Duke Robotics Club's Oogway and Crush: AUV Systems for RoboSub 2025

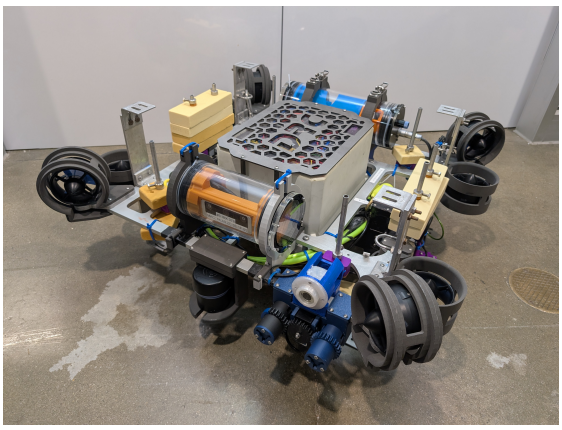
Maxwell Lin, Hung Le, Maanav Allampallam, Saagar Arya, Patrick Zheng, Nathanael Ren, Mathew Chu, Ren Staveteig, Vedarsh Shah, Carson Brantley, Joyce Hu, Lily Zheng, Ivan Chen, Niko Weaver, Raine Cheng, Jill Wang, Siddharth Kini, Ahaan Shah, Avrick Altmann, Srinath Iyer, Michelle Chen, Isabella Chen

***Abstract*—The Duke Robotics Club is proud to present our AUVs for the 2025 RoboSub Competition: *Oogway* and *Crush*. Now in its third year, Oogway returns with a new torpedo subsystem and upgraded high-fidelity mounts. Our first minibot, Crush, features a dual-capsule design that incorporates key insights from Oogway and other previous systems. Beyond hardware advancements, we also migrated our 50,000+ line codebase to ROS 2 and refactored core modules for a fully robot-agnostic architecture. Rigorous subsystem and integration tests ensure that both AUVs operate as cohesive and reliable systems at RoboSub 2025.**

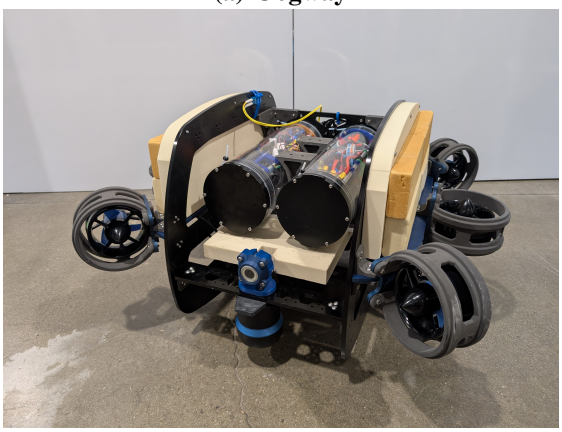
I. Competition Strategy

Our strategic vision centers on *core tasks*, which our robots are programmed to perform every run, and one *alternate task* which may be completed at the end of a successful run if more points are necessary. Our *core tasks* are implemented with multiple fail-safes to ensure reliability. Additionally, our new minibot enables us to execute distinct tasks in parallel — increasing overall throughput — or concentrate efforts on a single, challenging task to improve redundancy.

RoboSub's design goals focus on four fundamental principles: movement, vision, manipulation, and acoustic tracking. To accomplish our strategic vision we chose to focus on 3 of 4 design goals: movement, vision, and manipulation. This subset aligns with our team's strengths in controls, computer vision (CV), and mechanics. Our focus on these goals enables us to perform the gate, slalom, torpedo, and return home tasks (our *core tasks*) reliably, while still maintaining the flexibility to attempt the octagon task (our *alternate task*) if needed.



(a) Oogway



(b) Crush

Fig. 1: Duke Robotics Club's AUVs for RoboSub 2025

A. Collecting Data – Gate

The gate task must be completed by both Crush and Oogway to initiate a run. To maximize reliability for this required task, we employ a multimodal

approach: our CV subsystem can detect the gate from arbitrary angles and estimate the position of symbols underneath. If CV fails to locate the gate, the sonar subsystem provides an approximate bearing. As a final fallback, our control system can perform dead reckoning through the gate using odometry and state estimates. In all cases, we ensure traversal through the shark side of the gate.

In addition to the gate, Crush will perform both the coin flip and style tasks to earn bonus points. At RoboSub 2024, these tasks were reliably executed by Oogway. Delegating them to Crush now reduces sensor drift on Oogway and preserves its state-estimation integrity for later objectives.

B. Navigate the Channel – Slalom

After Crush completes the gate task, we use Crush's downward-facing camera in combination with our CV subsystem to find and track the path marker. After navigating in the direction of the slalom task, Crush uses its front-facing camera to identify and navigate through the three sets of pipes. Oogway will not attempt the Slalom task.

C. Tagging – Torpedoes

For the first time, the torpedo task is a core component of our competition strategy. After Oogway completes the gate task, we use its front-facing camera and CV subsystem to detect the torpedo board. We then compute the board's normal vector using sonar to align the vehicle head-on. Finally, HSV filtering centers Oogway on the target opening before launching the torpedo. We will first tag the shark, then the sawfish.

D. Ocean Cleanup – Octagon

Due to limited reliability in our acoustic tracking systems, we designate the octagon as our *alternate task*. If additional points are needed at the end of a run, Oogway will attempt to locate and surface inside the octagon using a combination of acoustics and sonar. Once Oogway completes its final *core task*, it can activate its acoustics tracking algorithm and performs wide sonar sweeps to estimate the octagon's location. Upon arrival, it uses its cameras and CV subsystem to identify the trash table and the two hanging images. Once centered above the table, Oogway will surface inside the octagon. Because this strategy is less reliable, we reserve it for the end of a run to avoid surfacing outside the octagon and prematurely ending Oogway's run. As our actuators are still in development, we will not attempt the object manipulation portion of this task.

E. Drop a BRUVS – Bins

To focus on greater reliability for other tasks, we will not attempt the bins task this year. At last year's competition, we were able to complete the bins task with our new marker-dropper subsystem; however, we found that navigation to the bins was challenging and unreliable.

F. Return Home

The addition of the return home task as a new final objective requires us to be strategic about delegating tasks between our two robots. If Oogway attempts the octagon task, there is a significant risk it will surface outside the octagon and be unable to attempt returning through the gate. Therefore, to ensure reliable completion of this *core task*, we delegate it to Crush, which will attempt it after completing the slalom task. Crush will use the same multimodal approach as in the initial gate task. If Oogway either completes or aborts the octagon task without surfacing, Oogway may also attempt the return home task to increase our chance of securing these points.

G. Inter-Vehicle Communication (IVC)

A key addition to our strategy this year is capturing IVC points. With two fully operational AUVs equipped with IVC modems, the robots can exchange messages throughout the run. Each time a robot begins or completes a task, it transmits a message to the other. After the run, both sets of timestamped message logs will be submitted to the judges to verify successful IVC.

H. Trade Offs Between Complexity and Reliability

Consistent with our strategic philosophy from last year, we emphasize reducing system complexity while maximizing reliability for both robots. This year, we again narrow our focus to a smaller set of tasks and, through extensive testing, fail-safes, and multiple methods of task completion, ensure that we can reliably complete our set of *core tasks* in any

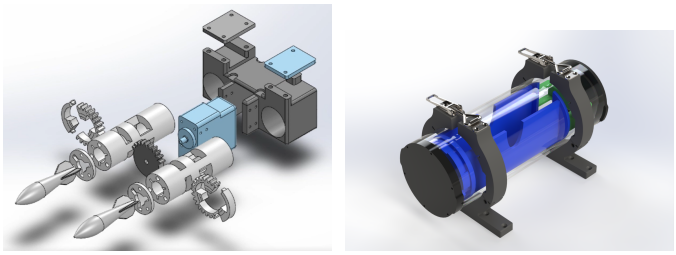
given competition run. By delegating tasks between the two robots, we also allocate more testing time per task, further increasing reliability.

II. Design Strategy

A. Mechanical Design

This year, we focused on refining Oogway by integrating a new torpedo subsystem and transitioning to higher-fidelity mounts. We also designed Crush, our new minibot, which features an aluminum frame, two isolated capsules, and a modular design.

1) Oogway Torpedo System: This is the first year that we have had a reliable torpedo system mounted on Oogway. The launcher system consists of two identical torpedo modules that slot into a servo housing. Each torpedo module uses a spring-loaded firing system that is actuated by a slip gear mounted on a waterproof servo. The modularity of the launcher design and the use of FDM printing for all parts allow for quick repairs and prototyping. The projectiles are also FDM printed, and we tested many iterations before settling on the final design (see Fig. 2a).



(a) Torpedo System **(b) Battery Clamp**

Fig. 2: New Oogway Mechanical Design Features

2) Oogway Battery Capsule Mount: A critical issue with the design of our previous battery capsule was its oblique mounting. This caused unequal drag, which led to yaw drift and had to be compensated for in software. To remedy this, we designed a releasable clamp system (see Fig. 2b) that allows the capsule to be lifted straight up, eliminating the need for angled insertion. We utilized metal latches and printed a clamp with hinges on each arm. This design not only significantly reduced our battery replacement time, but also offered the added benefit of modulating clamping pressure for a stronger hold at minimal cost to durability.

3) Crush Frame Design: We based Crush's frame design on our decision to isolate power and signal components in separate capsules (see Section II-A4). This separation improved noise isolation, simplified wiring, and directly influenced the dual-capsule layout and overall form factor.

Additionally, we performed fluid simulations (see Fig. 3) to refine the frame's geometry and minimize drag. This helped ensure smooth flow over the body and prevent unwanted pitch rotations. We also added bottom rails to the frame to support future expansion, such as sensors or task-specific mechanisms.

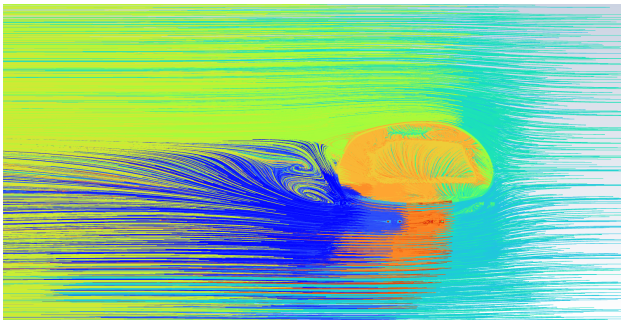


Fig. 3: Fluid Simulation of Crush

We also adopted a consistent mechanical architecture—standardized brackets, mirrored walls, and repeatable mount patterns—making parts interchangeable and significantly improving serviceability. The mirrored layout also simplified buoyancy tuning, as symmetric mass distribution made balancing more predictable.

4) Crush Capsule Design: Crush's electrical system is divided into two capsules: a power and signal capsule. These are built on a shared rail-based chassis that enables modular, efficient organization. Careful planning of internal component layout, with particular attention to wiring simplicity and space constraints, allowed over 10 tightly-packed components to be mounted securely. The power capsule features optimized placements for high-current components like ESCs and the battery, while the signal capsule prioritizes sensor alignment and modular accessibility. For instance, to support accurate orientation sensing, the IMU was placed near the system's center, and the fiber optic gyroscope was mounted in the yaw plane.

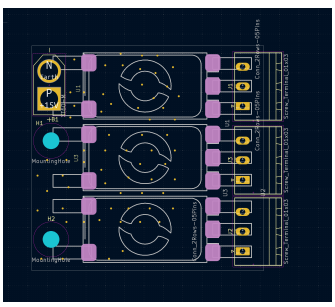
B. Electrical Design

Building on insights gained from Oogway's electrical architecture, we designed Crush's electronics stack to prioritize our philosophy of modularity and reliability.

1) Crush's Electronics Stack Architecture: Following last year's redesign of Oogway's electronics stack, we discovered the benefits of splitting the power system into two separate grids: one for low-power components, such as the CPU and gyroscope, and one for high-power components, such as the thrusters and battery. Despite these efforts, we still noticed drift and error in sensor readings, especially from the IMU, which is critical for determining the robot's state.

To address these issues, we designed Crush from the ground up with full electrical isolation between these two power grids. We placed the low-power electronics inside the signal capsule, and moved the high-power components into a second power capsule. This layout reduces electromagnetic interference and makes it easier to service each part of the system independently.

We began by prototyping the two power grids on wooden boards to visualize how the components would connect. We found that six inter-capsule connections were required. Because the signal capsule does not require high current, we used a waterproof Ethernet cable to route these lines, leaving two extra conductors as backups. For modularity, we connected all eight lines to a screw terminal inside the capsule, so we could route them to any component as needed.



(a) Schematic of ESC PCB

(b) Physical ESC PCB with 3 ESCs Mounted

Fig. 4: ESC PCB Schematic and Physical Layout

2) Printed Circuit Boards for ESCs: This year, we have also added two custom PCBs to Crush's battery capsule, each one holding three ESCs, enough to control all six thrusters (see Fig. 4). We designed these boards to fit cleanly within the mechanical constraints of the capsule and to simplify power distribution.

Each ESC receives power and sends signals through screw terminals, which enables fast and modular rewiring. We also used thermal vias and a strategic board layout to manage heat inside the sealed capsule during operation.

3) Robot Agnostic Electrical Communication: As part of Crush's development, we also redesigned our central electrical communication system to be completely robot-agnostic. The system uses serial communication and includes two Arduinos on each robot: one for thruster control and one for managing peripherals like sensors and servos.

We abstracted all hardware-specific code from the main communication layer, which allows us to use the same system across different robots. A configuration file defines which peripherals are connected. Our object-oriented design also makes it easier to scale and adapt the system to new hardware (see Section II-C2 for more details).

4) Gyroscope and Modem Integration: This year, we added two major components to the electrical systems on both Oogway and Crush: a fiber optic gyroscope and an acoustic modem.

In the past, our most persistent issue was drift in Oogway's angular position caused by imprecise yaw measurements from our IMU. This year, we equipped both of our AUVs with fiber optic gyroscopes, which offer extremely precise yaw-axis angular velocity measurements and are significantly less sensitive to temperature fluctuations and EMI.

Additionally, the newly integrated WaterLinked M16 modems enable underwater communication between our vehicles. Thanks to their omnidirectional transmission capabilities [1], the modems can send and receive signals from anywhere in the pool without requiring precise alignment or targeting.

5) Acoustics: Our acoustics system includes a custom-designed PCB with four band-pass Chebyshev filters and amplifiers, one for each of the analog hydrophones on our robot. The filtered data

from these boards is processed, and the azimuth of the pinger is reported to a ROS topic.

Initially, we used a two-pole Butterworth filter [2], but its gentle roll-off and the frequency overlap between the pinger and noise resulted in both being equally amplified. To address this, we theorized that a Chebyshev filter, with its sharper cutoff, would be better suited for this task. While the Chebyshev filter provided a steeper frequency response, the noise and pinger signal overlap still results in both being simultaneously amplified.

In the future, we plan to implement temporal filtering techniques that use the known ping time interval. Additionally, we are also exploring methods such as cross-correlation with expected ping shapes [3] and envelope following [4] to help identify pings from noise within the frequency band. We plan to further tune the Chebyshev filter's ripple and gain to reduce distortion and better visualize the ping shape across varying distances.

C. Software Design

In addition to hardware advancements, our software team migrated our 50,000+ line codebase to the ROS 2 framework. We also redesigned several core packages to support a fully robot-agnostic architecture, laying the foundation for faster and more scalable development. Our software is open-source and available on GitHub [5], supporting the wider AUV community. See Fig. 26 and Fig. 27 for our software control flow diagrams.

1) ROS 2 Migration: A major project this year was the migration from ROS 1 to ROS 2. In past years, we used ROS 1 as the underlying architecture for robot processes; however, with ROS 1 reaching EOL in 2025, it was crucial to migrate to ROS 2. We currently use the Jazzy Jalisco distribution of ROS 2 which provides long-term support until 2029.

We used ROS 2 migration as an opportunity for new team members to get aquainted with our robot systems and ROS. The process of migrating over 50,000 lines of code lasted from September to February, resulting in the successful migration of Oogway's original codebase.

2) Robot-Agnostic Codebase: Duke Robotics always designs our code with reusability in mind, but this is the first year we have simultaneously supported multiple robots in production. Our systems now dynamically adapt to the robot in use by reading the `ROBOT_NAME` environment variable, which determines the launch files and node parameters that are used. We leverage ROS 2's transform library to localize each robot's sensors, motors, and other components with respect to its own coordinate frame. This enables different configurations, such as camera placements, IMU offsets, or motor IDs, to co-exist in a unified codebase.

For example, robot-specific thruster configurations are abstracted through a modular controls package. A wrench matrix is calculated from the specific thruster orientations and positions of each robot, allowing us to convert arbitrary velocity commands into precise motor outputs for each of our unique robots. The task planning system builds on this abstraction: tasks like "navigate around a buoy" are written without reference to a particular hardware layout. This unified architecture allows us to streamline testing between robots and create more forward-compatible code.

3) Computer Vision (CV): The primary focus of the CV team this year was to clean up accumulated technical debt by rewriting most of our CV pipeline. The motivation here is twofold: first, last year, we had many separate files that were very similar; this repetition was inefficient and led to longer testing cycles. Secondly, adding Crush as a robot increased our desirability for modular and robot-agnostic software systems.

This cleanup involved both our HSV filtering algorithms and our YOLO models running on DepthAI hardware. For DepthAI, we consolidated distinct files that differed depending on the hardware into a single, hardware-agnostic file such that whenever we update our CV pipeline, we only need to update a single DepthAI file. We made similar modifications to our HSV filtering system: instead of multiple files for essentially the same algorithm but subscribing and publishing to different topics, we made a single class that can be easily extended and customized for specific detections. Since the base class remains the same, it is significantly easier to make changes to our HSV filtering pipeline. Ultimately, CV refactoring resulted in a reduction in lines of code by a factor of three.

III. Testing Strategy

A. Test Plan

Both Crush and Oogway's testing suites comes from a combination of simulated and real-world environments. Our test plan is as follows:

1) Test all compatible components in simulation. This allows us to verify core functionality, identify bugs early, and iterate quickly without risking hardware damage.
2) Test each component physically on the robot to ensure all components fit and connect properly. We run component-specific tests in the pool and in the sink when possible.
3) Once all components work separately, test the integration of components in the pool to mimic a real competition environment. Pool testing allows us to test the robot on competition tasks and ensure that we can complete each task reliably.

Following this strategy ensured that every component is individually reliable and our fully assembled robots can consistently complete all *core tasks*.

B. Simulated Testing

1) Software: We utilized our custom sim environment in CoppeliaSim—which includes models of our AUVs and competition props—to test controls and task planning algorithms. Before empirical testing, we ran each of our *core tasks* in the sim to ensure our code completed the task correctly.

2) Acoustics: While developing the algorithm underlying the acoustics stack, we created a simulated acoustics program. This involved simulating a pinger source and calculating the induced pressure waves. This, combined with encoded hydrophone geometries and artificial noise, provided realistic data to test our algorithm with.

C. Empirical Testing

1) Mechanical: All water-facing parts underwent extensive functionality and water-resistance testing. This often involved long-duration submersion tests in a sink. Once passed, the testing would escalate to a diving pool. This comprehensive approach guarantees all vital components will remain watertight during operation. Non-water-facing components go through the standard prototyping process, which involves testing and iteration on dry land.

2) Electrical: To evaluate long-term performance, we ran each electrical component through extended testing. Subcon and epoxy connections also underwent 24-hour submersion testing. After individually validating each component, we carried out full system integration tests to ensure the systems functioned properly in conjunction with each other. Communication systems were also tested in an end-to-end fashion, using mock components to ensure the signal was properly transmitted from the sensor to the computer.

3) Software: We tested our controls and task-planning systems through our Foxglove GUI [6], which allows us to set PID constants and observe setpoints. We visualized the task flow of the robot to verify the robot's state and actions.

4) CV: To test our object detection algorithm, we set up the gate task and manually moved the robot around to test all vision angles. We also tested CV-sonar integration, by matching the two data streams. We then tested movement autonomously, verifying the reliability of our gate task.

D. Testing Results and Integration Testing

Towards the end of our season, we tested the robot with each of our *core tasks* to verify that we could reliably compete in a competition environment. In total, we spent over 500 hours in simulated and empirical testing environments, resulting in Oogway prequalifing for the 2025 RoboSub competition.

IV. Acknowledgements

Duke Robotics Club is primarily sponsored by Duke University's Pratt School of Engineering. We are grateful to Ali Stocks and the Foundry staff for housing and supporting us. We are also indebted to Director of Undergraduate Student Affairs, Tarina Argese, for helping us plan events, our advisors, Professor Boyuan Chen and Professor Michael Zavlanos, for technical assistance, and the Engineering Alumni Council for invaluable club advice. Furthermore, we owe our success to our longtime sponsors: The Lord Foundation, Duke Student Government, General Motors, and SolidWorks. Finally, we thank RoboNation, whose commitment to RoboSub empowers student engineers like ourselves to explore our passion for robotics.

REFERENCES

[1] “Modem-M16,” Introduction - Documentation, https://docs.waterlinked.com/modem-m16/modem-m16/.

[2] “Example: Second-Order Butterworth Lowpass.” Accessed: Jul. 01, 2024. [Online]. Available: https://ccrma.stanford.edu/~jos/fp/Example_Second_Order_Butterworth_Lowpass.html

[3] "Reducing the Noise Floor and Improving the SNR with Cross-Correlation Techniques" https://www.zhinst.com/ch/en/blogs/how-reduce-noise-floor-and-improve-snr-employing-cross-correlation-techniques

[4] "Envelope Following" https://www.perfectcircuit.com/signal/envelope-followers?srsltid=AfmBOoq5-b0DV7ezMXFFGOni5nmeKdoe92O5eXLn-7WDdJGSPQjtjqYc

[5] “Duke Robotics Club · GitHub,” https://github.com/DukeRobotics.

[6] “Foxglove - Visualizing and Debugging Your Robotics Data,” Foxglove. https://foxglove.dev/.

[7] “Greenroom-Robotics/ros-typescript-generator,” Greenroom Robotics, Jun. 01, 2024. [Online]. Available: https://github.com/Greenroom-Robotics/ros-typescript-generator. Accessed: Jun. 25, 2024.

[8] “Thruster User Guide,” Blue Robotics, https://bluerobotics.com/learn/thruster-usage-guide/.

APPENDIX A: TEST PLAN & RESULTS

A. Mechanical

Mechanical testing is arguably the most critical aspect of verifying the performance of our AUVs. A failure here would disable all other subsystems. As described in Section III-C1, water-facing components were put through multiple rounds of testing.

1) Crush Capsule Mounts: Crush's electrical system is split between two capsules: signal and power. The signal capsule contains the computer, USB hubs, and sensors. The power capsule contains the battery, ESCs, and other high-current components.

Both capsules use the same rail-based chassis. Components slide into place along four long rails and connect through a central wiring path. Parts that interface are placed close together to keep wiring simple.

Our design of the power capsule focused on maximizing space efficiency. We needed to fit over ten components, along with their wiring, into a very constrained volume. To support this, we used the same rail-based system designed for the signal capsule. This approach ensured the components stayed stable and properly aligned, minimizing any unwanted movement within the assembly.

Since the battery needed to interface with output wiring from the capsule, we placed it at the rear. We mounted a busbar on top of the battery, surrounded it with a protective 3D-printed casing, and seated it on a supportive base to distribute its weight and prevent any components from rubbing against it.

Most of the remaining components were arranged in the front half of the capsule. To make the most of the limited space, we often placed multiple components on a single mount. Because we used six ESCs, our original plan was to mount three on one frame and three on another, with the Arduino breakout board on the back of one and the 16-channel PWM generator on the back of the other.

However, this configuration didn't leave enough room for the battery. So we revised the layout, placing both the Arduino breakout and PWM generator on the back of one ESC frame and leaving the other frame empty to accommodate the rear half of the battery. This change not only created more space for the fuse and other smaller components, but also simplified the overall wiring.

The design of the signal capsule is done in a similar manner. While some components could be placed where it is convenient, the location of the IMU was important. Since there were two separate capsules and the IMU needed to be close to the center of both, it was restricted to being placed to the side of the signal capsule. The legs of each mount utilized either the same or similar structure as that of the power capsule, where each only allowed for one degree of freedom, which is the direction that is moving along the rails. The sliding mechanisms of the frames allow for easy replacement of parts if they were to fail.

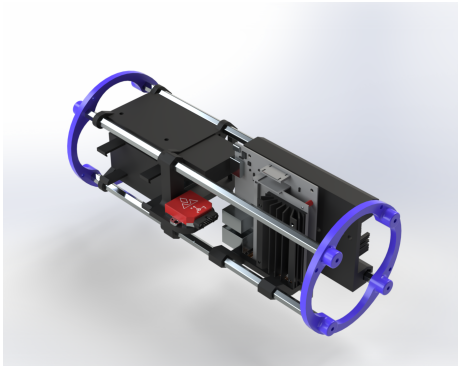


Fig. 5: Signal Capsule

2) Oogway Nylon Transition: Last year, we used PLA 3D prints for all of Oogway's external mounts. During testing, these parts slowly filled with water, which changed the robot's buoyancy over time and made tuning unreliable. We tried to reduce this effect by using 100% infill and sealing parts with nail polish. This year, we replaced all external parts with Nylon 12 printed on a FormLab's Fuse 1 Printer. Unlike PLA printed through FDM, Nylon 12 printed through SLS is always at 100 % infill and therefore does not absorb water, maintaining consistent properties throughout runs. The switch also improved strength and reliability without necessitating additional modification.

3) Oogway Upgraded Buoyancy System: Last year, we replaced zip-tied buoyancy blocks with a pole-based mounting system, which allowed faster adjustments, even while the robot was in the water. However, the PLA poles were fragile and often broke under stress. This year, we kept the 3D-

printed base but replaced the pole with a thin aluminum tube. This improved the system's strength and reliability, reducing the chance of failure during handling and testing.

4) Crush Buoyancy Mounts: We designed a new set of buoyancy mounts for Crush using Solidworks and VCarve. To avoid compromising the hydrodynamics of Crush, we created a tool path that matched the shape of Crush's sides and bottom. After experimenting with different placements, three large blocks were successfully mounted. We also added a smaller modular system to allow for fine tuning.

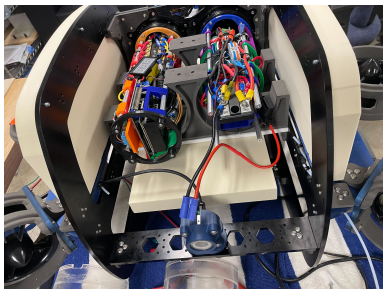


Fig. 6: Attached Buoyancy Blocks

5) Sensor Mounts: The following section outlines the design of Crush's sensor mounts.

General Component Mounting. We designed three different types of L brackets to mount all major components on Crush. Two of these brackets are used to mount the backplate of Crush (see Fig. 8 and Fig. 9) and a smaller, more robust L bracket is used for general mounting (Fig. 7).

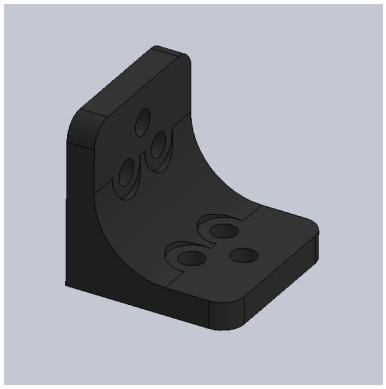


Fig. 7: Smaller, More Robust General Mounting L Bracket

Fig. 8: One Design of Crush Backplate L Bracket

Fig. 9: Another Design of Crush Backplate L Bracket

Capsule Mount. After finalizing the capsule mount designs (Fig. 10), the SLS-printed nylon capsule mount was attached to Crush's backplate. It holds the capsules at a set distance apart to prevent electro-magnetic interference between the components in each capsule.

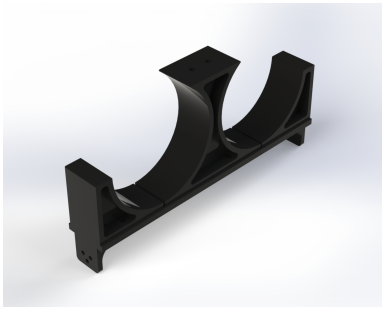


Fig. 10: Capsule Mount CAD

Thruster Mounts and Wings. To mount the thrusters (Fig. 12), cuts of aluminum wings (Fig. 11) are modeled in addition to the flat sections to mount three thrusters in a position that aligns with the center of the robot.

Hydrophone Brackets. An improved set of sen-

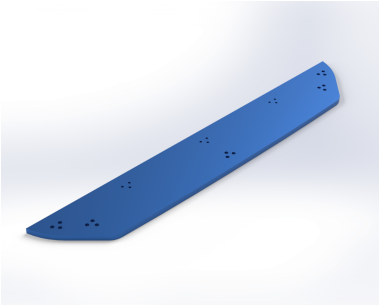


Fig. 11: Thruster Wing

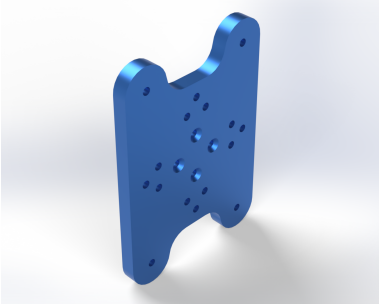


Fig. 12: Thruster Mount

sor mounts was redesigned to securely hold hydrophones and mount to different surfaces on Crush and Oogway. Using Solidworks, existing files were modified to match the mounting pattern on both AUVs. In addition, each mounting pattern was applied to both a straight bracket (Fig. 14) and an L-bracket (Fig. 13). The dimensions for the hydrophone-holding holes were tweaked to maintain a friction fit with the O-ring. After testing, the designs were finalized, 3D printed, and mounted.

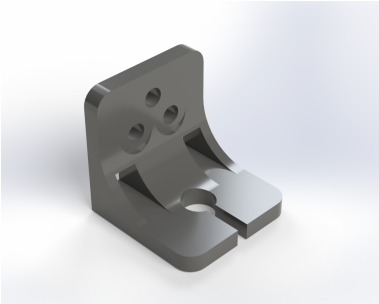


Fig. 13: Hydrophone Holder L Bracket

Marker Dropper. The existing marker dropper was redesigned in two ways.

1) The servo mount was redesigned to better accommodate the motor wiring on Oogway (Fig. 15 & 16).
2) The attachment mechanism was redesigned to be compatible with Crush's mounting holes (Fig. 17).

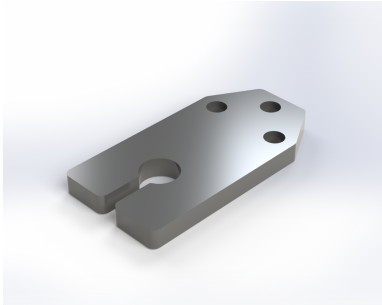


Fig. 14: Hydrophone Holder Straight Bracket

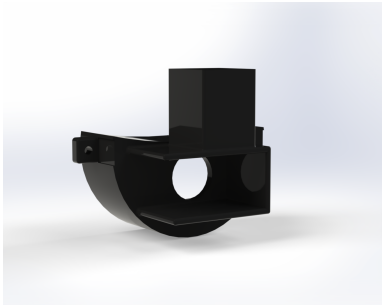


Fig. 15: Oogway Marker Dropper Back

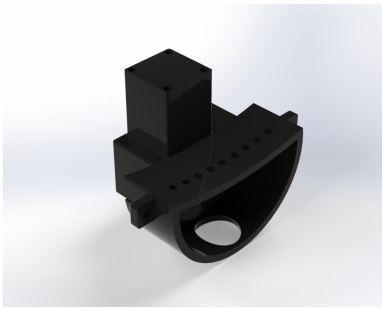
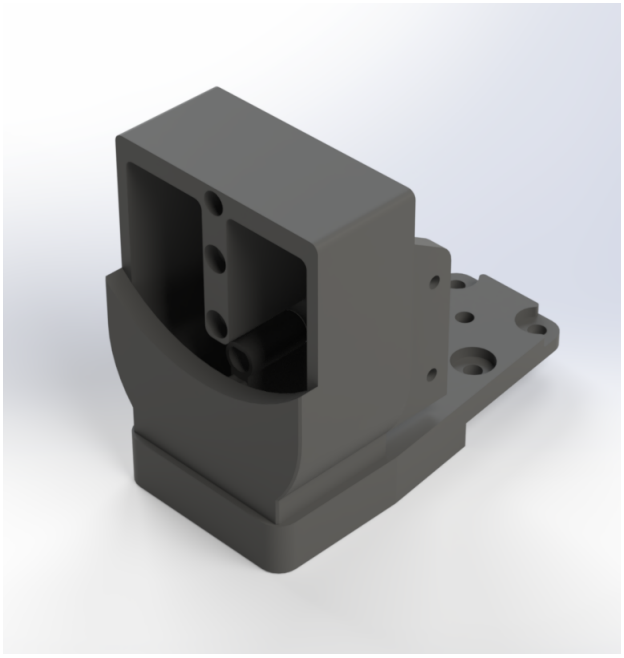


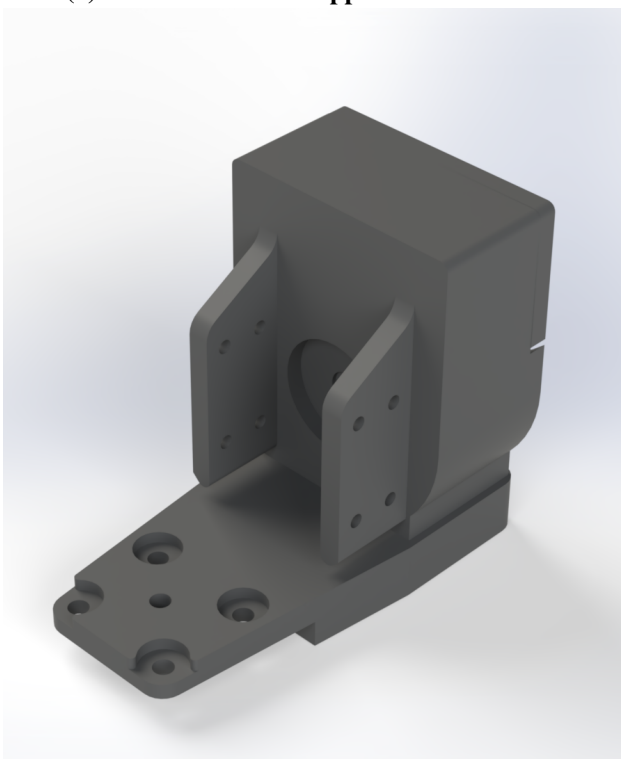
Fig. 16: Oogway Marker Dropper Front

B. Electrical

Rigorous testing is the foundational principle of our team's testing strategy. Our efforts focus on validating the core responsibilities of the electrical subsystem, power distribution, and data acquisition, in addition to research and development surrounding our acoustics system.



(a) Crush Marker Dropper without Cover



(b) Crush Marker Dropper Mounting Pattern

Fig. 17: Crush Marker Dropper Views: (a) Without Cover, (b) Mounting Pattern

1) General Component & Integration Testing: The electrical team prioritizes staged validation to ensure system reliability and reduce integration issues. Before any component is installed onboard the robot, it undergoes comprehensive functional testing on land and in isolation from the rest of the stack. This approach ensures that each subsystem operates correctly under known conditions before being subjected to the complex dependencies and constraints of our robotic systems.

Given the variety of components, from power regulators and motor controllers to environmental sensors and networking modules, this incremental approach allows the team to diagnose hardware issues, firmware bugs, and signal noise without the confounding variables introduced during in-robot testing. We conduct bench tests using custom-built systems and breakout boards, mimicking the real environment as much as possible while ensuring the system is easy to debug. This allows us to replicate conditions such as expected voltages, communication protocols, and signal timings.

Once a component passes standalone testing through its designated communication interface, it is temporarily integrated into the robot with minimal attachment points. This approach verifies proper functionality within the real system environment while maintaining ease of removal for rapid iteration and modification.

By the time full system integration occurs, each piece of the electrical stack has been tested both in isolation and within its local control domain. This greatly increases confidence in the system's stability and accelerates debugging when faults occur during integrated operation. Only after passing these staged tests does a component undergo underwater testing.

This pre-integration methodology has also reduced wear on critical hardware. Testing on land with external power supplies and development microcontrollers prevents unnecessary cycles on our main compute and power systems.

The testing workflow remains a core part of our design philosophy—ensuring that every connection, signal, and component is understood and trusted before it hits the water.

2) Acoustics Testing: Building on lessons from previous years, we opted to completely redesign our PCBs from the ground up. This redesign allowed us to explore new filter architectures—specifically Butterworth and Chebyshev filters—which offered

significantly greater customization and precision in signal conditioning.

Our testing pipeline begins at the earliest stages of design. Once a filter type is selected, we prototype it on a breadboard to approximate its behavior with real hydrophones and pinger signals. This validation helps us evaluate the filter's real-world effectiveness before committing to a final layout.

Following successful prototyping, we move on to schematic design and PCB fabrication. The printed circuit boards are then assembled and undergo rigorous system-level testing to assess their performance within the full acoustic signal chain.

Although these designs are still undergoing refinement, particularly in tuning filter parameters such as cutoff frequency, ripple, and gain, we anticipate that the resulting higher-fidelity acoustic data will enable more advanced signal processing algorithms to reliably detect pinger outputs, even in noisy or dynamic underwater environments.

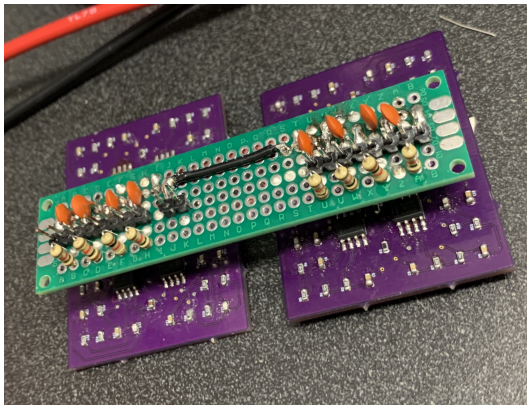


Fig. 18: Acoustics Filter and Amplifier Boards with BPF Backpack

C. Software

This section outlines our software testing procedures.

1) State: Our testing plan for diagnosing issues with robot state is as follows:

1) First, restart the robot, all Docker containers, and ensure that the Arduino is connected. Then turn the power DVL switch on.
2) Verify that all of the DVL, IMU, and pressure sensor are outputting to the correct ROS topics.
3) Verify that the robot-localization ROS package is running.

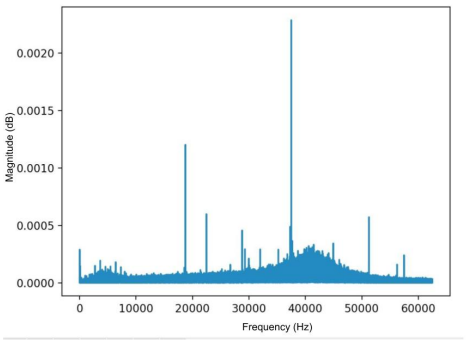


Fig. 19: Magnitude of Frequency Response of Filters in Pinger-less Pool. Bandpass 'hump' visible at ∼40kHz is about the same as the frequency range of our pinger.

4) Manually move the robot forward, to the left, and up. Verify that the DVL's velocities in each of these directions is positive.
5) Manually rotate the robot counter-clockwise about the z-axis. Verify that the IMU's yaw value is increasing.
6) Leave the robot stationary and observe the frequencies of the DVL, IMU, and pressure sensor. Verify that they are all above 20Hz.
7) Leave the robot stationary and observe the `/state` topic's covariance matrices. Verify they are not increasing over time.

2) Controls: Our custom controls package demands careful yet thorough testing. To prevent the robot from making any sudden, unintended moves, we disabled the thrusters during these tests.

1) We first checked that when controls received a setpoint, it computed the correct error values, which determine where the robot will attempt to move.
2) Once the errors were accurate, they were fed into the PID loops, which computed their integrals and derivatives. We made sure that these were the correct values, and tuned the parameters of the Butterworth filter for optimal smoothing of the derivative.
3) We then checked that the control efforts output by the PID loops would lead to the robot achieving its desired state.
4) We checked the dynamic offset added to the PID outputs to counteract the robot's positive buoy-

ancy. We rolled, pitched, and yawed the robot different directions and ensured that the offset vector always pointed straight up, regardless of the robot's orientation.

5) We then verified that the thrust allocations obtained from our quadratic programming solver matched our expectations.
6) We modified the PID gains and other values on-the-fly, and ensured that they updated the system's behavior as expected and were saved to disk so they could be reused in future runs.
7) With all parts of the system independently tested and verified to be working as expected, we finally enabled the thrusters and allowed the robot to move using the new controls system. The robot behaved as we expected, and didn't perform any movements that would damage itself.

3) Foxglove GUI: By automatically converting ROS message schemas to TypeScript types [7], we are able to enforce static type checking to validate the connections between our landside GUI and onboard ROS system. This means that *at development-time*, TypeScript will alert us if any concurrent work done on our ROS stack (e.g., a change to our PID service API) will interfere with our Foxglove GUI stack. This allows us to catch numerous bugs early, before any actual testing has occurred.

APPENDIX B: FULL ELECTRICAL SYSTEM DIAGRAMS

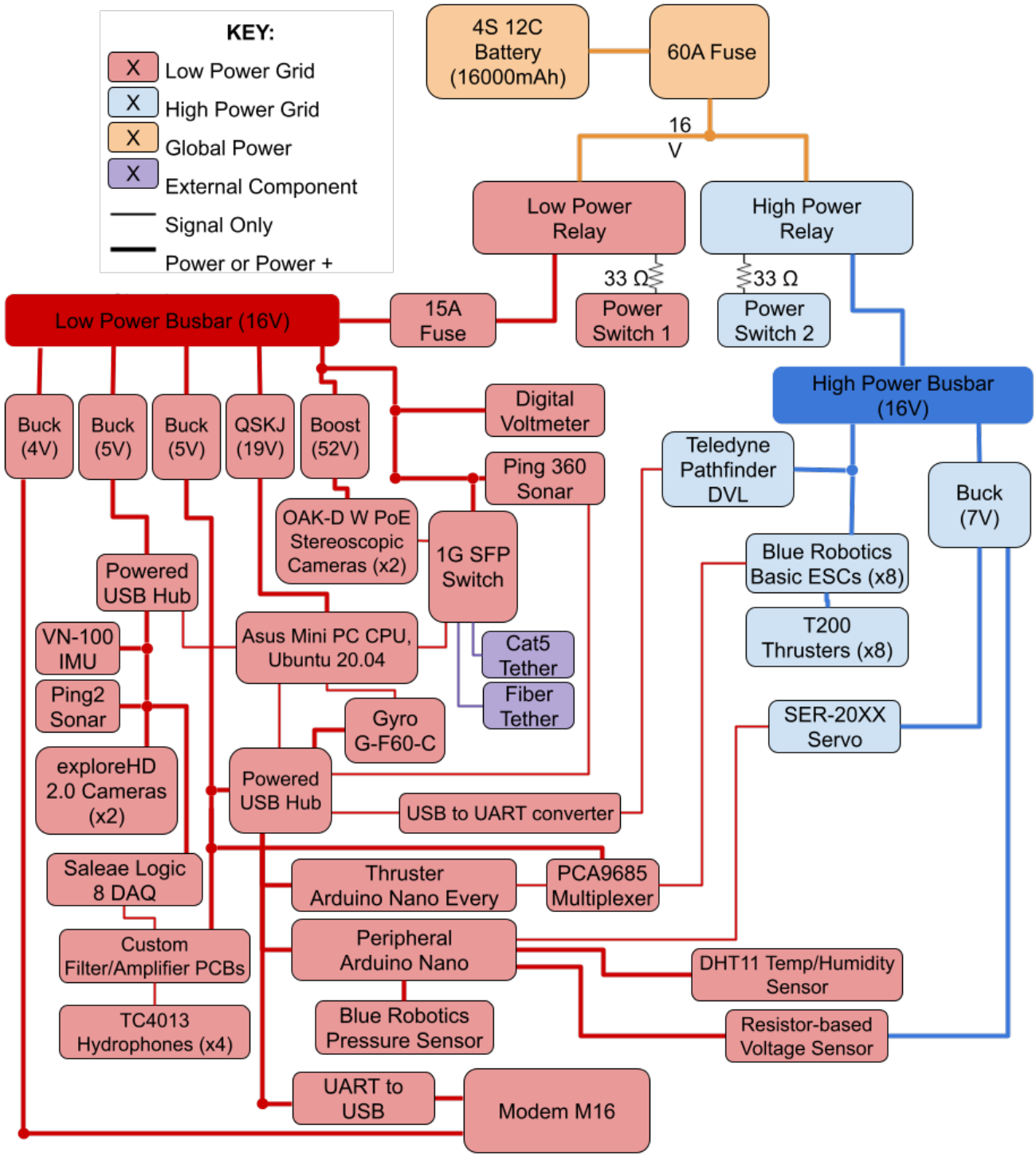


Fig. 20: Oogway's Electrical System Diagram

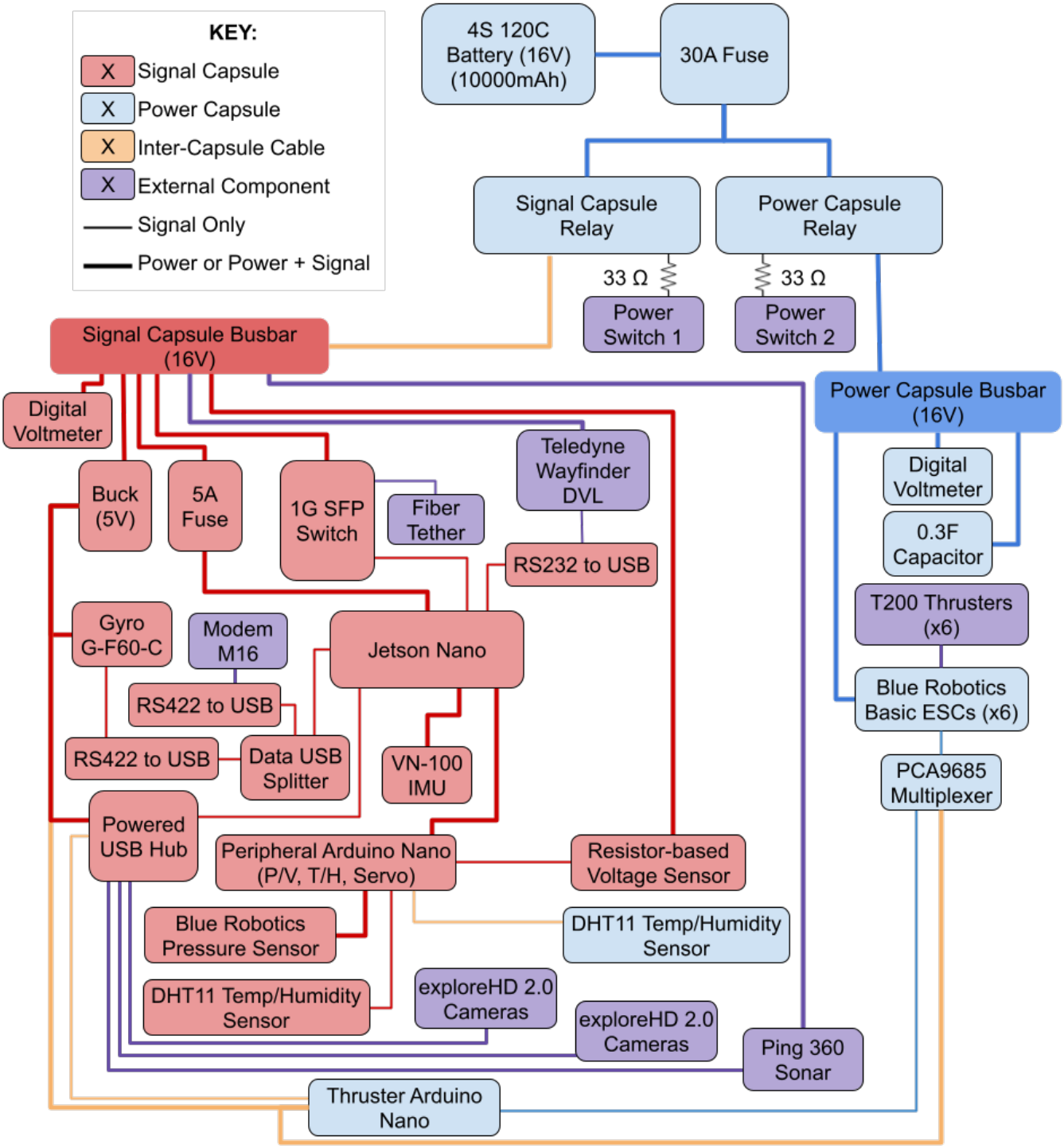


Fig. 21: Crush's Electrical System Diagram

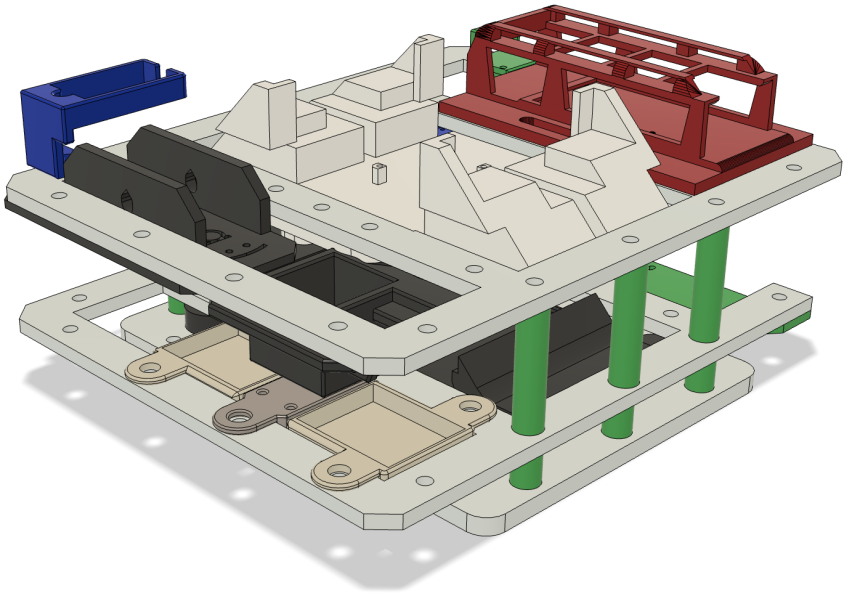


Fig. 22: Oogway's Electrical Stack Mounts – CAD

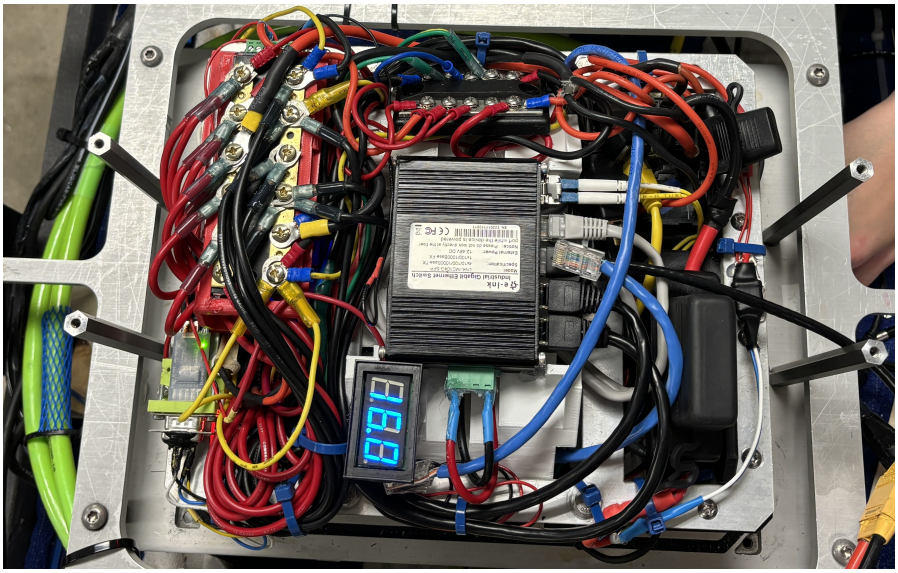


Fig. 23: Oogway's Electrical Stack – Populated, Top View

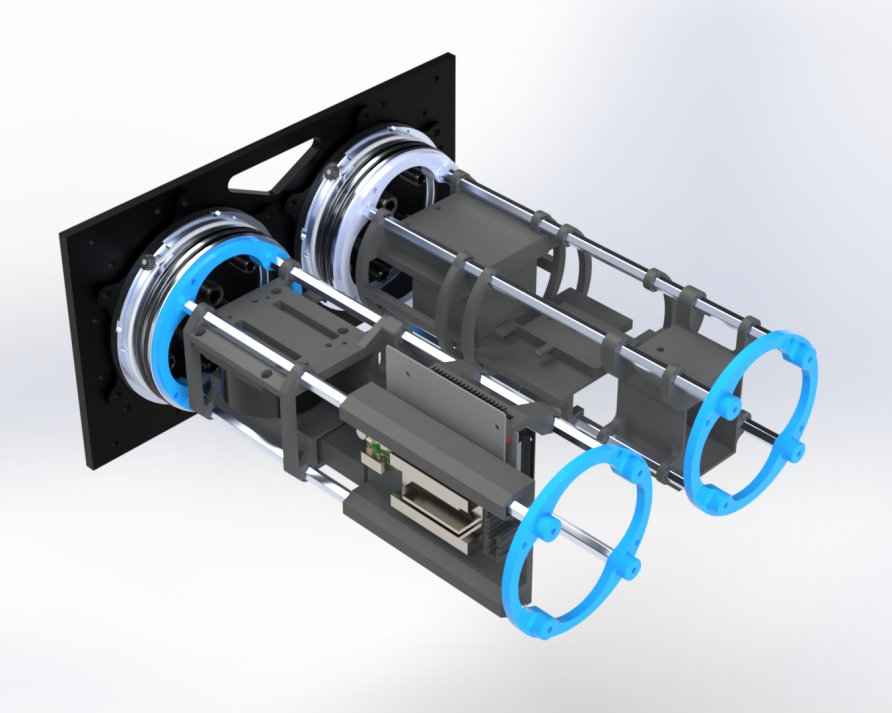


Fig. 24: Crush's Electrical Stack Mounts – CAD

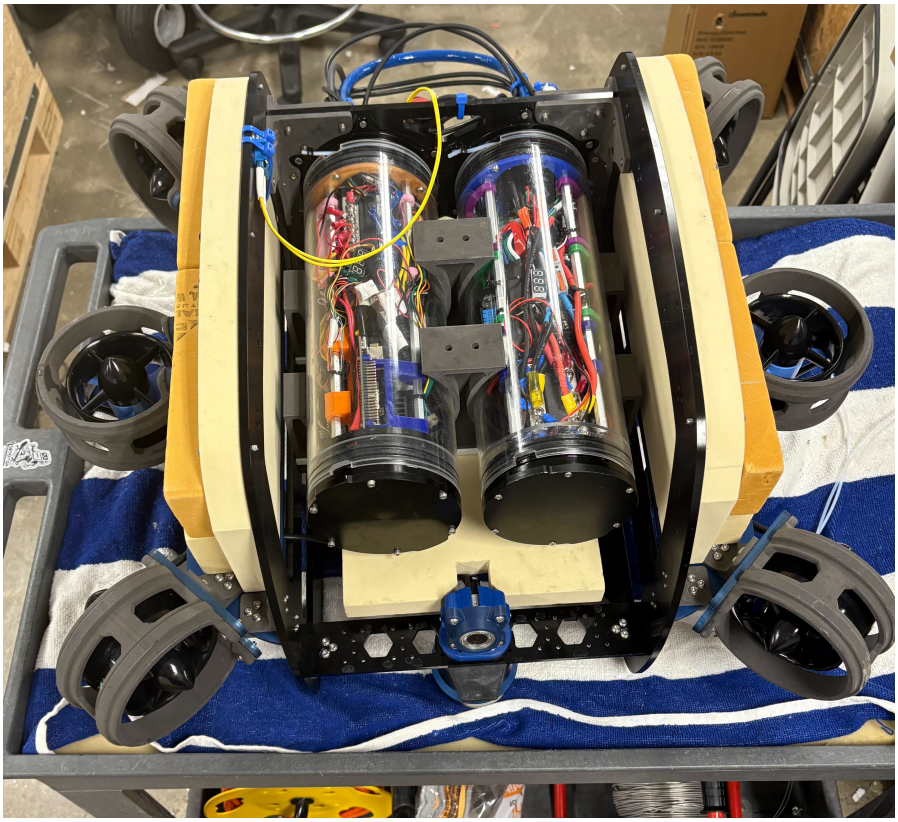


Fig. 25: Crush's Electrical Stack – Populated, Front View

Appendix C: Control Flow Diagram

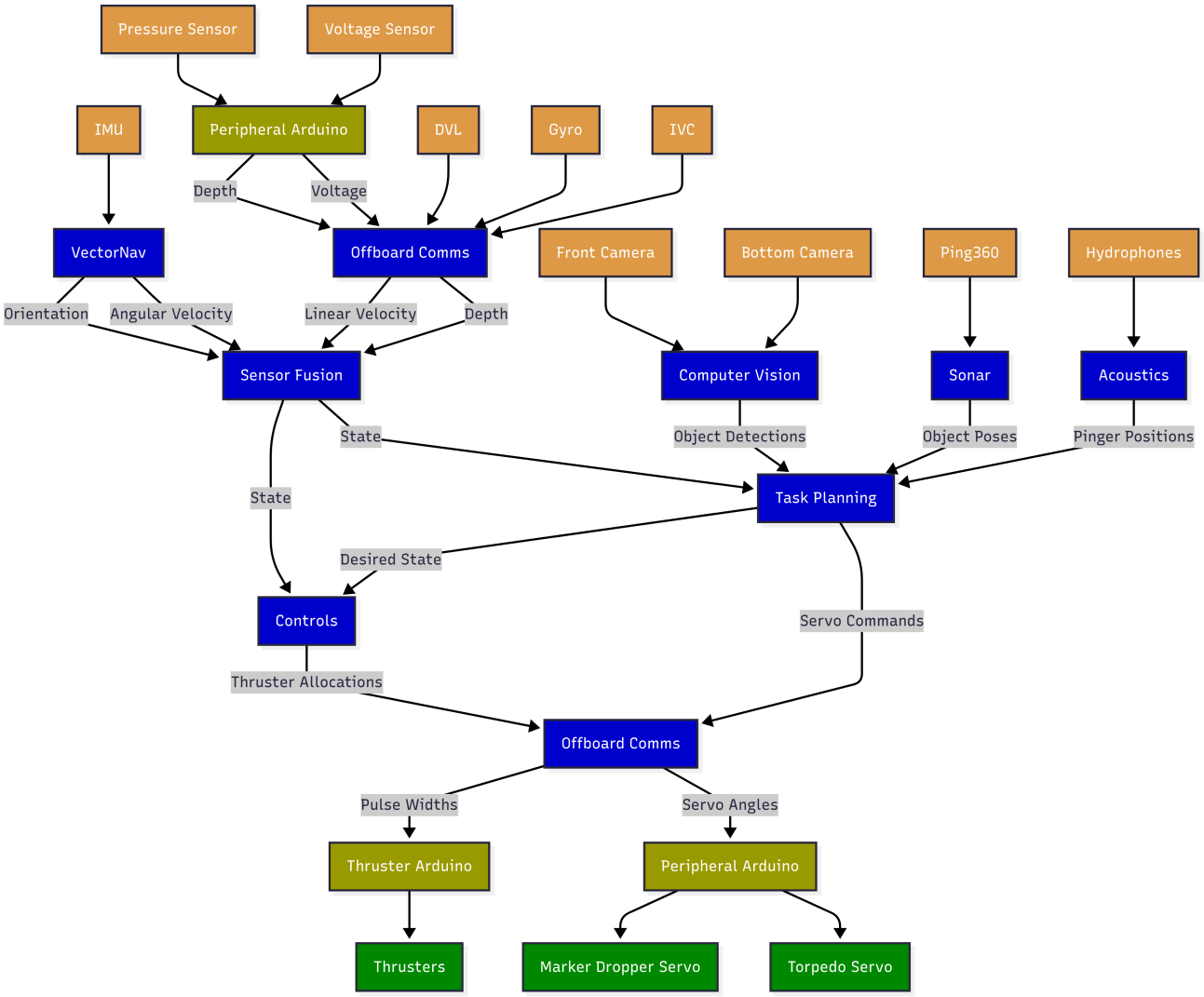


Fig. 26: Oogway's Software Control Flow Diagram

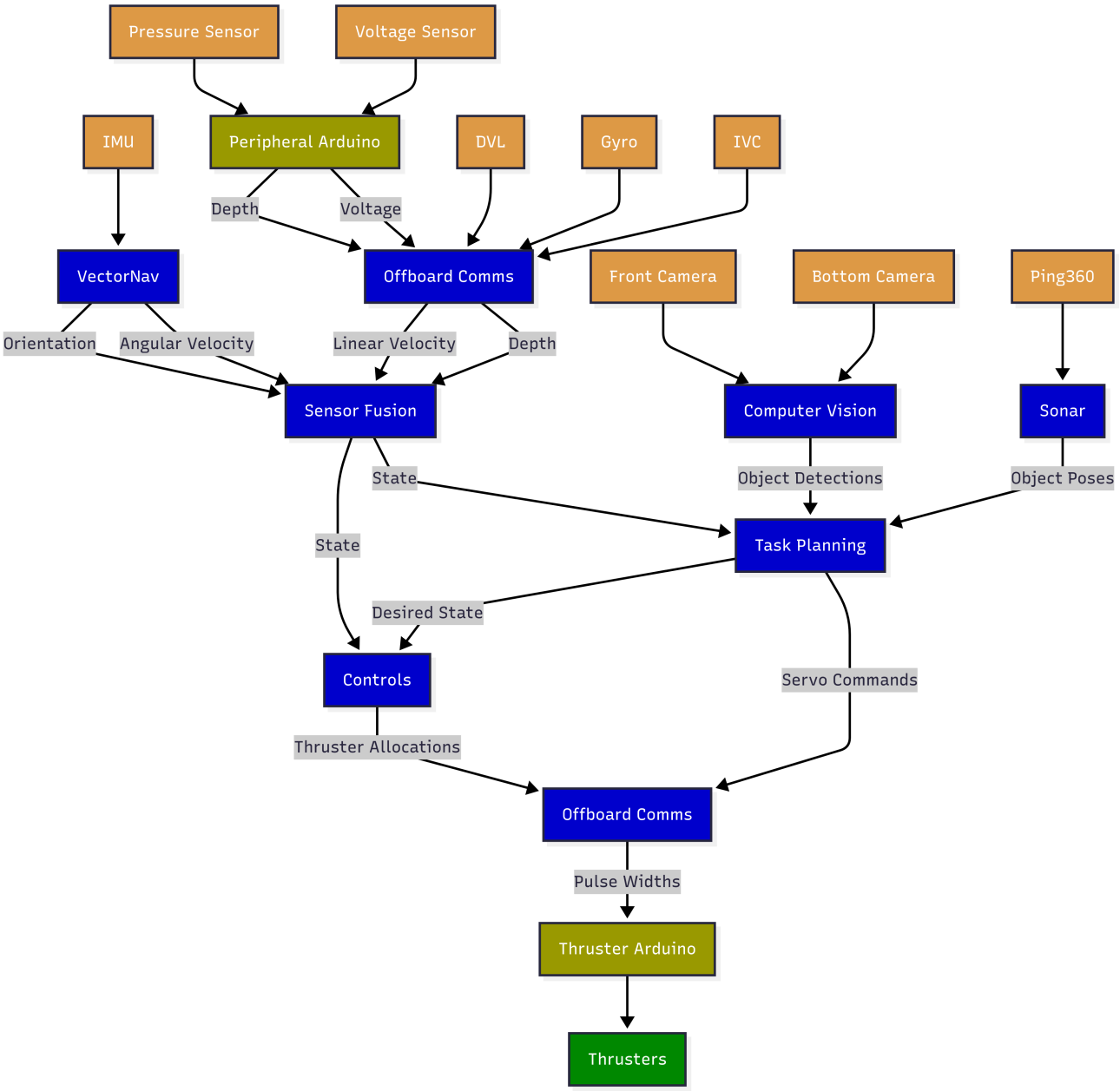


Fig. 27: Crush's Software Control Flow Diagram

APPENDIX D: THRUSTER ORIENTATIONS

When designing our new robot Crush, we took a new look at our thruster orientations to minimize unwanted torque caused by the spinning propellers. This was especially important because Crush does not have active pitch controls, and so we had to ensure that the arrangement of thrusters would not adversely pitch the robot while moving.

In this process, we reviewed the Blue Robotics thrusters guide [8], and noticed that the type of propeller installed (clockwise versus counter-clockwise), played a big role in determining which direction could produce the most thrust. If some thrusters were spinning in the optimal direction and others were not, this would create an imbalance, leading to unintentional drift, and residual torque.

To address this problem, we first identify the factors determined the directions the thrusters should spin in:

- **Propeller Type:** The type of propeller (CW or CCW) determines the optimal spin direction.
- **ESC Wiring:** The ESC wiring determines the actual direction the motor spins at a set PWM signal.

For clarity, we also defined two types of ESC configurations:

- **Standard ESC:** Given a PWM signal > $1500\mu s$, the thruster spins clockwise.
- **Flipped ESC:** Given a PWM signal > $1500\mu s$, the thruster spins counter-clockwise.

Now, based on how the 4 combinations of ESC configuration and propeller types are paired, we can classify the thrusters into two categories:

- **Normal Orientation:** A standard ESC with a CW propeller, or a flipped ESC with a CCW propeller, will produce the intended forward thrust when given a PWM signal above $1500\mu s$.
- **Reverse Orientation:** A standard ESC with a CCW propeller, or a flipped ESC with a CW propeller, will produce reverse thrust when given a PWM signal above $1500\mu s$.

To reconcile these differences, for each thruster, we tested the ESC to determine its configuration, then the propeller to determine its direction. For all reverse orientation thrusters, we flipped the PWM signal across the midpoint of $1500\mu s$.

This setup allows our controls algorithm to send each thruster a value indicating the desired forward thrust. A separate translation layer then converts this value into the correct PWM signal based on the thruster's orientation. This abstraction simplifies our control code and ensures that all thrusters respond predictably, regardless of their physical configuration.