Cyclone RoboSub

at UCDAVIS

Technical Design Report

2025-2026

2026 Technical Design Report

Leadership: Peter Webster, Jason Pieck, Luci Masselink, David Kou, Alexander Zamora, William Barber, Devi Amarsaikhan, Aditi Anand, Aubri Lee, Kekoa Olive, Allen Cooke, Anjani Gadkari, Nidhi Vadulas, Dilraj Gill, Luna Park, Danny Kwong, Michael Tisdale

Advisors and Mentors: Dr. Shima Nazari, Abhigyan Majumdar, Kory Haydon, Ziqiang (Joe) Zhu

***Abstract*—After their debut in RoboSub 2025, Cyclone RoboSub has spent the past year iterating on our vehicle, Manatee, to expand autonomous capabilities and ensure reliability for both RoboSub 2026 and environmental research deployments. While our competition strategy has remained similar to that of 2025, we've placed a renewed emphasis on testing and validation by implementing a rigorous weekly pool test schedule to ensure vehicle reliability. This year, Manatee features a new set of PCBs, updated hardware, and a rewritten software stack and MATLAB Simulink controller. By exercising methodical decision making and frequent testing, we aim to optimize our reliability and, in turn, maximize our performance at RoboSub 2026.**

I. Competition Strategy

A. Previous Competition Analysis

Our first conversations regarding competition strategy began with the team's founding in 2023. As a new team, we understood the importance of setting achievable goals, so we made a point of cultivating a solid understanding of the competition challenges to inform our team's expected abilities.

We began by breaking down the 2023 Team Handbook [1] to assess strategy. We cataloged the maximum available points associated with each task, along with the number of teams able to successfully attempt each task as seen in Appendix A. This allowed us to identify trends and pinpoint tasks that yielded the most points while falling within most teams' estimated abilities.

During our analysis, we noted that the majority of teams' points were earned by completing navigation-based tasks such as surfacing in the *Octagon* and passing through the *Gate*. When compared with precise manipulation tasks, we found that successful attempts were very uncommon. Based on this research, we prioritized navigation-based tasks while deprioritizing tasks highly reliant on manipulation and vision systems.

We repeated this analysis using the 2025 competition results and found the same trend as shown in Table II. The addition of the *Slalom* task created an even larger reward for prioritization of accurate navigation. These results have continued to motivate our focus on navigation-based objectives over manipulation tasks and is reflected in our competition strategy.

B. 2026 Competition Strategy

Similar to 2025, this year we are prioritizing navigation-based tasks which are not reliant on vision systems. This ensures that we are not forced to compromise any aspects of our competition strategy in the event that our vision pipeline is not ready for competition. Our strategy is as follows:

- **Begin Assessment (*Gate*):** Manatee begins each run with a coin-flip, turning to face the gate. The vehicle proceeds to submerge and pass under the right side of the gate, after which it will pause and complete two 360° roll rotations for style points.
- **Avoid Debris (*Slalom*):** Manatee next attempts the *Slalom* navigating between pre-defined waypoints.
- **Recon (*Bins*):** Then, Manatee navigates to the *Bin* task using a preset waypoint. Once the vehicle is positioned above the bins with the images in sight, a downwards facing camera provides localization information allowing Manatee to precisely line up above the bin of interest. once we are within target position range, both markers will be released.
- **Resupply (*Octagon*):** Manatee will navigate to the *Octagon* using a predefined waypoint, surface, and descend again to begin the journey home.
- **Return Home:** Lastly, Manatee will navigate back to the *Gate* using waypoints and pass through, surfacing, and completing the run.

This competition strategy applies the lessons learned from previous competition analysis, providing the baseline framework that informs of design requirements.

II. Design Strategy

Manatee was designed with simplicity and efficiency in mind. As our team's first vehicle, we chose to stick to well tested and proven solutions. Every part of our robot was inspired by extensive research and underwent multiple iterations and design reviews based around our design requirements detailed in Appendix B.

A. Design Requirements

Shortly after RoboSub 2025, our team committed to re-enter Manatee for the 2026 competition to ensure a stable platform for our software team to test and integrate systems year-round. To enable this testing plan, the team identified specific improvements that could be implemented while preserving the vehicle's core functionality. These improvements are a direct response to design goals which were tabulated into a set of mission requirements and, based on these mission requirements, a set of system requirements. Other design choices were informed by the vehicle's intended contribution to environmental research, as detailed in Appendix N. A full table of design requirements can be found in Appendix B.

B. Hull

a) *Chassis:* Manatee's chassis is a ¼-inch aluminum base plate cut on a water jet with a grid pattern of mounting holes. The modular hole-grid layout allows components to be easily mounted, adjusted, and swapped out. Manatee's

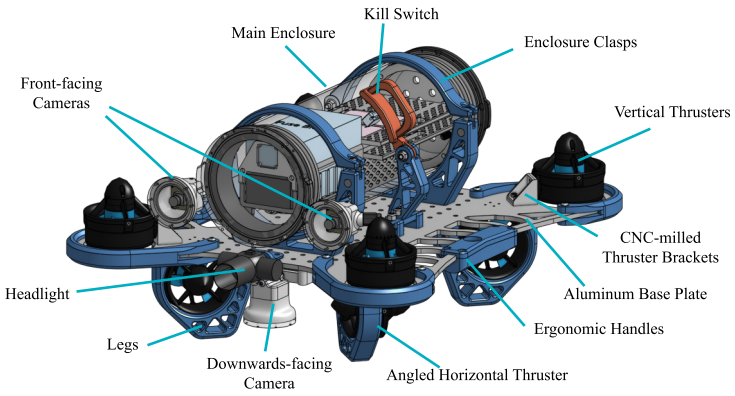

Fig. 1. CAD model of Manatee

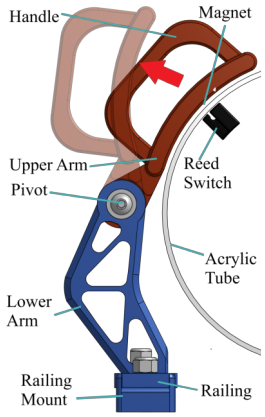

Fig. 2. Kill Switch System

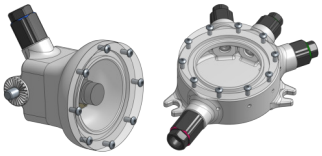

Fig. 3. Camera enclosure and junction splitter CAD

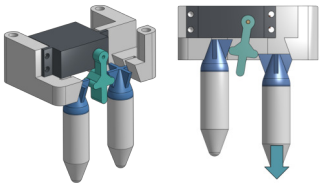

Fig. 4. Dropper Assembly CAD

primary waterproof enclosure, a 6″ acrylic tube, is mounted to this plate with a hinge-and-clasp system. To validate this enclosure's water-proofing, the team follows a vacuum test procedure outlined in Appendix D.

As seen in Fig. 1, the propulsion system follows a standard symmetrical layout [2], with two thrusters in each corner: one angled horizontally beneath the chassis and the other mounted vertically in line with the base plate. The angled horizontal thrusters allow Manatee to propel itself forward, sideways, or control yaw independently from roll and pitch. Meanwhile, the vertical thrusters allow independent control of the pitch, roll, and altitude of the vehicle.

b) *Stereoscopic Camera Enclosures:* This year, the controls and vision subteams wanted to implement stereoscopic vision as well as a downwards-facing camera, serving to meet system requirement SR-S- 2 and mission requirement MR- 1. Stereoscopic vision necessitates external camera enclosures, because the acrylic tube enclosure is not wide enough to give adequate camera spacing for an off-the-shelf stereo camera.

After looking into off-the-shelf enclosure solutions, we found research [3] indicating that SLA 3D-printed enclosures can achieve depth ratings far exceeding the depths we operate in, making them a viable solution for our camera enclosures.

Our custom enclosure, as seen in Fig. 3, is printed out of Formlabs Clear V4.1 Resin, which was selected selected for its high stiffness, low hygroscopy, low cost, and its ability to hold a strong tapped thread. There are three copies of this enclosure on Manatee: two facing forward for stereoscopic vision and one facing downwards. Inside each enclosure is a camera powered through a Blue Robotics Wetlink Penetrator. The camera sees through a clear ¼-inch acrylic plate. Between the acrylic plate and enclosure is an axially compressed O-ring to seal the enclosure. The enclosures are affixed to the robot with a bolt and a toothed crown for angular indexing.

To minimize the number of penetrators these cameras take up on the main enclosure, a 3-to-1 junction box was also manufactured via SLA 3D printing, shown in Fig. 3. In this box, the three camera cables merge into a single, 12-wire cable that only takes up one penetrator slot on the main enclosure.

c) *Kill Switch:* This year, the kill switch was redesigned to meet system requirement SR-ME- 1 and mission requirement MR- 3. The previous kill switch (a twist knob located between penetrators) proved to be inaccessible and potentially unsafe, prompting a redesign.

The new design uses a magnetic reed switch activated by a large pivoting handle, as seen in Fig. 2. The handle is centered on the robot, which allows for access from all angles and is a safe distance from all thrusters. The handle contains a neodymium magnet that aligns with the reed switch located inside of the acrylic tube. To disable the thrusters, the handle is pulled away from the tube.

d) *LED Indicator Lights:* To make operating Manatee more intuitive in alignment with system requirement SR-S- 5, the team decided to implement programmable RGB LED indicator lights. These lights are visible from the water's surface even when Manatee is fully submerged, and flash different patterns to communicate different operation statuses. A table of these patterns and their meanings can be found in Appendix K. When designing these patterns, the team adhered to specific principles for intuitive communication, which can be found in Appendix K.

The LED strips are controlled with the onboard ESP-32 microcontroller, which lets the lights communicate software crashes even if the main Raspberry Pi processor goes offline.

e) *Thermal Considerations:* To better assess Manatee's internal thermals, we conducted a series of tests detailed in Appendix J. Ultimately, we concluded that the duration of our run times and the heat generated were not a primary cause for concern and did not require active mitigation besides the existing systems which include a heat bank mounted to the ESC's and a small Raspberry Pi fan.

C. Manipulation

a) *Dropper:* The dropper is a servo-actuated mechanism used to individually release two small markers for the *Bin* task during competition runs. Once Manatee determines the location of the *Bin*, it centers itself and releases a marker using a servo-controlled cam that interfaces with the markers' fins, as seen in Fig. 4 . The specific cam profiles were iterated to maximize release reliability and consistency. The markers themselves consist of an aluminum head and 3D printed fins. Manatee's dropper servo receives signal commands from the onboard ESP-32, and is powered by the buck converter.

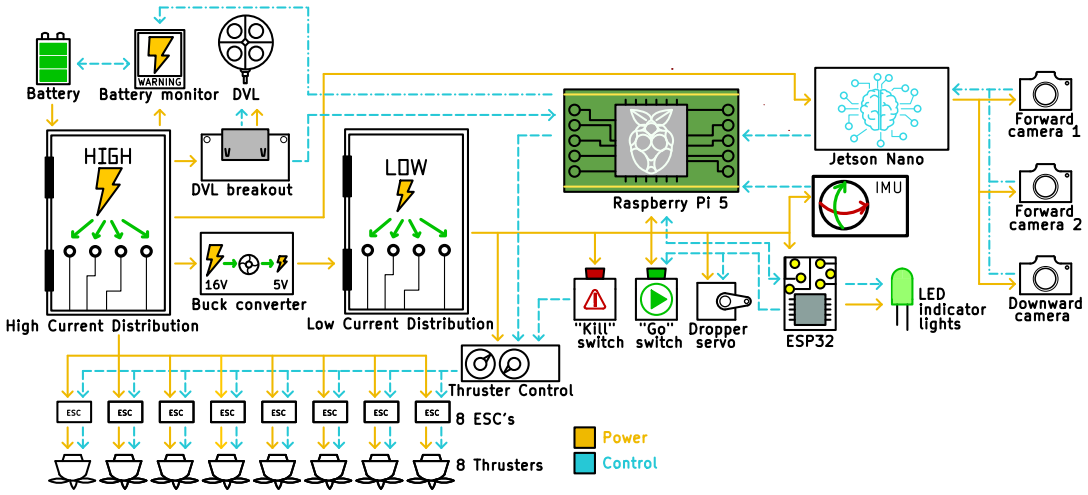

Fig. 5. Electrical Diagram

D. Electrical System

a) *Power Distribution:* Manatee operates off of a single 4s Lipo battery which supplies 16.8V to a fusebox which acts as our primary power distribution system. A buck converter provides a regulated 5V, 5A supply to all of the vehicle's microprocessors as seen in Appendix I. These include a Raspberry Pi 5 for primary control, an ESP-32 for the LED display, and a Pi Pico for thruster commands. A capacitor bank helps to regulate power to the processors and prevents voltage drops encountered in early testing.

b) *Battery Management System:* This year, we designed a Battery Management System (BMS), allowing us to meet requirements SR-ME- 1 and SR-ME- 4. The PCB will monitor and protect the 4-cell LiPo battery pack used to power Manatee. Refer to Appendix I for an in-depth explanation of our BMS.

c) *Thruster Control Board:* In accordance with requirement SR-ME- 3, we have designed the thruster control board to replace the previous protoboard in use. Implementing our thrust control onto a custom PCB improves reliability and allows for easier debugging. Refer to Appendix I for more information on the Thruster Control PCB.

E. Control System

a) *Overview:* Manatee's control system was designed with three unique operating modes in mind: **fully simulated, hardware-in-loop, and competition**. Each of these modes serves an important role in the development and testing process.

b) *Control Systems Competition Plan:* The competition run begins with the construction of a *mission file*. This file describes a decision tree of *commands* that the robot attempts to execute in series. Each command includes a *command id*, as well as multiple *command parameters* that define the behavior of the robot during operation. Commands are intended to be human-readable to allow rapid tuning of the mission file. This mission file is loaded into the robot via a wired connection prior to a run, then started using the "go" switch.

Within each controller time step, the following operations are performed:

1) The *Mission Manager* consults the mission file and the status of any ongoing commands and delivers a single command to the *Low Level Controller* which contains three elements in series: the *Command Executer*, *Guidance Law*, and *PID Controller*.

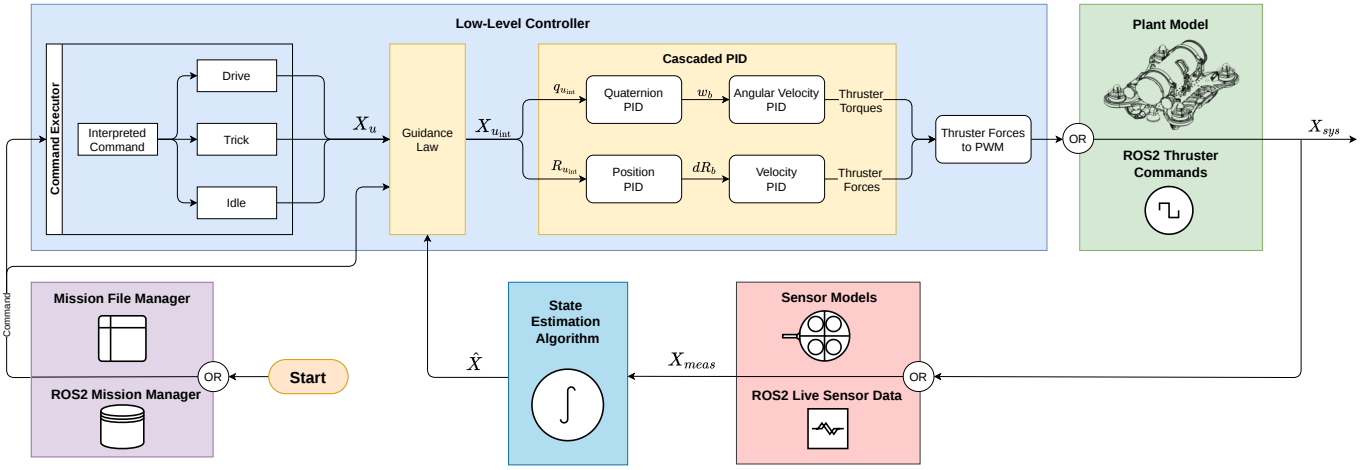

Fig. 6. Controller Diagram

2) The Command Executer parses the parameters of the command into a waypoint (position and attitude) in the world reference frame and sends it to the Guidance Law, then returns a *command status* to the Mission Manager evaluated based on sensor data, state estimation, the the command parameters.
3) The Guidance Law generates an intermediate waypoint based on the waypoint from the Command Executer to break up the maneuver into distinct turning, driving, and settling components depending on the distance to the waypoint and the relative attitude of the body with respect to the waypoint direction.
4) The PID Controller accepts the intermediate waypoint from the Guidance Law and generates the appropriate thruster commands to translate and rotate the robot to the waypoint.

Steps 1-4 are iterated through until the decision tree reaches an end point or the robot is deactivated using the "on" switch or the kill switch.

The design of the control system is subdivided between the Software and Control teams. The Software team handles sensor reading, thruster commands, subsystem communication, and the mission file. See Section II.E.e for more information on these topics.

c) *Controller Design in MATLAB and Simulink:* The Low Level Controller was developed using MATLAB and Simulink. Simulink was chosen both for the wide variety of useful design toolboxes and the low barrier of entry for engineering students with prior MATLAB experience. The Low Level Controller and the State Estimator were designed as a ROS2 [4] node that input sensor data and commands from the mission file, and outputs individual thruster commands. This node is then generated into C++ using the Simulink Coder Toolbox for operation on the robot.

The fully-simulated mode is intended to be run entirely on one computer without needing connection to the real robot. This mode is the most complex, because it must include

1) A model of *Mission Manager* to test the Low Level Controller's ability to execute commands and transition between commands.
2) The *Low Level Controller* containing the *Command Executer*, *Guidance Law*, and *Cascaded Controller*.
3) The plant model of the robot simulating 6-dof dynamics, thrusters, buoyancy, gravity, and hydrodynamic forces.
4) An environment and camera model using Unreal Engine co-simulation for development of computer vision algorithms.
5) A sensor model for the Waterlinked A50 DVL and the InertialSense IMU.
6) A state estimator to fuse camera, IMU, and DVL data into a state estimate.

The hardware-in-loop mode is the next step in the testing process. This mode is enabled by the Simulink ROS2 Toolbox. The controller is run in MATLAB on an external computer and tethered to the robot to command thrusters and read sensors. In this mode, items 1, 2, and 6 from the list above are run in Simulink, while the plant model, environment model, and sensor models are replaced with the real physics, cameras, and sensors.

Finally, in the competition mode, the Low Level Controller and State Estimator ROS node are exported to C++ from the Simulink model. The *Mission Manager* model is then replaced by the implementation from the Software team, as detailed in Section II.E.e. This three-part approach allows the validation of system level logic entirely in simulation so pool-testing time is spent debugging complicated integration issues, tuning the controller, and mission files.

The Simulink project contains various other useful tools, including parameter estimation tools for the drag and inertia matrices based on pool-testing data, custom animation and plotting code to speed up debugging, and controller tuning utilities to tune the 12 PID controllers in the cascaded PID controller. For a more detailed description see Appendix E.

d) *Computer Vision:*

The vision system was designed to support autonomous waypoint navigation by detecting mission objects and estimating their position relative to Manatee. To accomplish this, the team selected the YOLO-keypoint framework from Ultralytics [5] due to its real-time inference performance and flexibility across both monocular and stereoscopic vision pipelines.

For deployment, the keypoint model is exported to ONNX and optimized with TensorRT on the NVIDIA Jetson Orin Nano [6] for realtime inference. Vision outputs are then transmitted through ROS for integration with controls and system architecture.

Because underwater training data is limited, the team used a multi-source dataset strategy combining footage from previous competitions, controlled pool testing, and synthetic data generated in Unreal Engine. Camera calibration was performed to support consistent geometric interpretation of the camera images and to enable future depth-aware perception work. Stereo depth estimation and rectification were also developed as part of ongoing experimentation, but remain under active development.

Additional implementation details are provided in Appendix F.

e) *Software Integration:*

This year we rewrote our software from scratch, using a ROS-first approach from the start. To ensure safety, our ROS-based heartbeat system (explained below) stops the robot if any node in the control pipeline fails. ROS2 also provides a unified build and testing system: with one command each, our system can be compiled and tested, and run. In designing our system around small independent ROS nodes, we allowed for increased modularity: the API between nodes is defined by the topics, services, or actions they communicate with. Modifying, replacing, or simulating components becomes much easier with this approach, which has been particularly important for our MATLAB controller testing (see Section II.E.c).

Fig. 7 describes Manatee's software functions. Its primary components include the mux, thrust interface, mission manager, feedback controller, and operator interfaces. The feedback controller is discussed in depth in Section II.E, while the following aims to overview the remaining components.

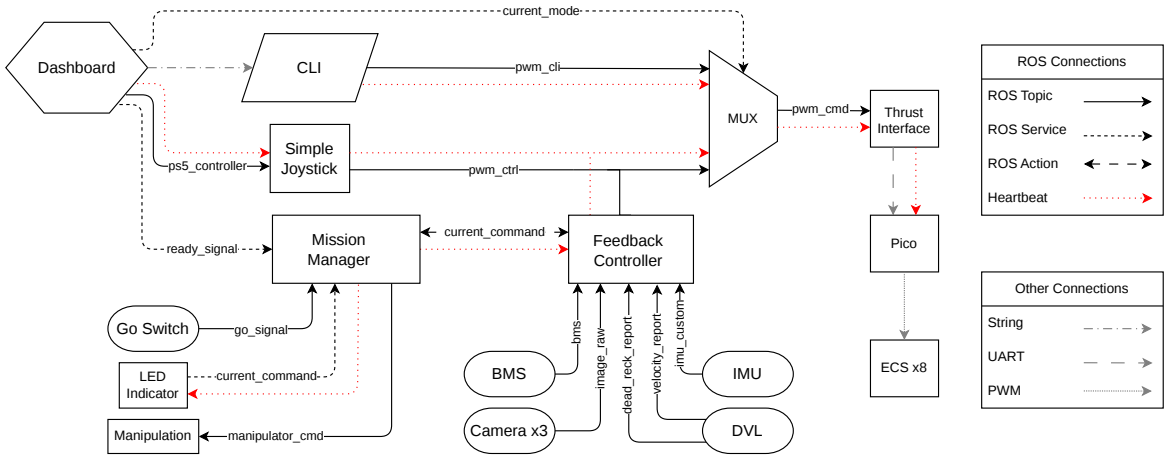

Fig. 7. Software Architecture Diagram

Autonomous mission execution is handled by the *mission manager*. Mission logic is defined as a composable behavior tree powered by BehaviorTree.CPP [7]. The library is paired with the Groot2 IDE [8], which allows the operator to modify the mission file graphically without modifying the underlying C++ code.

Manual control is available through two interfaces. Joystick allows for manual testing and simplifies poolside recovery, and the CLI allows for scriptable invocation for automated testing.

The mux serves as the software arbiter of the robot, forwarding the specified thruster command channel to the thrust interface. It lets the user switch robot control between the command line, the joystick controller, or the mission manager, in addition to featuring a "disabled" mode which repeatedly outputs a stop command.

The thrust interface is the software component that operates the robot's thrusters. It subscribes to the mux's output, validates it, and forwards to a Raspberry Pi Pico through UART, which sends PWM signals to the thrusters.

For safety, each node in the control chain uses a heartbeat system: it must publish an empty "heartbeat" message to a topic that is listened to by the next node downstream. If the downstream node does not detect a heartbeat, it sends a stop command downstream from itself. This ensures that if any node crashes or becomes otherwise unresponsive, the robot comes to a safe stop. Heartbeats are shown in red on Fig. 7. The heartbeats are also monitored by Tidalwave, providing an easy overview of the status of each node.

The two sensor nodes each layer upon lower level drivers. The IMU is served by the manufacturer-provided InertialSense ROS2 node [9], which publishes orientation, angular rate, and acceleration to the network. The DVL communicates directly over UART, with the DVL node sending control signals (reset, calibration, etc.) and receiving dead-reckoning and velocity data.

Manipulator control is handled by a dedicated manipulator node. When triggered, this node sends a command via micro-ROS to an ESP-32 that directly controls the deployment servo. Keeping the manipulator on a separate microcontroller from the thrusters ensures that a fault in the dropper path cannot interfere with propulsion.

When performing an autonomous run, we cannot provide a final "go" signal through software (i.e. Tidalwave). The "go" switch serves this purpose. It is a physical switch monitored by code running on the ESP-32, which signals the current state to the mission manager. If the mission manager has already been sent a "ready" signal from Tidalwave, then the activation of the switch triggers the start of a run.

III. Testing Strategy

A. *Lessons from our Rookie Year (RoboSub 2025)*

The pervasive theme across the previous year's technical challenges was untested systems. A combination of electrical and software unknowns and fault unfamiliarity made troubleshooting extremely challenging, especially under the pressure of competition. One thing was abundantly clear: we needed to make real-condition testing the driver of our development process.

This year, we've implemented better testing procedures both in software and in our development cycle. We now conduct weekly pool tests with a rigorous procedure, described in Appendix D. Holding weekly pool tests creates a recurring artificial deadline that drives progress and ensures that our systems experience real conditions year-round, as opposed to the month leading up to competition. Our software now includes a hardware in the loop setup through MATLAB and multiple fail-safes within our software architecture. This renewed emphasis on real condition testing has rapidly increased our development speed and exposed system faults that otherwise went unnoticed.

B. *Controller Testing*

As described in Section II.E, controller testing was done in three distinct operating modes: fully-simulated, hardware-in-loop, and competition. This design is derived from system requirement SR-S- 4. This strategy also allows many issues to be identified and resolved in simulation so valuable pool test time is well spent.

The strategy for testing each new feature in the controller is as follows. First, the feature is written on a work-branch of the Controls Github repository. The new feature is rigorously tested in the fully-simulated version of the controller to make sure the code is logically correct and free from bugs. Once this round of testing is satisfied, the feature is merged into a pool-testing branch. At the next pool test, the feature is validated using the hardware-in-loop mode. The controller running in Matlab still at this stage is advantageous because it means much of the same Simulink infrastructure for plotting, tuning, and modifying used by the fully-simulated control mode is available for use. Finally, a feature that has passed both stages of validation can be moved to the main branch where it will be C-Code generated for use in the untethered competition mode.

C. Software Testing

Software undergoes a multi-stage testing pipeline to ensure stability, with a stable `main` branch that has been pool-tested and verified, and a `staging` branch for features that are hardware-tested but not pool-tested. Features are developed in work branches, and must pass all unit tests (automatically verified via GitHub Actions) before being merged. This pipeline ensures that any code that is on the `main` branch has been rigorously tested and will be stable and accurate. For more information, see Appendix G.

D. Vision Testing

A staged validation strategy was applied to the vision system. The team used simulation environments to rapidly tune parameters and generate synthetic datasets for training and validating object detection and keypoint localization. To reduce reliance on pool access, a handheld camera testing setup was developed for evaluating models on physical hardware and refining perception parameters. This workflow accelerated iteration speed and enabled continuous validation of the perception system before pool testing.

E. Component Level Testing

Each component of our electrical system was tested before being implemented. After each new addition, we performed a full test of all systems to determine if any new issues had been introduced. This included methodically and redundantly checking all electrical connections and data feeds under stressful conditions.

F. System Level Testing

Our team employs a test-early and often strategy by performing weekly pool-tests. Consistent weekly goals guided the team to prioritize the next viable product of the work, checking regularly that individual members' and subteams' work was compatible with the larger design. Routine pool testing often identified issues at the interface between subsystems that were difficult to identify in subsystem level testing. In addition to testing the robot, preparation for the system tests also revealed the need to design well-documented standard operating procedures (SOPs). These SOPs cover a range of critical topics including software configuration, battery and electrical safety and continuity, hull leak testing, and data logging. A full breakdown can be found in Appendix D.

G. Human Machine Interfaces

The introduction of rigorous testing meant that team members were more reliant on the feedback from our systems. The LED indicators (Section II.B.d), while effective at quickly communicating status information, are limited to that. To assist with high fidelity monitoring, we developed a graphical user interface (GUI).

The Tidalwave GUI (labelled as "Dashboard" in Fig. 7) provides the robot operator with the camera feed, IMU data, and system-state topics relevant to robot operation. It was built with QT for cross-platform compatibility, and embeds joystick inputs and mission file readings into the dashboard. A full breakdown can be found in Appendix L.

During manual control, this allows the operator to command the vehicle and observe its response in the same window. During automatic control, team members can monitor the robot for anomalous behavior and sensor data.

Acknowledgements

We would like to extend a huge thank you to our sponsors, including Citris and the Banatao Institute, WaterLinked, Onshape, InfoSec Decoded, the UC Davis Tech Foundry, the UC Davis Engineering Dean's Executive Committee, and our generous community. In addition, we would like to thank our parents, and the UC Davis Engineering Student Design Center for the tools and workspace.

We would like to extend a special thank you to our advisors Dr. Shima Nazari and Abhigyan Majumdar for their continued support and insight. We would also like to thank all of our highly dedicated team members! A full list of contributors can be found in Appendix P.

References

[1] [Online]. Available: https://robonation.org/app/uploads/sites/4/2023/06/2023-RoboSub_Team-Handbook_v2.0.pdf
[2] [Online]. Available: https://bluerobotics.com/learn/thruster-usage-guide
[3] [Online]. Available: https://formlabs.com/white-papers/3d-printing-watertight-enclosures-and-pressure-testing-results/
[4] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022, doi: 10.1126/scirobotics.abm6074.
[5] G. Jocher, J. Qiu, and A. Chaurasia, "Ultralytics YOLO." [Online]. Available: https://github.com/ultralytics/ultralytics
[6] NVIDIA Corporation, "Jetson Orin Nano Super Developer Kit." 2024.
[7] [Online]. Available: https://www.behaviortree.dev/docs/intro
[8] [Online]. Available: https://www.behaviortree.dev/groot
[9] [Online]. Available: https://docs.inertialsense.com/
[10] A. H. De Ruiter, C. Damaren, and J. R. Forbes, *Spacecraft dynamics and control: an introduction*. John Wiley & Sons, 2012.
[11] [Online]. Available: https://www.mathworks.com/help/control/ref/pidtuner-app.html
[12] [Online]. Available: https://www.mathworks.com/discovery/parameter-estimation.html
[13] [Online]. Available: https://waterlinked.com/dvl/dvl-a50
[14] [Online]. Available: https://www.onshape.com/en/resource-center/
[15] Blue Robotics, Inc., "Low-Light USB Camera (R1)." 2026.
[16] B. Sekachev *et al.*, "opencv/cvat: v1.1.0." [Online]. Available: https://doi.org/10.5281/zenodo.4009388

Appendix

A. *Task Breakdown*

We measured success rate as a team's ability to achieve the maximum points possible within a given task.

TABLE I
2023 Task Rankings

Name	Max Points	# Attempted	% Attempted
Gate: Maintain Fixed Heading	150	24	100.0%
Gate: Pass Through	100	24	100.0%
Gate: Style	800	18	75.0%
Start Dialing: any, Correct side	600	13	54.2%
Gate: Flip Coin	300	13	54.2%
Chevrons: Surface in area	1000	6	25.0%
Goa'uld (Torpedo): Any	500	4	16.7%
Location: Any bin/Correct bin	1000	3	12.5%
Time Bonus	100	2	8.3%
Inter-Vehicle Com	1000	2	8.3%
Random Pinger: Task 2	1500	2	8.3%
Random Pinger: Task1	500	2	8.3%
Goa'uld (Torpedo): Correct	500	2	8.3%
Location: Remove Lid	500	1	4.2%
Chevrons: Surface w/object	400	0	0.0%
Chevrons: Drop Object	200	0	0.0%
Chevrons: Object on table	500	0	0.0%
Chevrons: Correct Sequence	300	0	0.0%

For 2025, we also choose to evaluate whether teams who attempted a task managed to successful acquire the maximum amount of point.

TABLE II
2025 Task Rankings (with success rate)

Name of task	Max Point Value	# Attempted	% Attempted	# Sucessful	% Sucessful
Dry weight	126	34	100.0%	34	100.0%
Under weight requirement	(+) values	31	91.2%	31	91.2%
Gate: Pass through	100	21	61.8%	21	61.8%
Gate: Maintain control	150	20	58.8%	20	58.8%
Gate: Coin flip	300	15	44.1%	15	44.1%
Gate: Style (Yaw / Roll, Pitch)	1600	15	44.1%	9	26.5%
Channel: Any, Correct	1200	8	23.5%	4	11.8%
Channel: Correct Depth	600	8	23.5%	4	11.8%
BRUVS: Border, Wrong, Correct	1200	2	5.9%	2	5.9%
Tagging: Any	1200	2	5.9%	1	2.9%
Tagging: Correct Sequence	1600	0	0.0%	0	0.0%
Tagging: Far torpedos	600	2	5.9%	1	2.9%
Ocean cleanup: Surface in area	800	4	11.8%	4	11.8%
Ocean cleanup: Surface w/ object	1200	1	2.9%	1	2.9%
Ocean cleanup: Drop object	600	1	2.9%	1	2.9%
Ocean cleanup: Basket wrong, Correct	2100	1	2.9%	1	2.9%
Ocean cleanup: Object ± count, Correct	1000	1	2.9%	1	2.9%
Return Home	300	4	11.8%	4	11.8%
Random Pinger: First task	500	2	5.9%	2	5.9%
Random Pinger: Second task	1500	1	2.9%	1	2.9%
Inter-Vehicle Communication	1000	5	14.7%	0	0.0%
Finish Mission T.tt minutes	T.tt * 100	2	5.9%	0	0.0%

B. Design Requirements

TABLE III
Mission Requirements

ID	Requirement
MR-1	The vehicle shall be autonomous
MR-2	The vehicle shall be fully submerged during operation
MR-3	The vehicle shall have an easily accessible kill switch
MR-4	All thrusters shall have guards that shield the props
MR-5	The vehicle shall be usable in real research deployments with human control

TABLE IV
Mechanical & Electrical System Requirements

ID	Type	Description	Motivation
SR-ME-1	Safety	Manatee shall be safe to handle, both in and out of the water	A safe to handle vehicle will protect people handling it, as well as the vehicle itself.
SR-ME-2	Research	Manatee's research sensors shall be easy to decouple from the rest of the vehicle	Sensors used in research deployments should be easily mounted and removed between deployments
SR-ME-3	Reliability	All electrical and mechanical features shall be securely fixtured	Temporary fixturing techniques like tape and weak electrical connections frequently disconnect and compromise operation
SR-ME-4	Reliability	Manatee's electrical systems shall be validated for continuous operation	To avoid overheating, excessive power drain, and component damage during sustained operation

TABLE V
Software System Requirements

ID	Type	Description	Motivation
SR-S-1	Reliability	Manatee shall make autonomous decisions based off of context	A rigourous control system allows the robot to reliably navigate its surroundings
SR-S-2	Capability	Manatee shall understand its position with respect to the world frame	To provide the control system with context for its decisions
SR-S-3	Capability	Manatee shall have physical joystick control	To drive Manatee during research deployments and to expedite pool testing procedures
SR-S-4	Capability , Reliability	Manatee's software shall be verifiable without uploading to the robot	To allow code blocks to be tested and developed without putting Manatee out of commision.
SR-S-5	Capability	Manatee shall have an intuitive interface to communicate with users during wired testing	It is important that users can access core metrics such as battery voltage and camera feeds with minimal training

C. Bill of Materials

TABLE VI
Main Vehicle Bill of Materials (Total Cost: $8,597.35)

Item	Cost/Unit	Unit Type	Qty.	Total Cost	Description	Supplier	Design Category
Basic ESC	$40.00	Each	8	$320.00	ESCs that control Manatee	BlueRobotics	Boards/ Sensors
Raspberry Pi	$110.00	Each	1	$110.00	Manatee's primary logic board	Amazon	Boards/ Sensors
ESP-32 Devkit	$10.00	Each	1	$10.00	Peripheral microcontroller for Manatee	Digikey	Boards/ Sensors
Waterlinked A50 DVL	$4,069.00	Each	1	$4,069.00	DVL for vehicle navigation	Waterlinked	Boards/ Sensors
Inertialsense IK-1-IMX-5	$350.00	Each	1	$350.00	IMU for vehicle navigation	Inertialsense	Boards/ Sensors
Custom PCB	$4.27	Each	5	$21.35	Custom thruster control board	JLC PCB	Boards/ Sensors
Buck Converter	$5.00	Each	1	$5.00	High-to-low power converter for Manatee	Aliexpress	Electrical
Low Light USB Camera	$120.00	Each	3	$360.00	USB cameras for vision	BlueRobotics	External Cameras
Resin, SLA 3D Printed Parts	$0.08	Millileter Volume	350	$28.00	SLA resin used for external camera enclosures	Formlabs	External Cameras
50mm x 3.5mm O-rings	$20.00	Set	1	$20.00	O-rings for external camera enclosures	McMaster Carr	External Cameras
Wetlink Penetrator	$13.00	Each	21	$273.00	Penetrators that feed into the main enclosure	BlueRobotics	General Electrical
10000mAh 4S 15C LiPo Pack	$93.00	Each	1	$93.00	Manatee's onboard battery	Hobbyking	General Electrical
Fuse Block	$42.00	Each	1	$42.00	Fuse block with integrated screw terminals for wire routing	Amazon	General Electrical
ATC Fuse Set	$12.50	Set	1	$12.50	ATC fuses for electrical safety	Amazon	General Electrical
Kill Switch Penetrator	$28.00	Each	1	$28.00	Kill switch integrated into a penetrator port	BlueRobotics	General Electrical
Misc. Electrical Components (Aggregate)	$35.00	Aggregate	1	$35.00	Aggregate of small electrical components in Manatee	Multiple	General Electrical
Filament, FDM 3D Printed Parts	$23.00	Kilogram	4	$92.00	PETG Filament for structural components	Bambulab	General Mechanical
Stainless Steel Mounting Hardware (Aggregate)	$0.25	Per Piece	100	$25.00	Estimated aggregate price of all nuts, bolts, and washers on Manatee	McMaster Carr	General Mechanical
6" Acrylic Tube	$150.00	Each	1	$150.00	Tube for main enclosure	Plasticraft	Hull
O-Ring Flange	$90.00	Each	2	$180.00	Flanges for main enclosure	BlueRobotics	Hull
Acrylic End Cap	$53.00	Each	1	$53.00	Clear end cap for front of enclosure	BlueRobotics	Hull
Aluminum End Cap - 15 x M10 Hole	$58.00	Each	1	$58.00	Drilled end cap for cable penetrators to feed into the main enclosure	BlueRobotics	Hull
Routed Aluminum Base Plate	$340.00	Each	1	$340.00	CNC-routed 1/4" base plate that Manatee is built around	???	Hull
T-200 Thruster	$230.00	Each	8	$1,840.00	Thrusters on Manatee	BlueRobotics	Hull
Magnetic Reed Switch	$8.00	Each	1	$8.00	Reed switch for kill switch	Amazon	Kill Switch
Waterproof Magnets	$24.50	Set	1	$24.50	Magnets for kill switch	totalElement	Kill Switch
Waterproof Servo	$50.00	Each	1	$50.00	Servo for dropper mechanism	SAVOX	Manipulation

TABLE VII
RESEARCH MODULE BILL OF MATERIALS (TOTAL COST: $306.50)

Item	Cost/Unit	Unit Type	Qty.	Total Cost	Description	Supplier	Category
Arduino Uno R4	$27.00	Each	1	$27.00	Microcontroller for research module	Amazon	Electronics/ Sensors
pH meter	$25.00	Each	1	$25.00	pH meter for research module	Atlas Scientific	Electronics/ Sensors
8GB MicroSD Card	$10.00	Set	1	$10.00	SD cards on research module	Amazon	Electronics/ Sensors
SD Card Module	$7.50	Each	1	$7.50	SD card module for data logging on research module	Amazon	Electronics/ Sensors
Real Time Clock	$7.00	Each	1	$7.00	RTC for research module	Adafruit	Electronics/ Sensors
O-Ring	$5.50	Set	1	$5.50	O-rings to seal research module	McMaster Carr	Hardware
6" Diameter x 1 1/2" 6061 Aluminum Disk Stock	$69.00	Each	2	$138.00	Disk stock to be machined down for flanges of research module	McMaster Carr	Hardware
Underwater Headlight	$30.00	Each	1	$30.00	Headlight for visibility in murky conditions	Amazon	Hardware
Wire Quick Connectors	$10.00	Set	1	$10.00	Electrical connectors for research module	Amazon	Hardware
Filament, FDM 3D Printed Parts	$23.00	Kilogram	0.5	$11.50	PETG 3D Printed parts in research module	Bambulab	Parts Aggregate
Electrical Wires and Addons (Aggregate)	$10.00	Aggregate	1	$10.00	Aggregate of small electrical components in the research module	Multiple	Parts Aggregate
Stainless Steel Mounting Hardware (Aggregate)	$0.25	Per Piece	20	$5.00	Stainless steel screws for mounting hardware	McMaster Carr	Parts Aggregate
6" Acrylic Tube	$20.00	Each	1	$20.00	Acrylic tube for research module enclosure	Plasticraft	Hardware

TABLE VIII
THRUSTER TEST FIXTURE BILL OF MATERIALS (TOTAL COST: $491.60)

Item	Cost/Unit	Unit Type	Qty.	Total Cost	Description	Supplier	Category
T-200 Thruster	$230.00	Each	1	$230.00	Thruster used on test rig	BlueRobotics	Electronics/ Sensors
Basic ESC	$40.00	Each	1	$40.00	ESC to control Thruster	BlueRobotics	Electronics/ Sensors
Load Cell	$11.50	Each	2	$23.00	Load cells to measure thrust	Digikey	Electronics/ Sensors
Amplifier	$5.00	Each	2	$10.00	Amplifiers for load cell signal	Amazon	Electronics/ Sensors
Raspberry Pi Pico	$12.00	Each	1	$12.00	Microcontroller for test rig	Amazon	Electronics/ Sensors
10000mAh 4S 15C LiPo Pack	$93.00	Each	1	$93.00	Battery to power test rig	Hobbyking	Electronics/ Sensors
3/4" Schedule 40 PVC Pipe	$0.50	Foot	20	$10.00	PVC pipes for structure	Home Depot	Hardware
Fishing Line	$7.00	Each	1	$7.00	Low-stretch line to transmit tensile loads	Amazon	Hardware
PVC Elbows, Assortment	$0.70	Each	22	$15.40	PVC elbows for structure	Multiple	Hardware
Plastic Bearings	$1.65	Each	8	$13.20	Plastic ball bearings for slide travel	Amazon	Hardware
Filament, FDM 3D Printed Parts	$23.00	Kilogram	1	$23.00	PETG 3D printed parts	Bambulab	Parts Aggregate
Electrical Wires and Addons (Aggregate)	$10.00	Aggregate	1	$10.00	Aggregate of small electrical components and wiring	Multiple	Parts Aggregate
Stainless Steel Mounting Hardware (Aggregate)	$0.25	Per Piece	20	$5.00	Stainless steel screws for mounting hardware	McMaster Carr	Parts Aggregate

D. Pool Test Procedure

This document outlines the standard operating procedures, safety protocols, and pre-submersion verification checklists for the AUV system. It is designed to ensure system integrity, operational safety, and clear team communication during developmental and pool testing phases.

---

a) *Pre-Test & Prep (At the Shop):*

Pre-Test Meeting: Before leaving the shop, verify the robot's configuration:

- **Electrical Configuration:** Is the robot in a fully configured state? Is everything plugged in?
- **Pressure Readiness:** Is the robot able to maintain pressure?
- **Inventory Check:** Do we have everything? (Cross-reference the packing lists below based on the specific testing operations).
  - Pool Test Packing List
  - Tool / Competition Packing List

Before Sealing the Tube: Visually inspect the robot interior:

- No loose wires
- No bridged connections
- No signs of water damage
- Raspberry Pi is turned on

b) *Tube Sealing & Pressure Testing:*

After Sealing the Tube:

- **O-Ring Check:** Does the seal around the o-rings look good?
- **Post-Seal Wire Check (Second Visual Inspect):**
  - Do any of the wires look bent?
  - Did any of the wires get pulled out during sealing?
- **Pi Status:** Is the Raspberry Pi green?

Pressure Hold Test:

- **Hold Test:** Confirm pressure holds at **15 mmHg for 10 minutes**.

Troubleshooting Pressure Failure (If it does not hold):

- Visually inspect o-rings for proper seal.
- Ensure the tube is completely closed.
- Inspect the pressurizer to make sure it has a good seal.
- Tug cables running into the Wetlink penetrators to check for leaks.

c) *Physical & Electrical Readiness:*

The Wiggle Test:

- Give all components a gentle wiggle to see if they move or come loose (*be EXTRA careful with the wires*).

Troubleshooting Loose Components:

- **Serious Issue?:** Alert Hull Lead (Luci) if it's serious (use best judgment).
- **Loose Bolt?:** Tighten it back into place.
- **Loose Wetlink?:** Take it apart and put it back together by reseating the wires.
- **Loose Wire?:** Plug it back in or tighten the connector.

Battery & Thruster Testing:

- **Operational Voltage:** Check that the battery is at operational voltage (**16.8V**) using the battery charger or a multimeter.
  - *If not at correct voltage:* Plug in the battery charger and charge the battery until it reaches 16.8V.
- **Thruster Spin Test:** Turn on the thrusters one-by-one to make sure they behave as expected (*on dry land only!*).

d) *Final Ready-for-Water Verification:* Perform these final safety checks right before putting the robot in the water:

Electrical:
- Battery is plugged in.
- Components light up and/or make startup sounds:
  - ESCs
  - Raspberry Pi
  - Sensors
  - Cameras
- Thruster wires are completely out of the way of the thruster blades.
- Ethernet cable is organized and will not get caught in moving parts.
- Kill switch is off.

Mechanical:
- Tube clasps are secured down (tube is fully secured).
- Pressure plug is all the way in.
- Bumpers and thruster guards are secured.

e) *In-Pool Operational Protocol:* To maintain safety and clear communication in a noisy environment, use this structured call-and-response protocol for every test run.

Team Roles:
- **Operator:** Controls the software, triggers the code execution, and directs the timeline.
- **Diver:** Serves as the physical safety monitor in the water, handles the physical robot, and commands emergency stops.
- **Timekeeper:** Tracks elapsed seconds/minutes and monitors actual execution times against expected duration.

Step-by-Step Test Run Script:

Step	Initiator	Action / Verbal Command	Purpose
1	**Operator**	Shouts: **"OP READY!"**	Signals that software is fully configured and primed to run.
2	**Diver**	Shouts: **"DIVE READY!"**	Confirms that the diver is prepared for robot activation.
3	**Operator**	Shouts: **"Running [Action] for [X] seconds!"** *(e.g., "Running thrusters forward for 10 seconds!")*	Clearly establishes the test scope and expected duration for the deck and the diver.
4	**Diver**	Shouts: **"Copy: [Action] for [X] seconds!"**	Verbally repeats and confirms understanding of the test window.
5	**Operator**	Shouts: **"Timekeeper, mark!"** and executes the code.	Triggers the Timekeeper to start the stopwatch.
6	**Timekeeper**	Tracks time. Announces milestones if necessary, comparing actual duration against the expected test limit.	Keeps the team aware of runtime progress.

Kill Robot Protocol: If the robot exceeds its designated test duration, behaves erratically, or poses a hazard, execute the following emergency workflow:

1) **Diver** must shout: **"KILL!"**
2) **Operator** immediately triggers a **Software Kill** (if there is no immediate danger to personnel or hardware).
3) **Diver** executes a physical **Hardware Kill** if the robot is at risk of harming itself or others.
4) **Diver** brings the robot back to the pool edge.
5) **Operator** unplugs the tether/ethernet cable immediately upon retrieval.

f) *Post-Test Clean Up & Storage:* When testing is complete, follow these cleanup procedures:

Electrical Cleanup:

- Plug the battery into the charger and set to **"Discharge"** to bring the battery down to storage voltage (**14.8V**).
- Organize all wires so they are neatly plugged in and tucked out of the way of the tube closing.

Mechanical & Tool Cleanup:

- Close the main tube.
- Ensure the pressure plug is in.
- Return all tools to the toolbox.
  - Verify that all tool sets are complete and no items are missing.

E. MATLAB Implementation

This section elaborates on the discussion of the Matlab and Simulink model of Manatee from Section II.E. We will

1) provide a tour of the Matlab codebase, highlighting major features,
2) zoom in on the Low Level Controller to discuss design, tuning, and performance,
3) explain the workflow for Unreal Engine cosimulation, preliminary results, and future plans,
4) show the procedure for communicating from Matlab to the ROS2 network for Hardware-In-Loop ,operation, and
5) and show the procedure for C++ code generation from a Simulink model.

Finally, we include a discussion on the estimation of unknown vehicle parameters from pool testing data, specifically the design of a thruster test-fixture to estimate the time-response of the T200 thrusters used on the robot.

a) *Overview:* Fig. 8 shows a recent version of the Simulink model used for the fully-simulated operation. We will follow the flow of information through this model to understand its structure before expanding on specific components.

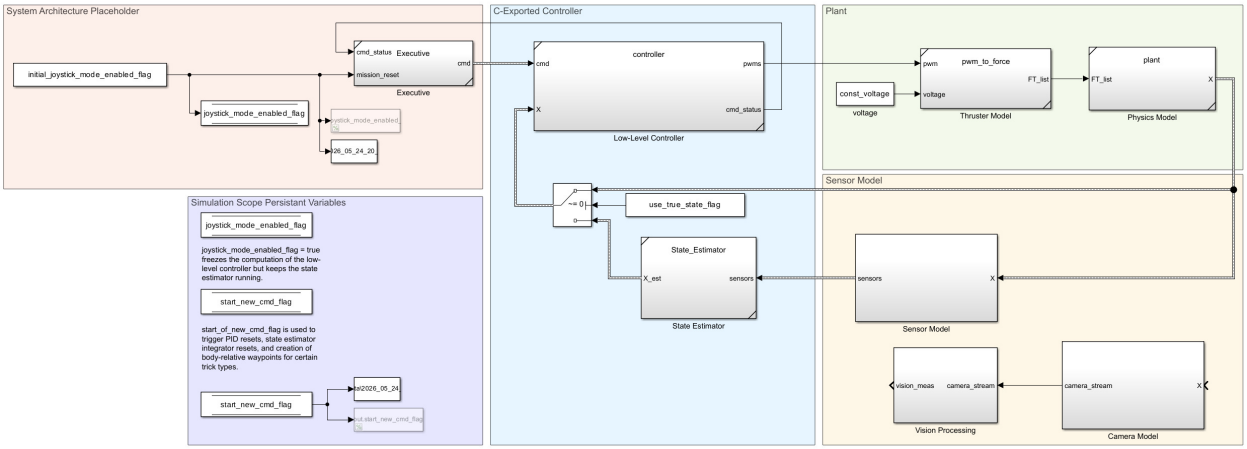

Fig. 8. Simulink Model

The Simulink model is run using an initialization script called the *init file*. This script is responsible for loading the necessary vehicle parameters into the model workspace, including mass and volume information for buoyancy calculations, thruster and sensor positions and angles, drag coefficients, and so on. The init file is also responsible for setting initial conditions of the vehicle in the scene and toggling various "flag" settings. These flags modify simulation behavior in useful ways for debugging, for example turning off gravity, overwriting measured state errors with "injected" values, and determine whether the controller runs using the true simulated vehicle state or a state estimate. This script-based approach lets us rapidly sweep settings and interrogate specific simulation behaviors for more efficient debugging.

Data is saved from the simulation using both Simulink To-Workspace and To-File blocks, whichever is desired for a given test. The data saving method can be programmatically selected from the init file. These blocks save all the data at a user-controllable data rate to be post-processed after the simulation is run. The codebase includes custom plotting code quickly view data, compare different signals in a single simulation run, and compare signals across different simulations.

When the init file is run, a mission file is loaded for the Simulation to use. In the Matlab model, this init is simply a list of commands to be executed in series. This is sufficient to validate that the controller can execute a single command and switch between any two commands. Validation of the full branching decision tree is left to the Software team. The minimal copy of the executive controller in the Matlab model accepts the mission file and sends one command at a time to the Low Level Controller. The executive determines when to advance to the next command based on the command status returned to it from the Low Level Controller as well as a allowable command duration specified in the mission file.

The Low Level Controller receives the state estimate and the command and determines the thruster commands needed to execute that command. More details will be provided for the Low Level Controller the following section.

The thruster commands are fed into the plant model, which uses a custom implementation six degree-of-freedom dynamics and quaternion kinematics to simulate the motion of Manatee under the influence of drag, buoyancy, gravity, and the thrusters. This simulation is updated at 1000 Hz, although the controller itself typically operates at a 100 Hz update rate. This plant model calculates the ground truth location of the vehicle for the fully-simulated operating mode. This section is removed and replaced with real physics for the hardware-in-loop and competition operating modes.

For the fully simulated mode, we also need to fake noisy sensor and camera data for controller validation. This is accomplished in the sensor model, where the Intertiasense IMU and Waterlinked DVL are simulated. The IMU simulation uses Matlab's built in IMU sensor model, while the DVL model is a simple custom model that adds white noise and a slow-drifting bias to the true states. Camera simulation is a much more involved task, which will be expanded upon in a subsequent section.

The last major feature is the state estimation portion. The Waterlinked DVL has built-in Kalman filter that provides sufficiently high quality attitude and dead-reckoning position estimates, but at too-low a data rate (5Hz) to be used directly in the controller. Our state estimation system fixes this by integrating between the DVL reports using the IMU to provide the controller with state estimates at 100 Hz. This approach has shown success in initial testing, but is expected to give poor performance for longer tests due to accumulating dead-reckoning error and IMU drift, neither of which the controller currently accounts for. The long term plan is to use a simultaneous localization and mapping (SLAM) algorithm enabled by the camera system to periodically "reset" the dead reckoning error through repeated observation of fixed object, although this system is still early in development. In the mean time, we hope to quantify the circular error probable for Manatee's position navigating via dead reckoning for various mission lengths.

b) *Low Level Controller:* The operation of the Low Level Controller is broken into three stages: Command Executer, Guidance Law, and the PID Controller.

The Command Executer inputs the command from the mission file. This algorithm follows a separate execution path for each command type, ultimately outputting a attitude and inertial position waypoint for the next stage of the Low Level Controller. For some command types, such as *drive to world waypoint*, the waypoint is included in the command parameters. For other commands, like *position relative to object*, the Command Executer must calculate the appropriate world waypoint based on the sensor measurements and the current state estimate. The Command Executer also evaluates the success criteria for each command, which typically consists updating a timer when the state estimate is within a specified tolerance of the waypoint.

The Guidance Law is the next stage of the Low Level Controller. This function accepts the waypoint from the Command Executer and compares it to the current state estimate. It then breaks the maneuver into distinct *turn*, *drive*, and *settle* phases. During the turning phase, the Guidance Law generates an intermediate waypoint that causes the robot to rotate to face in the direction of the world waypoint without trying to correct position. Next, the robot drives forward until the position error is within a tolerance. Finally, in the settling phase the controller corrects both position and attitude to meet the target from Command Executer. This strategy of distinct turning and driving modes makes the robot more stable for normal operation by avoiding sideways motion, which typically causes the robot to roll. It also makes it easier to modify certain phases of the maneuver, for example by adding an artificial roll error specifically in the driving phase of the maneuver to cause the robot to do a barrel roll.

The last stage of the Low Level Controller is the PID. This controller accepts the waypoint (or intermediate waypoint in the case of turning or driving mode) and determines the motor commands to reach it. The PID is a cascaded architecture with two parallel branches. The first branch contains a PID controller that maps position error to a velocity target in series with a second PID mapping velocity error to a force command. The second branch maps quaternion error to a target angular velocity, then angular velocity error to a torque command. Finally, the force and torque commands are mapped to the individual thruster forces for all eight thrusters. This architecture makes the system easier to tune, because the velocity and position control can be tuned independently. It also allows for separate output and integral saturation points for each state of the controller.

PWM values for the motors are calculated by interpolating between values on a 3D lookup table that calculates the pwm based on battery voltage and the force command.

c) *Unreal Engine Cosimulation:* The objective of the Unreal Engine Cosimulation is to use Unreal Engine's shader system and lighting engine to simulate camera sensor input. The simulated camera data can then be used to train vision models and validate controller vision feedback through a Simulink model running an unreal executable. In this Simulink subsystem, blocks from the Simulink 3D Animation toolbox are used to set the position of an Unreal Engine actor in the 3D scene. Attached to the actor in Unreal Engine are camera models with camera intrinsics, similar to the real cameras. Other actors exist to simulate course props such as the gate and slalom courses that can be arranged in the sim environment directly from MATLAB. By publishing the robot's coordinates from a physics simulation in MATLAB, camera data can be simulated in an Unreal underwater lighting environment that has underwater caustics, chromatic aberration, radial distortion, water surface reflections, and depth based color absorption. Camera images from the unreal environment can then be tested for feature extraction and stereo vision pixel matching algorithms.

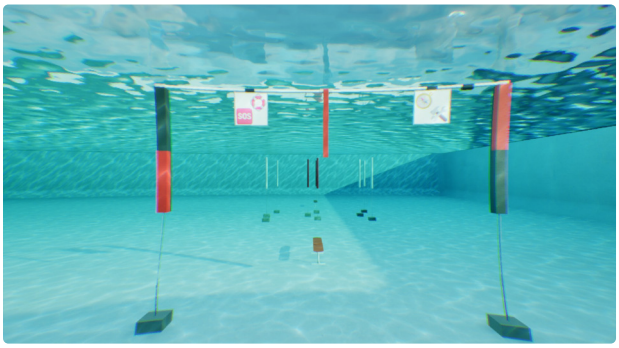

Fig. 9. Camera Output from Unreal Simulation

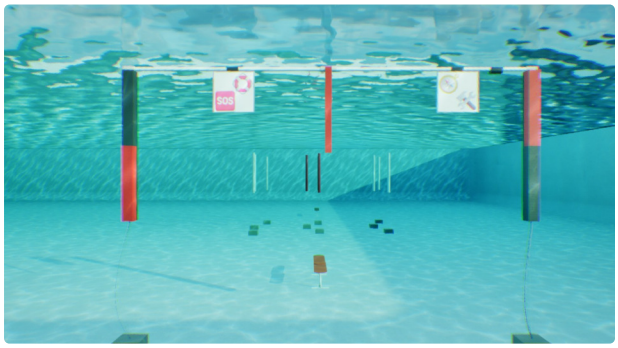

Fig. 10. Rectified (Stereo vision: aligned planes/undistorted) Image of Gate

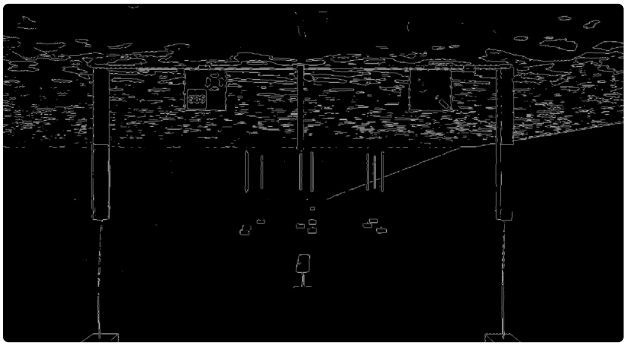

Fig. 11. Edge Detection using Sobel Filter and Local Maximums

Fig. 12. Pixel Disparity map between left right camera views.

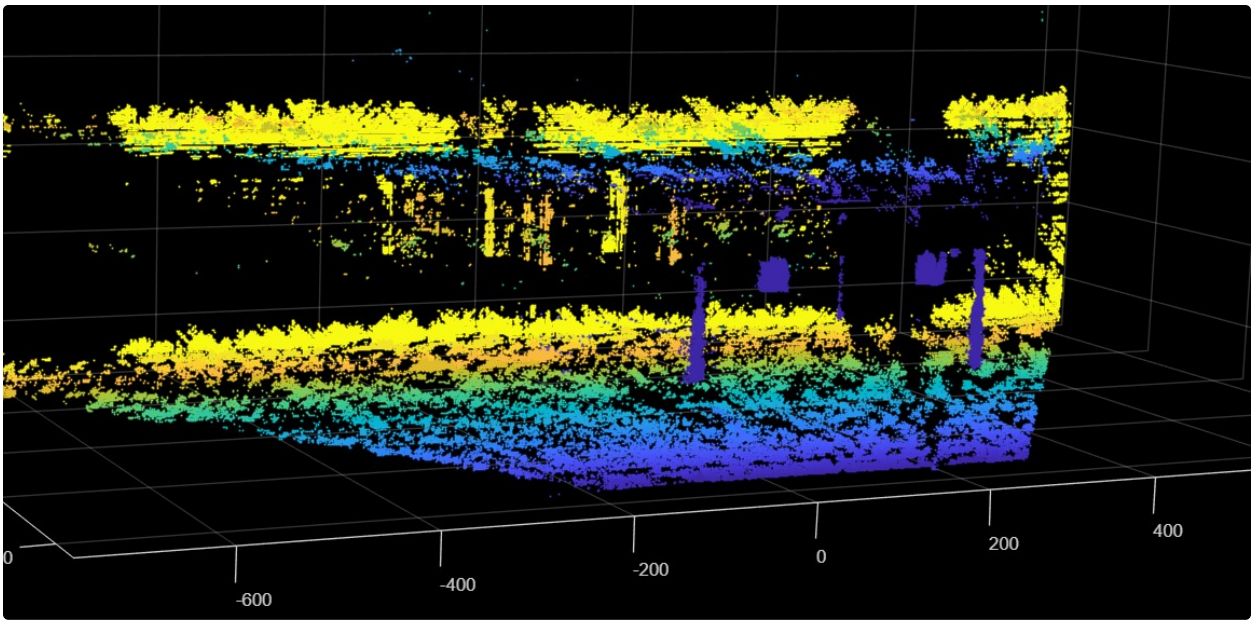

Fig. 13. Reconstructed 3d Point Cloud of Gate & Pool Derived from Disparity Map

d) *ROS2 Integration:* To enable our Hardware-In-The-Loop testing this year, we utilized Simulink's ROS2 Toolbox to provide inputs and outputs.

Inputs to the HIL Simulink models were given by ROS2 Subscriber blocks and outputs were ROS2 Publisher blocks. Both kinds of these blocks were kept in for code generation.

Our first integration of the ROS2 structure was through the development of a joystick enabled mode as shown Fig. 14.

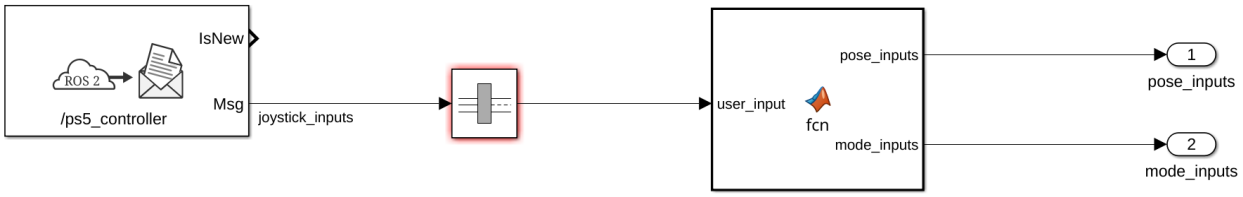

Fig. 14. Simulink Implementation of the ROS2 subscriber for joystick inputs.

Simulink's implementation of ROS2 Publisher blocks are not data type or name agnostic. To maneuver around this restriction we named signal traces to match the ROS2 publisher blocks desired name. Conversion of data types were also matched with the expected data type from ROS2 publisher blocks.

An example of the above is provided below in Fig. 15.

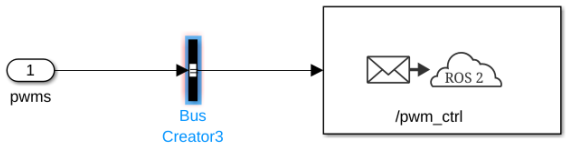

Fig. 15. Hardware settings of the ROS2 codegen.

e) *Code Generation:* MATLAB Simulink enables the code generation of our various subsystems after HIL testing. Code Generation follows a variety of hardware specific settings as shown in Fig. 16.

While Simulink codegen offers the ability to rapidly test, develop, and then deploy control software from a single tool, there are a variety of unknowns that have tripped us up in deployment.

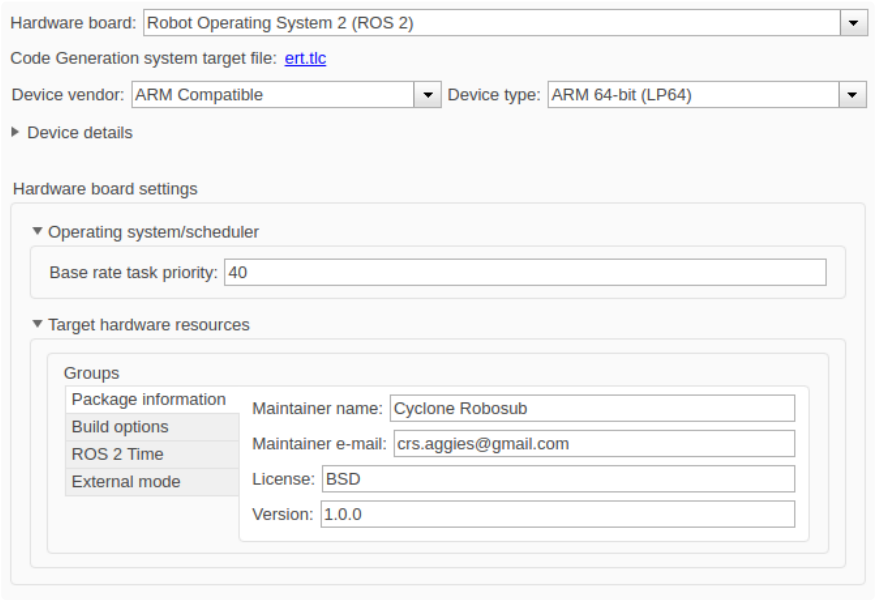

Fig. 16. Hardware settings of the ROS2 Code Generation.

An issue worth remarking is the mismatch between simulated control iteration frequency and the compiled iteration frequency at deployment. To mitigate this issue a sleep time was added to match the simulated frequency. However, further investigation into the code gen settings in Fig. 16 might give us a proper solution without having to force the the frequencies to match.

f) *Parameter Estimation:*

Parameter estimation of coefficients such as drag and wrench matrices was performed in order improve the fidelity of Manatee's dynamics simulation. Before performing parameter estimation, command data from pool testing was thresholded in order to isolate specific maneuvers on which to perform analysis. From there, Simulink Design Optimization experiment and simulator objects were created in order to correspond collected IMU and DVL data with signals in the dynamics Simulink model. Simulink Design Optimization parameter estimation tools were then implemented to tune the selected parameter to collected data using least squares method.

F. Machine Learning Vision Model

The YOLO-keypoint model was trained using transfer learning on a combination of real pool footage and synthetic images, with datasets split into training and validation sets to monitor overfitting. The model detects mission objects and localizes keypoints such as gate corners and bin features. Trained models were exported to ONNX and optimized with TensorRT for real-time inference on the Jetson Orin Nano. A handheld stereo validation platform was developed to accelerate iteration and enable pool-independent testing, supporting synchronized stereo image capture, TCP streaming, and adjustable camera baseline for rapid calibration and parameter refinement.

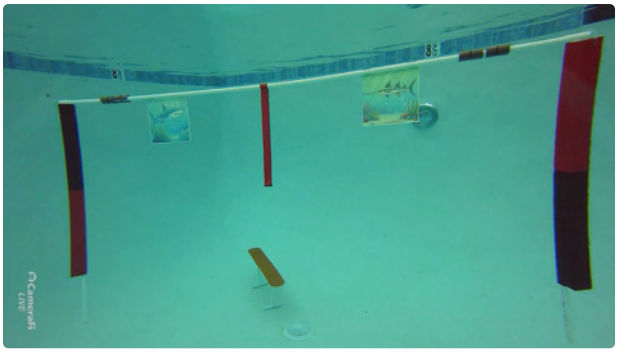

Fig. 17. AUV Testing

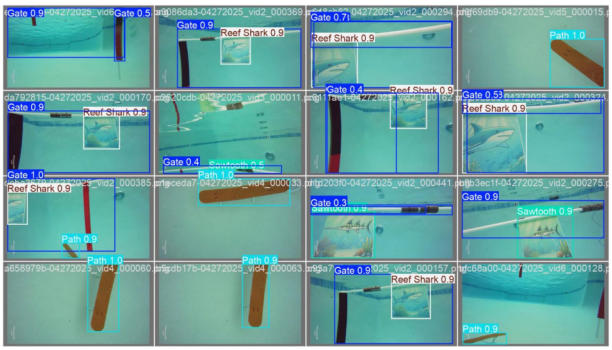

Fig. 18. Vision Model Testing Dataset Sample

G. Software Unit Testing and Stability Procedures

To ensure reliability, code is assumed to be nonfunctional until testing is complete and the feature is merged into a stable branch. The software subteam maintain two primary stable branches: `main` and `staging`. Features on the `staging` branch have been dry-tested, pass all unit tests, and include relevant documentation. Features in the `main` branch have additionally passed a pool test. When a new feature is needed, we create a GitHub issue with specifications, desired input and output, and relevant resources.

Code is first developed in feature-specific work branches, generally branched from the `staging` branch, with unit tests developed alongside the feature to the specifications outlined in the GitHub issue. Once the feature is complete and passes all of its unit tests, a pull request is created into the `staging` branch. GitHub Actions will automatically run the entire project's unit tests: if any fail, the pull request will be denied.

Before the pull request into `staging` will be merged, the feature undergoes a code review from a leadership member and a dry test on the robot hardware. If these are successful, the pull request is merged and the feature is tested in the water at the next weekly pool test. If this also succeeds with no issues, then the `staging` branch is merged into `main`.

We use the GTest framework for our C++ unit tests and pytest for our Python unit tests. Every function is rigorously tested to ensure that it behaves correctly under all inputs, including edge cases. Our automated tests publish to or subscribe to relevant relevant topics, to simulate node input and validate node output respectively. For nodes that communicate over serail (like the Pico and DVL), we use mock file descriptors rather than real serial port file descriptors. These mock file descriptors are UNIX pipes that allow us to provide input and validate output in a manner that is transparent to the tested node.

H. Thruster Test Fixture

Currently, little information is publicly available on the transient response of Blue Robotics' T200 thruster to PWM value inputs, so previous simulation models of Manatee relied on lookup tables of PWM values to steady state thrust values. The team also designed custom thruster guards, pictured in Fig. 30, and wanted to quantify its thrust effects. Thus, to refine controller development, development a submodel of thruster to increase Simulink dynamics model fidelity was undertaken.

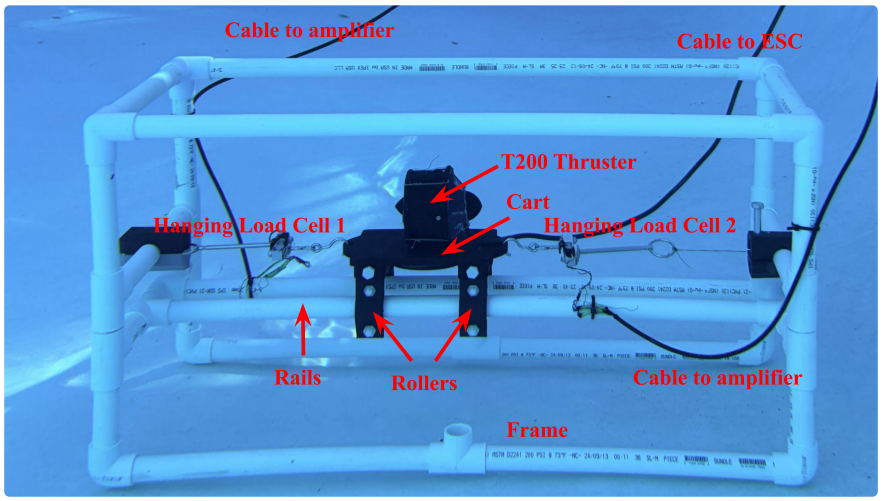

Fig. 19. Thruster Modeling Test Fixture

In order to gather the data necessary to create a submodel, design criteria for a test fixture was developed which included isolating axial thrust and minimizing wake interference. To isolate the axial force, a test fixture that constrained all measurements to forces along the direction along the rotor axis. Thus, a low-friction rail system was implemented that constrained the motion of the thruster along a single axis. This rail system consisted of two rails on which a small cart was mounted. The cart included mounting for a thruster and rollers (consisting of low friction bearings in contact with the rail's surfaces) to constrain the motion of the cart along the direction of the rail. To minimize wake interference, a test fixture that avoided free surfaces and distance any sensing modules from the thruster was desired. These decisions influenced the choice to construct a structure that could be placed at the bottom of a pool and the choice to use hanging load cells as a sensing unit. In development, the limitations of the sensing unit to respond to compressive forces were recognized, influencing the decision to implement two hanging load cells that held the cart in tension to ensure that thrust would be transduced regardless of thrust direction.

The electrical system of the test fixture was divided between data acquisition and sending PWM signals to the thrusters. The data acquisition system consisted of two hanging load cells connected to amplifier boards that also had analog digital conversion capabilities. The system to send signals consisted of Blue Robotics's basic ESC 500 and a T200 thruster. In order to interface with both the thruster and sensor units, a Pi Pico was selected.

Testing software was divided into two phases, one focused on planning, and the other focused on execution of tests. On the planning side, square, sine, and ramp PWM functions csv files were generated in MATLAB. On the testing side, Thonny IDE was used to read these csv files to have the Pi Pico supply PWM inputs to the ESCs. While running through a PWM function file, the Pi Pico also received a voltage output from each amplifier, writing this value into a results csv file containing time stamps, both output voltages, and a PWM values.

Preliminary results of testing included the observed difference in thrust resulting from the implementation of thruster guards. Future work to implement a thruster model into the larger Manatee model is still in process. Currently, this is taking place with MATLAB's System Identification tool box and by tuning a a first principles model both using the data collected with the test fixture.

I. Electrical System

Our power distribution system is split into two distinct pathways: high power (16V) and low power (5V). This separation helps isolate sensitive components from high-current loads, improving system safety, organization, and troubleshooting. For full overview refer to the electrical diagram below.

A battery is used to supply 16V to the High Current Board (HCB), which is protected by our Battery Management System (BMS) (discussed below). The HCB directly powers the Electronic Speed Controllers (ESCs), thrusters, and the Doppler Velocity Log (DVL) breakout. The HCB feeds into a buck converter which steps the voltage down from 16V to 5V. The lower voltage allows the Low Current Board (LCB) to integrate with our low power components that feed into our control system.

The control system—"brain" of our robot—includes a Raspberry Pi 5, Jetson Nano, Raspberry Pi Pico, and ESP-32. Each of these components handles a critical piece of functionality for our Underwater Vehicle:

- The Raspberry Pi 5 manages system-level integration and sensor fusion (e.g., DVL data).
- The Raspberry Pico generates Pulse-Width Modulation (PWM) signals to drive the thrusters and fires the software kill switch.
- The ESP-32 governs servo control for the manipulation subsystem, handles the LED indicators, and go switch.
- The Jetson Nano handles computer vision data from the cameras as well as Inertial Measurement Unit (IMU) data, informing decisions made by the control system.

One of the key features of our system is the kill switch, which safely cuts power to the high-power lines, such as the thrusters, while maintaining power to the Brains. This enables safer debugging and rebooting procedures without a full power-down of the system.

a) *Battery Management System:* Given the cost and safety concerns with lithium-ion batteries, one of our primary objectives this year was designing a battery management system. We designed a custom battery monitoring system to monitor and protect a 4-cell LiPo battery pack. Built around the MAX17320 IC from Analog Devices, this system track per-cell voltage, current draw, and temperature via thermistors and communicates to the rest of the robot over I2C. The system provides both soft warnings (LEDs, messages for over/under voltage, over/under current, thermal events, and low power states) and hard cutoffs through charge and discharge MOSFETs that the chip can toggle in response to detected faults.

On the hardware side, the board is designed to handle the battery's maximum current rating (which leaves a significant margin of safety for our actual system's current draw), using a parallel array of low-resistance SMD shunt resistors and 2oz copper pours to manage heat dissipation. Cell balancing is handled passively through the MAX17320′s built-in balancing circuitry with external resistors to limit current.

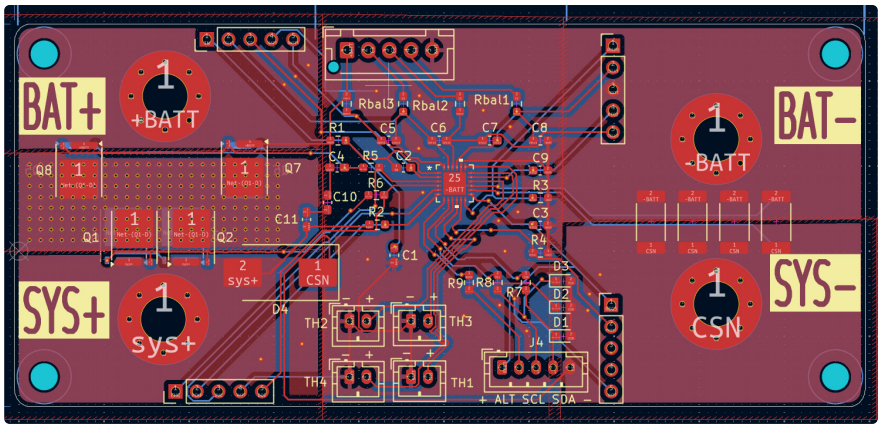

Fig. 20. BMS Layout

b) *Thruster Control Board:* Initially, our thruster PWM output signals and kill switch input signals, described in Section II.B.c, were run through a Raspberry Pi Pico mounted to a protoboard. This protoboard broke out the GPIO pins to screw terminal connectors that would then connect to our thruster ESCs and kill switch. However, this system had insecure connection points, was prone to creating shorts, and made it inconvenient to debug the Pico.

The Thruster Control Board is an improvement upon our initial protoboard design with a custom-designed PCB. Not only does it reduce the overall size profile with more efficient routing and a ground plane, but it replaces the screw terminals with labeled JST-PH connectors for the ESCs and kill switches. These latched connectors simplify wiring and prevent incorrect connections. Each PWM and kill switch connector is also paired with an LED to visually indicate PWM and kill switch status. Each PWM and GPIO pin is also broken out with 2.54mm headers, which can be monitored with a logic analyzer for more advanced debugging. The final improvement made was the addition of an RC low-pass filter for the kill switches on the board. This works in tandem with software debouncing to prevent false switch triggers by debouncing the switches and mitigating noise.

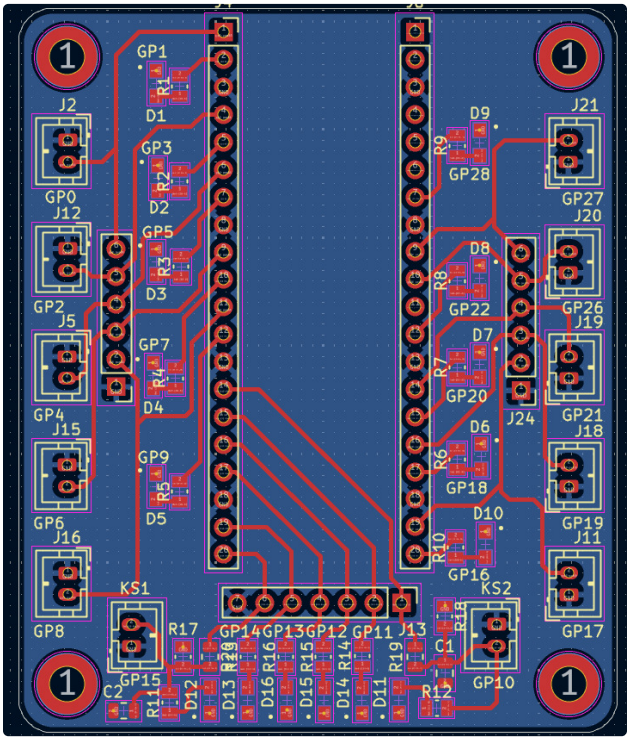

Fig. 21. Thruster Control Board Layout

J. Thermal Testing

After many of our pool tests, we noticed elevated surface temperatures of our components, and ambient temperatures in our tube, causing concern that our components may be approaching their throttling or maximum operating temperatures. In order to determine the need for a cooling system, we assessed the core and surface temperatures of the Raspberry Pi 5, Blue Robotics electronic speed controllers, and the 16V to 5V buck converter—components that should have the highest temperatures as they are either our main processors or have a high current draw running through them. To collect this data, we averaged the core temperatures transmitted to the computer and attached temperature probes to the surface of the components, sending that data to a Raspberry Pi Pico.

For the Raspberry Pi, the core temperatures typically stayed between 60ºC and 65ºC, far under its documented throttling temperature of 80ºC. For the Blue Robotics electronic speed controllers, we have already attached them to a copper heat bank, and they have not exceeded a surface temperature of 34ºC, far under its documented operating temperature of 85ºC. Lastly, the buck converter also manages itself well thermally, as we only measured an average surface temperature at 30ºC while its documented maximum operating temperature is 75ºC.

From our experimental data, we see that all of our components perform well under their documented temperature limits, and as such we currently do not see a need to implement an advanced cooling system. However, in the future, as we adjust existing components or add new ones, we will continue to monitor their temperatures and performances to determine if a cooling system may be necessary.

K. LED Strip Indicator

During AUV operations, the vehicle is frequently changing through the following states

- **Driving** - moving to the next programmed way-point
- **Seeking** - looking for an object using onboard vision system
- **Tracking** - with an object identified, move with respect to that object
- **Trick!** - motion programmed to be fun
- **Standby** - programmed wait time for the next command
- **Error** - robot not functioning as expected

To ensure that the patterns can be easily and intuitively interpreted, the team adhered to the following specific principals:

1) Users should not need a key to understand what a sequence represents; patterns should rely on intuitive symbolism (colors and sequences) corresponding to their message.
2) Patterns must be visually distinct in shape, speed, and color to maximize differentiability.
3) Patterns must not solely rely on color to communicate information. Color is easily distorted in underwater environments.

The following figures show the LED strip sequences, with each row representing a moment in time.

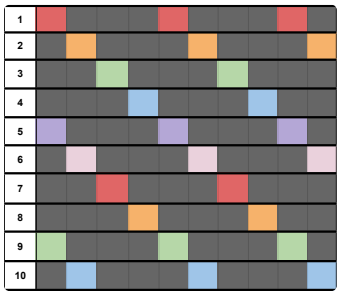

Fig. 22. “Shifting” represents the Drive to World Waypoint (Driving) command. The consecutive light shifts should invoke a forward feeling.

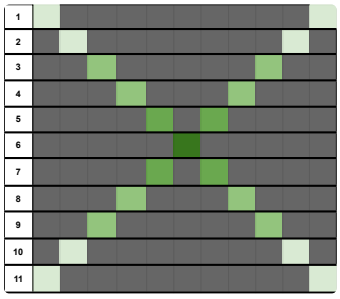

Fig. 23. “Cross” represents the seeking portion of the Drive to World Waypoint (Seeking) command. The traveling lights indicate searching.

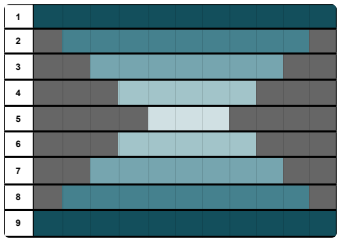

Fig. 24. “Trailing” represents the Track Object Waypoint (Tracking) command. The trail towards the center indicates a focus on a found object. Trailing makes use of intensity and timing to signify focus.

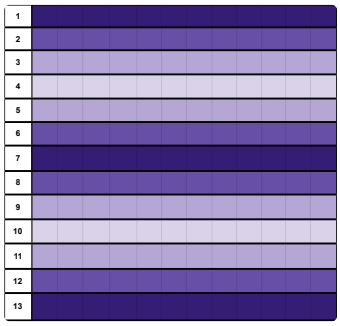

Fig. 25. “Pulse” represents the Idle / Standby command. Pulse makes use of light intensity to create a heartbeat effect.

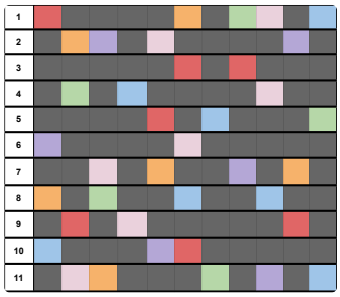

Fig. 26. “Twinkle” represents the Trick command. The colors and random twinkle pattern create the sense of fun!

L. Graphical User Interface

Due to the unique needs of our drivers and robotic systems, we opted to develop our very own interface from scratch. To make this possible, Cyclone RoboSub recruited UC Davis students from the School of Design.

The designs you see below were made in the design software Figma, and then transferred to Qt.

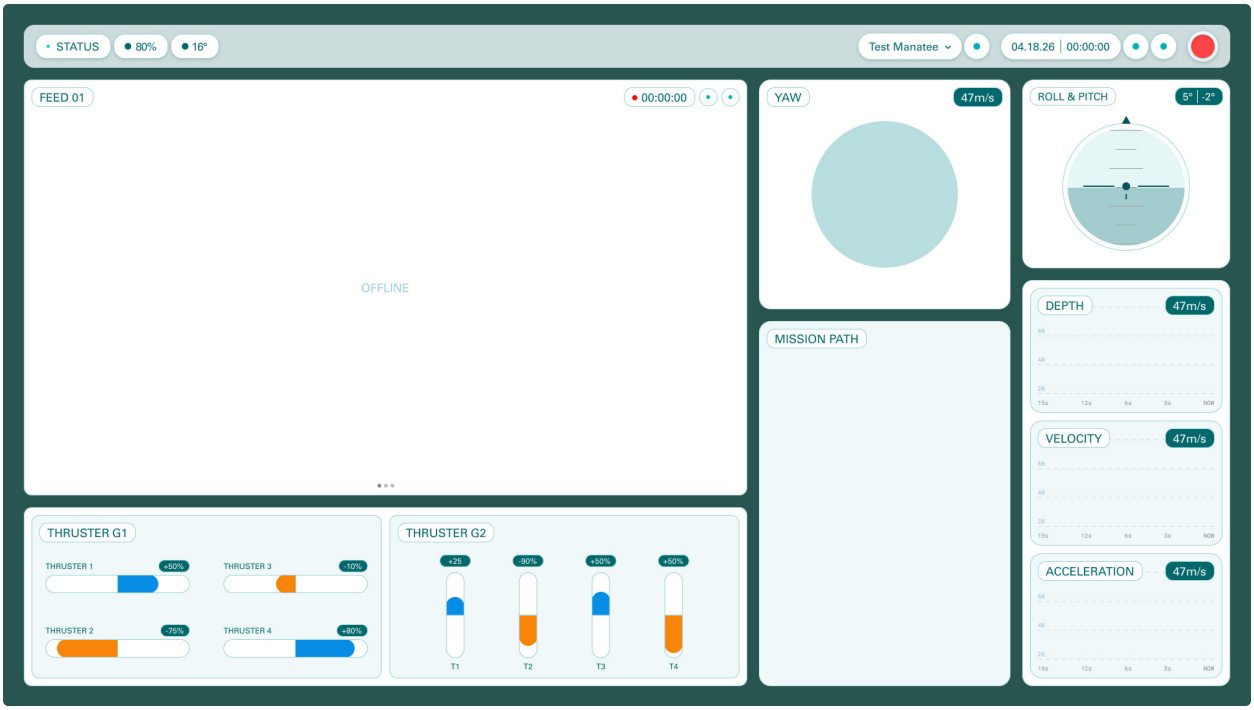

Fig. 27. Overall design of GUI Tidalwave Interface

Fig. 28. Design elements like dropdowns and graphs of Tidalwave Interface

a) *Code Implementation:* Tidalwave Interface provides the operator and the team with data visualization, monitoring, and control over the robot's state. These capabilities are achieved and implemented in C++ 20 through a Model-View-Controller architecture.

The Model component defines, attributes, and allocates common data that the robot and interface will exchange for their own respective needs, i.e., a protocol. Interfaces deal with data variables that are inherently different and therefore need their own allocation and definition. In this situation, the variables range from the state of the robot's modular systems to the physics state of the robot's movement. However, the variables are often categorizable and therefore should be organized and recognizable to the interface and developers as much as possible for ease of future development and testing.

The View component displays information and receives operator input information by utilizing the Model's allocation and implementing it through the Qt framework. The framework's abundant C++ API and Qt files help reduce the development time of both designers and developers with UI/UX elements that normally divert energy, time, and motivation from the implementation of necessary features and components inside the Controller and Model. Qt provides specialized development environments for respective skillsets and goals to enable an efficient workflow between designers and developers without each side having to leave their comfort zones. The API includes event-driven functions to help capture operator input and induce state changes on the robot and, in the future, the interface.

The Controller component serves as the kernel of the interface. As time progresses, the state and data variables of the robot and the user will change. The Controller's code deals with these transformations and facilitates the communication between the robot, the interface, and, eventually, through the View, the operator. The communication of telemetry between the robot and the interface is achieved through ROS2. ROS2 essentially reduces the concurrent workload of information to zero using the message-passing software paradigm. With the use of publishers, subscribers, and servers, data defined in the Model for the interface can be exchanged with the robot while allowing for Tidalwave Interface to deal with the operator's request from the View in parallel. C++ threads and monitors defined and used in the Controller facilitate the intra-process communication of Tidalwave Interface between the Model and the View.

The Model-View-Controller architecture allows our software team to follow the engineering design and testing process by enforcing modularity of components and providing a development workflow that allows for scalability of testing requirements, feature complexity, and requirement changes. Developers start to develop their own muscle memory in implementing new changes to the Tidalwave Interface through the architecture's intentions and division of labor foundations.

M. Robot Change Log

Despite using the same vehicle from the 2025 competition, we have included a slew of improvements across the board intended to refine its design. The following is a list of changes we made that were not mentioned in the main body of the report.

a) *Robot Handling:* The need to improve the structural integrity of Manatee was made apparent during pool testing, where one of the thrusters cracked due to a collision. Adding thruster guards for the angled horizontal thrusters was done to prevent further damage to both the frame and the thrusters. To improve ease of handling for Manatee, we also added handles that made transportation safe and easier for team members.

b) *Thruster Guards:* The fast-spinning propellers on Manatee's eight thrusters pose a danger towards people handling it, as well as the thrusters themselves. These issues became particularly apparent when Manatee was deployed in one of Davis' outdoor ponds. During this deployment, Manatee's thrusters were repeatedly caught in netting, roots, and all sorts of debris, damaging its propellers and risking mission failure. In addition, engineers who were handling the robot expressed concern towards having their fingers so close to fast-spinning propellers. These problems clearly called for a solution: a "thruster guard" that would block fingers and debris, while having minimal impact on the thrusters' thrust characteristics.

Initially, the team tried to address this issue with fully 3D-printed thruster guards, shown in Fig. 29. Radial grates were spaced such that fingers and large debris could not reach the propeller, and the entire assembly would cleanly clip onto the thruster without any secondary components. Unfortunately, this design was only good for addressing one problem: the safety of people handling the robot. It's efficacy was subpar for outdoor deployments, where fine detritus was still able to get past the large gaps between rings and wreak havoc on the propellers.

Fig. 29. The previous thruster guard design.

Addressing this secondary design criteria required a pivot in thinking. While plastic 3D printing has its benefits in automating manufacturing and allowing for repeatable & complex geometries, its relatively coarse resolution made it infeasible to block fine detritus while minimizing thrust loss. An alternative material and manufacturing method was required.

To meet this goal of shrouding the thrusters, a guard was designed with 1mm aluminum mesh mounted to a 3D-printed rim, as seen in Fig. 30. This design was chosen because it protects thrusters from a wide range of debris sizes while having minimal impact on the thrusters' thrust curves.

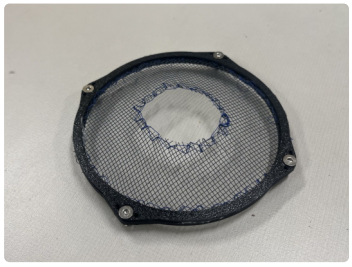

Fig. 30. One of the new mesh guards

A fine 1mm aluminum mesh proved to be the perfect answer to this call. While the guard's rim is still 3D-printed, the rest of the guard (the functional part of it) is made of the metal mesh. This mesh is mechanically clamped by the rim. This design meets all of our pre-defined design criteria; the guard effectively blocks large and small debris, and testing has shown it has minimal effect on the thrusters' thrust curves. A full design revision can be found in.

c) *Internal Electronics Mounting:* The main design goal for the electronics mounting structure is maximizing the efficiency of space usage in the waterproof enclosures. Last year, our initial design featured a T-shaped layout designed for visibility, accessibility, and modularity that features a large surface area and clean wire organization. This year, after keeping the electrical system largely constant, we observed that the current configuration meant that components often came unplugged or damaged during testing as a result of shifting in the tube. Thus, our modularity had caused system unreliability over time.

Because the vehicle's core electronics had reached a stage of permanent integration, the requirement for modular flexibility was superseded by the need for structural stability and wire protections. The primary success criteria for the redesigned bracket included the establishment of permanent component footprints, integrated wire routing channels, and a low-profile geometry that adhered to the physical constraints of the cylindrical hull. Furthermore, the design had to facilitate rapid maintenance while providing a dedicated, non-reflective channel for LED status indicators to ensure they did not interfere with the the front-facing cameras.

N. Environmental Research

Beyond the competition, Cyclone RoboSub is contributing to environmental research efforts through field deployments and interdepartmental collaborations. Equipped with a detachable module containing independently wired sensors to measure temperature and depth, the vehicle is able to collect environmental data. Currently, Manatee has completed deployments in the UC Davis Arboretum, conducted research for the Tahoe Environmental Research Center, and is gearing up for a research project with the UC Davis Bodega Marine Lab in Fall 2026.

Fig. 31. Deploying Manatee and attached research module into the UC Davis Arboretum

In order to fulfill our goals with data collection, Cyclone RoboSub developed a module to attach to Manatee. This independent module utilizes an Arduino Uno writing data to an SD card onboard, independent of Manatee. Most notably, our design is modifiable, and can swap between sensors depending on its purpose. So far, we have tested with temperature, pressure, and pH sensors, but it is capable of handling up to six unique sensors. In combination with the Doppler Velocity Logger, IMU, and the onboard RTC module, we combine geospatial records with various data trends from our sensors.

a) *Mechanical Design:* The mechanical design for the research module called for the ability to attach and detach from Manatee, house module-specific electronics in a watertight space, and hold multiple sensors relevant to data collection. The resultant model can be seen in Fig. 32 and Fig. 33, and is comprised of a grid mounting plate, a top and bottom end cap, an internal mounting plate, a sensor crown, and an electronics enclosure.

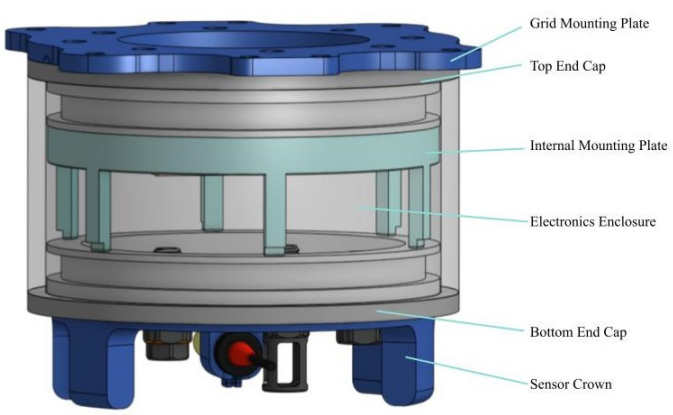

Fig. 32. CAD assembly of research module

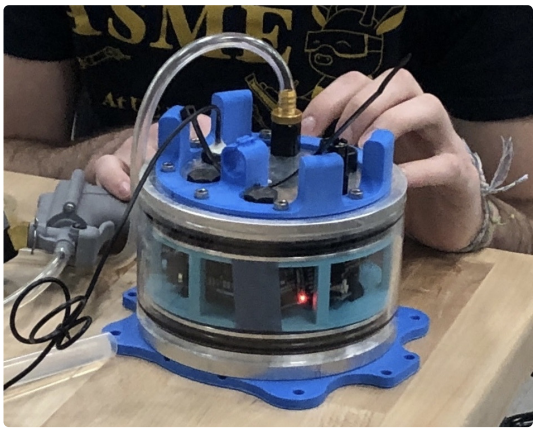

Fig. 33. Physical assembly of research module

For the main body, which is made up of the endcaps and electronics enclosure, we wanted to use materials that are resistant to corrosion and wear from deployments and general handling.

- ***Electronics Enclosure***: To ensure that we could properly read indicator lights from our electronics, monitor possible waterlogging of the enclosure, and still meet the materials requirement for the main body, we acquired acrylic tube and cut it to length.
- ***End Caps***: The end caps were designed to fit into and effectively seal both ends of the acrylic enclosure. Aluminum disc stock was processed via CNC mill to obtain the general form of each end cap. Then, with a 3D-printed jig for hole positioning, we were able to drill press the bottom and top end caps for their respective purposes; the first to fit various sensors and allow for the attachment of the sensor crown, and the latter to attach to the module's mounting plate. Lastly, each end cap was turned to create dimensionally precise O-ring grooves. The finished products and their jig can be seen in Fig. 34.

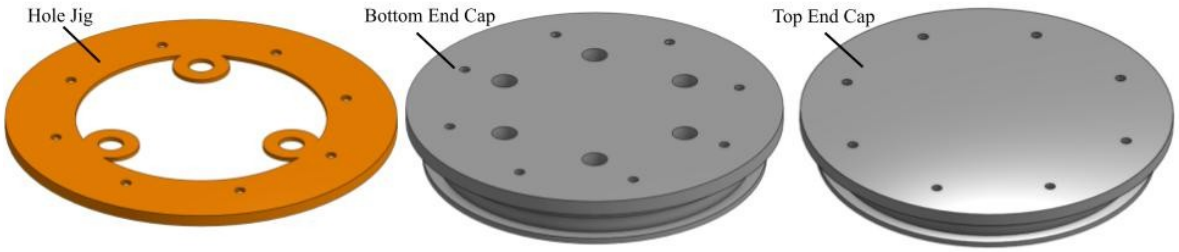

Fig. 34. CAD models of hole jig and end caps

For the various attachments within and upon the module, we opted to use 3D printing for more efficient and flexible design iterations. Each are pictured below in Fig. 35.

- ***Sensor Crown***: The sensor crown was designed to hold our pH sensor probe, with a simple clamp design featuring a slight gap for easy removal of the sensor's wire. Screw holes along the circumference of the crown allow for us to fasten it to the corresponding screw holes on the bottom end cap. Lastly, the four filleted "legs" extending from the surface of the crown prevent Manatee from resting on the pH sensor and other sensitive instruments protruding from the bottom end cap when the vehicle is set down.
- ***Internal Mounting Plate***: The internal mounting plate is meant to facilitate easier organization of our research module's electronics. It features small slots that velcro can be slipped through in order to bind our I2C level converters and battery down to the plate. Additionally, it has screw holes for fastening both the Arduino Uno and pH meter down. Lastly, it has a small divot for the battery to rest in, as well as a one centimeter lip to retain stray wires.
- ***Grid Mounting Plate***: The grid mounting plate has tapped holes that align with the screw holes in the top end cap, as well as holes around its perimeter that allow us to attach the whole assembly to the bottom of Manatee.

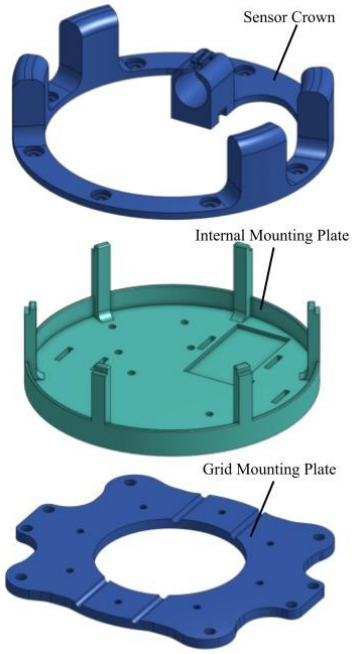

Fig. 35. CAD of 3D printed attachments

b) *Electrical Design:* Our electrical design is centered around the Arduino Uno, with every component wired to the board. Our main goal was to create a universal design, which meant having a modular interior. Our choice in I2C level converter enables us to switch the sensors out, allowing us to reuse the same mechanical and electrical design. The module is powered by a rechargeable 9 Volt battery, which is optimal for the short term profiles our team runs with the module. Connections are secured using Wagos and electrical tape, condensing the wiring. In the future, we plan on improving our design by switching to Raspberry Pi, allowing for GPIO use, as well as better data storage. We also plan on including a PCB, further cleaning up the wiring while still allowing for modularity.

c) *Software Design:* Our software was created in Arduino IDE and runs through our Arduino Uno. The main focus of the code is the states it steps through, from the startup cycle path to the continuous data collection phase. Once the code has initialized, it reads the sensors attached: the RTC, temperature, pressure, and pH sensors. All of these data points are consolidated into a singular packet and saved to an SD card, where it can easily be interpreted and graphed. While this is the current code setup, our team is working on porting this to Python and Raspberry Pi, in tandem with the electrical changes.

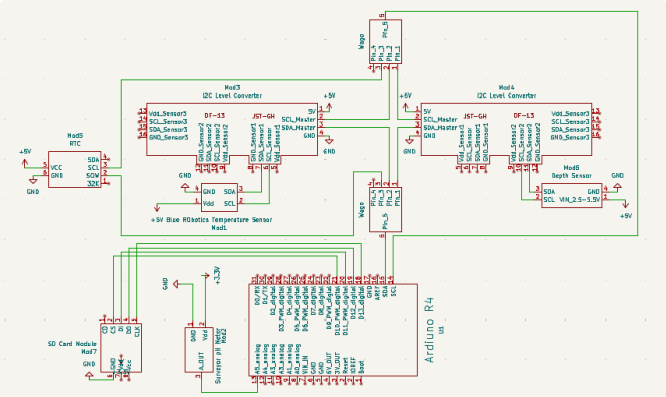

Fig. 36. Circuit diagram of the research module

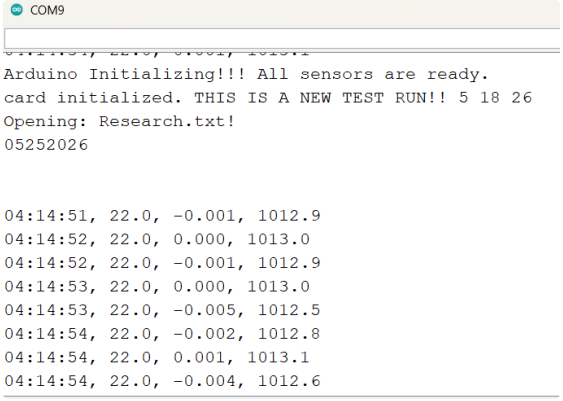

Fig. 37. Serial monitor

d) *Waterproofing:* We use a number of methods to waterproof the research module. To account for gaps between the end caps and the electronics enclosure, we fit large O-rings into the grooves of each of our end caps and applied molycoat, a lubricant to help increase contact. For the bottom end cap, through-holes that are not occupied by sensors are "plugged" by bolts fitted with O-rings. Lastly, as a precautionary measure, we wrap the circumference of the end cap-enclosure interface with marine tape.

Before each deployment and pool test, we retest the research module's resistance to water leakage with the following procedure:

1) Remove electronics from electronic enclosure.
2) Reseal research module, and remove cap from water-tight enclosure vent. Affix the pressure pump's tube to this slot.
3) Using the pump, increase the module's interior pressure to 15 atm. Ensure that the pressure stays constant above water.
4) Submerge the research module completely, simultaneously inspecting the pump's dial for changes in pressure.
    a) If the dial reads constant pressure, the module is watertight and can be left submerged for any desired amount of time.
    b) If the dial reads a change in pressure, the module likely has a leak and should be removed from the water and inspected.

Fig. 38. Pressure testing research module

5) In either case, the module should be inspected for water leakage. Small droplet trails along the enclosure walls often indicate problems with the end caps' O-rings, while droplets along the wires may indicate leakage from ill-sealed sensors or bolts in the end caps.

e) *pH Calibration:* Integrated into the code is a pH sensor calibration. In order to output consistent pH values, we undergo the following pH procedure.

1) Connect Arduino R4 to computer, and restart the program.
2) Open the Serial Monitor to the correct port.
3) Unscrew the cap on the pH sensor. Carefully rinse with MilliQ water, gently wipe off, and place in 7 pH solution.
4) Wait 10 seconds, and input "CAL, 7" into the Serial Monitor. Remove from solution, rinse with MilliQ water, gently wipe off.
5) Repeat for "CAL, 10", and if desired, "CAL, 4". If the readings need to be cleared, type "CAL, CLEAR" into the terminal to reset the readings.

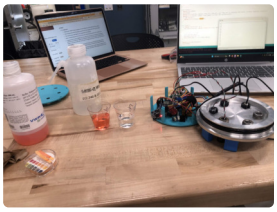

Fig. 39. pH calibration

f) *Extraneous Mechanical Design:* As the research team evolves and takes on different tasks for different deployments, we are sometimes asked to collect data that involves more than just the research module. Extra modeling and attachments are required to meet those needs.

- ***Front-Facing Headlamp Mount***: For our May 2026 TERC deployment, we anticipated that we would need a strong light source in order to collect visual data from our front-facing camera in turbid, algae-rich waters. After modeling our selected headlamp, we designed a simple mount to attach to Manatee's chassis and the back of the headlamp's body. The complete assembly can be seen in Fig. 40.
- ***Downwards-Facing Headlamp Mount***: Though we did not end up needing a downwards-facing light for our TERC deployment, future imaging might benefit from having a light source for our downwards-facing camera as well. We plan to design a simple mount for a downwards-facing light in the coming year.
- ***Light Logger Mount***: For next year's partnership with Bodega Marine Lab, we plan to design a mount for a photosynthetic active radiation (PAR) sensor that will be provided to us, as pictured in Fig. 41.

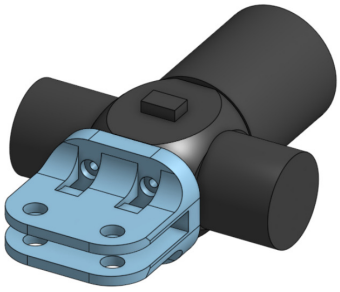

Fig. 40. CAD model of headlamp and headlamp mount

Fig. 41. Odyssey Xtreem photosynthetic active radiation logger

g) *Research Deployments:*

- UC Davis Arboretum Transect – The UC Davis Arboretum is a waterway within the campus which follows Putah Creek. Manatee, equipped with our research module, collected temperature, pH, and depth data at Lake Spafford in the UC Davis Arboretum during both practice and full deployments.
- UC Davis Tahoe Environmental Research Center - In May 2026, we successfully completed our first serious field test of Manatee's research capabilities, benefiting from the water clarity and low current conditions of Lake Tahoe. To assist with TERC field researcher Brandon Barry's aerial imagery of near-shore algal growth, we recorded the lake floor with our forward-facing and downward-facing cameras. We are now processing the visual data and interpolating it with dead reckoning report positional data from the DVL.

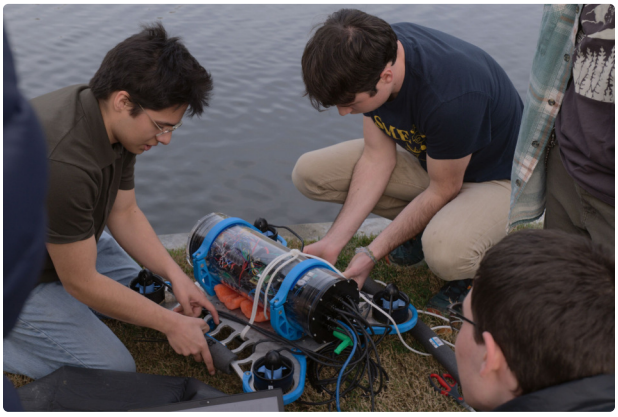

Fig. 42. Deployment at the UC Davis Arboretum

Fig. 43. Dock deployment at Tahoe City Marina

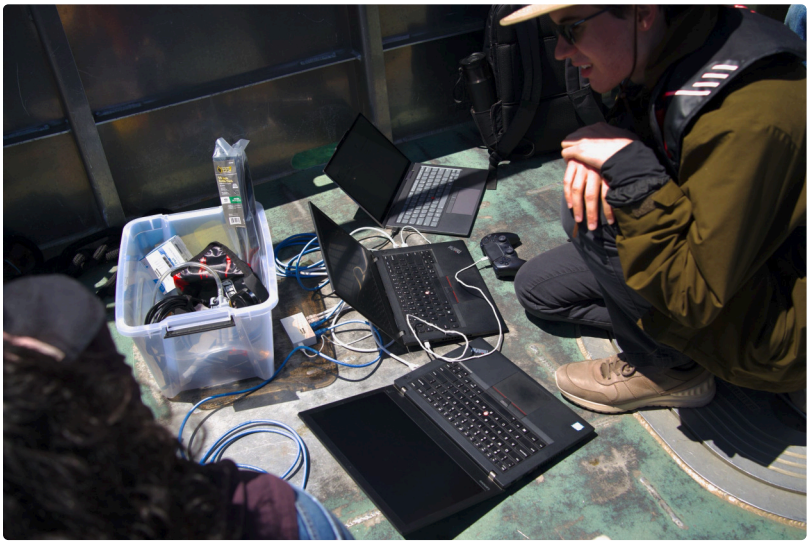

Fig. 44. Students working aboard the R/V Bob Richardson

Fig. 45. Students deploying Manatee in Lake Tahoe to collect underwater periphyton imagery

O. New Member Training

Introducing new members to the team and ensuring they have a meaningful experience is one of the fundamental missions of our team. We leverage accessibility in a number of ways including the decision to omit membership dues and open leadership applications to anyone with an interest regardless of experience.

To ensure a positive new member experience, all sub-team leaders are responsible for preparing work packages for projects. Sub-team leads then delegate these work packages based off of the incoming members' experience levels and motivation.

Work packages vary in scale from full scale projects to small tasks that can assist the overall goal of the sub-team. As opposed to last year, full scale projects go through preliminary and critical design reviews. We believe in providing real-world hands-on experiences to all our members.

a) *Battle Boats:*

2024 saw the introduction of a new team member training activity called Battle Boats. This group-based, competition-style technical training served as a fun and educational introduction to marine robotics for new members. Details of the program are listed below. This past year we had 50+ students participate, setting the fast-paced, community-oriented tone that we would maintain throughout the year!

- Premise - Team members would randomly be assigned to groups of 5 and tasked with the design and build of a small, autonomous boat capable of navigating a floating course. Trainings would be held to teach members the basic skills needed to build the vehicle and members could decide who would focus on which area: software, electrical, or mechanical.
- Competition - At the end of the 4 week training, teams would have the opportunity to test their boats on the course and tweak their designs. Teams were provided with ultrasonic sensors and Arduino Nanos, along with basic electrical supplies. Boats were 3d printed and each team was given a filament cap of 200kg.
- Results - 12 different teams competed with over 60 students participating in the challenge. The competition style trainings encouraged personal investment and forced teams to think critically about how they designed their vehicles. Each battle boat was completely unique and teams were introduced to concepts of autonomy, propulsion, and buoyancy.

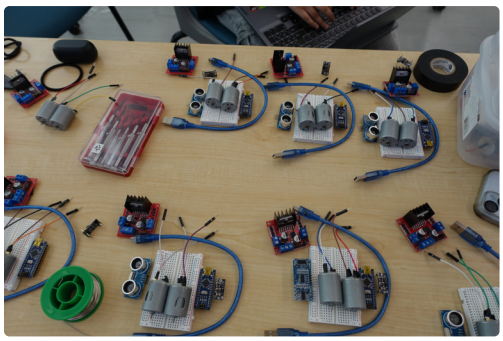

Fig. 46. Provided Supplies

Fig. 47. Battle Boats Course

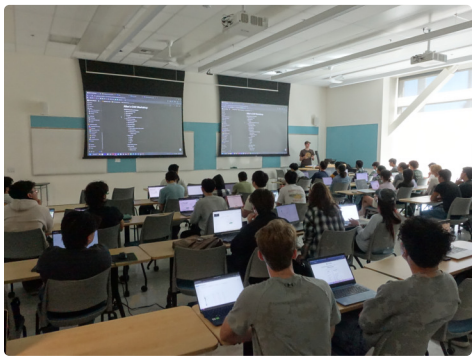

Fig. 48. CAD Workshop

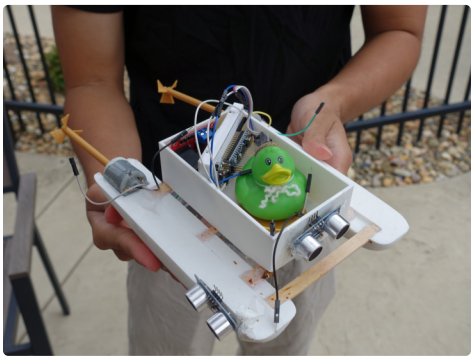

Fig. 49. Example Battle Boat

P. Complete Member List

Name	Subteam(s)
Aditi Anand	Research
Aishwarya Tawade	Hull, Research, Mechanical
Albert Vo	Controls, Vision, System Architecture
Alexander Zamora	Controls
Allen Cooke	Mechanical
Andrew Li	Hull, Public Relations, Mechanical
Andrew Ngo	Vision
Andrii Hurzhyi	Hull, Manipulation, Electircal
Anirudh Yanamandra	Hull, Electircal
Anthony Yan	Hull, Electircal
Anya Dhillon	Hull, Manipulation, Mechanical, Electircal
Bryan Lei	Hull
Calen Everage	Controls, Electircal
Chonticha Phanphanit	Hull, Public Relations, Business
Danny Kwong	Software
David Kou	Manipulation
Declan Whitlock	System Architecture, Public Relations
Devansh Katiyar	System Architecture, Software
Devi Amarsaikhan	Vision
Diego Yu Huang	Manipulation, Mechanical
Dilraj Gill	Electrical
Divyansh Singh	Vision
Frank Faulkner	System Architecture, Software
Grant Judkins	Research
Hannah Kench	Research
Ian Hsu	Research
Isa Khalak	Hull, Electircal
Isabella Mo	Electircal
Issac Mooc	Hull, Electircal
Jason Pieck	Software, Hull
Jason Kai	Hull, Mechanical, Electircal
Jingrui Wang	Controls

Name	Subteam(s)
John Olive	Electrical, PR
Keymora Panaligan	Hull, Manipulation
Kory Haydon	Controls
Lauren Mizerak-Mohun	Research, Mechanical, Electircal, Software
Liam Multhaup	Controls, Vision
Logan Wong	Hull
Luci Masselink	Hull
Luna Lark	Software
Mae Ngo	Controls, Research
Marco Chen	Electircal
Mason Lafleur	Electircal
Matthew Lin	Controls
Maya Malinowski	Controls
Michael Tisdale	Electrical
Minh Ho	Hull, Manipulation, Electircal
Naia Dalal	Electircal
Nidhi Vadulas	Electrical
Peter Webster	Research, Hull
Quyen Tan	System Architecture
Rajeev Raghuram	Vision, Software
Ruby Stanton	Controls
Samantha Tsai	Vision, Software
Sherry Lei	Vision, System Architecture, Electircal, Software
Stanley Jiang	Hull, Electircal
Tanishq Dwivedi	System Architecture, Software
Tristan Najarro	Hull, Mechanical
William Barber	System Architecture
Wingyan Yu	Controls, Electircal
Winter Chan	Manipulation, Electircal
Yannie Li	Hull, Research
Ziqiang Zhu	Vision
Zoe Chan	Hull, Mechanical, Electircal