Guppy: RoboSub 2026 Technical Design Report

Washington State University, Pullman, WA (Palouse RoboSub)

Cole Wilson, Allison Kim, Andy Sorge, Brady Hogg, Natalie Massarenti, Daniel Bereza, and Oliver Ehlers

with Adrian Nelson, Alex Johnson, Alexander Bruhl, Alexander DiRezza, Ben Rao, Carson Roskelley, Colton Raulerson, Dylan Dean, Gregory Jabido, Gunnar Oye, Henry Brunz, Jaden Panasovets, Jay Michalczuk, Joshua Jamrosz, Kaitlyn Dunphy, Lindsey Muilenburg, Lucas Lovrien, Nicholas Horner, Pyotr Korotkov, Reagan Fallon-Small, Ryden Handsaker, Suchith Sunku, Thomas Bissell, Timothy Guier, Vanessa Anderson, and Wyatt Ballweber

***Abstract*—Palouse RoboSub is returning to competition for the first time since 2018 with their debut AUV "Guppy," after a complete post-Covid team rebuild. Guppy's electrical, software, and mechanical architectures have been completely designed and manufactured in-house by a team of roughly 35 undergraduates, with the goal of pre-qualifying for the RoboSub 2026 competition. An emphasis has been placed on creating a reliable, viable AUV for the team's first year back, with the plan to extend Guppy's functionality in the coming years. A comprehensive overview of our competition and design strategy follows.**

I. Competition Strategy

A. Team Background

While its current inception has only existed for about a year, Palouse RoboSub originally started as an electrical engineering capstone team at Washington State University (WSU) in 2011, and quickly grew to a large club competing for the first time in 2012. For the years of 2014 and 2015, the University of Idaho joined the team as well. Despite the team's success and growth, 2018 was the last year they competed. Additionally, with the advent of COVID, the team did not return until 2025.

The club started fresh with an entirely new group of members, no sponsors, and almost no funding. The team has seen remarkable growth in the past year and a half, growing from three members in Fall 2024 to roughly 35 members in Spring 2026. Composed primarily of underclassmen in their first or second year of engineering, the team has been approaching the RoboSub competition from the standpoint of a completely rookie team with little to no prior experience in robotics.

As we are essentially a brand-new team, the focus of the past year has been getting up to speed with making AUVs, ROS 2 programming paradigms, control theory, custom PCB development, and mechanical manufacturing and assembly of watertight enclosures. We are all extremely proud of our submarine Guppy, which is primarily designed as a platform for future expansion as we gain competition experience. We've also focused on community outreach and events (Appendix A).

It's important to note that all development, manufacturing, assembly, and testing of Guppy and associated control software occurred between the months of April 2025 and May 2026. Washington State University is on the semester system, and finals end the first week of May. Therefore, nearly all physical development was completed before May.

Fig. 1. Guppy: Palouse RoboSub's Autonomous Underwater Vehicle.

B. Core AUV Competition Goals

Our main hope for Guppy in RoboSub 2026 was to pre-qualify for competition, which we have successfully accomplished! For this to happen, we had to break the process down into the following sub-goals:

- Research and design an AUV concept.
- Manufacture the mechanical components of the AUV.
- Design and assemble an electrical harness.
- Develop and write manual control software.
- Develop and write autonomous control software.
- Test and iterate on the design.

As we were on a very tight timeline, this was our primary goal for competition: *to have a working submarine that can successfully navigate autonomously under the water.* Anything above and beyond that was fantastic, but not our primary objective. We planned to leave school at the beginning of May with a solid platform for future development, and one that could hopefully get us some points in competition.

As for concrete task goals in competition, we planned for the tasks that primarily focused on mobility (rather than additional subsystems such as an arm). These were the starting gate (Begin Assessment), the slalom task (Avoid Debris), surfacing inside the octagon (Resupply), Return Home, and (pre) qualification.

We are mainly aiming for "Core" points for these tasks according to the task capability matrices, although we haven't been able to complete a full points-based analysis because the points for the tasks have not yet been released. To complete

some tasks, including the "Advanced" tracks, we need more testing and integration of our vision systems. Currently, most of the tasks we can complete are based on dead-reckoning, however over the summer we hope to finalize the vision system.

C. Additional AUV Goals

Despite the focus on mobility-only tasks, our club had several teams developing additional capabilities. These include torpedoes (Deploy) and a claw (Recon and Resupply). However, due to time constraints, these features will likely have to be pushed to next year. Our rubber-band-powered torpedo prototype has shown extreme potential, and there is a chance we will be able to attempt it at competition. Our claw is also quite far along in its design but needs a bit more development time due to task objects not matching our size expectations after our initial design.

As we plan to re-use Guppy in the future, we will have plenty of time to develop these capabilities over the next year. We plan to move to a much more rapid development/test cycle next year, which will allow us to compete in 2027 with torpedoes and a claw. Our current task capabilities are listed below:

Task	Planned	Capability Matrix
Prequalification	Complete	–
Begin Assessment (Gate)	Yes	Advanced
Avoid Debris (Slalom)	Yes	Advanced
Resupply (Octagon)	Yes	Core
Return Home	Yes	–
Deploy (Torpedoes)	Potentially	Core
Recon (Bin)	No	–

Table 1. Current Guppy task capabilities for RoboSub 2026.

D. Documentation and Media Goals

Another major focus of this year for Palouse RoboSub was building up our documentation, team website, media presence, and code repositories. We've completely revamped the team website, using modern Next.js web frameworks and a component-based architecture. We've also integrated a custom documentation website which automatically pulls from our code repositories' README files. Each ROS package has up-to-date documentation and installation instructions. We plan to have a strong backbone of documentation and information for future team members and to increase approachability for new members and sponsors.

E. Business and Financial Goals

Finally, this year we placed a huge emphasis on initiating contact with sponsors and fundraising. For our club to continue functioning and ascend to a higher level of competition, ongoing financial backing and support is necessary. WSU does not provide funding for our team, so all monetary support comes from sponsors, family donations, or personal team member donations. To compete at a high level, and afford advanced sensors such as a DVL, we heavily rely on sponsors and external funding.

We reached out to many companies, both local and global, via email, cold-calls, and associated contacts. Many companies offered us a significant discount on their products, which we are extremely grateful for. Additionally, several companies sponsored us with direct monetary donations. Through a crowdfunding campaign ("CougStarter") in Fall 2025, our club raised a record $7k, which enabled us to purchase a heavily discounted DVL from our sponsor WaterLinked. Our club successfully went from nearly zero resources to raising over $14k total in sponsorships, hardware, and fundraising. We plan to continue to reach out to sponsors next year, as well as to maintain the relationships we already have, to fund future development and travel.

II. Design Strategy

A. Mechanical Architecture

Guppy weighs roughly 34.1 kg (75.2 lbs.) and measures approximately 27" × 25" × 15" (including thrusters). It has eight T200 thrusters and can move in all six degrees of freedom. Guppy is composed of two tubes atop an undercarriage, and four custom milled endcaps, each with 16 penetrator holes. To access the internals, one removes the two endcaps on the rear of the submarine, which allows you to then take off the tubes. After this, the only thing left is the electrical shelving, which remains attached to the front two endcaps.

Fig. 2. Haas VF2SS with a TRT210 trunnion and a Guppy endcap.

The mechanical architecture of Guppy is broken up into several main subsystems: internal shelving and mounting, endcaps and tubes, baseplate and undercarriage, sensor housings, and tools (torpedo/claw). Each of the subsystems had unique challenges and features, as well as different material properties and manufacturing processes.

1) Internals: The internal shelving allows for mounting electronic components, as well as clocking to the frame in order to keep the internal IMU level. The shelving itself is lasercut acrylic with M3 holes on a 10mm grid. The shelving supports are 3D-printed CoPA Nylon, and the electronics mounts and LED support rings are PLA. There is also some aluminum ballast (in addition to exterior weights) to make Guppy heavier.

2) Endcaps and Tubes: Guppy has two 16.5" long hulls that are 8" inner diameter and ¼" thick acrylic tubing. Each tube has two custom endcaps, which are 5-axis 6061 Aluminum parts machined by club members in the school's CNC shop on a Haas VF2SS with a TRT210 trunnion. Each cap has two o-ring grooves for Parker ORD-5700-265 o-rings. Each cap has 16 M10 through-holes designed to fit Blue Robotics penetrators. The exterior rim of each endcap additionally has tapped 8/32 holes for mounting to the battery hull and baseplate. Each cap is rigidly attached to the baseplate using three of these mounting holes, which ensures proper hull alignment.

3) Baseplate and Undercarriage: The baseplate is 0.235" 6061 Aluminum with an iso-grid hole pattern and several tapped bolt holes for mounting the endcaps, undercarriage, and motor mounts. The undercarriage is made of 1" × 1" square tubing, with various gussets and brackets to affix it to the plate. There are also printed "feet" to protect the corners. Our motor mounts are SLA resin prints designed to stay rigid over time. Initially, we tried printing our mounts out of CoPA, although we found that it became too ductile under the constant stress of the motors, so we switched to SLA for increased rigidity.

The undercarriage is designed to be a versatile mounting point for peripherals such as camera housings, claws, torpedoes, lights, and more. We will make extensive use of this next year during iterative prototyping and development.

4) Camera Housings: Each camera housing is a solid enclosure printed in resin. This allows us to be sure of the housings watertightness, and is significantly cheaper than machining an enclosure. The front of each enclosure is a flat acrylic plate that is bolted on with an o-ring to seal it, and the rear fits two M10 size penetrators for power and data. The current iterations of the housings are designed for the smaller Flir lens sizes, although future models are in development to accommodate larger fisheye lenses.

5) Tools and Peripherals: Both the claw and the torpedoes were initially printed with PLA for prototyping, although the final iterations are nylon CoPA prints. The claw consists of a single servo motor which drives a gear to work the actuator. The torpedoes contain rubber bands wound tightly before launch, releasing their elastic energy as they travel through the water. In order to release each torpedo, a small servo lifts a retaining arm, allowing each torpedo to shoot forwards.

For more in-depth information on our mechanical architecture, along with renderings and drawings, please see Appendix H.

B. Electrical and Sensor Architecture

Guppy's electrical architecture is composed of harnesses connecting COTS sensors with our computer and several custom PCBs designed and assembled in-house by our electrical engineering team. Analysis, schematics, and renderings for these boards can be found in Appendix C. Aside from the PCBs, the architecture is broken up into the CAN bus, ESCs, sensors, batteries, and connectors.

1) CAN Bus: Our robot is divided cleanly into two separate hulls: Actuation and Compute. For each computer or chip on the submarine to communicate between the tubes with each other, we use a CAN bus. One terminus of the network is at the LattePanda Sigma computer in the Compute hull, and the other is at the last custom PCB in the Actuator hull. Our custom PCBs use CAN, and are nodes on this network. This enables us to run only one data wire between the hulls, saving on penetrator space while leaving room for future development. If we add additional boards or devices to a tube, we only need to splice into the existing CAN pair to gain full access to the entire network. We give each message type on the bus a unique identifier, bitmasked to the board it is related to. This system lets us quickly identify which CAN packet originates from where, and allows future extension of the bus.

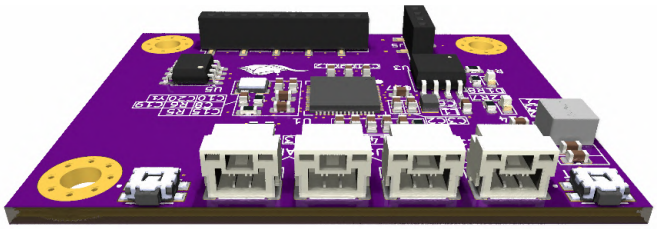

Fig. 3. CAD rendering of V2 RP2350 Mainboard used on Guppy.

2) Electronic Speed Controllers (ESCs): We use ReadyToSky 35A ESCs running BLHeli_S firmware. These motor controllers are connected to our PCB using a PWM daughterboard, which stacks on the base custom PCB. For more information on this setup, see Appendix C. The ESCs are connected to eight T200 thrusters using soldered inline bullet connectors.

3) Sensors: Several of Guppy's sensors are COTS sensors we have either bought or inherited from previous teams. For an IMU, we have a sponsored VN-100 from VectorNav Technologies which provides us with high-resolution inertial measurements and orientation data. We also use a WaterLinked DVL-A50 as our Doppler Velocity Log, which provides us with very accurate relative velocity data, as well as dead-reckoning position estimates. Both these sensors communicate directly with our single-board-computer, the LattePanda Sigma from DFRobot.

Additional sensors include two rotary switches from Blue Robotics and bolt-style reed switches from McMaster Carr which serve as generic GPIO inputs and power switches. We also incorporate a Blue Robotics High-Resolution Depth/Pressure sensor for accurate Z-axis measurements. We also rely on several Logitech USB webcams, as well as several USB and Ethernet Flir/PointGrey high resolution cameras for our vision system.

4) Batteries: We are using 5S Lithium Polymer (LiPo) batteries for Guppy, primarily because we already had them on hand from previous members and they were held correctly at storage charge. Our main competition batteries are 22Ah 40C 5S LiPo batteries from MaxAmps, and we also use a 5S2P 12Ah battery from ThunderPower as well, for testing and bench tasks. This capacity allows extended testing periods without needing to change batteries. There were initially concerns at running the T200 thrusters above their rated maximum voltage of 20V, although we did extensive research and found historically that this has not been an issue, and we did not encounter any issues in our testing. The batteries plug into a FlipSky anti-spark switch which serves as our main on/off relay for the sub.

5) Connectors: We use several standard connector types on our submarine. JST SM connectors were used for all inline splices, such as CAN connections from and to each endcap and through-penetrator Ethernet splices. Locking JST GH connectors are used on all PCB connections, such as for CAN, power, and PWM outputs. We use XT-30 connectors for all 12V power, XT-60 connectors for inter-hull power, and XT-90 for our main battery input.

The electrical wiring and architecture of Guppy is extensive and cannot be represented fully here. Full architecture and wiring diagrams can be found in Appendix G at the end of this report, and on our website (robosub.eecs.wsu.edu).

C. Software Architecture

The code for Guppy is split primarily into three parts: the ROS-based code that drives most of the "thinking" and decision making which runs on the LattePanda Sigma, the embedded code that runs on the custom RP2350 mainboards, and the control code that runs in a separate loop that does the physics calculations in parallel with the ROS code.

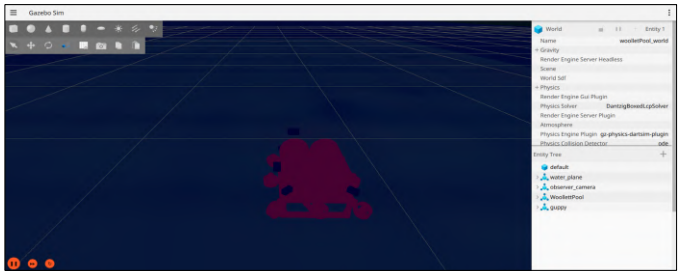

Fig. 4. Guppy's main ROS code being run in a Gazebo simulation.

1) ROS 2 Jazzy Architecture: Our team is using a completely ROS 2 based approach for our software stack. We chose this for several reasons, including interoperability with C++ and Python, which are languages familiar to beginner programmers at WSU. This works well with existing library implementations for our sensors. For a detailed overview of our ROS nodes and topics, please see Appendix F. We placed significant emphasis on separation of responsibility into modular nodes, something ROS excels at. At the core of everything is our state machine, which keeps track of several states: STARTUP, DISABLED, HOLDING, TASK, TELEOP, NAV, and FAULT (see Appendix D for corresponding LED patterns). Depending on the state, different command velocities are passed into the chassis controller (see below). Simultaneously, we run a majority of our navigation and decision code as ROS actions within a behavior tree implemented with the BehaviorTree.CPP library. This lets us easily sequence actions and fall back when encountering errors.

For localization, we rely on a custom pipeline based on an iterative PnP-based image solver. This system takes in odometry info from our DVL and barometer, and fuses that with detections from our cameras. We use OpenCV and YOLO to filter image detections and a PnP algorithm to determine relative pose. This pose is then updated over time in relation to other known objects and the robot's previous and current odometry to generate a fused, accurate pose. Not only does this system allow for accurate global navigation, but also local, task-relative localization based on the emoji glyphs as well.

CAN integration is easy with our custom ROS/SocketCAN bridge, which publishes received CAN IDs as ROS topics and provides a service for CAN transmission. We publish the state of the external switches and knobs over CAN which means we can toggle submarine state (for example, to disable the sub) purely by using a traditional ROS subscription. This modular workflow allows for easy integration of new sensors and data paths, as well as future simulation of embedded code and CAN devices.

Learning ROS was new to every single person on the computer science team this year, and much of our progress has come in the past six months as we learned from the ROS docs, as ROS is not taught at WSU. As we continue to grow and develop our codebase, we plan to develop our vision subsystem fully, as well as adhere to stricter ROS style guidelines. Currently, we place a large focus on consistent git usage and code style, as discussed in Testing Strategy later on. For more details on the ROS code, see Appendix F.

2) Embedded RP2350 Architecture: Guppy makes use of two RP2350 microcontrollers (one for each hull). They communicate over CAN to the main ROS computer and reflect Guppy's current state using the LEDs in their hull. See Appendix D for what the LED colors mean. The board in the Actuator hull handles the e-stop and thrusters. The other board in the Compute hull handles sensors over I2C and may handle other peripherals in the future.

The embedded code for Guppy is divided into modules and libraries. Each module represents the custom code for a particular board and each library is a piece of reusable code. We have a shared library "Guppy Lib" which contains CAN and PWM helpers alongside a custom LED library to manage the light patterns for different states.

3) Chassis Control Code: The chassis control code runs on a separate thread from the ROS node that interfaces with it handling thrust allocation, drag adjustment, and position holding. The thrust allocation is handled as a box-constrained least-squares problem, which is then translated into a Quadratic Programming (QP) problem as shown in Appendix B [1]. Several layered PID controllers handle velocity and position control in all six axes, and are configurable via ROS parameters. The code also compensates for wrenches induced by the robot's center of pressure and buoyancy, allowing nearly perfect buoyancy and attitude control.

This section of code (*outlined thoroughly in Appendix B*) is critical to Guppy's success, and is highly optimized. One loop of the control-critical code lasts roughly 250 µs (several kHz). We take pride in the capabilities of our control code, and it provides a fantastic base for building autonomous behavior. A major focus of this year for the computer science team has been building and testing this foundation to ensure a strong footing for future years.

D. Systems Integration

The club implemented the NASA/DoD design review lifecycle system into regular operations, requiring regular design reviews for submarine components. Guppy was kicked off by a System Design Review (SDR) in April of 2025, during which everyone got together and suggested potential research and ideas for a new AUV. In addition, we strive to have each mechanical component undergo both a Preliminary (PDR) and Critical (CDR) Design Review before being fully integrated into the sub.

Another major organizational structure we focused on during the main integration phase was daily deadlines revolving on these PDR and CDR milestones. From early January to early March (our in-water deadline), each subsystem had a dedicated lead and deadline. These timelines were reviewed at every general meeting, and tracked via a large, shared spreadsheet. The system was extremely motivating, and greatly contributed to successfully meeting our self-imposed in-small-pool testing deadline of March 1.

In future years, we hope to continue the design review system, and maintain strict deadlines and integration cycles. Now that the submarine base is fully built and running, we are aiming for faster iteration times broken into sprints between each regular pool test date.

III. Testing Strategy

A. Simulation

As stated before, a major goal of this year was gaining feature parity in simulation of the major control aspects of Guppy. The club knew going into the year that we would have very limited in-water testing time compared to other teams and would be running on a very tight timeline. By simulating our codebase, we were able to verify control and navigation algorithms well before the submarine was mechanically assembled and ready for in-water testing. As our academic year ends in the first week of May, we knew we would have little to no test time in the summer months either. By effectively simulating the robot to perform as it does in real life, we are able to continue testing and development of task code and navigation well into the summer, without the need for the AUV to be physically present.

For our simulation, we use the Gazebo (Harmonic) simulator as a base, while maintaining a custom thrust plugin to handle motor thrusts and bridging sensor data to and from ROS2 via Gazebo's ROS bridge. Our custom simulation environment (titled "GNCea") aims to be flexible and extensible for future years, and even other teams as well. The codebase lives apart from our main Guppy code and is maintained separately as well. This allows us to separate responsibility of the simulator from the submarine and maintain an AUV-agnostic environment for simulation.

B. Driver Station Setup and CLI

In all in-water testing environments (small or large pool), we used a "Driver Station" SSD that was set up with Ubuntu 24.04 and the requisite control software. No matter which physical host the SSD was plugged into, the operating system on the device would remain the same. This enables us to have repeatable testing regardless of the venue.

Additionally, we have a small command line interface (CLI) that allows us to quickly test and iterate on concepts. For example, we can run commands such as `state nav`, `goto x y z`, `restart_guppy`, or `estop` to run important or commonly used routines faster.

C. Small Pool

Fig. 5. The small pool set-up in our shop space prior to a KSED demo.

Due to a significant lack of funding and large pool availability, most in-water testing was performed in a small 8' diameter, 30" deep Intex above ground pool set up in a room across from our shop. This allowed for rapid iteration and prototyping in a controlled environment without the hassle of regularly transporting equipment across campus. With the addition of a

TDK-Lambda SWS600L-15 external power supply "Shock Monster," we were able to run Guppy tethered to wall power without the need to recharge batteries. However, we ran into significant issues while tuning PID and control algorithms in the small pool, due to the boundary conditions caused by waves in the relatively small volume. We would also regularly encroach upon our DVL's minimum range and would run into interference from the pool's walls as well, causing our localization estimates to drift drastically.

Despite these issues, the small pool still proved to be a useful component of our testing. We were successfully able to tune motor control and rudimentary navigation in the above ground pool, and it provided a test environment for mechanical prototyping as well. The pool was also useful for community outreach and events (Appendix A), and for easy access at all hours, with club members consistently working on Guppy after class and jobs until around 2 a.m.

D. Large Pool

Fig. 6. Guppy overlooking WSU's Gibb Pool on our first testing day.

Our club tested twice in Washington State University's Gibb Pool. Gibb Pool has eight 25-yard lanes between 3.5 and 9 feet deep, along with another 14-foot-deep rectangular section measuring roughly 10 × 20 meters. This deeper dive tank section is where we did our testing and prequalification run.

The first test day primarily focused on validating our navigation and control algorithms. As discussed before, the boundary conditions observed in the small pool posed some issues with tuning. Luckily, we were able to quickly identify these and correct them in the larger pool. We observed significantly less position drift and loss from our Doppler Velocity Log data in the larger pool. This by far surpassed our expectations and allowed us to use DVL based navigation instead of camera navigation for prequalification due to time constraints.

The second day (about 4 days later during Finals Week) focused on operating safely and effectively without a power tether. Our first day was run using external power, but for prequalification we transitioned to using a battery. Initially we operated with only an Ethernet data tether, allowing us to send commands and debug the submarine as we performed further testing. As this was the first full run making use of battery power, we regularly checked battery voltage during this time to make sure we didn't overuse them. Finally, we disconnected the Ethernet tether, and Guppy successfully finished the prequalification run autonomously.

In the future, we plan to spend significantly more time at the large pool. With weekly or bi-weekly testing, we know that our development productivity will greatly increase. A major goal of next year will be placing the team on a regular development cycle focused on consistent pool time. For this to happen, we will need to reach out to sponsors and/or raise a significant amount of money to be able to afford the rental fee as WSU does not provide this resource support for our club. However, this fee is well worth the experience and testing time our team would get. With more pool use, we will develop more exact testing schedules and procedures to make the most of our time.

E. Continuous Integration and Declarative Software Testing

Our computer science team uses proper GitHub continuous integration patterns to rapidly develop new features. We make use of GitHub actions to automatically lint pull requests and build the code in the cloud to catch any errors before they are merged. We also maintain a protected main branch with feature branches that are regularly merged. We utilize the Nix build system to ensure repeatable, safe, and deterministic builds on members' computers and to avoid versioning issues, by incorporating Nix Flakes into our repos. Next year we hope to move to NixOS on the robot computer as well, creating even more deterministic builds.

ACKNOWLEDGMENT

Palouse RoboSub recognizes the generous support of donors during the team's Fall 2025 CougStarter campaign, as well as continued support from associated family members and relations. Palouse RoboSub also owes gratitude towards their many corporate sponsors, including JoeScan, Tektronix, OSH Park PCBs, Blue Robotics, Real Digital, Moscow Apparel, SolidWorks, VectorNav, WaterLinked, LattePanda, and Alaska Airlines. Next, the authors would like to thank Washington State University's Frank Innovation Zone (FIZ) student makerspace, as well as the Cougar Shop and Kurt Hutchinson. The club would also like to thank our faculty advisor Maynard Siev, former graduate advisor Cody Longwell, college club coordinator Gary Offerdahl, college recruitment coordinator Lisa Carmack, and purchasing coordinator Tammy Kelly for their generous and continuing support of the club. Finally, Palouse RoboSub thanks Crimson Robotics, and WSU 3DPC for sharing shop space, tools, and knowledge, and the Society of Women Engineers and VCEA Club Council for event planning.

REFERENCES

[1] S. Caron, "Conversion from least squares to quadratic programming," *Scaron.info*, Nov. 07, 2023. https://scaron.info/blog/conversion-from-least-squares-to-quadratic-programming.html (accessed May 20, 2026).

APPENDIX A
COMMUNITY OUTREACH AND EXPOSURE

As an engineering club at a major state university, we feel it is very important to inspire future generations of engineers and scientists. We do this through various community and school outreach events, as outlined below:

A. University Recruiting and Expo Events

Palouse RoboSub participates in several campus-wide club fairs and recruiting events, such as the Inside Scoop club fair for freshmen each August. This is where we get most incoming students interested in the club, as well as upperclassmen who stop by. The club also participated in the Voiland College (VCEA) Innovation Expo, in which engineering clubs from across the campus demonstrated their projects to the Pullman community and several high schools. This event also gave VCEA administration and leadership an opportunity to view the clubs and even take a turn at driving Guppy around!

The club also attends high school recruiting events for the engineering college, bringing informational materials and props to presentations around campus. The club's space is also a regular stop on the engineering tours taken each day by multiple families with potential new WSU students. Club members regularly discuss their experiences at WSU and in the club with potential pre-freshman and answer questions.

B. Kids' Science and Engineering Day (KSED)

Fig. 7. A parent driving Guppy around during our KSED Demo Day.

Another large event that the club prepares for every year is the Kids' Science and Engineering Day (KSED), run by the WSU Society of Women Engineers chapter. During KSED, elementary- and preschool-aged children from the neighboring counties participate in fun engineering challenges and experiences throughout the day.

Palouse RoboSub is regularly a big hit at these events, as we allow the students to drive the submarine around as we answer questions about it. The event normally occurs in March and is a fantastic way to reach out to the community and inspire young minds. Many of our members are alumni of FIRST Robotics, where early access to STEM is an important value. In previous years, before Guppy was built, we had a small purely teleoperated submarine named SharkBait that served the same purpose. Sadly, SharkBait has since been harvested for Guppy's thrusters and is no longer operable, although he will be remembered for his years of exemplary service.

APPENDIX B
CONTROL ARCHITECTURE AND MATHEMATICS

Guppy's control code runs in parallel with the ROS code on the LattePanda Sigma computer and is managed by a ROS node that communicates with it. The highly optimized closed loop control uses the Eigen and Proxsuite C++ libraries in a layered PID approach. The control structure is responsible for several important tasks, which are outlined in each section below.

A. Velocity Control

At its very core, the control code subscribes to the commanded velocity topic (`/cmd_vel`) and the current odometry state topic (`/odometry/filtered`) and attempts to match their 6D vectors of velocities. This is done through feedback controllers on each axis's velocity error coupled with feedforward terms to combat drag. As velocity in each degree of freedom (three horizontal axes and three rotational) increases, drag in that axis also increases (approximately) quadratically.

Because of this, we use traditional PID controllers to handle most of the velocity control (typically acceleration or deceleration), and a small feedforward term based on the calculated drag to hold a constant velocity. While the code has this capability, we have found in our testing with Guppy that we can rely primarily on velocity error PID to maintain consistent velocities, as proportional feedback control gives us acceptable tracks. However, the functionality is there, and we intend to tune the controls even further with more testing time in the future.

B. Drag Moments

With many AUVs and ROVs, the center of pressure is not aligned with the center of mass during movement. This means that movements in any linear degree of freedom can additionally cause unwanted torques. In our experience with Guppy, we have not run across this issue in any noticeable way, due to the relatively slow speeds and its large mass. However, as we implement more tuning and additional mechanisms, this will come into play.

To address these unwanted wrenches, the control code incorporates a "drag effect matrix" which is multiplied by the drag vector to adjust the torques and forces accordingly. The columns of the matrix correspond to the axis that is "causing" the unwanted motion, while the rows correspond to where the motion actually occurs. In an idealized scenario, this would be the 6×6 identity matrix. However, if motion forward in the X

direction causes an unwanted yaw (due to an uncentered drag force), we can adjust the drag effect matrix to counter it like so:

$$V_{drag} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0.1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.1 \end{bmatrix}$$

The above equation shows how the large 6 × 6 matrix can affect the drag feedforward term with an added 0.1 in yaw offset to counteract the drag moment. This can easily scale with multiple axes, allowing for very precise tuning of the submarine's motion. Note that all code for Guppy follows the standard ROS axis convention of Z-up and X-forward.

C. Restoring Forces

There are several restoring forces that act on Guppy at any given time, primarily gravity and buoyancy. Guppy is very positively buoyant while unpowered and must hold itself underwater using thrust. To maintain pseudo-neutral buoyancy, we calculate the difference between the gravity and buoyancy forces in the Z-axis, and then transform that vector to the AUV's coordinate frame by multiplying it by the inverse quaternion of the sub's current orientation. We then add this three-dimensional vector to the existing linear feedforward velocity terms. This allows the sub to stay underwater no matter which way it is rotated!

This correction only accounts for "neutral buoyancy" though and does not offset the restoring torques that the buoyancy wrench imparts on the sub as well. To combat this, we take the rotated "center of buoyancy" offset from the center of mass, and cross-product this vector with the unrotated buoyancy force from before. This essentially computes the sub's tendency to right itself, and the inverse can be applied to the controller's outputs to negate this.

D. Station Keeping

When we aren't commanding specific velocities, we need the submarine to hold position unless disabled. This is achieved partially via linear and angular velocity PID, because if the setpoint is zero velocity, the sub will resist motion. However, it will not return to its original position if it does move. We have overcome this by switching to a position based PID controller when the commanded velocity is less than some deadband value. The position value is measured from the fused "odometry" value from the DVL and other sensors like the barometer and cameras. The linear PID is very simple, although the angular feedback can get slightly more complicated.

For angular orientation error, we compute the error quaternion $q_{err} = q^{-1} \otimes q_{desired}$ where q is the current orientation state of the submarine, and decompose it into its imaginary parts x, y, and z. For small angular error, these error components can be directly inputted into roll, pitch, and yaw feedback controllers. Again, we use a deadband value to determine when to apply each axis's feedback, although we must carefully apply them in pairs as rotational motion can be tightly coupled across the axes. This quaternion error approximation breaks down for large angular error, but that state should never occur provided that the initial state is the desired one and that the system is correctly tuned.

Not only does this station keeping allow us to hold our pose, but we also expose a `/reset_holding_pose` service that allows other ROS nodes to change the holding state position. This is used by our navigation actions, which decompose trajectories into a series of staggered waypoints, each of which can be applied to the station keeping code in a timed sequence to follow a path. This is exactly what helped us complete our prequalification run in early May.

E. Thrust Allocation

After all these individual components assemble into one large six-dimensional AUV wrench vector, we need to convert that into an eight-dimensional motor thrust vector. To do so, we first converted all motor positions into a 6 × 8 matrix of motor coefficients. This is done by taking their x, y, and z locations as well as their spherical angles φ and θ and calculating the cartesian coefficients of each motor. Next, we solve the following least squares problem:

$$A\vec{x} = s$$

Where A corresponds to the motor coefficient matrix, $\vec{x}$ to the eight motor thrusts, and s to the desired output wrench. We solve this using least squares, because we need to constrain the motor outputs on their maximum forwards and reversed thrusts. There is also no guaranteed exact solution under saturation (at very high speeds or some combined movements), which means that simply solving this matrix equation will often fail. That makes this a box-constrained least-squares problem.

To optimize the solving of this and implement it in an existing library (Proxsuite), we have converted this problem into a Quadratic Programming (QP) problem instead. Many least-squares minimization problems can be rewritten as QP problems, as noted by S. Caron [1]. Caron describes how to perform this transfer, where we end up with:

$$\min_{\vec{x} \in \mathbb{R}^8} \frac{1}{2} \vec{x}^T H \vec{x} + g^T \vec{x}$$

given that:

$$A\vec{x} = s$$
$$\vec{T}_{lower} \leq \vec{x} \leq \vec{T}_{upper}$$
$$H = A^T W A$$
$$g = -A^T W s$$

…where A and s are still the motor coefficient matrix and desired output wrench. W corresponds to an optional weight matrix to weight one axis more than the other in the solution. The $\vec{T}$ constants refer to the lower and upper thrust bounds of the T200 thrusters (in SI units). In this form, the QP problem is

easily applied directly to Proxsuite, and can take advantage of previous loop states to better inform the next guesses. To prevent "stable" states in which opposing thrusters fire at full force (yet still cancelling out), we add a very small addition where $H = A^T W A + 0.1 I_8$.

APPENDIX C
CUSTOM PCB RENDERINGS AND DISCUSSION

All custom circuit boards were manufactured by our sponsor OSH Park PCBs, and our team assembled the components by hand. The CAD design was done in KiCAD, and all designs are public on our team's GitHub organization.

A. Power Distribution Board (PDB)

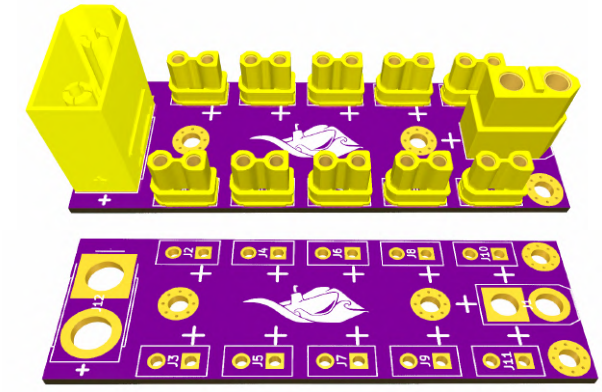

Fig. 8. Renderings of the Power Distribution Board (PDB) in KiCAD.

The power distribution board is made of extra 2 oz copper to handle high current, and consists of several PCB-mount XT connectors of varying size. These boards provide an excellent and neat way to distribute power, as compared to a traditional bus bar. The genders of the XT-60 and -90 connectors can be easily changed to fit the required configuration, as we do in the Compute hull. While the Actuator hull's PDB runs at unregulated 5S battery voltage (16V-21V), the Compute hull's PDB is spliced off of a buck converter, and powers its components with regulated 12V power.

B. RP2350 Mainboard (V2)

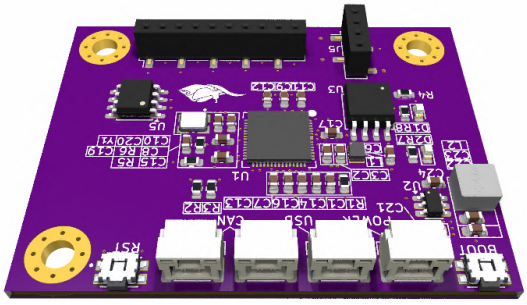

Fig. 9. CAD rendering of version 2 of the RP2350 mainboard.

The main powerhouse of our team's custom PCBs is a mainboard developed around the RP2350 microcontroller. Packaged with flash memory, a CAN controller, and many GPIO and PWM output ports, the board is designed to serve as a sort of "base" board for additional daughterboards or "hats" to stack on top of.

The board features a reset and boot pushbutton, as well as power, CAN in, CAN out, and USB debugging locking JST ports. The two SMD socket headers are for stacking additional boards on top and expose power and IO to the daughterboard. The buttons and JST plugs face sideways, to allow access even when a board is stacked. To access the USB interface, team members can plug a small USB-C adapter board to the USB and POWER plugs. The board also has two red LEDs: one that indicates power, and another that shows activity via a GPIO pin.

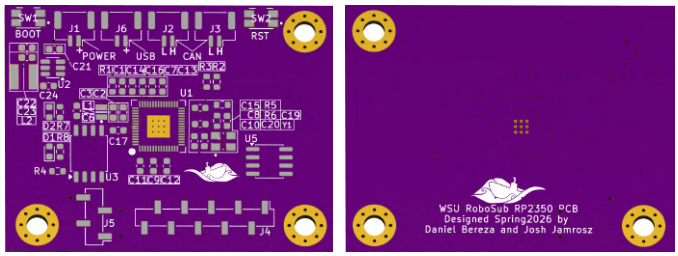

Fig. 10. Top and reverse views of the RP2350 board's silkscreen.

We chose to use the RP2350 microprocessor rather than an STM or ESP based one for several reasons. First, STM processors are notoriously difficult to program and manage, and we liked the configurability and reputation offered by the RP2350 controller. We determined that ESP boards would be overkill for our use case, given that they have WiFi and Bluetooth capabilities. Finally, the PIO capabilities of the RP chip coupled with unparalleled PWM channels made the RP2350 chip a clear choice.

As with all of our custom PCBs, the mounting holes are spaced on a 10mm grid pattern to accommodate our electrical shelving.

C. PWM Daughterboard

The PWM daughterboard is currently our only daughterboard designed for the RP board, although several more are in the works for next season. The daughterboard simply exposes 10 PWM channels as ground/signal pairs via locking JST GH connectors. We use one of these in our Actuator hull to drive the 8 BLHeli_S ESCs for the T200 thrusters, as well as the disable/enable reed switch and the tube's LEDs.

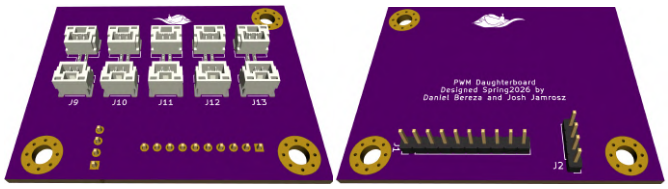

Fig. 11. Top and reverse views of the PWM daughterboard in CAD.

The board neatly stacks on the base board, allowing a compact layout and decreasing the number of wires. The board also acts as a ground bus, letting each JST connector share a common ground without the need for a separate ground splice to each motor controller.

D. Version 1: Original RP2350 Development Mainboard

Before its current "V2" designation, we manufactured an additional development board of the RP2350 PCB, the "V1." Renderings are shown here for posterity. While the overall design stayed the same, we changed the layout to accommodate daughterboards, added mounting holes, and decreased the regulator size.

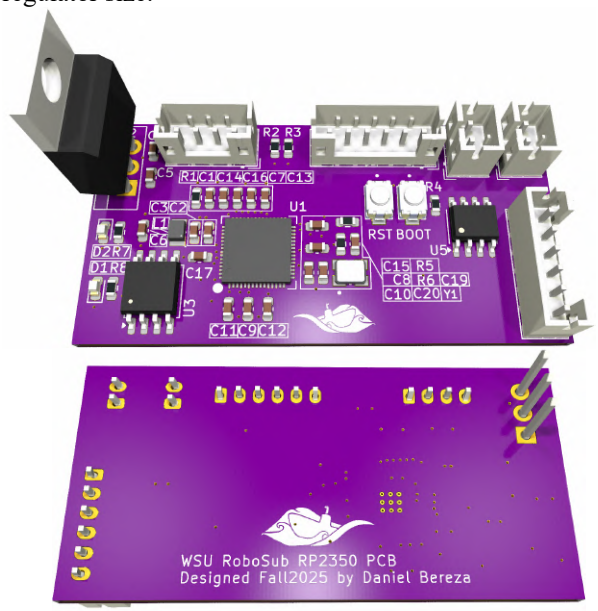

Fig. 12. Top and reverse views of the original RP dev board in CAD.

E. Anti-Spark Switch Control ESP

Our FlipSky anti-spark relay switch, which controls the main power input to the submarine, comes with a momentary button switch. While this is great for skateboards and bikes, we prefer the Blue Robotics rotary switches. We use a small ESP powered off a buck converter directly from the battery to read the state of the external switch and act as a proxy to send signal pulses to the main anti-spark switch. While this is not a custom PCB, we have plans to make it one in the future and extend our power capabilities. For more details on how this works, see Appendix G for additional schematics.

APPENDIX D
LED PATTERNS AND STATES

State	LED Pattern	Description
STARTUP	Chasing White	Booting up.
DISABLED	Solid Green	No actuation.
TELEOP	Flashing Green	Human driver.
HOLDING	Solid Red	Neutral Z-axis.
NAV	Flashing Red	Navigation.
TASK	Flashing Red	Task code.
FAULT	Flashing Red/Blue	Error/Danger.

Table 2. Possible states, descriptions, and LED color patterns.

APPENDIX E
GUPPY RENDERINGS AND PHOTOS

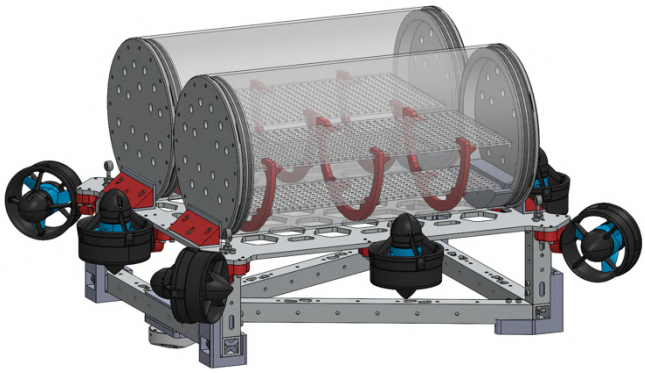

Fig. 13. A CAD rendering of Guppy in SolidWorks.

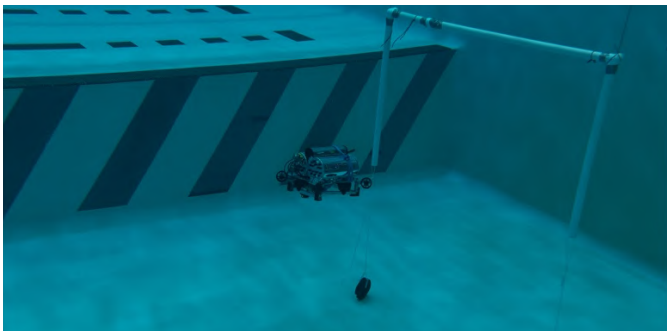

Fig. 14. Guppy completing prequalification in Gibb Pool at WSU!

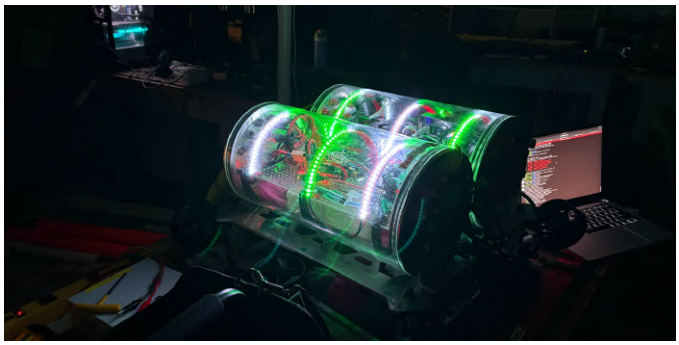

Fig. 15. Testing of Guppy's LEDs and associated code in a dark shop.

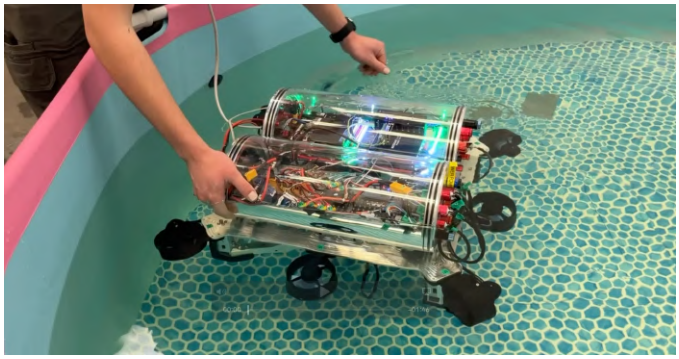

Fig. 16. Guppy's first time in the water while being powered.

APPENDIX F
SOFTWARE ARCHITECTURE AND DIAGRAMS

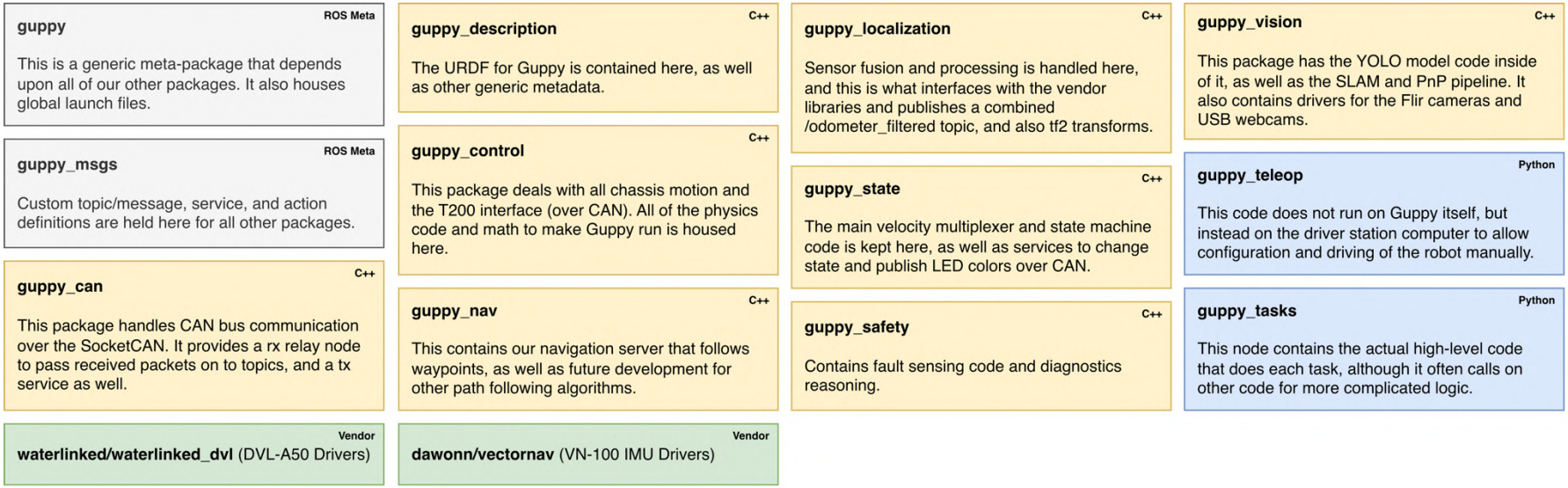

Fig. 17. ROS package organization.

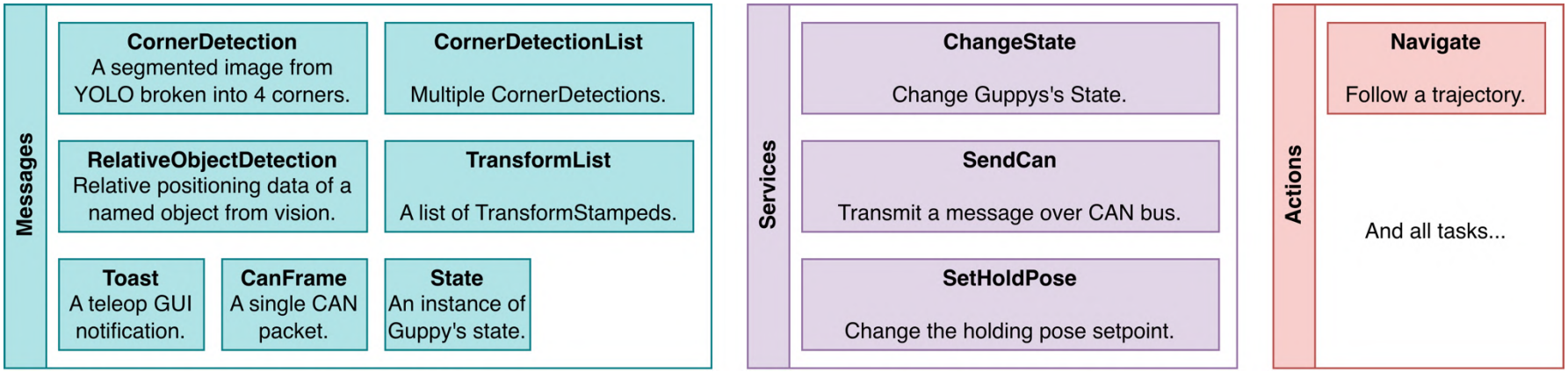

Fig. 18. ROS custom message/service/action definitions.

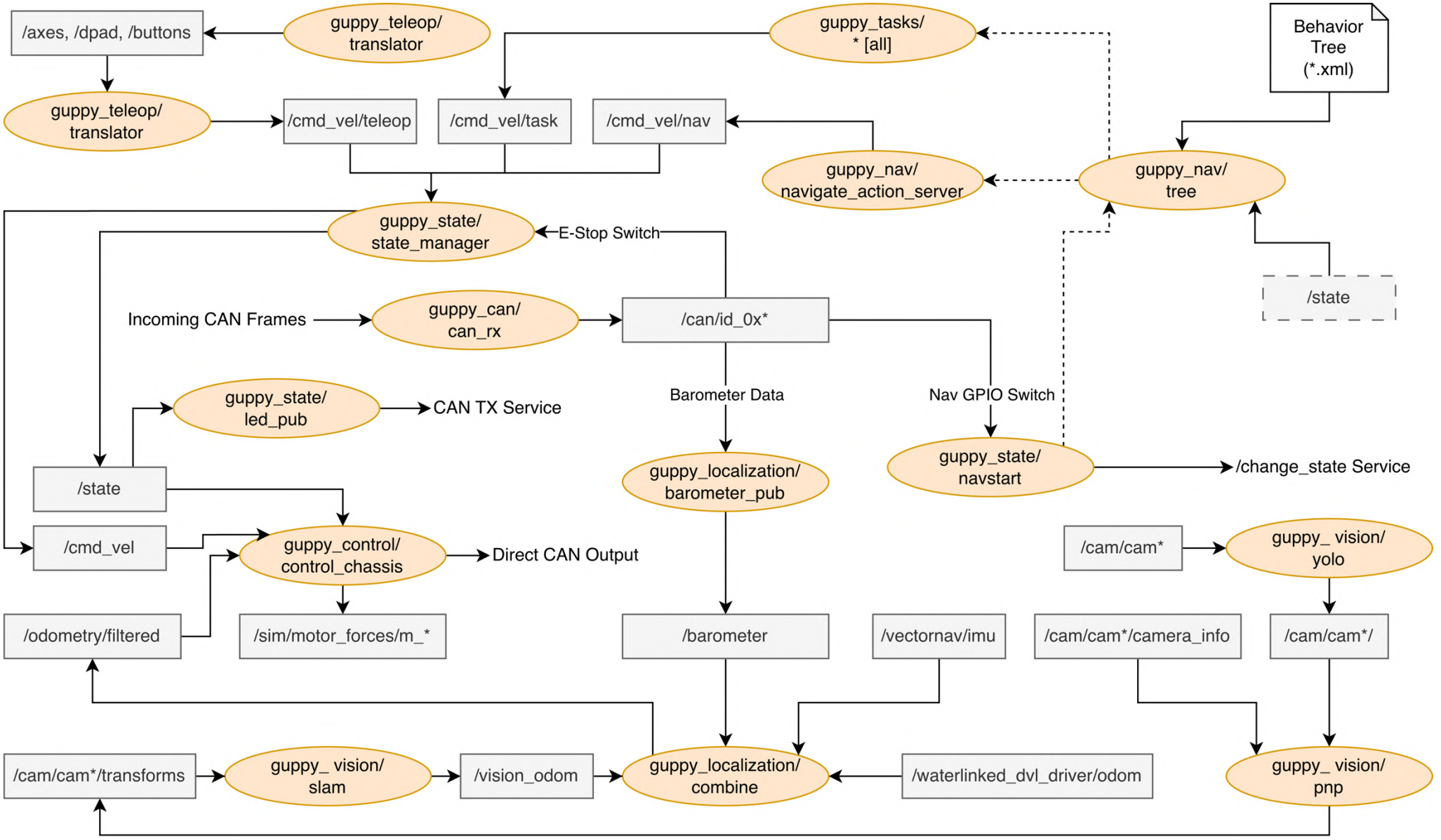

Fig. 19. ROS node graph of Guppy's code.

APPENDIX G
ELECTRICAL ARCHITECTURE AND SCHEMATICS

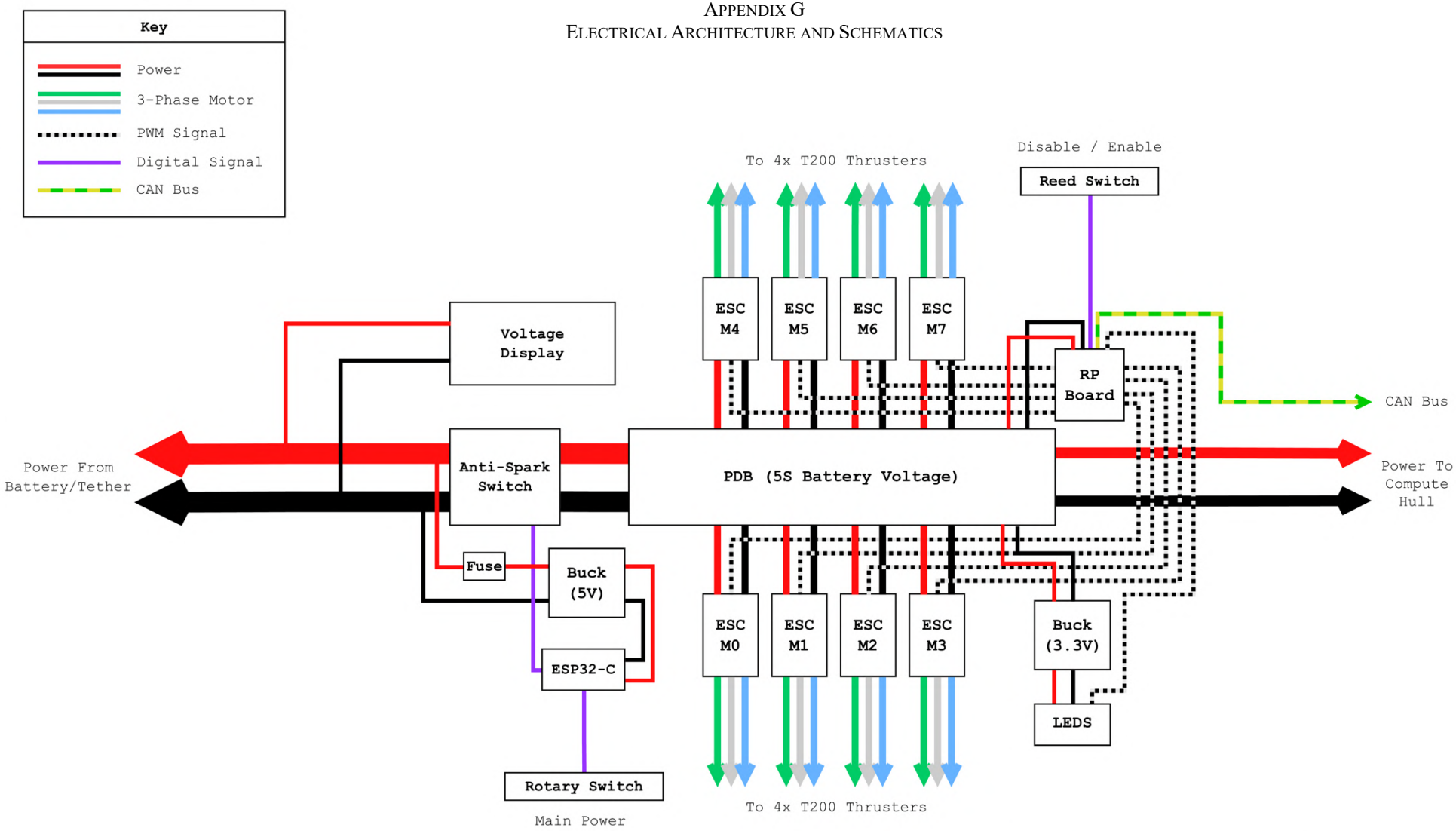

Fig. 20. Actuator hull wiring diagram.

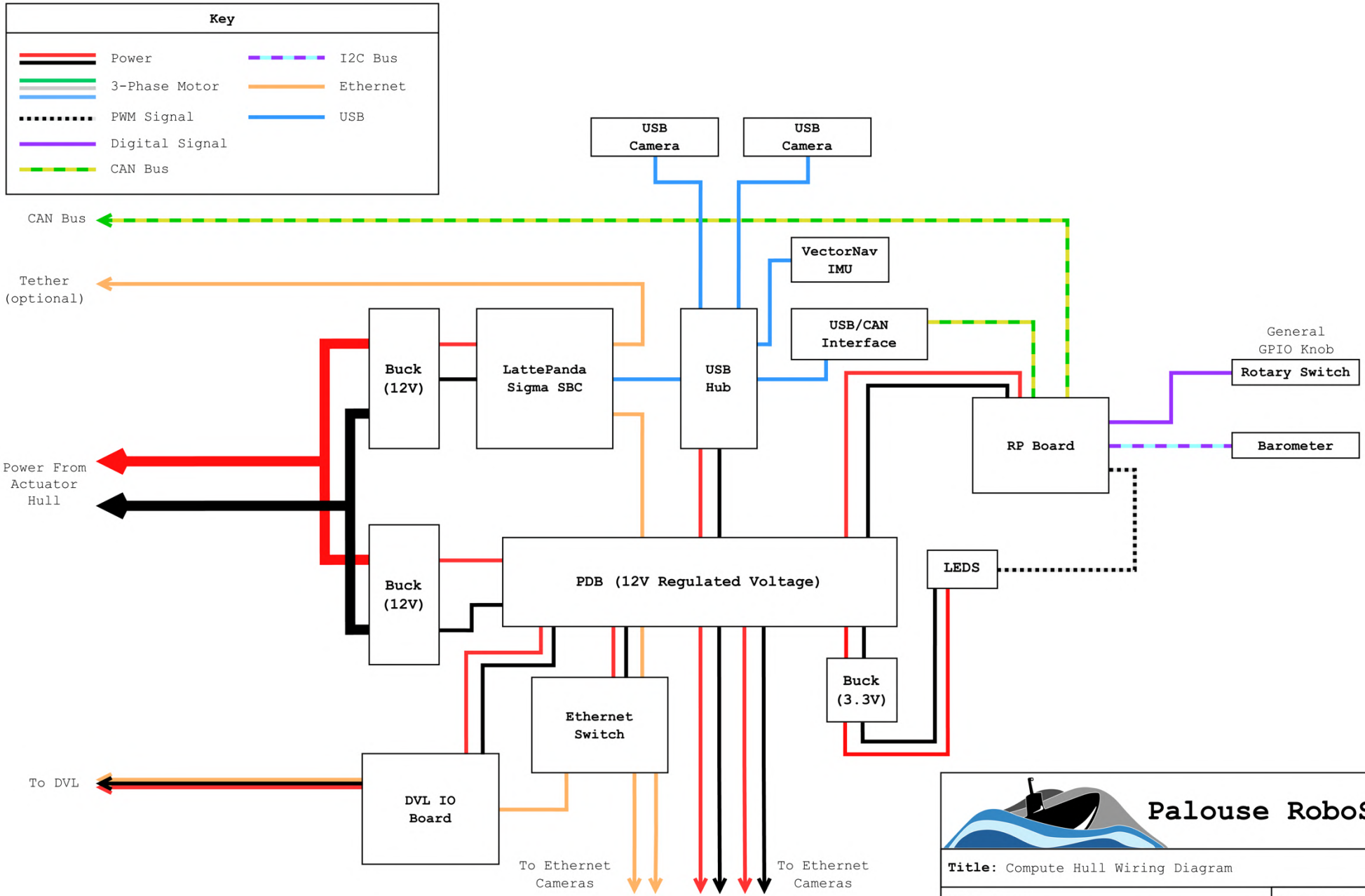

Fig. 21. Compute hull wiring diagram.

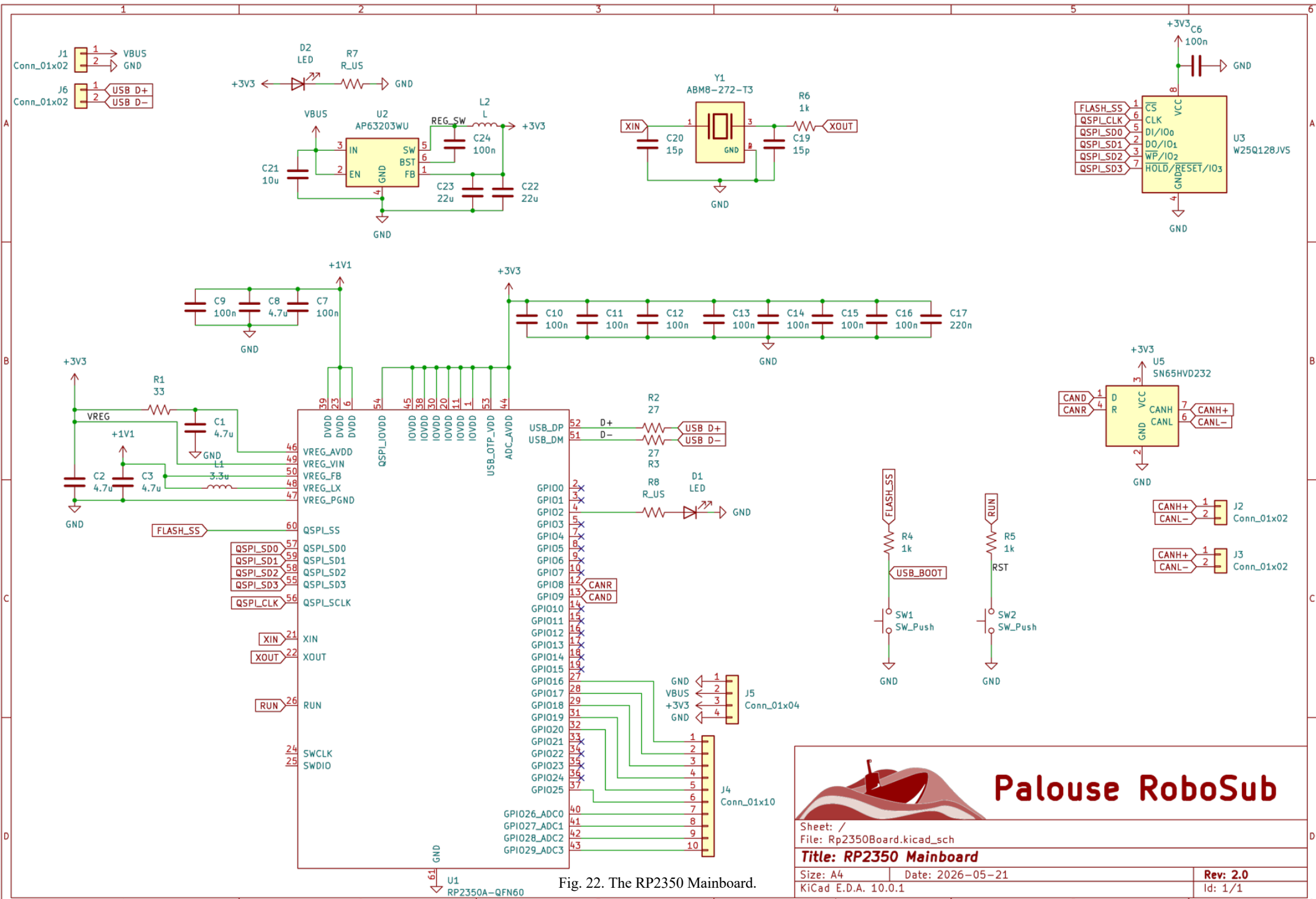

Fig. 22. The RP2350 Mainboard.

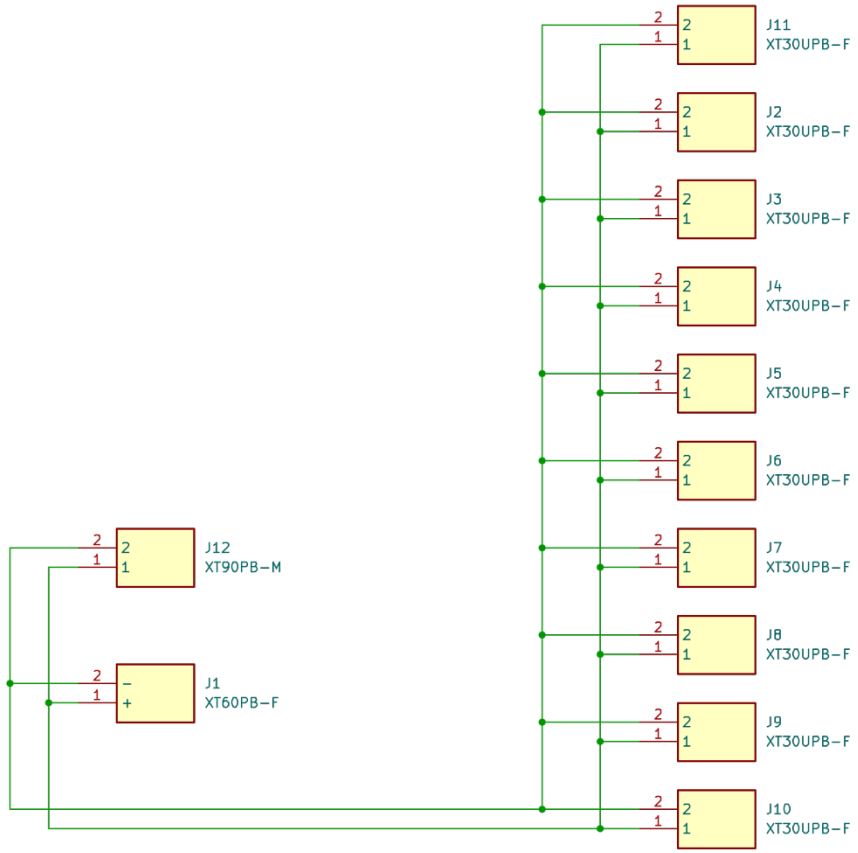

Fig. 23. The PDB board.

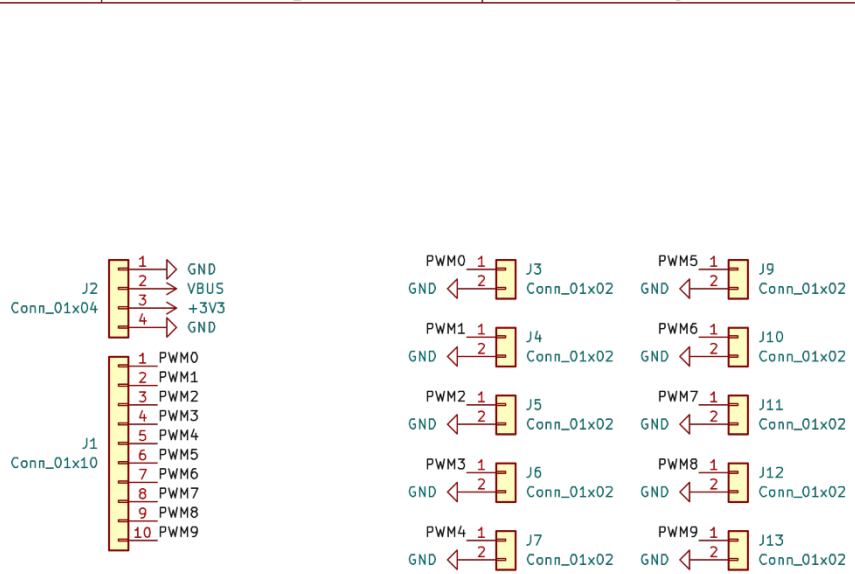

Fig. 24. The PWM daughterboard.

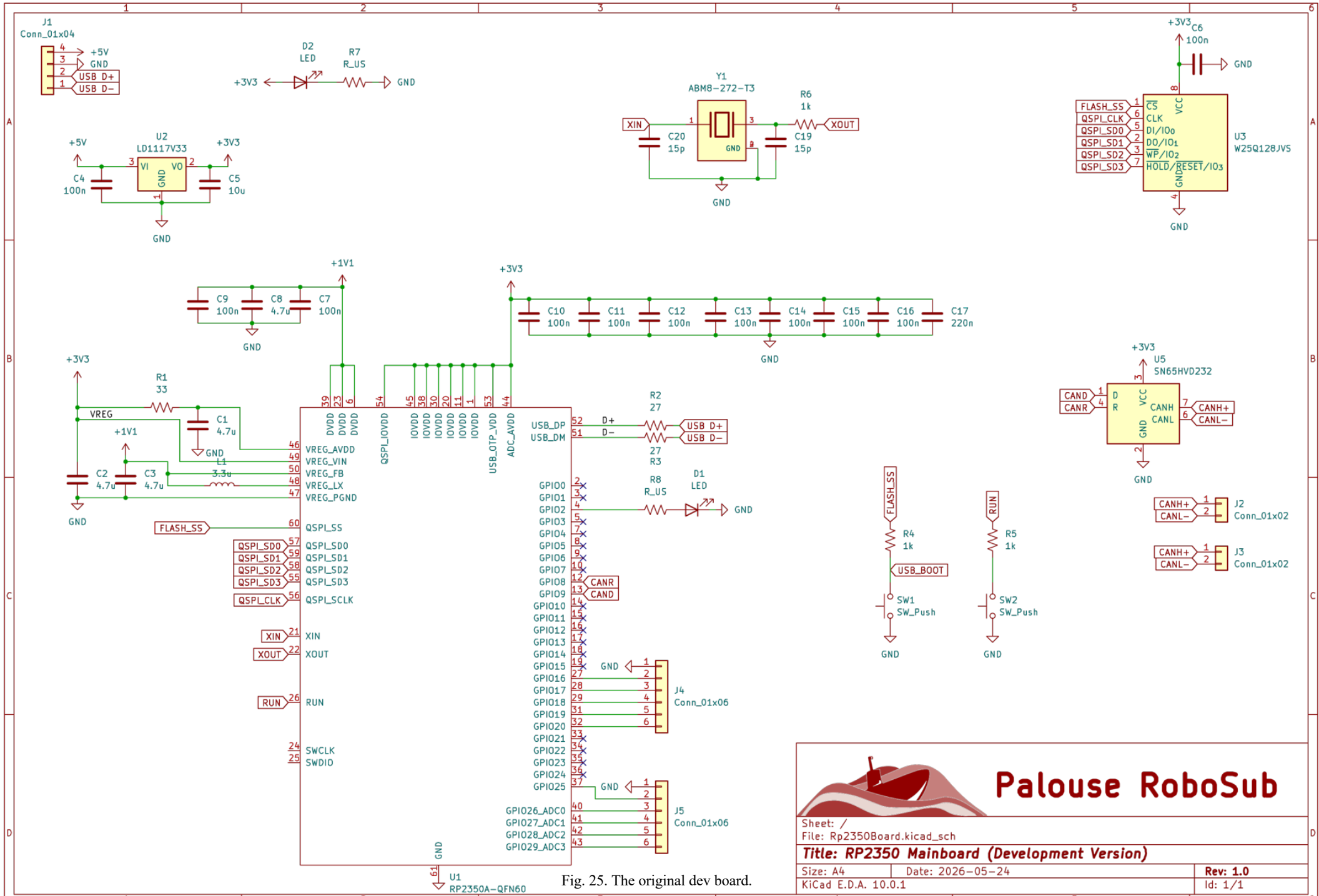

Fig. 25. The original dev board.

APPENDIX H
MECHANICAL ARCHITECTURE AND DRAWINGS

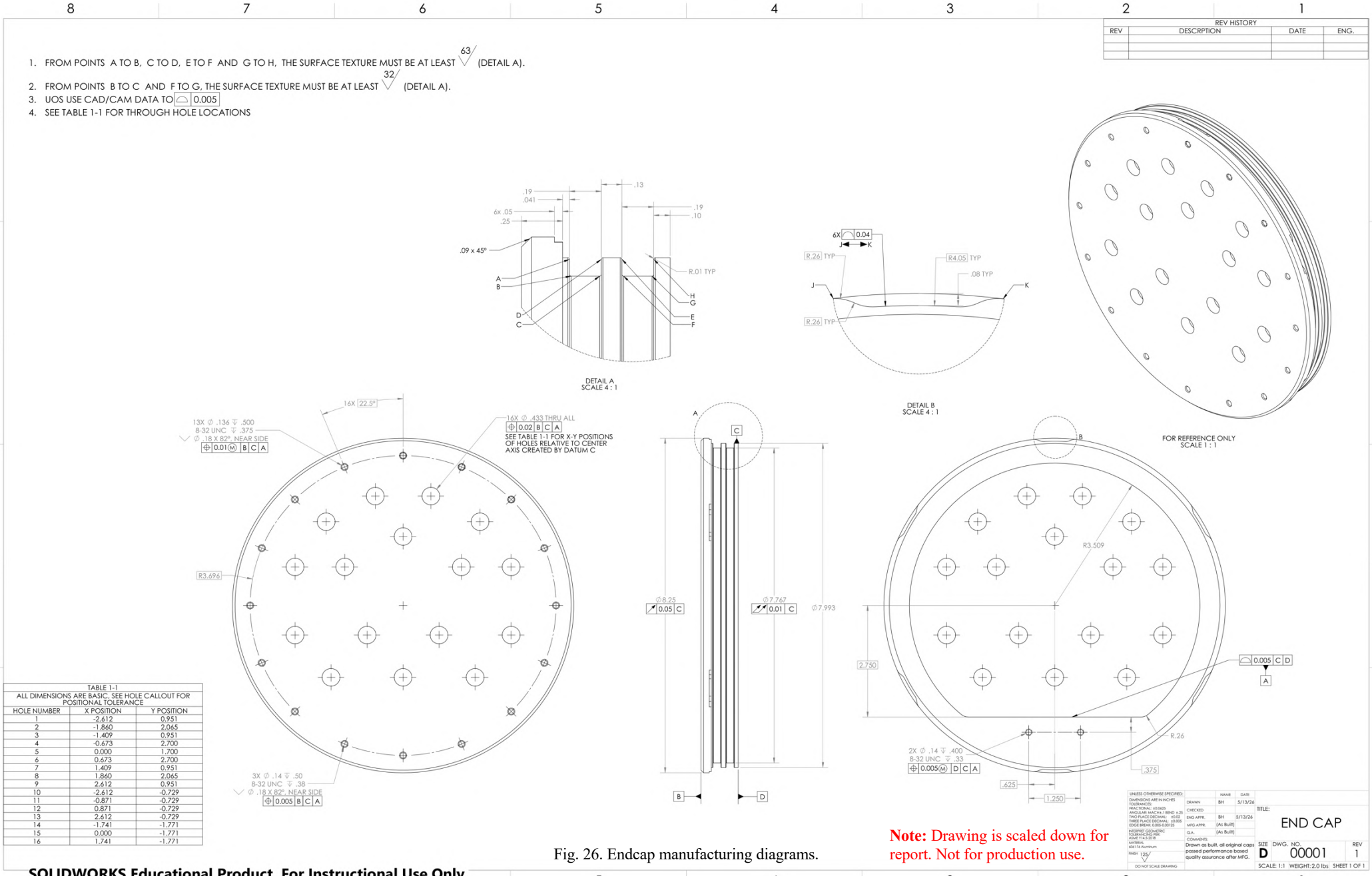

TABLE 1-1
ALL DIMENSIONS ARE BASIC, SEE HOLE CALLOUT FOR POSITIONAL TOLERANCE

HOLE NUMBER	X POSITION	Y POSITION
1	-2.612	0.951
2	-1.860	2.065
3	-1.409	0.951
4	-0.673	2.700
5	0.000	1.700
6	0.673	2.700
7	1.409	0.951
8	1.860	2.065
9	2.612	0.951
10	-2.612	-0.729
11	-0.871	-0.729
12	0.871	-0.729
13	2.612	-0.729
14	-1.741	-1.771
15	0.000	-1.771
16	1.741	-1.771

Note: Drawing is scaled down for report. Not for production use.

Fig. 26. Endcap manufacturing diagrams.

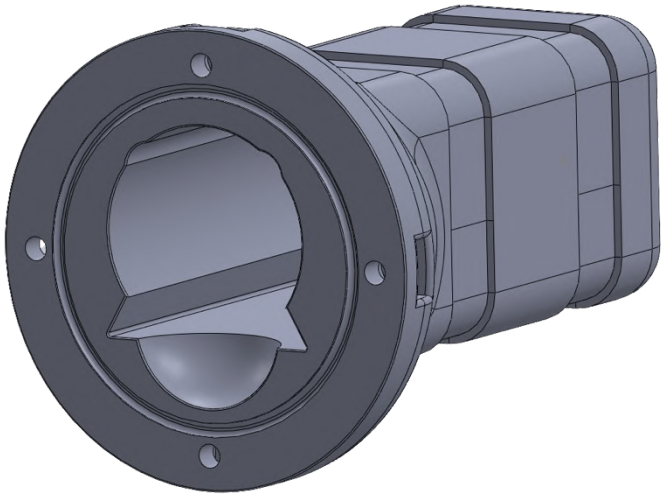

Fig. 27. Resin waterproof enclosures for the Flir cameras.

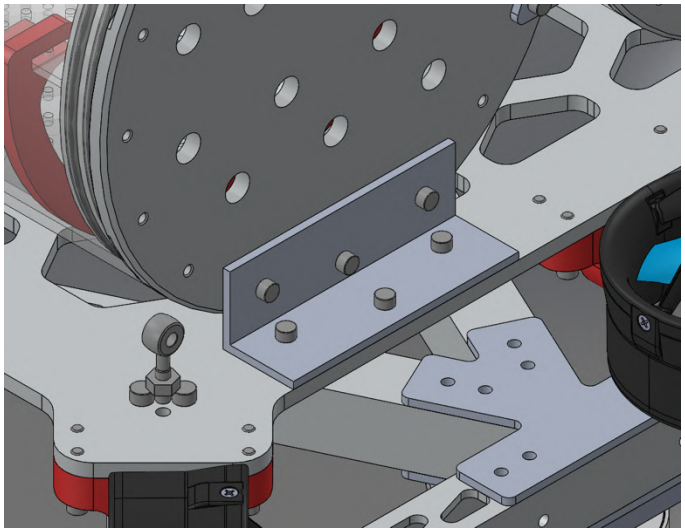

Fig. 28. CAD of the endcap/baseplate mounting interface.

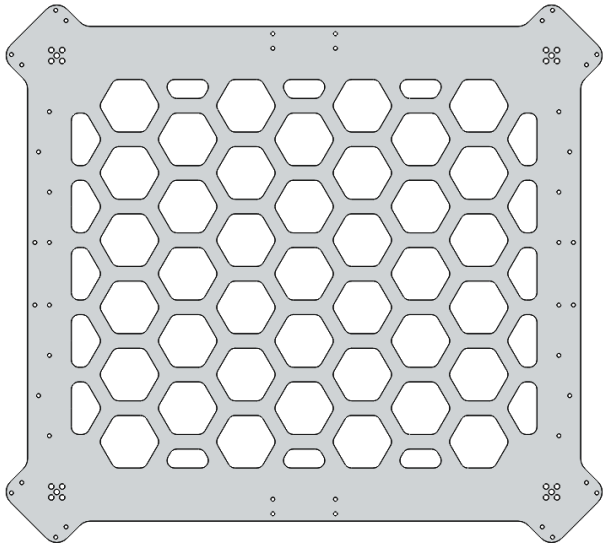

Fig. 29. CAD overhead view of the baseplate.

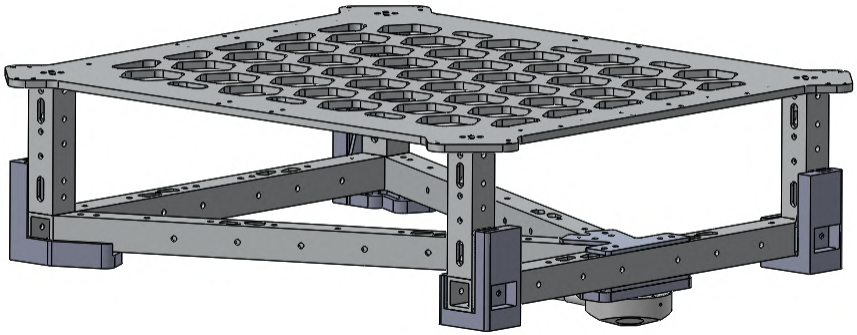

Fig. 30. Guppy's undercarriage in CAD.

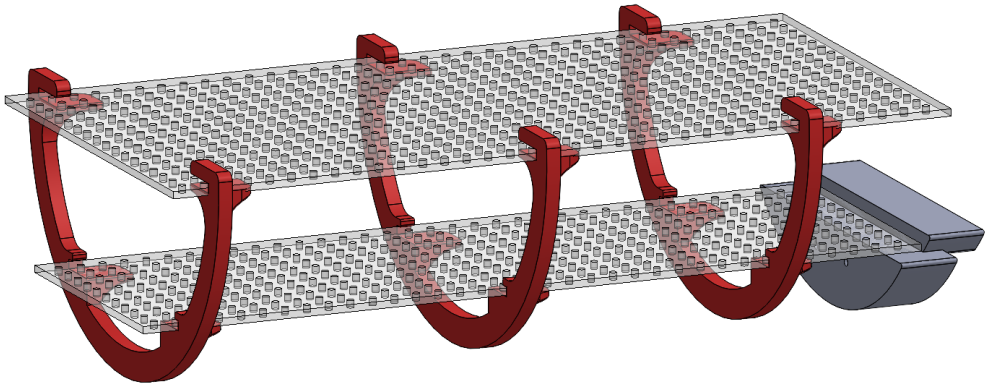

Fig. 31. Guppy's internal electrical shelving.

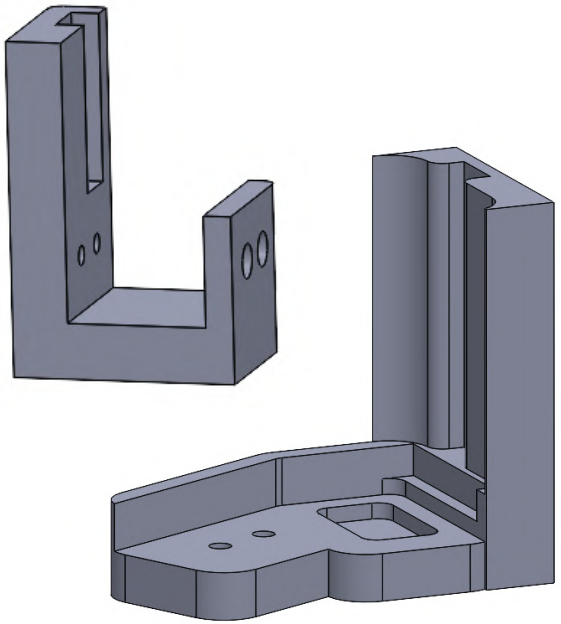

Fig. 32. Guppy's "feet" or protective boots.

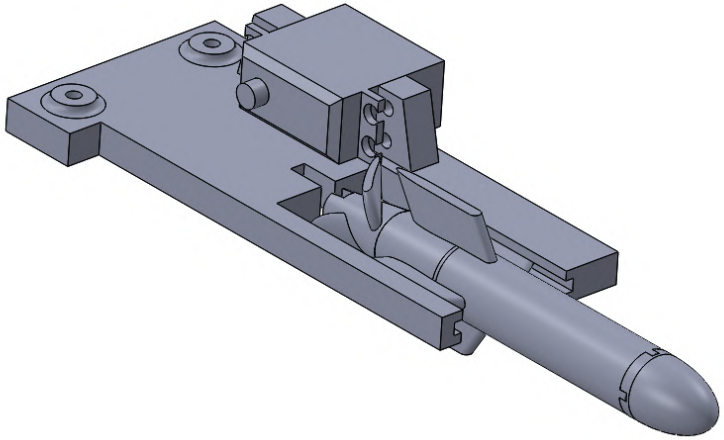

Fig. 33. The prototype torpedo assembly in CAD.

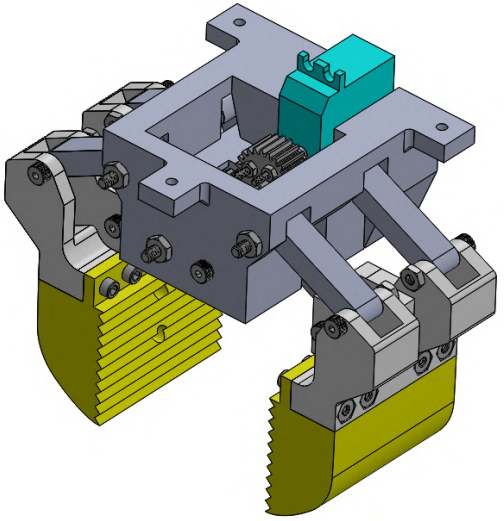

Fig. 34. The prototype arm assembly in CAD.

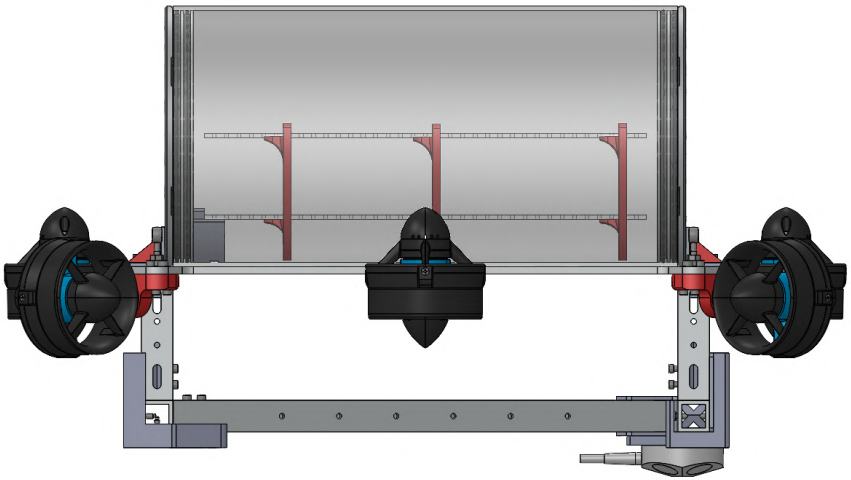

Fig. 35. Guppy CAD side view.

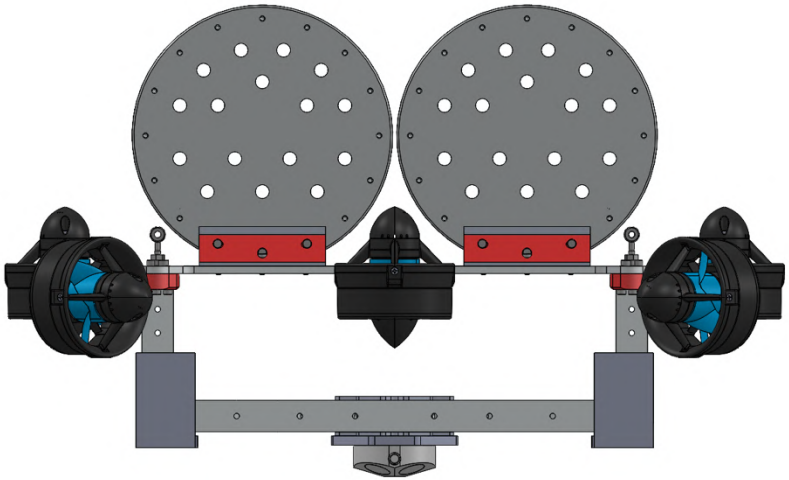

Fig. 36. Guppy CAD front view.

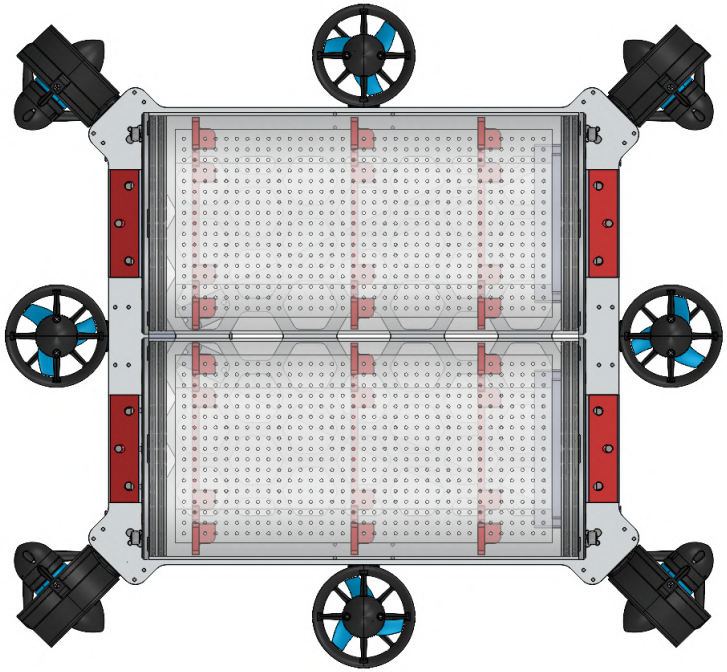

Fig. 37. Guppy CAD top view.

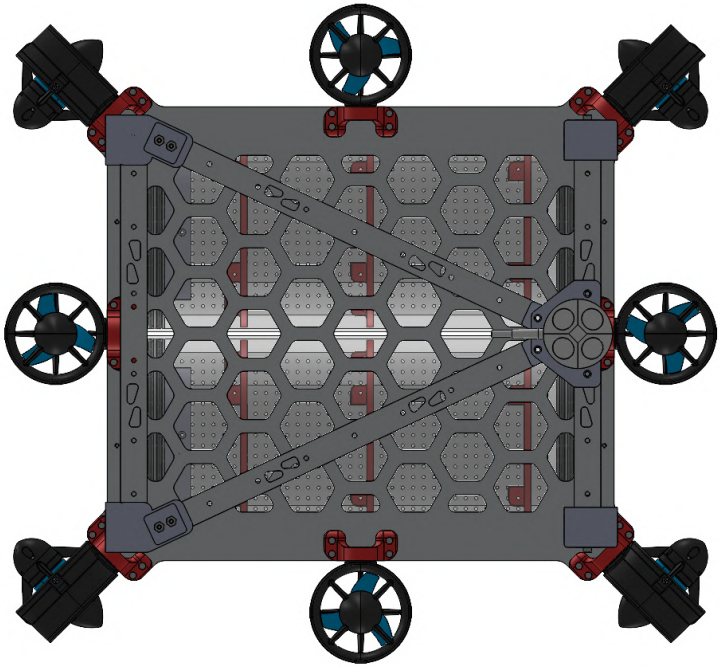

Fig. 38. Guppy CAD bottom view.

APPENDIX I
COMPONENT SPECIFICATIONS

Component	Vendor	Model	Specs	Custom / Purchased	Cost	Date Purchased
Frame	Online Metals	6061 Aluminum		Custom	$600	2025
Hulls	Blue Robotics	Acrylic Tubing	8” Inner Diameter	Custom	Legacy	Legacy
Waterproof Connectors	Blue Robotics	M10 Penetrator		Purchased	$5 ea.	2026
Thrusters	Blue Robotics	T200		Purchased	$230 ea.	2025
Motor Controllers	ReadyToSky	35A Micro Electronic Speed Controller	BLHeli_S Firmware, 35A	Purchased	$40 ea.	2026
High Level Control	Raspberry Pi	RP2350	Custom RP2350 Board with PWM output connectors	Custom	$23 ea.	2026
Actuators	Hobby Fans	RC Digital Servo	40kg Torque, Metal gear	Purchased	$25 ea.	2026
Battery	MaxAmps	22Ah 5S LiPo		Purchased	Legacy	Legacy
Convertor & Regulator	Nilight	24V to 12V Converter + 10A Regulator		Purchased	$15	2026
CPU	LattePanda	Sigma SBC	Intel Core i5-1340P, 32GB DDR5	Purchased	$560	2025
Internal Network	CAN Bus	CAN 2.0A	500kbit/s, 11-bit identifiers	Custom		2026
External Comm Interface	Blue Robotics	Fathom ROV Ethernet Tether	Neutrally Buoyant, 4 twisted pair, RJ45	Purchased	Legacy	Legacy
Compass & IMU	VectorNav	VN-100		Purchased	Sponsored	2025
DVL	WaterLinked	DVL-A50	Standard, 300m rated	Purchased	$4570	2025
Manipulator		Custom Design	CoPA Nylon Printed	Custom		2026
Algorithms: Vision	OpenCV, YOLO					
Algorithms: Localization & Mapping	PnP, SLAM					
Algorithms: Autonomy	BehaviorTree.cpp					
Open-Source Software	Ubuntu Noble, Nix, ROS2 Jazzy, KiCAD					
Programming Languages	C++, Python					

Table 3. Component specifications, prices, and vendors for the 2026 AUV Guppy.