RoboBoat Technical Report

Team Name Beaver AUV	**School / Organization** Beaver Country Day School
Competition Year 2026	**Boat Name** YellowFin

Abstract

Yellowfin is a fully autonomous surface vehicle designed to compete in RoboBoat 2026 as a first-year team operating under a short three-month build window. The team's competition strategy prioritizes qualification through reliable completion of the required start gate and basic navigation tasks before attempting higher complexity challenges. This strategy directly shaped the system architecture, which emphasizes stability, simple mechanical construction, and a robust autonomy stack built around proven sensing and control methods. Mechanically, the vehicle uses two sealed PVC pontoons and a modular aluminum frame to provide stability and rapid assembly. Electrically, the system is built around a sealed battery system under 60 V, fused power distribution, and four Blue Robotics T200 thrusters controlled through independent ESCs to provide redundancy and safe operation. The autonomy system integrates GPS, IMU, and camera sensing into a unified state estimation and path planning framework implemented through the team's software library. By focusing on reliable sensing, repeatable control, and fail-safe shutdown behavior, the team aims to demonstrate consistent straight-line navigation, gate traversal, and multi-task operation required for qualification while preserving a platform that can be expanded for future challenges.

Introduction

The Beaver RoboBoat team entered RoboBoat 2026 with the goal of qualifying as a first year team operating under a three month development window. The team's mission was to design and field a vehicle that could reliably complete the required entry and exit gates and at least two additional autonomy tasks, rather than attempting a wide set of complex behaviors with low reliability. This philosophy led to a design approach that emphasized simplicity, robustness, and rapid integration over mechanical complexity.

The team selected the navigation channel and speed challenge tasks as secondary objectives because they build directly on the same sensing, control, and propulsion capabilities required for gate traversal. These tasks require accurate heading control, stable propulsion, and basic waypoint navigation but do not demand specialized mechanical systems such as manipulators or actuated payloads. By choosing tasks that share common system requirements, the team was able to concentrate development and testing on a smaller set of critical subsystems.

A major constraint throughout the design process was the short timeframe between team formation and the competition. With limited access to manufacturing resources and little opportunity for mechanical iteration, the vehicle was designed around components that could be assembled quickly and tested early. This constraint favored a modular aluminum frame, off the shelf thrusters, and 3D printed mounts that could be redesigned rapidly if failures occurred.

The overall approach was to shift complexity away from the mechanical system and into the software stack, where the team had prior experience from RoboSub. By relying on software based perception, state estimation, and control, the team was able to implement task behaviors such as gate detection and channel following without adding mechanical mechanisms that would increase risk. This strategy allowed Yellowfin to remain mechanically simple while still supporting the autonomy capabilities required to attempt qualification at RoboBoat 2026.

Mechanical Design

The central electronics enclosure is a waterproof case that was selected for accessibility reasons and its lightweight design. The team has made modifications to the enclosure by adding wet-link hull penetrators for the motors and adding the kill switch button. Around the electronics enclosure sits the main frame, which is made out of 40x40mm T-slotted aluminum. In an effort to make the boat mostly in house, many of the attachment methods are through 3D printing. The electronics enclosure is attached to the aluminum frame through custom-made mounts as are the motors. Two 3 ft long 3.5 in diameter PVC pipes act as our main pontoon floats, designed in order to provide more stability in rough water than a flat-bottomed boat would, which was the original design.The boat has four Blue robotics T-200 motors which are arranged in a diamond configuration on the two pontoons. This motor setup was the same setup on the team's RoboSub submarine, and the experience the team already had with this configuration made it an easy decision to continue with the same motor placement. Additionally, the diamond configuration provides a wide range of movement including forwards, backwards, side to side, and diagonally, and can turn without the need for a rudder. The two PVC pontoon floats are sealed with PVC glue and marine epoxy (Check when we seal them). The electronics enclosure box has a gasket itself, and every other hull penetration is sealed with an O-ring, including the four wet-link connectors and the kill switch button. The team faced a tight timeframe this year, and because of that, ultimately decided to make a boat which gave the team the best chance to compete, even if it meant that our final product was not as polished as we wanted it to be. In light of that, we opted for PVC pipes over custom fiberglass floats because the team wanted to ensure that the boat would be ready in time to make it into the water. The team still has plans to incorporate the fiberglass floats and an overall more polished design next year, but due to the limited time, the team had to make the tradeoff.

Electrical system

The power system operates with two separate circuits, with one circuit powering the computer systems and sensors, and another one powering the motors and ESCs. The boat has multiple voltage regulators protecting key components, and has fuses after each battery which protect the electronic components.

The computer system is made up of a Raspberry Pi and an Arduino on a custom board, which are able to control the boat autonomously. The Pi is the main processor, and the Arduino acts as the motor controller and is connected to a relay setup. The boat can also be remotely controlled through and RC receiver, which is also connected to the relay setup. The relay setup can be triggered through the receiver to shift control between the Arduino and the receiver, meaning that the boat can be both remotely and autonomously controlled. The receiver is also linked to the kill switch, which is a large relay connected to a button, featuring a button for the physical kill.

Software and controls

```
def velocity(self):
        heading, forward = self.state()
        self.error = self.map.angle_to(heading)
        heading_signal = self.heading_pid.signal((self.error + np.pi) % (2 *
np.pi) - np.pi)
        return self.final_velocity(heading_signal, forward)
```

As the team began with a sub and transitioned to a boat, and as the team does not have pool access for testing, a major goal was to make sure the code can be portable between many hardware surfaces. The various surfaces available include the boat and sub, a simulation for simple testing, and a hovercraft for advanced testing. To make the code transferable between many surfaces, the majority of the code is published on PyPi under the name EZAUV. This will allow for our code to be used by other teams, especially beginning teams who do not yet have a mature codebase. The core principle of the library is to abstract as much of the hardware away from the user as possible throughout the code, such that the user only has to provide base hardware interfaces in the initialization of the program and from then on can interact with the sub through only high-level functions. Inertia builder: To simplify the process of creating the needed inertia data, EZAUV provides an InertiaBuilder class. This class takes in a set of InertiaGeometry objects, and computes a total moment of inertia tensor. InertiaGeometry is an abstract class with some built-in helper methods to assist in creating new types of geometries, and EZAUV also provides a few pre-made subclasses: Sphere, HollowCylinder, and Cuboid. The vehicle can be 'modeled' out of these geometries, and the final tensor will be computed automatically. One of the most important parts of the code is the motor controller, which solves for the needed motor PWM signals to reach a given total translational and rotational acceleration vector. To solve for the speed given, the program solves a mixed-integer quadratic programming (MIQP) problem. The optimizer takes into account maximum and minimum thrust, motor deadzones, and nonlinearities in the PWM-to-thrust curve. If the wanted speed is infeasible, it can either solve for the nearest feasible speed or solve for the nearest feasible multiple of that speed (either adding a vector epsilon term, or multiplying by a scalar epsilon term). This abstract motor controller allows a single acceleration vector to give the same results across any vehicle.

To localize itself and build a map of its environment, the boat combines the data received from the IMU, GPS, and camera. EZAUV uses a Kalman filter to predict state, and the camera data is used to build a map with circular markers which describe buoys and goals. To create a path, the library initially takes in a set of simple goals (e.g. go to (10,10) and avoid buoys, then come back) which can each be described as one or multiple goal locations. EZAUV then discretizes the map into a grid, expands the obstacles by the size of the vehicle to avoid collisions, and runs Dijkstra's algorithm to find an optimal path to the goal. Once the path is created, the vehicle follows a pure pursuit algorithm, constantly chasing a lookahead point at a set distance on the path. Once the vehicle gets close enough to the goal it switches to a stopping algorithm. It prefers to move straight forward, but can be allowed through function options to also reverse (which can be useful if it overshoots) or strafe (which can prevent issues from drifting off the path).

By default, the motor controller takes in a local space acceleration vector, but if rotation is provided in the sensor data EZAUV also accepts global space acceleration vectors. When providing velocity or location rather than direct acceleration vectors, EZAUV automatically creates PIDs on velocity, and optionally on heading depending on the settings used. Wanted velocity vectors can also be provided in either global or local space, but local space velocities are significantly more stable and it is generally recommended to manually convert global spac velocities to local before using them.

Perception and vision systems

We are using an ExploreHD waterproof camera for buoy detection. The code takes an image about every 1/30th of a second and runes processing on the image to detect the buoys. Once the buoys are found the computer then crops the image to the size of the buoy and then repeats the process for every buoy detected in the image. Once this is finished, the cropped images are then run though openCV to find the average color of the buoy and find out what type of buoy it is. Now that the boat knows this, it calculates how far the buoy is from the GPS's general location using the camera (this can be done because we know the dimensions of the buoys), and then marks it on the map. For image recognition we used YOLOv11 because it was the fastest and latest YOLO model available to us (excluding version 12, however it specializes in other tasks removing the background around the recognized objects, therefore 11 was a better choice when prioritizing speed). Specifically we settled on the small YOLO model, working up from the nano version because it was a good tradeoff between speed and performance. We found that the nano version had trouble distinguishing people's faces from buoys. For our dataset we sourced many images particularly from roboflow and COCO for images of things that were not buoys. Our final dataset was over 18,000 images with about a 3:1 ratio of images of buoys and images without buoys. The most challenging part of this model was finding the data. Many datasets were available however many seemed quite poor for our purpose.

All of our data is processed locally in realtime on the onboard Raspberry Pi 5 computer that gets data from out camera and feeds it into the YOLO model which is running through the artificial intelligence NPU (neural processing unit, Hailo 8L, 26 TOPS) that is connected to the Raspberry Pi 5 via PCIe. From there results are returned from the NPU back to Raspberry Pi and then the average color is found using openCV. We find the location of the object using the formula $F = (P \times D) / W$ where F is the focal length, P is the width in pixels, D is the target number we solve for (distance), and W is the width of the object. We can then use $\tan^{-1}$ (or arctan) to find the angle relative to the camera ($A = \tan^{-1}(\text{Size} / \text{Distance})$).

The system has a couple of limitations. First off, we are limited by the somewhat weak NPU that we are working with that only enables us to run small models with small image sizes (currently 640x320). If we had a faster NPU then we would be able to predict smaller buoys and see more. Additionally, our math is not optimal and we would be better off if we had a second camera. This would provide us with much higher accuracy and enable better pathfinding. Our GPSs limits our accuracy too. We currently have two GPSs and to reduce noise error we average the location of our GPSs to find the median location and therefore reduce the amount of error in our location. However we do not have a GPS equipped with RTK that would allow us to have a much more accurate location.

Testing and validation

The team custom made a hovercraft that acts as a test bed for the boat. The hovercraft is designed to mimic movements of the boat, allowing for us to test without the need for a pool. To test the software, the library EZAUV includes a 2-dimensional simulation. The simulation models acceleration, velocity, and position, as well as random error (e.g. wind, etc.), measurement error, a simplified drag model, and motor deadzones and nonlinearities. We used our hover craft to understand performance metrics and results. The simulation was effective in finding many bugs/issues, especially when those issues wouldn't have appeared in simple automatic bug checks. It had some issues when the simulation failed to model reality; in the previous competition, the code did not account for motor nonlinearities, and as it wasn't considered in the simulation it wasn't discovered till arriving at the competition. Nearly all bugs found were found by testing in the simulation. The simulation allowed for the team to find errors in the code, making it possible for the code system to be refined. Additionally, the hovercraft allowed the team to refine motor placement and redesign the computer stack. However, because it is the team's first year competing at RoboBoat, more improvements based on competition performance will come next year.

Safety considerations

For our Electrical safety systems fuses are placed in the battery circuits to protect the important electronic components in Yellowfin. The fuses ensure that excessive current can not damage the boat's computer stack or motor controls. In addition to the fuses, the relays which switch between RC and autonomous control default to RC, meaning that in the case of any type of failure, Yellowfin is defaulted to remote control. Yellowfin is positively buoyant and weighs well under 140 pounds. It also has both the required kill systems. The boat has a button that triggers a large relay, shutting down power to the motors.

Project management

Teams are split into Mechanical, Electrical, Software, and Business and Operations. Software meets weekly on Mondays. Electrical meets weekly on Wednesdays during office hours. Business and

Operations also meets weekly, with subteam checkins throughout the week. Leadership syncs happen weekly for updates and cross team alignment. As previously mentioned, short notice of the team attending RoboBoat meant that work on Yellowfin only started in early November. The team started by taking many systems from the old sub and integrating them into the new electronics enclosure. Afterwards, the frame of the boat was developed and the two pontoons were attached to make the full boat. At the same time, the GPS and obstacle avoidance systems were in development, as well as the RC to autonomy system.

We are sponsored internally by donors committed to strengthening our stem and robotics programming. Through the team's generous donors, all travel is covered. The team is also given a budget of $2500 for teams to allocate towards equipment or mechanizing as they see fit. Students submit requests and they are reviewed by a board of Business and Operations team members and faculty advisors before being approved. Team leaders meet weekly to discuss rising problems, accomplishments and future goals. As the team works to create a new vehicle, sub-teams collaborate frequently in the communal design level space.

Challenges and lessons learned

The team's primary takeaways from the design process are the importance of time management and a step by step approach to designing the boat from scratch. The team functioned extremely well when given a complete design and concrete tasks, as well as a deadline. In the future, and when the team redesign's Yellowfin into a more permanent design next year, these takeaways will be the foundation of the team's design process.

Conclusion

Yellowfin's system is designed to give the team the best chance at qualification with the most speed and efficiency in the design process. The team's competition strategy revolves around utilizing a more experienced software design to compliment a first time boat. The team has been rushed to prepare for the competition, but is ready and excited to compete, with the main goal of qualification. With more experience and time in the coming years, the team will greatly refine its systems, adding components such as fiberglass pontoons and an improved electronics system. This competition will mark an important milestone in building Beaver's RoboBoat team for the future.

References

- Software libraries and datasheets:
    - Gurobi
    - SciPy
    - EZAUV (our own code)
    - NumPy
    - PyGame (for simulation rendering)
    - ffmpeg (For simulation rendering)
- RoboBoat Handbook