Worcester Polytechnic Institute Robotics Engineering Program

SailBot: Autonomous Sailing Robot

A Major Qualifying Project Report
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science by:

Connor Burri - Computer Science
Nickolas Eusman - Computer Science & Mechanical Engineering
Nathan Jackson - Robotics Engineering & Computer Science
Michael Laks - Robotics Engineering
Colin Scholler - Robotics Engineering
Abhijay Thammana - Computer Science & Electrical and Computer Engineering
Luke Zebrowski - Robotics Engineering

Project Advisors:
Professor William Michalson
Professor Kenneth Stafford
Date: May 6, 2021

Abstract

The 2020-2021 SailBot MQP improves upon a multi-year implementation of a robotic sailing system known as *The Great Awake*. The team worked to increase the functionality and reliability of the mechanical, electrical, and software systems throughout the SailBot. The primary goal in this iteration was to implement a robust hardware and software architecture, create and update autonomous systems, and ensure a smooth transition to the next team by simplifying the design and creating thorough documentation

Contents

Figures

1. Introduction

The 2020-2021 SailBot Major Qualifying Project (MQP) is a continuation of four previous MQPs. The goal of this project is to design and build an autonomous sailboat with control and design goals developed around the rules of the International Robotic Sailing Regatta (IRSR). However, due to the cancellation of the 2021 competition for COVID-19 safety, our goals were altered to be able to be demonstrated without the competition. The goals for this year's MQP team are to improve mechanical and electrical systems to increase their longevity and reliability while also refactoring the software systems to make them easier to expand upon. The tasks we aimed to complete by the end of the MQP are:

- A fleet race using RC communication to navigate a closed course
- A precision navigation test involving autonomously navigating point to point within 50 meters with 3-meter radius for the points
- A collision avoidance test to be able to autonomously round buoys
- An endurance test of sailing a pattern for 6 hours
- A search and rescue task of signaling once the boat finds a buoy within a circle of a 50-meter diameter

The team improved upon several systems including the entire software and electrical controls systems of the hull and trim tab, and lastly the data dashboard for the boat. The onboard computers were changed from a combination of an Arduino Mega and a BeagleBone Black to a Jetson Nano. The software system was completely rewritten in ROS2, a much more widely used operating system for robotics. The trim-tab microcontroller was switched from a Teensy 3.6 to an

Arduino MKR 1010. The new data dashboard and simulation was created to allow testing for our new algorithms.

2. Background

2.1 Sailing Basics

In order to design a robot that is capable of effectively sailing under remote control (RC) or autonomous control (AC), it is important to have a basic understanding of how to sail a traditional sailboat. Not only is it important to understand the fundamental physical forces that make sailing possible, but it is also important to understand the systems and practices sailors have developed and refined over the centuries. This section will introduce the points of sail and rights of way that all sailboats must operate under as well as the specific physics involved in sailing the WPI SailBot.

2.1.1 Points of Sail

Points of sail are what amounts to a system for determining how to adjust a sailboat's sails in order to achieve maximum forward motion. A sailboat's point of sail at any moment is based on the direction the sailboat is trying to move relative to the direction of the wind at that moment. A boat with sails adjusted appropriately will always move faster than an identical boat on the same point of sail with inappropriately adjusted sails. The driving force of a sailboat is primarily the result of lift on its sails, though it is also affected, to a lesser degree, by drag. Lift, and by extension lift-induced drag, is produced when the air pressure on the side of the sail facing the wind (the windward side) exceeds the air pressure on the side of the sail facing away from the wind (the leeward side). The lift produced on any point of the sail is always perpendicular to the airstream flowing over that point while the drag is always parallel. Because of this it is best to adjust the

sails so that the air flows over their entire length while still creating a difference in pressure between their windward and leeward sides.

2.1.2 Physics of Sailing SailBot

Our sailboat works slightly differently than a traditional sailboat. Instead of using a traditional canvas sail with lines to control the sail's trim, our sailboat uses a rigid airfoil with a trim-tab to control the airfoil's trim. As a result, our sailboat makes use of similar mechanics as those employed by a plane's wings. The trim-tab works by interrupting the flow of air around the leeward side of the airfoil and using it to induce a force on the trailing edge of the airfoil to oppose the force of the wind on the windward side of the airfoil. Due to the structure of the design, the point where the force of the wind on the airfoil is equal to the force of the wind on the trim-tab is the point at which the airfoil is trimmed to the proper point of sail.

2.1.3 Rights of Way

The rules of right of way are a set of rules used during the potential impending collision of two vessels to determine which vessel has the right and responsibility to keep moving forward (the stand on vessel) and which vessel has the responsibility to get out of the way (the give way vessel). These rules primarily revolve around the difference in maneuverability of the two vessels with only one exception.

The rules for maneuverability are pretty straightforward: with speed comes maneuverability and maneuverable boats are always the give way vessels. Motorboats are often faster than sailboats, which are themselves often faster than man powered boats. Therefore, in general, man powered has the right of way over sail power, and sail power has the right of way over motor power. Of course, in the situation where the motorboat is less maneuverable than the

sailboat, such as when it is in distress or when its sheer tonnage makes turning difficult, the motorboat claims right of way. These maneuverability rules also extend to collisions between sailboats. If two sailboats are travelling in the same direction (on the same tack), which is determined by the windward side of the boat, there are two scenarios in which they might collide. Either a faster boat is behind a slower boat and is threatening to ram their stern or the two boats are next to each other with the windward boat attempting to travel on a lower point of sail than the leeward boat. In the first example, the faster boat is overtaking the slower boat and therefore is clearly more maneuverable. In the second example the windward boat is blocking the wind from hitting the leeward boat, causing the leeward boat to lose maneuverability and gain right of way.

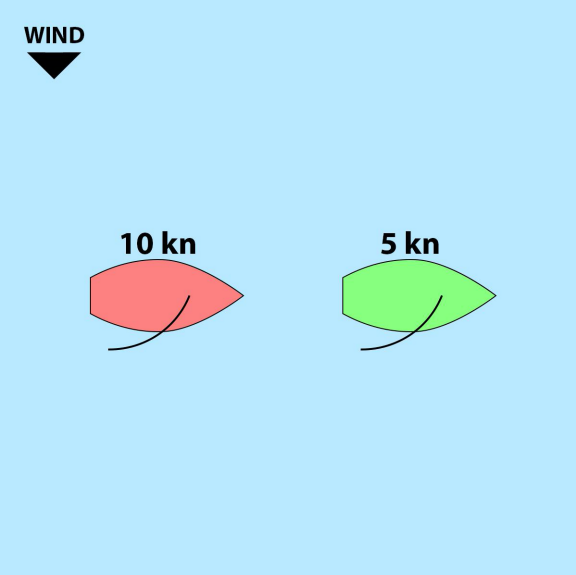

Figure 2.1: The overtaking rule

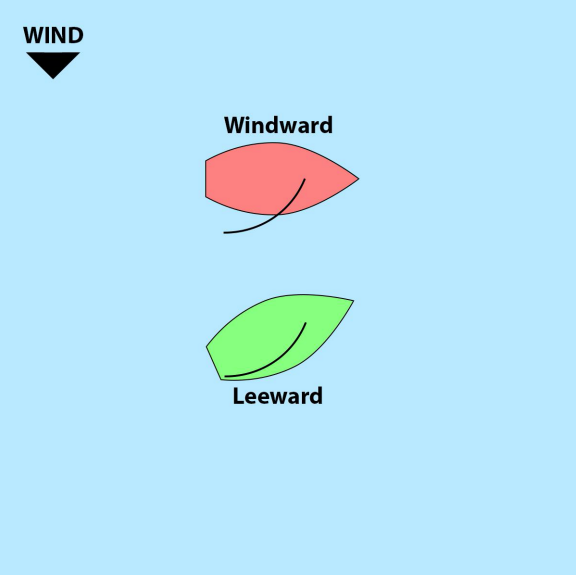

Figure 2.2: The leeward rule

The one exception to the maneuverability rule comes up when two boats are approaching each other on different tacks. In this case the boat on the starboard tack has right of way over the boat on the port tack.

2.2 SailBot Competition

The SailBot competition, officially known as the International Robotic Sailing Regatta or IRSR, is a set of challenges in which unmanned sailing robots compete against one another. The challenges are meant to test the boat's ability to sail successfully both under RC control and autonomous control. The challenges focus on the vessel's ability to sail quickly, avoid obstacles including other boats, carry payloads, and navigate effectively. There are two classes of vessels which compete in the IRSR, SailBot Class and Open Class. SailBot class vessels are under two meters in overall length and at least 50% of the team must be secondary or undergraduate students. Open Class vessels are any vessel under four meters with no other official classifications. Historically the WPI team has focused on two challenges, the fleet race, and the endurance race. The fleet race is a race under RC control where all of the competing SailBots have to sail the course shown below in *Figure 2.3*. The first boat to finish wins the race and scores are given based on finishing position. The second challenge, the endurance race, asks boats to autonomously navigate around four buoys as many times as possible in the span of seven hours. The path for the endurance race is shown below in *Figure 2.4*. Points are deducted if teams need to use RC control to right the boat, or swap batteries. The competition has seven different challenges in total, each of which cover different areas of strength for the boats.

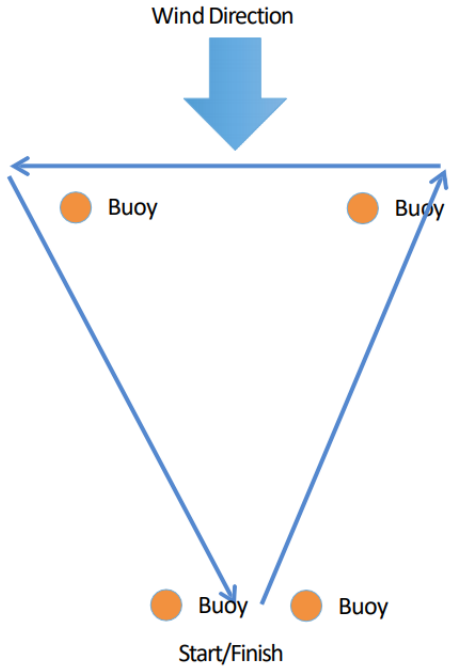

Figure 2.3: Course of the Fleet Race

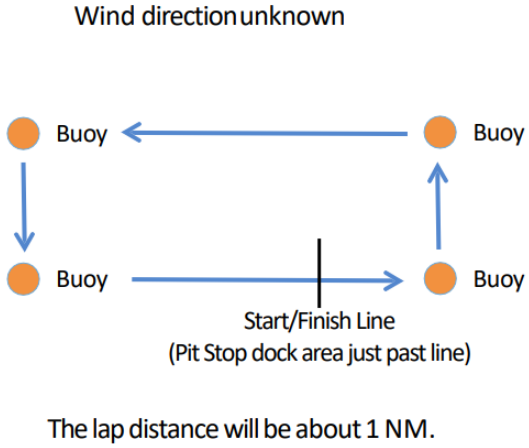

Figure 2.4: Course of the Endurance Race

2.3 Past WPI SailBots

This project is a continuation of previous student projects and is currently on its 5th year. The following sections describe the state of the boat at the beginning of this academic year.

2.3.1 Mechanical Systems

This year we are in a unique position where we are entering the project with a boat equipped with a functioning rigid wing sail. In prior years when the sail design was not yet finalized students had the additional challenges of both designing a new sail and designing the sail control surface. These prior designs have included a winch-based cloth sail, a fixed cloth sail, and a rigid wing sail as is currently on the boat. Of these previous designs a rigid wing sail was chosen due to its several key advantages. The first of these is that considerable weight can be saved as it does not require a

winch to adjust the sail and control surface. Additionally, there are few moving parts making it less mechanically complex, and therefore less prone to breakage. Finally, a rigid wing sail allows for an increased level of control over the movement and propulsion of the boat that otherwise could not be provided.

At the start of this year the rigid wing sail was in almost complete working order, requiring very little work to get it functionally seaworthy. The current sail consists of three parts, the lower rigid wing sail, the trim-tab and control box, and the upper rigid wing sail. The lower section of sail contains the mast which slots into the ship's hull and allows for free rotation of the wing. Atop the lower section of the sail fits the control box which is constructed from lightweight insulation foam. This box contained a Teensy 3.6 with Wi-Fi chip to control the trim-tab and receive commands from the hull's BeagleBone Black through an on-board router. Additionally, inside the control box is a continuous encoder to read wind direction, a splash proof servo for trim-tab control, and four status LEDs for reading error codes from shore. Behind the control box extends a carbon fiber rod which attaches the trim-tab to the control box, with a second carbon-fiber rod allowing the control box's servo to adjust the trim-tab. Finally, the upper section of the rigid wing can be placed above the control box to allow for additional lift in low wind conditions.

The hull of the SailBot consists of four main mechanical systems, the moveable ballast, the keel, the rudders, and the hull itself. All but the movable ballast was left in good working condition at the close of the last iteration of this project. The hull of the boat was made of fiberglass by the 2016-2017 SailBot MQP team using a custom form shaped similarly to a racing yacht. With multiple openings and high stresses due to the heavy keel, the hull is prone to breakage and leaks. That being said, the hull currently has no leaks and is in good condition. The SailBot has dual rudders controlled by a single servo. The dual rudders are angled such that when the SailBot is in

its optimal trim one of the rudders is nearly perpendicular with the surface of the water to maximize its effectiveness. The dual rudders controlled by a single servo does cause a slight amount of drag but previous groups have found that drag to be negligible. The keel of the boat was redesigned by last year's SailBot team to be stronger, more structurally sound, and more hydrodynamic. The keel cavity in the hull had to be modified to fit the redesigned keel and is prone to leaks and damage. The moveable ballast is a roughly 8 kg block of lead attached to a carbon fiber arm which is driven through a gearbox by an automotive window motor. The moveable ballast has two hard stops on either side to prevent it from over-rotating and damaging the hull but contact with these hard metal stops still causes damage to the systems gearbox due to over torquing the gears. Last year's team machined new shafts for some of the gears to prevent the gears from spinning around the shaft like it had done in years prior. They also planned on attaching a potentiometer to the moveable ballast system for control purposes. Due to COVID-19 they were unable to come to campus to implement that plan.

2.3.2 Electrical Systems

Our predecessors in the SailBot MQP faced many obstacles and challenges to design the current iteration of the SailBot. The Electrical System that was idealized, but only partially realized from the 2019-2020 team can be seen in *Figure 2.5*.

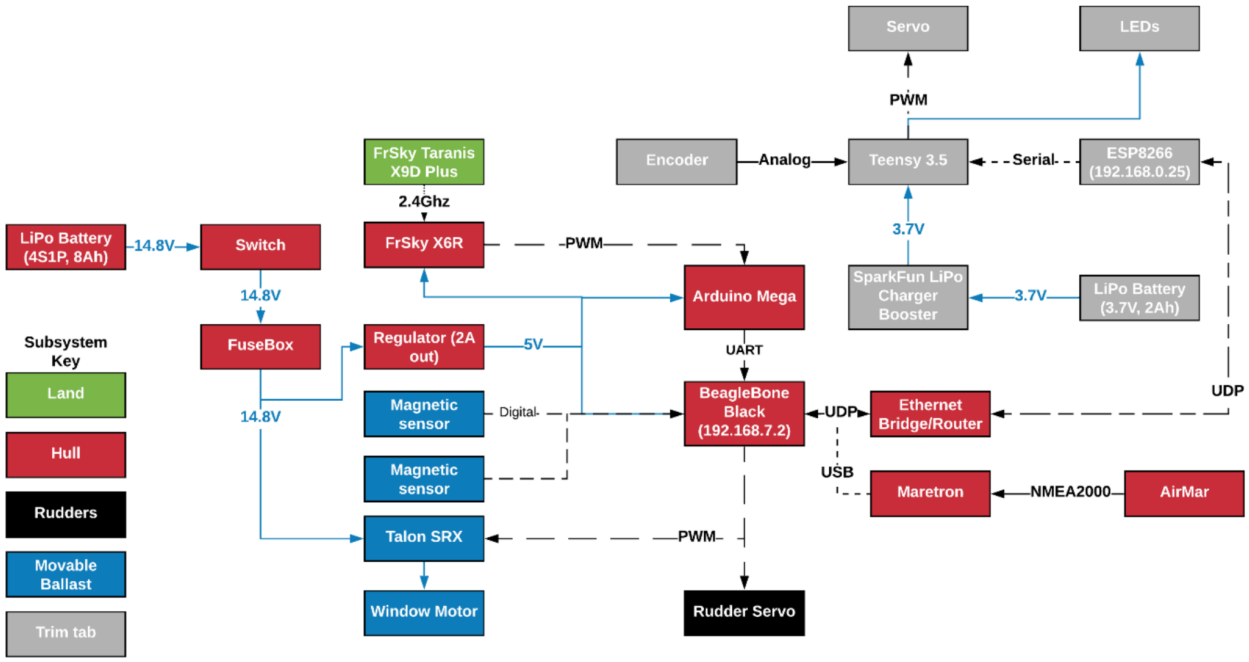

Figure 2.5: 2019-2020 Electrical Overview

The system centered around the BeagleBone Black (BBB), which served as the main microcontroller which was deemed technologically capable for accomplishing all the boat's tasks. However, the BBB did not contain enough PWM ports, so the Arduino Mega acted as an intermediary between the BBB and the radio controller (FrSky Taranis X9D Plus) receiving PWM signals from the RC controller, and converting them to a serial message to the BeagleBone, who in turn actuates the control systems. The radio receiver (FrSky X6R) connects to the Arduino and sends the corresponding signals for the control of the boat to the BBB over a UART connection. This connection was never finished or tested due to the team's inability to come to campus.

The BeagleBone Black was also connected to an onboard Ethernet Bridge/Router. This connection allowed the BeagleBone to be wirelessly connected to the Teensy, the microcontroller for the trim-tab. This was advantageous as it allowed the Teensy housing to remain sealed, and the mast to remain freely pivoting.

Most of the electrical sensor and component control was done through the Airmar which was connected to BeagleBone Black. The Airmar is a critical component to the SailBot. It hosts a wide range of sensors including the magnitude and direction of the wind, GPS, and compass reading. This system required a Maretron to convert the Airmar's NMEA 2000 messages to the BBB over a USB connection. The 2019-20 team planned on moving the Airmar to a more desirable position but were unable to due to campus closure. They instead started to run simulations to find the optimal position for the Airmar. These were also unable to be completed. Hall effect sensors were also about to be placed to act as limit switches for the moveable ballast and were connected to the BBB over Digital ports. The rudder servo control was done through a PWM port from the BBB.

The ballast control was planned to be done through the BBB, but it was unable to be completed due to campus closures. The state of the boat as is, contained a CTRE Hero connected to the TalonSRX motor controller for ballast controller.

The Teensy board is an isolated electrical system on the trim-tab. It contains its own power supply and only connects to the primary electrical system over a network connection. One of the Teensy's functions is to operate the angle of the trim-tab using a servo located within the trim-tab housing. It also controls four LEDs attached to the exterior of the trim-tab housing which act as a simple debugging interface for the trim-tab subsystem. The Teensy's final use is to serve as an interface to the hull subsystem to send the airfoil's relative wind direction to the hull subsystem and receive back servo positioning commands based on that particular wind direction and desired point of sail.

Power for the rest of the components in the hull was distributed from a 14.8V lithium polymer battery. The battery is connected to a magnetic power switch and then into a fuse box

where it is connected to the hull components. The full 14.8V was only sent to the Talon while a Buck Converter stepped down the remaining power to 5V where it was sent to the rest of the components, the Arduino, the BBB, the Receiver and sensors.

2.3.3 Software Systems

The previous iterations of the WPI SailBot's software could be split into two sections: software for radio control, and software for autonomous control. The SailBot Competition necessitated the inclusion of manned control. Previous teams had implemented manned control by multiple means including using a chase boat to stay connected over Wi-Fi, using an RC controller, and connecting to the boat over an ethernet bridge. The software for automation of the boat usually consisted of software built for each SailBot competition challenge, with reusable code for specific tasks within each challenge, such as rounding a buoy. Additional software was built for computer vision systems, as part of a separate MQP.

The previous MQP utilized a similar software approach to previous years, relying on a microcomputer which interpreted radio control commands and actuated the proper mechanical systems such as rudders and ballast. Additionally, the previous year made use of a Wi-Fi network to connect the trim-tab control and the hull systems. They would communicate over websockets and utilize Google's Protocol Buffers system (protobuf). The trim-tab receives commands over Wi-Fi from the BBB and sends back information on its current position and wind direction.

Additionally, the previous team implemented a debugging web server, which aggregates data across all systems, and displays it to the user on a html template.

3. Purpose and Project Goals

Throughout our project we have two main achievements in mind, the first is to continually improve the reliability and sustainability of the WPI SailBot project and the second is to achieve all of our design goals.

In order to continually improve the reliability and sustainability of the SailBot Project our team determined it to be essential that we ensured a smooth transition between project groups. To achieve this, our team strived to keep project longevity and sustainability in mind throughout the design and engineering process. Fostering this spirit of maintainability is of paramount importance for the long-term success of the project.

Below our design goals are listed as five objectives roughly based on the official SailBot competition goals. These competition goals serve as a benchmark to determine our project's degree of success.

1. The SailBot allows for a human operator to use full RC control to navigate a closed course.
2. The SailBot is able to autonomously navigate between two points placed a maximum of 50 meters apart with an accuracy of within 3 meters of the target point.
3. The SailBot is able to autonomously round buoys.
4. The SailBot is able to continuously sail a predetermined closed pattern for a minimum of six hours.
5. The SailBot is to be able to sail a search pattern within a predetermined 50-meter diameter circle in order to locate a randomly placed buoy, then signal the GPS coordinates of the buoy to shore.

4. Methodology

4.1 Trim-Tab

The trim tab is an essential element of the boat, creating the lift necessary for the boat's movement. The trim tab sits midway up the sail and has the following features: a servo to control its angle, a relative wind indicator for finding max lift, a MKR Wi-Fi 1010 microcontroller to control the systems, and a splash proof enclosure. All these elements are discussed further below.

4.1.1 Mechanical

The trim-tab control housing serves two mechanical roles. The first is to serve as a connection point between the primary lower sail section and the secondary upper sail section in low wind scenarios when both sections are necessary. The second role is to adjust the position of the trim-tab to generate propulsion through lift from the mainsail. This is achieved through the simple action of adjusting the embedded splash proof servo position which drives a carbon fiber pushrod to align the trim-tab to a position predetermined by the low-level algorithm.

4.1.2 Electrical

The trim-tab control housing has two primary electrical roles that are managed by an Arduino MKR1010 which is overseen by the Jetson Nano. The MKR1010 is controlled by the nano over a socket connection established by the onboard Wi-Fi chip. Power is provided by a 3.7v 2000mAh LiPo.

The first role that the MKR1010 manages is controlling the trim-tab's servo position which is determined by commands sent by the Jetson Nano. The second, is to read the relative wind

direction from the forward mounted absolute encoder in order to determine if any adjustments should be made to the servo. The final role is to relay the current state of the trim-tab to the Jetson Nano in order to collect data on how we are sailing given various conditions.

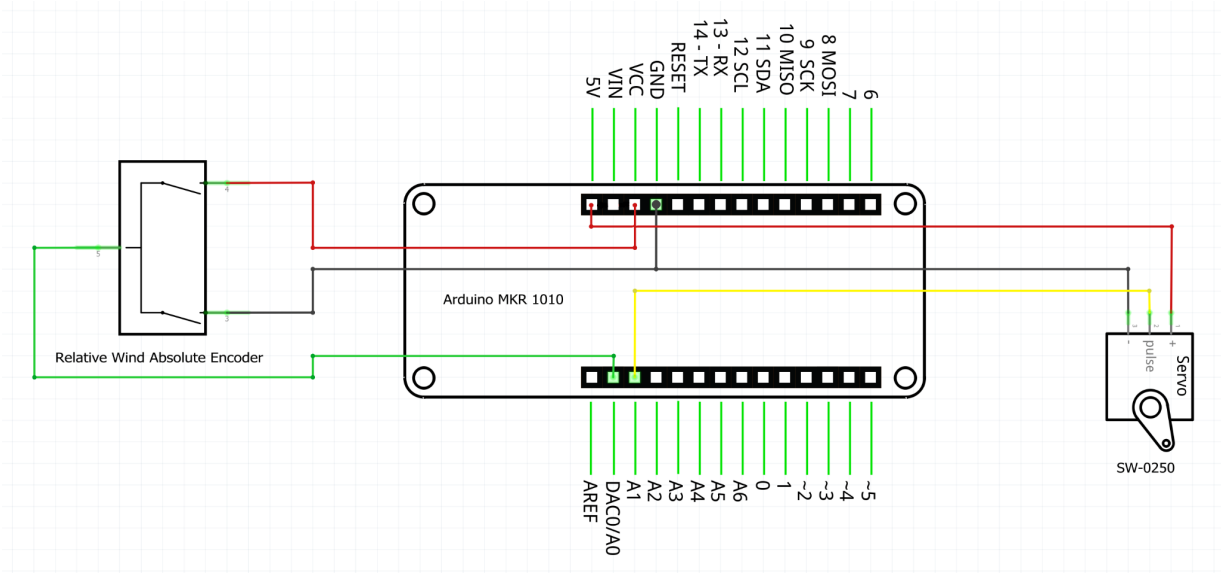

Figure 4.1: trim-tab Control Housing wiring schematic

4.1.3 Software

The software flashed onto the MKR1010 can be generalized as the slave end of a master/slave communication model, whatever the Jetson Nano commands, the trim-tab obeys. The master end which computes the state of the trim tab is further documented in section 4.3.1.6. The trim-tab will only receive one of six states from its master in order to affect the servo position.

The last of these states is the manual state which determines a set angle for the servo and allows for the trim tabs position to be determined via RC using the left joystick on the Taranis controller. The remaining states are used for autonomous sailing. The first and second states are to generate maximum lift from port and starboard sides respectively depending on which direction

the wind is coming from. The third and fourth states are used to create maximum drag automatically on either the port or starboard side depending on the wind direction. The fifth state is to automatically go into minimum lift, this sets the servo to be centered as to not generate lift or drag for the mainsail.

4.2 Hull

4.2.1 Mechanical

The hull's mechanical systems serve an important role in sailing the SailBot by keeping the boat in good trim (~20°) and assisting with maximizing rudder authority. The movable ballast had some problems identified with it such as the lack of feedback sensors to stop the arm from over torquing and backlash in the gearbox, but it functions. The other mechanical system on the hull, the rudders are in perfect condition and no modifications have been made to the system.

4.2.2 Electrical

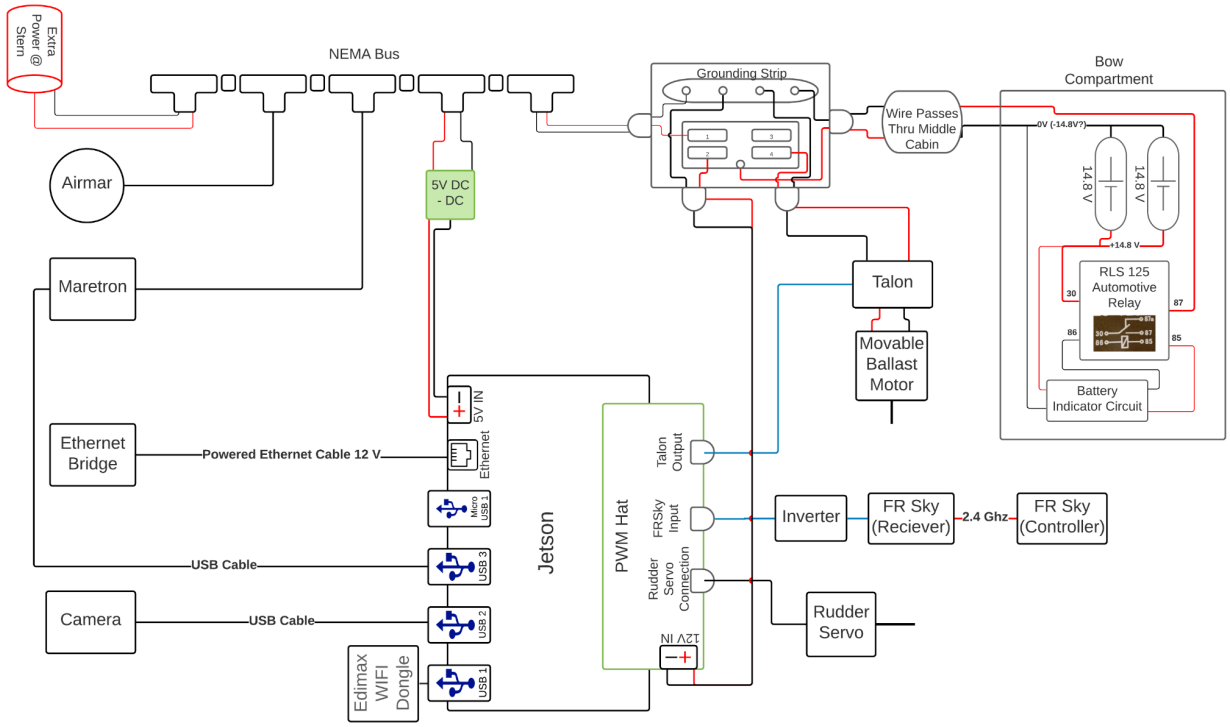

Figure 4.2 Hull Wiring System

The electrical system on the hull contains most of the brains and the brawn of the SailBot. The hull electrical system runs on the NMEA 2000 bus, a wiring and communications standard for marine applications. The components in the hull attached to this NMEA bus can be broken up into multiple categories: sensors, motors, telemetry devices, boards, intermediary devices and batteries. The Airmar weather station is the main sensor aboard SailBot which provides sensor data such as wind speed and direction and GPS position data that the boat uses to navigate and sail effectively. The full array of sensors in the Airmar weather station can be seen in figure 4.3.

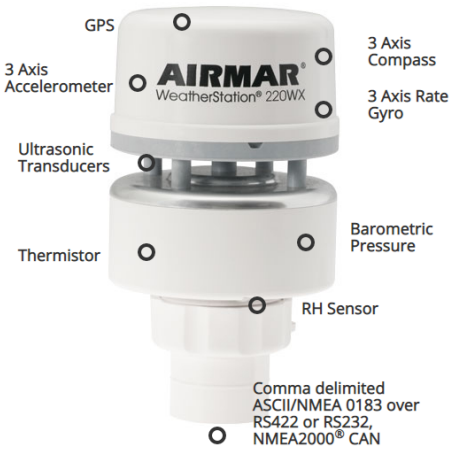

Figure 4.3 Airmar Weather Station Sensor Array

The board that controls the SailBot and connects the electrical system together is the Jetson Nano. This board handles all processing and control for the hull systems and communicates with the trim-tab Nano through a Wi-Fi connection. The Jetson creates a network using the Edimax EW-7611. The hull holds two motors in its electrical system, the first being a servo motor which turns both rudders simultaneously that is controlled by the Jetson microcontroller, and the second is the movable ballast motor. This motor for the ballast draws the most power from the system's batteries and to achieve an accurate power draw estimate, the system had to be tested. Using an ammeter and a constant voltage power supply, the movable ballast was tested using PWM signals from an Arduino, attached to the Talon SRX motor controller which runs the motor. After testing the movable ballast for current draw, multiple other components were also tested using an ammeter and a constant voltage supply to come up with the overall power draw estimate shown in the table below.

Device	Peak Power Draw (W)	Estimated Power Draw (W)
Jetson	20	15
Airmar	.9	.85
Maretron	.6	.6
Router	1.62	1.62
XR6 Transceiver	.5	.5
Transceiver to UART	.1	.1
Camera	1.1	.9
Talon SRX	.6	.6
Movable Ballast Motor	336	46.4
Rudder Servo	2.45	0.5
Total:	**363.87 Watts**	**67.07 Watts**

Figure 4.4: Table system of power draw estimates

From these estimates a new set of batteries to drive the boat's electrical system was decided upon. A goal of the team is to have the boat continuously sail for up to 6 hours, so the batteries had to be able to provide enough power to do the 6-hour run as well as some other shorter challenges. 2 LiPo batteries were selected because they were light and provided enough power to run the system. The run time estimate is shown in the table below.

Battery	Quantity	Total Power (Wh)	Max Discharge (A)	Max Run Duration @ Avg Power Estimate (H)	Weight (lbs)
20000mAh 4S 12C LiPo Pack	2	592	120	8.83	7.8

Figure 4.5 Battery runtime estimate

The various components of the hull's electrical system had to be contained within 3 housings to keep them away from any moisture that accumulates inside of the hull. The majority of the hull's components were housed in the hull's main housing pictured below. 3-D printed mounts allowed the components to be seated in the housing and fully removable for future updates to wiring. A series of waterproof panel mount connectors were attached to the side of the housing to allow for modular connection to the components inside. A separate housing holds the fuses that connect directly to the batteries. The batteries are also housed in a 3-D printed modular housing and sit at the bow of the boat.

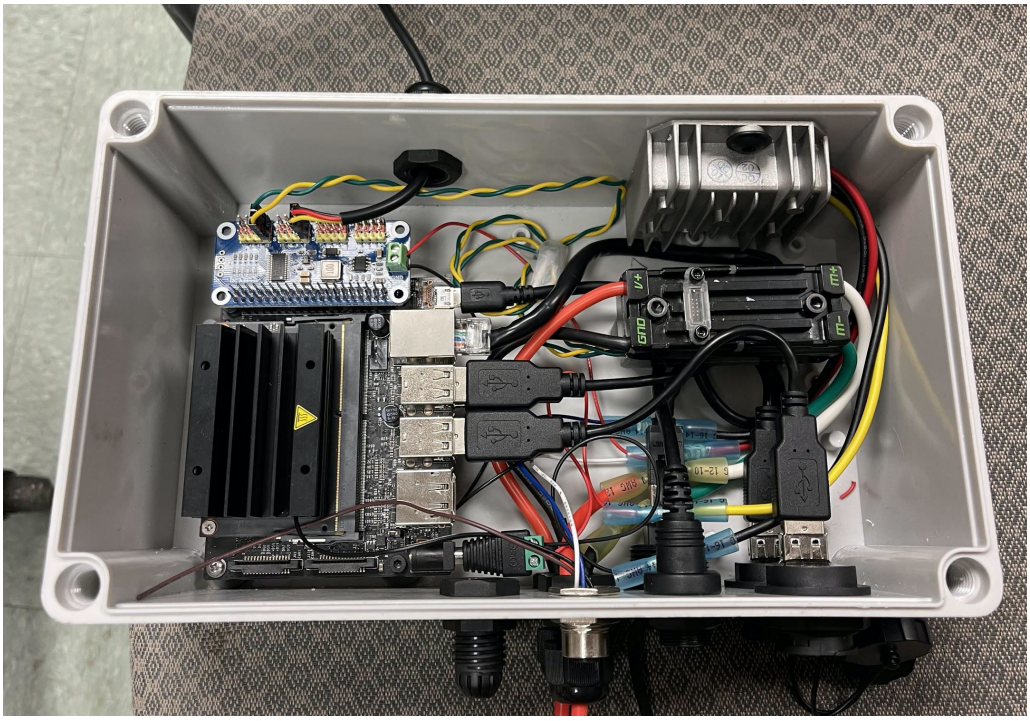

Figure 4.6: Hull Electrical Component Housing

The hull electrical components in Figure 4.6 control all the aspects of the boat. For specifics, the black microcontroller with the large heatsink on the left is the Jetson. The PWM Hat is the blue board placed on top of the Jetson's header. The silver box object on the top right is the DC-to-DC converter which steps the voltage down from battery voltage (16 V) to 5 V. The smaller black box pictured mid right is the Talon motor controller for the Ballast. We initially had some concerns over the temperature of the components in this box, but they were alleviated after testing. We measured these temperatures from the Jetson, which were well within the operating range of -40℃ to 85℃. The first measurement was with the boat running outside the water for 25 minutes under RC testing. The second measurement was taken after the boat had been operating for over a half hour in the water. After putting it in the water we actually observed the temperature decrease, likely due to the water cooling the hull.

Time	CPU Temp (℃)	GPU Temp
25 Minutes	61.5 ℃	57 ℃
90 Minutes	54.5 ℃	50 ℃

Figure 4.7: Jetson Testing Temperatures

4.3 Software controls

The following sections outline the various software controls used by the boat, including the ROS control architecture, the autonomous algorithms, and the telemetry dashboard.

4.3.1 ROS2 Architecture

The software for SailBot is written on the second robot operating system (ROS2) framework. In ROS2, you create a series of nodes, which communicate by publishing and subscribing to topics. For our software we have split the code into six nodes: airmar_reader, pwm_controller, serial_rc_receiver, control_system, teensy_comms, and debug_interface. These nodes handle different aspects of the boat but can be broken down into nodes which supply information (airmar_reader and serial_rc_receiver), nodes which make decisions (control_system), nodes which control the boat's actuators (pwm_controller and teensy_comms), and nodes that record and visualize data (debug_interface).

Below shows our six nodes, shown in green squares, as well as the physical systems they interact with in blue, and software systems they interact with in red. The black arrows in the diagram show the publishing and subscribing of different topics, where the arrow goes from the publisher to the subscriber. The basic flow of the topics shows how information is passed from the

Airmar and RC into the control node and out to the nodes controlling actuators. The debug interface collects from all topics.

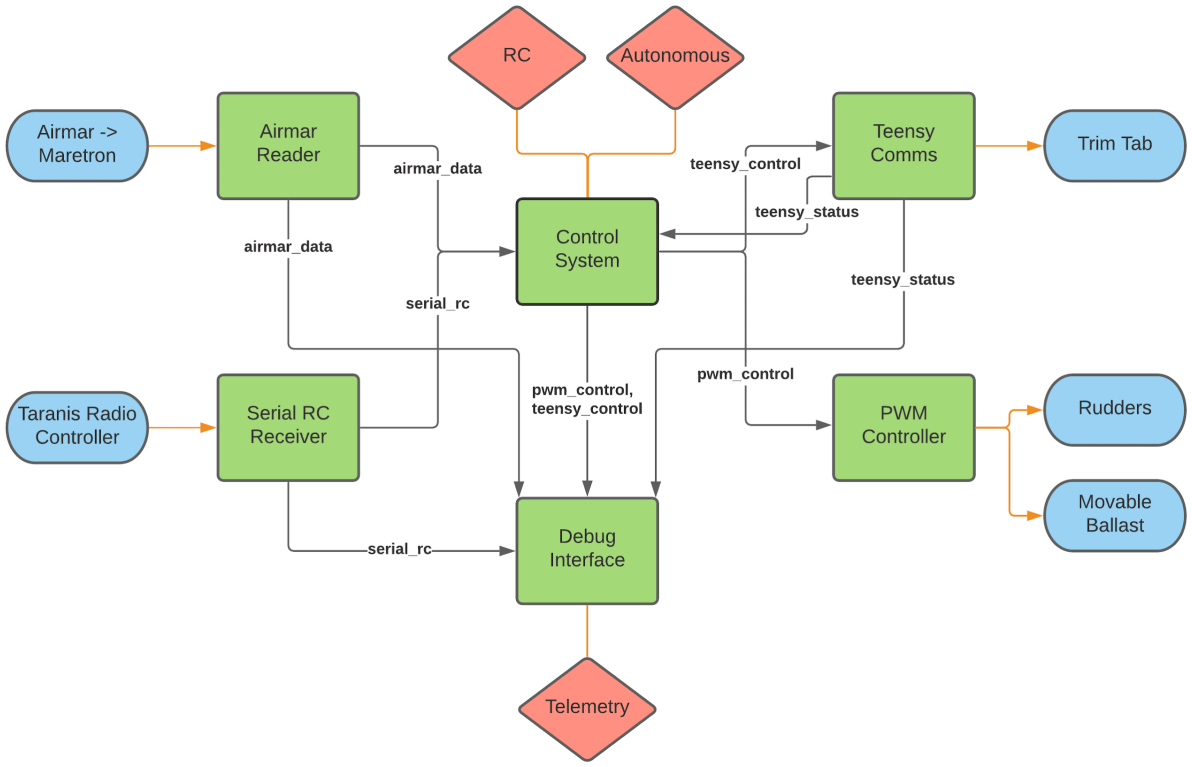

Figure 4.8: ROS2 node architecture diagram

The nodes separate concerns between different subsystems of the boat and the following sections further describe all nodes individually.

4.3.1.1 Control System Node

The control systems node is responsible for the interpretation of data, and decision making for the boat. It can operate in three different states: full RC, semi-autonomous, and autonomous. If in RC mode, it will take inputs from the serial RC receiver node and send the appropriate trim

tab commands to the Teensy Comms node as well as ballast and rudder movements to the PWM controller node. If in semi-autonomous mode, the control system will take over the control of the ballast and trim tab, leaving only the rudders to be steered by RC. The control system accomplishes this by putting into effect the ballast algorithm (discussed further in section 4.3.3.3), as well as using the relative wind direction from the Airmar to assign a trim tab state. A running list of ten wind readings is kept by the node, and the median value is used to determine trim tab state. If the boat is put into autonomous mode, the control system will handle the execution of the autonomous control algorithms, including the A*, P2P and ballast algorithms. These are further discussed in the algorithms section 4.3.3. The control system passes in GPS, wind direction (again a median of 10 is taken), compass data, and tracking data all from the Airmar, and receives back instructions to execute the algorithms.

4.3.1.2 Debugging Interface Node

The Debugging Interface Node is responsible for gathering data related to different systems on the boat and relaying that data to a central dashboard where it can be processed. This helps by giving us a wider picture of the overall status of the boat at any given moment, which is extremely useful for assessing flaws in the robot's behavior. This also allows the aggregation of all collected data, which can be used later for debugging.

4.3.1.3 Airmar Reader Node

The Airmar Reader Node is responsible for interpreting the output coming from the Airmar and making it available for use by other nodes. It does this by decoding the NEMA messages from the Maretron, using the unique type code for each piece of data being interpreted and translating

to a readable format. The node then writes the interpreted data to a json and publishes it on the Airmar Data topic, where the data can be accessed by any node with a subscription.

4.3.1.4 PWM Control Node

The PWM Control Node subscribes to the PWM Control topic and uses the information published on it to control the Raspi PCA9685 Servo Driver. The servo driver is used to control the rudders as well as the moveable ballast with the talon motor controller.

4.3.1.5 Serial RC Receiver Node

The serial RC receiver node is tasked with being the link between the hull and the Taranis remote controller, which communicates with the FrSky receiver in the hull. The node decodes six channels from the remote control, including the 4 channels for the two joysticks and two state switches. The state switches are positioned on the top left and top right of the face where the antenna is located and correspond to states 2 and 1 respectively. When both switches are in the upward position (toward the user), the boat will be in full RC mode. During this, the left joystick's left to right motion will control rudders, and it's up and down motion will control the trim tab. The right joystick's left to right motion controls the movable ballast. When the state 1 switch is put downward, the boat enters semi-autonomous mode. In this mode the movable ballast will keep an ideal trim automatically, and the trim tab will adjust its state based on the wind direction relative to the hull. All controls are disabled except for the rudders in this mode. When the state 2 switch is put downward, the boat enters into autonomous mode, and will navigate to set points based on the autonomous algorithms.

4.3.1.6 Teensy Comms Node

The Jetson Nano uses the python socket library to communicate with the teensy in the trim-tab, the same library used by the BeagleBone Black in the previous year. However, rather than continuing to use the Nanopb library to encode the data, we decided to switch to using JSON. Python has a built in JSON library that allows for simple encoding and decoding and using JSON strings also allows us to publish the messages to ROS topics to another without having to worry about declaring unique message types. The values the Nano sends are the desired state of the trim-tab as well as the control angle if the trim-tab is to be set to the manual state.

The trim tab states are determined based on relative wind angles to the hull. The angle goes between 0 and 360, with 0 being wind coming directly at the bow, 90 being wind coming from the starboard side of the hull (wind going toward port), 180 being wind coming from the aft (heading downwind), 270 being wind coming from the port side of the hull (wind going starboard), and 360 is equivalent to 0. The states of the trim tab are broken down as follows. State 0, max lift port, is used when the angle of the wind relative to the hull is between 45 and 135. In this state, the trim tab uses the PID of the relative wind indicator on the trim tab to place itself in a position of maximum lift (generally around 30 degrees from the wind). This state covers the points of sail of beam reach, close reach, and broad reach, and thus we need lift for maximum speed. If the relative wind angle of the hull is between 135 and 180, we are headed downwind, and will enter state 2, which is the state of max drag port. In this state, the trim tab moves to the max angle to create drag enabling the boat to travel downwind. Similarly, when the relative wind angle to the hull is between 180 and 225, we enter state 3, max drag starboard. This works the same as before, but the position of the trim tab will be flipped to the other max angle. When the relative wind angle to the hull is between 225 and 315, we enter state 1, max lift starboard. This state also makes use of the trim

tab's relative wind indicator, and tries to reach a state of maximum lift, where the wind relative to the trim tab is at an angle of roughly 30 degrees. This state is the inverse of state 0. Any angle of wind relative to the hull which falls between 315 and 45 is considered upwind, and thus the trim tab will be set to state 4, the state of min lift. In this state the trim tab will move to the middle position, where it acts similarly to a weathervane, generating the minimum possible lift. The final state is state 5, the manual state. In state 5, a control angle is sent with the state, which manually sets the servo position for the trim tab. This state is used in RC.

4.3.2 Telemetry Dashboard

The purpose of the telemetry dashboard is to aggregate all the data that is available on the boat for testing and debugging purposes. With the dashboard we can view the data collected in real time and adjust our sailing plans accordingly. We can also see if the sailing algorithms are being applied correctly.

4.3.2.1 Architecture

The data, to display on the dashboard, is sent from the Jetson to the computer on land. This is done using the pair of XPress Ethernet Bridges. The Jetson is connected to the Ethernet Bridge through its ethernet adapter. The ethernet adapter for both the Jetson and the land computer must be set to the Bridge's IP for this to work. The Bridge must also be connected to a 12 V power supply. Inside the boat we are using a 5V power supply and this seems to work fine. After the hardware is configured, the software can be started. The data from the Jetson is received in a simple Node.js server hosted on the land computer. The data is sent in a post message with a JSON format to a certain routing path. This can be sent once the server is starting and listening on the set port. The server works uses a couple libraries from the npm database: 'http', 'express', 'socket.io', and

'body-parser'. If a browser is open to the default route on the 'host:port' (localhost:3000) the server is listening on, the dashboard will be sent to display on the browser with the necessary code. Once the data is received from either the simulator or the boat, the data is sent out to all of the currently listening clients that have connected to the server. The structure of the code is a bit hard to understand, but the modules are all separate so they can be serviced individually. On the browser side we generate the entire dashboard for the user to see. Each of the different JavaScript files generate a different component on the screen, like the compass module, or the map module. All the data received from the boat is fed into the 'socketController.js' file which updates the corresponding components with information. For an in-depth explanation on how each of these modules work, please read through the GitHub[1]. Once started, the full server (implemented in 'server.js') is the one that actually makes a connection with the Jetson's debug node and receives data. The data is then saved to a database (csv file) and sent to each of the browser clients currently connected to the server.

[1] https://github.com/wpisailbot/sailbot20-21

4.3.2.2 Visualization

Figure 4.9: Telemetry Dashboard Visualization

The visualization is a standard HTML page with required JavaScript files. The dependencies involved in displaying data include: ‘socket.io’, ‘Google Maps JavaScript API’, ‘d3’, ‘d3gauge’. The dashboard works by creating divs for each element and then loading the corresponding code to generate each element. The elements are also created in their corresponding files, vectors in ‘initVectors.js’ etc. Almost all of the components for each element are hardcoded svgs. To animate between different states, d3 was used to interpolate the values in between for the svg transformations. The map was generated through the Maps API and the code for this is in the ‘mapController.js’ file. The center of the map is adjustable but hard coded to Regatta Point. The boat path will not save if you refresh the page, but it will generate a path between all of the given GPS coordinates in the given session. The waypoints are also currently hardcoded. The data to update the page is received through the socket.io library, and once a json is received the dashboard updates each of the components with their corresponding code. All of the updating is in the

‘socketController.js’ file. The webpage also allows you to view a debug mode where you can see all of the current grid points that will be used for A*. You can select and deselect points that will be available for sailing and adjusting the corner and center points on the webpage. You can lastly export a file that saves all of the data on the grid points and will be used on the Jetson for high level pathing.

4.3.2.3 Simulation

The simulator works by emulating the messages sent by the boat. There is a json string sent to the server using the ‘socket.io’ library. Each of the components have set ranges of possible values. The simulator either sets a random value to send, or if there is an option, it will send the set value on the webpage. Currently the simulator is very low level. There is a slider to set the magnitude and direction of the apparent wind. For navigation, the boat is sent a very long list of coordinates that will display a path starting from the dock and navigating around the buoys. The simulator will be using the algorithm (4.3.3) to generate coordinates based on the autonomous navigation of the boat. The lake will be divided into squares and then A* will be performed for the optimal boat path (4.3.3.1), which will be held in the simulator’s code. The location of the boat will be calculated based on the point-to-point algorithm (4.3.3.2) and the simulator’s values for the environment.

4.3.3 Sailing Algorithm

The sailing algorithm the team has created is broken up into three parts: A high level pathing algorithm, a low-level point to point travel algorithm, and a ballast control algorithm. The intended interaction of these is such that the high-level algorithm defines a route to travel by creating a series of waypoints which the point-to-point algorithm moves between. The ballast

algorithm runs independently of both these at a constant rate, updating the ballast angle to keep the boat in good trim.

4.3.3.1 Pathing (High Level) Algorithm

The high-level algorithm works by dividing the sailing area into a grid of squares, where each square is a node. The edges of each node denote directions in which the boat can sail. We decided on 8 directions (edges) from the boat's current position to the next possible position and these 8 directions represent different points of sail. The algorithm then removes nodes (squares) for obstacles including shallow water, docks, and land. An example is shown below.

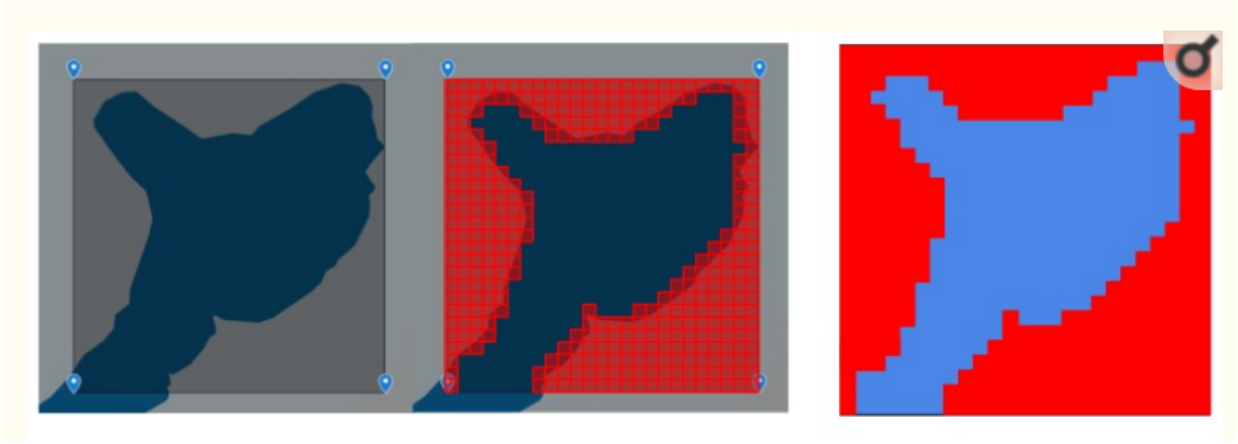

Figure 4.10: Demonstration of how the sailing area will be divided into a grid of sailable nodes. (Gonçalves da Silva Junior, 2020)

With this, the boat then determines the current true wind direction, its current square, and the destination or destinations. The boat then uses a modified A* algorithm to pathfind from each given GPS coordinate to the next to create a path around the course. The A* algorithm will choose the most optimal path to the destination using a heuristic equation that first based on the wind direction will set a weight to each 8 directions the boat can travel. These weights correspond to the speed the sailboat sails at, going a particular direction with respect to the wind. The weights are shown visually in figure 4.11. In all four examples it can be seen that going directly into the wind

is impossible so the weight will be extremely high. Heading 45° degrees into the wind either way is slow so it is represented by the orange arrows and heading directly downwind is similarly slow so it will be weighted the same. The reach position which is shown in light green (perpendicular to the wind) is a favorable direction so it will have less cost, and the most optimal sailing direction is broad reach (135° from the wind direction) which is shown in dark green below. This cost is written in the heuristic as *Speed W.R.T Wind Cost*.

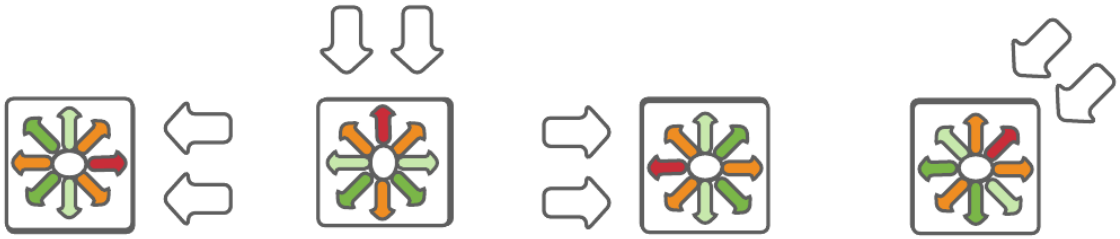

Figure 4.11: Wind Direction Speed Cost Diagram

Then based on the direction of the last movement a weight will be assigned to each direction, favoring continuing in the same direction, with the weight increasing as the boat has to turn more. This reduces the number of times the boat tacks while traveling upwind. This cost is represented in the heuristic as *Continuing Direction Cost.* The final part of the heuristic will be the diagonal distance from the node to the goal. This is calculated by taking the maximum of the absolute value of the difference between the x coordinate of the current node and the goal and the y coordinate of the current node and the goal. The pseudocode for this calculation is shown below:

$$Diagonal\ Distance\ =\ maximum\{|\ current\ x\ -\ goal\ x\ |,|\ current\ x\ -\ goal\ x\ |\}$$

This diagonal distance accounts for the eight directions of travel that the boat is capable of. This heuristic should lead to an A* algorithm that closely mimics the behavior of a real sailor. The Final heuristic is shown in the equation below

$$Cost = Speed\ W.R.T\ Wind\ Cost + Continuing\ Direction\ Cost$$

$$Heuristic = Cost + maximum\{|current\ x - goal\ x|, |current\ x - goal\ x|\}$$

The diagram below shows a course plotted from the green starting square to the final red square with wind in the westward direction around 4 buoys. The buoys here are shown in the orange squares.

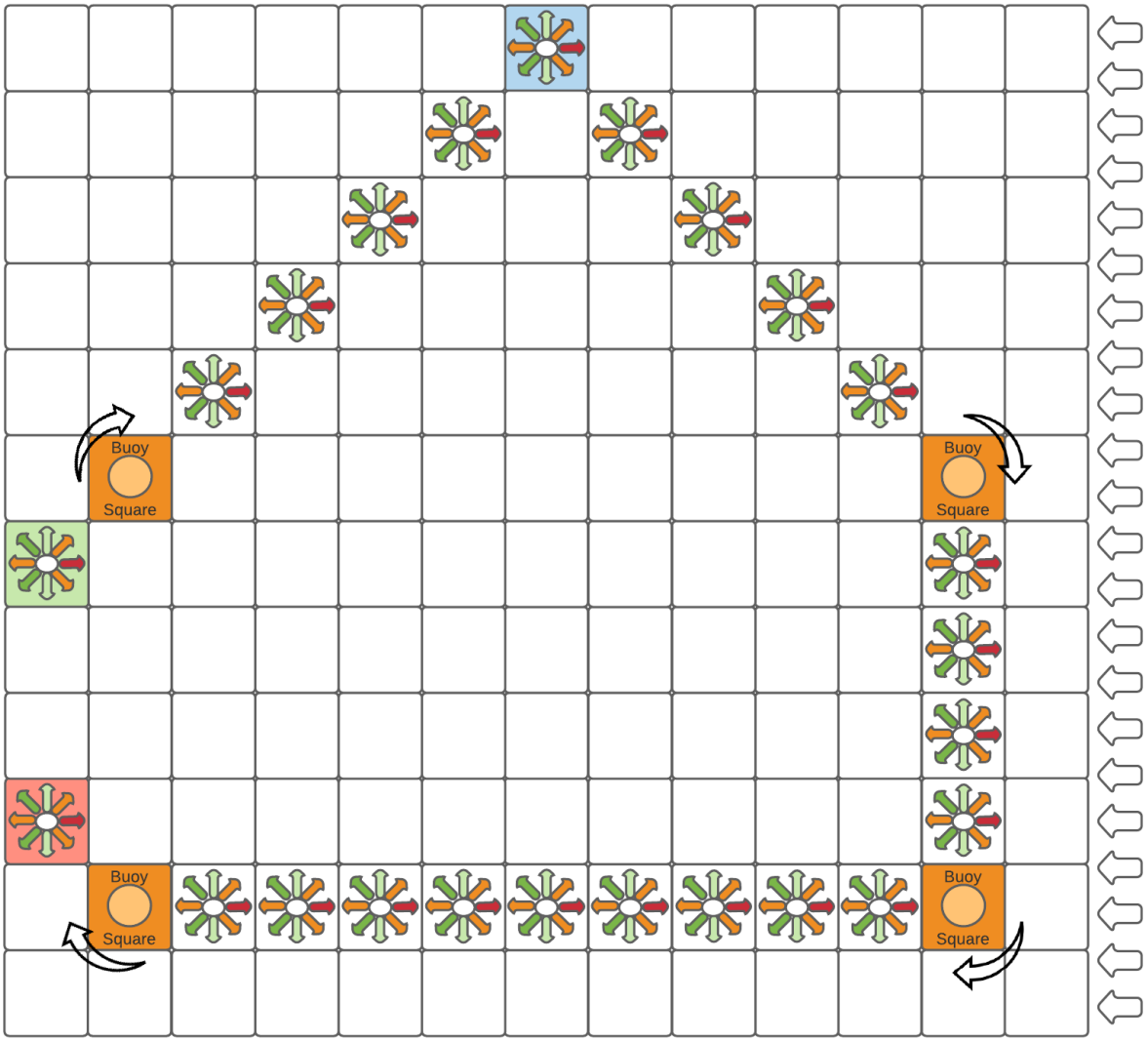

Figure 4.12: Example of High-Level Algorithm generation course. This example course requires sailing from the green square to the red square, around the orange buoy squares, with an easterly wind.

The algorithm additionally removes all intermittent points, leaving only the points where direction is changed, as seen by the blue highlighted square. By removing the intermittent points, the number of waypoints which must be reached successfully are reduced, reducing the possibility of the boat missing a waypoint. The blue waypoint and buoy squares are then given to the point-to-point algorithm as a series of points. When the boat enters a "buoy square" the boat will approach the buoy's GPS position within the square and then identify the buoy using the boat's

vision system. Then, based on the location of the next waypoint relative to the buoy the boat will perform a rounding maneuver to the port or starboard side and continue onto the next waypoint. When the boat enters a regular waypoint, it will move toward it's GPS coordinate until it is a certain distance away before moving to the next point. The logic for the pathing algorithm can be seen in the flowchart below.

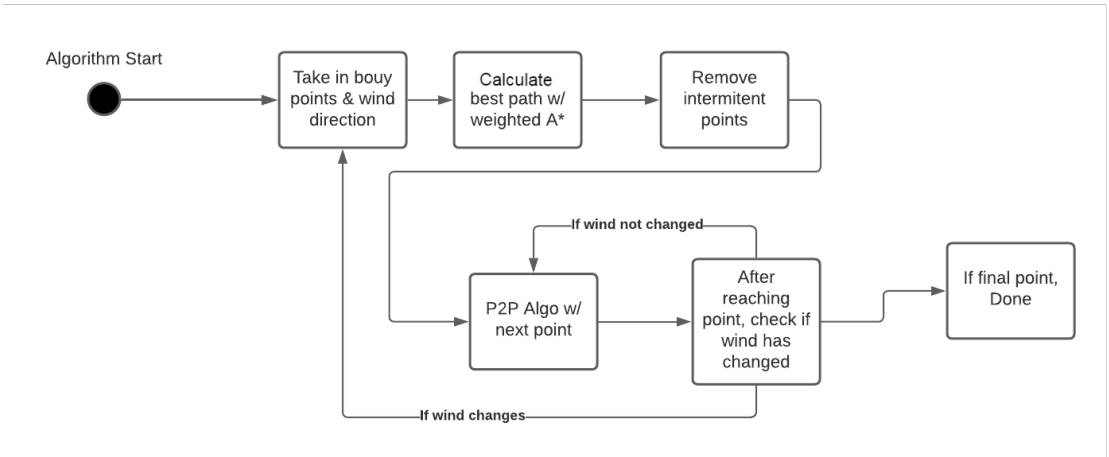

Figure 4.13: High Level Pathing Algorithm flowchart

4.3.3.2 Point to Point Algorithm

Our algorithm which handles sailing directly from one point to another works by taking inputs for the current location of the boat, a desired location, hull compass direction, the true tracking of the boat, and a vector representing relative wind angle. The reference frame of the boat is set such that the direction of true wind is at 180^{o}. An intended trajectory is calculated from the current location to the desired location and then compared to the reference frame. The trim-tab

state is set depending on the angle of the trajectory on the reference frame. Because sailing is dependent on the wind direction, if the wind changes by a significant amount, the path between waypoints generated by taking the previous wind into account will lead to an undesirable, unoptimized path for the boat. So, if the point-to-point algorithm is running and the wind direction changes more than 20 degrees, for a period of 20 seconds, the boat will keep its heading momentarily and then recalculate the A* algorithm again to find a path suited to the current wind conditions. The A* algorithm will also be recalculated if a couple other conditions are satisfied. If an obstacle is detected, such as a moving boat, A* will be recalculated with the obstacle's trajectory and COLREGS (Sailing regulations to prevent collisions) factored in. The next condition for recalculation is when the boat drifts over ⅓ of the distance off course from the projected trajectory between the starting point and the destination. Additionally, if a point that was not originally upwind becomes an upwind point due to wind change or drift, the A* will recalculate rather than attempting to tack. The final condition is when the wind or boat trajectory changes so the destination is upwind of the boat. All of these conditions will exit the point-to-point algorithm, recalculate A* with the current waypoints, and then return to the point-to-point algorithm with a new destination. An extensive flowchart of the algorithm can be seen in figure 4.14.

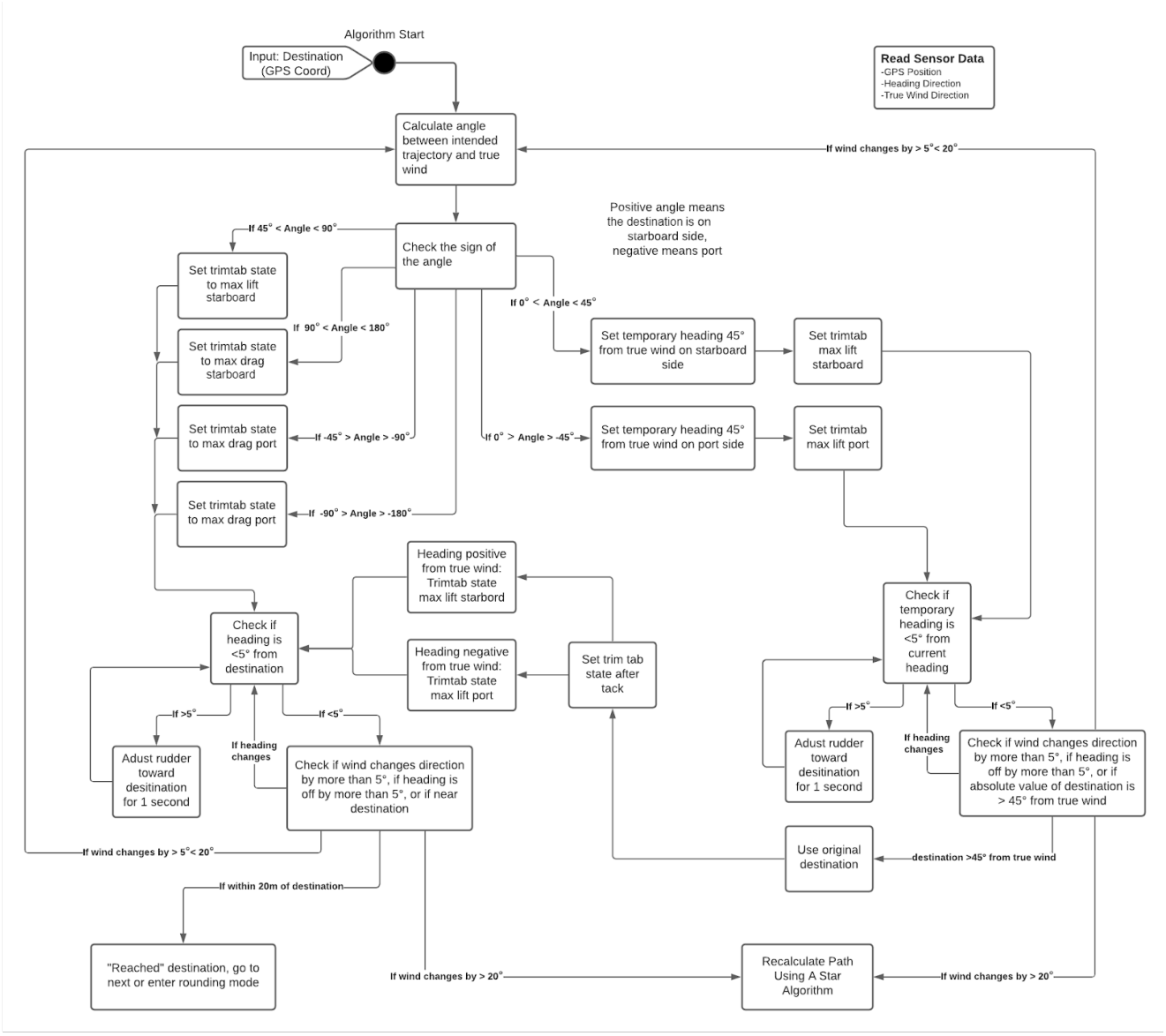

Figure 4.14: Point to Point Algorithm flowchart

The algorithm was simplified down to a three-state system, which can be seen in figure 4.15. In this version state 1 adjusts the trim tab, state 2 follows a temporary heading when going upwind, and state 3 heads toward the destination. The flow of states when not heading upwind is

state 1, state 3, done. The flow of states when heading upwind is state 1, state 2, state 1, state 3, done.

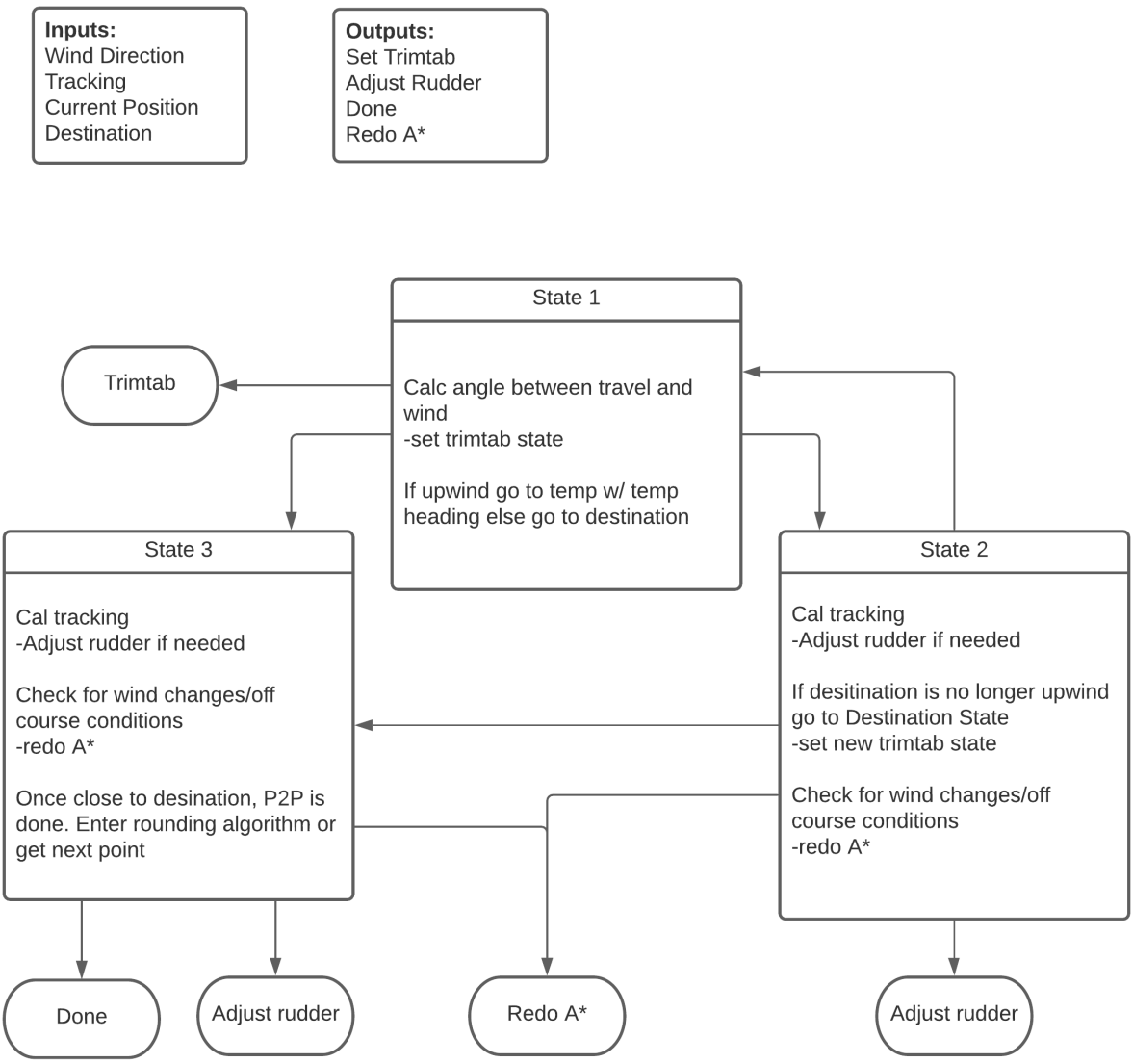

Figure 4.15: Three State Point to Point Algorithm flowchart

The implemented algorithm more closely follows this second version, while taking into account the conditions mentioned in the previous version. In implementation, The control system repeatedly gets an action from the algorithm and executes it, before feeding in new data and getting a new action. The action for state one determines the trim tab state in a similar way to the semi-

autonomous function discussed in section 4.3.1.6 and sets it before moving to state two/three depending on the wind direction. The action for state two checks if the destination is upwind, and if not, moves back to state one. Otherwise, the action in state two will return any rudder adjustments to keep following the temporary heading. In state three, the vector toward the destination is calculated, and compared against the tracking data to determine a deviation from the course, which then adjusts the rudders. The rudder adjustment is incremental, increasing in angle as the boat deviates for longer, and resetting when the boat is back on track. The third state also checks its distance to the destination and returns a done status when close to the destination. In the future, this will ideally be when buoy rounding is performed.

4.3.3.3 Ballast Control Algorithm

The Ballast Control Algorithm works to keep the optimal roll angle of 20° leeward to minimize hull drag while maximizing rudder effectiveness. To ensure this, we determine an ideal roll angle of 20° leeward based on the wind direction and continually check the boat's current roll against it, adjusting the ballast by 5° increments to attempt to maintain an ideal trim within a margin of ±5°. If the ballast reaches its maximum position and is still unable to stabilize the roll angle the trim-tab is instructed to reduce lift until the roll angle can be restabilized. A flowchart of the Ballast Control Algorithm can be found below.

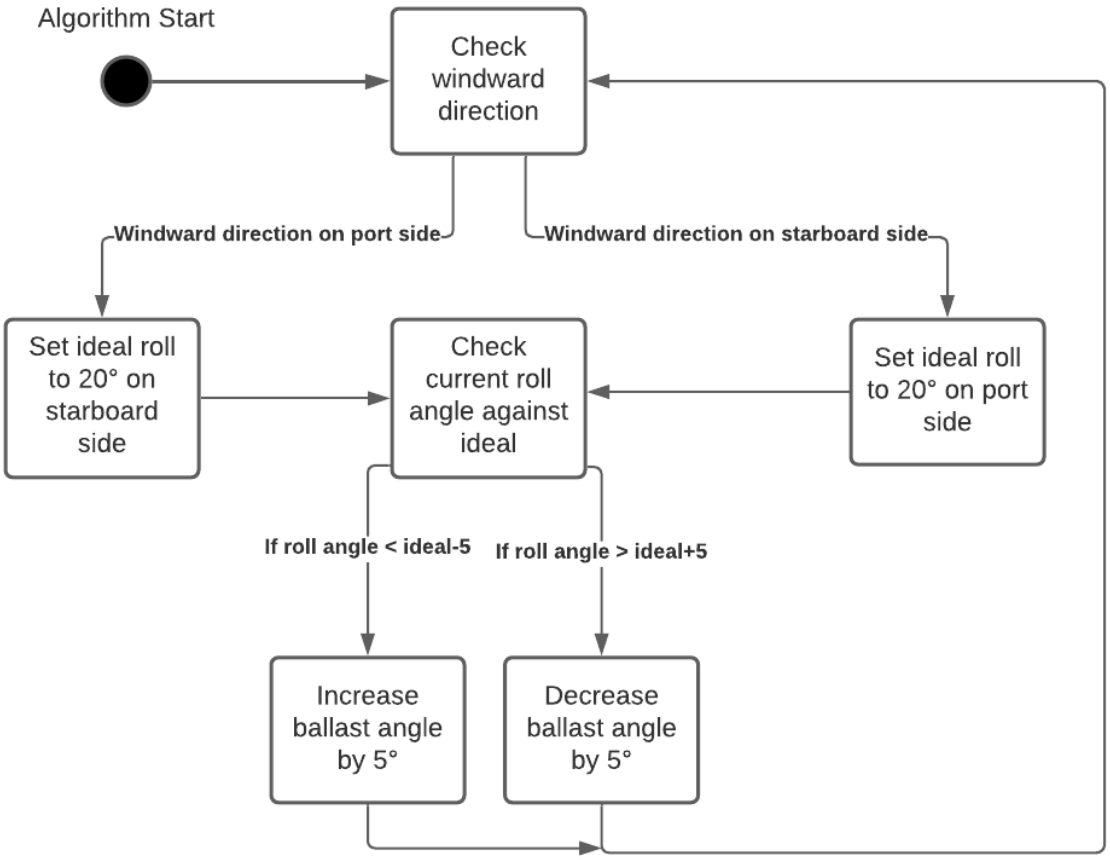

Figure 4.16: Ballast Control Angle Algorithm flowchart

5. Results

This section is divided into five subsections based on the project goals we defined at the beginning of the project; these are outlined in section 3. Each subsection will cover our successes, failures, and shortcomings in achieving those objectives.

5.1 Full RC Control

Our first project goal was to have full RC control to allow a human operator to navigate a closed course. This project goal was achieved and exceeded. Using RC control a human operator can navigate the boat either fully manually or semi-autonomously. Semi-autonomous control allows the operator to offload ballast and trim-tab control to the autonomous systems while still maintaining full control over the rudders. The boat has been demonstrated to be able to reach all points of sail while under RC control as well as perform tack and jibe maneuvers.

5.2 Autonomous Point-to-Point Navigation

This project goal required having the SailBot autonomously navigate between two points placed a maximum of 50 meters apart with an accuracy of within 3 meters of the target point. This project goal is still in the refinement process and cannot be considered fully achieved. The algorithms are both completed but not completely integrated or tested. The high-level pathing, A* algorithm is completely implemented in its own module that generates GPS points for the boat to sail to. The algorithm still needs to be able to regenerate a list of points whenever certain conditions are met. The point-to-point algorithm must still be debugged to be tested. Both algorithms have not been tested on the water.

5.3 Autonomous Buoy Rounding

This project goal simply called for the SailBot to autonomously round buoys. This objective will likely not be completed before the conclusion of this year. Most of our work on this objective involved collecting and preparing resources that can be used by future teams to build a computer vision system. A Google Photos album containing our training data will be passed on to next year's team. Additionally, we identified the 2019 K.O.L.T.: Known Object Localization and Mapping MQP as a promising starting point for onboard computer vision to achieve this and other vision-based project goals.

5.4 Endurance Sailing

This project goal called for the SailBot to continuously sail a predetermined closed pattern for a minimum of six hours. This objective currently remains untested due to the prioritization of other project goals. However, the fundamental components to achieve this project goal are currently in place. The greatest potential untested factor preventing us achieving this goal is the battery life of the trim-tab and the hull. The hull's battery life has been calculated to be more than capable of powering the boat for a full six hours, although is untested. The trim-tab's power consumption is extremely variable and therefore has not been thoroughly examined to determine if it can operate for the full duration and is also untested. Additionally, the autonomous point-to-point objective, as described in section 5.2, must be completed in order for us to test this goal without solely relying upon manual control. Through testing it was found that no water enters the hull, so it is mechanically capable of staying on the water for a minimum of six hours.

5.5 Search and Rescue

Our final project goal requires the SailBot to sail a search pattern within a predetermined 50-meter diameter circle in order to locate a randomly placed buoy, then signal the GPS coordinates of the buoy to shore. The completion of this goal requires the completion of both the autonomous point-to-point objective, section 5.2, as well as the autonomous buoy rounding objective, section 5.3, and as such will likely not be completed by the end of this year. In addition to the completion of these two prior goals this goal requires a stable connection between the SailBot and the shore, which has been reliably established.

6. Future Work

This year our team kept a strong focus on sustainability of the SailBot project. Our goal was to make our successors transition into the project as painless as possible. Following this goal, we outline here the vision we have for this project in the coming year.

6.1 Finishing Uncompleted Project Goals

Our project goals turned out to be more ambitious than we were reasonably able to achieve and as such we were not able to complete them all.

6.1.1 Computer Vision

Implementing reliable computer vision aboard SailBot is a big leap forward to achieving full autonomous sailing. In order to set up next year's team for success we purchased and mounted a Stereolabs Zed 2 stereoscopic camera. We chose this camera since the K.O.L.T. MQP was designed for this camera and with it can tackle the complex problem of determining object distance and location.

The implementation of computer vision will allow for the completion of both the buoy rounding and search and rescue project goals. Additionally, it will allow for the development of more advanced sailing algorithms with the ability to avoid both static and moving obstacles, ex. other boats and docs.

Our team suggests that the computer vision be implemented in its own ROS node, which will communicate directly with the control system to identify objects and their position. This way, the control system can pass the relevant information to the buoy rounding algorithm/search and rescue algorithm which the control system will ideally manage, just like our other algorithms.

6.1.2 Refine Sailing Algorithms

Sailing is a deceptively complex activity and as a result there are always additional variables that were not originally considered. Our implemented sailing algorithms attempt to take into account as many factors as possible, however we were not able to verify that all were covered sufficiently. We suggest that the following year's team considers improving our algorithms by testing further and reevaluating our logic. The state of the algorithms is explained more in the following paragraphs.

The ballast algorithm currently is unable to work with its intended 20-degree angle of roll, as the ballast at full extension does not consistently move the boat to 20 degrees. This can be fixed by either reducing the roll of the ballast algorithm to a lower value, which is unideal, or by increasing the ballast's weight as intended. The ballast was built with more weight in mind but has been reduced to the lack of electronic motion limiting, as well as the instability of the current gearbox. Improving these is discussed in 6.1.3. Given the physical state of the ballast, the algorithm has been reduced to a roll of 12 degrees for testing. Hall-effect sensors could also be implemented to stop the ballast once it reaches its full range, preventing the motor from attempting to move the ballast outside of its range if it has not yet hit the ideal angle of roll.

The point-to-point algorithm is nearly complete. The algorithm must first be debugged and completely integrated into our control systems node to work as intended. The values for things such as the heel angle, points of sail angles, and rudder angles must also be tuned to work with our components. After these things are tuned the algorithm must be tested on the water to be able to navigate all the points of sail autonomously. Lastly, some of the triggers to back out and recalculate the A* will need to be implemented, i.e., the wind angle changes significantly for a longer duration.

The A* algorithm has a bit more work to be considered complete. The algorithm outputs the correct set of points that the boat must sail, but the points are not currently in use. The algorithm must be integrated into the point-to-point algorithm so we can determine when we need to recalculate the course, i.e., when the wind changes or the boat is significantly off course. The algorithm must also be tested before use. Lastly, we must integrate the option to sail the different events that are involved in the SailBot competition.

6.1.3 Rebuilding Ballast Gearbox with Electronic Limiting

The movable ballast system has been a problem on the boat for many years and it needs both mechanical and electrical improvements. When the ballast was attached it was found there is a significant amount of backlash between gears. It was also found during testing the movable ballast swings very quickly, reducing the gear ratio would allow the ballast to move slower and in a more controlled manner. Both of these mechanical issues lead to the suggestion of rebuilding this gearbox to reduce the backlash between gears and reduce the ballast speed. Through testing it was also found that the movable ballast had a hard time getting the boat to 20 degrees of roll to be in perfect trim so a heavier ballast might be needed to reach this roll angle.

An additional problem with the ballast is that there is no sensor to tell what angle the arm is at nor a sensor to relay if the arm has hit its physical limits. The easiest solution to this would be to attach physical limit switches or hall-effect sensors to send feedback when the arm has reached its limit. An encoder on the arm would also provide feedback and avoid the problem of the arm swinging past its limits and over torquing the gearbox.

6.2 SailBot Upgrades

SailBot is a complex robot with many interacting parts that have experienced a varying degree of refinement. Because of this, there are many upgrades that can still be done to improve the reliability and effectiveness of the robot. This section will cover our recommendations for where next year's team should begin.

6.2.1 Rebuilding Airfoil Mast

During an on the water test the carbon fiber tube that runs the length of the sail mast snapped. From our analysis this was due to several factors, most influentially the relatively small size of the tube. In order to allow us to continue to test the boat without rebuilding the airfoil we inserted an aluminum rod into the broken section of the carbon fiber as a temporary fix.

Although this solution does work it is far from elegant or permanent. We recommend that the following year's team conducts a stress test of the mast and rebuilds the full airfoil with a more appropriately sized carbon fiber rod. We assume the last rod broke because a hole was drilled exactly in the spot with the highest stress (just below the sail), so if a rod with a significantly wider diameter would not snap.

6.2.2 Refining SailBot's Draft

When the current hull was designed SailBot utilized sealed lead-acid batteries to power the hull. Since then, the batteries have been switched to much lighter and more powerful lithium-polymer batteries. This, however, came with the unintended consequence of altering the boat's current draft to such an extent that the transom does not sit even to the waterline as the bow is significantly lighter than the boat's stern. The solution to this issue is relatively straightforward,

either more weight needs to be added to the bow or weight from the stern compartment needs to be moved forward to the bow compartments.

6.2.3 Rebuilding the Trim Tab

The trim tab has been a continual source of strife between the boat and our team. The trim tab's control housing must be able to communicate with the hull and correctly adjust the trim tab angle, while also remaining as light as possible. These constraints have led us to the current functional iteration, however there is still more to be desired. The foremost issue that needs addressing is the lack of rigidity in the rod that supports the trim tab and connects it to the trim tab housing. This rod often bends and twists in real world tests. We advise the future team to consider, when rebuilding the trim tab housing, to use a square tube that spans across the trim tab housing to connect both the trim tab as well as the relative wind encoder. This will increase stability throughout the housing.

Additionally, next year's team should take a close look at the current draw of the trim tab servo in regard to the capabilities of the battery. We are of the belief that under high strain conditions, and when the battery is not fully charged, it is likely that the servo's current draw is resulting in a microcontroller brownout. Although we have not confirmed this through testing, we recommend next year's team attach a capacitor to the servo in order to smooth out current fluctuations.

6.2.4 Refine Connection to the Trim Tab

The wireless socket the Arduino MKR 1010 uses to communicate with the Jetson Nano is an extremely convenient feature of SailBot, however it is not the most robust connection and occasionally disconnects. This disconnection is a problem that our team has attempted to address

but has been unable to fully resolve. We suggest that our successors begin with converting the Wi-Fi connection to Bluetooth.

A switch from Wi-Fi socket communication to Bluetooth Low Energy (BLE) communication with a central and peripheral device would provide several advantages. Bluetooth excels in many of the Wi-Fi socket's shortcomings, mainly dropping clients and reconnecting, and more reliable communication within a 30m range and through walls. The MKR 1010 used in the trim tab currently has BLE capabilities, as well as a supporting library, ArduinoBLE. If using BLE, the MKR could create a BLE service with characteristics for the trim tab state, relative wind indicator angle, and the servo angle, which the jetson could connect to once the service is advertised by use of a UUID. This allows a two-device connection which will reconnect without regard to a "client" besides use of the correct UUID and would effectively pass all the necessary information.

If a robust wireless solution cannot be developed, we suggest they consider exploring the use of a hardwired barrel jack RX/TX connection. This connection would run the length of the mast to connect the hull to the trim tab via an RX/TX connection. Such a connection would still use a dedicated power supply and micro controller for the trim tab, but would eliminate the need for a wireless connection, while still allowing the airfoil free rotation. This solution however would require waterproofing, which adds an additional level of complexity.

7. Bibliography

- Silva Junior, A. G., Henrique dos Santos, D., Fernandes de Negreiros, A. P., Boas de Souza Silva, J. M., & Gonçalves, L. M. (2020). *Image that will be used to generate the blocks matrix* [Digital]. High-Level Path Planning for an Autonomous Sailboat Robot Using Q-Learning. *Sensors, 20*(6), 1550. doi:10.3390/s20061550

8. Appendices

8.1 FrSky Taranis Remote Control Labeling

1. The “State 1” switch, which toggles the RC and semi-autonomous modes further described in section 4.3.1.1. When flipped toward the user the boat will be in RC mode, when flipped to the middle or far position, it will enter semi-autonomous mode.
2. The “State 2” switch, which toggles the boat into autonomous mode. Similarly, to switch 1, when pointed toward the user the boat is in RC, when put to the middle or far position, it will enter autonomous mode.
3. The left right movement of this stick controls the rudders on the boat when in RC or semi-autonomous modes. Controls are inverted: pointing the stick left will move the boat right.
4. The up down movement of this stick controls the trim tab position in RC mode. The position of the stick directly translates to the trim tab position: when the stick is at the highest position the trim tab will be all the way to one side, when the stick is at the lowest position the trim tab will be all the way to the other side.
5. The left right movement of this stick controls the moveable ballast. The ballast will move left or right as long as the stick is not centered. Thus, holding the stick left/right will move the ballast until the user stops holding the stick left/right. When looking from the stern of the boat towards the bow moving the stick left will move the ballast left (port) and moving right will move it toward the right (starboard), this will cause the boat to roll to the port or starboard side respectively.

8.2 GitHub Repository

All of the software for this project can be found on the wpisailbot GitHub under the sailbot20-21 repository found here:

https://github.com/wpisailbot/sailbot20-21

Extensive documentation can be found on the repository by navigating between its directories and reading their relevant readme files.

8.3 Pre-Test Checklist

This is the list compiled by this year's team of things to bring to a water test.

- Taranis Controller (charge overnight)
- Taranis Charger
- Charged hull batteries
- Charge and assemble trim tab
- Laptop w/ telemetry
- Laptop charger
- Electrical power strip
- Extension cord
- Boat
    - Hull
    - Keel
    - Sails
    - Rudders
    - Trim tab
    - Ballast
    - Airmar
- Fastener box
    - Keel bolt
    - Sail dowels (1 wood, 1 carbon fiber)
    - 2 rudder screws
    - 2 ballast screws and gear
    - 6 Airmar screws and washers
- Micro-USB
- Telemetry module
    - Ethernet bridge
    - Power and ethernet cord
    - Ethernet cord
    - Ethernet to USB
    - Barrel jack power
- Buoy
- Toolbox
    - Screwdriver
    - Allen keys
    - Pliers

    - Wire & cutters/strippers
    - Other tools may prove useful
- Duct tape
- Multimeter
- Hull stand
- Life jackets
- Super glue