Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Tom-Are Eidal

Advancements in Control System, Hardware, and Testing for an Autonomous Sailing Boat

Master's thesis in Engineering and ICT
Supervisor: Andreas Echtermeyer
June 2023

NTNU
Norwegian University of
Science and Technology

Tom-Are Eidal

Advancements in Control System, Hardware, and Testing for an Autonomous Sailing Boat

Master's thesis in Engineering and ICT
Supervisor: Andreas Echtermeyer
June 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Abstract

Autonomous surface vehicles have proven themselves to be great data collectors. In the need of understanding a changing environment, these vessels are a great option. A combination of high mobility and openness to customization, together with low power consumption and risk makes these the perfect candidates. The end result of this project aims to be one of these vessels, and to be an environmentally friendly, independent collector of important data from large areas and remote locations.

Previous teams have come a far way in developing a boat ready for sailing the rough seas. A catamaran hull was designed for low power consumption and high mobility. A sail has been developed to catch the wind to generate propulsion. A keel and a rudder was added to control the boat and increase its accuracy and mobility. Last year, a control system was made to combine all these into an operable boat, able to sail on its own.

This thesis picks up right after the development of the control system. Several issues have been found in the existing system. Further tests have been conducted to explore these, and to uncover any other problems. New solutions have been found and implemented into the system. This report goes through these solutions and their reasoning, anchored in the necessary theory. The solutions have also been verified by testing the affected subsystems.

With the changes to solve existing issues, further progress could be made. The range of directions of sailing depending on the wind has been extended to all angles with the addition of tacking capabilities. On its mission as a data collector, the boat will need reliable sensors to catch useful data. A system for implementing new sensors is found, and showed in this report. This includes the steps needed to upload and log the data for later use or distribution. Lastly, testing availability has been greatly increased. Further development will require complex avoidance and guiding systems, which require a lot of testing. A land-based version of the boat built on a trolley with wheels, as well as movement simulation has been added for this purpose.

Sammendrag

Autonome overflatekjøretøy har vist seg å være gode datasamlere. Disse fartøyene er i dag et godt valg på grunn av behovet for mye data for å forstå det endrende miljøet. En kombinasjon av god mobilitet og tilpasningsmuligheter, sammen med lavt energiforbruk og lav risiko, gjør disse fartøyene til perfekte kandidater. Sluttresultatet av dette prosjektet sikter på å være en miljøvennlig, uavhengig samler av viktig data fra store områder og utilgjengelig steder.

Tidligere grupper har kommet langt i utviklingen av en båt som er klar til å seile på havet. Et katamaran-skrog har blitt designet for lavt energiforbruk og god manøvrerbarhet. Et seil har blitt utviklet for å fange vinden for fremdrift. En kjøl og et ror har blitt satt på båten for å styre den og øke dens presisjon og manøvrerbarhet. I fjor ble et kontrollsystem laget for å kombinere alle disse elementene til en fungerende båt som kan seile av seg selv.

Denne avhandlingen tar over rett etter utviklingen av kontrollsystemet. Flere problemer er blitt oppdaget i det eksisterende systemet. Tester har blitt gjennomført for å utforske disse nærmere, og for å avdekke eventuelle andre problemer. Nye løsninger har blitt funnet og implementert inn i systemet. Denne avhandlingen går gjennom løsningene som er funnet og forklarer grunnlaget for de. Løsningene ble også testet og verifisert ved egne tester designet for de påvirkede sub-systemene.

Etter at løsningene på eksisterende problemer ble implementert, åpnet muligheten for videre utvikling seg. Vinklene båten kan seile til som følge av vindretningen har blitt utvidet til å gjelde alle retninger ved å få "å gå over slag" (engelsk: tacking) til å virke. På oppdraget sitt som datasamler trenger båten sensorer for å innhente nyttig data. Et system for å legge til flere sensorer er funnet og vist i denne avhandlingen. Dette inkluderer også opplasting og logging av dataen for senere bruk eller utsendelse. Til slutt har tilgjengeligheten for testing vært et viktig punkt å forbedre. Videre utvikling av båten vil kreve avanserte systemer for å styre unna objekter og ha bedre navigasjon. Dette vil kreve mye testing for å verifisere. En versjon av båten som kan kjøre på land er laget nettopp for dette. I tillegg har kode blitt lagt til for å simulere bevegelse.

Acknowledgment

I would like to thank supervisor Dr. Andreas Echtermeyer for introducing me to the project, and the support provided through the project. A big thanks to last year's team, Adrian Skogstad Pleym and Magnus Westbye Ølstad for their help in understanding the project and making progress. I would also like to send out a special thanks to Hanna Aksetøy Aalmen for endless support and good ideas during the project.

Table of Contents

List of Figures

List of Tables

1 Introduction

Autonomous surface vehicles have over the recent years proven themselves capable of being great data collectors. Largely enhanced by modern technology, power consumption and size can be smaller than ever, and the capabilities larger. The use of this makes it possible to completely replace human crew aboard vessels. This can extend mission time and independence, creating a sensor station able to collect data for years. Without the risk of human health, harsh environments can be traversed. Together with no check ups needed, missions can be long lasting and cover great areas of interest.

We are currently witnessing a growing pattern where more extreme weather and rapidly changing environments are expected to occur. To understand the changes, and be as prepared as possible, a considerable amount of data is needed. Catching it early means data collection from remote, cold places, heavily exposed to the weather. Simultaneously, green energy has never been more important. An autonomous surface vehicle can satisfy both worlds. Small size and low power consumption makes it a smarter choice than huge vessels with human crew. Its robustness and mobility enables it to cover large areas, and to be where it is needed. With technological development, customization for current needs will increase, together with its range, operability and efficiency.

The goal of the project of which this thesis is part of, is developing such an autonomous surface vehicle. In its final state, it needs to be capable of surviving travels between mid-Norway and Iceland. This is an area that can have bad storms, and carries a lot of naval traffic. Thus, requirements to operability and object avoidance is necessary. Testing is of great importance when developing such solutions, and will be focused on during the project. Previous work, as well as the objects covered and the scope of this thesis will be further described in the next sections.

1.1 Team and previous work

The project of developing an autonomous surface vehicle was started in 2019 by Dr. Andreas T. Echtermeyer. Master theses on hardware have since then been written by Maria Dyrseth, Sverre Gauden, and Almar V. Brendal. Then, a control system was developed by Adrian Skogstad Pleym and Magnus Westbye Ølstad.

The fall of 2022, Dr. Andreas T. Echtermeyer took initiative to a pre-study thesis on further development of the control system, as well as one to develop a new hull for testing. This master thesis builds on the pre-study of the control system, and the work done in the theses of previous years.

1.2 Problem description

This thesis continues the development of a control system made in 2022. Several issues were discovered during testing and these will be worked on. This thesis focuses mostly on software, but some hardware will be improved as well. As the system and the coming needs are increasing in complexity, facilitating for easier testing will be done. Lastly, a system for data collection with sensors will be started. These focuses are means to reach an autonomous surface vehicle with the operability, robustness and sensory capabilities to complete a set mission.

1.3 Scope of this thesis

This thesis documents the further development of the software and hardware developed over the previous years. It describes the preliminary testing and prototyping steps done to uncover the cause of current issues. A theory base will showcase the best approach for prototyping and the fundamental knowledge base of sailing and the components used. The implementation and reasoning for changes made will be covered. This thesis also aims to guide in practices and use of the boat for further development and project hand over.

1.4 Objectives

The objective of this thesis is to start readying the boat for its mission. This means solving previous issues and starting a system for collection of data of interest. The boat should be able to sail to its target with all wind directions, even coming directly from the target. It should be more resilient to external factors, and the electronic hardware updated as such. The boat should be able to sense and report data from sensors of external interest. Further development should be facilitated for by making testing easier, faster and more available.

1.5 Thesis structure

Chapter 1 introduces the thesis and its scope and objectives,

Chapter 2 provides the necessary knowledge base,

Chapter 3 describes the existing solutions from previous teams,

Chapter 4 shows preliminary testing to explore current issues,

Chapter 5 presents the issue fixes and further development,

Chapter 6 presents the results achieved and verifying tests,

Chapter 7 suggests actions for the development going forward,

Chapter 8 concludes the master thesis and its objective achievements.

2 Theory

In order to further develop the autonomous boat, the theory behind has to be understood. Previous development has build a complex system including a dashboard app as a human-machine interface for communicating with the boat. A lot of complicated electronics is used for sensing the environment and steering the boat. The needed theory for understanding the boat and its development will be presented in this section. Since this heavily relies on the findings of the pre-study report, the majority of the theoretical framework is based on its findings.

2.1 Sailing

Before, sailing was the main way of traveling the seas. In recent times, more reliable and faster methods of travel have emerged, leaving sailing mainly for recreational activities. However, over the last years, a new use of sailing has been found. Long sustaining vessels take advantage through low power needs and a lack of necessity regarding constant speed and check-ups. This is why the vessel of this project is using a sail. A control system to output angles based on position is all that is needed, and the sail will harvest the powers of nature. To understand how to properly use this opportunity, this section will dive into the mechanics of sailing.

2.1.1 Anatomy of a sailboat

Firstly, a few terms and names of parts of a sailboat should be established. The small vessel of this project will still contain most of the sailing mechanics of others roaming the seas, even with its smaller size and simplicity. To start with, we have the *deck*, the top floor of the boat. The *mast* is usually erected straight up from this floor. The sail is held by the mast, but usually not alone (*Anatomy of a sailboat* 2022). A cross beam is usually placed horizontally on the mast, stretching the sail when it is a flexible one. However, this boat uses a stiff, fixed sail and as such, does not need a cross beam.

To ensure buoyancy, the *hull* creates a waterproof and often hydrodynamic body in the water (*Anatomy of a sailboat* 2022). It also helps with control of the boat, together with the *keel* beneath it. The keel acts like a weighted fin, stabilizing the boat with a counterweight and counteracting some of the forces from side wind by acting on the water. A similar, but not weighted, device sits behind it, at the back end of the boat. This is here to steer the boat, and is the *rudder*.

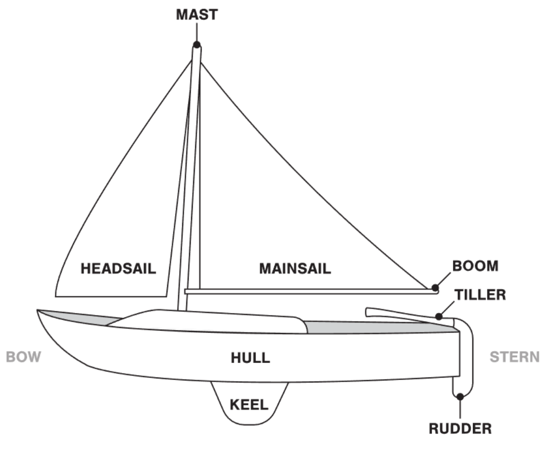


Figure 1: The anatomy of a sailboat.

Source: https://www.sailrite.com/anatomy-of-a-sailboat

On a boat, the four sides also have their own names. The *bow* marks the front end of the boat, while the *stern* marks the back end (*Sailing terms everyone should know* 2022). The left side is called *port*, and the right side *starboard*. With the part naming established, we are now ready to dive deeper into how the different parts are used to harness the wind for propulsion.

2.1.2 Points of sail

As mentioned, the main propulsion of the vessel in this project will be by sail, harnessing the power of the wind. Wind speed and direction are factors we can not control. Regardless, we need ways of ensuring best use of the wind at hand. How to use the wind best is naturally dependent on the wind direction and strength, but also the direction and speed of the boat (*Point of sail* 2022). Following is the different points of sail, denoting how to adjust the sail to the current conditions.

The point of sail denotes the wanted direction of travel with regards to the direction of the wind. It is split into eight intervals, each covering 45 degrees of a whole circle. 0 degrees is assigned sailing straight into the wind (Rousmaniere 1999). From the 0-degrees-line, the scale is symmetrically increasing both directions, until 180 degrees. This denotes sailing directly with the wind.

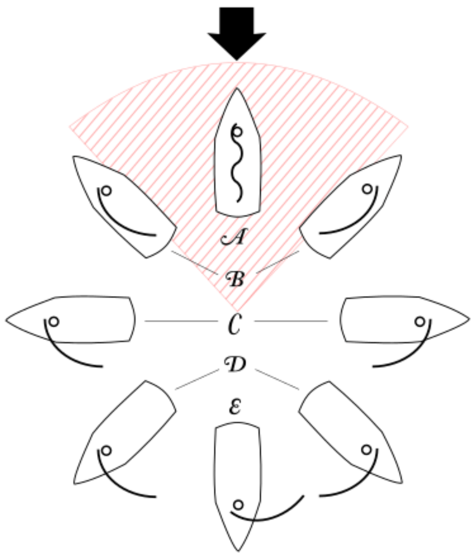


Figure 2: The different points of sail.

Source: https://en.wikipedia.org/wiki/Point_of_sail

Sailing against the wind, at 0 degrees, does not yield any propulsion in the wanted direction. Surrounding the 0-degrees-line, we have a *no-go zone*, usually extending to about 30-50 degrees both ways. The exact number is depending on the properties of the boat (Kimball 2009). Right outside the no-go zone, at about 45 degrees, the point of sail is called *close-hauled*. The best way of utilizing the wind here is angling the sail in the wind much like a wing. This generates "lift" in the wanted direction by creating a low pressure zone in front of the sail (Kimball 2009).

Sailing perpendicular to the wind direction, we have *beam reach* (Rousmaniere 1999). On either side of beam reach, we also find *close reach* and *broad reach*. Close reach is more towards sailing against the wind, while broad reach is closer to sailing with the wind. For all these called reaching, sailing is again mostly generating propulsion from using the sail as a wing. However, as the point of sail transitions from broad reach and further towards sailing with the wind, the sail is more dependent on drag (Jobson 2008). As the point of sail is closing in on 180 degrees, the sail stops working as a wing and starts working more like a parachute. At the end of the transition, where the sailing direction is directly with the wind, we have *running downwind*. At this point of sail, the sail is only generating forward force from the wind by drag.

Table 1: The different points of sail.

No-go zone	Trying to sail against the wind within 30-50 degrees
Close-hauled	Sailing right outside the no-go zone
Beam reach	Sailing directly perpendicular to the wind direction
Broad reach	Sailing between beam reach and with the wind
Downwind	Sailing with the wind

2.1.3 Using the wind for fastest sailing

Without thinking too much about it, running downwind might to many seem like the fastest option of sailing. This is where the direction of the wind and the force, intuitively lines up with the direction of travel. However, for most sailboats, this is not the fastest way of travel (Andy 2016; Bethwaite 2007). At this point of sail, the sailing speed is limited to the speed of the wind. Factoring in drag from the water, and we can not even reach this speed. The highest speed is rather reached when sailing beam reach, perpendicular to the wind. Using the sail as a wing, the achieved forces can propel the boat to a higher speed than that of the wind. Even when the wanted direction is directly with the wind, it can be beneficial to include some degree of reaching. This maneuver is called *beating*, and consists of sailing at an angle a bit off the wanted direction to reach higher speeds. Of course, for the maneuver to be faster than running downwind, the resulting added speed has to beat the added distance of not sailing the direct route.

Benefiting from reaching is also used in the opposite direction, when the target lies within the no-go zone (Cunliffe 2016). In this case, the direct route is not possible, as the sail has no way of generating force in the wanted direction. Instead, we can use *tacking*. This a maneuver where the boat is sailing close hauled, right outside one of the sides of the no-go zone. After sailing some distance, building up enough speed, the boat can tack through the no-go zone and continue sailing clause hauled on the other side of the no-go zone. Sufficient speed is necessary to pass through the wind, only by rudder control (Cliffe 1994; *Sailing terms everyone should know* 2022). In case a lack of speed to pass through, the vessel will run to a halt. The only option is to change direction, possibly somewhat with the wind and away from the target. As speed is regained, tacking towards the target may be tried again.

2.1.4 True wind versus apparent wind

When deciding point of sail, one must differ between apparent and true wind. As sailing gives the boat speed in a certain direction, an opposing wind vector will add to the existing wind. The true wind is the one observed by a stationary object, and can be hinted at by the wave direction on the surface of the water (Jobson 1990). One can also calculate the true wind from the apparent wind by knowing the speed and direction of travel. Adding the opposite of the travel vector to the apparent wind will remove the added wind from sailing, resulting in the true wind. Thus, when deciding on sail angle and point of sail, the opposing vector to that of the travel has to be subtracted from the measured wind to figure out the true wind.

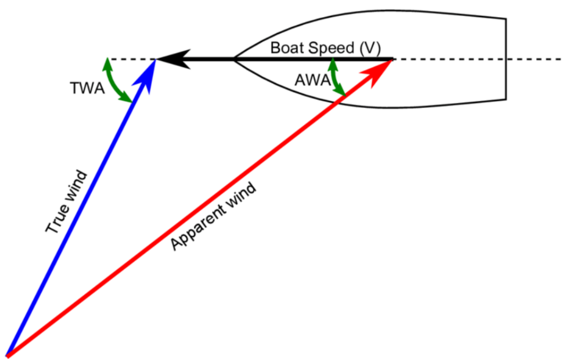


Figure 3: True versus apparent wind.

Source: https://www.researchgate.net/figure/Apparent-and-true-wind_fig6_257143505

2.1.5 Sources of sailing errors

When sailing, a boat is reliant on compass courses and some form of map to navigate. This sounds good, but they are subject to errors. Deviations in course, speed or unwanted movements can make travels totally miss its target. Most errors can be compensated for by digital aids, like GPS. By checking the current position with the wanted position, correcting actions can be done. Yet, it is still good to know of the errors that can occur.

When using a compass for navigation, two effects can throw the boat off course. The first being when the compass is pointing in the wrong direction, called *variation* (*Misvisning og deviasjon* 2022). The compass is functioning on magnetics, and should point to the north pole. However, the magnetic pole of the earth does not align with the actual north pole (Holtet 2022). This throws the compass off in varying degrees depending on where on earth's surface the vessel is traveling.

The second effect that can alter the course is *deviation*. This also makes the compass stake out the wrong course, but this time due to disturbances in its surrounding magnetic field (Kjerstad 2022). Electronics or cargo aboard, or the materials of the boat itself can create magnetic fields. This affects the compass by dragging it in other directions than that of the magnetic field of the earth. It can be compensated for by making sure the conditions are consistent and account for this interference. This depends on the local magnetic fields not to be too strong. A better way, and one that should be followed when possible, is to place the compass away from interfering parts and cargo.

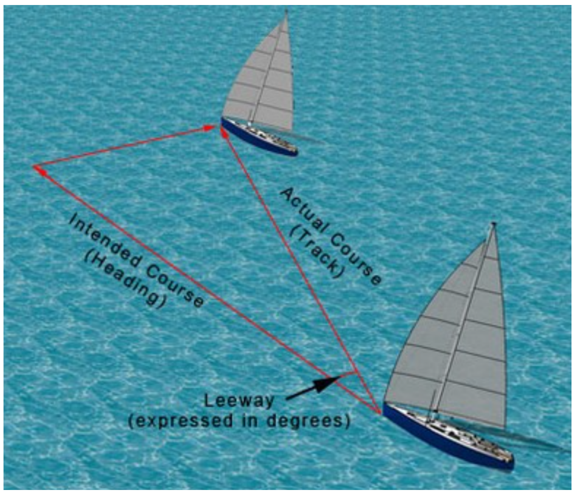


Figure 4: Faulty course due to leeway.

Source: https://www.sail-world.com/Australia/Measuring-your-leeway-for-better-sailing/-71221?source=google

Lastly, the effect of forces acting perpendicular to the direction of travel is called *leeway*. Wind on the sail and waves pushing from the side will push the boat in an unwanted direction (Osnes 2022). When measuring the speed in the direction of travel only, leeway is not detected. The shape of the hull and the keel will resist the unwanted movement, but will not completely neglect it. Over longer travels, leeway can build up to huge errors. Steering slightly against the forces, or compensating using GPS can be done to neglect the effect.

2.2 Control electronics

For the boat to be autonomous, it is crucial to have sensors to take in variables from the environment, and actuators to act on them. Necessary sensors are installed on the boat from previous work, and their use cases are described in section 3. The sensors are read, and the data is used by small computers, or microcontrollers, aboard the boat. This section will look at how these microcontrollers are used to operate the boat.

2.2.1 Raspberry Pi and Arduino

For main control of the boat, the brain, a Raspberry Pi 4 is used. It is a small single-board computer (*About us* 2022), originally meant for educational purposes. Because of its simple use in areas like weather monitoring and robotics, it has grown in popularity. These areas are also great for our purpose in this project. The Raspberry Pi was chosen because of its capabilities, price and easy-to-use modularity (Upton 2022). It hosts many interfaces by default, including USB and HDMI ports, as well as ways of connecting it to networks. A huge advantage is also the fact that it is a computer, making it possible to run custom software of our own choosing on it.

Although a lot of options are available, the Raspberry Pi is by default running the Raspbian operating system (*Raspberry Pi OS* 2022). It has a lot of similarities with Linux, and is controllable with most of the same text commands. Raspbian also supports Python, used extensively in this project. Together with the capability of running custom code, and the default interfaces, equipment like GPS and 4G modem is easily connected for tasks on the boat.

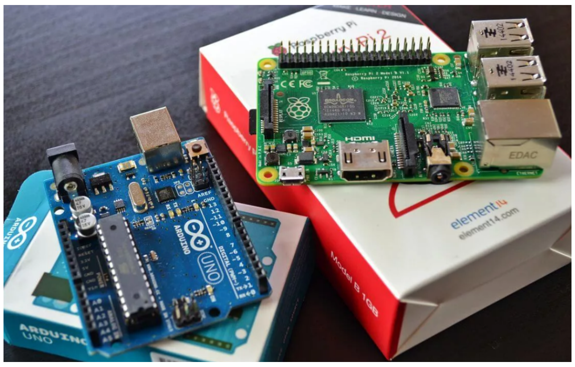


Figure 5: A Raspberry Pi and an Arduino Uno.

Source: https://pimylifeup.com/raspberry-pi-vs-arduino/

Using the interfaces on the Raspberry Pi, two Arduinos are also connected. These split up the system in more manageable chunks, and help communicate with motor controls and sensors. Arduinos are single board microcontrollers built with both input and output pins (*What is Arduino* 2022). While not as open as the Raspberry Pi, they can still run small premade programs. This makes them able to receive input data, act on them, and send output data to control. This is done through built-in input/output (i/o) pins (*What is Arduino* 2022). These are individually read or controlled pins, used for monitoring and controlling actuators and sensors on the boat.

2.3 Sailing control

Traditionally, sailboats have been reliant on people for observing the conditions and adjusting control surfaces accordingly. Essential to this project is to avoid this, and replace people with a limited amount of sensors and power. Luckily, there have been developed algorithms for effectively sailing autonomously. This section will look further into the chosen solutions to control the boat in this project.

2.3.1 Path planning and course control

To calculate the travel to the next target, the boat needs to know two things. The first is where it is now, given as a set of coordinates. The second is the position of the target (Saoud et al. 2015). In the long run, the boat should also know of any obstacles on its path, including heavy weather. With the current system, deviation and variation are accounted for, but leeway will have to be compensated for under travels. One solution is to use a feedback loop, where the input is current velocity and orientation of the boat. These can be obtained from the GPS and the IMU aboard the boat (Zhenyu Yu 2008). Then the wanted and actual direction of travel can be cross referenced, and the rudder adjusted accordingly (Roland Stelzer 2007).

During travels, wind and forces from the water will most likely change. Especially for great distances. This means the leeway effect will change accordingly. Not to speak of the propulsion of the boat, which is dependent on the wind and other parallel forces. This means the course of the boat will have to be adjusted from the direct route during travels (Saoud et al. 2015). For now, the control of the boat is only taking in the current state of the surrounding environment. This only includes adjusting the sail to different points of sail, as well as rudder control.

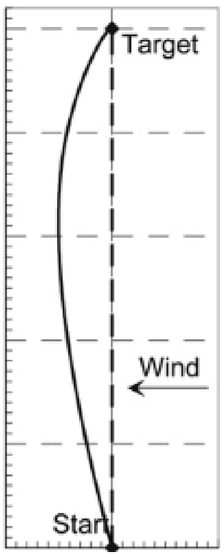


Figure 6: The planned path between start and target, and the followed path with leeway.

Source: (Roland Stelzer 2007)

2.3.2 Control surfaces

To steer the boat on the correct course, and manage best use of the sail, a split control scheme is used. This means the control of the rudder and the control of the sail is made independent from each other (Patrick F. Rynne 2010). When a path is planned, a point of sail is chosen based on best fit with the current wind. This decides the position of the sail to get the proper angle of attack. With the separation of the controls, this is all the sail has to worry about. At the same time, the rudder is independently controlled to steer the boat on the correct course.

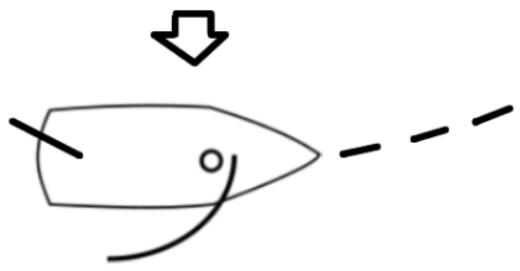


Figure 7: Path following with rudder and angle of attack control with sail.

Source: Own image based on https://en.wikipedia.org/wiki/Point_of_sail

Thus, the existing solution on the boat is to use the sail for maximizing propulsion from the wind, while the rudder is used to control the heading. In addition, the rudder will adjust for leeway under travels, with the help of GPS data. As these controls are crucial for the boats progress on voyages, the sensors need to be reporting the correct measurements. They need to report correct data, and with minimum delay. Without this, wrong angles and states can be set, and in the worst case, the boat might come to a complete halt. This is especially true when sailing against the wind. Having enough speed to pass through the no-go zone, and maneuvering correctly during the pass, requires precision and speed from the sensors (Saoud et al. 2015).

2.4 Prototyping

Prototyping is a method used for developing through testing. During this project, testing will be crucial to ensure quality and ending up with well working solutions. On its mission, the boat will be almost inaccessible, and fixes are near impossible on a running basis. Incorporating prototyping in the process will uncover problems early, and give a trustable system on deployment.

Prototypes are made to express a design for incremental or radical improvements (Elverum and Welo 2015). Although today's technology has given many simulation programs, nothing will be as accurate as physical models. Building a full size model of the boat of this project yields a perfect test setup. This gives a physical platform for assessing actual solutions in a real environment and with accurate properties. It is also the only way of making sure the full boat will work properly before its mission. On the other hand, testing of complex systems can be demanding and take long time (Elverum and Welo 2015). Because of this, a proper understanding of prototyping is important to be effective. This lets us decide what to test and how to extract what we can from the tests.

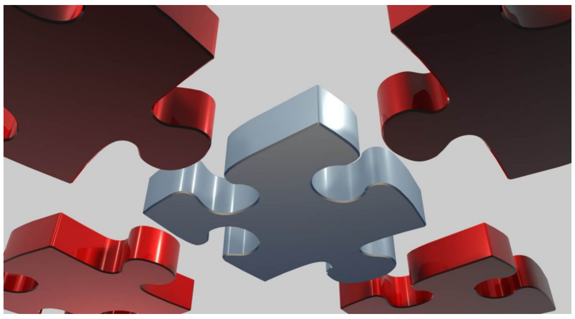


Figure 8: Modular testing can be achieved by locking the influence of other parts, and focusing on only one part.

Source: https://www.webomates.com/blog/manual-testing/modularity-in-test-case-writing-is-critical/

Prototypes provide the means for examining design problems and evaluating solutions (Houde and Hill 1997). It is important to choose the scope of a prototype carefully. When developing new products, many believe that a modular testing design is the best. This locks the variation in interference from other parts and lets you focus on how one part is working. Under known conditions, the variation can be removed by staying consistent with all parts not currently up for testing (Houde and Hill 1997). This leads to proper understanding of how individual parts work, and the results of changes to it.

A happy bi-effect of prototyping can be that new ideas and concepts are often discovered accidentally (Elverum and Welo 2015). When the system is tested, new ways of manipulating it or other aspects not earlier thought of might come to life by itself. Some project specific examples of this could be transportation problems, rudder angle control and sailing properties. The discoveries made in this way are just as important, and big part of why testing with prototypes is so important. There is no other way that can give such a complete run down of how the system actually performs (Elverum and Welo 2015).

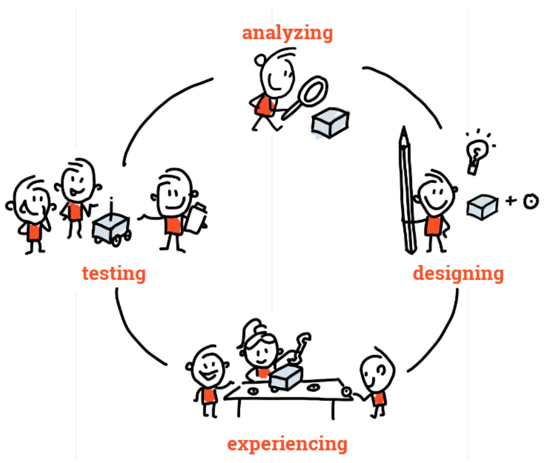


Figure 9: Exploring and discovering through prototyping.

Source: https://www.uid.com/en/news/prototyping

For this project, the boat being build is the prototype. Constant testing should be a natural part of its development. However, testing these complex systems can be hard, especially when trying to analyze what influence the different parts have on results. A big help is to single out parts as much as possible. Modules of the boat can be faded out, kept still, to lock its influence to a known constant. An example is to use rudder control to keep the boat on a straight course. Combining this with known wind conditions from the wind sensor, the sail can be tested. Setting different angles of attack and points of sail, the forces can be measured. The data will yield answers as to how effective the sail is, and if the coded angles are the best ones.

Many of the prototyping aspects described thus far was originally meant for software, and has been altered for hardware developing. This means most of the same principles work well for software development as well. This is important for the project, as all the control algorithms are made with software. Directional prototyping is a great tool to figure out whether discovered problems have roots in hardware or software solutions (Elverum and Welo 2015). This is a way of constructing tests to put wanted systems to the test and figure out the roots of the problems. Good planning with thoughts of expected results in different cases are crucial, and might be hard. At the end, this will likely be a way of discovering problems and finding solutions to them before the boat sets out.

3 Existing solutions

This is not the first year a thesis has revolved around developing the boat for this project. A lot of solutions are already found, and prototypes built. The focus of this thesis will mainly be the further development, but it is important to understand what is being built upon. This section is in large parts built on the project thesis, used to prepare for this work. It will briefly explain the relevant, existing solutions.

3.1 Electronics on board

An advanced system of on board electronics has been built. It manages the control, logging and actuation of the boat. In addition, it communicates with a land-based server for remote control and monitoring. Self-sustaining systems for monitoring health and power of the system are also implemented. This section will briefly explain the main parts of the hardware system at work here.

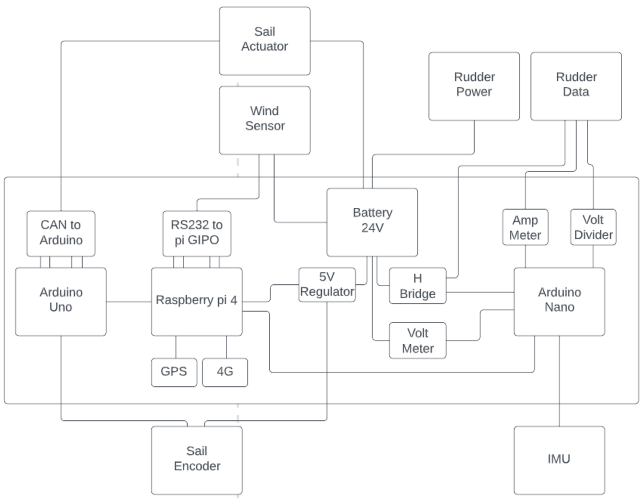


Figure 10: Schematics of electronics on board.

3.1.1 Raspberry Pi

For the main brain of the boat, a Raspberry Pi is used. It communicates with the other boards and runs the main code. The Raspberry Pi is the first board to start when powering up the boat, and will initialize and power the other microcomputers. Through communication with the land-based server via a 4G modem, it receives the last target coordinates. It reads sensor data off the other microcomputers, including wind and current orientation. With this information, the further travel is calculated on the Raspberry Pi. The heading is decided, and signals to position the sail and the rudder are communicated out. During travels, sensor data about heading and position is regularly checked with the IMU and the GPS.

3.1.2 Arduinos

Two Arduinos are set up to handle direct control and communication with sensors and actuators. These are controlled by the Raspberry Pi, and sends the data for further calculations to it (Pleym and Ølstad 2022). Using several microcontrollers in the structure also benefits from splitting up the code. It results in easier maintenance and debugging as it reduces the size and complexity of one single program. Reading of sensors and control of actuators can then be divided into separate classes and methods in the code, all tied together with the Raspberry Pi.

Of the two Arduinos, one is of the type Nano and the other of the type Uno. The Nano handles reading the IMU and controlling the rudder actuator. The Uno reads the sail encoder and controls the sail actuator. For communication between the sail actuator and the Arduino Uno, a CAN to Arduino interface is necessary. This is because CAN is the default protocol for the encoder (Pleym and Ølstad 2022). The Arduino Nano is in addition part of the self-monitoring system. Via a voltmeter, it is connected to the batteries, monitoring the power left.

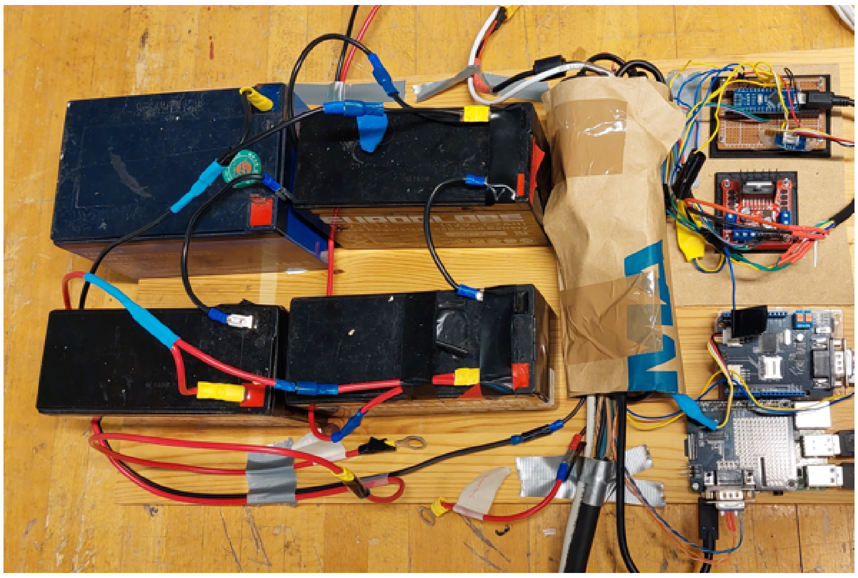


Figure 11: Power supply on the left and control electronics on the right. From the top: Arduino Nano, H-bridge, Arduino Uno, Raspberry Pi.

3.1.3 Power supply

To power the boat on its travels, four lead-acid batteries are used. Each have a nominal voltage of 12 volts. They are parallel connected in two pairs and each pair connected in series to achieve a voltage of 24 volts. As some parts of the electrical system requires 5 volts, a regulator is connected between the battery output and the necessary parts. The batteries are the only source of power during travels, and these running out will be the end of any mission. The boat will in this case lose all control and communications with land. This gives high risk of losing the boat. This is why the power of the batteries are constantly watched by the Arduino Nano. Measures to resupply the batteries during voyages are in the workings.

3.1.4 Actuators

Actuators are the motors used to control the angle of sail and the rudder on the boat. They are controllable motors where specific angles or positions can be set. For the sail, the actuator rotates directly to a worm gear, connecting the mast and the sail. Rotational motion is shifted 90 degrees with the worm gear, while it also shifts up the torque heavily. With the rudder, a linear movement from the actuator is translated into rotational movement about the rudder rod. This is done by a mechanism which also holds the rudder and actuator together. As these are the only influence the boat can have on its travel and orientation, accurate measurements of their angles are necessary.

The rudder actuator angle is changed via an H-bridge controlled by signals from the Arduino Nano. The H-bridge is connected to the batteries and will supply power to the actuator based on signals from the Nano. This includes the possibility of the H-bridge reversing the voltage, changing the movements of the rudder to the other way. The actuator is built to return data about its extended position. By measurements, the relation between how extended the actuator is and the resulting angle of the rudder is programmed (Pleym and Ølstad 2022). Together, this gives full control of the rudder's angle in the water.

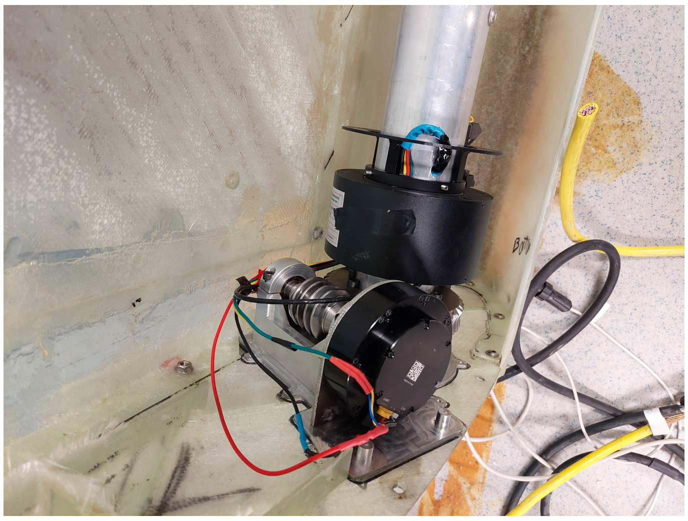


Figure 12: Inside the sail: the actuator and the position sensor.

The angle of the sail is set with a rotational actuator, controlled by the Arduino Uno. The actuator is placed within the sail, protecting it and giving it access to the sail and the mast. The worm gear used lowers the required strength of the actuator due to the gear ratio. This also makes it easier on the actuator to hold its position in strong winds. The actuator is made to return its position, but unfortunately only as a relative position to where it started (Pleym and Ølstad 2022). As the position needs to be known, even after resets, a rotary encoder was developed. By reading movements of a disk with an IR-sensor, it returns the position of the sail to the Arduino Uno.

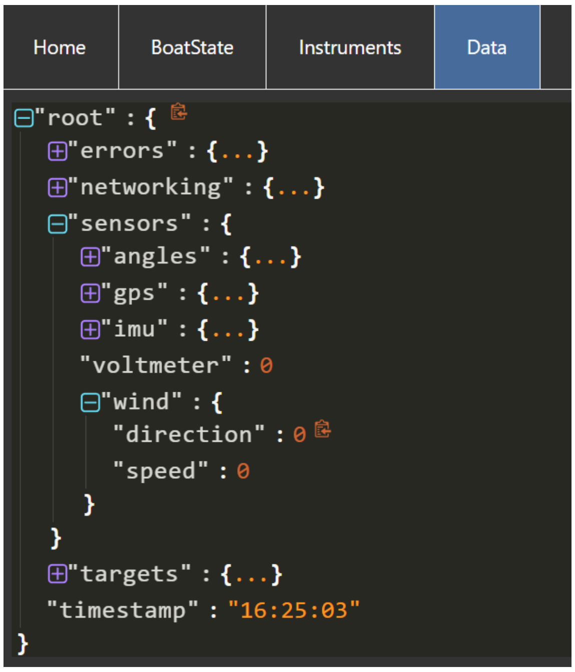
3.1.5 Positioning

Aboard the boat, a GPS-antenna and an IMU is used for checking the boat's position and orientation. These are read by the Raspberry Pi. The GPS gives the position, used for navigating and compensating for leeway. This ensures travels hit their target. The position from the GPS can also be used together with environment data, to mark the belonging location. At the same time, the IMU is used to check the roll, pitch and yaw of the boat. This is used for current heading and sent to the Raspberry Pi for navigation calculations. This is also a control of how much the boat is tilting due to forces from the wind and water. Tilting over can be prevented with this by steering the boat to a more stable orientation.

3.2 On-shore server

An on-shore server has previously been developed to handle communication and logging. It is also hosting the dashboard app. This is useful for remote control of the boat. The logging can be used for test analysis, one of the cases used later in this thesis. In addition, the logging will save last known position, orientation and conditions This can be crucial in the event of something happening to the boat, leaving it inoperable.

The communications and logging will give frequent updates to a reachable server. This is a good way of getting data of interest, tagged with a location and time. This way, environmental and weather data can be collected with low cost. Future development is likely to add more sensors on requests. The data from these can just as easily be reported via the land based server.



```
⊟"root" : {
  ⊞"errors" : {...}
  ⊞"networking" : {...}
  ⊟"sensors" : {
    ⊞"angles" : {...}
    ⊞"gps" : {...}
    ⊞"imu" : {...}
    "voltmeter" : 0
    ⊟"wind" : {
      "direction" : 0
      "speed" : 0
    }
  }
  ⊞"targets" : {...}
  "timestamp" : "16:25:03"
}
```

Figure 13: Some of the data retrievable from the server.

Source: https://ntnu-autoboat.web.app

The server was set up to shut down when not in use to save cost and energy. This happens without breaking the system and can be rebooted when needed (Pleym and Ølstad 2022). Logging also comes in here, as the last states are being saved, and ready for use on reboot. However, it should be noted that a reboot of the system can take a minute. On remote control from the dashboard app, one thus has to expect a small delay before communications are up and running.

3.3 Dashboard app

The dashboard app, mentioned earlier in the report, is the human-machine interface. It communicates with the boat and lets one remotely control the boat and read data from it. When connected to the boat, the dashboard app displays live updated information about the boat. This includes position, heading and speed, but also critical information like errors and warnings. Via the dashboard app, different states can be set for the boat. In addition, target locations can be added, deleted or edited here. It is an easy to use human-machine interface, that gives control and monitoring of the boat.

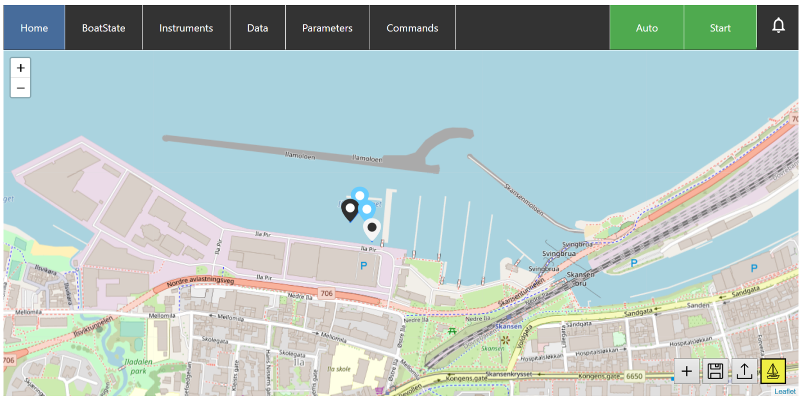


Figure 14: The dashboard app used for control of the boat.

Source: https://ntnu-autoboat.web.app

The dashboard app communicates via the on-shore server, and talks with the Raspberry Pi. The Raspberry Pi is the main brain of the boat, and thus have access to all the data from sensors and states of actuators. The dashboard app is divided with tabs, accessible with the buttons in the top of the figure above. Different actions and monitoring are available for the tabs. The tabs include boat orientation and movement, data from sensors, states of server communication and commands. This gives full control of the boat, even when it is far away. In the picture below, one tab is shown, displaying the state, orientations and movements of the boat.

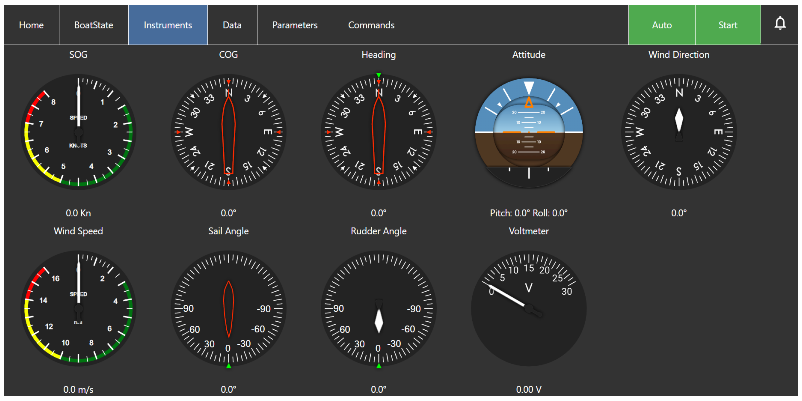


Figure 15: A view of the instruments on board, showing orientation and health metrics.

Source: https://ntnu-autoboat.web.app

4 Preliminary tests

After previous development of the sailboat, several solutions were discovered not to work properly during sail tests. Tests have been constructed to uncover these, and to catch other problems like them. Following the principles of prototyping described in 2.4, the tests will be designed based on the results from the sea trial conducted in (Pleym and Ølstad 2022). The objective of the tests is to find the reason for the faults, and fix them, described in 5.1 This section will describe the different test setups, and show and discuss the results.

4.1 Sail angle with simulated wind

Previous testing discovered faulty or unresolved angles of the sail during trials. Built in maneuvers had some issues preventing them from working properly. In short, this test is thus to simulate a wide range of wind directions and observe the resulting sail angles. The objective is to verify correct sail angles during all three programmed sailing modes: normal, tacking and beating, as described in 2.1.3.

4.1.1 Test setup

With the limiting space inside, the sail had to be disassembled to fit in a position allowing it to turn all the way. Unfortunately, the aluminum in the joint of the mast right above the motor assembly was stuck. Probably due to small bends resulting from the sea trial. The joining part was cut and a new one made to replace it. The new part is made slightly smaller to avoid similar situations in the future.

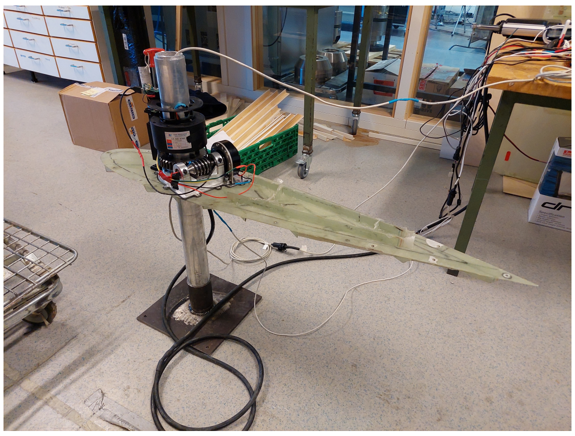


Figure 16: The half mast from disassembled sail.

With the shortened mast, wiring for the wind sensor was easier to access. This is important as manipulation of the wind sensor is the base of the test. For this purpose, a stand and holder for a fan was made. It fits around the fastening sleeve of the wind sensor, used on the mast. Around it is a ring, able to slide around the sleeve. The ring holds a stand for a fan. This assembly allows a fan to constantly blow on the wind sensor, while the angle it blows from can be changed by the sliding feature. This setup will be used for several tests with simulated wind.

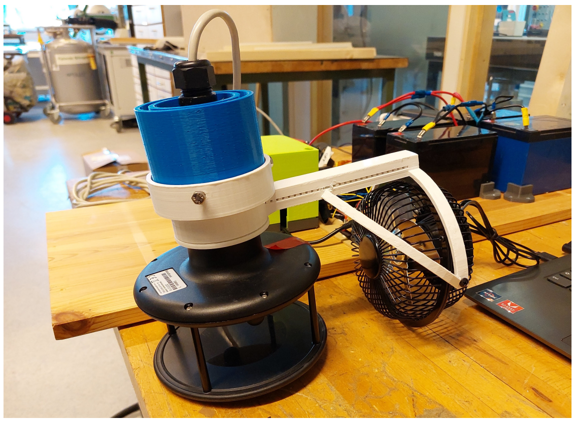


Figure 17: Fan holder for wind sensor.

Using the electronics board with the batteries, all was ready for the test. The Raspberry Pi was already set up in the system to receive data from the wind sensor and transmit it via the 4G modem. On the other end, the server would store the data in log files for later analysis. The test was done by setting a target for the boat to sail to. Then the wind direction was adjusted to cover all directions with small increments. The sail position was observed and the logs stored for analysis.

4.1.2 Results

During the test, the sail was observed to react quickly to changes in wind direction. The wind sensor incrementally updates the wind direction, and for large, sudden changes in direction, it will take some time for it to report the new direction. The updates are, however, fast enough for the sail never to have to wait to proceed to the new position. It is, as described in (Pleym and Ølstad 2022), limited to 6.43 degrees per second with no load.

During the test, a lot of data points were stored in the log. To help analyze them, two python scripts were made to visualize some important aspects of the results. The first, shown in figure 18, is a plot of the two values, wind direction in blue and sail angle in orange over, time. The graph shows the earlier observed quick reaction and reposition of the sail to changes in wind. The sail angle is constantly a small offset from the wind direction. This is to keep the angle of attack to about 10 degrees, as found in (Pleym and Ølstad 2022).

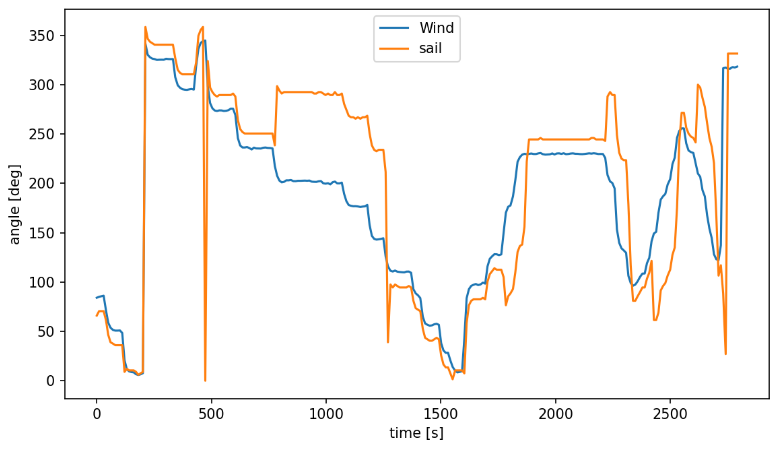
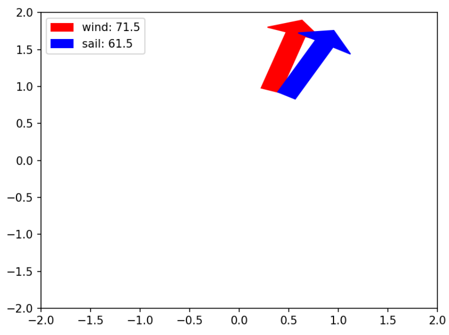
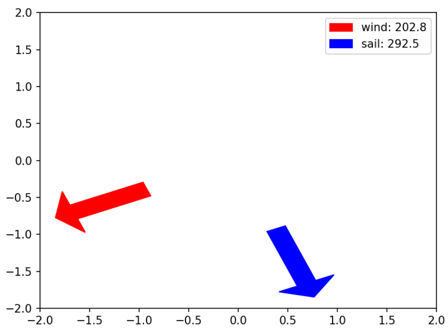


Figure 18: The wind direction and sail direction, shown as angles over time. The drop spikes are a result of the angle jumping around from 359 degrees to 0.

Several ranges in the graph deviate from this rule of an angle of attack of 10 degrees. Rather, the difference in angle between the wind and the sail is here about 90 degrees. One example is around the 1,000 seconds mark. These areas show the direction of wanted travel and the wind direction coming together. As mentioned in 2.1.2, this means the point of sail is entering downwind. Because of this, the sailing enters drag mode, and the sail is set to 90 degrees off of the wind direction to maximize drag.



(a) 10 degrees angle of attack.



(b) 90 degrees angle of attack.

Figure 19: Arrows showing the wind direction and sail angle for each logged point.

To further help verify the angles, the figure above shows arrows for each data point displaying the wind and sail direction. This helps visualize how they will be positioned relative to each other. These figures show the same results as the graph, but was a good tool to understand the different cases.

This test has proven that the sail is able to position itself to wanted angle of attack during normal sailing operations. It is also happening relatively fast. Beating and tacking was not achieved during the test. Probably, this is because the GPS can not get a lock of position inside the building. This results in the boat not being able to calculate the midway points, thus not putting one inside the no-go zone or in a direction downwind. As tacking and beating includes switching directions after a certain distance of travel, a moving test outside will be constructed to test these.

4.2 Test tacking

The previous test was useful for checking the movement of the sail during general sailing operations. Although it did not yield the hoped for results with regards to tacking and beating, it was not wasted. The test gave insight about the system and sensors used, and also showed the way to design a test where the maneuvers would happen. As discussed in section 2.4, this is a common way to happily, accidentally discover new things. This time, the test will be done with a larger subsystem than the previous test. It will also happen outside to give the GPS a chance for a lock.

4.2.1 Test setup

For this test, almost the entire boat's system was built into a trolley. It has handles, used to control it. Because of this, and as the sail is large, heavy and hard to move around, only the base of it was included. This contains the control system and actuator, and can also fit the wind sensor on top. The other part missing is the rudder, which is not needed as the test is not in water. Rather, it will be controlled by the handles of the trolley, and will manually be moved in the direction decided by the navigation system aboard. As this will be a valuable way to test later design and control system changes, the trolley will be further described and shown in section 5.2.

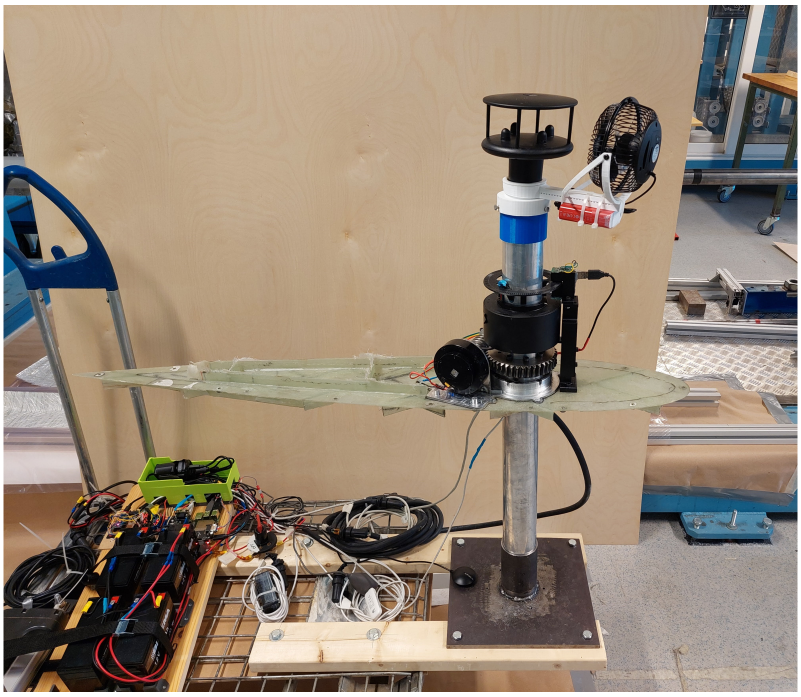


Figure 20: Test trolley with the sail base.

The fan attachment was again used with the wind sensor to simulate certain wind conditions. Tacking and beating require wind from specific angles. This means the test should be done on a day with relatively little wind, to keep the uncontrollable variations to a minimum. A parking spot was originally the planned place to do the test, but the asphalt and pebbles made a noisy and shaky system. For later testing, a football field might be a good option. This gives enough space to follow the path chosen by the navigation system. Due to unstable weather and snow covered fields, the test was redesigned to be done in the lab with adjusted fans. A fans at the correct distance were used as a stable wind source, and a mounted fan represented the head wind. The trolley was steered according to the directions shown by the navigation system in the dashboard app. During the test, the sail angle, along with other positional and sensor data was logged for later analysis. The way around the GPS lock and getting positional data was solved by simulating movements, further described in section 5.3.

4.2.2 Results

During the test, the web application was used to monitor the states of the system. As the boat enters tacking mode, red markers are placed on the map in the application. These show midway points, calculated at an angle out from the target to sail outside the no-go zone, as described in section 2.1.2. The graph also show a varying wind angle as the boat is turning back and forth between the sides of the no-go zone. The horizontal flat parts show close-hauled sailing outside the zone, while the steep parts shows the resulting wind as the boat is turning though the eye of the wind.

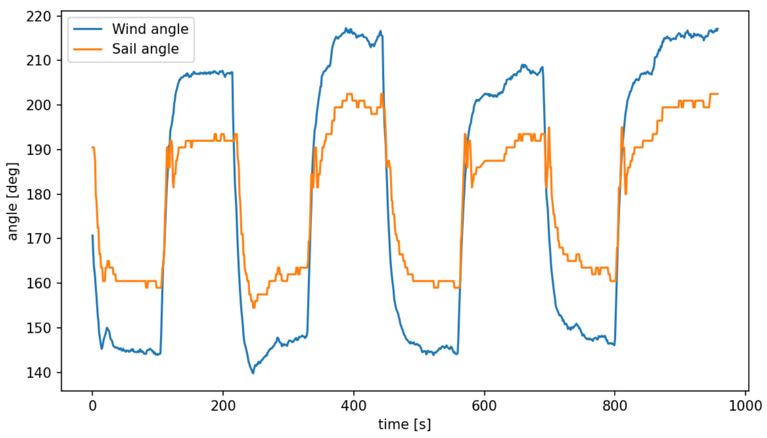


Figure 21: The wind and sail direction shown in angles from the boat.

During the test it was discovered a small uncertainty in the algorithm of the boat. Even when the wind from the fan was set up directly against the boat in the direction of the target, there was a noticeable skewed distribution to one of the sides. This is most likely due to the same wrongful calculation of true wind as further described in the next test, 4.3. The tacking test was still able to tack, but the boat had to be turned more to one side than to the other to be contempt with the wind angles to sail in. This was shown in the web application by recalculation and moving of the midway points. This mostly happened on one side, most likely due to the biased no-go zone. It happens because the boat replaces the midway points if it discovers one is inside the no-go zone. This can be seen in the code in the appendix.

The test done by sea trial at the end of last year's team suggested another problem for tacking. This is keeping the angle of attack at the set value, also during turning through the eye of the wind. This problem is confirmed in this test, as shown by the graph. A difference in sail angle and wind angle of 10 degrees is mostly kept where possible. Keep in mind that small differences in the graphs in the steep areas can show a big difference in angle. The constant angle of attack tries to use the sail even as it enter the no-go zone. This results in a lot more, unnecessary drag during a crucial time of the maneuver. During the sea trial, the boat was reported to even move backwards due to the misfitting use of the sail.

This test done confirms some of the problems discovered earlier, and also deepens the understanding as to why it is happening. With this knowledge, section 5.1.1 and 5.1.3 takes these further and suggests new solutions.

4.3 True versus apparent wind

Previous year's testing discovered faults in the placements of midway points when tacking. A well calculated placement of these are necessary for generating propulsion in the wanted direction. The cause of misplacing these were found to be in the calculations for true wind. Calculated as described in section 2.1.4, the resulting true wind found by the algorithm was discovered to be wrong, and the cause of misplacing midway points. A test was designed to elicit the causes to fix them, as described in section 2.4.

4.3.1 Test setup

For calculating the true wind from the apparent, measured wind, the velocity of the boat needs to be known. As discovered during the test of the sail, 4.1, the GPS could not get a lock inside buildings. The velocity measurements are based on data from the GPS, thus the test was necessarily performed outside. In addition, some roughly stable velocity in an appropriate range was needed. As the option comes easy, the natural pick was to rig the necessary equipment in a portable manner and use a bike. The test was done on a day with relatively little wind and on mostly straight lines to keep external conditions roughly stable.

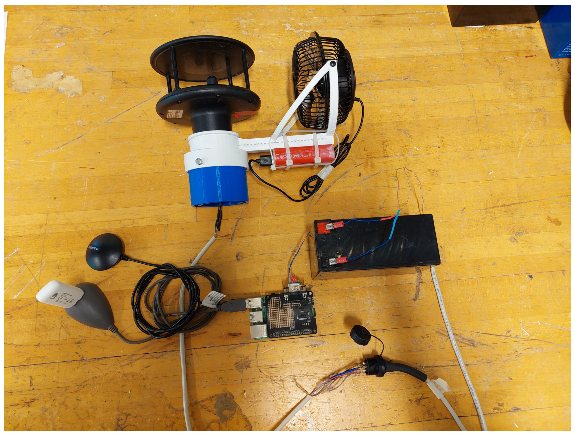


Figure 22: The setup used to bring the needed equipment on a bike, for testing wind calculations.

A subsystem of the electronics for the boat was needed for this test. A backpack was big enough, and easy to bring on a bike. The pack contained the Raspberry Pi, GPS, 4G modem and a connection to the wind sensor with a fan. The Raspberry Pi is necessary for reading and controlling the sensors and communication to store data on the server. The GPS was used to keep track of positioning and calculate velocity from it. The 4G modem connects the Raspberry Pi with the server and lets it log the values during the test. The fan was used to simulate a true wind during the test. Lastly, the wind sensor measured the apparent wind, resulting from natural wind, the fan, and the wind occurring due to the movement.

A few adjustments had to be made for the test to work. First, the wind sensor is driven by 12-24 volts. The cable usually in the boat was loosened from its connections to connect the Raspberry Pi to the wind sensor, and the power lines to one of the four 12 volt batteries. To skip the cable inside the mast, a new one was made with small clamps for each connector pin on the outgoing cable and on the wind sensor. This also extended the cable, allowing the wind sensor to be held out of the pack while on the bike. Secondly, a 5 volt 3 amps power pack was brought to power the Raspberry Pi. Third, a similar pack was used to power the fan on the wind sensor. This used the same setup as shown in figure 17 to simulate wind.

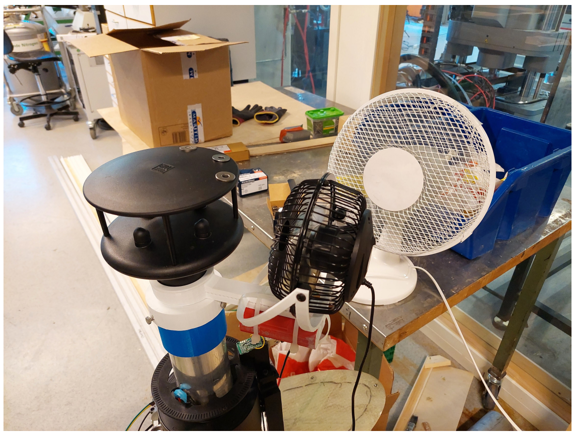


Figure 23: The fans setup to simulate true wind and head wind as the velocity of the boat was changed.

The setup has been made and verified to be working, and can thus be used for further testing if needed. However, finding a suitable day with little wind and roads not covered in snow proved difficult. The time was of the essence as the results from the preliminary tests were to be used for further development. As such, it was decided to redesign the test to be able to do it in the lab. The same setup was used, but the bike was replaced with fans and simulating GPS movements as described in 5.3. The speeds of the winds were measured to place the fans at correct distances to align with the simulated velocity of movement. This new setup proved valuable to get the test done, and gave results later used to understand and fix the issues.

4.3.2 Results

From the graph, the first thing to notice is the difference in true wind and measured, apparent wind. A big point is that with no velocity on the boat, the true wind and the apparent wind are reported to be the same. This is a correct calculation. As the boat stands still, the wind sensor aboard becomes a stationary point of measuring. This means the measured wind is by definition the true wind, as described in 2.1.4. The sharp upwards curve of the graph at these points, where the velocity is 0, is due to the transition. In the test, the boat velocity was suddenly set to zero, and the fan shut off at the same time. However, the momentum keeps the fan spinning, giving the boat a head wind gradually descending as the fan slows down. This gradual slow down in combination with the sudden 0 speed, temporarily makes the calculations wrong. This will, however, not happen during actual sailing.

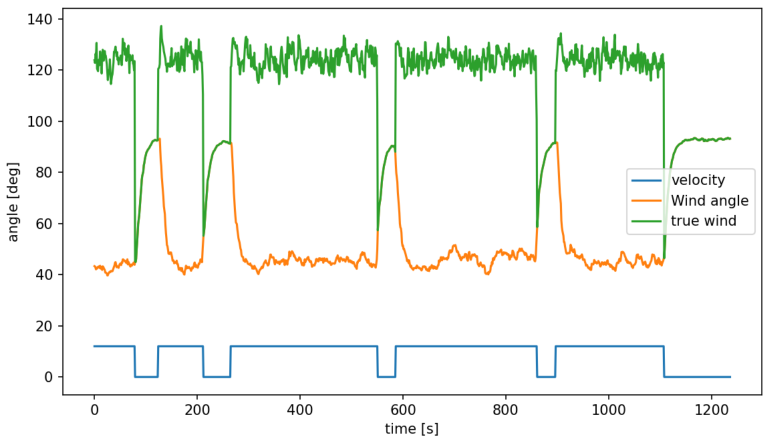


Figure 24: Measured wind and a varying boat velocity on 0 or 1.2 m/s shown with a resulting, calculated true wind.

The wind angle-line shows the apparent wind, measured by the wind sensor. While the boat is sailing, shown by the velocity-line, the measured wind lies around 45 degrees. This is the resulting angle from the two wind applied. They are parted by 90 degrees, placed at around 0 and 90 degrees. As they also have similar speeds, the resulting wind angle will be somewhere around the middle, at 45 degrees, which is what we see. As soon as the boat velocity is set to 0, and the fan shut off, we can see the measured wind reacting to it. As soon as the fan has stopped, it becomes clear that the apparent wind is converging in on 90 degrees. This is also a good result, as then only the fan oriented on the right, at 90 degrees is blowing.

This last result does also show a fault in the calculations. As mentioned, the true wind fan is placed at 90 degrees. This is also shown in graph by two things. As mentioned, these are the angle the apparent speed is converging on with no boat velocity, and also shown by the resulting wind being at 45 degrees. However, this 90 degree angle is not what is shown in the graph for the true wind. Rather, it fluctuates at around 125 degrees, 35 too much. This means the calculations and algorithms are on track, and acting correctly to different states, but the true wind angle calculation is returning the wrong values. The cause and fix of the problem will be continued in section 5.1.1.

5 Method

A lot of progress has been made by previous development. Yet, not everything is working as intended. To get the boat ready for its mission, the focus will be to straighten out all mistakes. Previous testing has pointed out some problems, but not all causes are known. Last chapter focused on more testing to find the cause of known problems, as well as search for any other ones in crucial operations. This chapter will build on the acquired knowledge from these sources, and focus on using it to get the boat ready. In addition, it describes the steps done for improving testing capabilities and availability for further development.

5.1 Work on existing issues

From previous development and testing, some issues have been discovered. These have been further explored in the previous section, section 4, to find the cause and implications of the issues. After this exploration, a better understanding has been built up as to what is needed of the subsystems to work properly. This section will go over the solutions found and changes made to these previously discovered problems.

5.1.1 True wind versus apparent wind

As described, the Raspberry Pi acts as the brain of the boat. Thus, this is where the calculations for navigating is happening. As a part of calculating the route to the next target from the current location, the true wind is needed. Though the sail is angled based on the felt wind, the true wind dictates the heading. For example, a target to be reached with the wind coming from the same direction will result in a heading within the no-go zone. Using the true wind, a midway point with a heading just outside the no-go zone will then be calculated.

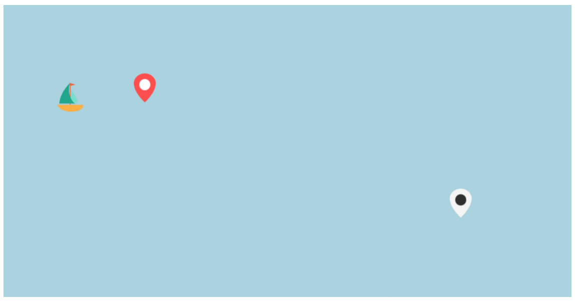


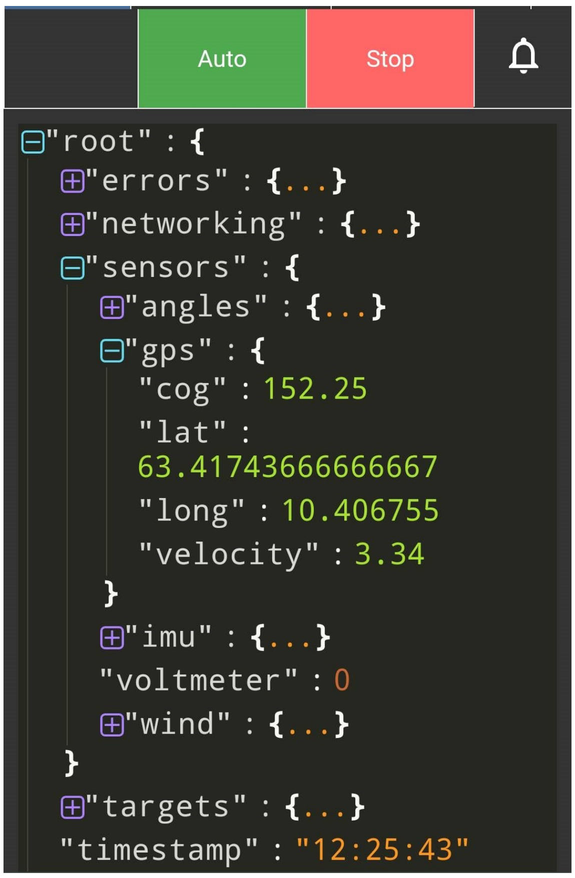
Figure 25: Midway point in read on the way to a target marked in white. The wind is from an east-southeast direction, making the target be within the no-go zone.

The true wind cannot be directly measured from the boat, as it is moving. As described in section 2.1.4, it can be derived using the measured wind and data from the GPS about the velocity of the boat. As the angle of the wind is defined by the direction of the boat, the resulting felt wind due to the velocity of the boat will always be close to the same angle. Making turns can shift this angle a bit as the direction of travel is not directly with the direction of the boat. This will however not be much, and most readings will happen with the boat traveling with its direction.

During earlier testing, and the testing described in 4.3, the calculation of true wind in the above manner has been discovered to be wrong. This results in bad decision making and placements of midway points. Worst case, this can give the boat a heading in which it cannot sail due to entering the no-go zone. It is, however, not the formula in it self that is wrong. The equation used is from (*Calculating apparent velocity and angle* 2023), and is

$$\alpha = \arccos\left(\frac{A\cos\beta - V}{\sqrt{A^2 + V^2 - 2AV\cos\beta}}\right). \tag{1}$$

Where A is apparent wind velocity, V is boat wind, β is the angle of the apparent wind, and α is the true wind angle. The boat wind, V, is thereby the wind coming from the movement over ground of the boat. It can also be denoted that the entire denominator within in the inverse cosine is the true wind speed.



```
"root" : {
  "errors" : {...}
  "networking" : {...}
  "sensors" : {
    "angles" : {...}
    "gps" : {
      "cog" : 152.25
      "lat" :
      63.41743666666667
      "long" : 10.406755
      "velocity" : 3.34
    }
    "imu" : {...}
    "voltmeter" : 0
    "wind" : {...}
  }
  "targets" : {...}
  "timestamp" : "12:25:43"
```

Figure 26: Velocity from GPS read off from the web application on a mobile phone.

As mentioned, the equation is not faulty, but the results still were. Through testing in section 2.1.4, it was discovered that the reason for the wrong calculations was an unexpected unit of the speed output from the GPS. It deviates from the standard unit of m/s, and rather reports the speed in knots. The conversion from knots to m/s happens by multiplying the amount of knots with a factor of approximately 0.514. To fix the problem, this multiplication is added in the code in the true wind angle-method input. The navigation program, developed from earlier and changed for improvements is in the appendix, B.

5.1.2 State of electronics

From previous development, the electronics were found to be sub-optimally assembled. During simple, stationary testing, sounds from arcing could be heard. This means isolation between electrically leading parts of the system with voltage applied is too low, or something is broken. In this case it was both that the metal of wires and solder points were not covered and that they in addition were loose. Free to move, these parts touched and made small sparks. Several other points were found to have bad isolation and wiggle room to touch other conductive parts. Lastly, solder points were found to be brittle. This was part of the problem with conducting parts touching, but also makes bad physical attachments prone to break easily.

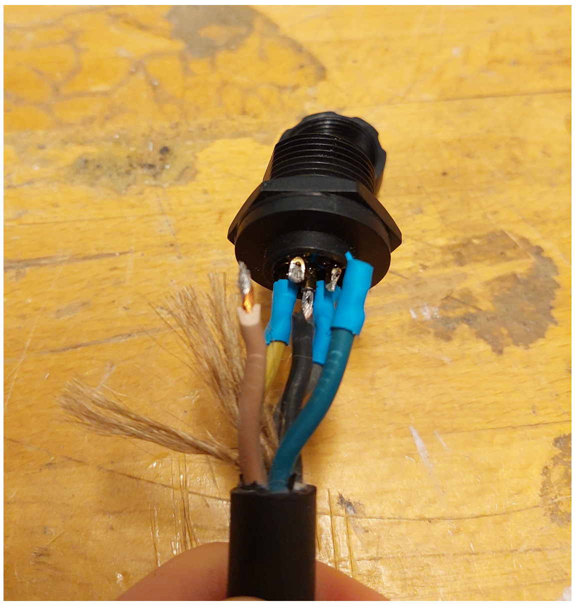


Figure 27: Example of brittle solders breaking.

The reason for brittle solder points is often too low temperature of the bonded materials. Melting the solder, but not letting the other metals heat up enough is often the cause. Covers were cut away to reach solder points along wires and on connectors. The found problem areas were resoldered, this time with enough time and heat to get a proper attachment. As the boat has many solder points in many connector solder cups, it is worth noting that not all are resoldered. Most have held up thus far, but it is good to look out for the rest as loose attachments can short anything.

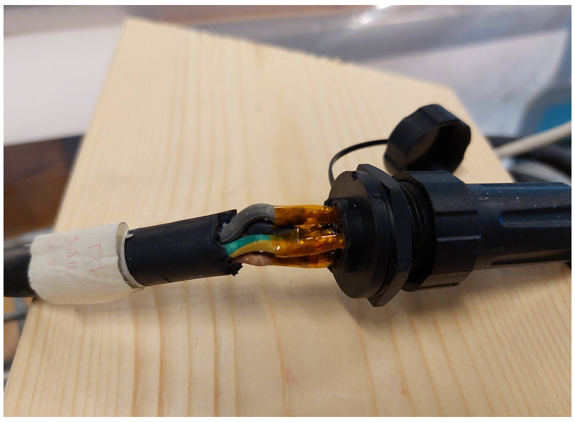


Figure 28: Example of good solder with enough heat and covered with an isolating, heat resistant material.

In any case, making sure a non-conductive material separates parts with applied voltage is a good precaution. This makes it harder to short out the power lines which can possibly destroy electronic equipment. In addition, it rules out any data transmitting wires from touching others. This is a common problem requiring a lot of debugging to understand why the data is wrong or systems are not working. All points found lacking isolation has been covered with heat shrink tubing or masking tape for electronics. This will add a preventative layer from future development and testing accidentally burning something or corrupting data readings. In addition, the wires and cables have been collected in common routes to increase strength, lower complexity of overview and improve effectiveness of visual inspections.

5.1.3 Tacking

After last year's development, tacking was discovered to behave suboptimally. During the sea test in (Pleym and Ølstad 2022), the boat lost all its speed when trying to turn through the wind. It was also discovered a fault in the calculation of true wind. This is crucial to the performance of tacking. In addition to work on this calculation, the program has been altered to try to minimize the drag on the sail while turning through the wind.

The fault in the calculation of true wind has been previously fixed as described in section 5.1.1. This should increase the tacking performance by making sure the angles are correct. A wrong calculation of true wind will result in one of two things. When the calculation is wrong, it sets a wrong angle on the true wind, extending it in one direction. In the first outcome, when the boat tries to sail close-hauled on the same side as the wrong true wind, the angle of sailing is exaggerated. It is longer away from the wanted direction than necessary, making a longer trip. The boat will, however, have good wind conditions, as it is more closing in on beam reach, as mentioned in section 2.1.3.

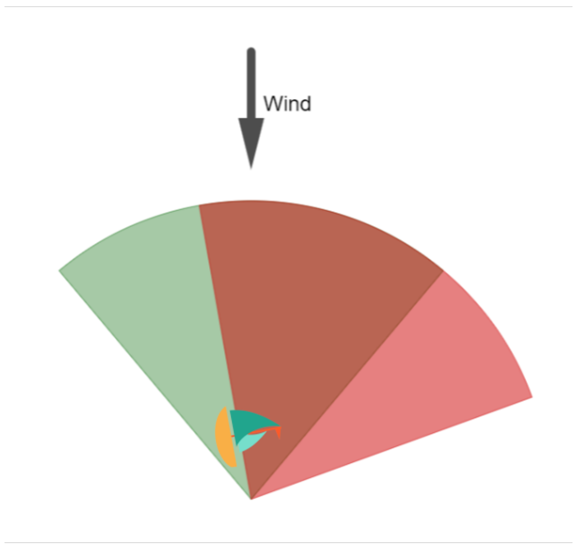


Figure 29: True wind (green) and false true wind (red). When it thinks it is sailing close-hauled, the boat can be trying to sail within the actual no-go zone. This is by the edge of the green zone within the red zone.

When the boat then tries to turn through the eye of the wind, it will find more problems in the second outcome. This is when the boat decides to turn to the opposite side of the false no-go zone, where it is inside the actual no-go zone. It will then not have enough of an angle from the true wind to enter close-hauled sailing. Rather, it will stay inside the no-go zone, potentially losing all speed. If lucky, it can sail close to the edge of the no-go zone, but with a heavily lowered speed. This will most likely not be enough to pass through to the other side when again turning through the eye of the wind. With a correct calculation of true wind, these problems can be avoided. Thus this will contribute tremendously in helping the boat being able to do the tacking maneuver.

When tacking, enough speed is crucial to able to sail through the eye of the wind. To further help the boat making this maneuver, the required speed can be lowered. This can be achieved by making sure the sail turns as the boat turns through the wind. Using this to place the sail in a neutral position, directly up against the wind can help by drastically reducing drag. This will lessen the amount of which the boat is slowed during the time it has no way of generating positive propulsion. As the boat is sailing close-hauled with an angle of attack of 10 degrees, little adjustments are needed to set the sail in a neutral position with the round edge towards the wind.

To achieve this, the sailing program has been updated with some new logic. The navigation-file is where the sail and heading calculations are done, and thus this file contains all the necessary changes. The file is shown in the appendix, B. The previous logic sets up midway points on an angle off from the target to get outside of the no-go zone. The sail should be in a neutral position only when we are between these midway points. For this, a variable is added to the code to keep track of whether we are coming from one of these midway points.

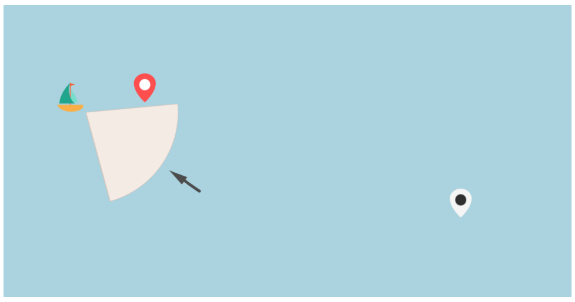


Figure 30: Midway point marked with no-go zone and wind direction.

As long as the variable telling if we are coming from a midway point is true, the angle of attack is set to 0. This sets the sail in a neutral position, following the wind while sailing through it. Three cases alter the variable. First, if correct heading is reached, we have sailed through the wind and came out successfully on the other side. The boat should keep sailing on this side, close-hauled. Secondly, the next point is not a midway point, but a way point. This means the next target is not within the no-go zone, and tacking is not necessary. Thirdly, when a target is reached, and this target was a midway point, the variable is again set to true. This will last until one of the above cases is reached, resetting the variable to false, and the angle of attack to 10.

5.1.4 Batteries

In the beginning of this year of the project, it was discovered a lack of power from the batteries. On simple in-door tests with newly charged batteries, they could simply not supply enough power. As they are supposed to support the boat in harsher conditions and over longer time, and are crucial for the normal operations, this was a problem. During these tests, components in the system would shut down due to lack of power supply. The system would thus not work properly.

For each of the test instances where the power supply was a problem, a standardized procedure was followed. The full system, or subsystems of it, was to be used. The batteries were charged beforehand, using a 29.4 volts power supply, and a current limit of 4.32 amps. This is 30% of the capacity of the batteries, used as described in their manual (*Blyakkumulator* 2023). The values were both doubled from the recommended values for one battery because of the setup used with the four batteries, as described in section 3.1.3. The batteries were then charged until the amps out of the power supply showed about 3% of the capacity. This is 0.216 amps for one battery, and 0.432 for the setup used on the boat.

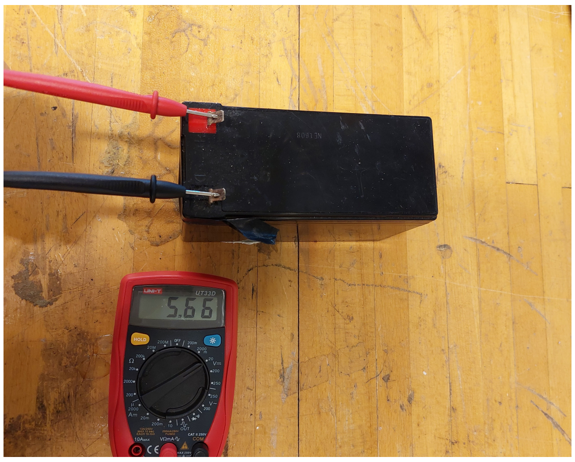


Figure 31: Measuring of batteries for logging. As shown, the voltage is way lower than the nominal 12 volts.

After the problem was discovered, tests were done on single batteries. From a full charge, the voltage was regularly measured. Over short amounts of time, the voltage dropped significantly, indicating an unhealthy battery. From the manual (*Blyakkumulator* 2023), it turns out the lead acid batteries, used for the boat, can handle low charge over time poorly. Most likely, the battery cells have become damaged because of this between project hand-overs.

For further development and testing of the boat, a set of four new, similar batteries have been acquired. To keep them working optimally, they should regularly be fully charged going forward. They are happy as long as long periods of time with low charge can be avoided. When the boat is ready, it is recommended to switch the batteries out. Replacing them with types that can handle low charge for extended amounts of time is optimal. Suggested are lithium based batteries, which thrive on a lower than fully charged level.

5.2 Building a test trolley

As described in section 2.4, testing is an important part of development. As sea trials are now cumbersome and requires planning, transport and several people, an easier way of testing is needed. Rather than full blown sea trials, a test trolley was built. It is a trolley carrying all the control equipment of the boat, as well as most of the hardware, sensors and control surfaces. This makes testing of entire systems in real life available by just going outside to an open field. Using a phone or computer, the intended headings can be observed and followed by pushing the trolley around. This is an incredible possibility to thoroughly test new and old solutions in real life.

5.2.1 Installing the sail

An earlier iteration this year on the trolley proved the original size too small to fit the length of the sail. This meant a limited angle of possible movements of the sail, and would also require software changes to hinder the sail from going there. A thought was to remove the sail altogether, and only follow directions given by the control system and push the trolley accordingly. However, this would take out an important test parameter, namely the possibility to observe correct positioning of the sail under different maneuvers.

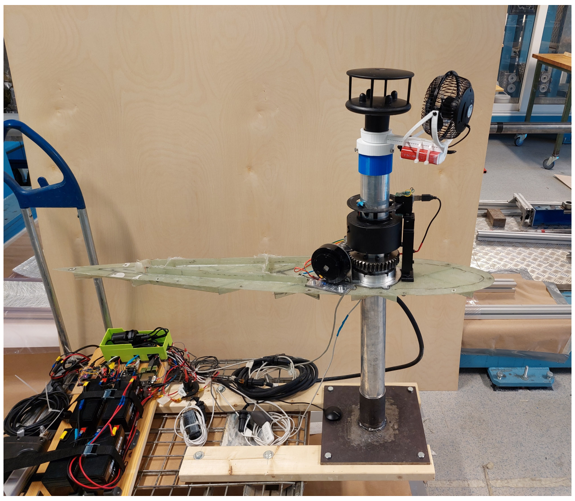


Figure 32: The sail base mounted on the trolley with beams.

To be able to fit the sail properly on the trolley, allowing easiest operation and full range testing, a few adjustments were made. Firstly, the size of the trolley was extended. This was necessary to allow full range of the sail. The solution was to bolt two wooden beams to the trolley, making a platform for the sail extended out in the front. A wooden board was used underneath to secure the bolts from slipping through the net of the trolley. A previously made base was used for fastening the sail, and already has holes for bolting it.

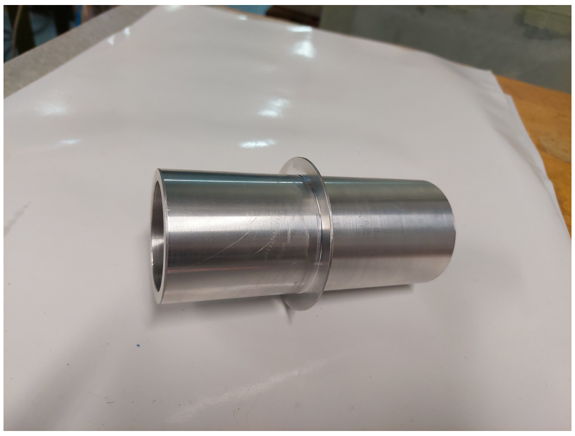


Figure 33: The new plug made for mast assembly. Made a bit smaller to replace the cold welded plug.

Secondly, the mast was removed. Using the full sail is unnecessary as all directions are shown with the base. Also, all the control electronics is mounted on the base, and the wind sensor can still be used atop the base. For the splitting of the mast to be possible, the old plug between the base and the rest of the mast had to be cut. The fit was too tight, and as both parts were made of aluminum, a partial cold welding had happened. The plug was cut and removed using a bayonet saw to reach within the mast rods. A new plug was ordered with a slightly looser fit. It might also be coated to avoid cold welding again. Two bolts will secure the three parts together.

5.2.2 The new electronics board

In the pre-study (Project Thesis, 2022) it was discovered that the old solution for containing the electronics had to be switched out. The box used was simply too large to be able to fit on the smaller prototype with the new hull. For this reason the electronics were moved over to a wooden board. This is a quick and durable solution for rapid testing. The board is also made small enough to fit on the trolley made for testing. However, as described in the pre-study thesis, the lack of dimensions and complete design of the new hull left the positioning of the electronics open. This has now changed and enough information is available for positioning and fastening the electronics.

A requirement for rapid testing in this scenario is that the electronics can be easily loosened from the board. This ensures availability to switch parts, check connections and debug isolated devices. Yet, the devices should be securely fixed and not able to move around or fall loose. To meet these requirements, bands and strings where chosen, in addition to some 3D-printed brackets designed for the batteries. Strong bands are used with the brackets on the batteries as their weight is substantial. The brackets will stop any movement along the plane of the board, while the bands will hold the batteries tight to this plane. For the microcomputers, the simple solution of strings where chosen. It is strong and flexible enough to make a secure fix to the board. This while keeping the possibility of untying or cutting the strings if necessary.

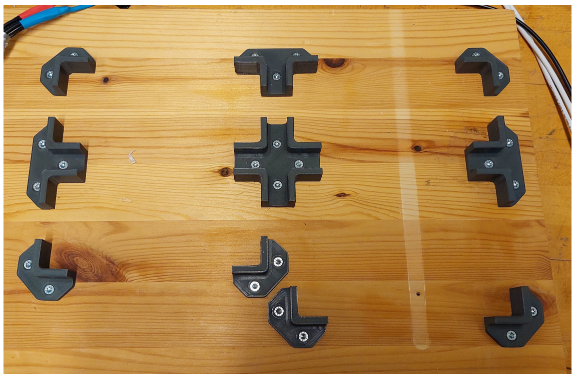


Figure 34: 3D-printed battery brackets, set up in their positions around the batteries.

The position of the electronics on the board where chosen based on wiring and natural grouping. The batteries as one group and the small, boarded controllers as another need to be connected within themselves. Thus, the batteries are placed together, and the other electronics is placed together. This has also helped cleaning up the wiring, trying to collect it in more of a wiring harness. This makes the connections clearer and will help debugging and visual inspections. Another advantage of placing the batteries together is that they can all be stored as low as possible in the boat. Placing the board with the batteries down will thus lower the center of gravity of the boat. Lastly, this makes the smaller electronics go on top. These are the ones connected to most devices out on the deck, and the connections will with this be shorter and easier available.

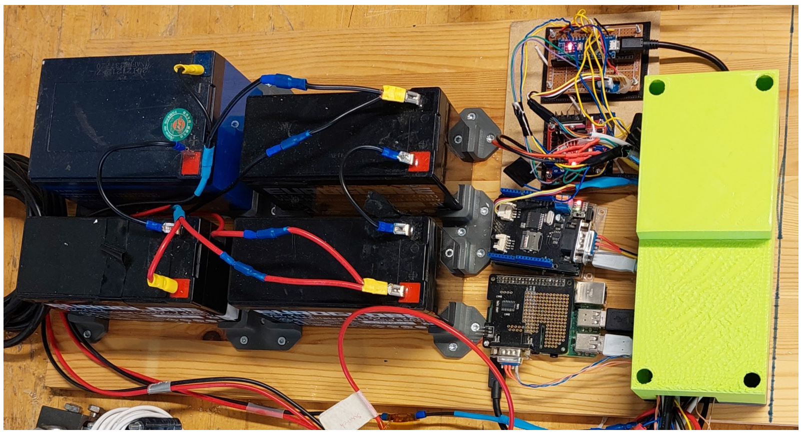


Figure 35: The new electronics board.

To collect all the wiring in one place, a box was designed and 3D-printed. It consists of a floor and a box lid that slide together around the wiring. It has slits to allow wires and cables in and out. This box will act as a protector to the wiring when moving the board around and when testing. A big advantage is also to collect all the outgoing cables and wires together for strength. Forces will pull the wires during testing or sailing, and single wires can easily snap or damage their isolation in these cases. Collecting all together through a single hole in the wire box will give some strain relief. In addition, they are clamped together with firm rubber bands to spread the forces. As the box is fastened to the electronics board, single wires and cables to the control devices will not be ripped out.

5.2.3 Simulating wind

To properly test different parts of the boat under different conditions, a controllable, simulated wind is necessary. This gives the means to test the sail angling, reaction and different added maneuvers. The tests should be constructed with care and proper focus, as described in section 2.4. To simulate different wind conditions for the tests, a fan with a holder was constructed. This is the same setup used for the tests in sections 4.2 and 4.3.

The setup needed to be light enough to be put a top the sail, where the wind sensor is. It also needed the ability to be manipulated into different directions. This is to achieve different states, point of sails and maneuvers. One of the main maneuvers to be tested is tacking. To meet these requirements, a ring mount with a jig for a fan was designed. It is simple and light in the construction. The ring mount was printed in two separate parts to be able to thread the other one on before gluing it shut. This secures the fan holder while still enabling the fan to swing around the wind sensor. The flexibility in the fan holder lets two rubber knobs be tightened between the jig and the fan to secure the fans position.

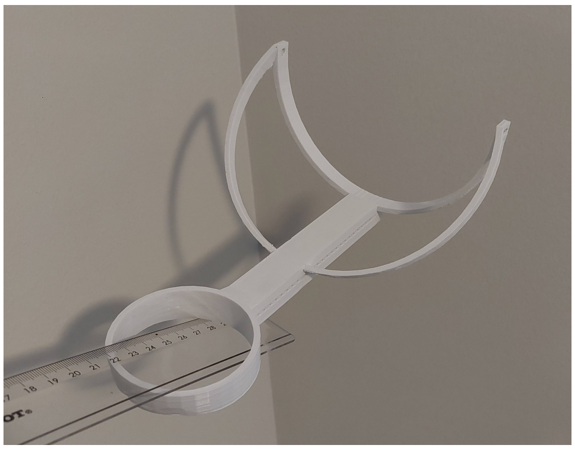


Figure 36: 3D-printed holder for a fan.

5.2.4 Fastening of small equipment

All the loose equipment meant to be mounted around the boat had to be secured safely, but available on the trolley. As it is a test rig, fastening equipment permanently is a bad idea. The sail and electronics board have already been placed, with the latter using the same straps that hold the batteries to secure itself to the trolley. With its heavy weight from the four batteries, its placement is also a counterweight to the protruding mast and sail jig. As mentioned, the wind sensor still fits atop the partial mast, still very much able to read the wind.

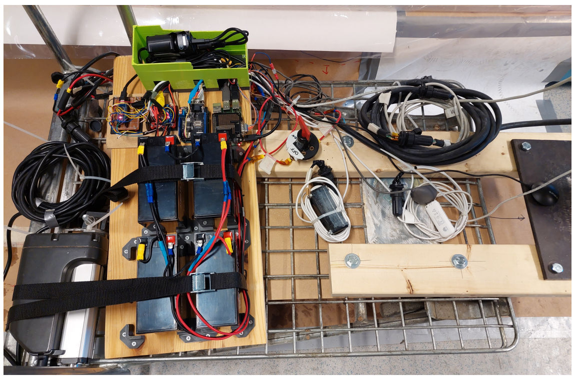


Figure 37: The trolley surface with cables, controllers and sensors.

For the smaller parts, sensors are mostly placed around stable places. Rolls of wires and cables are placed low on the trolley, and secured with strip ties. These also make stable placements for sensors, as well as giving them a dampening ground to lay on.

The rudder actuator is also mounted on the trolley. This gives a good indication if the heading is not close enough to the one set by the navigation system. The actuator is mounted in the back with the actuated arm clearly visible to read out changes. It is locked in place with one of the straps from the batteries, long enough to secure both.

5.3 Simulating movement

As discovered in section 4.1, the boat would not enter maneuvers like tacking without a GPS lock. This is because these maneuvers are dependent on a distance traveled, and are blocked without GPS lock. In addition, missing GPS lock means there are no positional data to be used for calculations. This makes the navigation uncertain. Because of this, navigation calculations have been limited to only steer properly when all necessary sensor data is available. As a result, testing is made harder. As described in section 2.4, the possibility to test subsystems of the boat is important for qualitative progress. Thus, it should be possible to do tests while not having all sensors available.

Testing is made harder when not available to do in the lab. Testing outside introduces a lot of unnecessary factors. Testing with these will be important at a later stage, but testing with controllable environment is important for debugging and testing new features. Doing it inside eliminates a lot of variation. At the same time, weather will not be a factor. In the testing stage, the equipment aboard the boat is usually not water proof, as it is time consuming and wasteful. The wind is also an important factor for testing, and control over it is essential for testing new sailing programs. Aspects of the environment and of the boat is also much more available for manipulation during testing. Last, testing inside also gives time and room to constantly check values and states on a connected computer nearby.

These reasons are why a program to simulate movements of the boat was made. It enables testing on sail control and navigation controls while in a controlled, still environment. With the switch of one variable, a GPS lock is no longer necessary for the navigation calculations to run properly. The variable is a Boolean, set to true when wanting to simulate travel. It is passed on to the GPS-program, which retrieves location data and updates the coordinate state of the boat. When the variable is true, the retrieval is replaced by a method. This method is a simple logic, traveling at a speed set in the first lines of the method. It will iterate the boat towards its next target, a way point or midway point. The distance of travel is updated by the formula

$$dist = speed * dt * (1 or - 1)/coordlength, \tag{2}$$

where $speed$ is the one set in the method, dt is the delta time since last update, and $1 or - 1$ is used for direction. Dividing by the coordinate length is necessary to be able to add the resulting value to the coordinates. Each latitude has a set distance between each other of about $111km$. The distance between longitudes are varying as you move north or south, but one degree is about $49.8km$ at the testing site.

All the necessary values in the method are retrieved from the navigation class. It saves the time of last update and the previous coordinates to be updated from. The navigation class also holds the next target of the travel, including midway points when they are necessary. The code changed can be found in the appendix, C.

5.4 Adding a sensor

The mission of the boat will most likely be data collection of different environmental aspects. As such, it will need a collection of sensors to capture this data. As the control system and sailing algorithms are closing in on a state fit for the mission, sensors will also have to be implemented. They need to get power and control from the existing system, while having a portal for saving the data. This will also be a place to get the data to send it to interested parties. The control system has its way to go before being ready, but implementation of sensors can already be done to prepare for the mission. This also makes sense from a planning view, as the sensors also need testing and integration time with the control system.

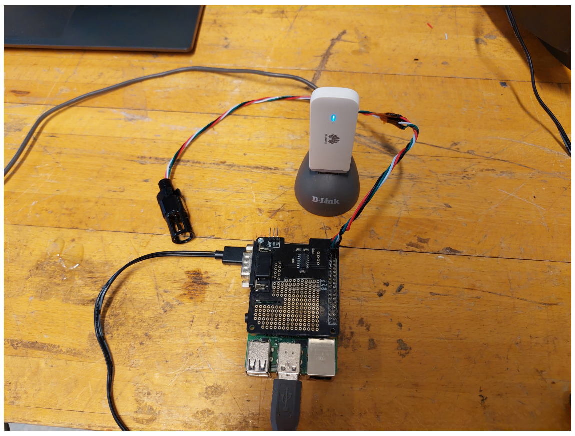
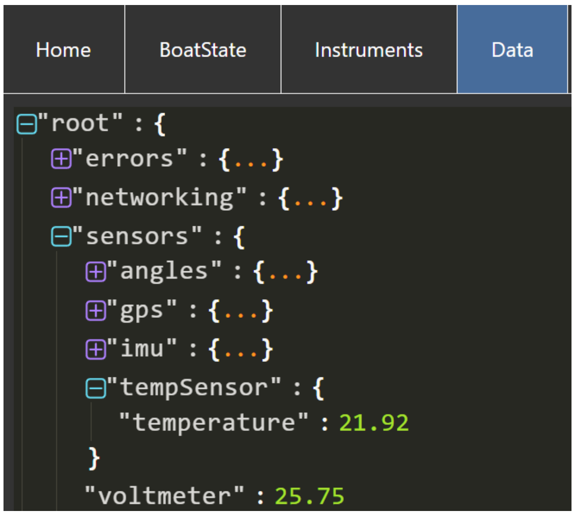


Figure 38: Implementation of temperature sensors with test setup.

For a test, and beginning of sensor data, one temperature sensor has been implemented. This will most likely be usable in the future, and the process will showcase how more sensors can be added. An extended hub might be needed in the future to connect all wanted sensors. For this test, the implementation and data collection were in focus, and so the sensor is simply connected directly to the Raspberry Pi, using its I^{2}C (Inter-Integrated Circuit) interface. This is a standard protocol used by a lot of sensors and are in addition quite similar to the alternatives. Four wires are connected to the pins on the Raspberry Pi, consisting of power, ground and two communication pins of the I^{2}C protocol. After enabling this protocol on the Raspberry Pi, the sensor is available for the software.



```
⊟"root" : {
  ⊞"errors" : {...}
  ⊞"networking" : {...}
  ⊟"sensors" : {
    ⊞"angles" : {...}
    ⊞"gps" : {...}
    ⊞"imu" : {...}
    ⊟"tempSensor" : {
      "temperature" : 21.92
    }
    "voltmeter" : 25.75
```

Figure 39: Together with the other sensors, the temperature sensor data is also available in the web app, and is also logged.

To add new sensors to the system, a few changes will have to be made in the code. First, a file for the sensor is added. This makes the sensor class, and holds initialization functions as described by the documentation for the specific sensor. This class is used by the run program to initialize the sensor, make a connection with it and set up a way of retrieving data. Next, it has to be added to the run program, to assign it a thread to keep running in the background. This is done by adding a sensor process and assigning a thread to it. Next, the data from the getter function in the sensor file has to be added to the states. This makes it available for uploading to the web app. Last, the thread and process has to be added to the exit event to close the connection properly on system shutdown. All the details of the temperature sensor and the changes in the code are in the appendix, D.

6 Results

Following this year's development, the boat is a few steps closer to being ready for its mission. The system has been mapped out, and the code commented in more detail. Hopefully, this will ease the take over for further development. Most of the practical sailing problems have been sorted out from the sea trial conducted last year. Yet, due to the building of a new hull taking longer than expected, the new solutions have not been tested in a full scale. The following subsection will show some tests performed to verify the solutions found for problematic subsystems.

By building a test trolley with supporting simulation capabilities, testing have become a lot easier. Before the boat is ready, a lot of software has to be developed, and testing will be crucial. The test trolley makes a simple, movable system for testing response to several inputs. For now it has been sail control based on wind inputs and heading. In the coming years, obstacle avoidance and some sort of awareness to other vehicles has to be developed. Testing during development of this will benefit greatly from the mobility and possibilities of a hand-controlled boat on wheels. In addition, software developed for simulating coordinates and movement is invaluable for testing. With these, a lot of tests can be done quickly and accurately, still placed inside the lab.

For tacking, correcting the calculation of true wind is expected to help a great deal. Following the sea trial from last year and the results for it in their report, (Pleym and Ølstad 2022), this fix should help on the problem with speed. Correcting the sail angles and making sure the boat is outside the no go-zone will help it catch the wind better. It will also increase the time of which the wind is available for propulsion. In addition, as suggested in their report, the sail is now being assigned a neutral position to minimize drag. This will help it during the crucial time of turning through the eye of the wind.

The next steps on the development on the boat will most likely be dependent on more sensors. Either mission specific for picking up valuable data, or more sensors to perceive the surroundings, e.g. for object detection and avoidance. A sensor has been added for testing, and was chosen to be useful on the mission. Its implementation shows how new sensors can be added to the existing system. The implementations is also described in section 5.4, and can be seen in greater detail in the files in the appendix. In addition, previously implemented sensors used for sailing shows how values can be used for navigation tasks on the boat.

6.1 Concluding tests

As a proper sea trial was not possible, the changes made had to be verified in other ways. Tests have been set up to look at how the changed subsystems work and perform. The tests are mostly done in the same manner described in section 4, originally designed to uncover any current flaws of that version. This section will describe the tests and go over the meaning of the results.

6.1.1 Tacking

Following the test in section 4.2, showing the problems, section 5.1.3 came up with new solutions. It was hoped to get a new sea trial to test all changes. Unfortunately, building of a new hull was not finished before this year's project end. Instead, a small test, similar to the one in 4.2 was conducted to verify the changes made to the algorithms. All changed code will be in the appendix for anyone interested. Version control is on GitHub for further development.

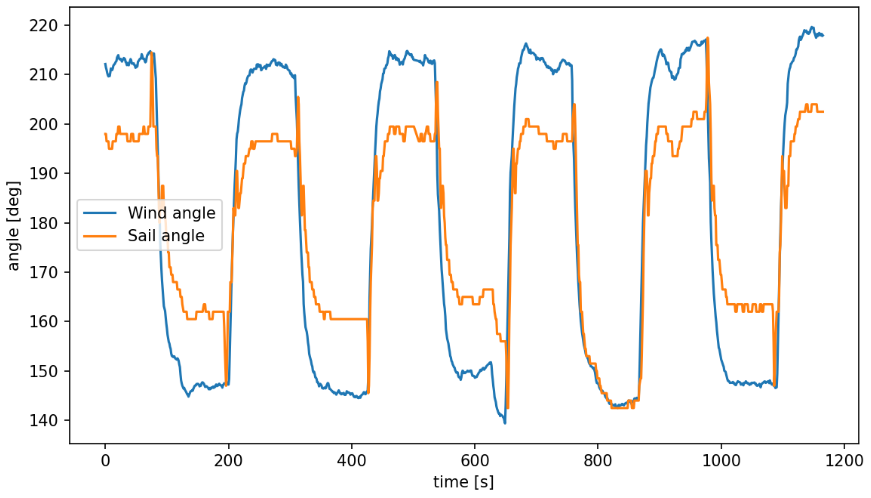


Figure 40: Wind and sail angles from the new test.

The skewed distribution shown in section 4.2 can no longer be seen during tacking. Correcting the true wind calculation removed this unbalance around the 180-degrees line. From the graph, one can see that both sides are extended similarly out from the 180 degree zero-line. After adjusting the angles 180 degrees for easier reading, this line is what marks the wind coming directly against the boat. With a constant true wind direction, the lines extend about 20 degrees out from the zero line. This is due to the boat turning during the tacking maneuver. The turning keeps the boat close to the wanted course, while staying outside of the no-go zone. This even maneuvering shows results hinting at a working tacking maneuver. This is promising towards a sea trial that should happen early in next year's development.

Another problem discovered in last year's development, (Pleym and Ølstad 2022) is also shown in section 4.2. It is further described and explored in section 5.1.3, and is regarding the sail angle during tacking. As the boat is turning through the eye of wind, it needs to minimize drag to maximize chance of successful maneuvering. After the changes done, the results in figure 40 shows a promising new trend. As the boat tacks, which can be seen by the changing wind, the sail angle follows it. This means that the boat collects speed during close-hauled sailing, then neutralizes the sail as soon as it starts turning. The resulting angle of attack of 0 degrees means the sail is performing the least it can as the wind tries to push against the wanted direction.

6.1.2 True wind

After the calculation of true wind was discovered to be wrong and its implications, a fix was done in section 5.1.1. Following this, a new test was conducted to verify the solution. The test was done in a similar manner to the one described in section 4.3. This solution is also a great part of solving the tacking issues seen from before. As mentioned in the previous test, this solution will also have to be tested in a sea trial. Only then can the full system be tested in the real application for verification. This test will, however, verify the solution as a sub-system, independently of all others. The changes made to the algorithm can be found in the code in the appendix.

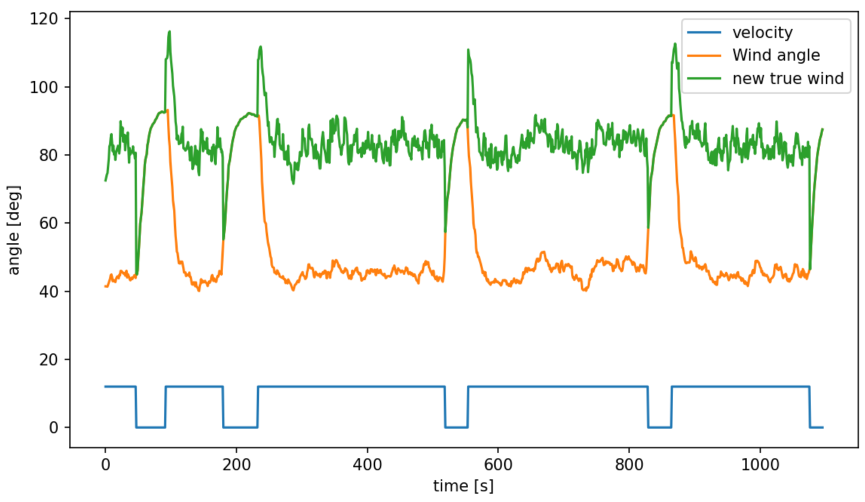


Figure 41: The measured, apparent wind shown with the velocity of the boat and the resulting, calculated true wind.

The graph above shows the true wind calculated by the algorithm in green. The blue line shows whether the boat has a velocity or not, and the orange line is the apparent wind, measured by the wind sensor. One can see the measured wind changing direction as the boat has a velocity, creating a head wind, which is correct with reality. For the test, the head wind is coming from directly in front of the boat, and the true wind is set to the right, at 90 degrees. When both winds play a role, it is the task of the algorithm to determine the true wind based on the speed of which the boat is moving. The green line clearly show a calculated true wind of around 90 degrees while the boat is moving. This is the correct calculation, as the true wind fan is placed at 90 degrees.

The big spikes in the green and orange lines are two different phenomena happening in the test. To start with the orange lines, the steep increases are a result of the boat velocity. This creates head-wind, interfering with the true wind. This increases the angle until the measured angle is the result of the two input speeds and angles. For the green line, the spikes are happening because of the nature of the fan. It is stopped as the boat speed is set to 0, but its angular momentum keeps it going for a while. This creates a gradually descending head wind while the velocity is suddenly 0. Where the green and orange line aligns, the boat is stopped and the head wind disappears. This makes the measured wind the true wind, aligning the two lines.

7 Further work

Several of the changes made described in this thesis have been verified. Yet, a full blown sea trial has not been conducted due to the testing hull not being ready on time. To properly test the system, especially with the new hull, is therefore strongly suggested. This will map out any remaining issues, and might uncover new ones. Importantly, it can also be used to tune the values of the system with the new hull. This includes no-go zone limits, optimal angle of attack on the sail and angles for switching between sailing modes. The hull and its robustness will also be an important item of the test.

The navigation system will at some point need an upgrade. Awareness of land and reefs marked on the map will have to be included in the navigation system. Sea maps are usually well updated and shows most stationary hazards in the water. In addition, sensory systems for detecting objects will have to be added. This is important to be able to avoid mobile or new objects, such as other boats or installations not yet marked on a map. The route planning will have to take any obstacles into consideration to avoid them. This might also be a good idea for weather forecasts. Taking wind direction and speed, waves and heavy rain into path planning can save on maintenance needs, power consumption and speed of travel.

A system for uploading and logging sensor data is already in place, but changes might be needed. A platform where interested parties can log in and get their ordered data might be a possibility. For now, the commercial sensor data is uploaded with the operational data from the boat. In the future, the load might have to be split up. If the amount of data sent is greatly increasing, a prioritization system can be needed. For example if the boat has bad reception, some boat states can be prioritized for monitoring and orientation. An independent system with a transmitter and a web based platform might be a good solution if the load is heavy.

Lastly, two systems for safety in operability can be a good investment of time. The first is a low power state. This can be while waiting for new missions or target areas, or when traveling to new ones. Sensors not needed for navigation and their logging can be turned off to save on power. The same can be done with other systems not crucial for long travels. In addition, a good system of power regeneration should be added. Already a solar panel can fit in the sail, which can be a good solution. Another one could be a generator in the water running on the power harvested with the sail. The second is to add some sort of safety system aboard. For some components, this can be redundant copies, taking over if something happens to the original. Another solution might be an emergency mode, where speed and travel is minimized. At the same time, only crucial data can be transmitted, like coordinates. At least, some sort of fail-safe system should be implemented to minimize damage and loss. These are some suggestions to get started on further development towards a boat ready for its mission in the years to come.

8 Conclusion

The development of the autonomous surface vehicle has been continued. During this year, several issues discovered or suspected from previous testing has been worked on. It has been a fitting mix of elaboration by testing, and continued work to sort the problems out. Simultaneously, testing has been made a lot easier. Software has been written to support several likely scenarios in testing, and hardware has been built to enable thorough testing on land. Overall, the state of software has improved, as well as some hardware. Although a sea trial was not possible, due to external reasons, several tests have been done on affected subsystems to verify the fixes.

After testing, problems with sailing and sensor impressions have been solved. The wrong calculation of true wind has been corrected, which also improved the prospect of tacking maneuvers. The tacking has also benefited from updated algorithms, neutralizing the sail when turning through the eye of the wind. Overall operability has also been increased from previous development. This is due to two factors. The first is a change of batteries. These do now have more capacity, and work properly. The factors that decayed the old ones have been listed in this report to avoid the same mistakes again. The second is an improvement of electronic connections. Wires and cables have been put in a way to maximize strength against pulling forces. This, together with resoldering bad solder points should make a more rugged system, more resistant to damage.

With the improvements on tacking, the boat should be more capable of sailing to its target. This is regardless of the wind. Existing sailing algorithms have in the earlier sea trial proved themselves capable, while tacking was the one lacking. With the improvements, tacking is a promising maneuver the boat can use when the wind is coming from the direction of the target. The electronics system's increased resistance to damage enables more firm sailing and longer up times without hardware failing. This will apply to the mission, and is already in play for testing.

The availability of testing has also improved immensely with this year's development. A rig built on a trolley enables testing on land, making it available at the lab and with no equipment for transportation. Specific input can be easier managed to test specific solutions, cases or aspects. To go with it, simulation software has been built to overcome challenges met with getting a GPS lock while testing. This has made stationary testing possible, with immense improvements to controlling the environmental factors during testing. This has greatly enhanced testing in a good spirit of prototyping.

Lastly, the boat has a mission it should be able to complete. The sailing capabilities and ruggedness means nothing if the boat does not serve a purpose. For the mission, data about the environment is the main objective. Thus, a way of connecting sensors to the system has been shown. This is implemented to report to the Raspberry Pi. From there, communication with the land based server enables uploading the data, and logging it for later use or distribution. The steps needed are showcased, suggesting a method of implementation. The way of uploading and logging is a project standard, capable of extension to several new sensors.

Bibliography

About us (2022). Raspberry Pi Foundation. URL: https://www.raspberrypi.org/about/ (visited on 3rd Dec. 2022).

Anatomy of a sailboat (2022). Sailrite. URL: https://www.sailrite.com/anatomy-of-a-sailboat (visited on 28th Oct. 2022).

Andy Batchelor, Lisa B. Frailey (2016). *Cruising Catamaran Made Easy: The Official Manual For The ASA Cruising Catamaran Course*. American Sailing Association.

Bethwaite, Frank (2007). *High Performance Sailing*. Adlard Coles Nautical.

Blyakkumulator (2023). Biltema. URL: https://www.biltema.no/bil---mc/bilbatterier/blyakkumulatorer/blybatteri-12-v-72-ah-151-x-65-x-95-mm-2000047907 (visited on 13th Apr. 2023).

Calculating apparent velocity and angle (2023). wikipedia.org. URL: https://en.wikipedia.org/wiki/Apparent_wind (visited on 12th Apr. 2023).

Cliffe, Tom (1994). *The Complete Yachtmaster*. London: Adlard Coles Nautical.

Cunliffe, Tom (2016). *The complete day skipper: Skippering with confidence Right From the Start*. Bloomsbury Publishing.

Elverum, Christer W. and Torgeir Welo (2015). 'On the use of directional and incremental prototyping in the development of high novelty products: Two case studies in the automotive industry'. In: *Journal of Engineering and Technology Management* 38, pp. 71–88. ISSN: 0923-4748. DOI: https://doi.org/10.1016/j.jengtecman.2015.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0923474815000405.

Holtet, Jan A. (2022). *Misvisning*. Store norske leksikon. URL: https://snl.no/misvisning (visited on 2nd Dec. 2022).

Houde, Stephanie and Charles Hill (1997). 'Chapter 16 - What do Prototypes Prototype?' In: *Handbook of Human-Computer Interaction (Second Edition)*. Ed. by Marting G. Helander, Thomas K. Landauer and Prasad V. Prabhu. Second Edition. Amsterdam: North-Holland, pp. 367–381. ISBN: 978-0-444-81862-1. DOI: https://doi.org/10.1016/B978-044481862-1.50082-0. URL: https://www.sciencedirect.com/science/article/pii/B9780444818621500820.

Jobson, Gary (1990). *Championship Tactics: How Anyone Can Sail Faster, Smarter, and Win Races*. New York: St. Martin's Press.

— (2008). *Sailing fundamentals*. Simon and Schuster.

Kimball, John (2009). *Physics of sailing*. CRC Press.

Kjerstad, Norvald (2022). *Deviasjon (kompassavik)*. Store norske leksikon. URL: https://snl.no/deviasjon_-_kompassavvik (visited on 2nd Dec. 2022).

Misvisning og deviasjon (2022). Sjøakademiet AS. URL: https://www.xn--btfrerprven-x8a3wf.no/emner/retting-av-kurser/ (visited on 24th Nov. 2022).

Osnes, Andreas (2022). *Avdrift (sjøvesen)*. Store norske leksikon. URL: https://snl.no/avdrift_-_sj%C3%B8vesen (visited on 2nd Dec. 2022).

Patrick F. Rynne, Karl D. von Ellenrieder (2010). 'Development and Preliminary Experimental Validation of a Wind- and Solar-Powered Autonomous Surface Vehicle'. In: pp. 971–983. DOI: 10.1109/JOE.2010.2078311.

Pleym, Adrian Skogstad and Magnus Westbye Ølstad (2022). 'On development and validation of an autonomous sailboat'. In: URL: https://hdl.handle.net/11250/3023721.

Point of sail (2022). Wikimedia Foundation. URL: https://en.wikipedia.org/wiki/Point_of_sail (visited on 25th Oct. 2022).

Raspberry Pi OS (2022). Raspberry Pi Foundation. URL: https://www.raspberrypi.com/software/ (visited on 4th Dec. 2022).

Roland Stelzer, Tobias Pröll (2007). 'Autonomous sailboat navigation for short course racing'. In: DOI: 10.1016/j.robot.2007.10.004.

Rousmaniere, John (1999). *The Annapolis book of seamanship*. Simon and Schuster. URL: https://books.google.no/books?id=xRqzoX04v5AC&lpg=PR9&ots=RTUPaf4YkK&dq=the%20annapolis%20book%20of%20seamanship&lr&hl=no&pg=PR9#v=onepage&q=the%20annapolis%20book%20of%20seamanship&f=false.

Sailing terms everyone should know (2022). asa.com. URL: https://asa.com/news/2021/07/07/sailing-terms-you-can-use/ (visited on 6th Dec. 2022).

Saoud, Hadi et al. (2015). 'Routing and course control of an autonomous sailboat'. In: DOI: 10.1109/ECMR.2015.7324218.

Upton, Liz (2022). *The Raspberry Pi in scientific research*. Raspberry Pi Foundation. URL: https://www.raspberrypi.com/news/the-raspberry-pi-in-scientific-research/ (visited on 3rd Dec. 2022).

What is Arduino (2022). Arduino. URL: https://www.arduino.cc/en/Guide/Introduction (visited on 6th Dec. 2022).

Zhenyu Yu Xinping Bao, Kenzo Nonami (2008). 'Course Keeping Control of an Autonomous Boat using Low Cost Sensors'. In: pp. 389–400. DOI: 10.1299/jsdd.2.389.

Appendix

A run.py

```python
import os
from navigation import Navigation
import websockets
import asyncio
import json
import time
import sys
import signal
import threading
import urllib.request
from ip import getExternalIp, getInternalIp
from ngrok import Ngrok
from gps import GPS
from uno import Uno
from nano import Nano
from windsensor import Wind
from tempSensor import TempSensor
from auth import getTokenURI
from utils import get, recvLikeArduino, set
from errorHandler import ErrorHandler

exit_event = threading.Event()

STATE = {}
PARAMETERS = {}

def initSensors():
    global PARAMETERS
    global STATE

    global gps
    global uno
    global nano
    global wind
    global tempSensor
    global ngrok
    global errorHandler
    global exit_event
    global navigation
    global simulate_GPS

    errorHandler = ErrorHandler(debug=True)
    gps = GPS(errorHandler)
    uno = Uno(errorHandler)
    nano = Nano(errorHandler)
    wind = Wind(errorHandler)
    tempSensor = TempSensor(errorHandler)
    ngrok = Ngrok(errorHandler)
    navigation = Navigation(errorHandler)
    simulate_GPS = True #True will simulate GPS movement for
     ↪   testing. Default False


def gps_process():
    global gps
    global errorHandler
    while True:
        if exit_event.is_set():
            break
        if not gps.connected:
            if exit_event.wait(10):
                break
            gps = GPS(errorHandler)
            continue
        else: gps.getCoordinates(simulate_GPS, navigation)
    print("gps_process terminating")


def uno_process():
    global uno
    global errorHandler
    global parameterChanged
    msg = ""
    dataStarted = False
    dataBuf = ""
    messageComplete = False

    while True:
        if exit_event.is_set():
            break
        if not uno.connected:
            if exit_event.wait(10):
                break
            uno = Uno(errorHandler)
            continue

        msg, dataStarted, dataBuf, messageComplete =
         ↪  recvLikeArduino(uno.arduino, dataStarted, dataBuf,
         ↪  messageComplete)
        if not (msg == 'XXX'):
            uno.messageRecv(msg)
    print("uno_process terminating")

def nano_process():
    global nano
    global errorHandler
    global parameterChanged
    msg = ""
    dataStarted = False
    dataBuf = ""
    messageComplete = False

    while True:
        if exit_event.is_set():
            break
        if not nano.connected:
            if exit_event.wait(15):
                break
            nano = Nano(errorHandler)
            continue

        msg, dataStarted, dataBuf, messageComplete =
         ↪  recvLikeArduino(nano.arduino, dataStarted, dataBuf,
         ↪  messageComplete)
        if not (msg == 'XXX'):
            nano.messageRecv(msg)
    print("nano_process terminating")

def wind_process():
    global wind
    global errorHandler
    while True:
        if exit_event.is_set():
            break
        if not wind.connected:
            if exit_event.wait(10):
                break
            wind = Wind(errorHandler)
            continue
        else: wind.getWind()
    print("wind_process terminating")

def tempSensor_process():
    global tempSensor
    global errorHandler
    while True:
        if exit_event.is_set():
            break
        if not tempSensor.connected:
            if exit_event.wait(10):
                break
            tempSensor = TempSensor(errorHandler)
            continue
        else: tempSensor.getTemperature()
    print("tempSensor_process terminating")

def ngrok_process():
    ngrok_count = 0
    updated = False
    while True:
        if ngrok_count > 12: break
        url = ngrok.getNgrok()
        if not url or url == get(STATE, "networking.ngrok"):
            ngrok_count += 1
        else:
            set(STATE, "networking.ngrok", url)
            updated = True
        if exit_event.wait(10): break
    if not updated:
        errorHandler.setErrorCode("0010")
    print("ngrok_process terminating")

def navigation_process():
    global nano
    global uno

    while True:
        if exit_event.wait(0.2):
            break

        if wind.connected and (gps.connected or simulate_GPS) and
        ↪  nano.connected and uno.connected and get(PARAMETERS,
        ↪  "auto"):

            # Turns on the Nano's auto targeting after heading
            if (not navigation.autoOn):
                navigation.autoOn = True
                nano.execute("Nano!Navigator:Start")

            nextCoordinate = get(PARAMETERS, "path.head")
            if nextCoordinate:
                try:
                    if (get(PARAMETERS, "path.new")):
                        set(navigation.DATA, "waypoint", {})
                        navigation.findNextTarget(get(PARAMETERS,
                        ↪  "path"))
                        set(PARAMETERS, "path.new", False)
                set(navigation.DATA, "updateParameters",
                 ↪  True)

            now = time.time()
            if (now - navigation.lastUpdate <
             ↪  (get(PARAMETERS,
             ↪  'piparameters.navigationFrequency') or
             ↪  navigation.frequency)): continue
            navigation.lastUpdate = now

            # Dummy data: coordinates={"lat": 63.41634,
             ↪  "long": 10.41030833333, "velocity": 0,
             ↪  "cog": 0}
            # GPS data: coordinates=gps.COORDINATES

             ↪  navigation.calculateTargets(coordinates=gps.COORDINATES,
             ↪  wind=wind.WIND, heading=get(nano.DATA,
             ↪  "heading"), path=get(PARAMETERS, "path"))



            if (get(navigation.DATA, "nanoCommand") != ""):
                nano.execute(get(navigation.DATA,
                 ↪  "nanoCommand"))
                set(navigation.DATA, "nanoCommand", "")
            if (get(navigation.DATA, "unoCommand") != ""):
                uno.execute(get(navigation.DATA,
                 ↪  "unoCommand"))
                set(navigation.DATA, "unoCommand", "")

        except Exception as e:
           print(e)

        if (get(navigation.DATA, "updateParameters")):
            #Parameters changed
            print("Parameters changed")

# Turns off the Nano's auto targeting after heading
elif not get(PARAMETERS, "auto"):
    if (navigation.autoOn):
        navigation.autoOn = False
        set(navigation.DATA, "waypoint", {})
        nano.execute("Nano!Navigator:Stop")
    elif (navigation.depower):
        now = time.time()
        if (now - navigation.lastUpdate < (get(PARAMETERS,
         ↪  'piparameters.navigationFrequency') or
         ↪  navigation.frequency)): continue
```

```
                navigation.lastUpdate = now
                if abs(get(nano.DATA, "rudderAngle")) - 8 > 0:
                    nano.execute("Nano!Motor:SetPosition;0")
                set(navigation.DATA,
                ↪   "targetSailAngle",round(get(wind.WIND,
                ↪   "wind_direction")))

                ↪   uno.execute(f'Uno!Navigator:SetPosition;{get(navigation.DATA,
                ↪   "targetSailAngle")}')


    print("navigation_process terminating")

def handleNewCommand():
    command = get(PARAMETERS, "command")
    if command != None and command != "":
        if ("Nano!" in command and nano.connected):
            nano.execute(command)
        elif ("Uno!" in command and uno.connected):
            uno.execute(command)
        elif ("Pi!" in command):
            executeCommand(command)
        else:
            errorHandler.setErrorCode("0120")
        set(PARAMETERS, "command", "")
        return True
    else: return False

def executeCommand(command):
    command = command.split('!')[1]
    if command == "Main:Shutdown": os.system("sudo nohup shutdown -P
    ↪   now")
    elif command == "Main:Reboot": os.system("sudo reboot")
    elif command == "Errors:Clear": errorHandler.clearErrors()
    elif command == "Nano:Reconnect": nano.connected = False
    elif command == "Uno:Reconnect": uno.connected = False
    elif command == "Main:Depower": navigation.depower = not
    ↪   navigation.depower
    elif command == "Main:DeleteMidWaypoint": set(navigation.DATA,
    ↪   "midWaypoint", {})
    else:
        errorHandler.setErrorCode("0121")



async def foreground(token, uri, first_iter):
    global STATE
    global PARAMETERS
    global wind
    global tempSensor
    global uno
    global gps
    global ngrok
    global navigation

    if not token or not uri:
        return
    try:
        async with websockets.connect(uri) as websocket:
            await websocket.send(token)
            while True:

                if exit_event.is_set():
                    errorHandler.setErrorCode("0025")
                    set(STATE, "errors", errorHandler.errors)
                    await websocket.send(json.dumps({"action":
                        ↪  "update", "type": "state", "data": {"errors":
                        ↪  get(STATE, "errors")}}))
                    break
                try:
                    data = await asyncio.wait_for(websocket.recv(),
                     ↪  timeout=5)
                except asyncio.TimeoutError:
                    continue

                if data == None: continue
                data = json.loads(data)

                if get(data, "type") == "parameters":
                    PARAMETERS = get(data, "data")
                    wind.parameters = PARAMETERS
                    navigation.parameters = PARAMETERS

                set(STATE, "sensors.angles.sail",
                    get(uno.DATA, "sailAngle"))
                # Must be changed to encoder values
                set(STATE, "sensors.angles.rudder",
                    get(nano.DATA, "rudderAngle"))
                set(STATE, "targets.rudder", get(nano.DATA,
                 ↪  "targetRudderAngle"))

                set(STATE, "sensors.voltmeter", get(nano.DATA,
                 ↪  "voltmeter"))

                # Getting current coordinates
```

```
set(STATE, "sensors.gps.lat", get(gps.COORDINATES,
↪   "lat"))
set(STATE, "sensors.gps.long", get(gps.COORDINATES,
↪   "long"))
set(STATE, "sensors.gps.velocity",
    get(gps.COORDINATES, "velocity"))
set(STATE, "sensors.gps.cog", get(gps.COORDINATES,
↪   "cog"))

# Getting current wind status
set(STATE, "sensors.wind.direction",
    get(wind.WIND, "wind_direction"))
set(STATE, "sensors.wind.speed", get(wind.WIND,
↪   "wind_speed"))

#Getting current temp status
set(STATE, "sensors.tempSensor.temperature",
    get(tempSensor.TEMPERATURE, "temperature"))

# Getting imu data
set(STATE, "sensors.imu.heading",
    get(nano.DATA, "heading"))
set(STATE, "sensors.imu.pitch", get(nano.DATA,
↪   "pitch"))
set(STATE, "sensors.imu.roll", get(nano.DATA,
↪   "roll"))

# Getting target values
set(STATE, "targets.sail",
    get(navigation.DATA, "targetSailAngle"))
set(STATE, "targets.rudder",
    get(nano.DATA, "targetRudderAngle"))
set(STATE, "targets.heading",
    get(navigation.DATA, "targetHeading"))
set(STATE, "targets.depower", navigation.depower)
# Getting target waypoint
set(STATE, "targets.waypoint",
    get(navigation.DATA, "waypoint"))
set(STATE, "targets.midWaypoint",
    get(navigation.DATA, "midWaypoint"))

# Set error codes
set(STATE, "errors", errorHandler.errors)

#print(STATE)
# Connecting to websocket and sending data

# Updating when a command is executed
```

```python
                if handleNewCommand():
                    await websocket.send(json.dumps({"action" :
                     ↪  "update", "type": "parameters", "data":
                     ↪  PARAMETERS}))

                # Updating when a waypoint is visited
                if get(navigation.DATA, "updateParameters"):
                    await websocket.send(json.dumps({"action" :
                     ↪  "update", "type": "parameters", "data":
                     ↪  PARAMETERS}))
                    set(navigation.DATA, "updateParameters", False)

                if get(PARAMETERS, "start") or first_iter:
                    first_iter = False
                    await websocket.send(json.dumps({"action":
                     ↪  "update", "type": "state", "data": STATE}))

                await asyncio.sleep(get(PARAMETERS, "delay") or 0.5)


    except Exception as e:
        errorHandler.setErrorCode("0021", e)


def signal_handler(signum, frame):
    exit_event.set()


def wait_for_internet_connection():
    counter = 0
    while True:
        try:
            response = urllib.request.urlopen(
                'https://api.ipify.org', timeout=1)
            return
        except:
            counter += 1
            if (counter > 120):
                errorHandler.setErrorCode("Error: Could not establish
                 ↪  an internet connection.", "0022")
                sys.exit()
            time.sleep(1)
            pass


if __name__ == "__main__":

    signal.signal(signal.SIGTERM, signal_handler)
    signal.signal(signal.SIGINT, signal_handler)
    signal.signal(signal.SIGHUP, signal_handler)

    initSensors()

    wait_for_internet_connection()

    # Adding current IP to state
    external_ip = getExternalIp()
    set(STATE, "networking.piIp", external_ip)

    # Getting internal_ip
    internal_ip = getInternalIp()
    set(STATE, "networking.piLocalIp", internal_ip)

    token, uri = getTokenURI()

    g = threading.Thread(name='gps_process', target=gps_process)
    w = threading.Thread(name='wind_process', target=wind_process)
    t = threading.Thread(name='tempSensor_process',
     ↪  target=tempSensor_process)
    n = threading.Thread(name='ngrok_process', target=ngrok_process)
    a = threading.Thread(name='navigation_process',
     ↪  target=navigation_process)

    # Arduinos
    su = threading.Thread(name='uno_process', target=uno_process)
    sn = threading.Thread(name="nano_process", target=nano_process)

    g.start()
    w.start()
    t.start()
    n.start()
    a.start()

    su.start()
    sn.start()

    first_iter = True

    while True:
        asyncio.run(foreground(token, uri, first_iter))
        errorHandler.setErrorCode("0020", "Lost connection to
         ↪  websocket server")
        token, uri = getTokenURI()
        if exit_event.wait(5):
            print("Waiting for threads to terminate")
            g.join()
```

```
            w.join()
            t.join()
            n.join()
            a.join()
            su.join()
            sn.join()
            break

print("Main process terminating")
```

B navigation.py

```
import math
from utils import get, set
from haversine import haversine, inverse_haversine

class Navigation:

    def __init__(self, errorHandler) -> None:
        self.errorHandler = errorHandler

        # Update frequency
        self.frequency = 3 # 3 sekunder
        self.lastUpdate = 0

        self.goalRadius = 10 # meters

        self.beatingLength = 50 # meter. Should be changed to a
        ↪  function of the total distance to target

        self.autoOn = False
        self.depower = False

        self.beatingLimit = 30 # Wind direction ±beatinglimit
        ↪  defines no go zone
        self.beatingLimitBuffer = 10

        self.angleOfAttack = 10 # degrees from the wind

        self.dragModeLimit = 135 # +- from the wind
        self.dragMode = None

        self.parameters = {}
        self.from_midwayPoint = False #param to see if last visited
        ↪  was midway point

        # Data dictionary
        self.DATA = {"waypoint": {}, "midWaypoint": {},
        ↪  "targetHeading": 0, "targetSailAngle": 0, "nanoCommand":
        ↪  "", "unoCommand": "", "updateParamenters": False,
        ↪  "prevPoint": {"lat": 63.41634, "long": 10.41030833333,
        ↪  "velocity": 0, "cog": 0}, "prevTime": 0}

    # parameters, path
    def findNextTarget(self, path):

        head = get(path, "head")
        tail = get(path, "tail")
        waypoints = get(path, "waypoints")
        next = get(waypoints, head)

        while next != None:
            if (not get(next, "visited")):
                set(self.DATA, "waypoint", next)
                break
            next = get(waypoints, get(next, "next"))

        if (next == None):
            set(path, "pathComplete", True)
            set(self.DATA, "waypoint", get(waypoints, tail))
            set(self.DATA, "updateParameters", True)



    # coordinates == gps.COORDINATES,
    # wind == wind.WIND,
    # heading == float,
    def calculateTargets(self, coordinates, wind, heading, path):
        waypoint = get(self.DATA, "waypoint")

        # If path is not complete,
        # and waypoint is not set or waypoint is visited
        # => find next waypoint
        if ((not get(path, "pathComplete") and get(path,
        ↪  "pathComplete") != None) and ((not waypoint) or
        ↪  get(waypoint, "visited"))):
            self.findNextTarget(path)

        # Update variable with new waypoint
        waypoint = get(self.DATA, "waypoint")

        if (not waypoint): return

        #if the boat is from midway point and not reached heading:
        ↪  AoA should be 0 to pass through wind
        alpha = 0 if self.from_midwayPoint else get(self.parameters,
        ↪  'piparameters.angleOfAttack') or self.angleOfAttack
        sailAngle = self.calculateSail(wind, alpha,
        ↪  get(self.parameters, 'piparameters.dragModeLimit') or
        ↪  self.dragModeLimit)

        bearing = get(self.DATA, "targetHeading")

        distance = float('inf')

        #if midwaypoint exists
```

```
if get(self.DATA, "midWaypoint"):
    #if it is inside no go zone, not valid -> delete it
    if self.insideNoGoZone(get(self.DATA, "midWaypoint"),
    ↪   wind, coordinates):
        set(self.DATA, "midWaypoint", {})
    #if valid, then update bearing and distance based on it
    else:
        bearing = self.calculateBearing(coordinates,
        ↪   get(self.DATA, "midWaypoint"))
        distance = self.calculateDistance(coordinates,
        ↪   get(self.DATA, "midWaypoint"))

#if no midwaypoint and waypoint inside no go zone
if not get(self.DATA, "midWaypoint") and
↪   self.insideNoGoZone(waypoint, wind, coordinates):
    #create midwaypoint and update
    self.calcMidWaypoint(waypoint, wind, coordinates)
    bearing = self.calculateBearing(coordinates,
    ↪   get(self.DATA, "midWaypoint"))
    distance = self.calculateDistance(coordinates,
    ↪   get(self.DATA, "midWaypoint"))
#if no midwaypoint and waypoint outside no go zone, update
↪   based on it
elif not get(self.DATA, "midWaypoint"):
    bearing = self.calculateBearing(coordinates, waypoint)
    distance = self.calculateDistance(coordinates, waypoint)
    self.from_midwayPoint = False #turning though eye is not
    ↪   needed

#bearing not yet reached, then send to update
if bearing != get(self.DATA, "targetHeading"):
    set(self.DATA, "targetHeading", bearing)
    set(self.DATA, "nanoCommand",
    ↪   f'Nano!Navigator:SetTarget;{round(bearing)}')
else:
    self.from_midwayPoint = False #target heading reached,
    ↪   turned though the eye

#sail angle not reached, send to update
if (sailAngle != get(self.DATA, "targetSailAngle")):
    set(self.DATA, "targetSailAngle", sailAngle)
    set(self.DATA, "unoCommand",
    ↪   f'Uno!Navigator:SetPosition;{round(sailAngle)}')

# need better calculation
#if distance to target lower than threshold
if (distance < (get(self.parameters,
↪   'piparameters.goalRadius') or self.goalRadius)):
```

```
            #if reached target is midwaypoint, delete it and travel
            ↪  to waypoint
            if get(self.DATA, "midWaypoint"):
                set(self.DATA, "midWaypoint", {})
                self.from_midwayPoint = True #last visited was
                 ↪  midway point
            #if reached target is waypoint, mark as visited
            else:
                set(waypoint, "visited", True)
                set(self.DATA, "updateParameters", True)
        return

    def sign(self, x):
        if x == 0: return 1
        return x / abs(x)

    def insideNoGoZone(self, waypoint, wind, gps): # Checks if a
    ↪  coordinate (waypoint) is inside the no go zone. Returns
    ↪  true false.
        twa = self.trueWindAngle(A=(get(wind, "wind_speed")),
        ↪  V=(get(gps, "velocity")), beta=(get(wind,
        ↪  "wind_direction")))
        tb = self.calculateBearing(gps, waypoint)# angle to target
        ↪  (target bearing)

        sum = tb - twa + 540
        delta = (sum % 360) - 180
        if abs(delta) < self.beatingLimit:
            return True
        return False

    def calcMidWaypoint(self, waypoint, wind, gps):
        twa = self.trueWindAngle(A=(get(wind, "wind_speed")),
        ↪  V=(get(gps, "velocity")), beta=(get(wind,
        ↪  "wind_direction")))
        tb = self.calculateBearing(gps, waypoint) # angle to target
        ↪  (target bearing)

        sum = tb - twa + 540
        delta = (sum % 360) - 180


        pos = (get(gps, "lat"), get(gps, "long"))
        direction = self.degToRad((twa + self.sign(delta) *
        ↪  ((get(self.parameters, 'piparameters.beatingLimit') or
        ↪  self.beatingLimit) + (get(self.parameters,
        ↪  'piparameters.beatingLimitBuffer') or
        ↪  self.beatingLimitBuffer))) % 360)
        target = inverse_haversine(pos, (get(self.parameters,
        ↪ 'piparameters.beatingLength') or self.beatingLength),
        ↪ direction, unit='m')

        set(self.DATA, "midWaypoint", {"position": {"lat": target[0],
        ↪ "long": target[1]}})


    def calculateBearing(self, coordinates, waypoint):
        curLat = get(coordinates, "lat")
        curLong = get(coordinates, "long")
        tarLat = get(waypoint, "position.lat")
        tarLong = get(waypoint, "position.long")

        y = math.sin(tarLong-curLong) * math.cos(tarLat)
        x = (math.cos(curLat)*math.sin(tarLat)) - (math.sin(curLat) *
        ↪ math.cos(tarLat)*math.cos(tarLong-curLong))
        theta = math.atan2(y, x)

        bearing = (self.radToDeg(theta) + 360) % 360
        return bearing or get(self.DATA, "targetHeading")

    def calculateDistance(self, coordinates, waypoint):
        cur = (get(coordinates, "lat"), get(coordinates, "long"))
        tar = (get(waypoint, "position.lat"), get(waypoint,
        ↪ "position.long"))
        distance = haversine(cur, tar, unit='m') #in m
        return distance

    def degToRad(self, deg):
        return deg * (math.pi/180)

    def radToDeg(self, rad):
        return rad * (180/math.pi)

    def calculateSail(self, wind, alpha, limit = 135): # Beta = wind
    ↪   angle. Alpha = angle of attack. limit = Limit for changing
    ↪   between lift and drag mode
        beta = get(wind, "wind_direction")
        if isinstance(beta, str):
            return get(self.DATA, "targetSailAngle")
        if beta <= 180: # Wind hits starboard side of sail
            if beta <= limit: # We are not in drag zone (tail wind)
                if self.dragMode and beta <= limit - 10: # We are in
                ↪   drag mode are 10 degrees outside of drag zone
                    self.dragMode = None # turn drag mode off
            else:
                if not self.dragMode: # We are not in drag mode, but
                 ↪  in drag zone
                    self.dragMode = - 90 # Turn drag mode on
            sailAngle = beta - alpha + ((self.dragMode + alpha) if
             ↪  self.dragMode else 0)
        else: # Wind hits port side of sail
            if beta > (360 - limit):
                if self.dragMode and beta > (360 - limit) + 10:
                    self.dragMode = None
            else:
                if not self.dragMode:
                    self.dragMode = + 90
            sailAngle = beta + alpha + ((self.dragMode - alpha) if
             ↪  self.dragMode else 0)
        return sailAngle

    def trueWindVelocity(self, A, V, beta):
        W = math.sqrt(A**2 + V**2 - 2*A*V*math.cos(beta))
        return W

    def trueWindAngle(self, A, V, beta): #A: wind speed, V: boat
     ↪  velocity, beta: wind direction
        if A == None or V == None: return beta
        if A == 0 and V == 0: return beta
        V = 0.514*V
        alpha = math.acos((A*math.cos(self.degToRad(beta)) - V) /
                          self.trueWindVelocity(A, V,
                           ↪  self.degToRad(beta)))
        if beta <= 180:
            return self.radToDeg(alpha)
        return 360 - self.radToDeg(alpha)
```

C gps.py

```python
import serial
import os
import time
from findSensors import getPortOfVendor
from utils import get, set
import math as m

class GPS:
    def __init__(self, errorHandler):
        self.connected = True
        self.errorHandler = errorHandler

        try:
            vendor = os.getenv('GPS_VENDOR')
            port = getPortOfVendor(vendor)
            if port == None:
                self.connected = False
                self.errorHandler.setErrorCode("0001", e)
            else:
                self.gps = serial.Serial(
                    port=port,
                    timeout=3,
                    baudrate=4800,
                    xonxoff=False,
                    rtscts=False,
                    dsrdtr=False
                )
        except Exception as e:
            self.errorHandler.setErrorCode("0001", e)
            self.connected = False

        self.COORDINATES = {"lat": 0, "long": 0, "velocity": 0,
        ↪  "cog": 0}

    def getCoordinates(self, simu_GPS, navi):
        if simu_GPS:
            self.COORDINATES = simu_coord(navi)
            return

        counter = 0
        limit = 10
        if not self.connected: return

        while True:
            counter += 1
            if (counter == limit):
```

```python
    return

try:
    line = self.gps.readline()
except Exception as e:
    self.errorHandler.setErrorCode("0030", e)

try:
    line = line.decode('utf-8')

    values = line.split(",")

    if (values[0] == "$GPRMC"):
        lat = values[3]
        long = values[5]

        ns = values[4]
        ew = values[6]

        velocity = values[7]
        cog = values[8]

        if (lat == ""):
            continue
        if (long == ""):
            continue
        if (velocity == ""):
            continue
        if (cog == ""):
            continue

        latdec = float(lat[0:2]) + float(lat[2:])/60
        longdec = float(long[0:3]) + float(long[3:])/60
        velocity = float(velocity)
        cog = float(cog)

        if ns and ns == 's':
            latdec *= -1

        if ew and ew == 'w':
            longdec *= -1

        self.COORDINATES = {"lat": latdec, "long":
         ↪  longdec, "velocity": velocity, "cog": cog}
        return
except Exception as e:
    self.errorHandler.setErrorCode("0031", e)
    continue
```

```python
#if simuGps true, this method return simulated GPS coordinates
def simu_coord(navi):
    speed = 0.3 #m/s

    #previous coords
    prevPoint = get(navi.DATA, "prevPoint")
    if prevPoint:
        prevLat = get(prevPoint, "lat")
        prevLong = get(prevPoint, "long")

    #target coords, get midwaypoint if exists, else waypoint
    waypoint = get(navi.DATA, "midWaypoint") if get(navi.DATA,
    ↪   "midWaypoint") else get(navi.DATA, "waypoint")
    if waypoint:
        tarLat = get(waypoint, "position.lat")
        tarLong = get(waypoint, "position.long")
    else: #if no target, return previous point and no velocity
        return {"lat": prevLat, "long": prevLong, "velocity": 0,
         ↪   "cog": 0}
    if get(waypoint, "visited"): #return same as above if waypoint
    ↪   visited and no new set
        return {"lat": prevLat, "long": prevLong, "velocity": 0,
         ↪   "cog": 0}

    #increment x and y towards target
    if float(get(navi.DATA, "prevTime")) == 0:
        set(navi.DATA, "prevTime", time.time()) #set time first time
        return
    time_passed = time.time() - float(get(navi.DATA, "prevTime"))
    ↪   #time since last update
    #go left/down if passed, right/up if not
    curLat = prevLat + speed*m.cos(m.pi/6)*time_passed*int((1 if
    ↪   prevLat < tarLat else -1))/111000 #divide from meters to
    ↪   coordinates
    curLong = prevLong + speed*m.cos(m.pi/3)*time_passed*int((1 if
    ↪   prevLong < tarLong else -1))/49780

    #set time for next dt and return data
    set(navi.DATA, "prevPoint", {"lat": curLat, "long": curLong,
    ↪   "velocity": 0, "cog": 0})
    set(navi.DATA, "prevTime", time.time()) #save for reference dt
    return {"lat": curLat, "long": curLong, "velocity": speed, "cog":
    ↪   0}
```

D tempSensor.py

```python
import tsys01
from time import sleep

class TempSensor:
    def __init__(self, errorHandler):
        self.connected = True
        self.errorHandler = errorHandler

        try:
            self.sensor = tsys01.TSYS01() #default at I2C bus 1
            if not self.sensor.init():
                self.connected = False
                self.errorHandler.setErrorCode("0008")
        except Exception as e:
            self.errorHandler.setErrorCode("0008", e)
            self.connected = False

        self.TEMPERATURE = 0

    def getTemperature(self):
        if not self.connected: return

        while True:
            try:
                self.sensor.read() #update value from sensor
                #print(self.sensor.temperature())
                temp =
                 ↪  float("{:.2f}".format(self.sensor.temperature()))
                self.TEMPERATURE = {"temperature": temp}
                sleep(0.2)
                return
            except Exception as e:
                self.errorHandler.setErrorCode("0038", e)
                continue
```