CUSail: Cornell Autonomous Sailboat Team
Navigation Team
Fall 2025 Report

Emith Uyanwatte, Electrical & Computer Engineering 2026, 0 credits
Eric Cai, Computer Science 2028, 2 credits
Nicole Luo, Computer Science 2027, 2 credits
Liya Mei, Electrical & Computer Engineering 2028, 3 credits
Wang Mak, Electrical & Computer Engineering 2027, 3 credits

December 17, 2025

Abstract

This report describes the work of the CUSail Navigation sub-team during the spring 2025 semester. The focus of this semester was to integrate systems from the Fall to prepare for competition. On the electrical side, the system consists of a MiniPC and Teensy microcontroller for controls. The current codebase is run off of ROS2 Humble which is further connected via Rosbridge to a host of support programs. This includes: Webserver, Control App, Sandbox Simulator.

Contents

1 Boat Electronics Overview

(Emith Uyanwatte)

1.1 *Overview*

Last semester (FA24), a Raspberry Pi 4B was phased out in favor of a Peldan WI-6 as the main boat computer. This was done to increase the boat's absolute computational power of the boat at a significantly lower cost than the gold standard Nvidia Jetson line. In addition, this allows for native ROS2 support for any ROS2 version. This semester, the consequences of that decision were better understood. In particular, the thermal limits of the boat, despite the less efficient x86-based CPU, were better understood, as were the power limits. As of writing this report, the total power capacity of the boat is still being understood and tested for the competition's endurance challenge, but an aim is to have this tested before competition in 2025.

2 PCB Design

(Liya Mei, Jonah Conolly)

2.1 *Overview*

- **Design Considerations**: The design this year was primarily focused on separating dependencies by creating multiple separate PCBs for each action on the boat. By separating the power components from the control components, components can be easily replaced and tested separately, overall increasing stability across the system whilst reducing cross dependancies. Additionally, this electronics system was designed with redundancies in mind, fixing previous issues and implementing backup plans in the case of failure.

- **Design Overview**: Our previous PCB which housed voltage stepdowns from 22.2V to both 12V and 5V, alongside connections between the Teensy and necessary sensors and servos was phased out this semester, and will be replaced by the control board and power board. The control board is responsible for housing and facilitating communication between the Teensy, our RC Xbee, the VectorNav, WindVane (anemometer), and multiple servos. The power PCB is responsible for a 12 V stepdown and relay for the mini PC, a 5 V stepdown and two corresponding relays for the control board and VectorNav, as well as communication with the relays using an Xbee and Raspberry Pi Pico.

2.2 *Circuit Diagram*

- **Power Supply:** The boat's power system is powered by a 20V LiPo battery, selected for its long-lasting battery life. To meet the voltage requirements of various components, the power undergoes two stages of regulation. On the power board, the voltage is stepped down to 12V to supply the Mini PC, which receives power via a barrel jack. The voltage is then also stepped down to 5V to power the Teensy 4.0 microcontroller and the servos through a connection to the control board, as well as the VectorNav through a USB-A port. Both of these stepdown circuits were designed using the Texas Instruments's Power Designer with the TPSM84338 chip. The TPSM84338 chip is a buck converter chosen for its flexibility and protection functions, as well as its ability to provide higher currents (above 3A to support the Mini PC). These voltages are connected to three relays: one 12V relay, one 5V relay, and one relay for the VectorNav. The relays in collaboration with the Raspberry Pi Pico and Xbee allow for remote control of the power connection to the Mini PC, control board, and VectorNav.

- **Mini PC:** The Mini PC serves as the central processing unit. It collects data from the onboard sensors and sends control signals to the Teensy microcontroller. This year, the VectorNav (hich acts as both the GPS and IMU) will send sensor output data to the Teensy using the DE-9 connector, and then the Teensy will communicate with the Mini PC so the data can be parsed using Python.

- **Teensy Microcontroller:** The Teensy 4.0 microcontroller has 40 input/output pins and is connected to the Mini PC via USB Serial. It controls the servos using PWM signals, and the specific pins for the servos are defined in the code. The servos and Teensy are powered by the 5V stepdown and corresponding relay of the power board. The Teensy also handles communication with the anemometer, which measures wind speed and direction through its analog input pins (A0-A9). The anemometer is powered by the 3.3V output from the Teensy and shares a common ground with other components. The anemometer features four wires: green for wind direction output, yellow for power, red for ground, and black for wind speed, although the black wire is unused in the current configuration.

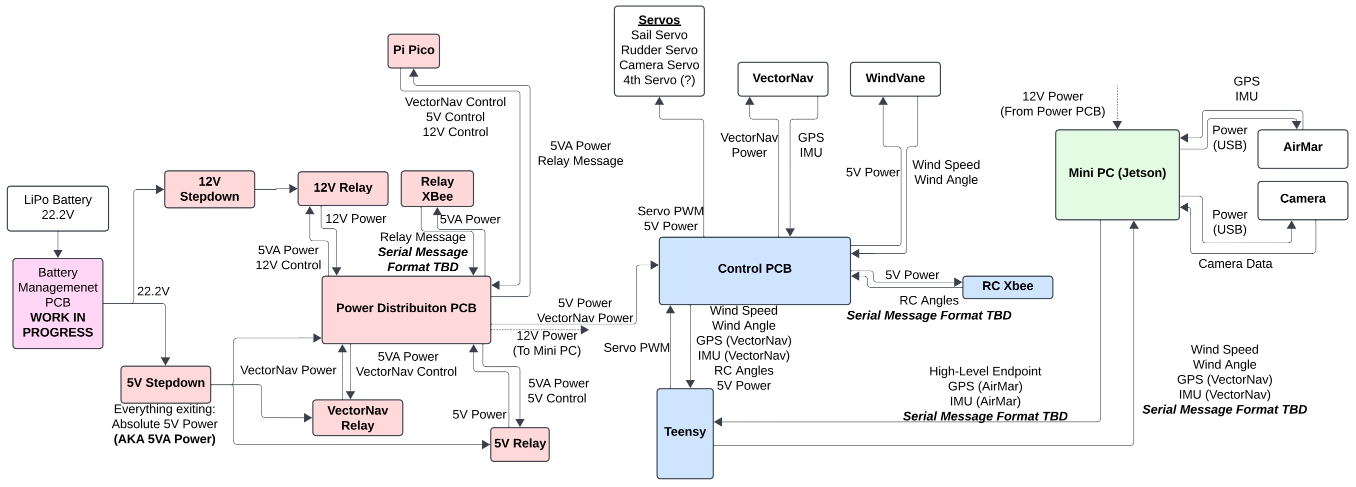

Figure 1: High-level Electronics diagram

2.3 PCB Design

This year, the team utilizes two separate PCBs to route sensor data and provide power to the electrical system while effectively dividing responsibility, increasing modularity, and optimizing efficiency. This allowed for a more resilient system that is easier to maintain and simplifies cable managment. The control PCB currently houses our sensors, VectorNav, servos, supporting circuitry, and microcontroller. The data lane from the VectorNav to the Teensy needs to be stepped down from 12V to 3V, so the Max3323E chip and its support circuitry connect the DE-9 connector to the Teensy. The Max3323E chip was chosen based off of its capability to perform stepdown for low power but high data rates. The control board is powered through a single 5V input received from the power board, where it is controlled by the 5V relay. The PCB also provides a 3.3V output. This output voltage comes from the Teensy 3.3V output. The entire PCB is common grounded to a single ground. The board also features additional headers to access the Teensy pins and power if additional servos or connections are necessary. The RC Xbee is also located on the control board, and allows for remote operation of the boat's Mini PC through wireless connectivity. The power board includes two buck converters, as well as three relays connected to a Raspberry Pi Pico and Xbee to allow for remote power control.

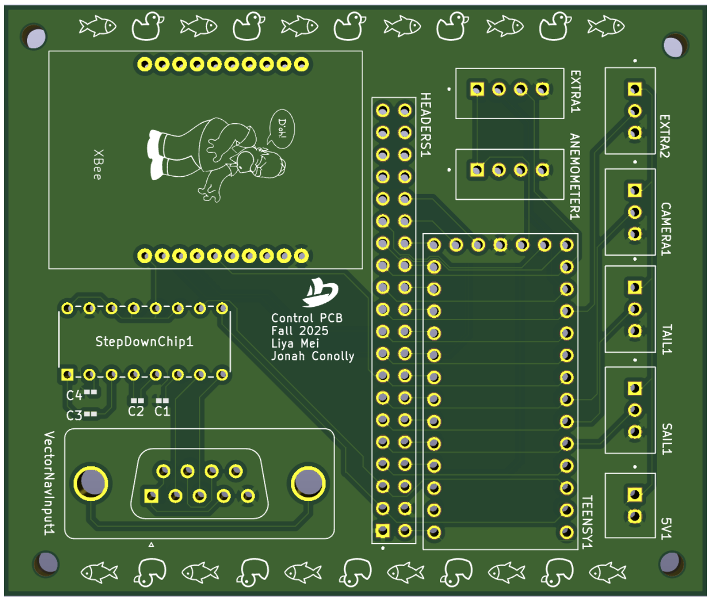

Figure 2: 3D Model of Control Board

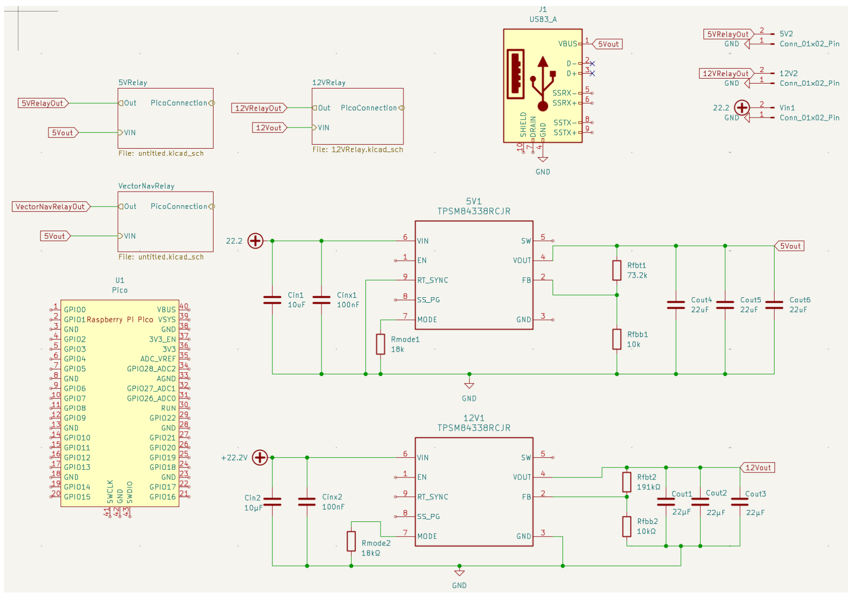

Figure 3: Schematic of Power Board

3 Microcontroller Rewrite

(Wang Mak, Linnea Furlan)

3.1 Overview

This semester's microcontroller work centered on preparing the Teensy for a larger computational role in the boat's control system. In the current system, the Teensy primarily executes commands, while navigation and decision-making remain on the mini-PC. To support a transition toward fully onboard control, development focused on reorganizing the firmware, improving communication handling, and adding safety mechanisms rather than implementing new navigation logic. FreeRTOS was explored as a potential framework for task-based scheduling, and core concepts such as task creation and scheduling were tested through small-scale demonstrations. However, due to the structure of the existing firmware and the absence of onboard navigation logic, full FreeRTOS integration was deferred. Instead, the codebase was refactored to support multiple command sources, standardized packet formats, and safe actuator control, establishing a stable foundation for future RTOS adoption and autonomous operation.

3.2 Initial System

At the beginning of the semester, the Teensy firmware supported a minimal monitoring and control loop designed to interface with the mini-PC. The microcontroller read sensor inputs, including wind direction from the anemometer, and received high-level sail and rudder commands over a single USB serial connection. These commands were directly translated into PWM signals to drive the servos, while feedback data such as wind angle and actuator positions were transmitted back to the mini-PC.

However, this architecture imposed several limitations. All navigation and control decisions were computed offboard, leaving the Teensy without autonomy or awareness of its control sources. The firmware supported only one command input and could not distinguish between autonomous and manual control. Packet parsing was tied to a single ROS-based serial format, making it difficult to extend the system to additional communication channels.

3.3 FreeRTOS Investigation

To support long-term goals of migrating navigation and decision logic onto the microcontroller, FreeRTOS was investigated as a task-based scheduling framework. Initial work focused on understanding core FreeRTOS concepts, including task creation, cooperative and preemptive scheduling, and timing primitives. These concepts were explored through small-scale demonstrations, such as LED-based tasks, to validate basic scheduler behavior and timing.

While FreeRTOS provides a clear model for structuring concurrent tasks, integrating it directly into the existing firmware proved premature. The current codebase was not sufficiently modular, with monitoring, parsing, and control logic tightly coupled in a single loop. Introducing FreeRTOS at this stage would have required a substantial refactor without immediate functional benefit, particularly since navigation logic still resides on the mini-PC. As a result, full FreeRTOS integration was deferred, and development shifted toward reorganizing the firmware into clearer functional components that can later be migrated into RTOS tasks with minimal disruption.

3.4 Firmware Refactor and Communication Design

Following the decision to defer full FreeRTOS integration, development shifted toward restructuring the Teensy firmware to improve modularity and extensibility. The codebase was refactored toward a C++-based design, separating monitoring, parsing, and control logic into clearer functional components. This reorganization reduces coupling between subsystems and prepares the firmware for future migration to task-based scheduling without requiring a full rewrite.

As part of this refactor, the firmware was extended to support multiple communication interfaces. In addition to the existing USB serial connection used for ROS-based commands from the mini-PC, a second serial interface was introduced for radio control via an XBee module. Each interface is handled by a dedicated monitor responsible for receiving and validating incoming data. By isolating communication handling at the monitor level, the system avoids confusing different command sources and allows each interface to change independently.

This design enables the Teensy to act as an intermediary that interprets commands from multiple sources rather than assuming a single upstream controller. The resulting architecture provides a clearer separation between input handling and actuation, which is important for both safety and future onboard autonomy.

3.5 *Packet Formats and Control Determination*

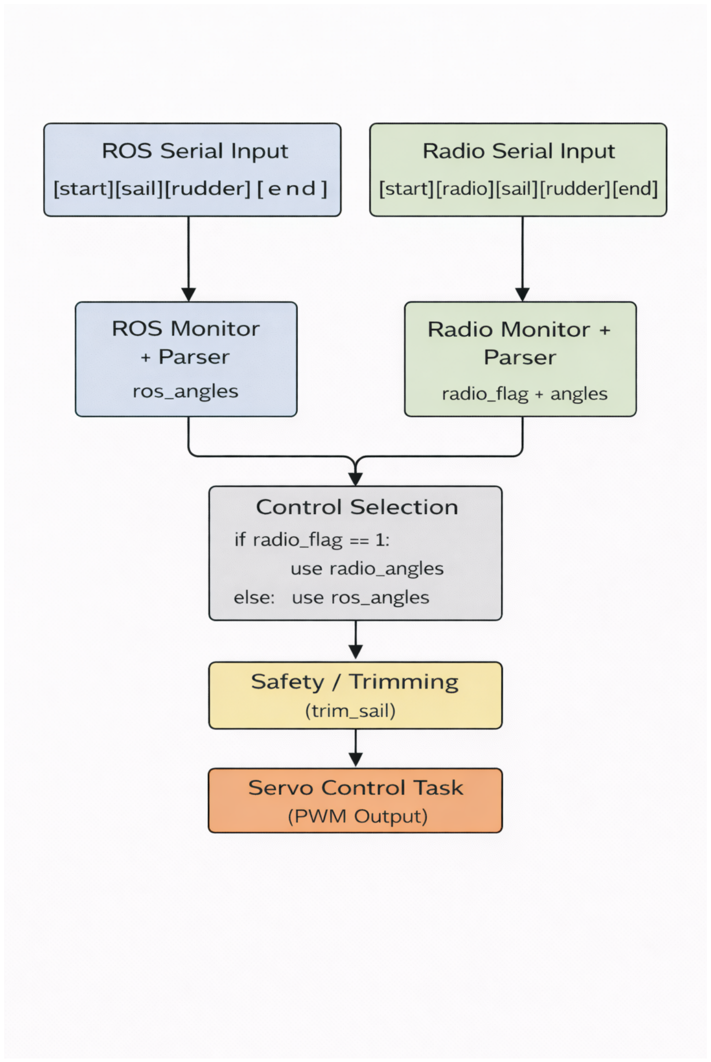

Figure 4: Control block diagram for serial command parsing and PWM actuation.

To support multiple command sources, serial packet formats were formalized for both ROS and radio inputs. ROS packets retain a simple structure consisting of a start flag, sail command, rudder command, and end flag. Radio packets extend this format with an additional control field indicating whether the radio intends to actively command the boat or relinquish control back to ROS. Rather than forcing a unified packet structure, each input source uses a dedicated parser tailored to its format.

Control determination is handled on the Teensy using both packet-level information and signal availability. If a valid radio packet explicitly asserts radio control, the Teensy prioritizes radio commands. If the radio indicates that control should return to ROS and recent ROS packets are available, the system follows ROS-based commands instead. This explicit control mechanism prevents ambiguous transitions between manual and autonomous operation and ensures predictable behavior during handoffs.

This approach improves robustness and establishes the Teensy as the central point for control selection, which is essential for safe operation and future onboard navigation. In the long term, migrating navigation and decision logic onto the microcontroller reduces reliance on a single high-power computing unit, lowering overall power consumption during longer missions and enabling greater system redundancy. Distributing control responsibilities across lower-power embedded systems also mitigates single-point failures associated with the mini-PC, supporting more reliable long-distance and endurance-focused operation.

3.6 *Current Status*

Safety checks were added at the microcontroller level to ensure actuator commands remain within valid operating bounds before being applied to the hardware. In particular, a trimming function enforces limits on sail commands, preventing invalid or unsafe outputs from propagating to the servos regardless of the command source. By placing these constraints directly on the Teensy, the system ensures that all actuator outputs are validated at the lowest level of control.

At the current stage of development, the firmware supports multiple monitoring and control tasks, including serial interfaces for both ROS and radio input, control selection logic, and safe servo actuation. While these components are implemented and functioning, further work is required to ensure that shared state and critical variables are consistently accessible at higher levels of the system. Formalizing how global state is exposed and updated will be necessary as the firmware transitions toward a fully task-based architecture.

With these foundations in place, development is now positioned to begin incremental integration with FreeRTOS and the migration of navigation logic onto the microcontroller. Existing functional components have been structured to map cleanly into RTOS tasks, enabling future work to focus on scheduling, inter-task communication, and higher-level decision making.

4 Sensors

(Ludvig Fellstrom, Emith Uyanwatte)

4.1 *Overview*

In prior semester, the team's main control algorithm only took directional measurements into consideration. This limited insight into the full performance of Red 40 and the Codfather. The idea of generating polar diagrams to visualize our performance envelope as well as the broader objective of enabling sustainable, yet persistent ocean measurement motivated the idea to add a wind speed measurement system to the rest of the stack of the sailboat. In the real world, autonomous sailing platforms such as Saildrone demonstrate how continuous, low impact data collection can expand our understanding of marine environments at scale. For these reasons, we implemented an interrupt-based measurement system within the existing sailbot stack, allowing for real time wind speed acquisition while not interfering with real time control.

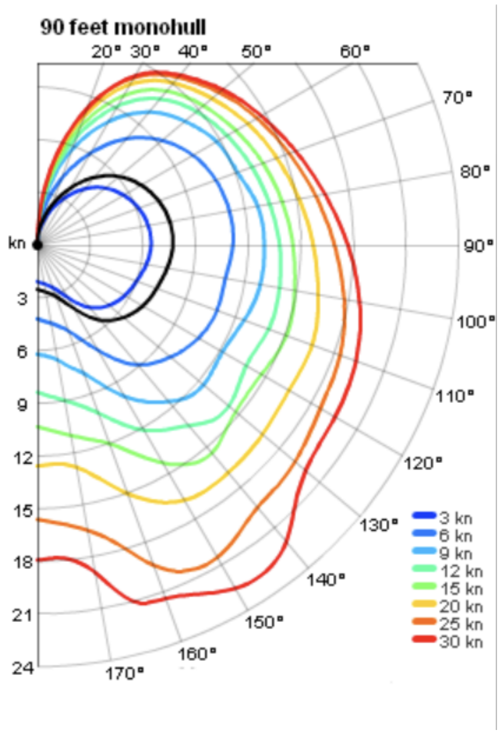

Figure 5: Polar diagram for 90 ft monohull.

4.2 Implementation

The procedure for measuring wind speed was implemented into the stack through interrupts. The wind speed pin is configured with an internal pull-up resistor (see circuit) and attached to an interrupt that triggers on each signal edge, a minimal interrupt service routine increments a pulse counter. While in the main execution loop, elapsed time since the previous update is computed. Finally, wind speed is calculated from the accumulated pulses using the relation $v = P{\cdot}2.25/T$, where P is the pulse count over the interval T. Wind direction remained untouched, being read synchronously through analog input and scaled to the 360° range before being stored. The interrupt handler was kept lightweight on purpose, all computation being performed in the main execution loop.

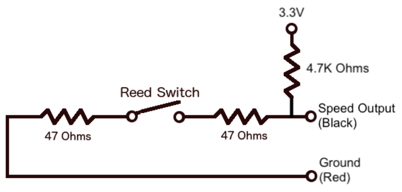

Figure 6: Wind speed circuit of anemometer.

4.3 Future Plans

There are several consideration that we should keep in mind when looking forward. The current implementation for wind speed samples over a fixed-one second interval and therefore produces quantized measurements. There is therefore design motivations to improve precision by restructuring the implementation fo the wind speed formula. By altering the way we track the time between successive pulses, we may raises concerns around added interrupt complexity but we may add potential interference with the rest of the control stack. Beyond firmware, the ability to have wind speed measurements paves the way for further visualization and research, including polar diagrams and possible methods for estimating wind vector fields across Cayuga lake. These tools raise broader questions on the applications of knowing wind speed. How can we use this data to alter our navigation algorithm? The sailing simulation?

5 Algorithm Design

(Nicole Luo)

Throughout the semester, our subteam developed a more principled and robust approach to autonomous sailing by refining the algorithmic framework used to control sail angles, rudder angles, and tacking or jibing behavior. This discussion focused on formalizing the black-box control structure, identifying the key environmental variables, and outlining a state-machine–style flow for decision making.

5.1 Black-Box Control Structure

The control algorithm accepts four primary inputs: wind speed, wind angle, the boat's UTM position, and the waypoint's UTM position. Converting GPS coordinates into UTM allows us to assume a flat Earth model and treat all positions as Euclidean points, enabling direct computation of distances and angles.

The outputs of the controller are the jib sail angle, main sail angle, and rudder angle. Operational constraints determined the feasible ranges:

- Jib Sail Angle: 0° to 90°
- Main Sail Angle: 0° to 90°
- Rudder Angle: $-25°$ to 25°

5.2 Wind-Based Mapping for Sail Angles

Wind angle is the dominant factor in determining both sail angles. Only the functional region between approximately 45° and 180° meaningfully produces lift; therefore, this interval is mapped linearly to sail positions (e.g., 45° $\rightarrow$ 0° and 180° $\rightarrow$ 90°). The jib and main sail should move in parallel, except on deep downwind courses where the jib becomes ineffective.

Past issues with accidental jibes under downwind conditions motivated discussion of shortening the functional range to avoid hazardous areas.

5.3 Rudder Logic and Tacking Behavior

Rudder decisions govern when the boat attempts to tack and how it maintains heading. When sailing upwind, the algorithm should keep the boat as close to the wind as possible, initiating a tack once the heading enters the no-go zone. A tack is considered successful if the boat maintains sufficient speed to cross the wind; otherwise, the rudder should return to neutral and allow environmental forces to rotate the boat out of stall.

To evaluate whether a tack is progressing correctly, the algorithm monitors the rate of change of the wind angle relative to the boat's heading, using onboard rotational data as a stability check.

5.4 Vector-Based Navigation

A major conceptual shift discussed this semester was moving from waypoint-targeted navigation to vector-based navigation. Instead of aiming directly at a distant waypoint—where angle calculations become numerically unstable—the boat selects a preferred sailing vector based on wind direction, staying on the optimal side of the no-go zone.

Two layers of logic were proposed:

1. **Small-scale behavior:** The boat should hold the edge of the no-go zone unless wind shifts exceed a defined buffer range. This buffer prevents micro-tacking from small fluctuations.

2. **Large-scale behavior:** For long-distance navigation, a bounding-box approach provides constraints that force periodic tacks to maintain a reasonable course toward the waypoint.

5.5 High-Level Tacking State Machine

The discussion culminated in a high-level flow outlining how the controller transitions between states:

- **Main Sailing State:** Sail as close to the wind as possible while monitoring which side of the wind provides the better long-term vector.

- **Transition Condition:** If the boat drifts to the wrong side of the wind, evaluate its velocity. If sufficient speed exists, execute a tack; otherwise, default to a jibe.

- **Tack Execution:** Apply maximum rudder until crossing the midpoint relative to the wind, then return to center. If the tack fails, reset and return to the main state.

- **Post-Tack Behavior:** After a tack, the boat naturally has reduced speed; the velocity condition prevents immediate oscillations or repeated failed tacks.

5.6 Future Work

Next steps include transforming the conceptual algorithm into a fully specified state machine with concrete variables, parameters, and pseudocode. Important tunable values include the no-go zone angle x, buffer constant C, wind-based thresholds for tacking versus jibing, and the rudder maximum. Further discussion and lake testing are required to finalize these parameters.

6 Computer Vision

(Eric Cai)

6.1 Overview

A computer vision system is required for the competition task of autonomously searching for a buoy in a 100 meter radius. Specifically, the computer vision system is responsible for recognizing the buoy from camera input. This is challenging because the buoy is a fairly nondescript round orange object, making false positive detections common. Updates to the computer vision system this semester attempt to mitigate these challenges by recognizing buoys with the You Only Look Once (YOLO) computer vision system.

6.2 Motivation

Work has been done in previous semesters on a computer vision system that used an OpenCV color filter to detected buoys by finding the largest orange object in the camera frame. This method is very consistent at detecting a buoy if one is in frame, but it also tends to return false positive detections when other orange objects (e.g., an orange life jacket or an orange jet ski) are in frame. This caused the boat to get "distracted" when running the search algorithm, as false positive detections diverted the boat from its search pattern. Therefore, a computer vision system that had a machine learning approach to recognizing buoys was proposed.

Research was done to find computer vision systems for object detection. YOLO is a computer vision system first released by researchers from the University of Washington and maintained by Ultralytics. This architecture was chosen because its efficiency makes it viable for real-time object detection, which is required by our boat.

6.3 Training and Fine-Tuning

Ultralytics releases a set of pretrained models for object detection. The latest model, YOLO11x, is used as a baseline for the boat's system. This was set up based on the Ultralytics object detection documentation. However, the pretrained model cannot recognize buoys, as they are not a class of objects present in the pretraining dataset. Therefore, fine-tuning the model with labeled buoy images is required. To avoid the time-consuming process of collecting and annotating a buoy dataset, the first attempt at fine-tuning the model was with a public dataset [1]. The model was trained on this dataset for 100 epochs (i.e., 100 passes through the dataset) and outputted the model weights that achieved the best Mean Average Precision (mAP). The mAP metric is the standard benchmark metric for object detection models and is derived from the average area under the precision-recall curve over a range of hyperparameters. This provides a single quantitative measure that comprehensively balances precision and recall. On the validation dataset, the model achieved a mAP of about 93%.

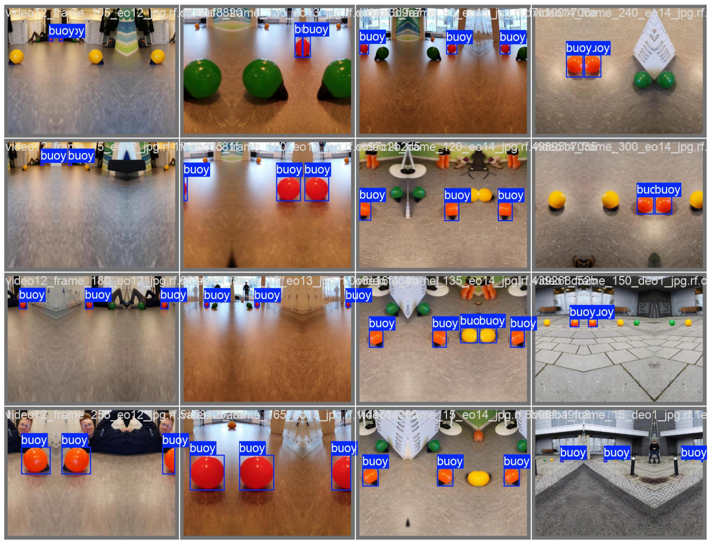

Figure 7: A batch of training images with annotations shown

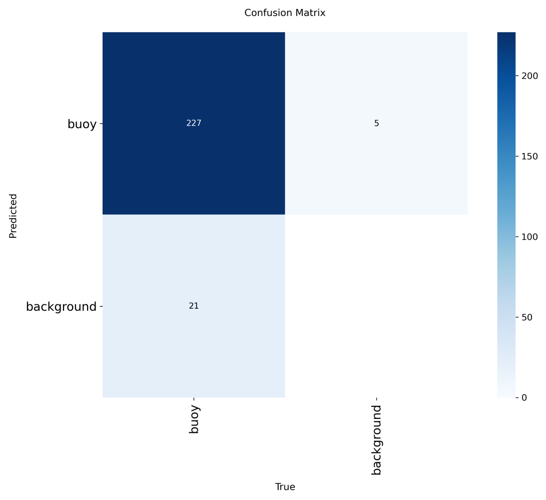

Figure 8: The confusion matrix for the results when running the model on the validation dataset

6.4 Optimizing Hyperparameters

One advantage of the YOLO architecture is that it gives detections a confidence score that quantifies how likely the detection is accurate. Hence, a hyperparameter that must be determined is the threshold for the confidence score needed to return a detection in the boat's search algorithm. Lake tests and land tests were run to observe the behavior of the system at different detection thresholds. It was found found that a detection threshold of greater than 0.8 eliminated nearly all false positives but greatly shortened the distance a buoy could be detected. Conversely, a detection threshold of less than 0.2 returned nearly all visible buoys but returned many false positives. Thus, a detection threshold of 0.6 was chosen, as it led to the model being fairly consistent in avoiding false positives while allowing for a detection range of about 40 meters.

6.5 Future Work

Although creating a custom buoy dataset was avoided due to time constraints, it does offer potential improvements over the current training dataset. This is due to a major limitation of public datasets: they have very few images of buoys at far distances. This was likely a contributing factor to the system's weakness in detecting buoys at range. Training the model on a custom dataset with more faraway buoy images would likely be advantageous.

Additionally, the overall search algorithm can be revisited to better integrate with the slightly different behavior of the new computer vision system. In previous semesters, work was done on writing a particle filtering algorithm to localize the buoy based on detections. Currently, this algorithm converges on the location of the buoy almost immediately after one is detected. Slowing down this convergence by averaging out the buoy position over more detections could better utilize

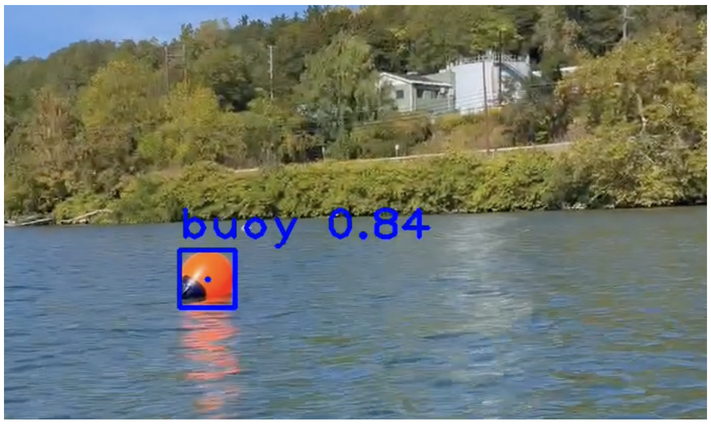

Figure 9: A buoy detection

the YOLO system's reductions in false positives, especially at closer ranges. Work could also be done on incorporating the confidence scores into how the particle filter algorithm weighs different detections.

References

[1] Bifrost, "Buoys dataset," https://universe.roboflow.com/bifrost/buoys-vcquk, mar 2024, visited on 2025-12-08. [Online]. Available: https://universe.roboflow.com/bifrost/buoys-vcquk

[2] J. W. L. J. Augustin Morge, Virgile Pelle, "Experimental studies of autonomous sailing with a radio controlled sailboat," *IEEE Access*, vol. 10, pp. 134 164–134 171, 2022.

[3] R. Stelzer, "Autonomous sailboat navigation novel algorithms and experimental demonstration," *Centre for Computational Intelligence, De Montfort University, Leicester*, pp. 60–140, 2012.

[4] J. S. B. A.-H. Kyle Lemmon, Ted Goodell, "Autonomous sailing across the great salt lake," *University of Utah, Department of Electrical and Computer Engineering*, pp. 1–9, 2020.