishGuts II (Right).

3.6. Software and Firmware

3.6.1. *Initial Software Installation*

A collaborative environment with version control was needed so that multiple members could work on the project at the same time. GitHub was chosen as the method for version control as it gives individuals the ability to work on the same project code and merge their work together while providing a history of changes made. Using Git also gives the Raspberry Pi access to the most up-to-date code given an internet connection, eliminating the need for physical access to the FishGuts for firmware updates. To reduce the learning curve for members unfamiliar with Git, GitKraken was set up to provide an intuitive GUI that visually represents Git commands and version control, removing the need to learn Git coding for members unfamiliar with Git.

Raspbian (Pi's native OS), was downloaded and flashed onto the Raspberry Pi with custom instructions to automatically connect to a member's wi-fi network and enable SSH protocol for remote terminal access. With this, the Raspberry Pi could be set up with the latest version of Python (3.9.2), Github, Arduino IDE, and VNC Viewer, which provides users full access to the Pi's desktop and user interface remotely.

To ease firmware debugging, Virtual Studio Code with Platformio was chosen as the IDE for programming the Arduino. This IDE allows for real time error correction when coding and provides smart code completion based on the variable types and installed libraries. For Python development, Pycharm was primarily used for benefits that include Intellisense code completion, and PEP 8 suggestions to assist in with maintaining standardized formatting practices.

3.6.2. Arduino Firmware

Preliminary firmware creation for the Arduino consisted of generating basic code to test reading of digital inputs, analog inputs, data from the I^{2}C sensors, and UART serial data. This firmware also provided PWM output signals to control the servo motor positions. Once the firmware was able to correctly read the sensors and control the servo motors, work began on implementing a PID controller that could modify the output of the servo positions based on external feedback from sensors. The PID functionality was initially tested by controlling the servo motors based on the orientation data provided by the BNO055 IMU sensor, essentially providing fins that were self-levelling by maintaining a horizontal position for any pitch angle.

With the main functions and components created, the FishBrains pseudocode was developed to provide a rough layout as to the final version of the firmware and what functionality needed to be added. The Arduino firmware pseudocode flowchart can be found in Appendix W.

On start-up, the Arduino firmware first performs an initialization check to ensure that each of the sensors are connected properly and can be initialized. If a sensor does not initialize, the function will sit in a loop and flash the corresponding LED to indicate that sensor is not connected. LED 1 corresponds to the BNO055, LED 2 corresponds to the Bar30, and LED 3 corresponds to the Ping sonar (see Figure 45). If all sensors correctly initialize, no LEDs will flash, and the program will move into the calibration function.

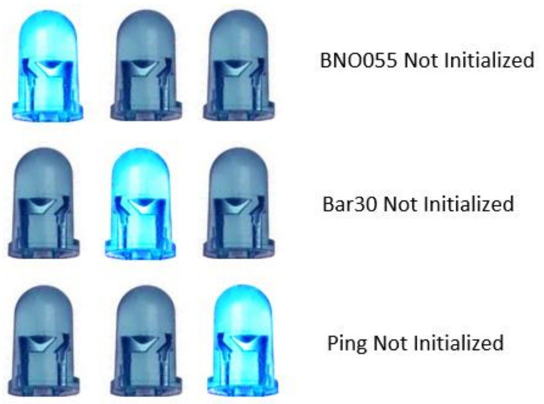


Figure 45. LED Indication of Sensor Initialization.

The calibration function ensures that the BNO055 IMU data being received is accurate. Each of the sensors must be calibrated before the main loop of the program will execute. To visually indicate the calibration status of the internal gyroscope, accelerometer, and magnetometer, the same system LEDs are used (see Figure 46).

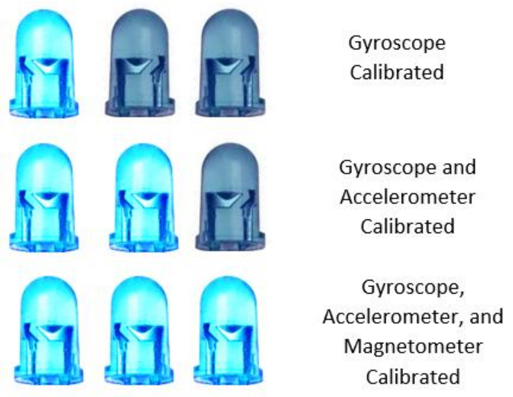

Gyroscope Calibrated
Gyroscope and Accelerometer Calibrated
Gyroscope, Accelerometer, and Magnetometer Calibrated

Figure 46. LED Indication of BNO055 Calibration.

During the calibration process for the BNO055, the gyroscope will typically calibrate on its own by leaving it stationary for a couple seconds. The accelerometer is calibrated by slowly rotating the BlueFish 180° then rotating back in 45° increments. The magnetometer is calibrated by holding the nose cone in the same spot and moving the tail of the BlueFish in a figure-8 pattern (see Figure 47).

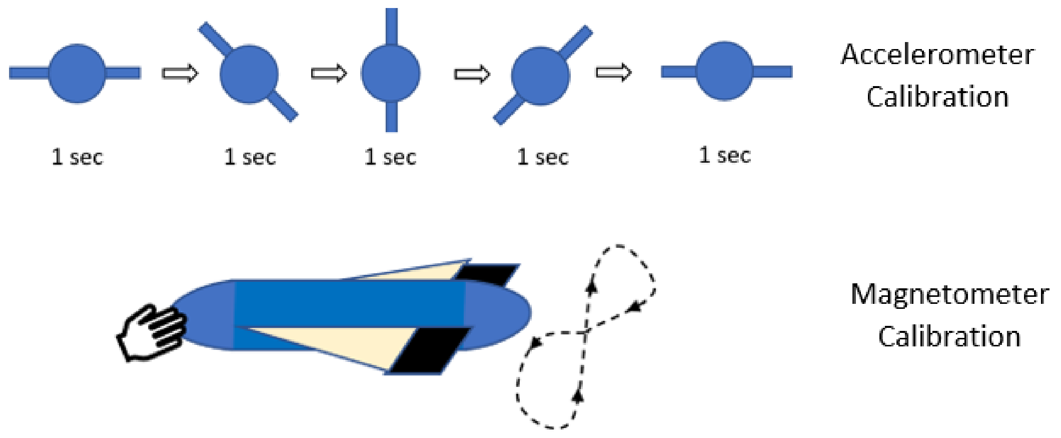


Figure 47. BNO055 Calibration Procedure

The final version of the Arduino firmware implements a state machine which allows for continuous operation in one of the pre-defined states/modes. The available states include an idle, constant depth, altitude, and surface mode. The mode is selected from the available options in the Raspberry Pi GUI and determines the behavior of the control system. In standby and surface modes, the Arduino does not read sensors or transmit data, but maintains the servo motor positions at the initial position (horizontal) and waits for a mode change from the Raspberry Pi. The inclusion of the surface mode was to accommodate a different constant servo position while being towed on the surface if required for stability.

In constant depth mode, the control system will compare the target depth value entered in the GUI to the depth values obtained from the Bar30 sensor. In altitude mode, the control system will compare the target altitude value entered in the GUI to the distance values obtained from the Ping sonar sensor. While operating in the constant depth and altitude modes, the Arduino will continuously read each of the sensors and update the global variable values. The Arduino will also transmit all the data collected from the sensors if the elapsed time exceeds the log period defined by the log rate set in the GUI. The data that is transmitted from the Arduino to the Raspberry Pi and the respective units of measure are summarized in Table 6.

Table 6: Sensor Data Transmitted from Arduino.

Output Variable	**Unit**
Altitude	m
Altitude Error	m
Depth	m
Depth Error	m
Pressure	kPa
Temperature	℃
Yaw	°
Pitch	°
Roll	°
Battery Voltage	V
Battery Current	A

Fail safe checks have also been implemented in the firmware to ensure that if the BlueFish exceeds the maximum depth, minimum altitude, or if a leak is detected, it will adjust its trajectory to avoid damage. This is accomplished by setting the servo motor positions to the angular position of maximum lift (18°) for rapid ascension. If a leak is detected by the leak sensor, the firmware will continuously flash all the LEDs, set the target depth to zero, and set the mode to constant depth.

If the BlueFish settings are changed at any time in the GUI, an external interrupt in the Arduino firmware is triggered by the Raspberry Pi which sets a flag in the ISR and instructs the program to read data from the serial port and update the global variable values.

In both constant depth and altitude modes, control of the BlueFish depth and distance from the seafloor is accomplished using a PID controller which compares the target value to the measured value and applies gains to adjust the servo motor position accordingly. PID gain values, K_P, K_I, and K_D, are difficult to determine theoretically, often requiring complex mathematical modelling. Typically, PID gain values for control systems are determined based on empirical data gained through testing the response of the control system. A summary of how each of the gains affect the dynamic response of the BlueFish is summarized in Table 7. For reference, the full Arduino firmware code can be found in Appendix X.

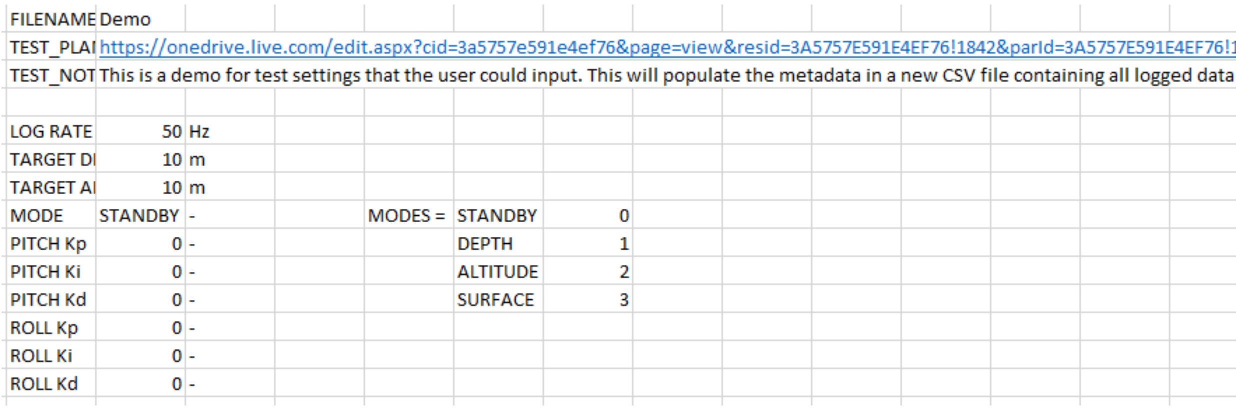
Table 7: PID Gain Effects on Control System Response.

Parameter Increase	Rise Time	Overshoot	Settling Time	Steady-State Error
K_p	↓	↑	Small Change	↓
K_i	↓	↑	↑	Large Reduction
K_d	Small Change	↓	↓	Small Change

3.6.3. Raspberry Pi Software

Outside of the native software included in Raspbian, VNC Viewer was added to the Raspberry Pi to allow remote desktop access to the Pi during operation. This means that code could be actively run and updated while operating the BlueFish along with additional features such as live plotting.

Python was the programming language of choice given its suitability to data and data science, vast selection of modules, available resources, and relatively easy learning curve. The first iteration of the program that communicated and with the Arduino firmware almost exclusively relied on the use of CSV files. The user would use a settings CSV (see Figure 48) to determine tune PID settings and change operation modes. If running, the python code could detect any changes made and push those changes to the Arduino while starting a new CSV file for data logging.



FILENAME	Demo					
TEST_PLAI	https://onedrive.live.com/edit.aspx?cid=3a5757e591e4ef76&page=view&resid=3A5757E591E4EF76!1842&parId=3A5757E591E4EF76!1					
TEST_NOT	This is a demo for test settings that the user could input. This will populate the metadata in a new CSV file containing all logged data					
LOG RATE	50	Hz				
TARGET DI	10	m				
TARGET AI	10	m				
MODE	STANDBY	-		MODES =	STANDBY	0
PITCH Kp	0	-			DEPTH	1
PITCH Ki	0	-			ALTITUDE	2
PITCH Kd	0	-			SURFACE	3
ROLL Kp	0	-				
ROLL Ki	0	-				
ROLL Kd	0	-				

Figure 48: Example Settings CSV

To improve user experience, it was decided that a GUI should be made to reduce the number of programs needed to be run at one time. Instead of opening both the python program and the settings CSV, a GUI would reduce the running programs to one. This also provided the benefit of allowing for different BlueFish settings to be saved and loaded into the program while running. The GUI was primarily built off the PyQt5 python module by Riverbank Computing. PyQt5 is a powerful module originally developed for C++ (Qt5) that is primarily built to give programmers the tools to build modern functional GUI's. The GUI accompanying the BlueFish, named Fish Command, consists of two primary tabs. The "Settings" provides the user the ability to create, save, and load settings, as well as push them to the BlueFish (see Figure 49). Shortcuts were also added to streamline saving ('Ctrl'+'S') and loading ('Ctrl'+'L') settings in the GUI.

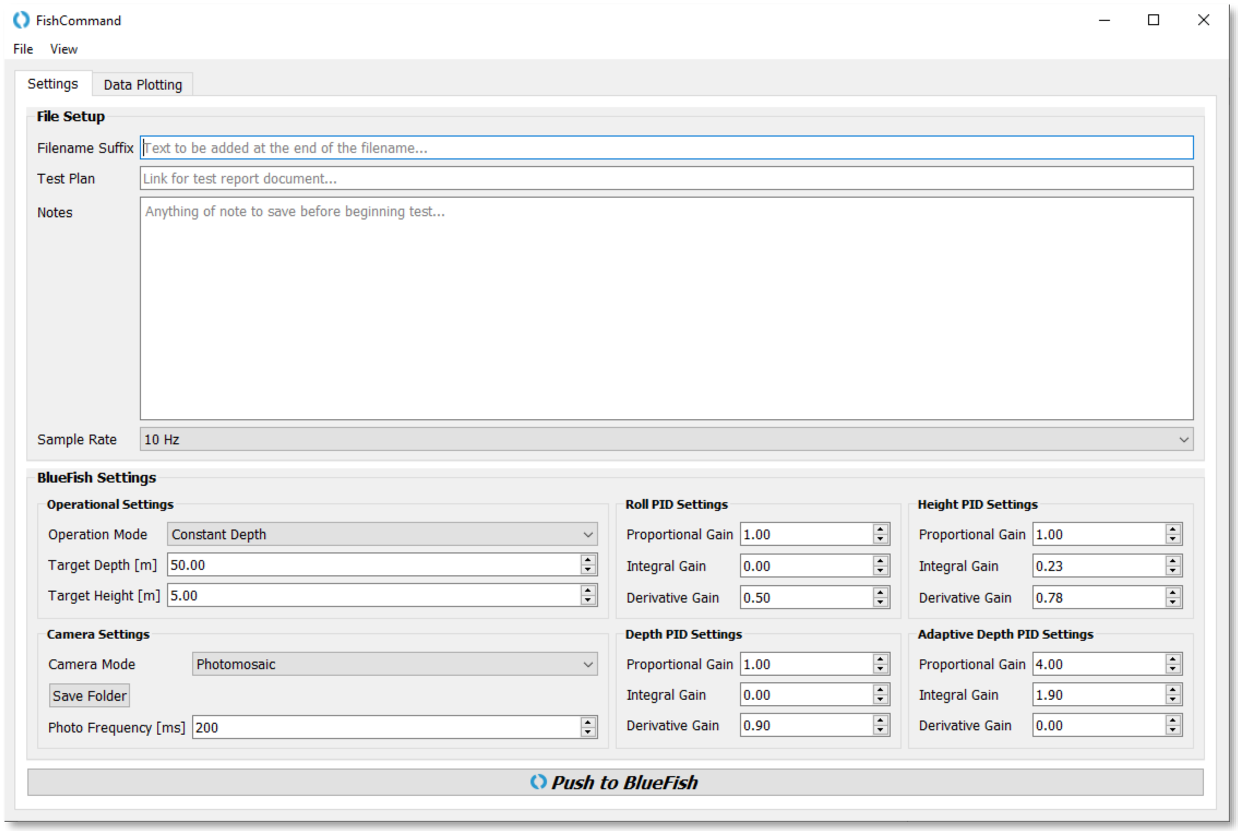


Figure 49: Blue Command Settings Tab.

As can be seen, there's also room for adding a "Filename Suffix" that will be automatically generated when the user is prompted to choose a save location and name, along with a "Test Plan" section for online report links and a "Notes" section for any specific aims with a test. Both the test plan and the notes section will be added to the metadata at the beginning of each data log CSV file. This all happens in the backend of the code, along with the Pi setting a digital pin high to trigger an ISR on the Arduino UNO before sending updated settings for the BlueFish to operate under.

Additionally, a "Data Plotting" tab (see Figure 50) was added to give the user the ability to see their data live as it is being captured. The implementation of this tab was largely inspired by the packaging constraints that prevented the use of a Pixhawk with QgroundControl to be used for live data visualization. However, although the code is near completion, this feature did not reach full functionality at the time of this report.

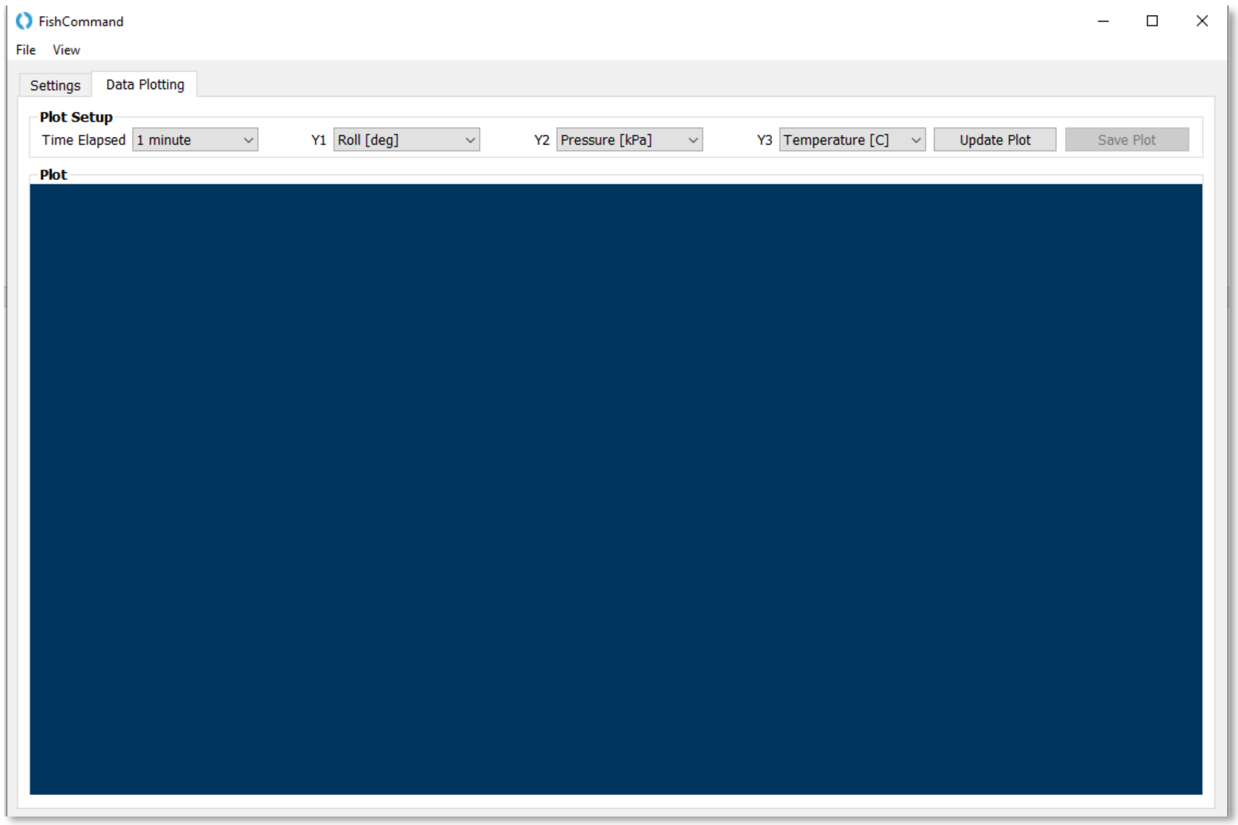


Figure 50: Fish Command Data Plotting Tab.

To implement functionality such as live plotting and photomosaic capabilities, previous coding methods and infrastructures used by the group were too slow, potentially leading to serial port buffer overflow and lost data. Therefore, threading was implemented to carry out multiple simultaneous processes, improving the overall performance and responsiveness of the program.

Threading essentially allows a program to handle multiple processes concurrently by creating separate threads of code that can operate independently in the same data space (see Figure 51). While the code cannot technically operate at the same time (python is inherently single threaded and process), I/O operations, such as opening a file and writing to it, or receiving serial data, can be done in the background while another thread is being executed. By implementing this, the code retained the ability to capture all data being sent from the Arduino while providing the user with a responsive GUI that will allow for future integration of live plotting and photo capture.

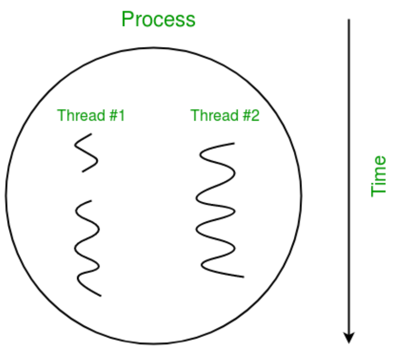
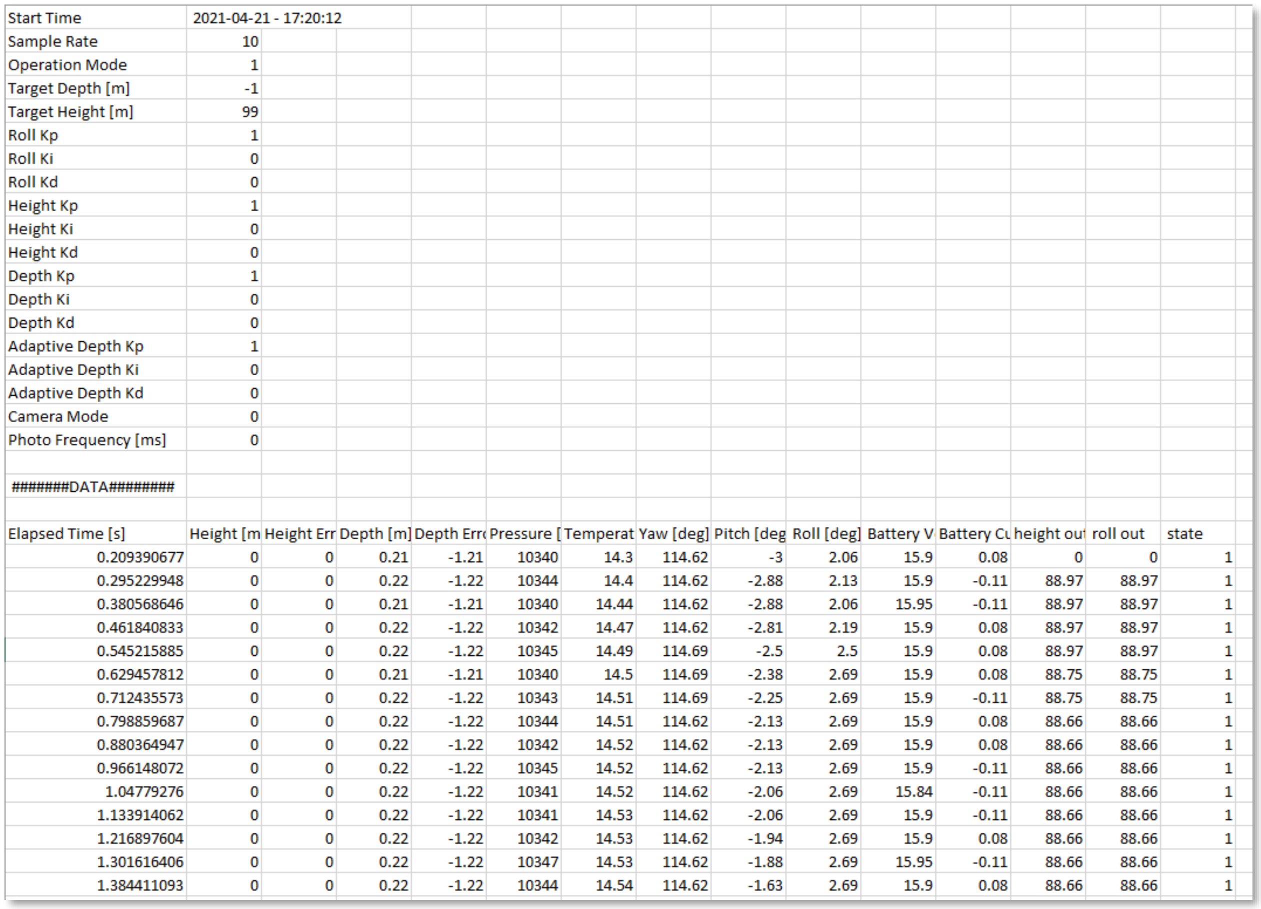


Figure 51: Visual Representation of Threading in Python.

Once live plotting is integrated into the final code, users will be able to select their desired time elapse to view in the plot, along with up to three different variables being sent from the Arduino. Pressing the “Update Plot” button will update these settings for the user instantly, while the “Save Plot” button will save a GIF or PNG image to the user’s desired location. The original implementation of this code used a separate thread to read the CSV actively being written by the logging thread. However, this added complexity and computational resources to the code as care needed to be taken to avoid a race condition where a program tries to read and write into a file at the same time.

The live plotting code uses matplotlib, a module developed to essentially wrap MATLAB plotting capabilities and syntax into a Python package. However, counter to the original code developed for Fish Command, this method requires that the plot figure be created and shown in the main GUI thread. In the latest iteration, the logging thread will save the data into a list of values before emitting a PyQt5 signal to a PyQt5 slot in the main thread, appending each line of data into a dataframe structure. This dataframe is will then be used to plot on “Data Plotting” tab. While most of the code is in place to do this, some work needs to be done to integrate the original separate thread code into the main GUI thread.

Next, the chosen solution for the generation of a photomosaic of the seafloor was to connect a USB camera which is available on the Blue Robotics website directly to the Raspberry Pi. The first iteration of the software integration involved using the Raspberry Pi fswebcam package which is written to the terminal. Next, the command needed to be looped, which could be accomplished using one of two methods. The first was to write a Bash script that would be called and executed from within the Python code. The other was to use a Python library which could write commands to the terminal. The latter was selected for its reduced interfacing. However, when the program was complete, it was found that it could only take a picture every 2.3 seconds, due to the method in which it could identify and utilize a connected camera. Additionally, the focus and zoom functionality of the fswebcam package was limited. At this point, other solutions were explored. Conveniently, the same Python library that was used to develop the BlueFish GUI, PyQt5, contained a library called QtMultimedia which could interface with a connected camera. This included the preliminary configuration of the camera and all the necessary zoom and focus functionality. This code can be found in Appendix AB. Unfortunately, it was found that the Raspberry Pi could not identify the QtMultimedia library and therefore could not identify the objects and methods used in the program.



Start Time	2021-04-21 - 17:20:12													
Sample Rate	10													
Operation Mode	1													
Target Depth [m]	-1													
Target Height [m]	99													
Roll Kp	1													
Roll Ki	0													
Roll Kd	0													
Height Kp	1													
Height Ki	0													
Height Kd	0													
Depth Kp	1													
Depth Ki	0													
Depth Kd	0													
Adaptive Depth Kp	1													
Adaptive Depth Ki	0													
Adaptive Depth Kd	0													
Camera Mode	0													
Photo Frequency [ms]	0													
#######DATA########														
Elapsed Time [s]	Height [m	Height Err	Depth [m]	Depth Err	Pressure [	Temperat	Yaw [deg]	Pitch [deg	Roll [deg]	Battery V	Battery Cu	height out	roll out	state
0.209390677	0	0	0.21	-1.21	10340	14.3	114.62	-3	2.06	15.9	0.08	0	0	1
0.295229948	0	0	0.22	-1.22	10344	14.4	114.62	-2.88	2.13	15.9	-0.11	88.97	88.97	1
0.380568646	0	0	0.21	-1.21	10340	14.44	114.62	-2.88	2.06	15.95	-0.11	88.97	88.97	1
0.461840833	0	0	0.22	-1.22	10342	14.47	114.62	-2.81	2.19	15.9	0.08	88.97	88.97	1
0.545215885	0	0	0.22	-1.22	10345	14.49	114.69	-2.5	2.5	15.9	0.08	88.97	88.97	1
0.629457812	0	0	0.21	-1.21	10340	14.5	114.69	-2.38	2.69	15.9	0.08	88.75	88.75	1
0.712435573	0	0	0.22	-1.22	10343	14.51	114.69	-2.25	2.69	15.9	-0.11	88.75	88.75	1
0.798859687	0	0	0.22	-1.22	10344	14.51	114.62	-2.13	2.69	15.9	0.08	88.66	88.66	1
0.880364947	0	0	0.22	-1.22	10342	14.52	114.62	-2.13	2.69	15.9	0.08	88.66	88.66	1
0.966148072	0	0	0.22	-1.22	10345	14.52	114.62	-2.13	2.69	15.9	-0.11	88.66	88.66	1
1.04779276	0	0	0.22	-1.22	10341	14.52	114.62	-2.06	2.69	15.84	-0.11	88.66	88.66	1
1.133914062	0	0	0.22	-1.22	10341	14.53	114.62	-2.06	2.69	15.9	-0.11	88.66	88.66	1
1.216897604	0	0	0.22	-1.22	10342	14.53	114.62	-1.94	2.69	15.9	0.08	88.66	88.66	1
1.301616406	0	0	0.22	-1.22	10347	14.53	114.62	-1.88	2.69	15.95	-0.11	88.66	88.66	1
1.384411093	0	0	0.22	-1.22	10344	14.54	114.62	-1.63	2.69	15.9	0.08	88.66	88.66	1

Figure 52: A CSV file generated by Fish Command while running the BlueFish during testing.

Overall, the code breaks down into 4 python files. The main thread that Fish Command runs on can be found in Appendix Y, where an instance of the GUI (found in Appendix Z) is created and given functionality. It is worth noting that this code also contains most of the code required to implement live data plotting, although it has been commented out to retain functionality. Appendix AA contains the csv logger code. When an instance of the "Logger" class is created in the main code, a CSV file is created where all metadata will then be populated at the top of the document (as seen in Figure 52). From there, the main code will start a logging thread from that class that will continuously check the serial port for data and push any data received into the CSV log file. The last Python file, found in Appendix AB, contains the code for taking pictures for photomosaics. In the future, this code will operate similarly to the CSV logger, running on its own thread.

3.7. Final Prototype Design

Using the information gathered over the duration of the project and design cycle, the final and fully integrated prototype, the BlueFish I (see Figure 53 and Figure 54), represents the culmination of the successes, failures, and hydrodynamic work over the course of the project and product design cycle. Together the BlueFry III, FishGuts II, FishBrains, and Fish Command form this final prototype. An exploded view, assembly drawing, and part drawings can be found in Appendix C, while a bill of materials and cost breakdown can be found in Appendix J.

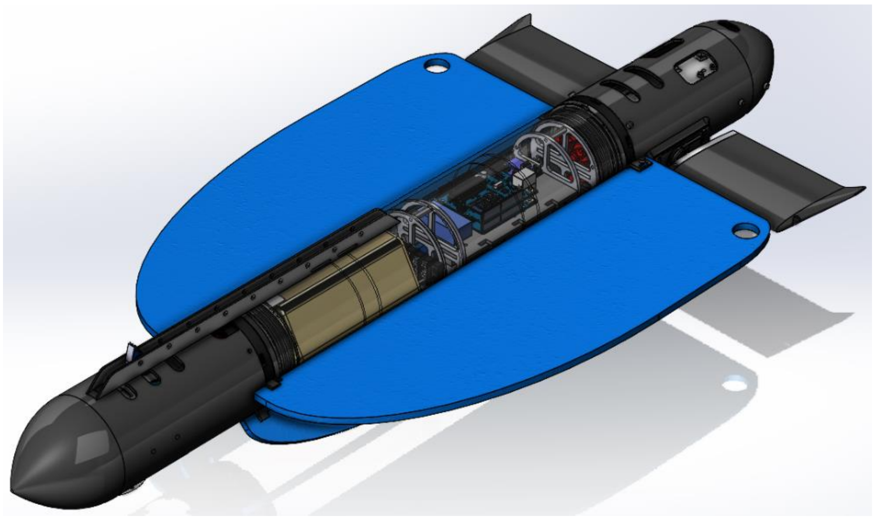


Figure 53: Top-Level Assembly CAD Model of BlueFish I.

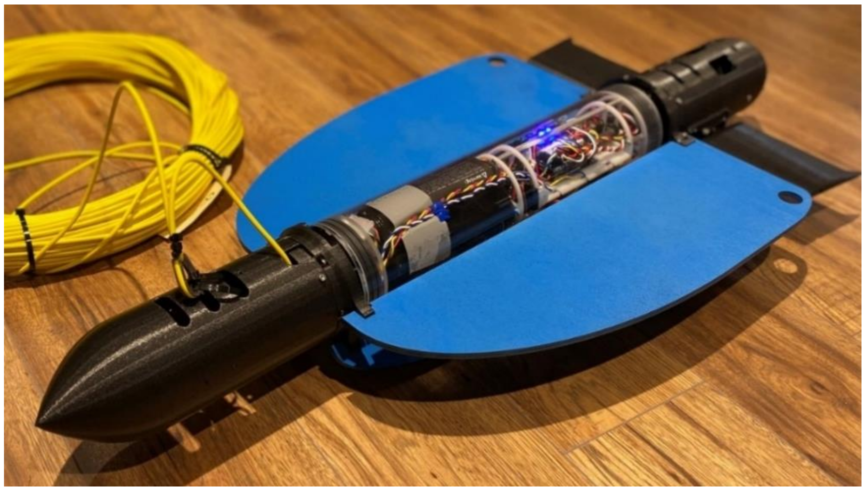


Figure 54: Full BlueFish Prototype.

4. Project Completion

The completion of this project includes the testing of the first functional prototype, the BlueFish, along with the evaluation of the prototype with respect to the user requirements and engineering specifications, which were produced and agreed upon in the project preparation. Since this prototype is intended to become a marketable product, the expected future work is also described below.

4.1. User Requirements and Engineering Specifications

The user requirements and engineering specifications tables laid a framework for the project and were referenced frequently during the design process. As the prototype was assembled the tables were used to set targets for testing.

4.1.1. User Requirements

Table 8: User Requirements

Priority	ID	Short Name	Must/ Should	Description
1	1	Self Controlled	Must	Can control and maintain its depth, pitch, and roll at any point, without dynamic surface control.
2	2	Payload Modularity	Must	Can accept a variety of payloads with little effort.
3	7	Blue Robotics Compatibility	Must	Compatible with the BlueBoat and may integrate with Blue Robotics components.
4	3	Standalone Unit	Must	Fully functional without any additional Blue Robotics accessories or components.
5	9	Size	Must	Easily and safely portable for one person.
6	8	Range	Must	Sustain relatively long trips with a payload(s) attached.
7	6	Operational Depth Range	Must	Be capable of diving deep enough for most researcher's and hobbyist's needs.
8	5	Affordability	Must	The unit must be affordable to hobbyists & researchers.
9	4	Topside Communication	Should	Capable of transmitting data to the surface and accepting a command to dive.
10	11	Failsafe Recovery	Should	Recoverable in the event of electrical or mechanical failures.
11	10	Photo Mosaic	Should	Capable of creating a photomosaic of a water body.

Table 9: User Requirements Rationale.

Priority	ID	Short Name	Must/ Should	Rationale
1	1	Self Controlled	Must	To provide unique capabilities and value to customers that do not currently exist in the market.
2	2	Payload Modularity	Must	To meet the dynamic and often specific needs of Blue Robotics' key target demographic.
3	7	Blue Robotics Compatibility	Must	The BlueFish and BlueBoat being compatible will create more opportunities for BlueBoat users and designing to accept Blue Robotics components increases use cases.
4	3	Standalone Unit	Must	To enable customers the ability to tow the BlueFish from any source.
5	9	Size	Must	To allow full operation by small research teams and hobbyists, and to fit inside the packaging constraints of the BlueBoat.
6	8	Range	Must	Researchers using equipment for these purposes prefer large sampling windows. Redeploying during testing is also often impractical.
7	6	Operational Depth Range	Must	Researchers will need to use at a range of depths for various research data requirements.
8	5	Affordability	Must	Blue Robotics aims to “enable the future of ocean exploration” through accessible equipment used to study the oceans.
9	4	Topside Communication	Should	Provide communication to increase functionality for users, allow for troubleshooting, and tune parameters.
10	11	Failsafe Recovery	Should	Reduce costs of failures (both fiscal and data). Client has stated this is not a necessity within the target price range.
11	10	Photo Mosaic	Should	One of the main use cases for the BlueFish; it is in line with the payload modularity requirement and is not required but an objective for the final prototype to prove functionality.

4.1.2. Engineering Requirements

Table 10: Engineering Specifications

Priority	ID	User Req ID Reference	Short Name	Value	Unit	Description
1	14	1,2,4,10	Depth Control	+/- 0.25	m	To maintain stability for payload measurements; side scan sonar requires minimal variation for accurate results.
2	13	1,2,10	Response Frequency	< 1/2	Hz	Must be stable enough to not oscillate at high frequency; specific payload readings are sensitive to high oscillations.
3	16	1,2,10	Pitch Control	+/- 3	deg	Must be stable enough to provide consistent data readings.
4	15	1,2,10	Roll Control	+/- 3	deg	Must be stable enough to provide consistent data readings.

5	7	1,4,10	Altitude	1 to 10	m	Must be stable enough to provide consistent data readings.
6	8	2,6	Depth Rating	100	m	Most data collection use cases will only require a maximum depth of 100 m.
7	5	2,10	Diameter	< 4	in	To maintain portability and reduce drag.
8	4	2,10	Length	< 1.1	m	To maintain portability and reduce drag.
9	2	8,9	Battery Longevity	10	hr	To attain sufficient operational data collection time for most researchers/use-cases.
10	3	7,8,10	Operational Speed	0.8 to 1.2	m/s	To retain compatibility with BlueBoat operational speeds.
11	1	2,3,5	BoM Cost	320 to 400	USD	Determines ability to sell, margins, and target demographics.
12	10	9,11	Buoyancy	Adjustabl e	-	To allow for varying payloads and retain a method of recovery in the event of electrical or mechanical failure.
13	9	7,8,9,10	BlueBoat Power Draw	< 100	W	BlueBoat must also be able to operate for 10+ hours when towing BlueFish.
14	12	2,8	Internal Accessibility	< 2	min	The batteries will need to be changed from the surface between uses.
15	6	8,9	Weight	< 30	lb	To maintain portability and safe transportation by an individual as defined by WorkSafeBC [20].

Table 11: Engineering Specifications Rationale.

Priority	ID	Short Name	Value	Unit	Justification
1	14	Depth Control	+/- 0.25	m	Accurate side scan sonar is very susceptible to changes.
2	13	Response Frequency	< 1/2	Hz	The product should track error smoothly and should not be overshooting and undershooting at a high frequency.
3	16	Pitch Control	+/- 3	deg	Must be able to maintain consistent pitch within 6 degrees for projected use cases.
4	15	Roll Control	+/- 3	deg	Must be able to maintain consistent roll within 6 degrees for projected use cases.
5	7	Altitude	1 to 10	m	Needs to actively maintain distance within 1-10 meters from sea/lake floor for photomosaics.
6	8	Depth Rating	100	m	Must be able to withstand temperatures and pressures depth of 100 m for most customer uses.
7	5	Diameter	< 4	in	Smaller than 4 inches in outer diameter, excluding fins, control surfaces, from the clients request to use a 3" enclosure.
8	4	Length	< 1.1	m	Less than the length of a BlueBoat (to fit packaging standards).
9	2	Battery Longevity	10	hr	Minimum 10 hr for highest consumption payload (side scan).
10	3	Operational Speed	0.8 to 1.2	m/s	Operates correctly at typical BlueBoat speeds.
11	1	Price	320 to 400	USD	Target BOM cost of the product for production runs of 100-500 a year, respectively.
12	10	Buoyancy	Adjustable	-	Should be adjustable in increments of 1/20 the mass of the BlueFish.
13	9	BlueBoat Power Draw	<100 (<60)	W	BlueBoat must draw less than 100W, should draw less than 60W in flat water towing the BlueFish.
14	12	Internal Accessibility	< 2	min	To access the internal components, it should take no longer than two minutes of disassembly.
15	6	Weight	< 30	lb	Should be light enough for one person to maneuver easily/safely.
16	11	Lifespan	> 3	yr	The product should have an expected lifespan of at least three years.

4.2. Final Prototype Design Evaluation

As previously explained in Section 2, the timeline did not allow for all the intended testing of the BlueFish prototype, despite the best efforts from the project team members. The testing that was completed is reported on below, along with the evaluation of the final iteration of the prototype with respect to the user requirements and engineering specifications.

As described in Section 3, the overall mechanical system of the BlueFish was designed and thoroughly analyzed. It was through this extensive and systematic process that this system progressed in a timely manner while also progressing with few instances of superfluous or redundant work. The mechanical system, through a great design effort and preliminary analysis, ultimately met the specifications as described by the user requirements. Using mounting holes on the endcaps, new payloads can easily be mounted to the system, greatly increasing the modularity of the BlueFish design. Additionally, the design intent is that various different nosecone configurations can be mounted for different testing purposes. For instance, the base nosecone can be manufactured without any cut-outs for the lumen and camera configuration, or if desired, a photomosaic variant can be manufactured, such as in the case of the final BlueFish prototype.

Moreover, the design was made with compatibility in mind. Using a Raspberry Pi and fathom interface, it allows for the use of typical Blue Robotics communication protocols. Additionally, the endcaps accept M10 penetrators, which are also manufactured by Blue Robotics. Similarly, the 3” cast acrylic main enclosure accepts a full-sized Blue Robotics battery. This makes the design ideal for towing behind a BlueBoat and communicating over the BlueBoat’s network. Thus, it is the BlueFish’s near seamless integration into Blue Robotics’ pre-established systems that allows the BlueFish to have a substantial amount of overall compatibility.

In addition, the BlueFish is a standalone unit and does not require additional Blue Robotics components to function. It can be towed behind any vessel capable of achieving the recommended speed. The only Blue Robotics accessory that is required is a second, topside, FXTI such that the ethernet connection is correctly transmitted to the user’s computer.

The BlueFish also successfully hit all its size requirements. As the full unit weighs approximately fifteen pounds (6.8 kg), it is easily transportable by a single person. As for the dimensions, excluding the fins, the entire unit is less than four inches in diameter at 3.5 inches. It is 0.89 m long, which is 19.4% less than the maximum length of 1.1 m specified in the engineering specifications.

Additionally, as long trips are often required for research purposes, the BlueFish can sustain an approximate seventeen-hours of consistent power draw with the photomosaic package. This is a 170% increase in the required, single-charge battery life of ten hours. Although it was not tested, the base (non-photomosaic) variant would draw even less power and would increase the BlueFish’s battery life even further. During testing with the final BlueFish prototype (a photomosaic variant), it could be seen that at operating conditions that the maximum current drawn was less than 0.5 A, averaging at approximately 0.1A, which was well within the theoretical expectations (see section 3.2.4, Appendix K, and Appendix R).

Next, the depth range requirement was achieved by using a Blue Robotics COTS enclosure with custom machined endcaps. Utilizing two radial seals for the flanges and an axial seal for the endcap, the only consideration for maintaining a watertight enclosure was pertaining to the flatness of the sealing surface. Fortunately, the custom components were produced by the same vendor that is currently used by Blue Robotics. As the enclosure is rated for 150 m of pressure, this well surpasses the 100 m requirement by 150%.

Like the aforementioned segments, minimizing components and producing an efficient design were important parts of the project. Thus, DFM and DFA were conducted to aid in the assemble-ability and cost of the BlueFish. While the prototype has an estimated cost of $1164.55 USD, the estimated retail price of the full BlueFish prototype, as seen in Appendix J, was $579.55 USD, which is greater than the initial $320 to $400 USD specification. However, as further prototypes are made, the cost is expected to decrease to within the initial budgetary constraints. A detailed bill-of-materials and price breakdown can be found in Appendix J for a base model production model, a photomosaic production model, as well as the final BlueFish prototype.

Furthermore, researchers should be able to communicate with the BlueFish and send commands to dive. A GUI was developed using the PYQt5 module to interface with the Raspberry Pi, which can send signals to the Arduino and engage different control states. It is recommended that the GUI be further developed for the end user, but the pre-existing method worked well for testing purposes. This signal is passed through a tether, presumably from Blue Robotics; however, different tethers can be used.

Moreover, several factors were implemented such that the BlueFish can be recovered if a mechanical or electrical failure occurs. An example is the leak sensors that are used to notify the user as if a leak were to occur. In addition, various prompts and errors were built into the code to notify the user if an error were to occur. Also, the BlueFish has available space for adjustable buoyancy pucks so that the prototype can be slightly positively buoyant. These implemented factors collectively help to monitor and prevent any catastrophic failures.

The final user requirement was that the BlueFish would be capable of creating a photomosaic of the seafloor. Thus, a camera and lumen were installed to "pulse" at a specific frequency. However, due to time constraints, a photomosaic was not made, but the code was tested on a separate Raspberry Pi.

The first user requirement, self-controlled, pertains to several of the engineering specifications that have not yet been mentioned above. For example, a controlling state machine was implemented on the Arduino such that the BlueFish could control depth, oscillation frequency, pitch control, and altitude. The algorithm was validated in Calgary but was not validated with the sensor that outputs the depth signal due to missing shipments. For this reason, these were only briefly tested. The results are further outlined in the following section.

4.3. Final Prototype Testing

The final prototype was delayed due to numerous unfortunate events, abbreviated debugging time, and faulty components. The Bottom Feeders were able to test for two days from a dock in Victoria Harbour. The first day was a waterproof validation test where the BlueFish was submerged in approximately 5m of water column for approximately an hour. During the test, no leaks were detected, and a visual inspection afterwards showed no signs of leaking.

The next day dynamic testing occurred with the completely BlueFish prototype. The goal of this testing was to see how the control system responded to being towed in the water, if it corrected within the error outlined in the engineering specifications, and we could push settings to the BlueFish while it was active. Seven tests occurred where data was collected and analyzed, as can be seen in Appendix AC. The testing environment was sub-optimal. Speed was held reasonably constant, but it was difficult to reach equilibrium with a short test distance. The results from testing are shown below; the data was logged in a CSV file through the Raspberry Pi.

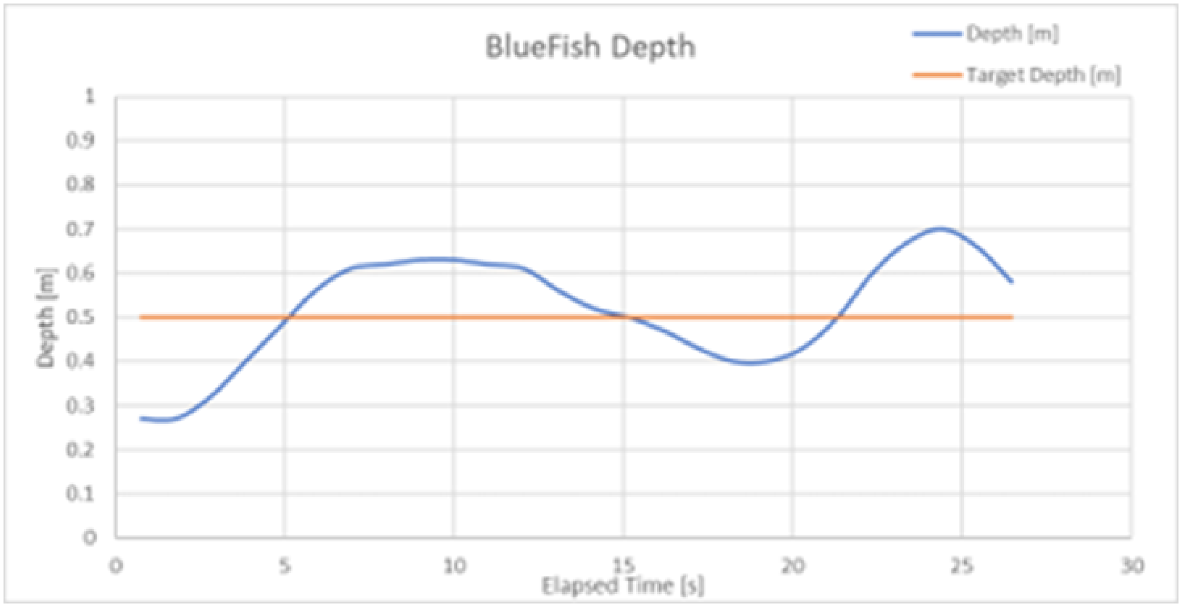


Figure 55: BlueFish data during dock testing in constant depth mode.

The first tests performed used constant depth mode. The Bottom Feeders input a set depth into Blue Command, which in-turn instructed the FishBrain to set a target depth of 0.5 m. In steady state, the depth was held between 0.3 and 0.7 m, with an average error of about 0.2 m. This error is less than the 0.25 m specified in the engineering requirements. Notably, this was achieved with no PID tuning, suggesting there is lots of room for improvement.

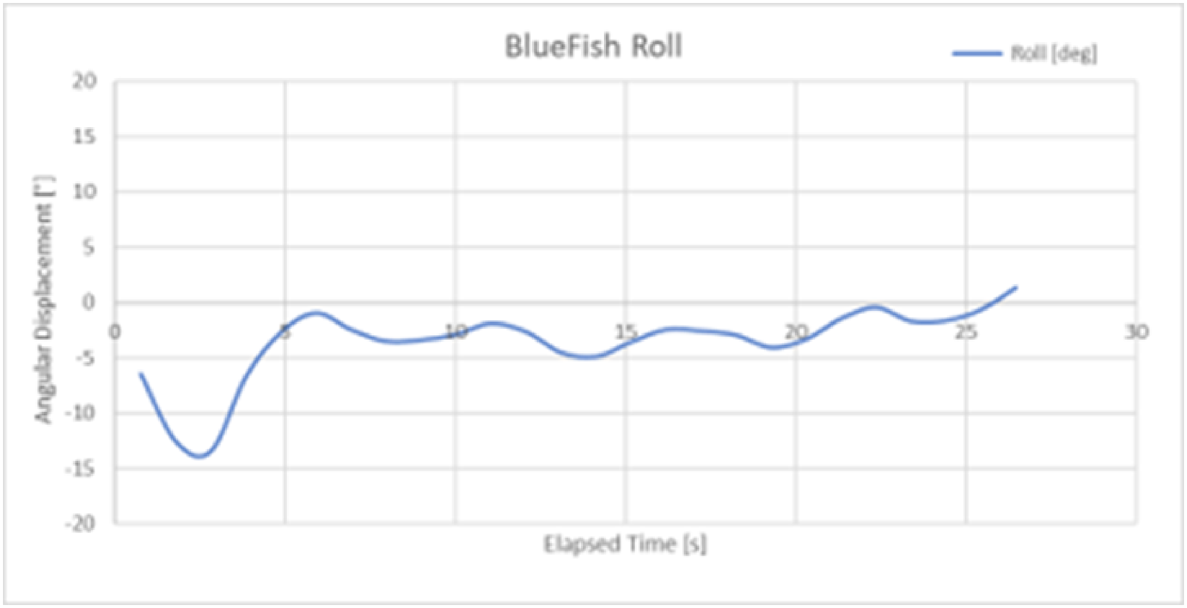


Figure 56: BlueFish roll data during dock testing in constant depth mode.

In Figure 56, the BlueFish's roll hovered between -1 degrees and -5 degrees. The observed offset was likely caused by towing the BlueFish slightly sideways from a dock. However, without tuning PID settings, the BlueFish reaches the (+/-) 3 degrees objective neglecting the offset. This pattern was consistent through testing when in steady state and will benefit from improved testing conditions and PID tuning.

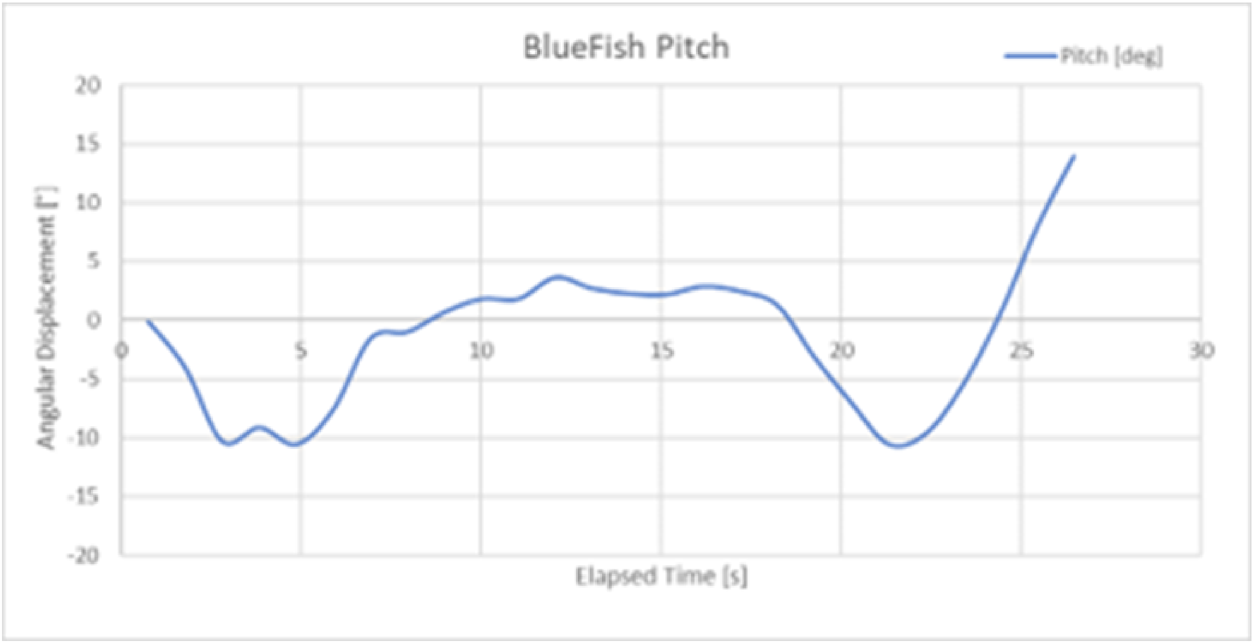


Figure 57: BlueFish pitch data in constant depth mode during dock testing.

This test also provided pitch results, including being lowered into the water in the first seven seconds and coming to a stop at around eighteen seconds. As stated earlier, it was difficult to keep a consistent walking speed with the BlueFish while pulling the tether along the dock and avoiding seaweed. The pitch stays between approximately -1 and 4 degrees for most of the test runs in steady state. If the middle portion of the data roughly represents steady state, the test was successful. It is important to note that conditions were far from ideal, and that with proper buoyancy tuning and testing on a boat, results would likely improve significantly. Additionally, PID tuning provides a lot of room for improvement, including the option to increase the magnitude and speed of corrections.

4.4. Future Work

The results from early testing are very promising. More testing will need to occur to continue tuning. Future tests should be performed in a boat, where speed can be kept constant, and the BlueFish can dive deeper.

4.4.1. Mechanical System

The mechanical system performs all the functions that we need for a prototype correctly. However, some last-minute recommendations from the client left the design in a more primitive state than previous. For rigidity in the 3D prints, the split nosecone and fishtail were merged. These will likely need to be split if vacuum casting is the preferred manufacturing method. When these parts are split, the mounts for the ping, lumen, and camera can be greatly simplified. Due to the manufacturing methods used by Blue Robotics manufacturer, the endcaps and wing mounting pieces can become one part, which will help with rigidity and cost. Internally, the Arduino should be removed as the navigator will become the new flight controller with the Raspberry Pi. The wires from the front of the BlueFish could be routed to the back to significantly decrease disassembly difficulty.

4.4.2. Control System

The gains should be adjusted as testing occurs, following what is recommended in Appendix I, and a second PID tuning test report should be made. The GUI has a some partially implemented functionality that would provide great benefit for future development and users. First, live data plotting needs to be fully implemented into the BlueCommand_main.py code (see Appendix Y). The code is roughly 90% complete but needs some finishing integration touches as it was originally developed to run on a separate thread from the GUI.

As mentioned before, the Python module used for the GUI included a library for the operation of a USB camera called QtMultimedia, which the Raspberry Pi was unable to identify. Further work is required for the integration of the focussing and zooming functionality and the debugging of the code seen in Appendix AB. There is another Python module applicable to the camera system called OpenCV, however, there was not enough time available to explore this new module. Additionally, one could rewrite the GUI, including the logging, plotting, and photographing functionality in the native language of the Qt library, that being C++, which could prove to be more reliable and would have a greater number of online resources and community support.

Lastly, there are a couple of known small changes that could be added to improve user experience:

- Occasionally, the GUI freezes when pushing the Standby settings to the BlueFish. This is likely a serial buffer that can be resolved by clearing both input and output buffers.
- Serial port designations can change for the Arduino, so a way to choose the serial port with the GUI would be beneficial.
- Currently, a prompt to calibrate the BlueFish is generated in the terminal when running the code before opening the Fish Command GUI is opened. However, it would be relatively easy to integrate a small pop-up to activate over the GUI when starting the code.
- It was noted that depth values are read as positive by the Arduino, but Fish Command passes a negative value. The sign of this needs to be reversed.
- Once code is complete, all code can be converted into a single executable (.exe) file.

5. Conclusion

This document outlined all work completed for the BlueFish project, including the project management, the design, analysis, and testing of the mechanical system, the development of a mechatronic system, the construction, testing, and evaluation of a functional BlueFish prototype. The prototype was found to meet the user requirements. The testing of the BlueFish prototype revealed that further debugging and refinement of the mechatronic system was required, however the mechanical system operated nominally. Overall, the project of producing an initial prototype of the BlueFish was successful and the prototype will see continued design and iteration of both the mechanical and mechatronic systems in the pursuit of a marketable product.

References

[1] S. University, "The NACA airfoil series," 1 February 2002. [Online]. Available: https://web.stanford.edu/~cantwell/AA200_Course_Material/The%20NACA%20airfoil%20series.pdf. [Accessed 10 February 2021].

[2] F. M. White, Fluid Mechanics, Rhode Island: McGraw-Hill Education, 2016.

[3] Wikipedia, "Falling Leaf," Wikipedia, 6 August 2015. [Online]. Available: https://en.wikipedia.org/wiki/Falling_leaf. [Accessed 22 April 2021].

[4] B. Robotics, "O-Ring Flange (3" Series)," Blue Robotics, 27 December 2017. [Online]. Available: https://bluerobotics.com/store/watertight-enclosures/3-series/o-ring-flange-3-series/. [Accessed 21 April 2021].

[5] B. Robotics, "M10 Cable Penetrator for 8mm Cable," Blue Robotics, 4 May 2016. [Online]. Available: https://bluerobotics.com/store/cables-connectors/penetrators/penetrator-10-25-a/. [Accessed 21 April 2021].

[6] B. Robotics, "Lithium-ion Battery (14.8V, 18Ah)," Blue Robotics, 11 July 2019. [Online]. Available: https://bluerobotics.com/store/comm-control-power/powersupplies-batteries/battery-li-4s-18ah-r3/. [Accessed 15 April 2021].

[7] I. W. R. S. C. Inc., "Standard Wire Rope Thimble," Industrial Wire Rope Supply Company Inc., 15 January 2015. [Online]. Available: https://industrialrope.com/shop/1-8-standard-thim18/. [Accessed 21 April 2021].

[8] BlueRobotics, "Low-Light HD USB Camera," BlueRobotics, 28 August 2017. [Online]. Available: https://bluerobotics.com/store/sensors-sonars-cameras/cameras/cam-usb-low-light-r1/. [Accessed 14 April 2021].

[9] BlueRobotics, "Lumen Subsea Light for ROV/AUV," BlueRobotics, 10 July 2018. [Online]. Available: https://bluerobotics.com/store/thrusters/lights/lumen-r2-rp/. [Accessed 14 April 2021].

[10] Wikipedia, "Raspberry Pi," Wikipedia, 5 January 2021. [Online]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi. [Accessed 21 April 2021].

[11] B. Robotics, "Fathom Slim ROV Tether," Blue Robotics, 3 November 2017. [Online]. Available: https://bluerobotics.com/store/cables-connectors/cables/fathom-slim-nb-1p-26awg-r1/. [Accessed 21 April 2021].

[12] B. Robotics, "Fathom-X Tether Interface (FXTI)," Blue Robotics, 7 August 2018. [Online]. Available: https://bluerobotics.com/store/rov/bluerov2-accessories/fxti-asm-r1-rp/. [Accessed 21 April 2021].

[13] G. Robotics, "Arduino UNO Rev.3," 14 October 2013. [Online]. Available: https://www.generationrobots.com/en/401867-arduino-uno-rev-3.html. [Accessed 29 February 2021].

[14] PX4, "mRo Pixhawk Flight Controller (Pixhawk 1)," 19 February 2021. [Online]. Available: https://docs.px4.io/master/en/flight_controller/mro_pixhawk.html. [Accessed 29 February 2021].

[15] BlueRobotics, "Ping Sonar Altimeter and Echosounder," BlueRobotics, 29 January 2019. [Online]. Available: https://bluerobotics.com/store/sensors-sonars-cameras/sonar/ping-sonar-r2-rp/. [Accessed 14 April 2021].

[16] BlueRobotics, "Bar30 High-Resolution 300m Depth/Pressure Sensor," [Online]. Available: https://bluerobotics.com/store/sensors-sonars-cameras/sensors/bar30-sensor-r1/. [Accessed 15 April 2021].

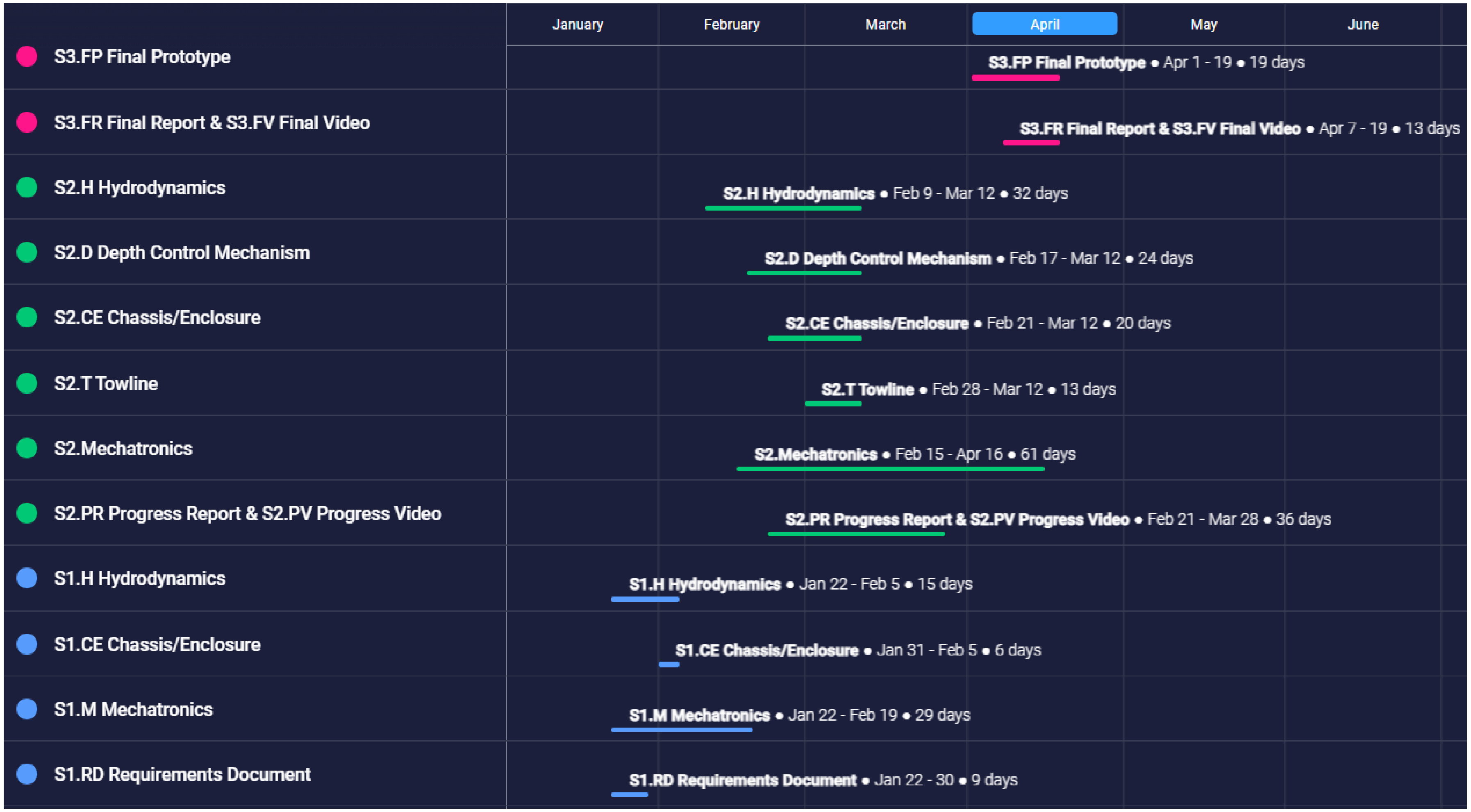
[17] Adafruit, "Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055," [Online]. Available: https://www.adafruit.com/product/2472#technical-details. [Accessed 16 April 2021].

[18] BlueRobotics, "Power Sense Module," [Online]. Available: https://bluerobotics.com/store/comm-control-power/elec-packages/psm-asm-r2-rp/. [Accessed 16 April 2021].

[19] D. R. Hobby, "CLS400MC Metal Gear standard 40kg Coreless Digital Servo," Doman RC Hobby, 15 February 2010. [Online]. Available: http://www.domanrchobby.com/content/?32.html. [Accessed 2021 April 14].

[20] "Lift/Lower Calculator," WorkSafe BC, [Online]. Available: http://worksafebcmedia.com/misc/calculator/llc/]. [Accessed 01 02 2021].

Appendix A – Gantt Chart for BlueFish Project



Gantt Chart for Sprints 1, 2, and 3.

Link to Monday.com Workspace: https://uvic410232.monday.com/users/sign_up?invitationId=13719070364424698000

Appendix B – Detailed Drawing Package

See the attached following pages.

Appendix C – Test Report No.1 – BlueFry I: Hydrodynamic Profile CFD Analysis

See the attached following pages.

Appendix D – Test Report No.2 – BlueFry I: Hydrodynamic Test

See the attached following pages.

Appendix E – Test Report No.3 – BlueFry II: NACA 0012 Hydrofoil CFD Test

See the attached following pages.

Appendix F – Test Report No.4 – BlueFry II: Diving Test

See the attached following pages.

Appendix G – Test Report No.5 – BlueFry III: Waterproof Validation Test

See the attached following pages.

Appendix H – Test Report No.6 – BlueFish I: Depth Control Test & PID Tuning

See the attached following pages.

Appendix I – Bill of Materials & Cost Breakdown

Item	MSRP (USD)	Discount	Final Price	Qty.	Subtotal
Electronic Components					
Servo Motor	$200.00	Yes	$80.00	2	$160.00
Raspberry Pi	$30.00	No	$30.00	1	$30.00
Navigator	$40.00	Yes	$16.00	1	$16.00
Bar30	$72.00	Yes	$28.80	1	$28.80
FXTI	$85.00	Yes	$34.00	1	$34.00
BLART to USB	$31.00	Yes	$12.40	1	$12.40
Leak sensors	$29.00	Yes	$11.60	1	$11.60
I^{2}C Level Converter	$15.00	Yes	$6.00	1	$6.00
Blue Robotics Components					
Penetrators	$4.00	Yes	$1.60	8	$12.80
Cast Acrylic Tube	$86.00	Yes	$34.40	1	$34.40
Vent	$9.00	Yes	$3.60	1	$3.60
O-Ring Flange	$24.00	Yes	$9.60	2	$19.20
Custom Components					
Fishtail	$30.00	No	$30.00	1	$30.00
Nosecone	$25.00	No	$25.00	1	$25.00
Static Fins	$7.50	No	$7.50	4	$30.00
Hydrofoils	$11.60	No	$11.60	2	$23.20
Foil Mounts	$3.70	No	$3.70	2	$7.40
Endcaps	$15.90	No	$15.90	2	$31.80
Wing retainers	$1.88	No	$1.88	8	$15.04
Tether Rack	$13.10	No	$13.10	1	$13.10
E-Tray	$3.60	No	$3.60	1	$3.60

E-Tray Holder	$2.80	No	$2.80	4	$11.20
McMaster-Carr Components					
Thimble	$3.12	No	$3.12	1	$3.12
M2 Fasteners	$0.06	No	$0.06	6	$0.36
M3 Fasteners	$0.24	No	$0.24	64	$15.41
M3 Nylock Nuts	$0.10	No	$0.10	20	$1.90
M4 Shoulder Screw	$2.81	No	$2.81	1	$2.81
#4 Wood Screws	$0.07	No	$0.07	4	$0.27
Stand-offs	$10.82	No	$10.82	1	$10.82
Base Production Model Total					***$579.93***
Extra Components for The BlueFish Prototype & Photomosaic Production Model					
Ping Mount	$5.50	No	$5.50	1	$5.50
Front Mount	$6.80	No	$6.80	1	$6.80
Battery	$289.00	Yes	$115.60	1	$115.60
Tether	$350.00	Yes	$140.00	1	$140.00
Ping	$279.00	Yes	$111.60	1	$111.60
Camera	$99.00	Yes	$39.60	1	$39.60
Camera housing	$15.90	No	$15.90	1	$15.90
Camera cover	$12.00	Yes	$4.80	1	$4.80
Lumen	$115.00	No	$115.00	1	$115.00
Arduino	$14.99	No	$14.99	1	$14.99
Arduino shield	$4.50	No	$4.50	1	$4.50
20 AWG wire	$10.00	No	$10.00	1	$10.00
O-ring	$0.33	No	$0.33	1	$0.33
Photomosaic Production Model Total					***$879.13***
Final Prototype Total					***$1,164.55***

Appendix J – BlueFish Power Requirements and Calculations

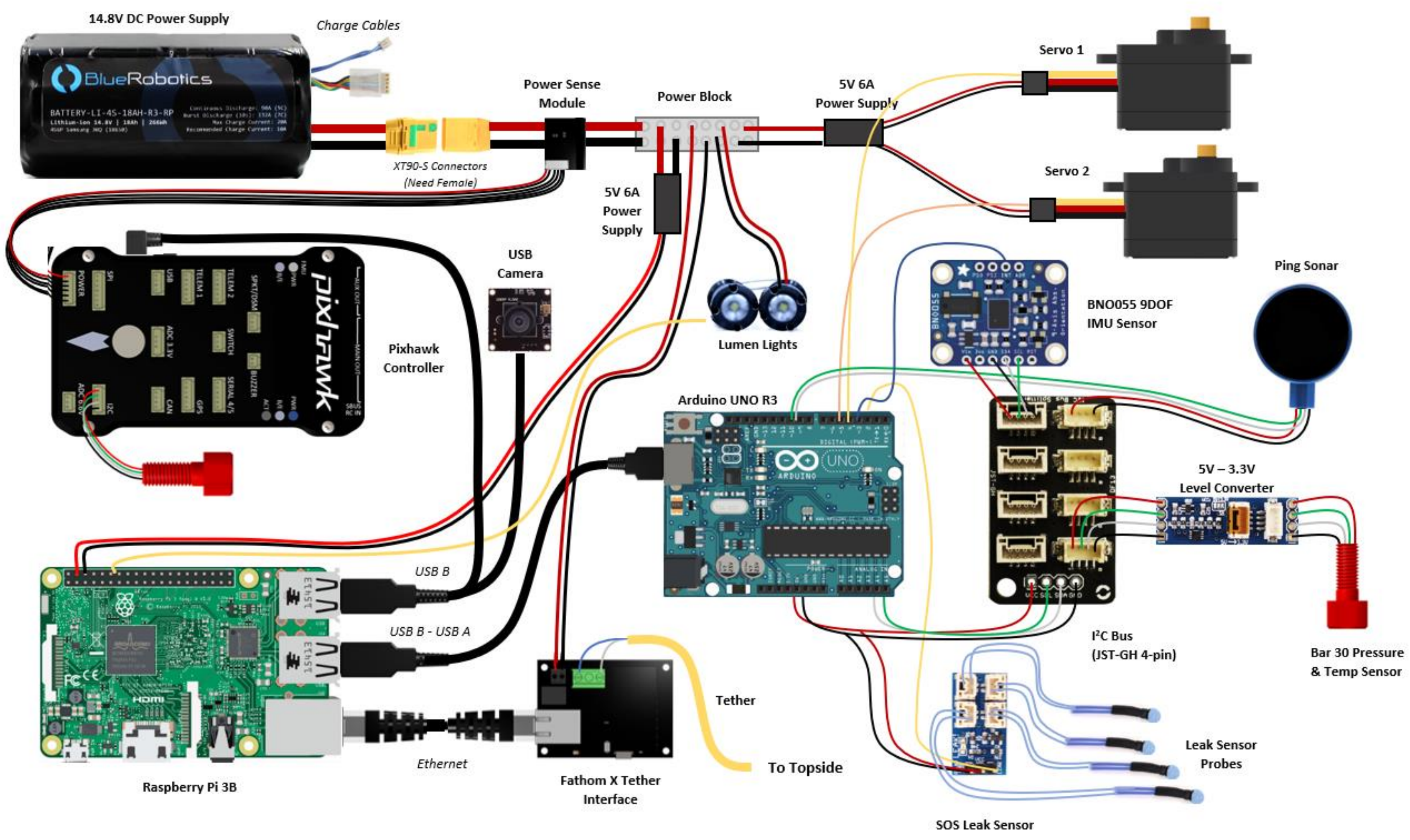
	SENSORS					ACTUATORS		COMMUNICATION			
	Bar 30	Leak	Ping	BNO055	Camera	Lumen	Servo 1	Pixhawk 4	Arduino UNO	Raspberry Pi 3B	FXTi
Number of Components	2	1	1	1	1	1	2	1	1	1	1
Power Source	Arduino 5V	Arduino 5V	Arduino 5V	Arduino 5V	Raspberry Pi	Battery	Power Supply	Rasberry Pi	Raspberry Pi	Power Supply	Battery
Min Input Voltage (V)	2.5	3.3	5	3.3	5	7	4.8	4.9	6	4.75	7
Max Input Voltage (V)	5.5	5	5.5	5	5	48	6	5.5	20	5.25	28
Ideal Voltage (V)	5	5	5	5	5	15	5	5	7	5	15
Max Current Draw (mA)	1.25	20	150	20	220	2143	700	500	1000	2500	357
Typical Current Draw (mA)	1	10	100	13	100	500	100	200	50	400	167
Total Max Current Draw (mA)	3	20	150	20	220	2143	1400	500	1000	2500	357
Total Typical Current Draw (mA)	2	10	100	13	100	500	200	200	50	400	167
Typical Wattage (mW)	10	50	500	65	500	7500	1000	1000	350	2000	2500
Notes	*Typical current not specified, est 1 mA	*Typical current not specified, est 10 mA	*Max current not specified, est 150 mA	*Max current not specified, est 20 mA	*Typical current not specified, est 100 mA	*15A/V at 100% duty. Typical assumes 50% @ 15V	*Typical draw is based on running @ no load	*500 mA max on USB	*400 mA max on USB, 1000 mA max on Vin (regulator) **typical current draw assumes power draw from sensors etc	*2500 mA max from Psupply **400 mA barebones draw 1200 mA max for total USB	*Current is based on max power draw of 2.5W
Type	I2C	Digital	Serial	I2C	Serial	Digital PWM	Digital PWM	Serial	Misc		Ethernet
Voltage Supply	5V Level Converter	Arduino 5V	Arduino 5V	Arduino 5V	5	Raspberry Pi	Arduino 5V	Rasberry Pi	Max 20 mA/pin	1200 mA max for all USB	-
Min Voltage (V)	2.5	3.3	3.3	3.3	5	3	-	4.75	3.3		-
Max Voltage (V)	3.6	5	5	5	5	48	-	5.25	5		-
Connector	DF13-4	0.1"	0.1"	0.1"	JST-PH/USB	Wire	0.1"	USB			Tether
Wires	4	2	4	4	N/A	3	3	N/A			-
Wire Size (ga)	22-24	22-24	24	22-24	N/A	22	22-24	N/A			-

$$Sum\ of\ Typical\ Wattage = 15{,}475\ mW = \mathbf{15.475\ W}$$

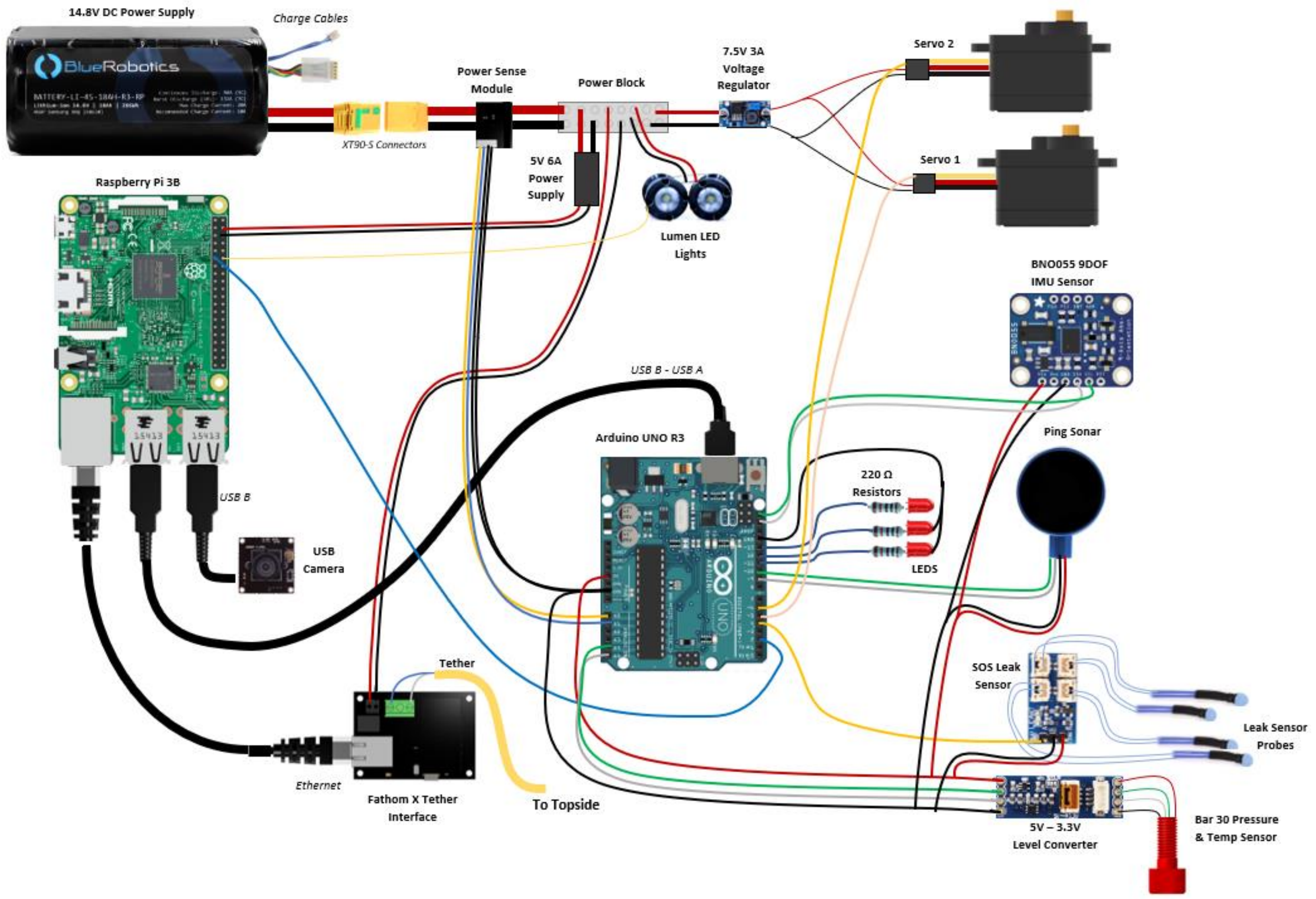
$$Battery\ Capacity\ (W*hr) = Battery\ Capacity\ (A*hr) * Nominal\ Voltage = 18\ A*hr * 14.8\ V = \mathbf{266.4\ W*hr}$$

$$Battery\ Charge\ Lifetime\ (hr) = \frac{Battery\ Capacity\ (W*hr)}{Sum\ of\ Typical\ Wattage\ (W)} = \frac{266.4\ W*hr}{15.475\ W} = \mathbf{17.21\ hr}$$

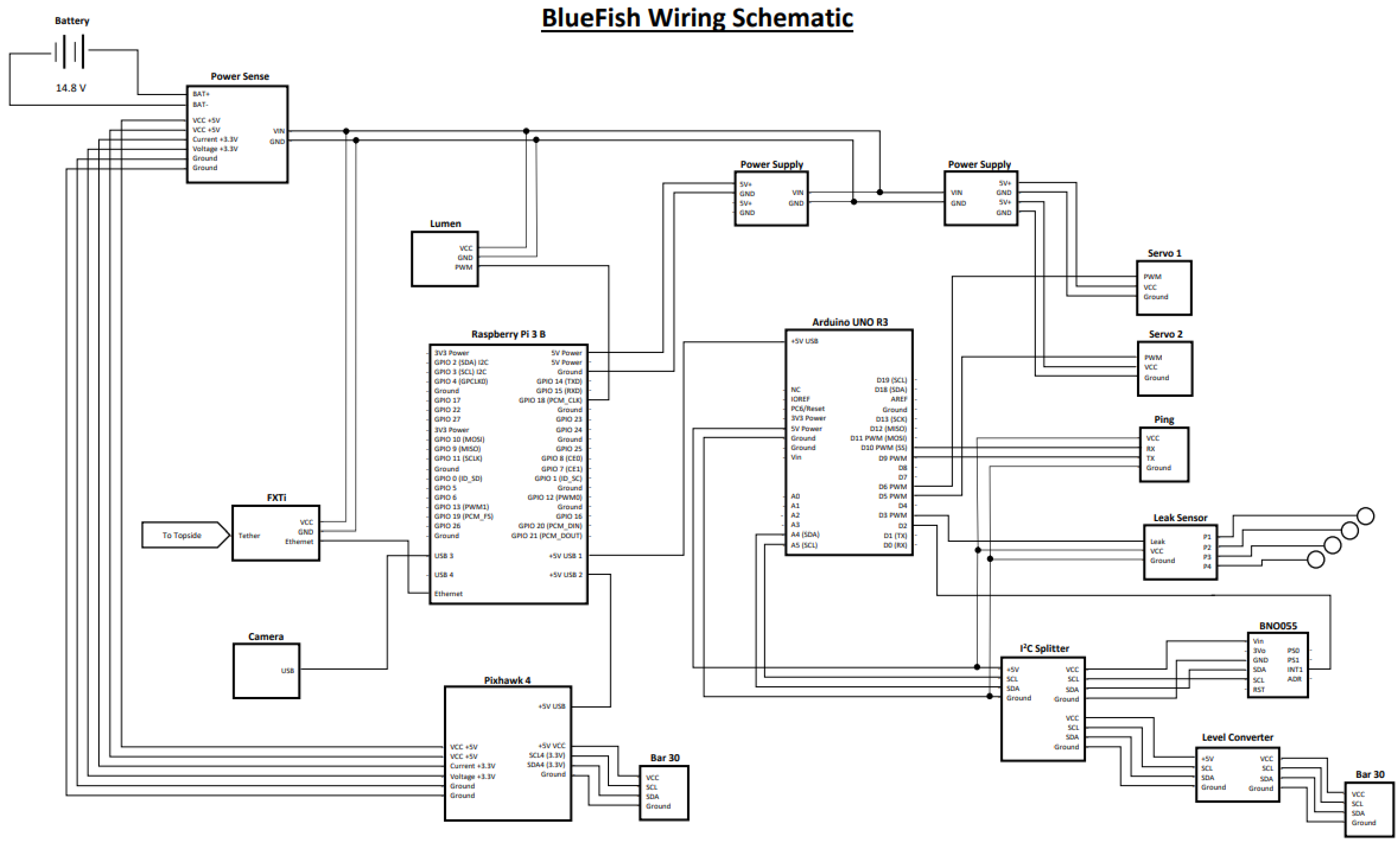
Appendix K – FishGuts I Component Connection Diagram



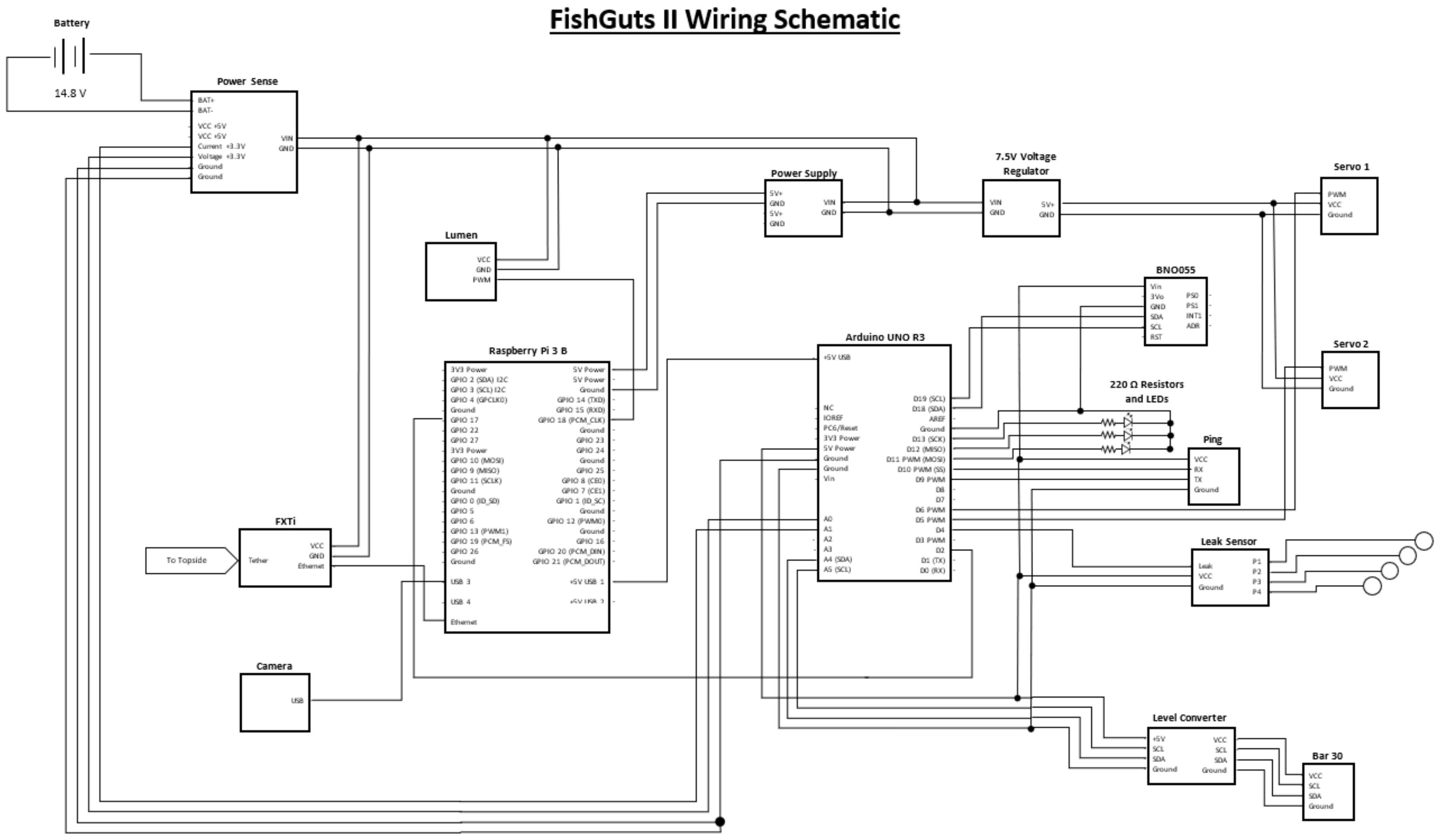
Appendix L – FishGuts II Component Connection Diagram



Appendix M – FishGuts I Wiring Schematic



Appendix N – FishGuts II Wiring Schematic

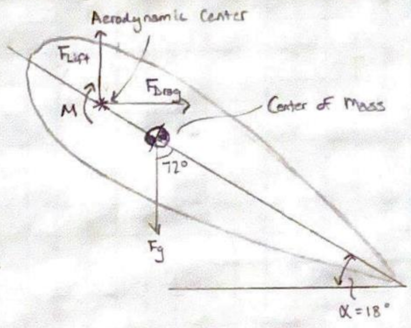


Appendix O – NACA 0012 Hydrofoil Loading Conditions Sample Calculations

NACA 0012 Hydrofoil Loading Conditions (Per Hydrofoil)

Input Parameters

- Density of Water, ρ_w: 997 kg/m^3
- Flow Velocity, u: 1.0 m/s
- Wing Span, S: 0.115m
- Wing Chord, C: 0.09m
- Gravity, g: 9.81 m/s^2
- Stall Angle, α: 18° or 0.314159 rad
- Planform Efficiency Factor, e: 1
- Center of Mass, COM: 0.03899m (from front)
- Object Volume, V: $6.5163 \times 10^{-5} m^3$
- Material Density, ρ_M: 1200 kg/m^3
- 2D Coefficient of Lift at α, C_ℓ: 1.25
- Moment Coefficient, C_m: 0.008
- Approx. 3D coefficient of Drag at 0°, C_{D_0}: 0.01

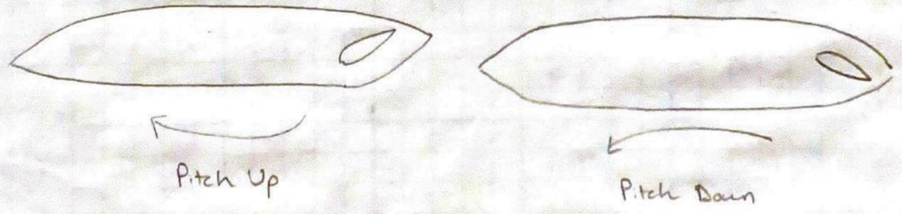


Calculations

- Wing Aspect Ratio, $AR = \frac{S^2}{S \cdot C}$ or $\frac{S}{C} = \underline{1.2778}$
- 3D Coefficient of Lift at α, $C_L = C_\ell \left(\frac{AR}{AR+2}\right)\alpha = \underline{0.1531}$
- 3D Coefficient of Induced Drag at α, $C_{D_i} = \frac{C_L^2}{\pi \cdot AR \cdot e} = \underline{0.00584}$
- 3D Coefficient of Total Drag at α: $C_D = C_{D_i} + C_{D_0} = \underline{0.01584}$
- Dynamic Pressure, $q = \frac{\rho_w u^2}{2} = \underline{498.5 \, Pa}$
- Pitching Moment, $M = C_m \cdot q \cdot S \cdot C = \underline{\pm 0.04128 \, Nm \; (5.8451 \, oz\text{-}in)}$
- Lift Force, $F_L = C_L \cdot q \cdot S \cdot C = \underline{0.7898 \, N}$
- Drag Force, $F_D = C_D \cdot q \cdot S \cdot C = \underline{0.0817 \, N}$

- Moment Arm Length of COM w.r.t. AC: $L_{M,COM-AC} = COM - AC = \underline{0.01649 m}$
- Mass of Object, $m = \rho_M V = \underline{0.0782 \, kg}$
- Weight of Object, $F_g = mg = \underline{0.7671 \, N}$
- Moment due to Gravity, $M_g = L_{M,COM-AC} \cdot F_g \sin(90° - \alpha) = \underline{0.0120 \, Nm}$
- Maximum Loading Conditions:
  - ↳ Pitching Up: $M_{PU} = M_g + M = 0.0533 \, Nm \; (7.5488 \, oz\text{-}in)$ ← Max. loading Condition
  - ↳ Pitching Down: $M_{PD} = M_g - M = 0.0292 \, Nm \; (4.1415 \, oz\text{-}in)$

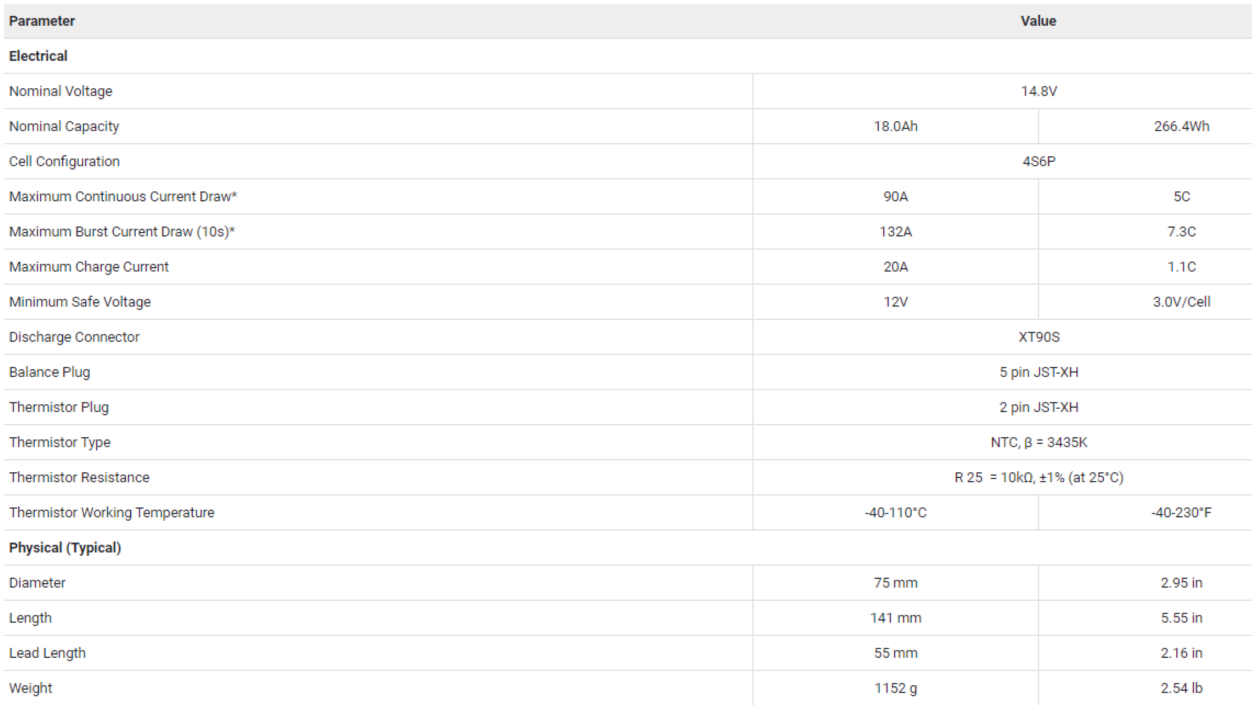
Note: Pitching up refers to the pitching up of the BlueFry / BlueFish and NOT the hydrofoil.



Appendix P – O-Ring Calculations

O-Ring Specifications		
Parameter	**Value**	**Unit**
Inner Diameter (ID)	48	m
Outer Diameter (OD)	51	m
Radial Cross-Section (CS)	1.5	m
Material	BUNA 70A	-
Recommended Compression		
Parameter	**Value**	**Unit**
Parker O-Ring	20	%
10% Compression Force	2.5	$^{lb}/_{in\ seal}$
20% Compression Force	6	$^{lb}/_{in\ seal}$
30% Compression Force	15	$^{lb}/_{in\ seal}$
Compression Limits		
Parameter	**Value**	**Unit**
Minimum	1.4 (13.33)	mm (%)
Mean	1.5 (20.00)	mm (%)
Maximum	1.6 (26.67)	mm (%)
Compression Force		
Parameter	**Value**	**Unit**
10% at 155.5 mm Circumference	68.1 (15.31)	N (lb)
20% at 155.5 mm Circumference	163.4 (36.73)	N (lb)
30% at 155.5 mm Circumference	408.5 (91.84)	N (lb)

Appendix Q – Battery Pack Specifications Sheet [6]



Parameter	Value	
Electrical		
Nominal Voltage	14.8V	
Nominal Capacity	18.0Ah	266.4Wh
Cell Configuration	4S6P	
Maximum Continuous Current Draw*	90A	5C
Maximum Burst Current Draw (10s)*	132A	7.3C
Maximum Charge Current	20A	1.1C
Minimum Safe Voltage	12V	3.0V/Cell
Discharge Connector	XT90S	
Balance Plug	5 pin JST-XH	
Thermistor Plug	2 pin JST-XH	
Thermistor Type	NTC, β = 3435K	
Thermistor Resistance	R 25 = 10kΩ, ±1% (at 25°C)	
Thermistor Working Temperature	-40-110°C	-40-230°F
Physical (Typical)		
Diameter	75 mm	2.95 in
Length	141 mm	5.55 in
Lead Length	55 mm	2.16 in
Weight	1152 g	2.54 lb

BlueRobotics
BATTERY-LI-4S-18AH-R3-RP
Lithium-ion 14.8V | 18Ah | 266Wh
4S6P Samsung 30Q (18650)
Continuous Discharge: 90A (5C)
Burst Discharge (10s): 132A (7C)
Max Charge Current: 20A
Recommended Charge Current: 10A

Appendix R – Servo Motor Specifications Sheet [19]

Item No.:DM-CLS400MD

Art No.:5432408

Brief: metal gears,coreless motor,40kg digital servo,2 ball bearings,standard servo size, output shaft 25T

Application:rc car,rc plane,rc helicopter,robot

Item No.	DM-CLS400MD
Operating Voltage:	6V~7.4V
Speed:	0.15 sec/60deg (6.0v) 0.13sec/60deg (7.4v)
Torque:	38.8kg.cm (6.0v) 40.6kg.cm (7.4v)
Size:	40mmX20mmX41mm
Rotation angle:	180degree(PWM500-2500us)
Weight:	76gram
Motor:	Coreless
Gear Train:	Metal(copper gear)
Ball Bearing:	2BB
Dead band width:	2usc
Interface:	(JR)
Wire length:	30cm
Type:	Digital,Aluminium heat sink,Coreless

High presicion metal gears made by CNC;

Select coreless motor,could compare with import one;

1million life Potentiometer;

expert and experienced in circuit for 8 years;

TOP choice on case plastic,and best mold manufacture

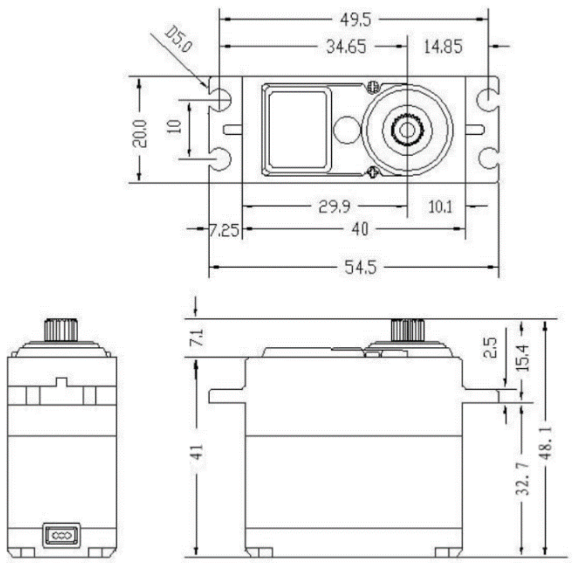
Standard Size,Coreless,digital,Titanium gears,25T,2BB

Futaba,JR,SANWA,Hitec compatible.

Waterproof

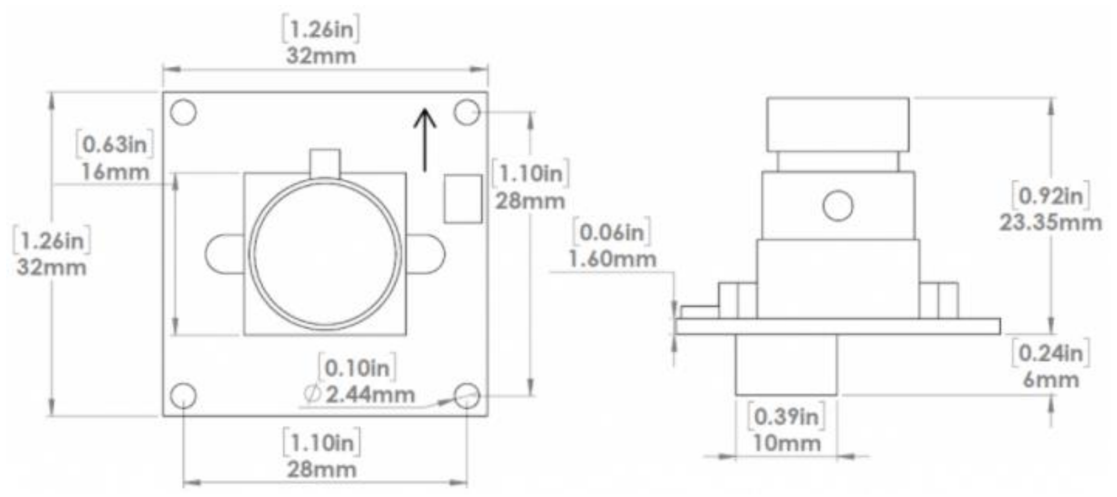
Install with 6pcs screws to make sure it is strong enough,

Aluminium middle case for heatsink

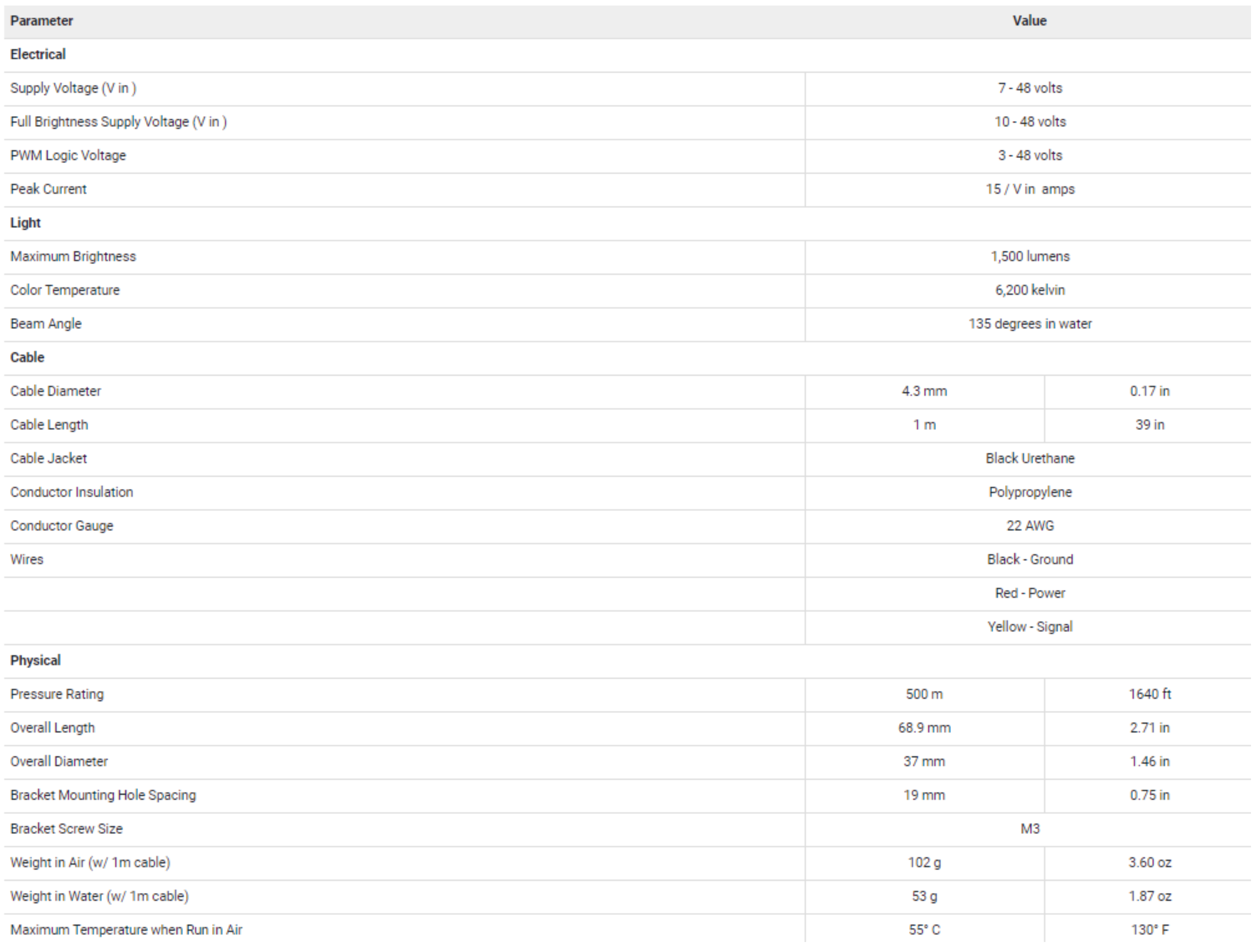


Appendix S – Low-Light HD USB Camera Specifications Sheet [8]

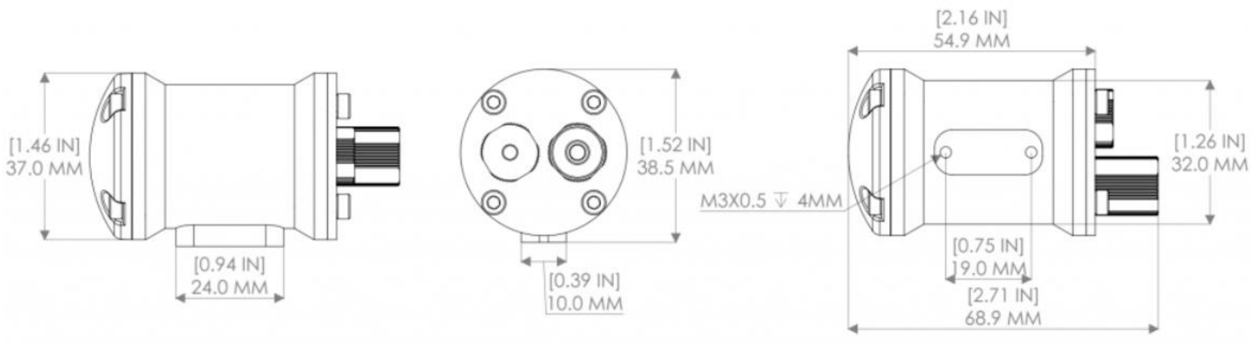
Parameter	Value
Physical	
Camera PCB Dimensions	32mm x 32mm
Mounting Hole Spacing	28mm x 28mm
Connector	JST-PH to USB
Performance	
Field of View (Horizontal)	80°
Field of View (Vertical)	64°
Focal Length	2.97 mm
Format	1/2.9"
Distortion	1%
Resolution	2.24 MP
Standard	1080p
Compression format	H.264 / MJPEG / YUV2 (YUYV)
Working Temperature	-20-75°C
Minimum Illumination	0.01 lux
Sensitivity	5.0V/lux-sec@550nm
Electrical	
Supply Voltage	5 volts
Max Power Draw	220mA



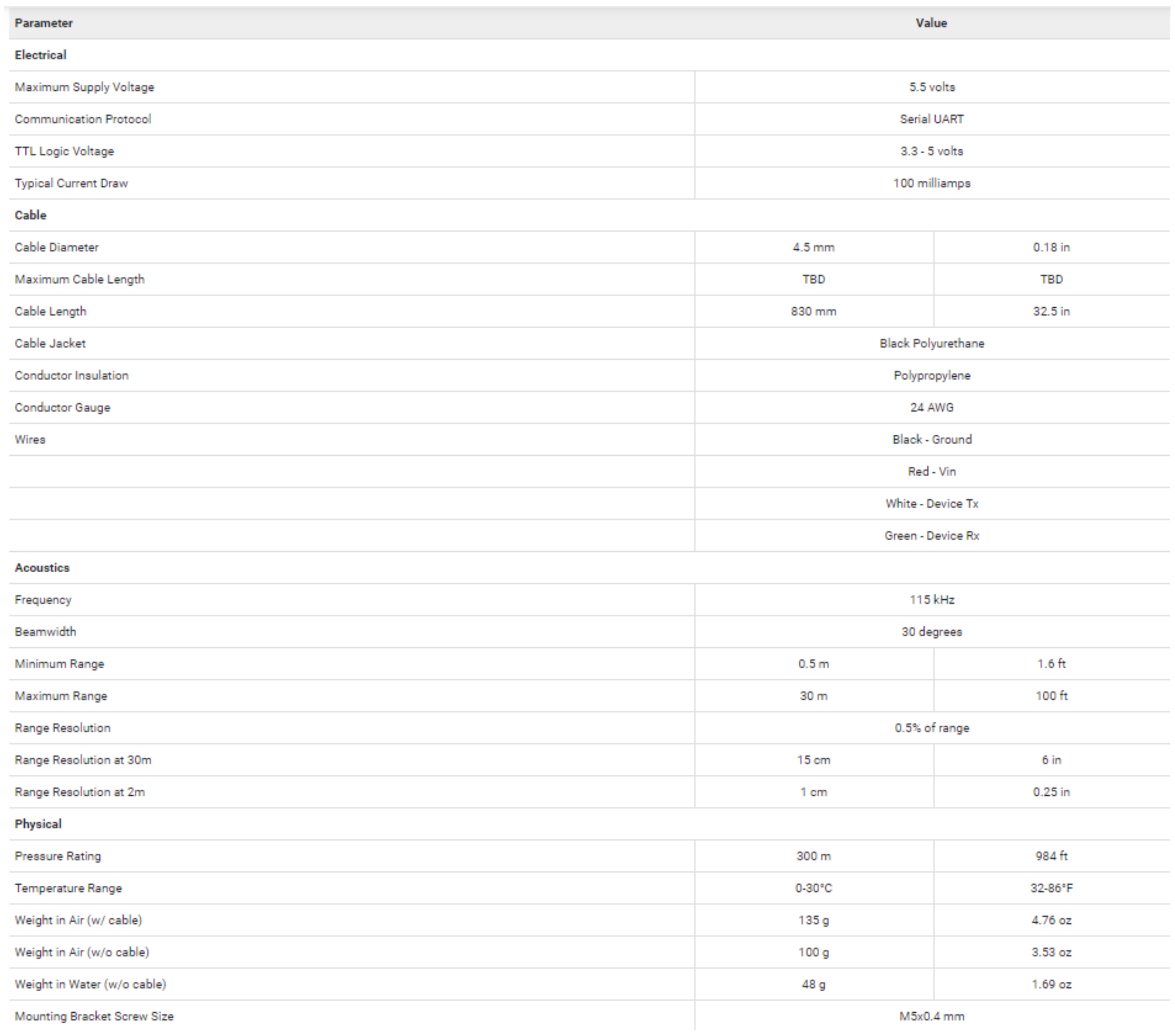
Appendix T – Lumen R2 Subsea Light Specifications Sheet [9]



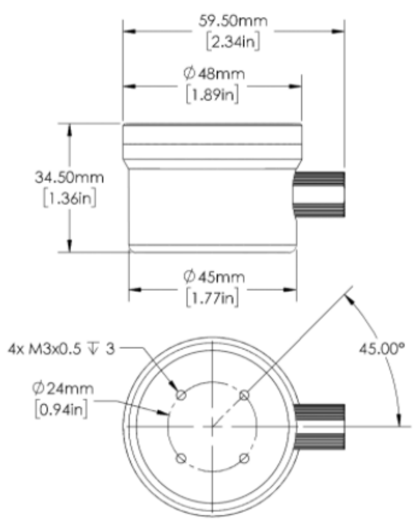
Parameter	Value	
Electrical		
Supply Voltage (V in)	7 - 48 volts	
Full Brightness Supply Voltage (V in)	10 - 48 volts	
PWM Logic Voltage	3 - 48 volts	
Peak Current	15 / V in amps	
Light		
Maximum Brightness	1,500 lumens	
Color Temperature	6,200 kelvin	
Beam Angle	135 degrees in water	
Cable		
Cable Diameter	4.3 mm	0.17 in
Cable Length	1 m	39 in
Cable Jacket	Black Urethane	
Conductor Insulation	Polypropylene	
Conductor Gauge	22 AWG	
Wires	Black - Ground	
	Red - Power	
	Yellow - Signal	
Physical		
Pressure Rating	500 m	1640 ft
Overall Length	68.9 mm	2.71 in
Overall Diameter	37 mm	1.46 in
Bracket Mounting Hole Spacing	19 mm	0.75 in
Bracket Screw Size	M3	
Weight in Air (w/ 1m cable)	102 g	3.60 oz
Weight in Water (w/ 1m cable)	53 g	1.87 oz
Maximum Temperature when Run in Air	55° C	130° F

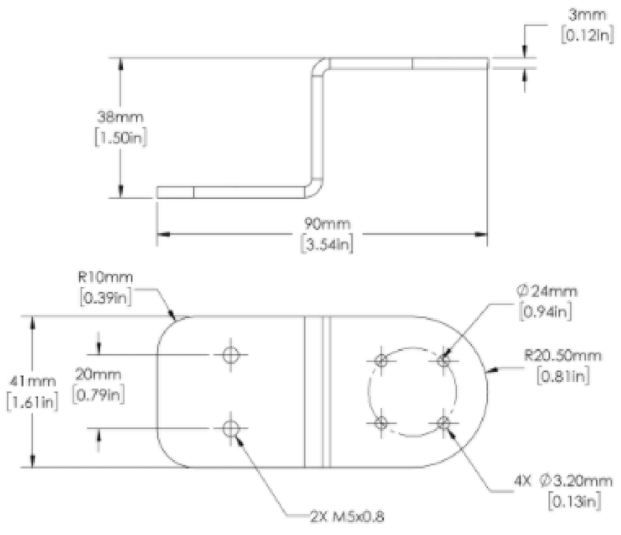


Appendix U – PING Sonar Specifications Sheet [15]

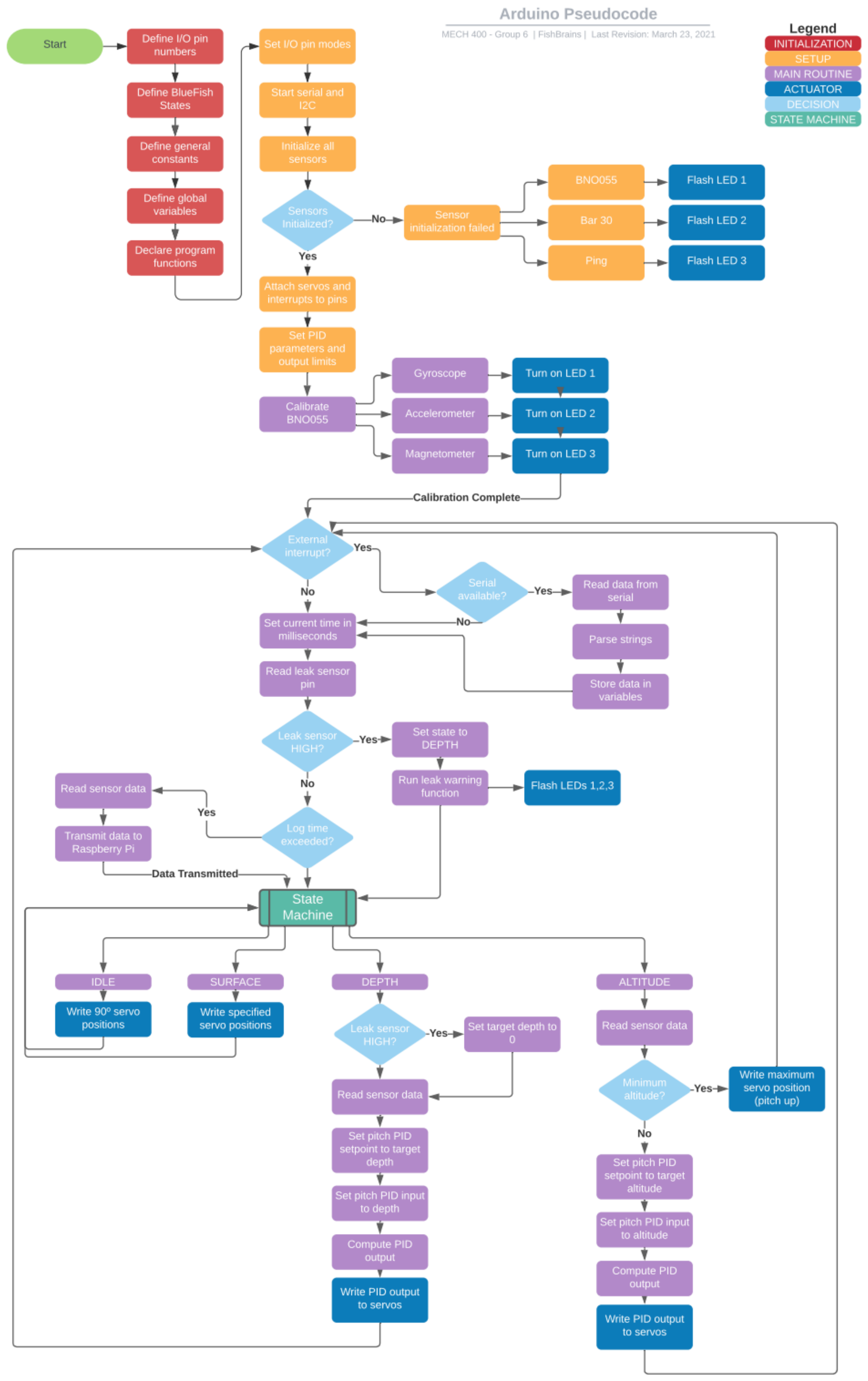


Parameter	Value	
Electrical		
Maximum Supply Voltage	5.5 volts	
Communication Protocol	Serial UART	
TTL Logic Voltage	3.3 - 5 volts	
Typical Current Draw	100 milliamps	
Cable		
Cable Diameter	4.5 mm	0.18 in
Maximum Cable Length	TBD	TBD
Cable Length	830 mm	32.5 in
Cable Jacket	Black Polyurethane	
Conductor Insulation	Polypropylene	
Conductor Gauge	24 AWG	
Wires	Black - Ground	
	Red - Vin	
	White - Device Tx	
	Green - Device Rx	
Acoustics		
Frequency	115 kHz	
Beamwidth	30 degrees	
Minimum Range	0.5 m	1.6 ft
Maximum Range	30 m	100 ft
Range Resolution	0.5% of range	
Range Resolution at 30m	15 cm	6 in
Range Resolution at 2m	1 cm	0.25 in
Physical		
Pressure Rating	300 m	984 ft
Temperature Range	0-30°C	32-86°F
Weight in Air (w/ cable)	135 g	4.76 oz
Weight in Air (w/o cable)	100 g	3.53 oz
Weight in Water (w/o cable)	48 g	1.69 oz
Mounting Bracket Screw Size	M5x0.4 mm	

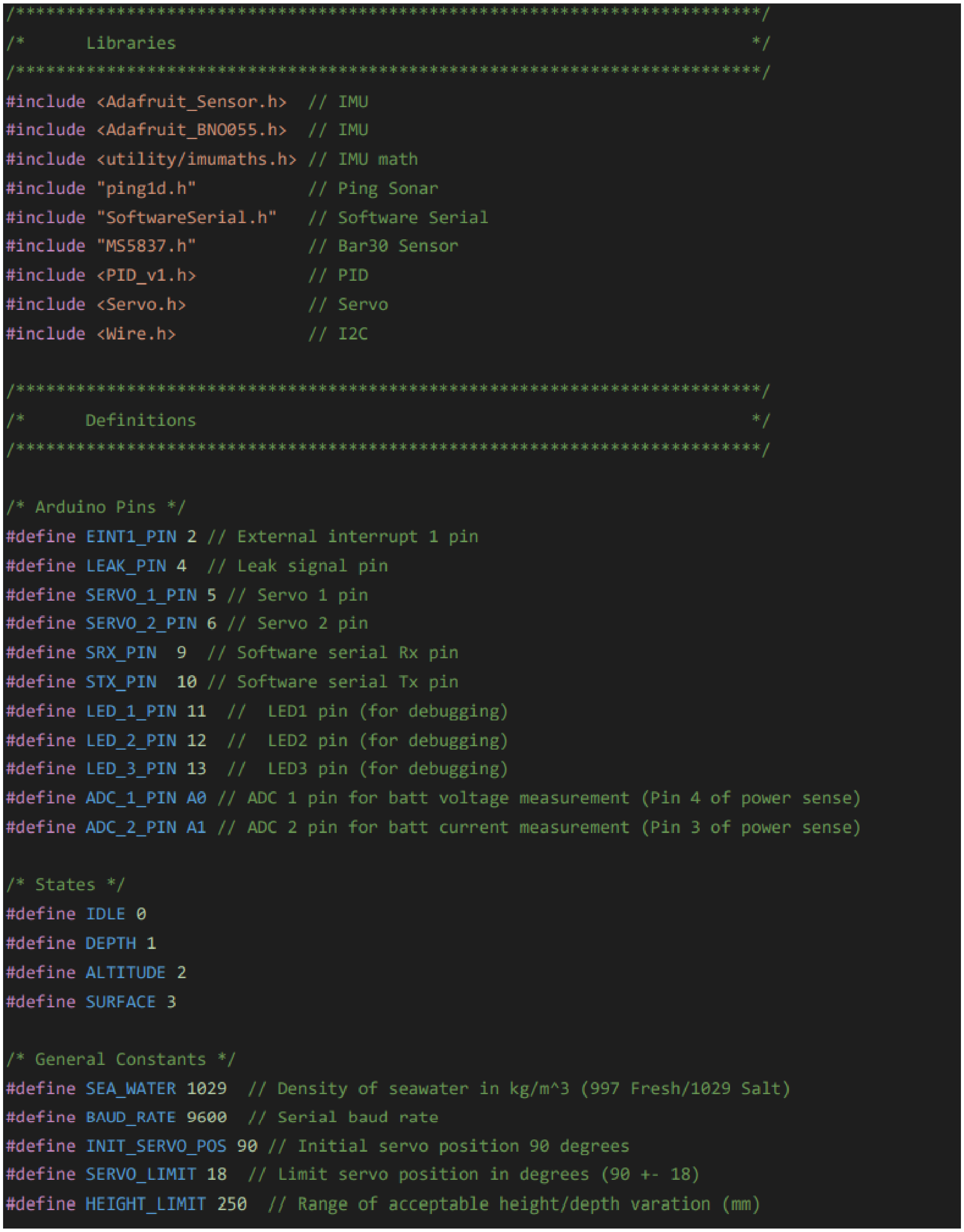




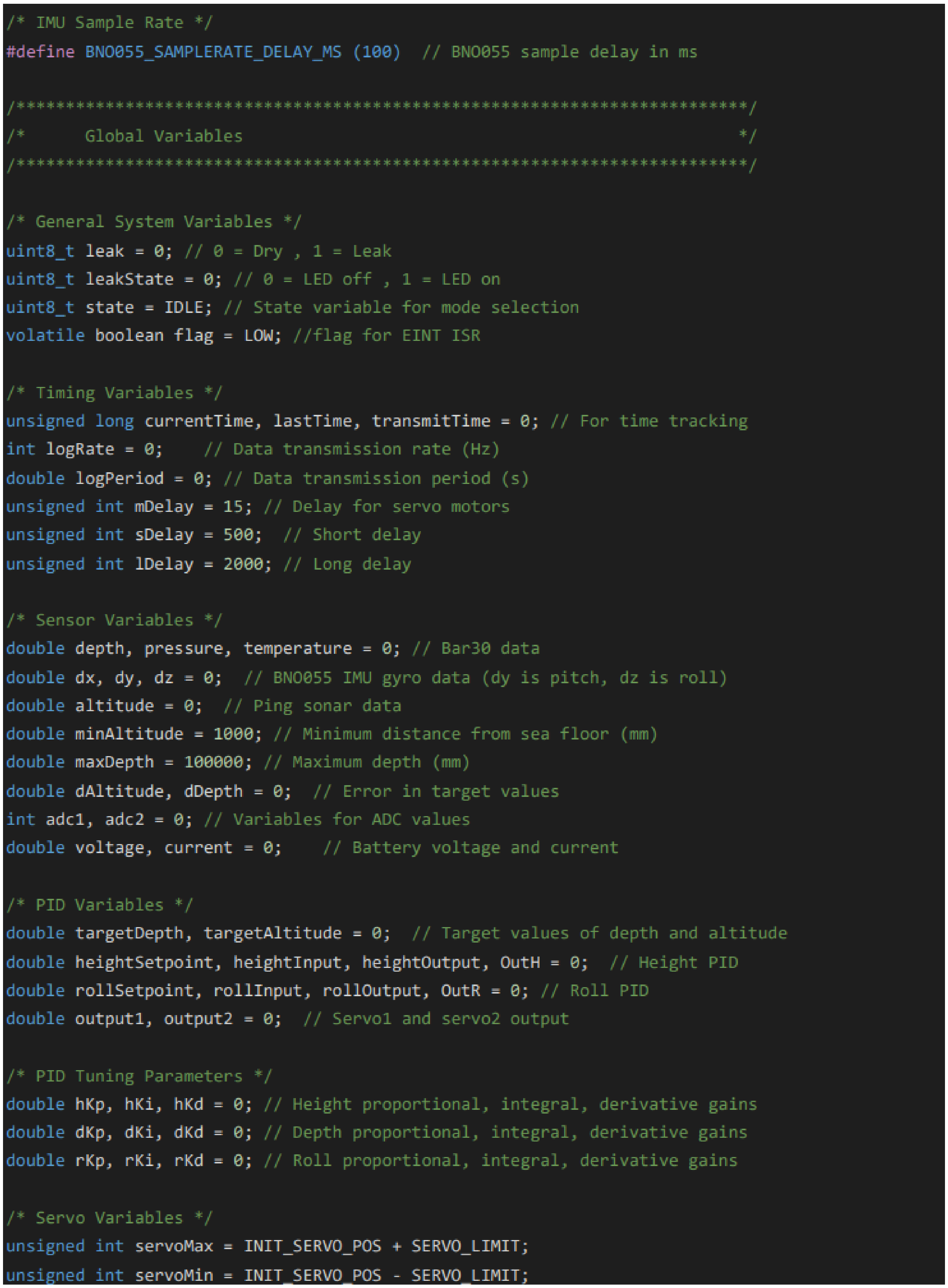
Appendix V – Arduino Pseudocode



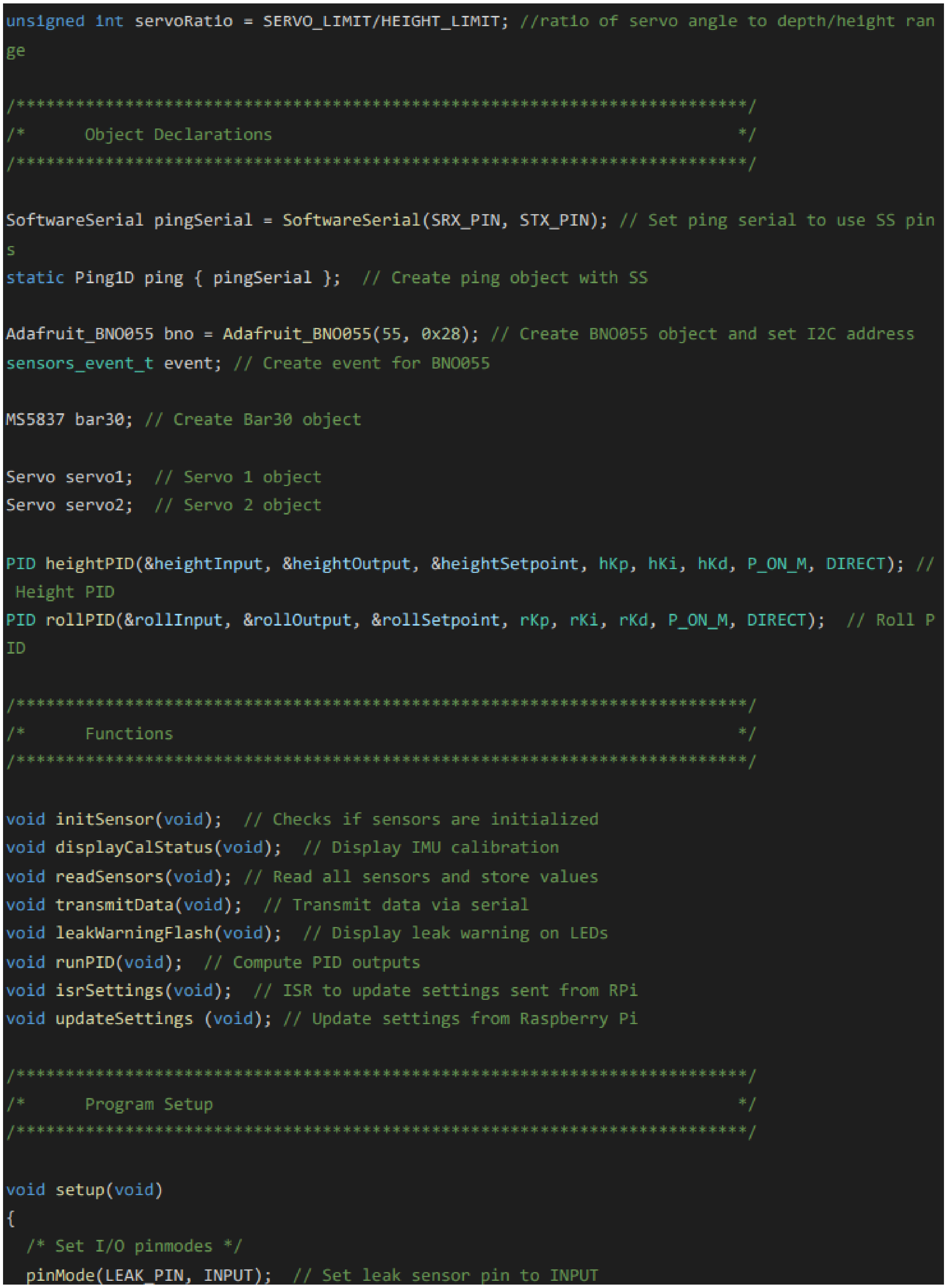
Appendix W – Arduino Firmware Code



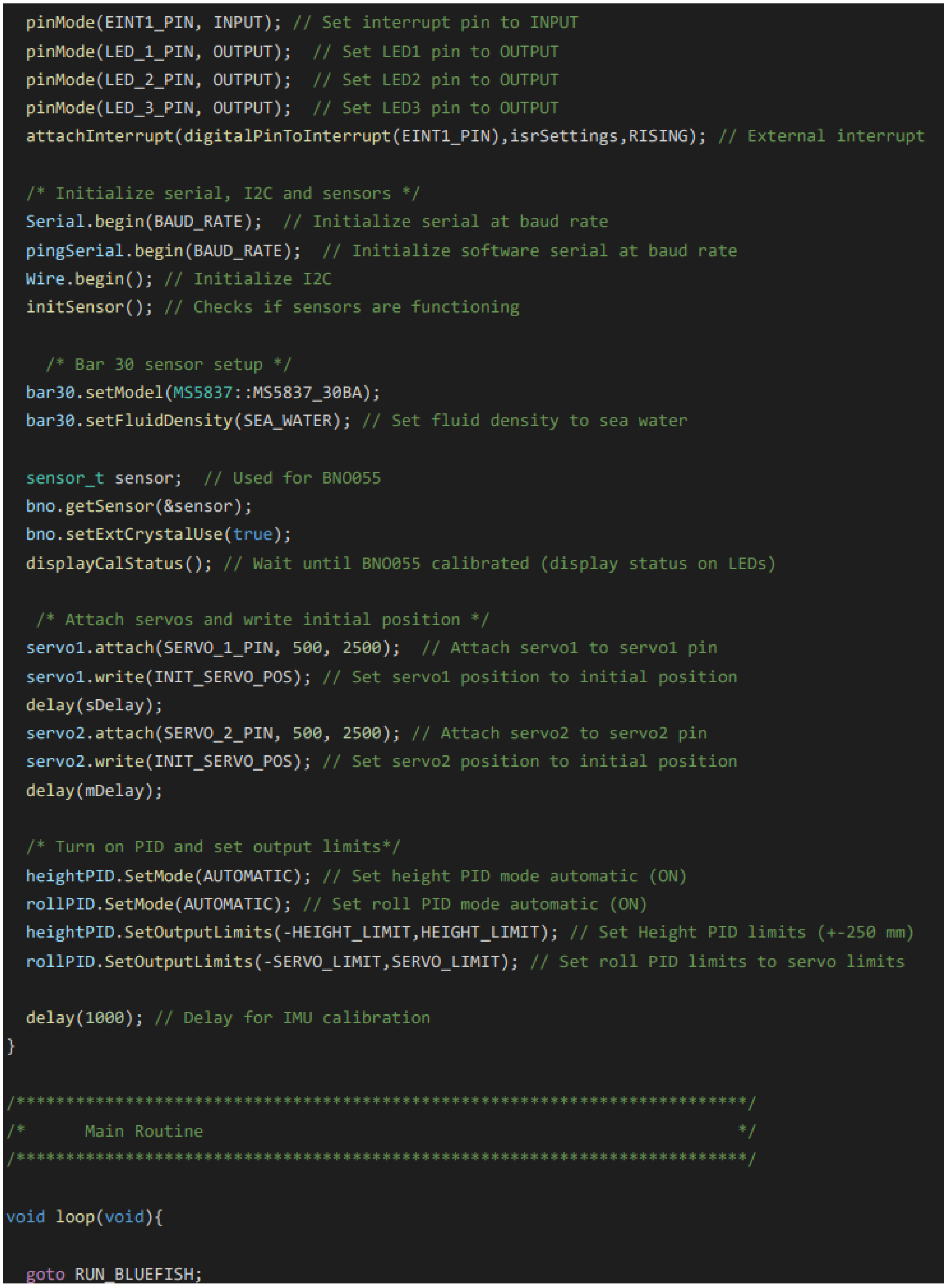
```
/**************************************************************************/
/*      Libraries                                                         */
/**************************************************************************/
#include <Adafruit_Sensor.h>   // IMU
#include <Adafruit_BNO055.h>   // IMU
#include <utility/imumaths.h> // IMU math
#include "ping1d.h"            // Ping Sonar
#include "SoftwareSerial.h"    // Software Serial
#include "MS5837.h"            // Bar30 Sensor
#include <PID_v1.h>            // PID
#include <Servo.h>             // Servo
#include <Wire.h>              // I2C

/**************************************************************************/
/*      Definitions                                                       */
/**************************************************************************/

/* Arduino Pins */
#define EINT1_PIN 2 // External interrupt 1 pin
#define LEAK_PIN 4  // Leak signal pin
#define SERVO_1_PIN 5 // Servo 1 pin
#define SERVO_2_PIN 6 // Servo 2 pin
#define SRX_PIN  9  // Software serial Rx pin
#define STX_PIN  10 // Software serial Tx pin
#define LED_1_PIN 11  //  LED1 pin (for debugging)
#define LED_2_PIN 12  //  LED2 pin (for debugging)
#define LED_3_PIN 13  //  LED3 pin (for debugging)
#define ADC_1_PIN A0 // ADC 1 pin for batt voltage measurement (Pin 4 of power sense)
#define ADC_2_PIN A1 // ADC 2 pin for batt current measurement (Pin 3 of power sense)

/* States */
#define IDLE 0
#define DEPTH 1
#define ALTITUDE 2
#define SURFACE 3

/* General Constants */
#define SEA_WATER 1029  // Density of seawater in kg/m^3 (997 Fresh/1029 Salt)
#define BAUD_RATE 9600  // Serial baud rate
#define INIT_SERVO_POS 90 // Initial servo position 90 degrees
#define SERVO_LIMIT 18  // Limit servo position in degrees (90 +- 18)
#define HEIGHT_LIMIT 250  // Range of acceptable height/depth varation (mm)
```



```
/* IMU Sample Rate */
#define BNO055_SAMPLERATE_DELAY_MS (100)  // BNO055 sample delay in ms

/***************************************************************************/
/*      Global Variables                                                   */
/***************************************************************************/

/* General System Variables */
uint8_t leak = 0; // 0 = Dry , 1 = Leak
uint8_t leakState = 0; // 0 = LED off , 1 = LED on
uint8_t state = IDLE; // State variable for mode selection
volatile boolean flag = LOW; //flag for EINT ISR

/* Timing Variables */
unsigned long currentTime, lastTime, transmitTime = 0; // For time tracking
int logRate = 0;    // Data transmission rate (Hz)
double logPeriod = 0; // Data transmission period (s)
unsigned int mDelay = 15; // Delay for servo motors
unsigned int sDelay = 500;  // Short delay
unsigned int lDelay = 2000; // Long delay

/* Sensor Variables */
double depth, pressure, temperature = 0; // Bar30 data
double dx, dy, dz = 0;  // BNO055 IMU gyro data (dy is pitch, dz is roll)
double altitude = 0;  // Ping sonar data
double minAltitude = 1000; // Minimum distance from sea floor (mm)
double maxDepth = 100000; // Maximum depth (mm)
double dAltitude, dDepth = 0;  // Error in target values
int adc1, adc2 = 0; // Variables for ADC values
double voltage, current = 0;    // Battery voltage and current

/* PID Variables */
double targetDepth, targetAltitude = 0;  // Target values of depth and altitude
double heightSetpoint, heightInput, heightOutput, OutH = 0;  // Height PID
double rollSetpoint, rollInput, rollOutput, OutR = 0; // Roll PID
double output1, output2 = 0;  // Servo1 and servo2 output

/* PID Tuning Parameters */
double hKp, hKi, hKd = 0; // Height proportional, integral, derivative gains
double dKp, dKi, dKd = 0; // Depth proportional, integral, derivative gains
double rKp, rKi, rKd = 0; // Roll proportional, integral, derivative gains

/* Servo Variables */
unsigned int servoMax = INIT_SERVO_POS + SERVO_LIMIT;
unsigned int servoMin = INIT_SERVO_POS - SERVO_LIMIT;
```

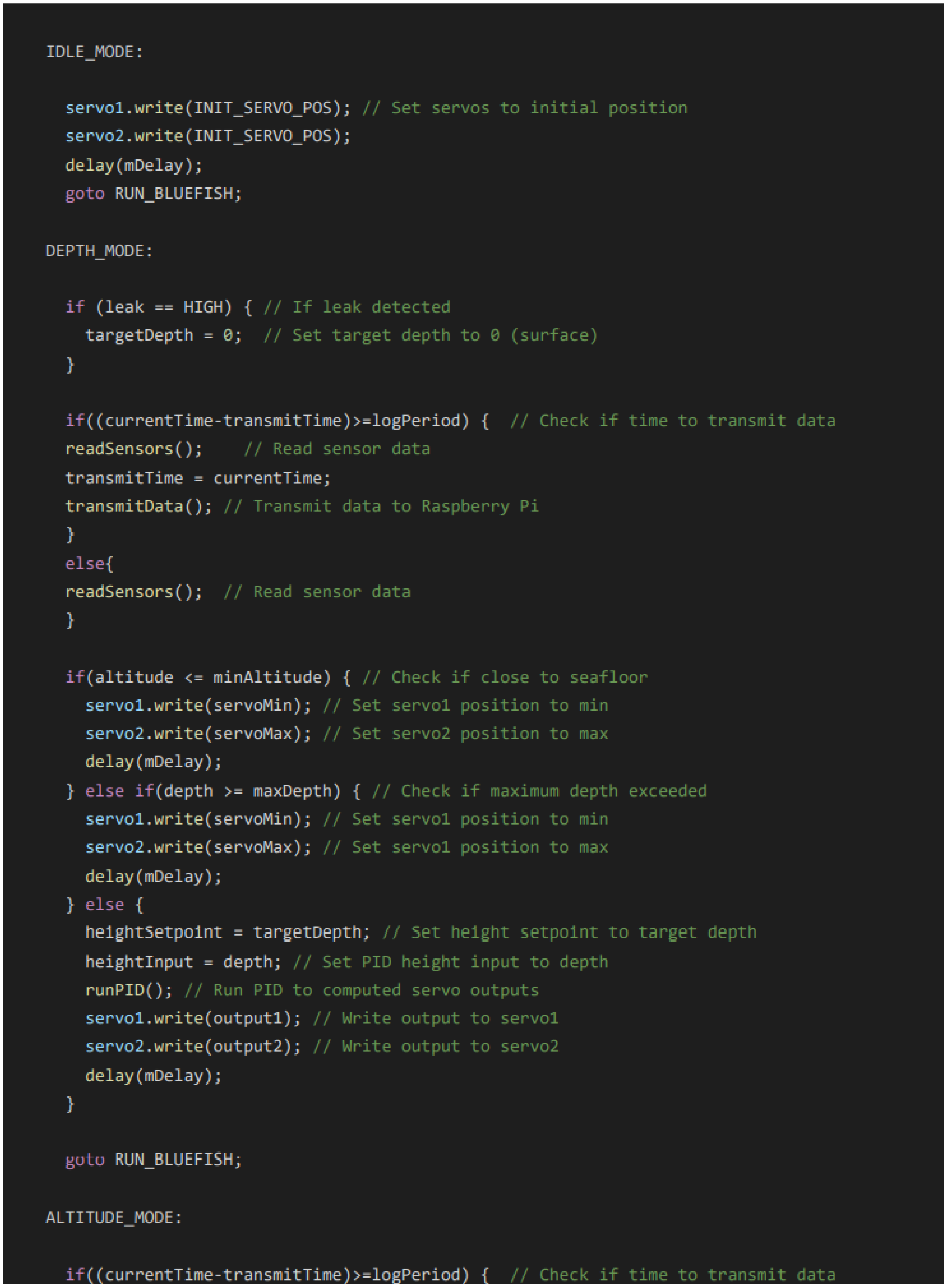


```
unsigned int servoRatio = SERVO_LIMIT/HEIGHT_LIMIT; //ratio of servo angle to depth/height ran
ge

/**************************************************************************/
/*      Object Declarations                                                 */
/**************************************************************************/

SoftwareSerial pingSerial = SoftwareSerial(SRX_PIN, STX_PIN); // Set ping serial to use SS pin
s
static Ping1D ping { pingSerial };  // Create ping object with SS

Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28); // Create BNO055 object and set I2C address
sensors_event_t event; // Create event for BNO055

MS5837 bar30; // Create Bar30 object

Servo servo1;  // Servo 1 object
Servo servo2;  // Servo 2 object

PID heightPID(&heightInput, &heightOutput, &heightSetpoint, hKp, hKi, hKd, P_ON_M, DIRECT); //
 Height PID
PID rollPID(&rollInput, &rollOutput, &rollSetpoint, rKp, rKi, rKd, P_ON_M, DIRECT);  // Roll P
ID

/**************************************************************************/
/*      Functions                                                           */
/**************************************************************************/

void initSensor(void);  // Checks if sensors are initialized
void displayCalStatus(void);  // Display IMU calibration
void readSensors(void); // Read all sensors and store values
void transmitData(void);  // Transmit data via serial
void leakWarningFlash(void);  // Display leak warning on LEDs
void runPID(void);  // Compute PID outputs
void isrSettings(void);  // ISR to update settings sent from RPi
void updateSettings (void); // Update settings from Raspberry Pi

/**************************************************************************/
/*      Program Setup                                                       */
/**************************************************************************/

void setup(void)
{
  /* Set I/O pinmodes */
  pinMode(LEAK_PIN, INPUT);  // Set leak sensor pin to INPUT
```

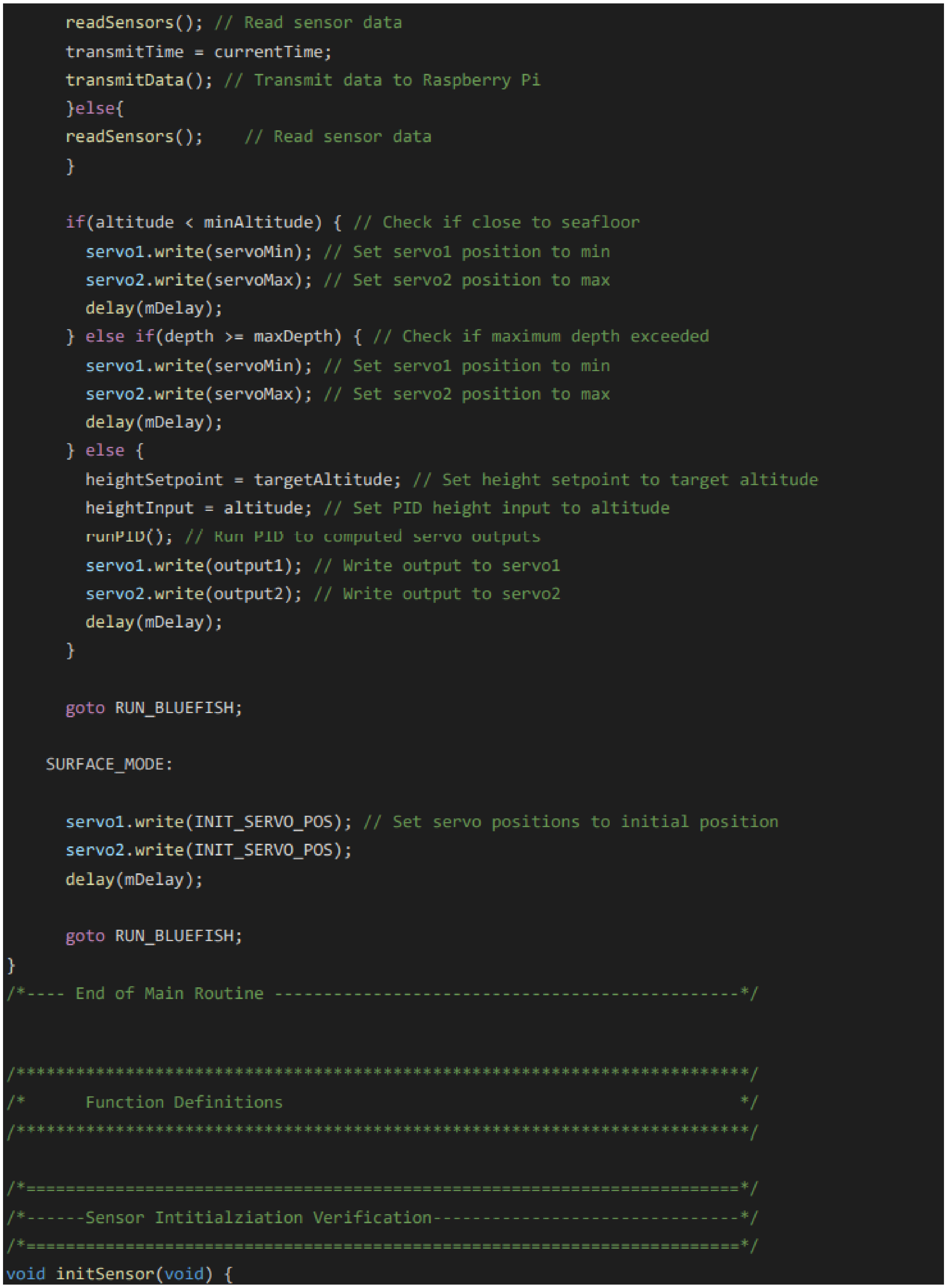


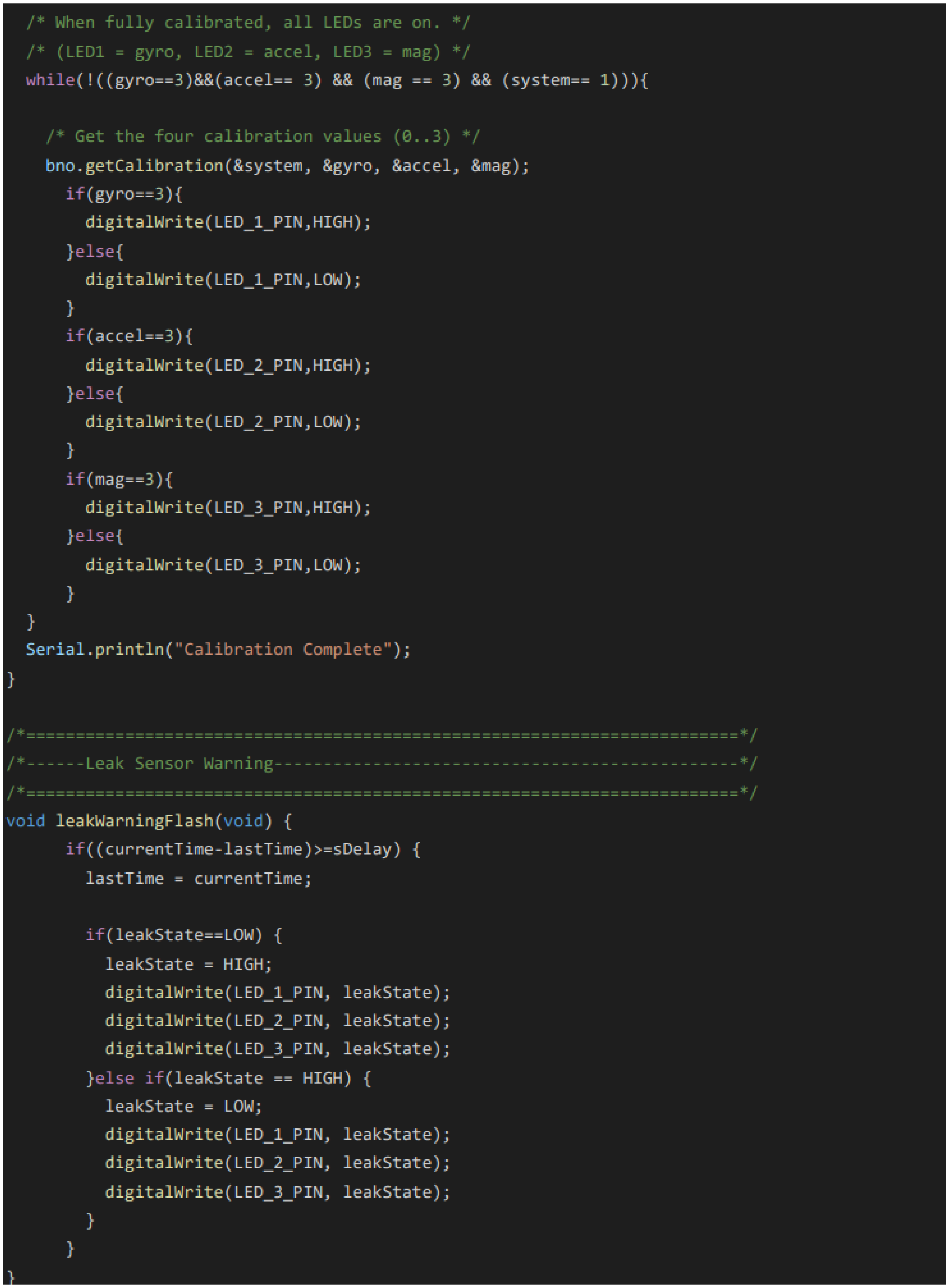
```
  pinMode(EINT1_PIN, INPUT); // Set interrupt pin to INPUT
  pinMode(LED_1_PIN, OUTPUT);  // Set LED1 pin to OUTPUT
  pinMode(LED_2_PIN, OUTPUT);  // Set LED2 pin to OUTPUT
  pinMode(LED_3_PIN, OUTPUT);  // Set LED3 pin to OUTPUT
  attachInterrupt(digitalPinToInterrupt(EINT1_PIN),isrSettings,RISING); // External interrupt

  /* Initialize serial, I2C and sensors */
  Serial.begin(BAUD_RATE);  // Initialize serial at baud rate
  pingSerial.begin(BAUD_RATE);  // Initialize software serial at baud rate
  Wire.begin(); // Initialize I2C
  initSensor(); // Checks if sensors are functioning

    /* Bar 30 sensor setup */
  bar30.setModel(MS5837::MS5837_30BA);
  bar30.setFluidDensity(SEA_WATER); // Set fluid density to sea water

  sensor_t sensor;  // Used for BNO055
  bno.getSensor(&sensor);
  bno.setExtCrystalUse(true);
  displayCalStatus(); // Wait until BNO055 calibrated (display status on LEDs)

   /* Attach servos and write initial position */
  servo1.attach(SERVO_1_PIN, 500, 2500);  // Attach servo1 to servo1 pin
  servo1.write(INIT_SERVO_POS); // Set servo1 position to initial position
  delay(sDelay);
  servo2.attach(SERVO_2_PIN, 500, 2500); // Attach servo2 to servo2 pin
  servo2.write(INIT_SERVO_POS); // Set servo2 position to initial position
  delay(mDelay);

  /* Turn on PID and set output limits*/
  heightPID.SetMode(AUTOMATIC); // Set height PID mode automatic (ON)
  rollPID.SetMode(AUTOMATIC); // Set roll PID mode automatic (ON)
  heightPID.SetOutputLimits(-HEIGHT_LIMIT,HEIGHT_LIMIT); // Set Height PID limits (+-250 mm)
  rollPID.SetOutputLimits(-SERVO_LIMIT,SERVO_LIMIT); // Set roll PID limits to servo limits

  delay(1000); // Delay for IMU calibration
}

/*****************************************************************************/
/*      Main Routine                                                         */
/*****************************************************************************/

void loop(void){

  goto RUN_BLUEFISH;
```

```
RUN_BLUEFISH:

  /* Check if Raspberry Pi updated settings */
  if(flag==HIGH){
    updateSettings();
  }

  /* Check for leak */
  leak = digitalRead(LEAK_PIN);   // Read the Leak Sensor Pin

  if (leak == HIGH) { // If leak detected
    state = DEPTH;  // Set state to DEPTH mode
    leakWarningFlash(); // Flash LEDs indicating leak
    goto MODE_SWITCH;
  }

  logPeriod = (1/logRate)*1000; // Convert log rate in Hz to period in milliseconds
  currentTime = millis(); // Set current time

  goto MODE_SWITCH;

MODE_SWITCH:

  switch(state) {

    case IDLE:
      goto IDLE_MODE;
      break;

    case DEPTH:
      goto DEPTH_MODE;
      break;

    case ALTITUDE:
      goto ALTITUDE_MODE;
      break;

    case SURFACE:
      goto SURFACE_MODE;
      break;

    default:
      goto RUN_BLUEFISH;
  }
```

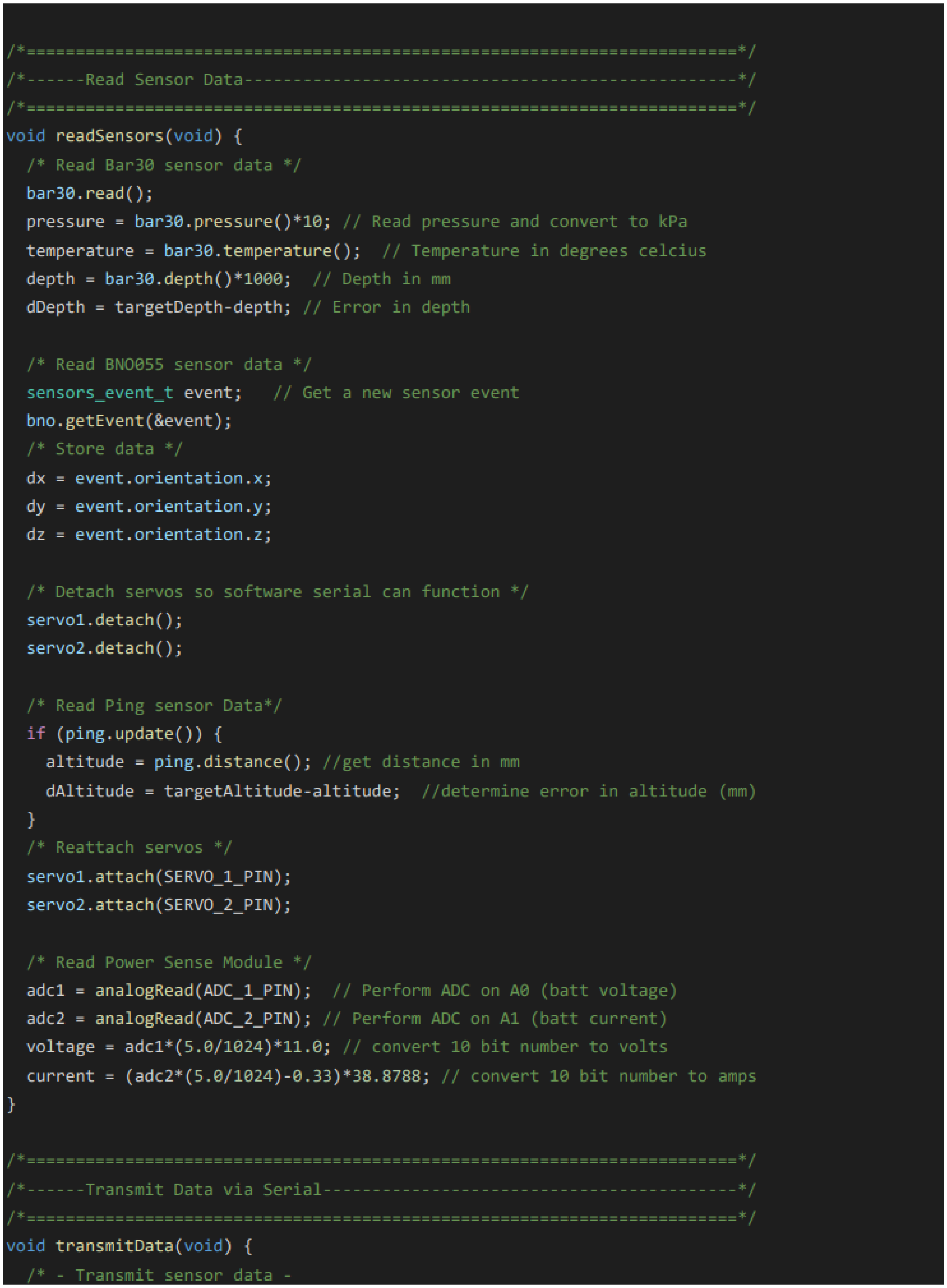


```
IDLE_MODE:

  servo1.write(INIT_SERVO_POS); // Set servos to initial position
  servo2.write(INIT_SERVO_POS);
  delay(mDelay);
  goto RUN_BLUEFISH;

DEPTH_MODE:

  if (leak == HIGH) { // If leak detected
    targetDepth = 0;  // Set target depth to 0 (surface)
  }

  if((currentTime-transmitTime)>=logPeriod) {  // Check if time to transmit data
  readSensors();    // Read sensor data
  transmitTime = currentTime;
  transmitData(); // Transmit data to Raspberry Pi
  }
  else{
  readSensors();  // Read sensor data
  }

  if(altitude <= minAltitude) { // Check if close to seafloor
    servo1.write(servoMin); // Set servo1 position to min
    servo2.write(servoMax); // Set servo2 position to max
    delay(mDelay);
  } else if(depth >= maxDepth) { // Check if maximum depth exceeded
    servo1.write(servoMin); // Set servo1 position to min
    servo2.write(servoMax); // Set servo1 position to max
    delay(mDelay);
  } else {
    heightSetpoint = targetDepth; // Set height setpoint to target depth
    heightInput = depth; // Set PID height input to depth
    runPID(); // Run PID to computed servo outputs
    servo1.write(output1); // Write output to servo1
    servo2.write(output2); // Write output to servo2
    delay(mDelay);
  }

  goto RUN_BLUEFISH;

ALTITUDE_MODE:

  if((currentTime-transmitTime)>=logPeriod) {  // Check if time to transmit data
```


```
      readSensors(); // Read sensor data
      transmitTime = currentTime;
      transmitData(); // Transmit data to Raspberry Pi
      }else{
      readSensors();    // Read sensor data
      }

      if(altitude < minAltitude) { // Check if close to seafloor
        servo1.write(servoMin); // Set servo1 position to min
        servo2.write(servoMax); // Set servo2 position to max
        delay(mDelay);
      } else if(depth >= maxDepth) { // Check if maximum depth exceeded
        servo1.write(servoMin); // Set servo1 position to min
        servo2.write(servoMax); // Set servo2 position to max
        delay(mDelay);
      } else {
        heightSetpoint = targetAltitude; // Set height setpoint to target altitude
        heightInput = altitude; // Set PID height input to altitude
        runPID(); // Run PID to computed servo outputs
        servo1.write(output1); // Write output to servo1
        servo2.write(output2); // Write output to servo2
        delay(mDelay);
      }

      goto RUN_BLUEFISH;

    SURFACE_MODE:

      servo1.write(INIT_SERVO_POS); // Set servo positions to initial position
      servo2.write(INIT_SERVO_POS);
      delay(mDelay);

      goto RUN_BLUEFISH;
}
/*---- End of Main Routine ----------------------------------------------*/


/***********************************************************************/
/*      Function Definitions                                           */
/***********************************************************************/

/*===================================================================*/
/*------Sensor Intitialziation Verification------------------------------*/
/*===================================================================*/
void initSensor(void) {
  int bnoC, bar30C, pingC = 0;  // Sensor status: 0 = connected, 1 = disconnected

  if(!bno.begin()){  //check if BNO055 initialized
    bnoC = 1;
  }else{
    bnoC = 0;
  }
  if(!bar30.init()) {  //check if Bar30 initialized
    bar30C  = 1;
  }else{
    bar30C = 0;
  }
  if(!ping.initialize()) {  //check if Ping initialized
    pingC = 1;
  }else{
    pingC = 0;
  }
   /* Blink LED while sensor is not initiailized */
   /* LED1 = BNO055, LED2 = Bar30, LED3 = Ping */
  while((bnoC==1)||(bar30C==1)||(pingC==1)) {
    if(bnoC == 1) {
      digitalWrite(LED_1_PIN,HIGH);
    }
    if(bar30C == 1) {
      digitalWrite(LED_2_PIN,HIGH);
    }
    if(pingC == 1) {
      digitalWrite(LED_3_PIN,HIGH);
    }
    delay(sDelay);
    digitalWrite(LED_1_PIN,LOW);
    digitalWrite(LED_2_PIN,LOW);
    digitalWrite(LED_3_PIN,LOW);
    delay(sDelay);
  }
}

/*==========================================================================*/
/*------Display Sensor Calibration Status -------------------------------*/
/*==========================================================================*/
void displayCalStatus(void) {
  uint8_t system, gyro, accel, mag = 0;

  /* While all values not calibrated, get calibration status.*/
  /* 3 means 'fully calibrated" and requires system > 0  */
```



```
  /* When fully calibrated, all LEDs are on. */
  /* (LED1 = gyro, LED2 = accel, LED3 = mag) */
  while(!((gyro==3)&&(accel== 3) && (mag == 3) && (system== 1))){

    /* Get the four calibration values (0..3) */
    bno.getCalibration(&system, &gyro, &accel, &mag);
      if(gyro==3){
        digitalWrite(LED_1_PIN,HIGH);
      }else{
        digitalWrite(LED_1_PIN,LOW);
      }
      if(accel==3){
        digitalWrite(LED_2_PIN,HIGH);
      }else{
        digitalWrite(LED_2_PIN,LOW);
      }
      if(mag==3){
        digitalWrite(LED_3_PIN,HIGH);
      }else{
        digitalWrite(LED_3_PIN,LOW);
      }
  }
  Serial.println("Calibration Complete");
}

/*============================================================================*/
/*------Leak Sensor Warning---------------------------------------------------*/
/*============================================================================*/
void leakWarningFlash(void) {
      if((currentTime-lastTime)>=sDelay) {
        lastTime = currentTime;

        if(leakState==LOW) {
          leakState = HIGH;
          digitalWrite(LED_1_PIN, leakState);
          digitalWrite(LED_2_PIN, leakState);
          digitalWrite(LED_3_PIN, leakState);
        }else if(leakState == HIGH) {
          leakState = LOW;
          digitalWrite(LED_1_PIN, leakState);
          digitalWrite(LED_2_PIN, leakState);
          digitalWrite(LED_3_PIN, leakState);
        }
      }
}
```



```
/*========================================================================*/
/*------Read Sensor Data--------------------------------------------------*/
/*========================================================================*/
void readSensors(void) {
  /* Read Bar30 sensor data */
  bar30.read();
  pressure = bar30.pressure()*10; // Read pressure and convert to kPa
  temperature = bar30.temperature();  // Temperature in degrees celcius
  depth = bar30.depth()*1000;  // Depth in mm
  dDepth = targetDepth-depth; // Error in depth

  /* Read BNO055 sensor data */
  sensors_event_t event;   // Get a new sensor event
  bno.getEvent(&event);
  /* Store data */
  dx = event.orientation.x;
  dy = event.orientation.y;
  dz = event.orientation.z;

  /* Detach servos so software serial can function */
  servo1.detach();
  servo2.detach();

  /* Read Ping sensor Data*/
  if (ping.update()) {
    altitude = ping.distance(); //get distance in mm
    dAltitude = targetAltitude-altitude;  //determine error in altitude (mm)
  }
  /* Reattach servos */
  servo1.attach(SERVO_1_PIN);
  servo2.attach(SERVO_2_PIN);

  /* Read Power Sense Module */
  adc1 = analogRead(ADC_1_PIN);  // Perform ADC on A0 (batt voltage)
  adc2 = analogRead(ADC_2_PIN); // Perform ADC on A1 (batt current)
  voltage = adc1*(5.0/1024)*11.0; // convert 10 bit number to volts
  current = (adc2*(5.0/1024)-0.33)*38.8788; // convert 10 bit number to amps
}

/*========================================================================*/
/*------Transmit Data via Serial------------------------------------------*/
/*========================================================================*/
void transmitData(void) {
  /* - Transmit sensor data -
  Sends data to Raspberry Pi in SI units*/
  Serial.print(altitude/1000);
  Serial.print(",");
  Serial.print(dAltitude/1000);
  Serial.print(",");
  Serial.print(depth/1000);
  Serial.print(",");
  Serial.print(dDepth/1000);
  Serial.print(",");
  Serial.print(pressure);
  Serial.print(",");
  Serial.print(temperature);
  Serial.print(",");
  Serial.print(dx);
  Serial.print(",");
  Serial.print(dy);
  Serial.print(",");
  Serial.print(dz);
  Serial.print(",");
  Serial.print(voltage);
  Serial.print(",");
  Serial.print(current);
}

/*===========================================================================*/
/*------Compute PID Output-----------------------------------------------*/
/*===========================================================================*/
void runPID(void) {

/* Check state and set corresponding PID gains  */
  if(state==DEPTH){
    heightPID.SetTunings(dKp,dKi,dKd);
  }
  if(state==ALTITUDE){
    heightPID.SetTunings(hKp,hKi,hKd);
  }
  rollPID.SetTunings(rKp,rKi,rKd);

  /* Compute height PID output */
  heightPID.Compute();
  OutH = heightOutput;

  /* Compute roll PID output */
  rollInput = dz;  // Roll input = z position
  rollPID.Compute();
  OutR = rollOutput;

  /* Compute combined roll and height outputs */
  output1 = 90 - (((servoRatio*OutH)-
OutR)/2);  // Convert height output to angular, add roll angle and average
  output2 = 90 + (((servoRatio*OutH)+OutR)/2);
}

/*========================================================================*/
/*------Interrupt ISR ----------------------------------------------------*/
/*========================================================================*/
void isrSettings() {
  flag = !flag;
}

/*========================================================================*/
/*------Update Settings from Raspberry Pi -------------------------------*/
/*========================================================================*/
void updateSettings() {
  if(Serial.available() > 0) {
    String temp = Serial.readStringUntil(',');
    logRate = temp.toDouble();
    temp = Serial.readStringUntil(',');
    state = temp.toInt();
    temp = Serial.readStringUntil(',');
    targetDepth = (temp.toDouble())*1000;
    temp = Serial.readStringUntil(',');
    targetAltitude = (temp.toDouble())*1000;
    temp = Serial.readStringUntil(',');
    rKp = temp.toDouble();
    temp = Serial.readStringUntil(',');
    rKi = temp.toDouble();
    temp = Serial.readStringUntil(',');
    rKd = temp.toDouble();
    temp = Serial.readStringUntil(',');
    hKp = temp.toDouble();
    temp = Serial.readStringUntil(',');
    hKi = temp.toDouble();
    temp = Serial.readStringUntil(',');
    hKd = temp.toDouble();
    temp = Serial.readStringUntil(',');
    dKp = temp.toDouble();
    temp = Serial.readStringUntil(',');
    dKi = temp.toDouble();
    temp = Serial.readStringUntil(',');
    dKd = temp.toDouble();
  }
}
```

Appendix X – Main BlueFish Command Code

Note that all live plotting functionality has been commented out in the following code.

```python
# Standard Modules
import os
import sys
import csv
from datetime import datetime

# Custom Modules
from csv_logger import Logger
from FishCommand import Ui_MainWindow

# Plotting Modules
# matplotlib.use('Qt5Agg')
# from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
# from matplotlib.figure import Figure
# import matplotlib.pyplot as plt
# import matplotlib.animation as animation
# import matplotlib.ticker as ticker
# import queue
# import pandas as pd
# from pandas import DataFrame as df
#from live_plotting import Plotter
from live_plotting import MplCanvas

# PyQt5 Modules for GUI
from PyQt5 import QtWidgets as qtw
from PyQt5 import QtCore as qtc

# Hardware Modules
import serial
import gpiozero


# Global Variables
INTERRUPT = gpiozero.LED(17)  # setup GPIO and ports for raspberry pi interrupt pin 11
(GPIO 17)
ARDUINO = serial.Serial('/dev/ttyACM0', 9600, timeout=.01)  # setup serial port, baud
rate, and timeout
os.environ['QT_QUICK_CONTROLS_STYLE'] = (sys.argv[1] if len(sys.argv) > 1 else
"Default")


class FishCommandWindow(qtw.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        # Setup GUI
        self.setupUi(self)
        self.connect_buttons()
        self.set_combobox_data()

        # Setup Logging Parameters and thread class
        self._is_logger_running = False
        self.logging_thread = qtc.QThread()
        self.settings = {}

        # Plot GUI integration
        # self.plot_settings = {}
        # self.Plotter.dataSignal.connect(self.on_change)
        # Plot canvas setup
        # self.canvas = MplCanvas(self, dpi=100)
        # self.gridLayout_6.addWidget(self.canvas, 1, 0, 1, 1)
        # Data for the Plot
        # empty dataframe for data
        # self.data = df()
        # self.get_plotting_data_thread = qtc.QThread()
        # self.is_plotter_running = False

        # The plot
        # self.fig, self.ax = plt.subplots()
        # self.num_rows = round(self.plot_settings['Elapsed Time [s]']) *
self.settings['Sample Rate']
        # self.y_data = pd.Series()
        # self.x_data = pd.Series()
        # ani = animation.FuncAnimation(self.fig, self.start_plotting, interval=10,
blit=True, save_count=self.num_rows)
        self.show()

    def connect_buttons(self) -> None:
        """Connect signals from each button to their corresponding methods"""

        self.actionSave_Settings.triggered.connect(self.save_settings)
        self.actionLoad_Settings.triggered.connect(self.load_settings)

self.pushButton_blueFishSettingsUpdate.clicked.connect(self.push_settings_to_bluefish)

self.pushButton_updateLivePlotSettings.clicked.connect(self.update_plot_settings)
        self.pushButton_saveLivePlot.clicked.connect(self.save_plot)
        self.pushButton_photoSaveFolder.clicked.connect(self.choose_photo_directory)

    def set_combobox_data(self) -> None:
        """Provide data values for combo boxes with units in text"""

        # sample rate in Hz
        self.comboBox_sampleRate.setItemData(0, 1)
        self.comboBox_sampleRate.setItemData(1, 5)
        self.comboBox_sampleRate.setItemData(2, 10)
        self.comboBox_sampleRate.setItemData(3, 25)
        self.comboBox_sampleRate.setItemData(4, 50)
        self.comboBox_sampleRate.setItemData(5, 100)

        # elapsed time on plot
        self.comboBox_plotTimeElapsed.setItemData(0, 5)  # 5 seconds
        self.comboBox_plotTimeElapsed.setItemData(0, 10)  # 10 seconds
        self.comboBox_plotTimeElapsed.setItemData(0, 30)  # 30 seconds
        self.comboBox_plotTimeElapsed.setItemData(0, 60)  # 1 minute
        self.comboBox_plotTimeElapsed.setItemData(0, 60*5)  # 5 minutes
        self.comboBox_plotTimeElapsed.setItemData(0, 60 * 10)  # 10 minutes
        self.comboBox_plotTimeElapsed.setItemData(0, 60 * 30)  # 30 minutes

    def save_settings(self) -> None:
        """Save csv with all metadata and settings"""

        option = qtw.QFileDialog.Options()
        file = qtw.QFileDialog.getSaveFileName(self, "Save BlueFish Settings",
"Settings.csv", "*.csv", options=option)
        if file[0]:
            with open(file[0], "w", newline='\n') as f:
                self.get_bluefish_settings()
                writer = csv.writer(f, delimiter=',')
                for label, data in self.settings.items():
                    writer.writerow([label, data])
        else:
            pass

    def load_settings(self) -> None:
        """Allow user to choose csv file and load bluefish settings into GUI"""
        option = qtw.QFileDialog.Options()
        file = qtw.QFileDialog.getOpenFileName(self, "Load BlueFish Settings",
"Settings.csv", "*.csv", options=option)
        if file[0]:
            with open(file[0], "r", newline='\n') as f:
                reader = csv.reader(f)
                self.settings = {rows[0]: rows[1] for rows in reader}
            self.set_bluefish_settings()
        else:
            pass

    def get_bluefish_settings(self) -> None:
        """Update settings dictionary with user inputs"""

        self.settings = {
            'Sample Rate': self.comboBox_sampleRate.currentIndex(),
            'Operation Mode': self.comboBox_operationMode.currentIndex(),
            'Target Depth [m]': self.doubleSpinBox_targetDepth.value(),
            'Target Height [m]': self.doubleSpinBox_targetHeight.value(),
            'Roll Kp': self.doubleSpinBox_rollP.value(),
            'Roll Ki': self.doubleSpinBox_rollI.value(),
            'Roll Kd': self.doubleSpinBox_rollD.value(),
            'Height Kp': self.doubleSpinBox_heightP.value(),
            'Height Ki': self.doubleSpinBox_heightI.value(),
            'Height Kd': self.doubleSpinBox_heightD.value(),
            'Depth Kp': self.doubleSpinBox_depthP.value(),
            'Depth Ki': self.doubleSpinBox_depthI.value(),
            'Depth Kd': self.doubleSpinBox_depthD.value(),
            'Adaptive Depth Kp': self.doubleSpinBox_adaptiveP.value(),
            'Adaptive Depth Ki': self.doubleSpinBox_adaptiveI.value(),
            'Adaptive Depth Kd': self.doubleSpinBox_adaptiveD.value(),
            'Camera Mode': self.comboBox_cameraMode.currentIndex(),
            'Photo Frequency [ms]': self.spinBox_photoFrequency.value()}

    def set_bluefish_settings(self) -> None:
        """Set BlueCommand UI values to those from the saved settings"""

        self.comboBox_sampleRate.setCurrentIndex(int(self.settings['Sample Rate']))
        self.comboBox_operationMode.setCurrentIndex(int(self.settings['Operation
Mode']))
        self.doubleSpinBox_targetDepth.setValue(float(self.settings['Target Depth
[m]']))
        self.doubleSpinBox_targetHeight.setValue(float(self.settings['Target Height
[m]']))
        self.doubleSpinBox_rollP.setValue(float(self.settings['Roll Kp']))
        self.doubleSpinBox_rollI.setValue(float(self.settings['Roll Ki']))
        self.doubleSpinBox_rollD.setValue(float(self.settings['Roll Kd']))
        self.doubleSpinBox_heightP.setValue(float(self.settings['Height Kp']))
        self.doubleSpinBox_heightI.setValue(float(self.settings['Height Ki']))
        self.doubleSpinBox_heightD.setValue(float(self.settings['Height Kd']))
        self.doubleSpinBox_depthP.setValue(float(self.settings['Depth Kp']))
        self.doubleSpinBox_depthI.setValue(float(self.settings['Depth Ki']))
        self.doubleSpinBox_depthD.setValue(float(self.settings['Depth Kd']))
        self.doubleSpinBox_adaptiveP.setValue(float(self.settings['Adaptive Depth
Kp']))
        self.doubleSpinBox_adaptiveI.setValue(float(self.settings['Adaptive Depth
Ki']))
        self.doubleSpinBox_adaptiveD.setValue(float(self.settings['Adaptive Depth
Kd']))

    def push_settings_to_bluefish(self) -> None:
        """ get user input settings, interrupt arduino program to update arduino
operational settings """
```

```
        INTERRUPT.on()
        if self._is_logger_running:
            self.stop_logging()
        # if self.is_plotter_running:
        #     self.stop_plotting()

        # get settings, and modify them for passing into the logger's meta data
        self.get_bluefish_settings()

        # Let user create file for data and start logger.plotter for non-standby modes
        if self.settings['Operation Mode'] != 0:
            filename = self.lineEdit_filenameSuffix.text()
            option = qtw.QFileDialog.Options()
            file = qtw.QFileDialog.getSaveFileName(self, "BlueFish Logging Data File",

(datetime.today().strftime('%Y_%m_%d - %H.%M') + ' - ' + filename +
                                                     '.csv'), "*.csv", options=option)
            if file[0]:
                self.start_logging(file[0])
                # self.start_plotting()
            else:
                self.comboBox_operationMode.setCurrentIndex(0)
                return
        self.get_bluefish_settings()
        for setting, value in self.settings.items():
            if setting in ['Camera Mode', 'Photo Frequency [ms]',
                           'Adaptive Depth Kp', 'Adaptive Depth Ki', 'Adaptive Depth
Kd']:
                pass
            else:
                if setting == 'Sample Rate':
                    value = self.comboBox_sampleRate.currentData()
                send_string = (str(value) + ',')
                print(send_string)
                ARDUINO.write(send_string.encode('utf-8'))
        INTERRUPT.off()

    def update_plot_settings(self) -> None:
        # if self._is_plotter_running():
        #     self.stop_plotting()
        #     self.is_plotter_running = False
        # self.start_plotting()
        # self.is_plotter_running = True
        pass

    def start_logging(self, filepath) -> None:
        """Start a logging thread and connect all signals and slots"""

        settings = self.settings
        # settings['Operation Mode'] = self.comboBox_operationMode.currentText()
        settings['Sample Rate'] = self.comboBox_sampleRate.currentData()

        self.logging_thread = Logger(0, ARDUINO, settings, filepath)
        self.logging_thread.start()
        self._is_logger_running = True

    def stop_logging(self) -> None:
        """Stop and eliminate logging thread, setting _is_logger_running to false"""

        self.logging_thread.stop()
        self._is_logger_running = False

    def choose_photo_directory(self):
        pass
    def get_plot_settings(self) -> None:
        """Update plot settings dictionary with current user input"""

        # self.plot_settings = {
        #     'Elapsed Time [s]': self.comboBox_plotTimeElapsed.currentData(),
        #     'Y1': self.comboBox_plotY1.currentText(),
        #     'Y2': self.comboBox_plotY2.currentText(),
        #     'Y3': self.comboBox_plotY3.currentText()
        # }
        pass

    def start_plotting(self) -> None:
        # """Start a logging thread and connect all signals and slots"""
        # self.get_plot_settings()
        # settings = self.settings
        # settings['Sample Rate'] = self.comboBox_sampleRate.currentData()
        # self.plotting_thread = Plotter(1, settings, self.plot_settings)
        # self.plotting_thread.start()
        pass

    def get_plot_data(self) -> None:
        # self.x_data = self.data['Elapsed Time [s]']
        # self.y_data = self.data[self.plot_settings['Y1', 'Y2', 'Y3']]
        pass

    def stop_plotting(self) -> None:
        # self.plotting_thread.stop()
        pass

    def save_plot(self) -> None:
        pass


if __name__ == '__main__':
    ARDUINO.flush()  # get rid of garbage/incomplete data
    INTERRUPT.off()  # make sure interrupt is low
    print("Calibrate the Fish")

    # wait for arduino to be calibrated
    arduino_calibration_status = '0'
    while arduino_calibration_status != 'Calibration Complete':
        arduino_calibration_status = ARDUINO.readline().decode('utf-8').rstrip()

    # open window after
    app = qtw.QApplication(sys.argv)
    win = FishCommandWindow()
    app.exec_()
```

Appendix Y – BlueFish Command GUI Code

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1122, 718)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("BR Logo.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        MainWindow.setWindowIcon(icon)
        MainWindow.setToolButtonStyle(QtCore.Qt.ToolButtonIconOnly)
        MainWindow.setTabShape(QtWidgets.QTabWidget.Rounded)
        MainWindow.setUnifiedTitleAndToolBarOnMac(False)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.centralwidget)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.groupBox_settings = QtWidgets.QTabWidget(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.groupBox_settings.sizePolicy().hasHeightForWidth())
        self.groupBox_settings.setSizePolicy(sizePolicy)
        self.groupBox_settings.setMaximumSize(QtCore.QSize(16777215, 16777215))
        palette = QtGui.QPalette()
        brush = QtGui.QBrush(QtGui.QColor(189, 219, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(189, 219, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(189, 219, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Button, brush)
        self.groupBox_settings.setPalette(palette)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.groupBox_settings.setFont(font)
        self.groupBox_settings.setAutoFillBackground(False)
        self.groupBox_settings.setStyleSheet("")
        self.groupBox_settings.setTabsClosable(False)
        self.groupBox_settings.setMovable(True)
        self.groupBox_settings.setObjectName("groupBox_settings")
        self.settingsTab = QtWidgets.QWidget()
        self.settingsTab.setObjectName("settingsTab")
        self.gridLayout = QtWidgets.QGridLayout(self.settingsTab)
        self.gridLayout.setObjectName("gridLayout")
        self.groupBox_blueFishSettings = QtWidgets.QGroupBox(self.settingsTab)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_blueFishSettings.setFont(font)
        self.groupBox_blueFishSettings.setAutoFillBackground(True)
        self.groupBox_blueFishSettings.setObjectName("groupBox_blueFishSettings")
        self.gridLayout_4 = QtWidgets.QGridLayout(self.groupBox_blueFishSettings)
        self.gridLayout_4.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstraint)
        self.gridLayout_4.setObjectName("gridLayout_4")
        self.groupBox_heightSettings =
QtWidgets.QGroupBox(self.groupBox_blueFishSettings)
        font = QtGui.QFont()
        font.setPointSize(8)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_heightSettings.setFont(font)
        self.groupBox_heightSettings.setObjectName("groupBox_heightSettings")
        self.formLayout_7 = QtWidgets.QFormLayout(self.groupBox_heightSettings)
        self.formLayout_7.setVerticalSpacing(7)
        self.formLayout_7.setObjectName("formLayout_7")
        self.label_heightP = QtWidgets.QLabel(self.groupBox_heightSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_heightP.setFont(font)
        self.label_heightP.setObjectName("label_heightP")
        self.formLayout_7.setWidget(0, QtWidgets.QFormLayout.LabelRole,
self.label_heightP)
        self.doubleSpinBox_heightP =
QtWidgets.QDoubleSpinBox(self.groupBox_heightSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_heightP.setFont(font)
        self.doubleSpinBox_heightP.setSingleStep(0.25)
        self.doubleSpinBox_heightP.setObjectName("doubleSpinBox_heightP")
        self.formLayout_7.setWidget(0, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_heightP)
        self.label_heightI = QtWidgets.QLabel(self.groupBox_heightSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_heightI.setFont(font)
        self.label_heightI.setObjectName("label_heightI")
        self.formLayout_7.setWidget(1, QtWidgets.QFormLayout.LabelRole,
self.label_heightI)
        self.doubleSpinBox_heightI =
QtWidgets.QDoubleSpinBox(self.groupBox_heightSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_heightI.setFont(font)
        self.doubleSpinBox_heightI.setSingleStep(0.25)
        self.doubleSpinBox_heightI.setObjectName("doubleSpinBox_heightI")
        self.formLayout_7.setWidget(1, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_heightI)
        self.label_heightD = QtWidgets.QLabel(self.groupBox_heightSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_heightD.setFont(font)
        self.label_heightD.setObjectName("label_heightD")
        self.formLayout_7.setWidget(2, QtWidgets.QFormLayout.LabelRole,
self.label_heightD)
        self.doubleSpinBox_heightD =
QtWidgets.QDoubleSpinBox(self.groupBox_heightSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_heightD.setFont(font)
        self.doubleSpinBox_heightD.setSingleStep(0.25)
        self.doubleSpinBox_heightD.setObjectName("doubleSpinBox_heightD")
        self.formLayout_7.setWidget(2, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_heightD)
        self.gridLayout_4.addWidget(self.groupBox_heightSettings, 0, 2, 1, 1)
        self.groupBox_cameraSettings =
QtWidgets.QGroupBox(self.groupBox_blueFishSettings)
        font = QtGui.QFont()
        font.setPointSize(8)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_cameraSettings.setFont(font)
        self.groupBox_cameraSettings.setObjectName("groupBox_cameraSettings")
        self.formLayout = QtWidgets.QFormLayout(self.groupBox_cameraSettings)
        self.formLayout.setObjectName("formLayout")
        self.label_photoFrequency = QtWidgets.QLabel(self.groupBox_cameraSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_photoFrequency.setFont(font)
        self.label_photoFrequency.setObjectName("label_photoFrequency")
        self.formLayout.setWidget(2, QtWidgets.QFormLayout.LabelRole,
self.label_photoFrequency)
        self.comboBox_cameraMode = QtWidgets.QComboBox(self.groupBox_cameraSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.comboBox_cameraMode.setFont(font)
        self.comboBox_cameraMode.setObjectName("comboBox_cameraMode")
        self.comboBox_cameraMode.addItem("")
        self.comboBox_cameraMode.addItem("")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole,
self.comboBox_cameraMode)
        self.label_cameraMode = QtWidgets.QLabel(self.groupBox_cameraSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_cameraMode.setFont(font)
        self.label_cameraMode.setObjectName("label_cameraMode")
        self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole,
self.label_cameraMode)
        self.spinBox_photoFrequency = QtWidgets.QSpinBox(self.groupBox_cameraSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.spinBox_photoFrequency.setFont(font)
        self.spinBox_photoFrequency.setMaximum(1000000)
        self.spinBox_photoFrequency.setSingleStep(100)
        self.spinBox_photoFrequency.setObjectName("spinBox_photoFrequency")
        self.formLayout.setWidget(2, QtWidgets.QFormLayout.FieldRole,
self.spinBox_photoFrequency)
        self.label = QtWidgets.QLabel(self.groupBox_cameraSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label.setFont(font)
        self.label.setText("")
        self.label.setTextInteractionFlags(QtCore.Qt.TextBrowserInteraction)
        self.label.setObjectName("label")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.label)
        self.pushButton_photoSaveFolder =
QtWidgets.QPushButton(self.groupBox_cameraSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.pushButton_photoSaveFolder.setFont(font)
        self.pushButton_photoSaveFolder.setObjectName("pushButton_photoSaveFolder")
        self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole,
self.pushButton_photoSaveFolder)
        self.gridLayout_4.addWidget(self.groupBox_cameraSettings, 1, 0, 1, 1)
        self.groupBox_operationalSettings =
QtWidgets.QGroupBox(self.groupBox_blueFishSettings)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.groupBox_operationalSettings.sizePolicy().hasHeightF
orWidth())
        self.groupBox_operationalSettings.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setPointSize(8)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_operationalSettings.setFont(font)

self.groupBox_operationalSettings.setObjectName("groupBox_operationalSettings")
        self.formLayout_3 = QtWidgets.QFormLayout(self.groupBox_operationalSettings)

self.formLayout_3.setFormAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Q
t.AlignTop)
        self.formLayout_3.setObjectName("formLayout_3")
        self.label_operationMode = QtWidgets.QLabel(self.groupBox_operationalSettings)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.label_operationMode.sizePolicy().hasHeightForWidth()
)
        self.label_operationMode.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_operationMode.setFont(font)
        self.label_operationMode.setObjectName("label_operationMode")
        self.formLayout_3.setWidget(0, QtWidgets.QFormLayout.LabelRole,
self.label_operationMode)
        self.comboBox_operationMode =
QtWidgets.QComboBox(self.groupBox_operationalSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.comboBox_operationMode.setFont(font)
        self.comboBox_operationMode.setObjectName("comboBox_operationMode")
        self.comboBox_operationMode.addItem("")
        self.comboBox_operationMode.addItem("")
        self.comboBox_operationMode.addItem("")
        self.comboBox_operationMode.addItem("")
        self.comboBox_operationMode.addItem("")
        self.formLayout_3.setWidget(0, QtWidgets.QFormLayout.FieldRole,
self.comboBox_operationMode)
        self.label_targetDepth = QtWidgets.QLabel(self.groupBox_operationalSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_targetDepth.setFont(font)
        self.label_targetDepth.setObjectName("label_targetDepth")
        self.formLayout_3.setWidget(1, QtWidgets.QFormLayout.LabelRole,
self.label_targetDepth)
        self.doubleSpinBox_targetDepth =
QtWidgets.QDoubleSpinBox(self.groupBox_operationalSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_targetDepth.setFont(font)
        self.doubleSpinBox_targetDepth.setSingleStep(0.25)
        self.doubleSpinBox_targetDepth.setObjectName("doubleSpinBox_targetDepth")
        self.formLayout_3.setWidget(1, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_targetDepth)
        self.label_targetHeight = QtWidgets.QLabel(self.groupBox_operationalSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_targetHeight.setFont(font)
        self.label_targetHeight.setObjectName("label_targetHeight")
        self.formLayout_3.setWidget(2, QtWidgets.QFormLayout.LabelRole,
self.label_targetHeight)
        self.doubleSpinBox_targetHeight =
QtWidgets.QDoubleSpinBox(self.groupBox_operationalSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_targetHeight.setFont(font)
        self.doubleSpinBox_targetHeight.setSingleStep(0.25)
        self.doubleSpinBox_targetHeight.setObjectName("doubleSpinBox_targetHeight")
        self.formLayout_3.setWidget(2, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_targetHeight)
        self.gridLayout_4.addWidget(self.groupBox_operationalSettings, 0, 0, 1, 1)
        self.groupBox_rollSettings =
QtWidgets.QGroupBox(self.groupBox_blueFishSettings)
        font = QtGui.QFont()
        font.setPointSize(8)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_rollSettings.setFont(font)
        self.groupBox_rollSettings.setObjectName("groupBox_rollSettings")
        self.formLayout_9 = QtWidgets.QFormLayout(self.groupBox_rollSettings)
        self.formLayout_9.setVerticalSpacing(7)
        self.formLayout_9.setObjectName("formLayout_9")
        self.label_rollP = QtWidgets.QLabel(self.groupBox_rollSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_rollP.setFont(font)
        self.label_rollP.setObjectName("label_rollP")
        self.formLayout_9.setWidget(0, QtWidgets.QFormLayout.LabelRole,
self.label_rollP)
```

```
        self.doubleSpinBox_rollP = 
QtWidgets.QDoubleSpinBox(self.groupBox_rollSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_rollP.setFont(font)
        self.doubleSpinBox_rollP.setSingleStep(0.25)
        self.doubleSpinBox_rollP.setObjectName("doubleSpinBox_rollP")
        self.formLayout_9.setWidget(0, QtWidgets.QFormLayout.FieldRole, 
self.doubleSpinBox_rollP)
        self.label_rollI = QtWidgets.QLabel(self.groupBox_rollSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_rollI.setFont(font)
        self.label_rollI.setObjectName("label_rollI")
        self.formLayout_9.setWidget(1, QtWidgets.QFormLayout.LabelRole, 
self.label_rollI)
        self.doubleSpinBox_rollI = 
QtWidgets.QDoubleSpinBox(self.groupBox_rollSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_rollI.setFont(font)
        self.doubleSpinBox_rollI.setSingleStep(0.25)
        self.doubleSpinBox_rollI.setObjectName("doubleSpinBox_rollI")
        self.formLayout_9.setWidget(1, QtWidgets.QFormLayout.FieldRole, 
self.doubleSpinBox_rollI)
        self.label_rollD = QtWidgets.QLabel(self.groupBox_rollSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_rollD.setFont(font)
        self.label_rollD.setObjectName("label_rollD")
        self.formLayout_9.setWidget(2, QtWidgets.QFormLayout.LabelRole, 
self.label_rollD)
        self.doubleSpinBox_rollD = 
QtWidgets.QDoubleSpinBox(self.groupBox_rollSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_rollD.setFont(font)
        self.doubleSpinBox_rollD.setSingleStep(0.25)
        self.doubleSpinBox_rollD.setObjectName("doubleSpinBox_rollD")
        self.formLayout_9.setWidget(2, QtWidgets.QFormLayout.FieldRole, 
self.doubleSpinBox_rollD)
        self.gridLayout_4.addWidget(self.groupBox_rollSettings, 0, 1, 1, 1)
        self.groupBox_depthSettings = 
QtWidgets.QGroupBox(self.groupBox_blueFishSettings)
        font = QtGui.QFont()
        font.setPointSize(8)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_depthSettings.setFont(font)
        self.groupBox_depthSettings.setObjectName("groupBox_depthSettings")
        self.formLayout_10 = QtWidgets.QFormLayout(self.groupBox_depthSettings)
        self.formLayout_10.setObjectName("formLayout_10")
        self.label_depthP = QtWidgets.QLabel(self.groupBox_depthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_depthP.setFont(font)
        self.label_depthP.setObjectName("label_depthP")
        self.formLayout_10.setWidget(0, QtWidgets.QFormLayout.LabelRole,
self.label_depthP)
        self.label_depthI = QtWidgets.QLabel(self.groupBox_depthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_depthI.setFont(font)
        self.label_depthI.setObjectName("label_depthI")
        self.formLayout_10.setWidget(1, QtWidgets.QFormLayout.LabelRole,
self.label_depthI)
        self.label_depthD = QtWidgets.QLabel(self.groupBox_depthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_depthD.setFont(font)
        self.label_depthD.setObjectName("label_depthD")
        self.formLayout_10.setWidget(2, QtWidgets.QFormLayout.LabelRole,
self.label_depthD)
        self.doubleSpinBox_depthP =
QtWidgets.QDoubleSpinBox(self.groupBox_depthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_depthP.setFont(font)
        self.doubleSpinBox_depthP.setSingleStep(0.25)
        self.doubleSpinBox_depthP.setObjectName("doubleSpinBox_depthP")
        self.formLayout_10.setWidget(0, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_depthP)
        self.doubleSpinBox_depthI =
QtWidgets.QDoubleSpinBox(self.groupBox_depthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_depthI.setFont(font)
        self.doubleSpinBox_depthI.setSingleStep(0.25)
        self.doubleSpinBox_depthI.setObjectName("doubleSpinBox_depthI")
        self.formLayout_10.setWidget(1, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_depthI)
        self.doubleSpinBox_depthD =
QtWidgets.QDoubleSpinBox(self.groupBox_depthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_depthD.setFont(font)
        self.doubleSpinBox_depthD.setSingleStep(0.25)
        self.doubleSpinBox_depthD.setObjectName("doubleSpinBox_depthD")
        self.formLayout_10.setWidget(2, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_depthD)
        self.gridLayout_4.addWidget(self.groupBox_depthSettings, 1, 1, 1, 1)
        self.groupBox_adaptiveDepthSettings =
QtWidgets.QGroupBox(self.groupBox_blueFishSettings)
        font = QtGui.QFont()
        font.setPointSize(8)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_adaptiveDepthSettings.setFont(font)
self.groupBox_adaptiveDepthSettings.setObjectName("groupBox_adaptiveDepthSettings")
        self.formLayout_5 = QtWidgets.QFormLayout(self.groupBox_adaptiveDepthSettings)
        self.formLayout_5.setObjectName("formLayout_5")
        self.label_adaptiveP = QtWidgets.QLabel(self.groupBox_adaptiveDepthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_adaptiveP.setFont(font)
        self.label_adaptiveP.setObjectName("label_adaptiveP")
        self.formLayout_5.setWidget(0, QtWidgets.QFormLayout.LabelRole,
self.label_adaptiveP)
        self.label_adaptiveI = QtWidgets.QLabel(self.groupBox_adaptiveDepthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_adaptiveI.setFont(font)
        self.label_adaptiveI.setObjectName("label_adaptiveI")
        self.formLayout_5.setWidget(1, QtWidgets.QFormLayout.LabelRole,
self.label_adaptiveI)
        self.label_adaptiveD = QtWidgets.QLabel(self.groupBox_adaptiveDepthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_adaptiveD.setFont(font)
        self.label_adaptiveD.setObjectName("label_adaptiveD")
        self.formLayout_5.setWidget(2, QtWidgets.QFormLayout.LabelRole,
self.label_adaptiveD)
        self.doubleSpinBox_adaptiveP =
QtWidgets.QDoubleSpinBox(self.groupBox_adaptiveDepthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_adaptiveP.setFont(font)
        self.doubleSpinBox_adaptiveP.setSingleStep(0.25)
        self.doubleSpinBox_adaptiveP.setObjectName("doubleSpinBox_adaptiveP")
        self.formLayout_5.setWidget(0, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_adaptiveP)
        self.doubleSpinBox_adaptiveI =
QtWidgets.QDoubleSpinBox(self.groupBox_adaptiveDepthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_adaptiveI.setFont(font)
        self.doubleSpinBox_adaptiveI.setSingleStep(0.25)
        self.doubleSpinBox_adaptiveI.setObjectName("doubleSpinBox_adaptiveI")
        self.formLayout_5.setWidget(1, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_adaptiveI)
        self.doubleSpinBox_adaptiveD =
QtWidgets.QDoubleSpinBox(self.groupBox_adaptiveDepthSettings)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.doubleSpinBox_adaptiveD.setFont(font)
        self.doubleSpinBox_adaptiveD.setSingleStep(0.25)
        self.doubleSpinBox_adaptiveD.setObjectName("doubleSpinBox_adaptiveD")
        self.formLayout_5.setWidget(2, QtWidgets.QFormLayout.FieldRole,
self.doubleSpinBox_adaptiveD)
        self.gridLayout_4.addWidget(self.groupBox_adaptiveDepthSettings, 1, 2, 1, 1)
        self.gridLayout_4.setColumnStretch(0, 2)
        self.gridLayout_4.setColumnStretch(1, 1)
        self.gridLayout_4.setColumnStretch(2, 1)
        self.gridLayout.addWidget(self.groupBox_blueFishSettings, 1, 0, 1, 1)
        self.pushButton_blueFishSettingsUpdate = 
QtWidgets.QPushButton(self.settingsTab)
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setItalic(True)
        font.setWeight(75)
        self.pushButton_blueFishSettingsUpdate.setFont(font)
        self.pushButton_blueFishSettingsUpdate.setAcceptDrops(False)
        self.pushButton_blueFishSettingsUpdate.setStyleSheet("border-color: rgb(0, 0, 
255);")
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap("BR Logo.png"), QtGui.QIcon.Normal, 
QtGui.QIcon.Off)
        self.pushButton_blueFishSettingsUpdate.setIcon(icon1)
        self.pushButton_blueFishSettingsUpdate.setAutoDefault(False)
        self.pushButton_blueFishSettingsUpdate.setDefault(False)
        self.pushButton_blueFishSettingsUpdate.setFlat(False)

self.pushButton_blueFishSettingsUpdate.setObjectName("pushButton_blueFishSettingsUpdat
e")
        self.gridLayout.addWidget(self.pushButton_blueFishSettingsUpdate, 2, 0, 1, 1)
        self.groupBox_fileSetup = QtWidgets.QGroupBox(self.settingsTab)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, 
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.groupBox_fileSetup.sizePolicy().hasHeightForWidth())
        self.groupBox_fileSetup.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(True)
        font.setWeight(75)
        font.setKerning(True)
        self.groupBox_fileSetup.setFont(font)
        self.groupBox_fileSetup.setAutoFillBackground(True)
        self.groupBox_fileSetup.setFlat(False)
        self.groupBox_fileSetup.setCheckable(False)
        self.groupBox_fileSetup.setObjectName("groupBox_fileSetup")
        self.formLayout_6 = QtWidgets.QFormLayout(self.groupBox_fileSetup)
        self.formLayout_6.setObjectName("formLayout_6")
        self.label_filenameSuffix = QtWidgets.QLabel(self.groupBox_fileSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(True)
        self.label_filenameSuffix.setFont(font)
        self.label_filenameSuffix.setObjectName("label_filenameSuffix")
        self.formLayout_6.setWidget(0, QtWidgets.QFormLayout.LabelRole, 
self.label_filenameSuffix)
        self.lineEdit_filenameSuffix = QtWidgets.QLineEdit(self.groupBox_fileSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(True)
        self.lineEdit_filenameSuffix.setFont(font)
        self.lineEdit_filenameSuffix.setClearButtonEnabled(True)
        self.lineEdit_filenameSuffix.setObjectName("lineEdit_filenameSuffix")
        self.formLayout_6.setWidget(0, QtWidgets.QFormLayout.FieldRole,
self.lineEdit_filenameSuffix)
        self.label_testPlan = QtWidgets.QLabel(self.groupBox_fileSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(True)
        self.label_testPlan.setFont(font)
        self.label_testPlan.setObjectName("label_testPlan")
        self.formLayout_6.setWidget(1, QtWidgets.QFormLayout.LabelRole,
self.label_testPlan)
        self.lineEdit_testPlan = QtWidgets.QLineEdit(self.groupBox_fileSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(True)
        self.lineEdit_testPlan.setFont(font)
        self.lineEdit_testPlan.setClearButtonEnabled(True)
        self.lineEdit_testPlan.setObjectName("lineEdit_testPlan")
        self.formLayout_6.setWidget(1, QtWidgets.QFormLayout.FieldRole,
self.lineEdit_testPlan)
        self.label_notes = QtWidgets.QLabel(self.groupBox_fileSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(True)
        self.label_notes.setFont(font)
        self.label_notes.setObjectName("label_notes")
        self.formLayout_6.setWidget(2, QtWidgets.QFormLayout.LabelRole,
self.label_notes)
        self.textEdit_notes = QtWidgets.QTextEdit(self.groupBox_fileSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(True)
        self.textEdit_notes.setFont(font)
        self.textEdit_notes.setObjectName("textEdit_notes")
        self.formLayout_6.setWidget(2, QtWidgets.QFormLayout.FieldRole,
self.textEdit_notes)
        self.comboBox_sampleRate = QtWidgets.QComboBox(self.groupBox_fileSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(True)
        self.comboBox_sampleRate.setFont(font)
        self.comboBox_sampleRate.setObjectName("comboBox_sampleRate")
        self.comboBox_sampleRate.addItem("")
        self.comboBox_sampleRate.addItem("")
        self.comboBox_sampleRate.addItem("")
        self.comboBox_sampleRate.addItem("")
        self.comboBox_sampleRate.addItem("")
        self.comboBox_sampleRate.addItem("")
        self.formLayout_6.setWidget(4, QtWidgets.QFormLayout.FieldRole,
self.comboBox_sampleRate)
        self.label_sampleRate = QtWidgets.QLabel(self.groupBox_fileSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(True)
        self.label_sampleRate.setFont(font)
        self.label_sampleRate.setObjectName("label_sampleRate")
        self.formLayout_6.setWidget(4, QtWidgets.QFormLayout.LabelRole,
self.label_sampleRate)
        self.gridLayout.addWidget(self.groupBox_fileSetup, 0, 0, 1, 1)
        self.groupBox_settings.addTab(self.settingsTab, "")
        self.plottingTab = QtWidgets.QWidget()
        self.plottingTab.setObjectName("plottingTab")
        self.gridLayout_6 = QtWidgets.QGridLayout(self.plottingTab)
        self.gridLayout_6.setObjectName("gridLayout_6")
        self.groupBox_plotSetup = QtWidgets.QGroupBox(self.plottingTab)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Maximum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.groupBox_plotSetup.sizePolicy().hasHeightForWidth())
        self.groupBox_plotSetup.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_plotSetup.setFont(font)
        self.groupBox_plotSetup.setObjectName("groupBox_plotSetup")
        self.gridLayout_2 = QtWidgets.QGridLayout(self.groupBox_plotSetup)
        self.gridLayout_2.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstraint)
        self.gridLayout_2.setContentsMargins(11, 0, 11, 5)
        self.gridLayout_2.setObjectName("gridLayout_2")
        self.label_plotTimeElapsed = QtWidgets.QLabel(self.groupBox_plotSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_plotTimeElapsed.setFont(font)
        self.label_plotTimeElapsed.setObjectName("label_plotTimeElapsed")
        self.gridLayout_2.addWidget(self.label_plotTimeElapsed, 0, 0, 1, 1)
        self.label_plotY1 = QtWidgets.QLabel(self.groupBox_plotSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_plotY1.setFont(font)
        self.label_plotY1.setObjectName("label_plotY1")
        self.gridLayout_2.addWidget(self.label_plotY1, 0, 2, 1, 1,
QtCore.Qt.AlignRight)
        self.label_plotY2 = QtWidgets.QLabel(self.groupBox_plotSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_plotY2.setFont(font)
        self.label_plotY2.setObjectName("label_plotY2")
        self.gridLayout_2.addWidget(self.label_plotY2, 0, 4, 1, 1,
QtCore.Qt.AlignRight)
        self.label_plotY3 = QtWidgets.QLabel(self.groupBox_plotSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.label_plotY3.setFont(font)
        self.label_plotY3.setObjectName("label_plotY3")
        self.gridLayout_2.addWidget(self.label_plotY3, 0, 6, 1, 1,
QtCore.Qt.AlignRight)
        self.pushButton_updateLivePlotSettings =
QtWidgets.QPushButton(self.groupBox_plotSetup)
```

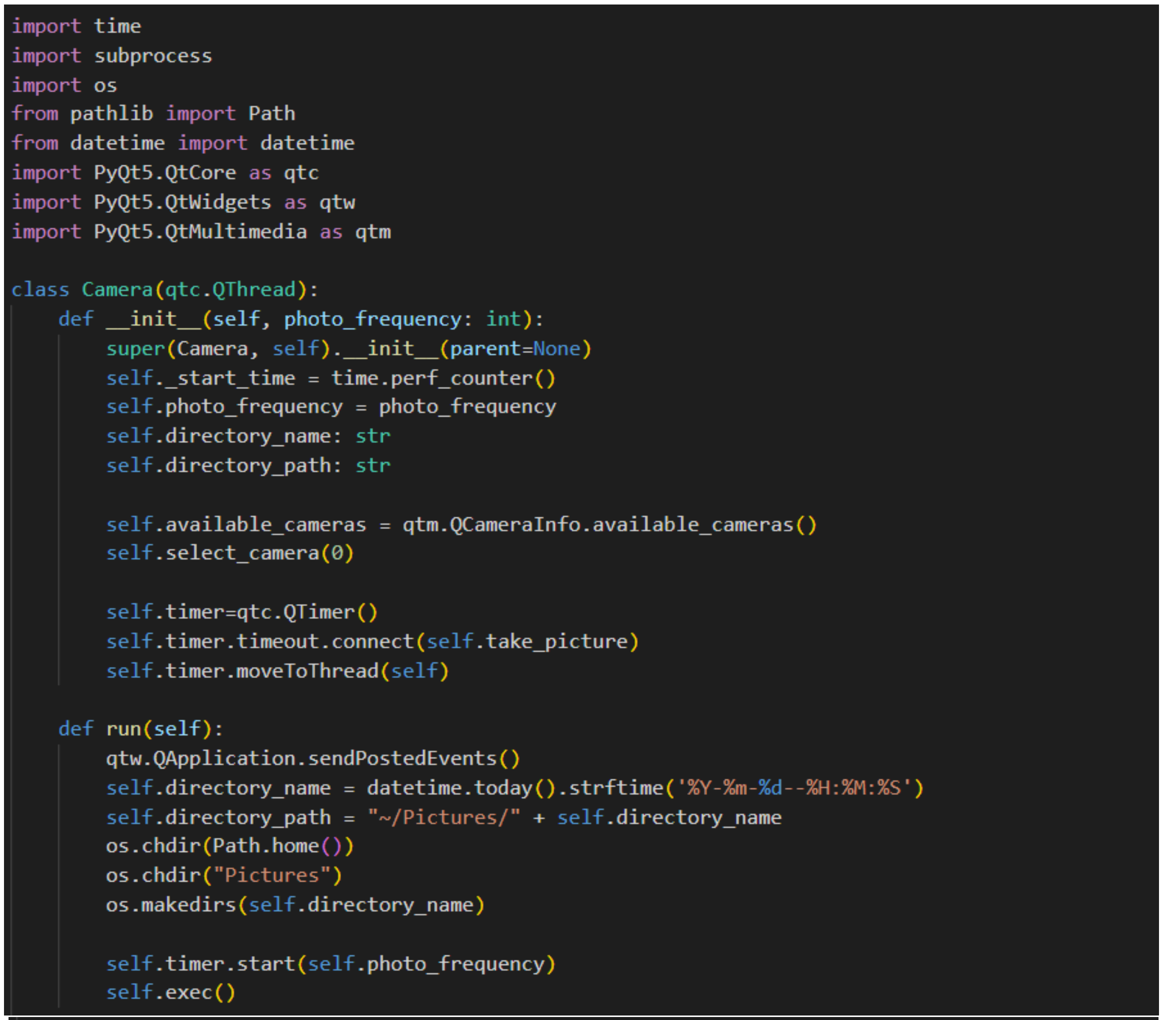
```
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.pushButton_updateLivePlotSettings.setFont(font)

self.pushButton_updateLivePlotSettings.setObjectName("pushButton_updateLivePlotSetting
s")
        self.gridLayout_2.addWidget(self.pushButton_updateLivePlotSettings, 0, 8, 1,
1)
        self.pushButton_saveLivePlot = QtWidgets.QPushButton(self.groupBox_plotSetup)
        self.pushButton_saveLivePlot.setEnabled(False)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.pushButton_saveLivePlot.setFont(font)
        self.pushButton_saveLivePlot.setObjectName("pushButton_saveLivePlot")
        self.gridLayout_2.addWidget(self.pushButton_saveLivePlot, 0, 9, 1, 1)
        self.comboBox_plotY2 = QtWidgets.QComboBox(self.groupBox_plotSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.comboBox_plotY2.setFont(font)
        self.comboBox_plotY2.setObjectName("comboBox_plotY2")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.setItemText(0, "")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.comboBox_plotY2.addItem("")
        self.gridLayout_2.addWidget(self.comboBox_plotY2, 0, 5, 1, 1)
        self.comboBox_plotY3 = QtWidgets.QComboBox(self.groupBox_plotSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.comboBox_plotY3.setFont(font)
        self.comboBox_plotY3.setObjectName("comboBox_plotY3")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.setItemText(0, "")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.comboBox_plotY3.addItem("")
        self.gridLayout_2.addWidget(self.comboBox_plotY3, 0, 7, 1, 1)
        self.comboBox_plotTimeElapsed = QtWidgets.QComboBox(self.groupBox_plotSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.comboBox_plotTimeElapsed.setFont(font)
        self.comboBox_plotTimeElapsed.setObjectName("comboBox_plotTimeElapsed")
        self.comboBox_plotTimeElapsed.addItem("")
        self.comboBox_plotTimeElapsed.addItem("")
        self.comboBox_plotTimeElapsed.addItem("")
        self.comboBox_plotTimeElapsed.addItem("")
        self.comboBox_plotTimeElapsed.addItem("")
        self.comboBox_plotTimeElapsed.addItem("")
        self.comboBox_plotTimeElapsed.addItem("")
        self.gridLayout_2.addWidget(self.comboBox_plotTimeElapsed, 0, 1, 1, 1)
        self.comboBox_plotY1 = QtWidgets.QComboBox(self.groupBox_plotSetup)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        self.comboBox_plotY1.setFont(font)
        self.comboBox_plotY1.setObjectName("comboBox_plotY1")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.setItemText(0, "")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.comboBox_plotY1.addItem("")
        self.gridLayout_2.addWidget(self.comboBox_plotY1, 0, 3, 1, 1)
        self.gridLayout_2.setColumnStretch(0, 1)
        self.gridLayout_2.setColumnStretch(1, 2)
        self.gridLayout_2.setColumnStretch(2, 1)
        self.gridLayout_2.setColumnStretch(3, 2)
        self.gridLayout_2.setColumnStretch(4, 1)
        self.gridLayout_2.setColumnStretch(5, 2)
        self.gridLayout_2.setColumnStretch(6, 1)
        self.gridLayout_2.setColumnStretch(7, 2)
        self.gridLayout_2.setColumnStretch(8, 2)
        self.gridLayout_2.setColumnStretch(9, 2)
        self.gridLayout_6.addWidget(self.groupBox_plotSetup, 0, 0, 1, 1)
        self.groupBox_plot = QtWidgets.QGroupBox(self.plottingTab)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox_plot.setFont(font)
        self.groupBox_plot.setObjectName("groupBox_plot")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.groupBox_plot)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.widget_livePlot = QtWidgets.QWidget(self.groupBox_plot)
        self.widget_livePlot.setStyleSheet("background-color: rgb(0, 53, 93);")
        self.widget_livePlot.setObjectName("widget_livePlot")
        self.verticalLayout.addWidget(self.widget_livePlot)
        self.gridLayout_6.addWidget(self.groupBox_plot, 1, 0, 1, 1)
        self.groupBox_settings.addTab(self.plottingTab, "")
        self.horizontalLayout.addWidget(self.groupBox_settings)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menuBar = QtWidgets.QMenuBar(MainWindow)
        self.menuBar.setGeometry(QtCore.QRect(0, 0, 1122, 26))
        self.menuBar.setObjectName("menuBar")
        self.menuFile = QtWidgets.QMenu(self.menuBar)
        self.menuFile.setAcceptDrops(True)
        self.menuFile.setObjectName("menuFile")
        self.menuView = QtWidgets.QMenu(self.menuBar)
        self.menuView.setObjectName("menuView")
        MainWindow.setMenuBar(self.menuBar)
        self.actionLoad_Settings = QtWidgets.QAction(MainWindow)
        self.actionLoad_Settings.setObjectName("actionLoad_Settings")
        self.actionSave_Settings = QtWidgets.QAction(MainWindow)
        self.actionSave_Settings.setAutoRepeat(True)
        self.actionSave_Settings.setObjectName("actionSave_Settings")
        self.actionMaximize = QtWidgets.QAction(MainWindow)
        self.actionMaximize.setObjectName("actionMaximize")
        self.actionNormal = QtWidgets.QAction(MainWindow)
        self.actionNormal.setObjectName("actionNormal")
        self.actionUndo = QtWidgets.QAction(MainWindow)
        self.actionUndo.setObjectName("actionUndo")
        self.actionRedo = QtWidgets.QAction(MainWindow)
        self.actionRedo.setObjectName("actionRedo")
        self.menuFile.addAction(self.actionSave_Settings)
        self.menuFile.addAction(self.actionLoad_Settings)
        self.menuFile.addSeparator()
        self.menuView.addAction(self.actionMaximize)
        self.menuView.addAction(self.actionNormal)
        self.menuBar.addAction(self.menuFile.menuAction())
        self.menuBar.addAction(self.menuView.menuAction())
        self.label_heightP.setBuddy(self.doubleSpinBox_heightP)
        self.label_heightI.setBuddy(self.doubleSpinBox_heightI)
        self.label_heightD.setBuddy(self.doubleSpinBox_heightD)
        self.label_cameraMode.setBuddy(self.comboBox_cameraMode)
        self.label.setBuddy(self.pushButton_photoSaveFolder)
        self.label_operationMode.setBuddy(self.comboBox_operationMode)
        self.label_targetDepth.setBuddy(self.doubleSpinBox_targetDepth)
        self.label_targetHeight.setBuddy(self.doubleSpinBox_targetHeight)
        self.label_rollP.setBuddy(self.doubleSpinBox_heightP)
        self.label_rollI.setBuddy(self.doubleSpinBox_heightI)
        self.label_rollD.setBuddy(self.doubleSpinBox_heightD)
        self.label_depthP.setBuddy(self.doubleSpinBox_adaptiveP)
        self.label_depthI.setBuddy(self.doubleSpinBox_adaptiveI)
        self.label_depthD.setBuddy(self.doubleSpinBox_adaptiveD)
        self.label_adaptiveP.setBuddy(self.doubleSpinBox_adaptiveP)
        self.label_adaptiveI.setBuddy(self.doubleSpinBox_adaptiveI)
        self.label_adaptiveD.setBuddy(self.doubleSpinBox_adaptiveD)
        self.label_filenameSuffix.setBuddy(self.lineEdit_filenameSuffix)
        self.label_testPlan.setBuddy(self.lineEdit_testPlan)
        self.label_notes.setBuddy(self.textEdit_notes)
        self.label_sampleRate.setBuddy(self.comboBox_sampleRate)
        self.label_plotTimeElapsed.setBuddy(self.comboBox_plotTimeElapsed)
        self.label_plotY1.setBuddy(self.comboBox_plotY1)
        self.label_plotY2.setBuddy(self.comboBox_plotY2)
        self.label_plotY3.setBuddy(self.comboBox_plotY3)

        self.retranslateUi(MainWindow)
        self.groupBox_settings.setCurrentIndex(0)
        self.actionMaximize.triggered.connect(MainWindow.showMaximized)
        self.actionNormal.triggered.connect(MainWindow.showNormal)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "FishCommand"))
        self.settingsTab.setStatusTip(_translate("MainWindow", "BlueFish Operating 
Parameters"))
        self.groupBox_blueFishSettings.setTitle(_translate("MainWindow", "BlueFish 
Settings"))
        self.groupBox_heightSettings.setTitle(_translate("MainWindow", "Height PID 
Settings"))
        self.label_heightP.setText(_translate("MainWindow", "Proportional Gain"))
        self.label_heightI.setText(_translate("MainWindow", "Integral Gain"))
        self.label_heightD.setText(_translate("MainWindow", "Derivative Gain"))
        self.groupBox_cameraSettings.setTitle(_translate("MainWindow", "Camera
Settings"))
        self.label_photoFrequency.setText(_translate("MainWindow", "Photo Frequency
[ms]"))
        self.comboBox_cameraMode.setItemText(0, _translate("MainWindow", "Off"))
        self.comboBox_cameraMode.setItemText(1, _translate("MainWindow",
"Photomosaic"))
        self.label_cameraMode.setText(_translate("MainWindow", "Camera Mode"))
        self.pushButton_photoSaveFolder.setText(_translate("MainWindow", "Save
Folder"))
        self.groupBox_operationalSettings.setTitle(_translate("MainWindow",
"Operational Settings"))
        self.label_operationMode.setText(_translate("MainWindow", "Operation Mode"))
        self.comboBox_operationMode.setItemText(0, _translate("MainWindow",
"Standby"))
        self.comboBox_operationMode.setItemText(1, _translate("MainWindow", "Constant
Depth"))
        self.comboBox_operationMode.setItemText(2, _translate("MainWindow", "Constant
Height"))
        self.comboBox_operationMode.setItemText(3, _translate("MainWindow",
"Surface"))
        self.comboBox_operationMode.setItemText(4, _translate("MainWindow",
"Shutdown"))
        self.label_targetDepth.setText(_translate("MainWindow", "Target Depth [m]"))
        self.label_targetHeight.setText(_translate("MainWindow", "Target Height [m]"))
        self.groupBox_rollSettings.setTitle(_translate("MainWindow", "Roll PID
Settings"))
        self.label_rollP.setText(_translate("MainWindow", "Proportional Gain"))
        self.label_rollI.setText(_translate("MainWindow", "Integral Gain"))
        self.label_rollD.setText(_translate("MainWindow", "Derivative Gain"))
        self.groupBox_depthSettings.setTitle(_translate("MainWindow", "Depth PID
Settings"))
        self.label_depthP.setText(_translate("MainWindow", "Proportional Gain"))
        self.label_depthI.setText(_translate("MainWindow", "Integral Gain"))
        self.label_depthD.setText(_translate("MainWindow", "Derivative Gain"))
        self.groupBox_adaptiveDepthSettings.setTitle(_translate("MainWindow",
"Adaptive Depth PID Settings"))
        self.label_adaptiveP.setText(_translate("MainWindow", "Proportional Gain"))
        self.label_adaptiveI.setText(_translate("MainWindow", "Integral Gain"))
        self.label_adaptiveD.setText(_translate("MainWindow", "Derivative Gain"))
        self.pushButton_blueFishSettingsUpdate.setText(_translate("MainWindow", "Push
to BlueFish"))
        self.groupBox_fileSetup.setTitle(_translate("MainWindow", "File Setup"))
        self.label_filenameSuffix.setText(_translate("MainWindow", "Filename Suffix"))
        self.lineEdit_filenameSuffix.setPlaceholderText(_translate("MainWindow", "Text
to be added at the end of the filename..."))
        self.label_testPlan.setText(_translate("MainWindow", "Test Plan"))
        self.lineEdit_testPlan.setPlaceholderText(_translate("MainWindow", "Link for
test report document..."))
        self.label_notes.setText(_translate("MainWindow", "Notes"))
        self.textEdit_notes.setPlaceholderText(_translate("MainWindow", "Anything of
note to save before beginning test..."))
        self.comboBox_sampleRate.setItemText(0, _translate("MainWindow", "1 Hz"))
        self.comboBox_sampleRate.setItemText(1, _translate("MainWindow", "5 Hz"))
        self.comboBox_sampleRate.setItemText(2, _translate("MainWindow", "10 Hz"))
        self.comboBox_sampleRate.setItemText(3, _translate("MainWindow", "25 Hz"))
        self.comboBox_sampleRate.setItemText(4, _translate("MainWindow", "50 Hz"))
        self.comboBox_sampleRate.setItemText(5, _translate("MainWindow", "100 Hz"))
        self.label_sampleRate.setText(_translate("MainWindow", "Sample Rate"))

self.groupBox_settings.setTabText(self.groupBox_settings.indexOf(self.settingsTab),
_translate("MainWindow", "Settings"))
        self.plottingTab.setStatusTip(_translate("MainWindow", "BlueFish Operation
Settings"))
        self.groupBox_plotSetup.setTitle(_translate("MainWindow", "Plot Setup"))
        self.label_plotTimeElapsed.setText(_translate("MainWindow", "Time Elapsed"))
        self.label_plotY1.setText(_translate("MainWindow", "Y1"))
        self.label_plotY2.setText(_translate("MainWindow", "Y2"))
        self.label_plotY3.setText(_translate("MainWindow", "Y3"))
        self.pushButton_updateLivePlotSettings.setText(_translate("MainWindow",
"Update Plot"))
        self.pushButton_saveLivePlot.setText(_translate("MainWindow", "Save Plot"))
        self.comboBox_plotY2.setItemText(1, _translate("MainWindow", "Depth Error
[m]"))
        self.comboBox_plotY2.setItemText(2, _translate("MainWindow", "Height Error
[m]"))
        self.comboBox_plotY2.setItemText(3, _translate("MainWindow", "Depth [m]"))
        self.comboBox_plotY2.setItemText(4, _translate("MainWindow", "Height [m]"))
        self.comboBox_plotY2.setItemText(5, _translate("MainWindow", "Pressure
[kPa]"))
        self.comboBox_plotY2.setItemText(6, _translate("MainWindow", "Temperature
[C]"))
        self.comboBox_plotY2.setItemText(7, _translate("MainWindow", "Yaw [deg]"))
        self.comboBox_plotY2.setItemText(8, _translate("MainWindow", "Pitch [deg]"))
        self.comboBox_plotY2.setItemText(9, _translate("MainWindow", "Roll [deg]"))
        self.comboBox_plotY2.setItemText(10, _translate("MainWindow", "Battery Voltage
[V]"))
        self.comboBox_plotY2.setItemText(11, _translate("MainWindow", "Current Draw
[A]"))
        self.comboBox_plotY3.setItemText(1, _translate("MainWindow", "Depth Error
[m]"))
        self.comboBox_plotY3.setItemText(2, _translate("MainWindow", "Height Error
[m]"))
        self.comboBox_plotY3.setItemText(3, _translate("MainWindow", "Depth [m]"))
        self.comboBox_plotY3.setItemText(4, _translate("MainWindow", "Height [m]"))
        self.comboBox_plotY3.setItemText(5, _translate("MainWindow", "Pressure
[kPa]"))
        self.comboBox_plotY3.setItemText(6, _translate("MainWindow", "Temperature
[C]"))
        self.comboBox_plotY3.setItemText(7, _translate("MainWindow", "Yaw [deg]"))
        self.comboBox_plotY3.setItemText(8, _translate("MainWindow", "Pitch [deg]"))
        self.comboBox_plotY3.setItemText(9, _translate("MainWindow", "Roll [deg]"))
        self.comboBox_plotY3.setItemText(10, _translate("MainWindow", "Battery Voltage
[V]"))
        self.comboBox_plotY3.setItemText(11, _translate("MainWindow", "Current Draw
[A]"))
        self.comboBox_plotTimeElapsed.setItemText(0, _translate("MainWindow", "5
seconds"))
        self.comboBox_plotTimeElapsed.setItemText(1, _translate("MainWindow", "10
seconds"))
        self.comboBox_plotTimeElapsed.setItemText(2, _translate("MainWindow", "30
secconds"))
        self.comboBox_plotTimeElapsed.setItemText(3, _translate("MainWindow", "1
minute"))
        self.comboBox_plotTimeElapsed.setItemText(4, _translate("MainWindow", "5
minutes"))
        self.comboBox_plotTimeElapsed.setItemText(5, _translate("MainWindow", "10
minutes"))
        self.comboBox_plotTimeElapsed.setItemText(6, _translate("MainWindow", "30
minutes"))
        self.comboBox_plotY1.setItemText(1, _translate("MainWindow", "Depth Error
[m]"))
        self.comboBox_plotY1.setItemText(2, _translate("MainWindow", "Height Error
[m]"))
        self.comboBox_plotY1.setItemText(3, _translate("MainWindow", "Depth [m]"))
        self.comboBox_plotY1.setItemText(4, _translate("MainWindow", "Height [m]"))
        self.comboBox_plotY1.setItemText(5, _translate("MainWindow", "Pressure
[kPa]"))
        self.comboBox_plotY1.setItemText(6, _translate("MainWindow", "Temperature
[C]"))
        self.comboBox_plotY1.setItemText(7, _translate("MainWindow", "Yaw [deg]"))
        self.comboBox_plotY1.setItemText(8, _translate("MainWindow", "Pitch [deg]"))
        self.comboBox_plotY1.setItemText(9, _translate("MainWindow", "Roll [deg]"))
        self.comboBox_plotY1.setItemText(10, _translate("MainWindow", "Battery Voltage
[V]"))
        self.comboBox_plotY1.setItemText(11, _translate("MainWindow", "Current Draw
[A]"))
        self.groupBox_plot.setTitle(_translate("MainWindow", "Plot"))

self.groupBox_settings.setTabText(self.groupBox_settings.indexOf(self.plottingTab),
_translate("MainWindow", "Data Plotting"))
        self.menuFile.setTitle(_translate("MainWindow", "File"))
        self.menuView.setTitle(_translate("MainWindow", "View"))
        self.actionLoad_Settings.setText(_translate("MainWindow", "Load Settings"))
        self.actionLoad_Settings.setShortcut(_translate("MainWindow", "Ctrl+L"))
        self.actionSave_Settings.setText(_translate("MainWindow", "Save Settings"))
        self.actionSave_Settings.setShortcut(_translate("MainWindow", "Ctrl+S"))
        self.actionMaximize.setText(_translate("MainWindow", "Maximize"))
        self.actionMaximize.setShortcut(_translate("MainWindow", "Ctrl+M"))
        self.actionNormal.setText(_translate("MainWindow", "Normal"))
        self.actionNormal.setShortcut(_translate("MainWindow", "Ctrl+N"))
        self.actionUndo.setText(_translate("MainWindow", "Undo"))
        self.actionUndo.setShortcut(_translate("MainWindow", "Ctrl+Z"))
        self.actionRedo.setText(_translate("MainWindow", "Redo"))
        self.actionRedo.setShortcut(_translate("MainWindow", "Ctrl+Y"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Appendix Z – BlueFish Command CSV Logging Code

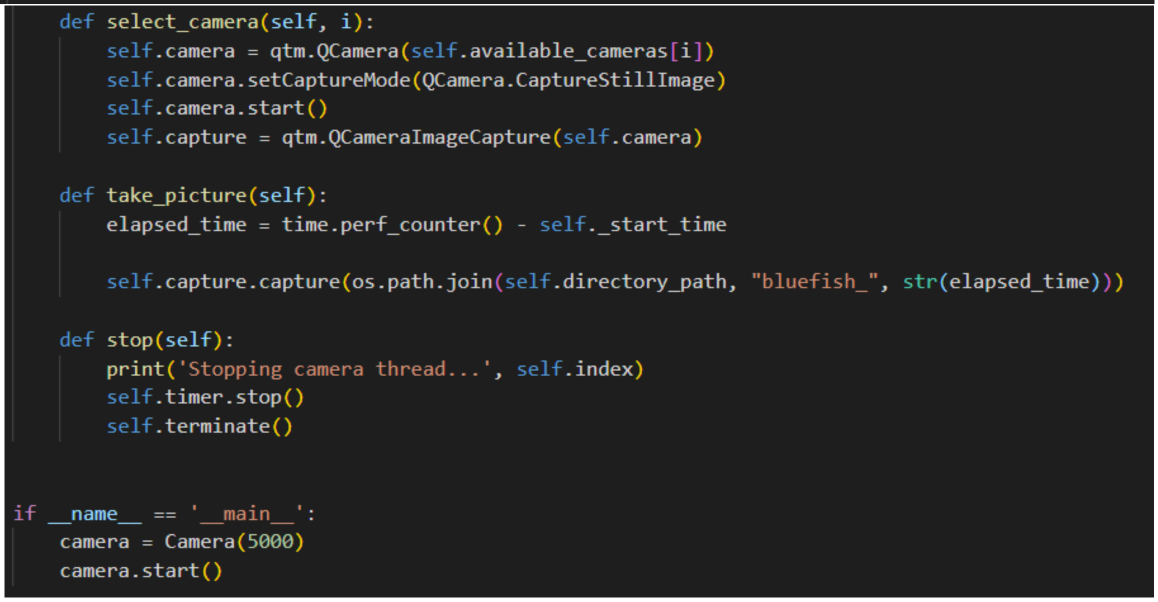
Please note that all live plotting functionality has been commented out in the code below.

```
import time
from datetime import datetime
import PyQt5.QtCore as qtc
import PyQt5.QtWidgets as qtw
from pandas import DataFrame as df


class Logger(qtc.QThread):
    def __init__(self, index: int, arduino, settings: dict, filepath: str):
        super(Logger, self).__init__(parent=None)
        self.ARDUINO = arduino
        self.filePath = filepath
        self.settings = settings
        self._start_time = time.perf_counter()
        self.sample_rate = settings['Sample Rate']
        self.data = df(columns=['Elapsed Time [s]', 'Height [m]', 'Height Error [m]' ,
'Depth [m]' , 'Depth Error [m]',
                                'Pressure [kPa]', 'Temperature [C]', 'Yaw [deg]',
'Pitch [deg]', 'Roll [deg]',
                                'Battery Voltage [V]',' Battery Current [A]'])
        self.mutex = qtc.QMutex()

        if settings['Operation Mode'] != 0:
            self.file = open(self.filePath, "a")
            print(self.filePath + " created")
            self.insert_meta_and_headers()
            self.file.close()

    def run(self):
        qtw.QApplication.sendPostedEvents()
        index = 0
        while True:
            line = self.ARDUINO.readline().decode('utf-8').rstrip()
            elapsed_time = time.perf_counter() - self._start_time
            if line:
                # parsed_data = line.split(',')
                # self.data.iloc[index] = [elapsed_time, parsed_data[0],
parsed_data[1], parsed_data[2], parsed_data[3],
                #                           parsed_data[4], parsed_data[5],
parsed_data[6], parsed_data[7], parsed_data[7],
                #                           parsed_data[8], parsed_data[9],
parsed_data[10], parsed_data[11], parsed_data[12]
                #                           ]
                # MyQtSignal.emit(self.data[index])
                self.file = open(self.filePath, "a")
                self.file.write(str(elapsed_time) + ',' + line + '\n')
                self.file.close()

    def stop(self):
        print('Stopping logging thread')
        self.terminate()

    def insert_meta_and_headers(self):
        # Insert metadata
        self.file.write('Start Time, ' + datetime.today().strftime('%Y-%m-%d -
%H:%M:%S') + '\n')
        for key, value in self.settings.items():
            self.file.write(key + ',' + str(value) + '\n')

        # create headers
        self.file.write(' \n ######DATA######## \n')
        self.file.write('\nElapsed Time [s],Height [m],Height Error [m],Depth
[m],Depth Error [m],Pressure [kPa],'
                        'Temperature [C],Yaw [deg],Pitch [deg], Roll [deg], Battery
Voltage [V],Battery Current [A] \n')
        self.file.close()
```

Appendix AA – Camera Code



```python
import time
import subprocess
import os
from pathlib import Path
from datetime import datetime
import PyQt5.QtCore as qtc
import PyQt5.QtWidgets as qtw
import PyQt5.QtMultimedia as qtm

class Camera(qtc.QThread):
    def __init__(self, photo_frequency: int):
        super(Camera, self).__init__(parent=None)
        self._start_time = time.perf_counter()
        self.photo_frequency = photo_frequency
        self.directory_name: str
        self.directory_path: str

        self.available_cameras = qtm.QCameraInfo.available_cameras()
        self.select_camera(0)

        self.timer=qtc.QTimer()
        self.timer.timeout.connect(self.take_picture)
        self.timer.moveToThread(self)

    def run(self):
        qtw.QApplication.sendPostedEvents()
        self.directory_name = datetime.today().strftime('%Y-%m-%d--%H:%M:%S')
        self.directory_path = "~/Pictures/" + self.directory_name
        os.chdir(Path.home())
        os.chdir("Pictures")
        os.makedirs(self.directory_name)

        self.timer.start(self.photo_frequency)
        self.exec()
```

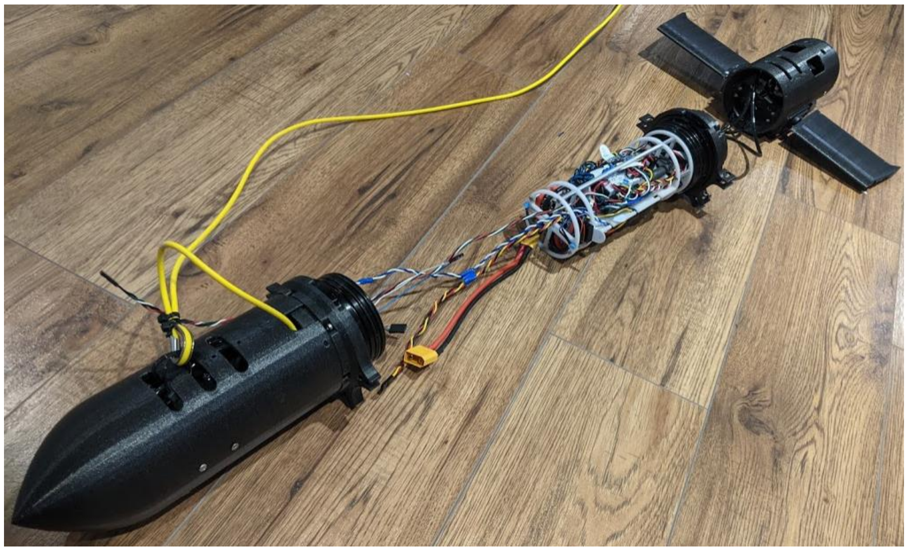


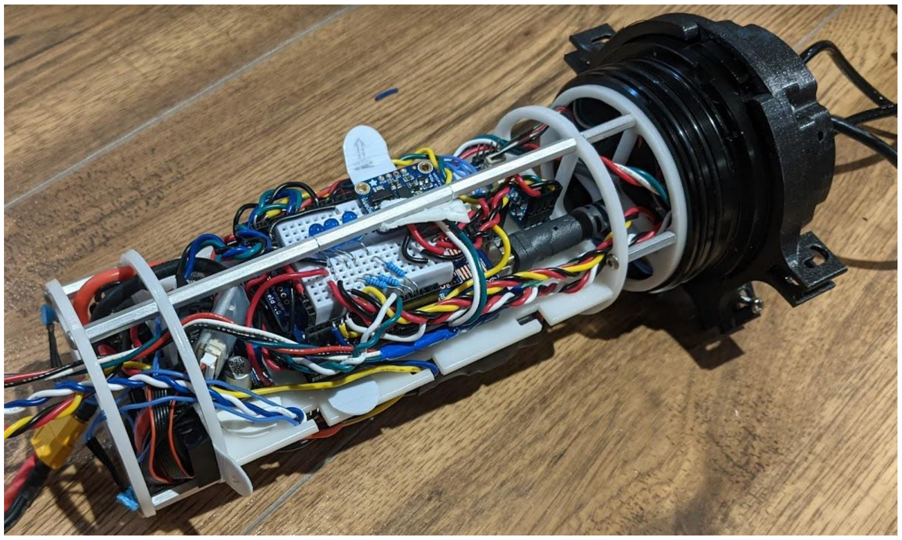
```python
    def select_camera(self, i):
        self.camera = qtm.QCamera(self.available_cameras[i])
        self.camera.setCaptureMode(QCamera.CaptureStillImage)
        self.camera.start()
        self.capture = qtm.QCameraImageCapture(self.camera)

    def take_picture(self):
        elapsed_time = time.perf_counter() - self._start_time

        self.capture.capture(os.path.join(self.directory_path, "bluefish_", str(elapsed_time)))

    def stop(self):
        print('Stopping camera thread...', self.index)
        self.timer.stop()
        self.terminate()


if __name__ == '__main__':
    camera = Camera(5000)
    camera.start()
```

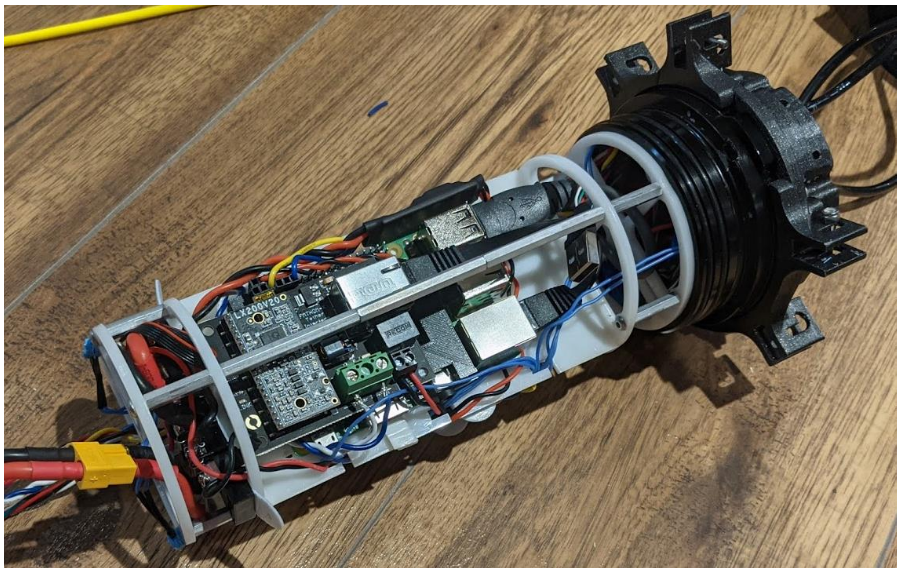
Appendix AB – Additional Test Data

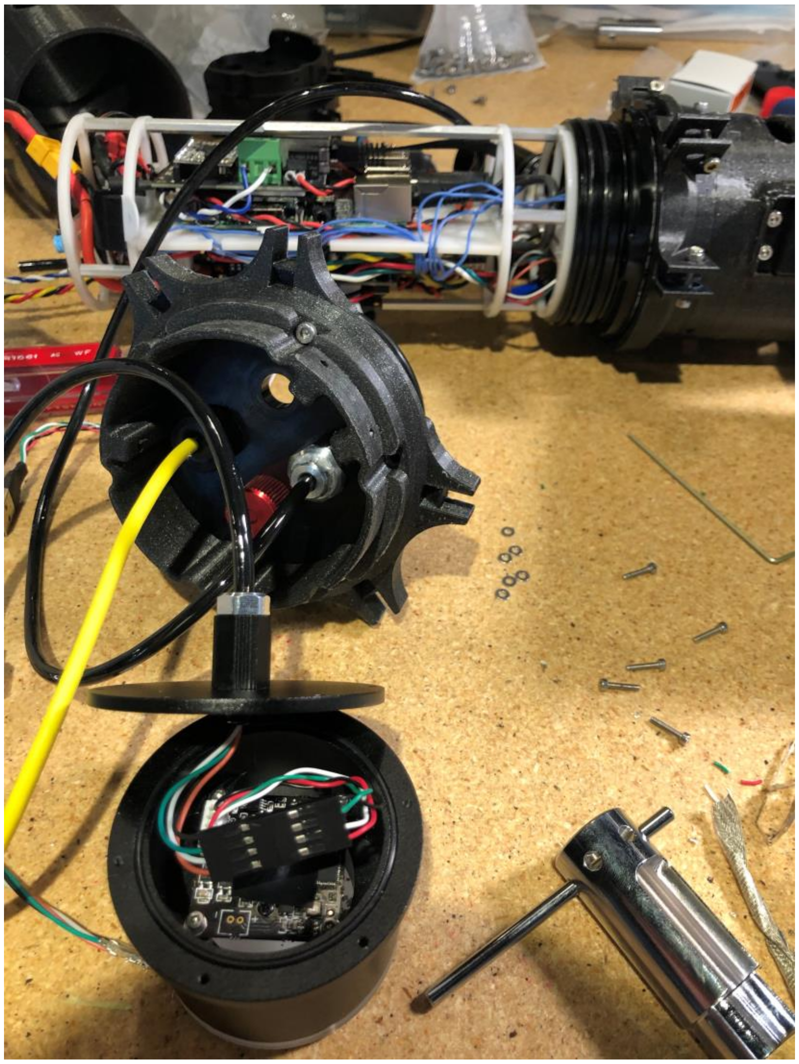
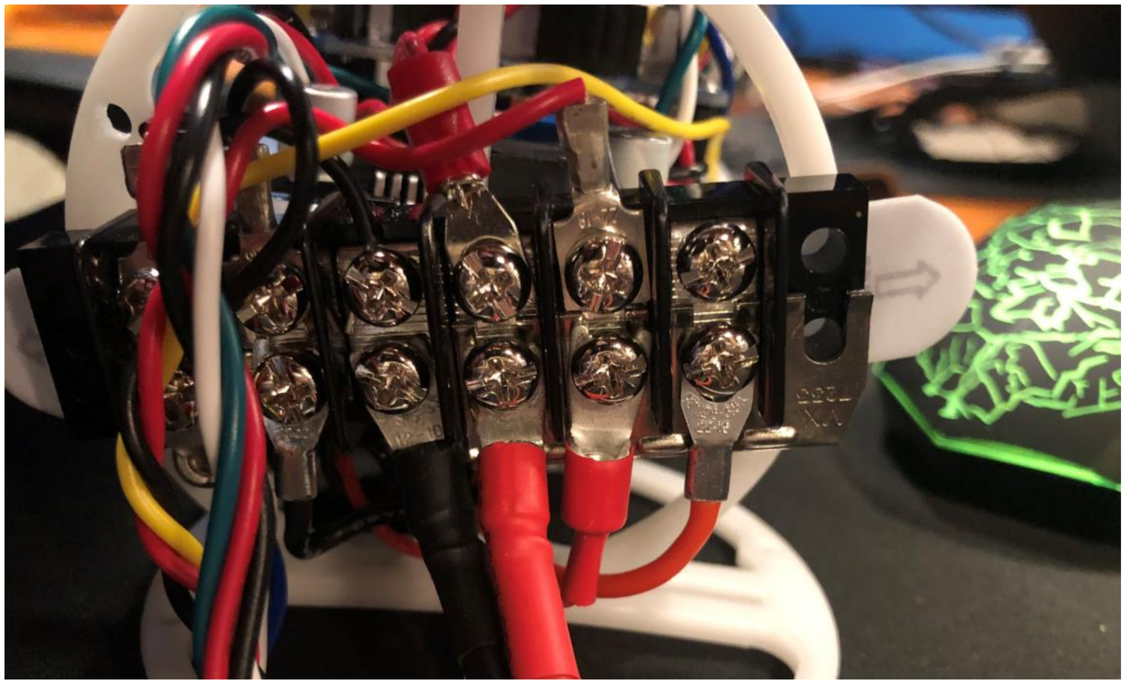
Please find a link to test data below:
https://1drv.ms/u/s!Anbv5JHlV1c6jkZC1Pn1K8rLmpKP?e=2AhGgM

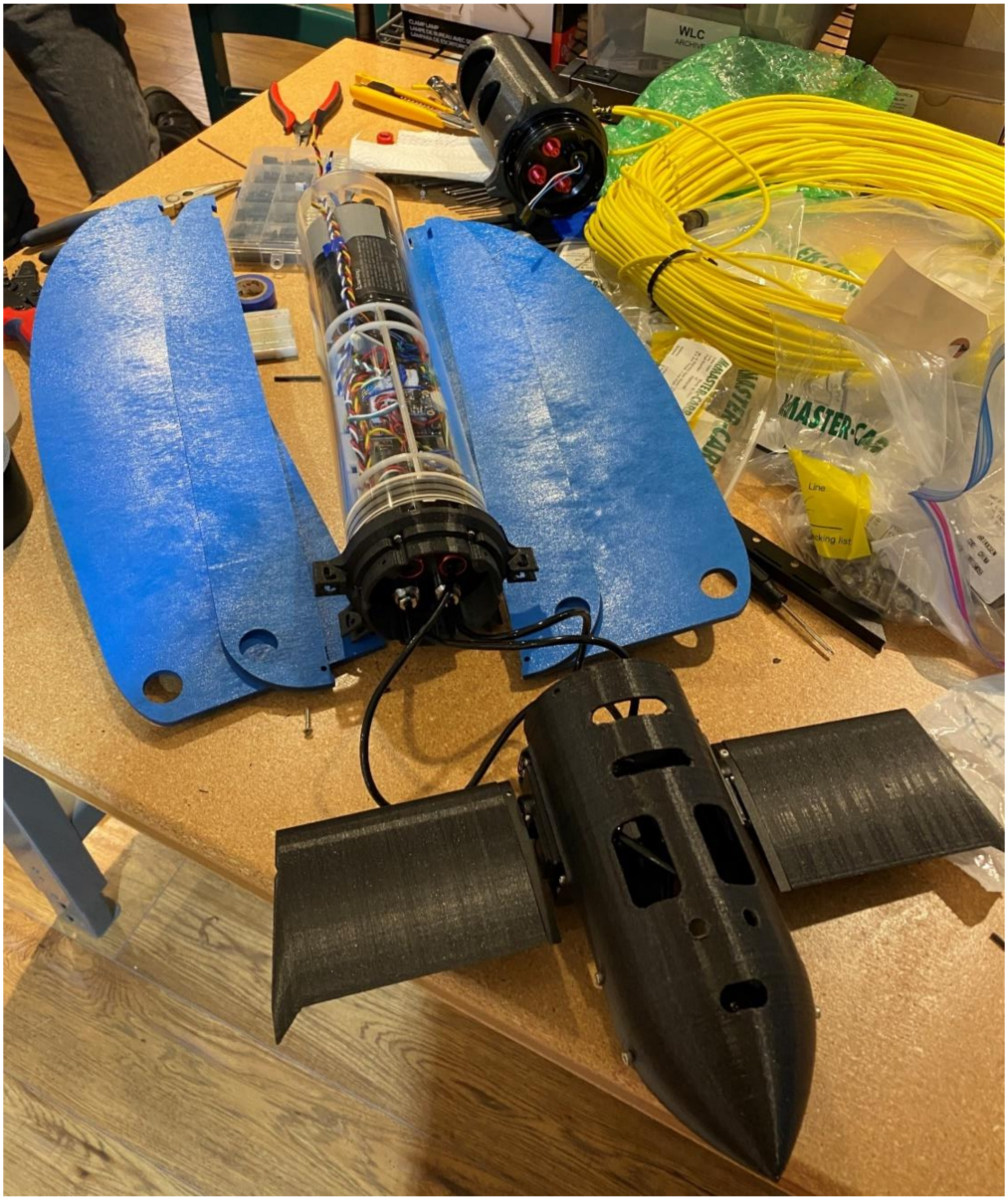
Appendix AC –Additional & Miscellaneous Images

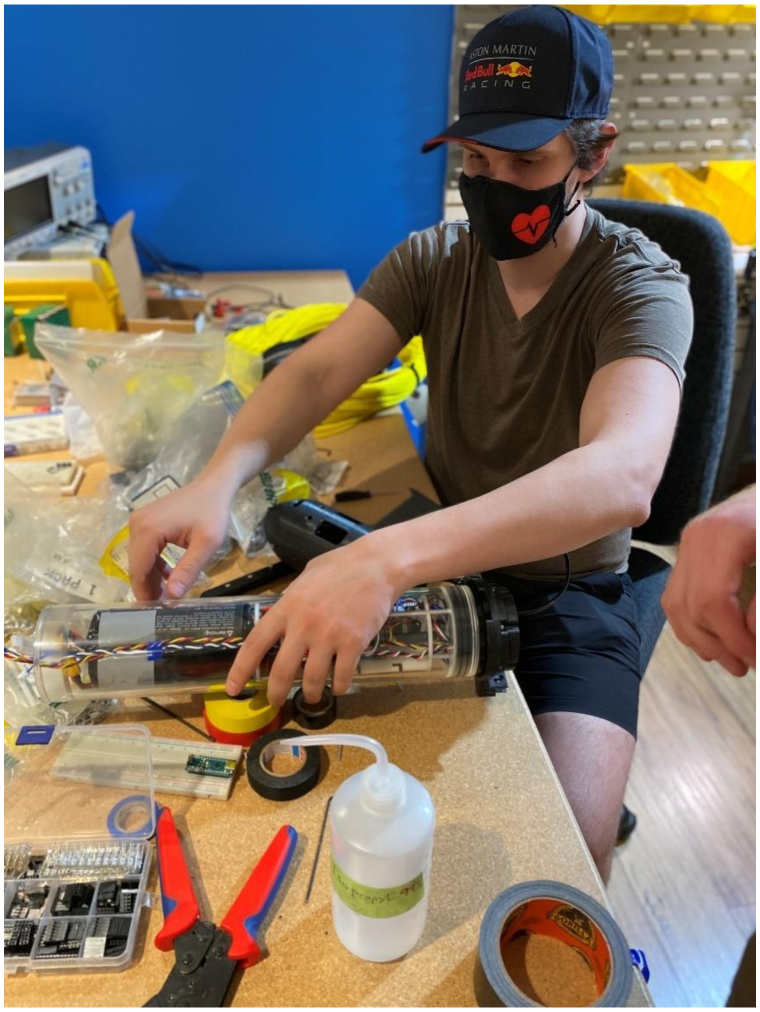


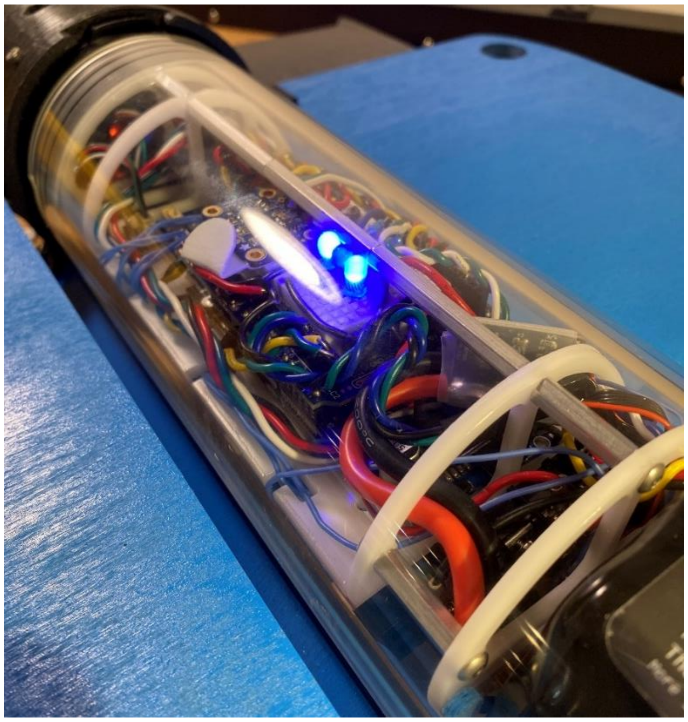







ASTON MARTIN
Red Bull
RACING



akro-mils.com

ULINE
1-800-295-5510
CANADA

UVIC
UVIC
ENGINEERING

UVIC
UVIC
ENGINEERING

BILL L LEDCOR