

Adrian Skogstad Pleym  
Magnus Westbye Ølstad

# On development and validation of an autonomous sailboat

Master's thesis in Engineering and ICT  
Supervisor: Andrei Lobov  
Co-supervisor: Andreas T. Echtermeyer  
June 2022



Norwegian University of  
Science and Technology



Adrian Skogstad Pleym  
Magnus Westbye Ølstad

# **On development and validation of an autonomous sailboat**

Master's thesis in Engineering and ICT  
Supervisor: Andrei Lobov  
Co-supervisor: Andreas T. Echtermeyer  
June 2022

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



---

## Abstract

Autonomous surface vehicles (ASVs) could be an effective method to monitor and collect oceanographic data. Wind power can make the operation cost-effective, more environmentally friendly, and work independently over long periods. With the emergence of cheaper, smaller, and more powerful computer and sensor technology over the last decade, ASVs can be developed at a lower cost and with more advanced capabilities.

This thesis develops a mechatronic system for an autonomous sailboat. A distributed software system is designed and developed to support testing, autonomy, and operator control. The system consists of three main parts: a client web application, a cloud server, and a sailboat system created with state-of-the-art technologies and with a focus on interoperability and reliability.

The mechanical system is comprised of a catamaran hull, a fixed-wing sail, and a rudder. An electronic system of sensors, actuators, and computers was created to enable the system to perceive and act on the surroundings. The thesis describes the development process from parts selection, mounting, and software integration. Some sensors and parts were designed and 3D-printed to achieve the goal of a working prototype within the project time frame.

A sea trial was conducted as a full system integration test. Weather conditions during the test were harsh, with temperatures below  $5^{\circ}C$ , and intermittent hail and rain showers. Wind speeds was ranging from  $0.5 - 11.4m/s$ , averaging at  $4.3m/s$ . The sailboat operated autonomously with its battery power, with two operators monitoring on shore. The test showed promising results, with the mechanical and electronic systems performing well. The cloud server did perform very well. However, there were some issues with the boat's software preventing it from sailing in many cases. Further development and testing are required to achieve a fully autonomous sailboat.

---

## Sammendrag

ASVer kan være en effektiv metode for å overvåke og samle inn havdata. Ved bruk av vindkraft kan operasjoner både være kostnadseffektive, miljøvennlige, og fungere uavhengig over lengre perioder. Med utviklingen av billigere, mindre, og kraftigere data- og sensorteknologier over de siste tiårene, kan ASVer bli utviklet til en lavere pris og med mer avanserte funksjoner.

Denne avhandlingen utvikler et mekatronikksystem for en autonom seilbåt. Et distribuert programvaresystem er designet og utviklet for å støtte testing, autonomi og operatørkontroll. Systemet er separert i tre hoveddeler; en webapplikasjon for klienter, en server i skyen, og et seilbåtsystem. Systemet ble laget med toppmoderne teknologier og ettersteber interoperabilitet og pålitelighet.

Det mekaniske systemet består av et katamaranskrog, et vingeseil, og et ror. Et elektronisk system av sensorer, aktuatorer, og datamaskiner ble satt sammen for at systemet skal kunne oppfatte og handle ut i fra omgivelsene. Avhandlingen dokumenterer utviklingsprosessen fra valg av deler, monteringsprosesser, og integrering av programvare. Noen av sensorene og delene ble designet og 3D-printet for å oppnå målet om en fungerende prototype innenfor prosjektets tidsramme.

En sjøtest ble gjennomført som en integrasjonstest av et fullt system. Værforholdene under testen var harde, med temperaturer under  $5^{\circ}C$ , og spredte hagl og regnbyger. Vindhastighetene varierte mellom  $0.5 - 11.4m/s$  og med et gjennomsnitt på  $4.3m/s$ . Seilbåten opererte autonomt under sin egen batterikilde og med operatører på land. Testen viste lovende resultater, både det mekaniske og elektroniske systemet utførte oppgavene godt. Skyserveren presterte veldig bra, men det var noen problemer med båten sin programvare som forhindret den fra å seile ved flere tilfeller. Videre utvikling og testing er nødvendig for å oppnå en helautonom seilbåt.

---

## Acknowledgements

We would like to thank supervisor Professor Andrei Lobov for introducing us to the project and for the advice and support provided throughout the work. Thanks also to co-supervisor professor Dr. Andreas T. Echtermeyer for help and guidance with the development, as well as for providing a workspace and arranging the important field test. A special thanks to fellow student Håkon Bakke for giving us invaluable help and suggestions in the lab and to Simon G. Gjerde for bringing the skills needed to manufacture a new part for the mast.

---

# Table of Contents

<b>List of Figures</b>	<b>viii</b>
<b>Abbreviations</b>	<b>xi</b>
<b>1 Introduction and overview</b>	<b>1</b>
1.1 Autonomous Surface Vehicles (ASVs)	1
1.2 Problem description	1
1.3 Team and previous work	1
1.4 Scope of this thesis	1
1.5 Objectives	2
1.6 Thesis structure	2
<b>2 Theory</b>	<b>3</b>
2.1 Digital Twin, a virtual projection of a real-world object	3
2.1.1 Data related technologies	3
2.1.2 High-fidelity modeling and simulation technologies	4
2.1.3 Human-machine interface	4
2.2 Distributed Systems	4
2.2.1 Cluster and grid computing	4
2.2.2 Cloud Computing	5
2.2.3 Pervasive Systems	6
2.2.4 Architectural tactics	6
2.2.5 Architectures for distributed systems	8
2.3 System engineering and Mechatronics	10
2.4 Data processing with noise filters	12
2.4.1 Kalman Filter	12
2.4.2 Exponentially Weighted Moving Average	12
2.5 Sailing theory	13
2.5.1 Wind zones	14
2.5.2 Sailing towards the eye of the wind	15
2.5.3 Downwind sailing	16



---

2.5.4	Optimal angles . . . . .	16
2.6	Algorithms . . . . .	17
2.6.1	Heading vs Course control . . . . .	18
2.6.2	Component based control system . . . . .	19
2.6.3	Route planning . . . . .	19
2.6.4	Haversine - Distance between two point on a sphere . . . . .	20
<b>3</b>	<b>Approach</b>	<b>21</b>
3.1	Organizing and building a sailboat . . . . .	21
3.1.1	Workflow . . . . .	22
3.2	On-land services . . . . .	22
3.2.1	Intermediary on-land server . . . . .	22
3.2.2	HMI for monitoring and control . . . . .	23
3.3	Sensors and communication technology . . . . .	24
3.3.1	Sensors . . . . .	25
3.3.2	Communication technology . . . . .	25
3.4	Actuators and power supply . . . . .	26
3.4.1	Actuator for moving the rudder . . . . .	26
3.4.2	Motor for moving the wing sail . . . . .	27
3.4.3	Power supply . . . . .	28
3.5	Software development for autonomy . . . . .	28
3.5.1	Data processing, noise filtering . . . . .	30
3.5.2	Security through authentication . . . . .	30
3.6	Algorithms required for autonomous sailing . . . . .	30
3.7	System testing . . . . .	30
<b>4</b>	<b>Implementation</b>	<b>32</b>
4.1	Documentation and workflow . . . . .	32
4.2	On land server and cloud computing . . . . .	33
4.2.1	Server hosting service . . . . .	34
4.2.2	Containerizing and building . . . . .	34
4.2.3	Run time optimizations . . . . .	34

---

---

4.2.4	Logging and cloud storage . . . . .	34
4.3	Dashboard . . . . .	34
4.3.1	Path planning . . . . .	35
4.3.2	Notification center . . . . .	35
4.3.3	Instruments and the voltmeter . . . . .	35
4.3.4	Commando system . . . . .	36
4.3.5	Data viewer . . . . .	36
4.4	Sensor technologies and solutions . . . . .	36
4.4.1	Sensor to determine the angular position of the sail . . . . .	36
4.4.2	Sensors for measuring heading, attitude, and location . . . . .	39
4.4.3	Sensor to measure wind direction and speed . . . . .	41
4.5	Actuators and electronics . . . . .	42
4.5.1	Sail actuator . . . . .	42
4.5.2	Rudder actuator and mount . . . . .	44
4.5.3	Electronics box . . . . .	47
4.6	Computer and software implementation . . . . .	48
4.6.1	Microcomputer - Raspberry Pi . . . . .	48
4.6.2	Microcontrollers - Arduinos . . . . .	50
4.7	Sailing algorithm . . . . .	52
4.7.1	True wind calculations . . . . .	52
4.7.2	Bearing and the no-sail zone . . . . .	52
4.7.3	Haversine distance . . . . .	53
4.7.4	Optimal sail angle . . . . .	53
4.7.5	Midway point and beating for upwind sailing . . . . .	54
4.7.6	Adjustable parameters . . . . .	55
4.8	Security through authentication . . . . .	55
4.9	Integrating whole system . . . . .	55
<b>5</b>	<b>Tests and results</b>	<b>58</b>
5.1	On sea trial . . . . .	58
5.1.1	Experiment set up . . . . .	59
5.2	Server and communication . . . . .	60

---

---

5.2.1	On land server . . . . .	60
5.2.2	Interoperability between Raspberry Pi and the Arduinos . . . . .	62
5.3	Acceptance test of the client-side application . . . . .	62
5.3.1	Authentication requirements . . . . .	62
5.3.2	Map requirements . . . . .	63
5.3.3	Data visualization requirements . . . . .	66
5.3.4	Command system requirements . . . . .	67
5.3.5	Data and parameter requirements . . . . .	67
5.3.6	Notification system requirements . . . . .	68
5.4	System unit tests . . . . .	69
5.4.1	System test of the sail . . . . .	70
5.4.2	System test of the rudder . . . . .	72
5.4.3	Noise filter test . . . . .	74
5.5	Algorithm performance test . . . . .	76
5.5.1	Beam reach . . . . .	76
5.5.2	Downwind sailing . . . . .	78
5.5.3	Beating sailing mode and tacking maneuvers . . . . .	81
5.6	Integrated system review . . . . .	83
<b>6</b>	<b>Conclusion</b>	<b>84</b>
	<b>Bibliography</b>	<b>85</b>
	<b>Appendix</b>	<b>89</b>
A	Source code . . . . .	89
B	Command system . . . . .	89
C	Readmes from GitHub repositories . . . . .	92
C.1	RaspberryPi readme . . . . .	92
C.2	Dashboard app readme . . . . .	104
C.3	WebSocket server readme . . . . .	107
C.4	Arduino readme . . . . .	110

---

## List of Figures

1	Cluster computer diagram . . . . .	5
2	Grid computer diagram . . . . .	5
3	Cloud computer diagram . . . . .	6
4	Top level of the service ontology, from [21] . . . . .	8
5	Interoperability by traditional middleware, from [19] . . . . .	8
6	Fully distributed unmanned aerial vehicle (UAV) system, from [22] . . . . .	9
7	System architecture overview, from [25] . . . . .	10
8	Components of a mechatronic system, from [27] . . . . .	11
9	Procedure for a mechatronic system, from [28] . . . . .	12
10	Sailboat anatomy from[33] . . . . .	13
11	Wind zones, from article[34] . . . . .	15
12	Beating against the wind[35] . . . . .	16
13	NACA0009 airfoil[44] . . . . .	17
14	NACA0018 airfoil[43] . . . . .	17
15	NACA0024 airfoil . . . . .	17
16	Heading control during beam reach, from [35] . . . . .	18
17	Course control vs Heading control - simulation from [46] . . . . .	19
18	Proposed general hardware architecture . . . . .	25
19	Visual representation of rudder arm calculation . . . . .	26
20	Proposed general software architecture . . . . .	29
21	A section of the Trello board . . . . .	32
22	GitHub Organization[64], Note: The organization is private, but can be accessed by request to authors . . . . .	33
23	Development of rotary encoder . . . . .	38
24	Rotary encoder for reading sail angle . . . . .	39
25	Details of inertial measurement unit (IMU) . . . . .	40
26	Wind sensor mast adapter . . . . .	42
27	Pictures from CubeMars' website [67] . . . . .	43
28	Manufacturing and testing of new mast stub . . . . .	44
29	Development of lever arm and actuator mount . . . . .	45

---

30	H-bridge to control rudder actuator . . . . .	46
31	Circuit diagrams of voltage divider and ammeter . . . . .	47
32	Details of electronics box . . . . .	48
33	Sequence diagram of sailboat system . . . . .	50
34	Sequence diagram of Arduino Nano . . . . .	51
35	$\frac{\text{Coefficient of lift}}{\text{Coefficient of drag}}$ vs $\alpha$ , from article [38] . . . . .	53
36	Tacking sequences, black lines indicate edges of the no-sail zone. Red marker is the midway point, and the black marker are the target. $\beta = 30^\circ$ , $\beta_+ = 10^\circ$ . . . . .	54
37	Hardware architecture diagram . . . . .	56
38	Software architecture diagram . . . . .	57
39	Wind speeds recorded by the sailboat during the entire test . . . . .	58
40	Sailboat on the trailer . . . . .	59
41	In-progress assemble of the sailboat . . . . .	59
42	Container instances . . . . .	60
43	Request count . . . . .	60
44	Request latencies . . . . .	61
45	CPU utilization . . . . .	61
46	Memory utilization . . . . .	61
47	Billable container time . . . . .	61
48	Authentication form . . . . .	62
49	Map with a path . . . . .	63
50	Homepage with the map . . . . .	64
51	From left to right: (1) New path, (2) download current path, (3) upload a path file, (4) edit boat's current path. . . . .	64
52	Path planner with a list of waypoints . . . . .	65
53	Path coordinates in order . . . . .	66
54	Instrument page . . . . .	66
55	Command page . . . . .	67
56	Command description . . . . .	67
57	Pages visualising data and parameters . . . . .	68
58	Notification message . . . . .	68
59	Notification center . . . . .	69

---

---

60	Set up of the sail control system . . . . .	70
61	Target sail angle, and actual sail angle (adjusted 180° to improve readability) . . .	71
62	Sail angle and apparent wind direction (adjusted 180° to improve readability) . . .	72
63	Set up of the rudder and the actuator . . . . .	73
64	Rudder and target angle . . . . .	74
65	Wind speed . . . . .	75
66	Apparent wind direction and heading (adjusted 180° to improve readability) . . . .	75
67	True wind direction (adjusted 180° to improve readability) . . . . .	76
68	Beam reach illustration . . . . .	76
69	Beam reach map . . . . .	77
70	Beam reach angles . . . . .	77
71	Beam reach speeds . . . . .	78
72	Downwind illustration . . . . .	78
73	Downwind . . . . .	79
74	Down wind sailing mode angles . . . . .	79
75	Down wind sailing test speeds . . . . .	80
76	Path traveled in down wind sailing mode . . . . .	81
77	Sailboat and a mid waypoint . . . . .	81
78	Sailboat struggling to tack (adjusted 180° to improve readability) . . . . .	82
79	Beating calculation error . . . . .	83

---

## Abbreviations

- AoA** angle of attack. 16, 19, 43, 52, 53, 55, 71, 76, 80
- ASV** autonomous surface vehicle. i, ii, 1–4, 16, 26
- CAD** computer aided design. 37–40, 45
- CAN** controller area network. 43
- CoG** Course over Ground. 18
- CSV** comma-separated values. 34
- DDS** data-distribution service. 8
- EWMA** exponentially weighted moving average. 12, 41, 74
- GCP** Google Cloud Platform. 33, 34
- GPS** global positioning system. 25, 41, 47, 76
- HMI** human machine interface. 2, 4, 10, 22, 23, 84
- I2C** inter-integrated circuit. 39
- IaaS** infrastructure as a service. 5
- IMU** inertial measurement unit. viii, 25, 39, 40, 51, 52, 62, 74, 75, 79
- IoT** internet of things. 6, 7
- IR** infrared. 37
- JSON** JavaScript Object Notation. 7, 35, 36
- JWT** JSON web token. 55
- MCU** main computing unit. 22, 24, 25, 30, 49, 62
- OMG** object management group. 8
- OWL-S** semantic markup for web services. 7
- PaaS** platform as a service. 5
- QoS** quality of service. 8
- SaaS** software as a service. 5
- SoG** Speed over Ground. 76
- UAV** unmanned aerial vehicle. viii, 8, 9
- URL** uniform resource locator. 7

---

**USB** universal serial bus. 39, 41, 48, 49

**VO** virtual organization. 5

**W3C** world wide web consortium. 7

**WSA** web services architecture. 7

**XML** extensible markup language. 7



---

# 1 Introduction and overview

## 1.1 Autonomous Surface Vehicles (ASVs)

With the focus on a sustainable ocean and the ongoing rapid changes in weather and climate, a considerable amount of data is required to understand the complicated interactions between the ocean, atmosphere, and the planet's inhabitants. Most common methods for data collection are done by large research vessels over a limited time or by stationary equipment like buoys. Such methods can be pretty resource-demanding or deliver unsatisfactory results.

New, improved, and promising technologies have the potential to evolve the future of ASVs. An uncrewed autonomous vehicle could be smaller, lighter, and more durable than anything made for humans. As a sailboat harnesses the wind's power, great distances can be covered with low energy consumption. Furthermore, long missions in threatening and unreachable environments could be possible while remaining effective and affordable. There are few commercially available options, but many studies on the topic have moved the area and interest forward. With the combined technological advances, more research on this topic should be conducted.

## 1.2 Problem description

This thesis continues the work on an ASV able to carry scientific equipment to perform research at sea over a long period. Additionally, the ASV aims to be affordable and easy to manufacture, although robust and reliable. During the voyages, the boat should be self-sustained and operate fully autonomous. A fundamental part of this goal is creating a prototype for further development and testing.

## 1.3 Team and previous work

In 2019, Dr. Andreas T. Echtermeyer instigated the task on an autonomous sailboat. Master theses on this project have previously been written by Maria Dyrseth, Sverre Gauden, and Almar V. Brendal, and their results are used in this thesis.

Professor Andrei Lobov assembled this team to work on a pre-study thesis[1] on electronics and communication in autumn 2021. This master thesis builds upon the work from this pre-study. Lobov is also the team's supervisor, with Echtermeyer as co-supervisor on the master thesis.

## 1.4 Scope of this thesis

This thesis documents an approach to developing and validating an autonomous sailboat. This report consists of a theoretical study of best practices and fundamental theories, a proposed approach, an implementation, and tests with discussed results. This thesis intends to document and guide others in following Dr. Echtermeyer's work or similar projects in the future.

---

## 1.5 Objectives

The main goal of this thesis is to create an approach to the development and validation of an autonomous sailboat. The sailboat should be able to perceive and act upon its environment. Further, a user-friendly human machine interface (HMI) should be available for monitoring and controlling the boat. Lastly, a server to act as a mediator between the boat and operator should be accessible with a focus on security and extendability. A goal is to validate the system by conducting a sea trial with a seaworthy autonomous sailboat and supporting on-land systems.

## 1.6 Thesis structure

Chapter 1 introduces the master thesis and previous work leading up to the project's current state.

Chapter 2 is a study of literature and provides the necessary technical background on the subjects.

Chapter 3 is a suggested approach for developing an ASV.

Chapter 4 presents a solution to the approach in chapter 3.

Chapter 5 introduces the tests and discusses the results.

Chapter 6 concludes the master thesis and achievements made throughout the project and outline the way forward.

---

## 2 Theory

This section is a literature study to identify best practices and technologies in pursuing the research objective. It contains a summary and explanation of some current state-of-the-art technologies, theories, and architectures based on numerous credible sources. First is the study of digitization of general real-world objects. It focuses on the challenges and benefits of digitization and how they can propose safety, control, and monitoring of an ASV. Following is a segment on different types of distributed systems and relevant architectures. The following subsection explains system engineering, mechatronic systems, and common noise filters for processing sensor data in such systems. These topics can prove valuable when building a system for an ASV and creating services for autonomy and communication. Lastly, two sections introduce the sailing theory and mention some algorithms for keeping course and creating a sailing path.

### 2.1 Digital Twin, a virtual projection of a real-world object

A Digital Twin is known as a virtual representation of a physical object. In recent years, it has been moving further with additional features, namely bidirectional transfer or sharing of data. These data include quantitative, qualitative, historical, environmental, and real-time data.

Since the early 2000s, interest in a digital representation of actual assets has increased. With the progress in ICTs in recent years, further advancements in developing and utilizing Digital Twins have been made. Technologies such as the Internet of things (IoT), artificial intelligence, cloud computing, high-performing cellular networks, and wireless sensor networks promote practical applications of a Digital Twin in multiple fields [2].

In order to leverage the advantages of a Digital Twin, a business should understand the characteristics and types of Digital Twins [2]. Fusing many different technologies and theories required to implement a digital twin can be pretty complex.

#### 2.1.1 Data related technologies

The basis of the digital twin is data. Numerous sensors, readers, and scanners to measure the environment and internal conditions are collected, processed, and transmitted in real-time (or close to real-time). Since the data can be vast, quickly changing, and of various types, some form of edge computing is often facilitated to reduce the network load. Microsoft Azure, Amazon Web Services, and Google Cloud Platform provide one of the most well-known cloud services. Cloud services can store, analyze, map, and process large-scale data reasonably cheaply.

Some communication technologies are more suitable for transferring data between the physical and digital twin. With a developed platform, additional sensors could be easily added to the physical twin, while a provided interface could make the sensor easily configurable remotely.

A digital twin can provide improved accessibility by allowing control over the physical twin through the virtual twin [3]. Such an improvement could be beneficial when the geographical location restricts control of the physical twin.

---

### 2.1.2 High-fidelity modeling and simulation technologies

Model representation of the physical twin contains semantic data and computer models typically used in combination with simulation software for testing numerous properties of the physical twin. Simulations with digital twins can predict structure fatigue, extending the physical twin's life[2]. However, a problem with simulation is the lack of real-time data, which makes the model or simulation static [4].

Tesla is a clean energy car designer and manufacturer known for using digital twins in simulations. Stress tests in scenarios have shown beneficial in further development of the car's safety by having a high-fidelity model of the car and its parts. Further, they have digital twin models of every car they have sold and provide software updates to the consumer based on collected data[5].

### 2.1.3 Human-machine interface

The goal of a user interface that a HMI offers is to allow humans more manageable and accessible control over a machine. Furthermore, an HMI has proved to be helpful when in need of adjusting an autonomous vehicle[6]. HMI allows an operator to, e.g., monitor and control processes and change objectives. Since the ASV is remote, a solution for the HMI should adapt to that situation by promoting accessibility. One solution is websites created as a single page application [7], mobile applications [8], or other software services that usually complement some other specific hardware [9]. Additionally, a HMI could create safety barriers to a system as it could reduce human errors[10]. In an article at Frontier's in AI[11] there were a HMI implemented for operators to aid an autonomous sailboat with missions and parameters. Furthermore, the HMI offered visualization of data sent by the on-sea sailboat allowing for further analysis of ocean conditions.

## 2.2 Distributed Systems

A distributed system is a set of independently operating computing nodes that collaborate so that it appears as a single coherent system for the user[12]. The nodes can consist of hardware and software components ranging from small embedded computers to powerful computers. Individual nodes do not provide much value in themselves, but when collaborating with other nodes in a distributed system, they can work towards a common goal and deliver high value. In order to achieve this, the nodes must be networked and managed so that they can communicate quickly, safely, and reliably with each other. One use case of a distributed system is for high-performance computing.

### 2.2.1 Cluster and grid computing

Cluster computing systems consist of smaller equal computers connected through a high-speed network to act as one powerful computer. Such systems became popular when the price and performance of regular personal computers improved enough that it made sense to connect many together. Cluster computers are often used for programs that run in parallel utilizing all the computing nodes. Usually, one computer in a cluster is assigned the master role. It handles the allocation of tasks to the other computers and provides an interface for the cluster user. Otherwise, the hardware, software, security, and network between the nodes are as identical as possible [12].

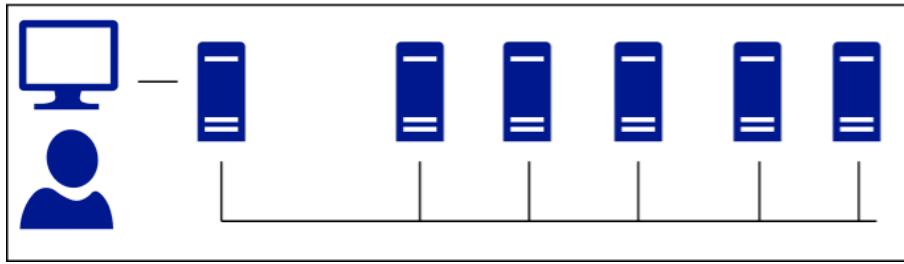


Figure 1: Cluster computer diagram

Grid computing is a system where the nodes focus on solving more specific tasks by providing resources such as storage, servers, computation, and applications. These resources can be geographically dispersed and shared between many users with different use cases. One grid cluster can support many Virtual organizations (VOs) which provides a private, customized execution environment. Users in the same VO can collaborate and access the same resources, which appear as if the organization owned the resources themselves [13].

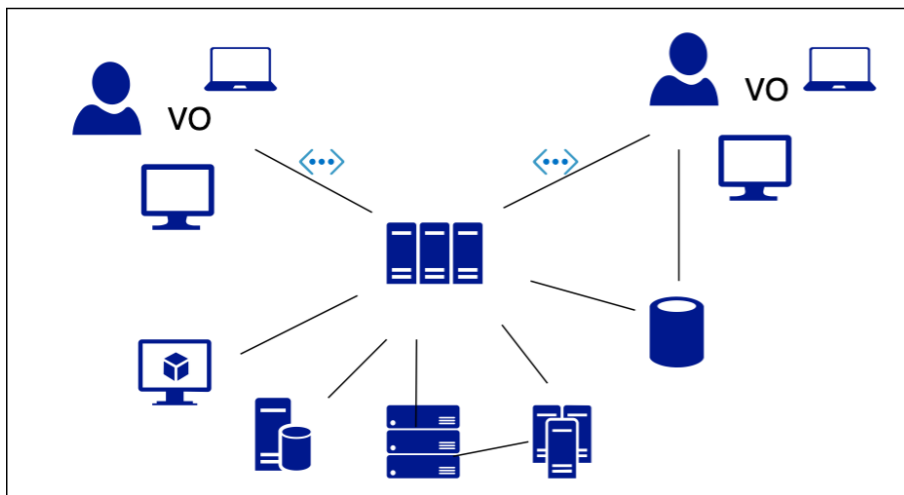


Figure 2: Grid computer diagram

### 2.2.2 Cloud Computing

While the two previous classes of distributed systems focus on how to structure compute nodes to achieve a specific goal, cloud computing is about making such systems available to customers by providing a dynamic infrastructure [12]. It can be defined as a pool of easily accessible virtualized resources which can dynamically and optimally scale to the current workload [14]. Usually, clouds can be divided into three different levels of abstraction, the first being infrastructure as a service (IaaS), the most barebone service, only providing the virtualized infrastructure for the customer to build its system on. The next level is platform as a service (PaaS) which offers a platform for systems to run on, making it easier for customers to deploy their software layer on a cloud service without spending time on the fundamental parts of the system while still benefiting from the dynamic resource allocation. The last level is known as software as a service (SaaS) [15]. Instead of running applications locally on a computer, it runs in the cloud and is accessed from the web. The applications can then be accessed everywhere on less powerful hardware and by multiple users simultaneously.



Figure 3: Cloud computer diagram

### 2.2.3 Pervasive Systems

Unlike systems with powerful computer nodes and defined interfaces for interaction, pervasive systems describe distributed systems that can be small, mobile, and wireless. They are meant to blend into everyday environments and use sensors and actuators to allow for interaction with users and the surroundings. Pervasive systems can be separated into three categories, ubiquitous computing systems, mobile systems, and sensor networks [12].

As the name suggests, ubiquitous computing systems are built into the surroundings and consist of sensors, actuators, and computing units spread across. They can communicate, interact with users, be context-aware, and operate autonomously, all while appearing transparent to the user[16]. Sometimes, the user does not realize that interaction with a ubiquitous system is happening. One example is traffic lights and pedestrian crossings where the lights communicate, and sensors tell if cars or people are present [17]. People interact with traffic lights daily without recognizing they have given it input and that the output is a result of it.

Mobile computing systems are systems where the location is expected to be changing over time [18]. It can be anything from mobile phones and internet of things (IoT) devices to connected vehicles. These systems require different techniques to enable dynamic discovery and direct communication. The last category of pervasive systems is networks of sensors. These systems can consist of thousands of connected sensors in a network. Middleware is used to utilize the data gathered, for example, by deploying applications.

### 2.2.4 Architectural tactics

#### Transparency

A distributed should feel and behave like a single coherent system for the end-user, making the physical properties of the system transparent [18]. The different types of distribution transparency can be separated into seven [12]. First is access transparency removing the details of how the data is represented and stored. Differences in naming, operations, and involved processes to present a

---

resource should not be visible to the end-user. The second transparency is location transparency. Users should not know where the resource is physically located. A typical example of this transparency is the use of uniform resource locator (URL) to provide a dynamic path to a resource. It also has the benefit of allowing resources to relocate without the user noticing, introducing relocation transparency. Cloud computing services are an example of servers located in different geographic locations, where resources can be shared and moved between them without the user noticing. Users can also initiate relocation of resources if it supports migration transparency—for example, mobile phones or mobile IoT systems.

Replication transparency is a special case of location transparency meant to improve user experience. Several copies of a resource are distributed in several locations providing services closer to the end-user with better capacity and redundancy. Except for better performance, the user should not be able to tell that several copies exist. Even if multiple users access the same resource, it should be handled so that the individual user does not notice it, known as concurrency transparency. As mentioned, redundancies are essential in a distributed system with replication transparency. If a failure occurs, it should be caught and automatically recovered without the end-user noticing any disruptions.

### **Interoperability**

The emergence of pervasive distributed systems, including ubiquitous- and mobile computing systems and sensor networks, are increasing the challenge and difficulty of connecting distributed systems together[19]. Interoperability is one of the more significant challenges when engineering distributed systems. The two main parts of this challenge are extreme heterogeneity and dynamic and spontaneous communication. The former describes the different systems that can be expected to communicate together. It ranges from supercomputers to mobile phones to small sensors. The latter highlights that connections are first made at runtime, making design or deployment decisions to enable interoperability harder.

Data heterogeneity is a result of the infinite number of different approaches to representing data. The plethora of methods creates a barrier for two systems to connect if they are not specifically designed to represent data similarly. There are solutions trying to standardize data representation like extensible markup language (XML) and JavaScript Object Notation (JSON). However, even when two systems use the same protocols, they might not share the same structure or naming of properties. Such restrictions are part of the semantic heterogeneity problem where data must have the same semantic representation for every participant. The web services architecture (WSA) is an architecture by world wide web consortium (W3C) that identifies global elements required to ensure interoperability between Web Services[20]. Efforts have been made to extend the Web Services description language to include semantics for data exchange. One known attempt is semantic markup for web services (OWL-S), an ontology of services providing functionality to allow software agents and users to discover, invoke, and compose automatically. Figure 4 shows three essential properties of a service provided by the ontology. ServiceProfile is the profile of the service and explains what the service provides for clients. ServiceGrounding defines how to interact with the service and details about transport protocols it supports. ServiceModel includes the process model describing how the service is used[21].

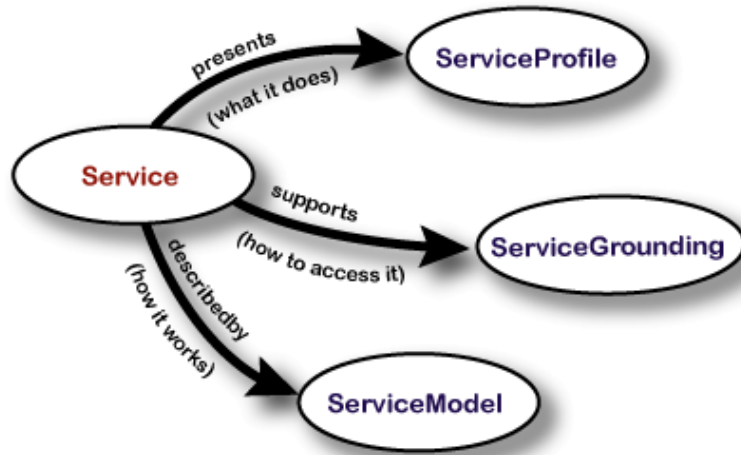


Figure 4: Top level of the service ontology, from [21]

Another way to solve interoperability is to use a middleware. Systems implementing the same middleware are ensured to be interoperable with each other. However, this solution introduces some new issues. Only systems implemented with the same middleware would be interoperable, creating new barriers against broad interoperability. Secondly, a universal middleware that supports all systems is very unlikely, given the vast array of distributed systems. The development of new distributed systems is also happening very fast, outpacing the creation of standards in the form of middleware. Over time old middleware will become obsolete and make systems using it outdated.

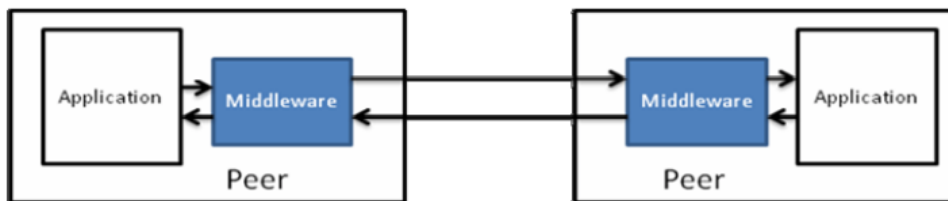


Figure 5: Interoperability by traditional middleware, from [19]

### 2.2.5 Architectures for distributed systems

An example of a distributed system architecture used for an UAV is the Manned-Unmanned Teaming research project conducted by the Institute of Flight Systems at Universität der Bundeswehr in München, Germany[22]. The project consisted of one or more UAV and a ground control station implemented as a fully distributed system, illustrated in Figure 6. They based the architecture on the data-distribution service (DDS) standard created by the object management group (OMG). It is a specification for publish-subscribe data distribution in real-time systems [23]. In a publish-subscribe system, subscribers can listen for data from one or more publishers. The communication is separated into different topics and provides a connection point between the participants. New publishers and subscribers can be added to existing topics without creating a new one. This solution enables a scalable system where different system parts can be developed, extended, and tested as individual modules. DDS is data-centric, which enables all the quality of service (QoS) parameters to be specified for each topic, giving developers fine control over the system design and performance [24].



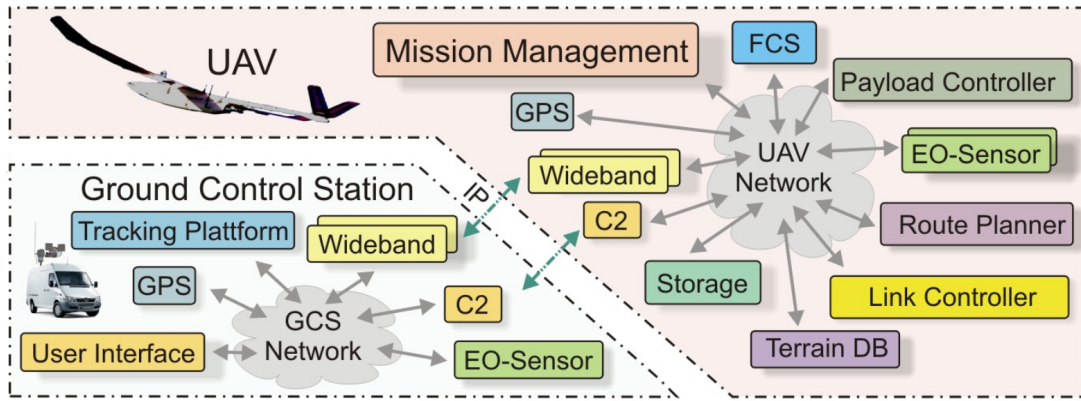


Figure 6: Fully distributed UAV system, from [22]

Another UAV project from China proposed a three-layer system architecture [25]. Figure 7 presents an overview of the architecture with the following layers: Executive layer, Cloud layer, and Human Operator layer. The executive layer contains the onboard flight controller, a communication link with the ground station, and a simulator. Next is the cloud layer, divided into a shadow layer and a core layer to increase modularity. The shadow layer act as a bridge between the UAV and the cloud and have a communication module and an abstraction module. The communication module enables support for different communication protocols, one of which is Websocket. It handles authentication with the cloud and as a message broker once the connection is established. The abstraction module has a remote control interface with all possible control actions to help client developers with limited knowledge of the hardware. A mission control component helps offload some of the autonomy processing from the UAV to the cloud, and a sensor manager helps standardize sensor data representation. The main component is the shadow files keeping the latest data from the UAV to create a digital twin in the cloud. This benefit is that other systems relying on data from the UAV can get data from the digital twin instead of requesting the UAV. This offloading can prevent overwhelming the onboard computer, synchronization issues, and data loss due to unstable wireless connection with the UAV.

The cloud core layer is responsible for the long-term storage of all data collected. It also hosts the central server, which all clients connect to in a scalable virtual environment, allowing more computer resources to be allocated as needed. Many autonomy calculations are also done in the cloud, offloading the onboard computer and allowing for more advanced algorithms. The last layer is the Human Operator layer containing a cloud service hosting a web application that communicates over Websocket. The web application is used to plan missions, control and manage the UAV, and other managing.

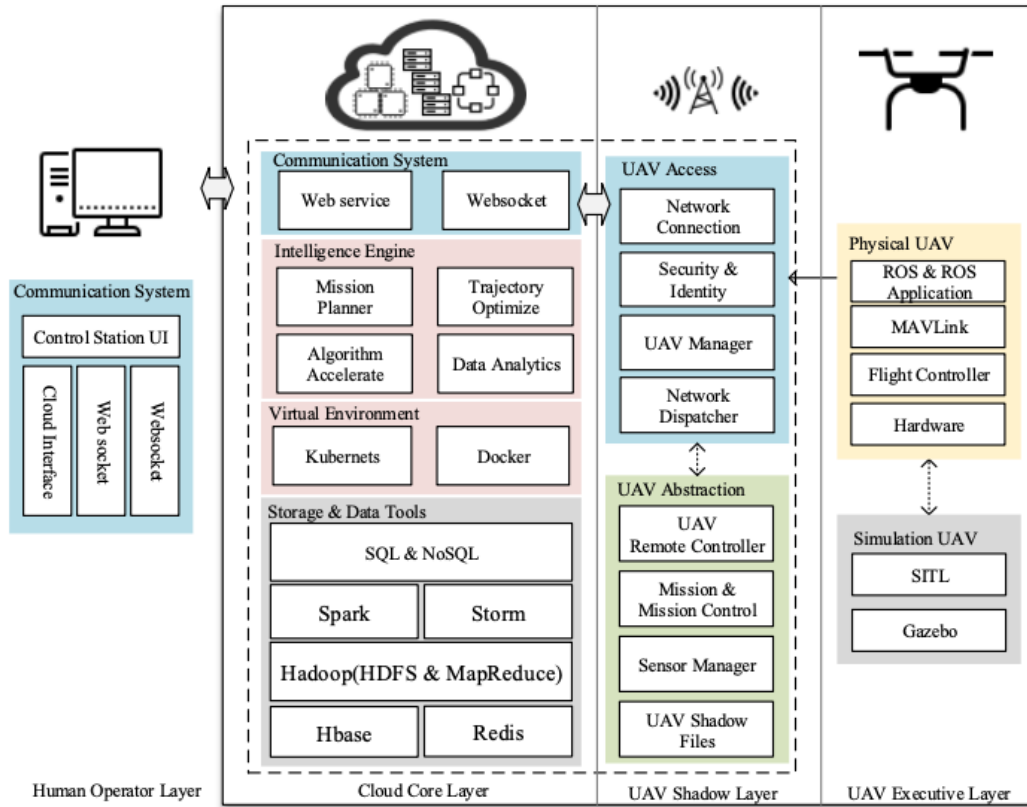


Figure 7: System architecture overview, from [25]

### 2.3 System engineering and Mechatronics

*Mechatronics* is a term that appeared in the early 70s with the development of computers and electronics. The terminology describes the combination of mechanical and electronic engineering[26]. Mechatronic systems often depend on the real-time computation of data gathered from their surroundings, which the system uses to alter the machine's state. Figure 8 shows an example of different components making up a mechatronic system. In the center are the computational unit containing computers, software, and interface to other systems. Connected to the computational unit is instrumentation providing information about the surroundings. On the other side is actuation, which enables the system to interact with the real world, known as the target system. Operators and parent systems can interact with the computational unit through an HMI or other integrated systems.

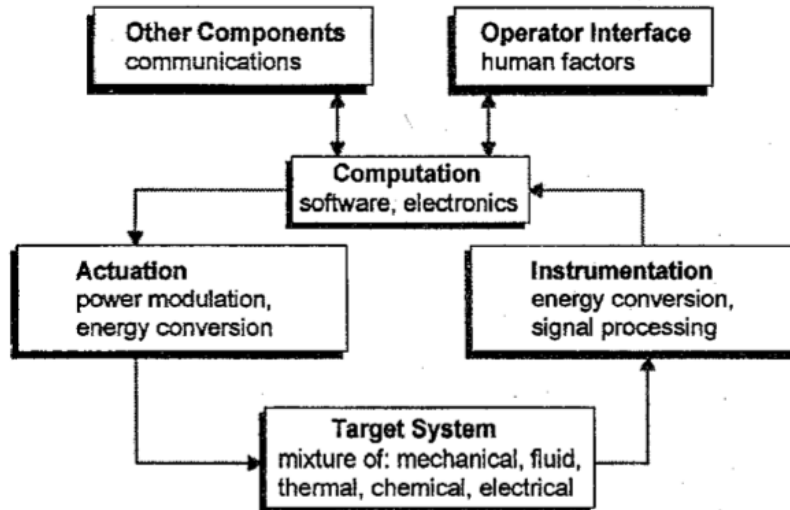


Figure 8: Components of a mechatronic system, from [27]

The procedure for the computation, actuation, target system, instrumentation loop in Figure 8 has been attempted generalized resulting in the illustration in Figure 9 [28]. It starts with object input perceived by sensors and processed by a signal processor before its sent to a microprocessor. The microprocessor controls power to actuators and uses a control loop mechanism to achieve the desired movement. The actuators act on the mechanical system. Encoders and sensors in this system send feedback to the microprocessor used to validate if the actuators acquired the desired output.

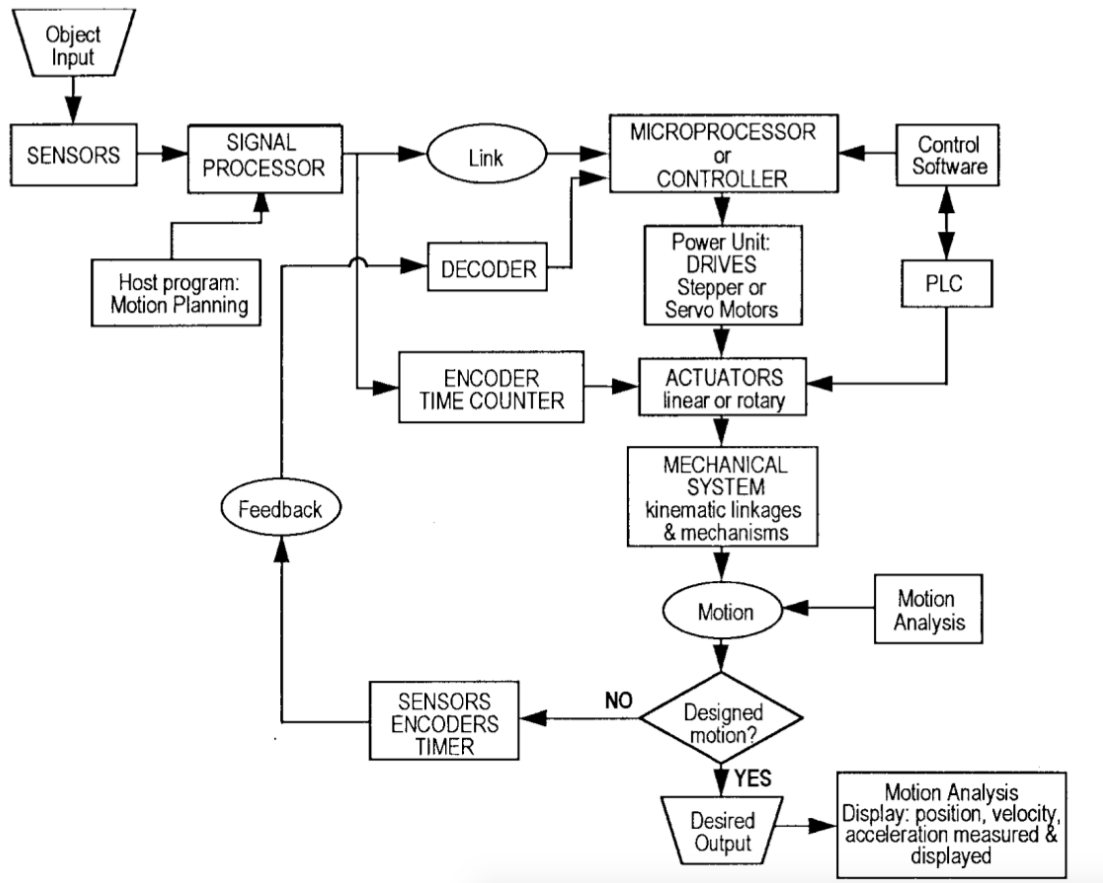


Figure 9: Procedure for a mechatronic system, from [28]

## 2.4 Data processing with noise filters

Introducing noise filters may improve the accuracy of single sensor measurements, particularly for highly-sensitive sensors. Inaccurate data can be detrimental for control systems dependent on reliable data[29].

### 2.4.1 Kalman Filter

According to a study in China[30], by introducing a filter known as *Kalman Filter*, the precision of low-cost sensors increased by 27%. Kalman filtering is an algorithm commonly used to reduce noise from sensor signals. By processing a series of inaccurate and uncertain measurements, the Kalman Filter produces estimates of hidden variables [31]. Furthermore, the filter provides predictions of future measurements based on past estimations. For smaller computers with low memory, a Kalman Filter is ideal. The memory usage is limited to the need of only storing the previous state of the sensor measurement. Thus, making the Kalman Filter suitable for embedded systems.

### 2.4.2 Exponentially Weighted Moving Average

An exponentially weighted moving average (EWMA) filter is used to smooth out measurement variations by weighting the average for the last period and the current measurement. Advantages

of this method are low computational cost, diminishing old data, and requiring minimum data. The stochastic process can be described as

$$\overline{S}_t = B[S_t + AS_{t-1} + A^2S_{t-2} + A^3S_{t-3} + \dots] \quad (1)$$

where constant  $B \in [0, 1]$  and  $A = 1 - B$  [32].  $\overline{S}_t$  is an estimate resulting from a weighted average between a new value  $S_t$  and all past values. The weighting is exponential, resulting in older values getting exponentially smaller for each new measurement. The influence of a new measurement  $S_t$  can be tuned by adjusting the weight  $B$ . If  $S_t$  is subject to much random variation, a lower  $B$  will yield a better estimation of the mean. However, if the mean is often changing, the weight  $B$  should be larger to diminish the effect of old values.

## 2.5 Sailing theory

A sailboat's main parts are the sail, rudder, keel, and hull. The purpose of the sail and rudder is to control the boat, where the sail's goal is to gather wind force to increase the velocity of the sailboat and the rudder to change its heading. Compared to motorized boats, a sailboat requires some speed to change its heading as it does not have a propeller providing force on the rudder.

At the bottom of the boat, a keel cancels out the lateral forces from the sail, and helps against capsizing. This is achieved by having foil under water with a heavy material in the bottom, lowering the center of mass of the sailboat. The hull is the boat's body, where the front is called the bow, and the back is often called the stern. Port and starboard are the left and right sides of the boat. An illustration of the sailboat's anatomy can be seen in Figure 10.

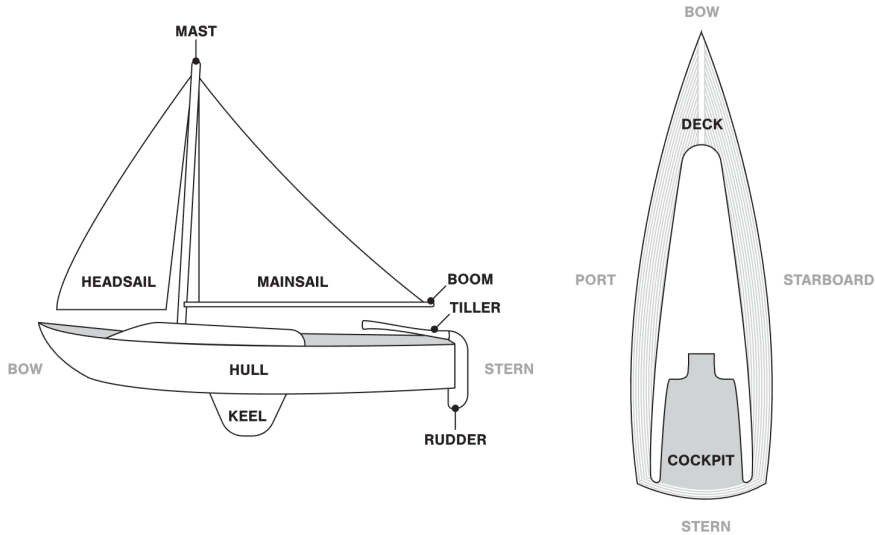


Figure 10: Sailboat anatomy from[33]

Some nautical terms used in this thesis can be found in Table 1.

---

Table 1: A table of nautical terms

**Nautical Terms**

---

**Apparent wind** - The wind measured from a moving instrument, e.g. a boat.  
**Beating** - To sail against the wind by doing a sequence of tacks.  
**Great circle arc** - The shortest distance between two points on the surface of a sphere.  
**Jibbing** - A technique where one crosses the wind direction with the stern against the wind.  
**Leeway** - The drift a boat can have which result in an angle from the boat's heading.  
**Tacking** - A maneuver where one crosses the wind direction with the bow against the wind.  
**True wind** - The wind measured from a stationary instrument.  
**Velocity made good** - A sailing term describing the speed relative to the wind direction.  
**Windward** - The side of the boat that is towards the wind.

---

*True wind* is the wind speed and direction that a stationary instrument could measure. *Apparent wind* is different as it considers wind speed, the wind instrument's velocity, and direction relative to a moving instrument.

### 2.5.1 Wind zones

During sailing, the boat can be in three situations: upwind, downwind, and beam reaching. In downwind, the sailboat can open the sail to gather wind and gain speed. When the wind hits the boat from the side at around  $90^\circ$ , it is called beam reaching, denoted as  $C$  in Figure 11. Upwind situations are when sailing against the wind. It is still possible to gain speed, but by sailing close-hauled, see  $B$  in Figure 11. The area of approximately  $30^\circ$  on each side of the wind direction is called the no-sail zone. A sailboat cannot sail forward in the no-sail zone since the angle of attack for the wind on the sail does not permit thrust. The smallest possible angle between the wind and the sailboat can be found through experiments, for example, in wind tunnels or field trials.

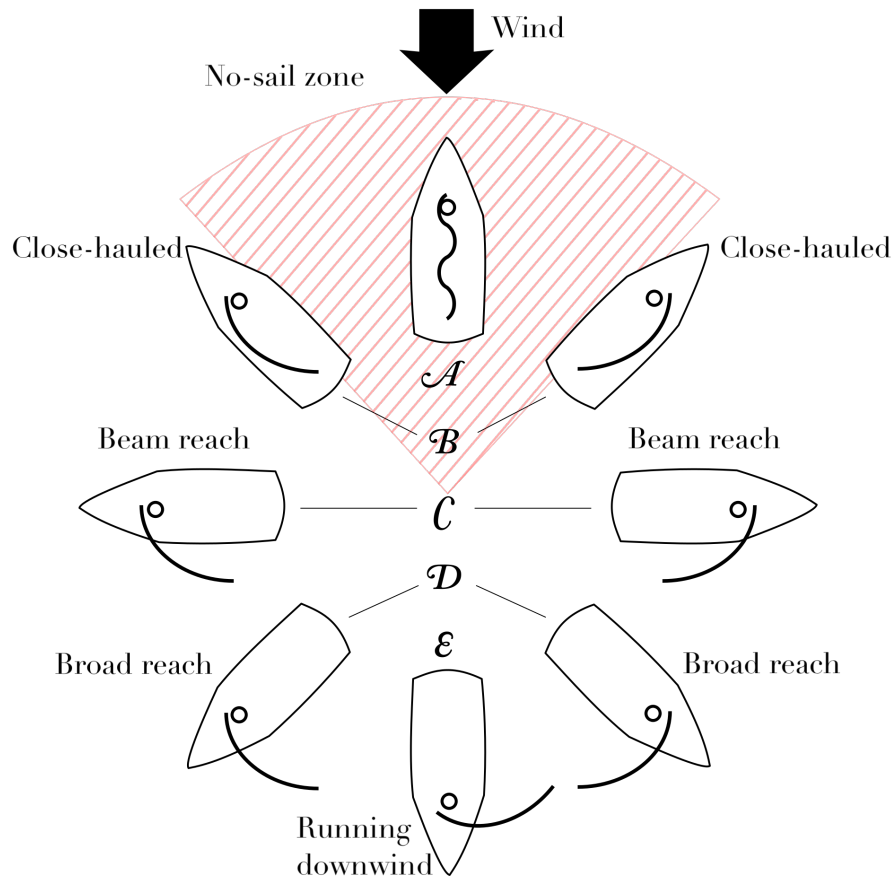


Figure 11: Wind zones, from article[34]

### 2.5.2 Sailing towards the eye of the wind

A typical strategy to reach a point that requires sailing straight against the wind is called beating. It is a technique where one is sailing in a zigzag maneuver. In order to accomplish beating, two typical sailing maneuvers are used, namely tacks or jibs. Tacks are done when one crosses into the wind, and the wind changes from one side of the boat to the other (for example, port to starboard or opposite). Between each tack the sailboat sails close-hauled outside the no-sail zone thus moving forward against the wind direction. Sufficient forward momentum is required to perform a tacking maneuver, in order to cross the no-sail zone. The frequency of tacking maneuvers depends on limitations such as obstacles. In theory, the fewer tacks, the faster one should reach the goal because of the speed loss during a tack. In Figure 12, there is an illustration of a series of tacks.

Jibs, however, do not cross the no-sail zone, but turns the opposite direction and crosses the wind direction downwind. This eliminates the need for sufficient forward speed, however, at one point the sailboat is sailing away from the target and therefore loses some progress.

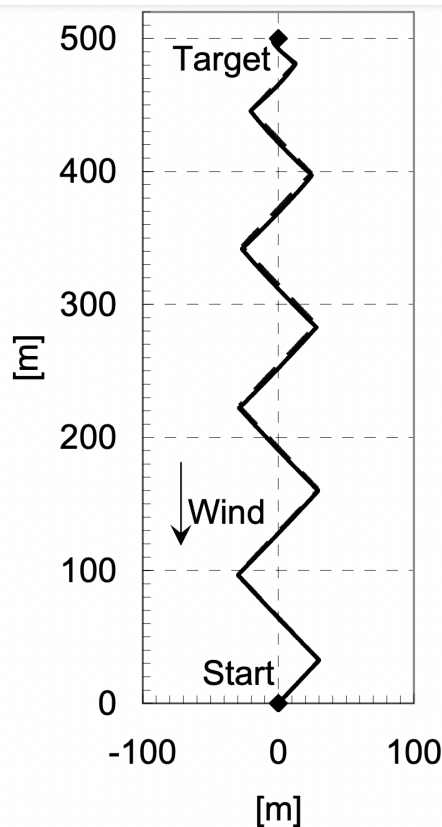


Figure 12: Beating against the wind[35]

### 2.5.3 Downwind sailing

According to a study on fixed-wing ASVs an efficient method for sailing downwind is to switch the sail configuration from a lift-generating to a drag-generating surface[36]. This is known as running downwind. It is achieved by increasing the angle of attack (AoA) to  $90^\circ$  when sailing downwind. The study calculated the optimal apparent wind angle to switch the configuration of their wing foil to be  $135^\circ$ .

Another solution is a similar technique as during upwind sailing, namely beating. According to an article by J. Otto Scherer[37], sailing straight downwind while relying on drag for thrust limits one to about half the speed of the wind. This limitation is due to the boat's resistance in the water and the apparent wind decrease at higher speed. Theoretically, in most conditions, sailing broad reach can result in higher speeds than compared to running because of the increased apparent wind.

### 2.5.4 Optimal angles

National Advisory Committee for Aeronautics develops standards for aircraft wings that have had growing popularity as wing sails for sailboats [38][39][40]. Even for more prominent companies, a wing sail has become the state of the art solution for autonomous sailboats[41][42].

As these wings mostly follow a standard, optimal angle of attack calculations applies for every wing of the same standard. From KTH Royal Institute of Technology, a paper[39] was published about a sailboat with a fixed-wing using the standard NACA0018[43], see Figure 14. After VLM



---

simulation, used for computing fluid dynamics, they concluded that a maximum angle of attack was  $12^\circ$ . An angle of attack higher than that would stall the wing.

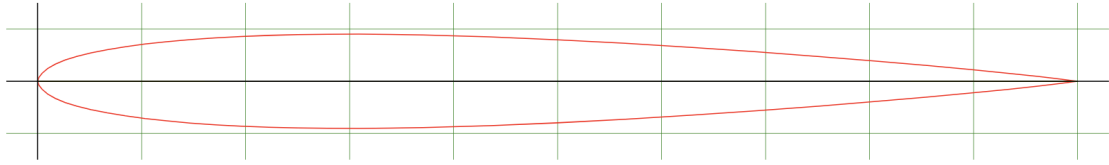


Figure 13: NACA0009 airfoil[44]

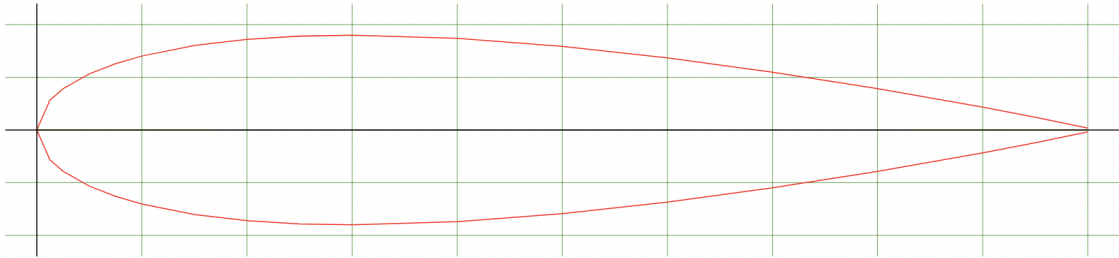


Figure 14: NACA0018 airfoil[43]

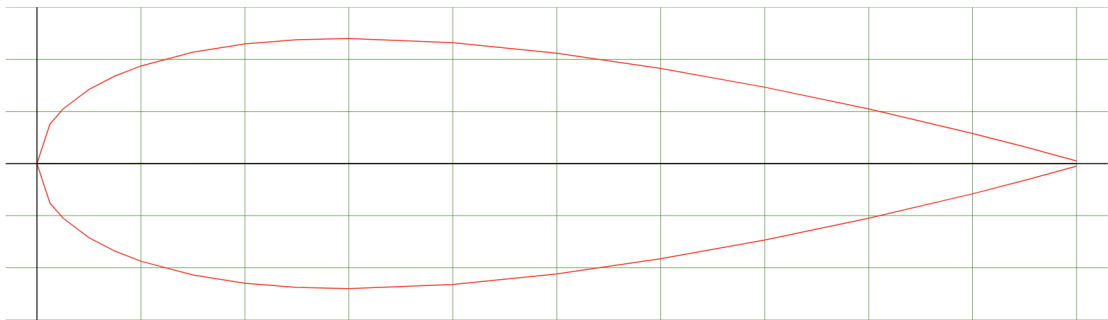


Figure 15: NACA0024 airfoil

Furthermore, an article published in the Journal of Fluid Science and Technology[45] had a similar result with a NACA0009[44] and a NACA0024 which is a thinner and thicker airfoil, respectively, compared to NACA0018. They performed a wind tunnel test on their foils, and the results were an optimal angle of attack of  $10^\circ$  before stalling.

From the article[38] by Dyrseth, a symmetrical wing sail allows for a greater span of angles compared to a non-symmetrical one. Further, a thin airfoil has limited interior space for a mast and electronics, while a thicker airfoil proved to stall earlier and be less stable[38]. After a CFD validation, the thesis concluded that a wing sail using the NACA0018 standard would have an optimal operating range between  $6 - 11^\circ$ [38]. Within this range, an optimal lift and drag ratio is attained.

## 2.6 Algorithms

In the following sections, a description of solutions found from numerous sources is mentioned that promote autonomy for sailboats. Furthermore, some challenges are highlighted for algorithms and

---

the task at hand.

### 2.6.1 Heading vs Course control

In order to traverse from point A to point B, the most common solution is a heading control[46][47][48]. The idea is to adjust the boat's heading to be the same as the bearing, which is the angle between north and a target from the boat's position. At short distances, it could be efficient enough, but during long voyages, several forces may be taken into consideration. The reason for this is the leeway effect, which diverges the course from the heading. Some of the reasons for drifting are the ocean current, but also wind forces[35][46], particularly during beam reach, as can be seen in Figure 16.

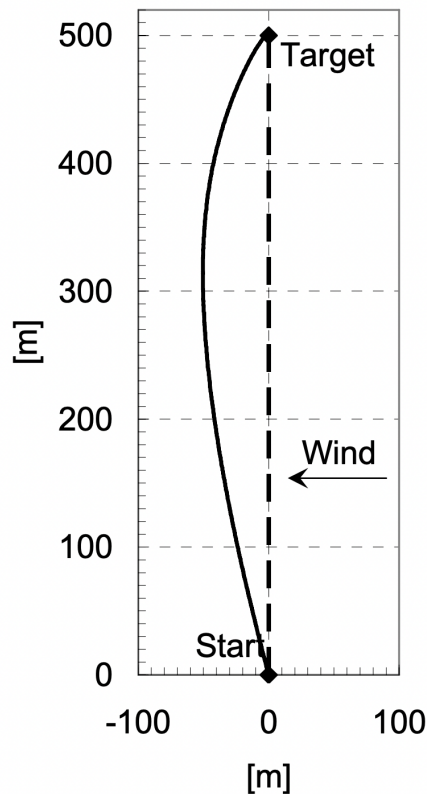


Figure 16: Heading control during beam reach, from [35]

A course controller could assist the boat from diverging from the line between the boat and the desired goal [46][35]. In order to consider the leeway effect, data about Course over Ground (CoG) or at least wind forces are necessary. This information allows for adjusting the heading of the boat to compensate for the drift resulting in a more straight traversal between the boat's starting position and the target position [46], see Figure 17. How the wind affects the boat depends on the boat's physical properties, and the amount of leeway may vary accordingly[35].

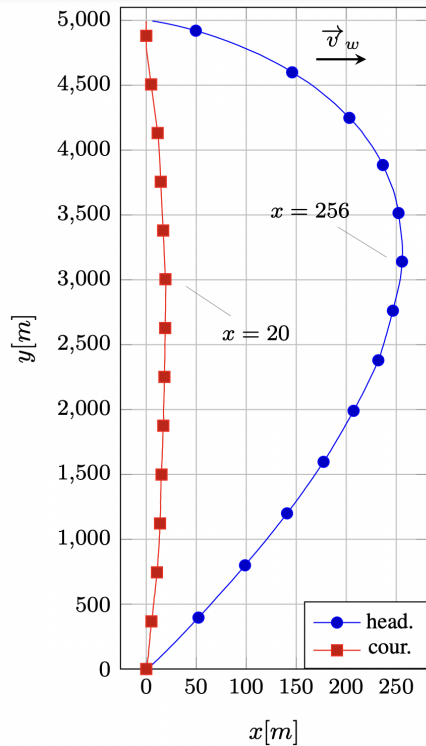


Figure 17: Course control vs Heading control - simulation from [46]

### 2.6.2 Component based control system

Traditional sailboats usually rely on a sailor’s ability to sense its environment and decide the next course of action based on the collected information. In these situations, a stable AoA and heading may not be the primary goal and are often unrealistic to achieve. Furthermore, sail angle can be changed by adjusting the sail or turning the boat. A sailor will often use a combination of these inputs to adjust sail angle. Robotic approaches often try to mimic human solutions [36], however, in this case, it might be easier to separate it in two individual tasks [36][49]:

- Optimal sail angle to maximise forward thrust
- Adjust rudder in order to reach desired direction

A decoupled component-based system can make the logic behind an algorithm easier to implement and understand [49]. In an article from Ocean Engineering 2010 [36], an independent system was developed and tested with a Velocity Prediction Program and a sea trial. The solution proved to be efficient. However, an open-ocean test in harsher weather was lacking.

### 2.6.3 Route planning

Sea path planning can be difficult depending on its complexity and the number of factors to consider. Path planning solutions, outside of sea traversal are used in numerous sectors such as aerospace[50], cars[51], robotics, and video games [52]. Some of the goals of these sectors are similar, and some are different. The shortest path is the most specific goal, but for some solutions,

---

the measured distance may not be the ideal for optimal traversal[46], as minimizing the time or resource used may be desirable.

A search space is the domain of all possible solutions for an algorithm's problem. Several solutions are used for dividing the search space into efficient data structures, for example, a Probabilistic Road Map[46]. The algorithm creates a graph by randomly placing points in an area. One possible disadvantage is that the algorithm prioritizes points of interest, such as fixed obstacles[46].

A more straightforward solution to reduce the search space is dividing the area into equally sized nodes in a grid[53]. Depending on the resolution of the search space and the search algorithm, it can be lightweight or quite heavy computationally.

Traversing a network representing possible paths can be solved by numerous known algorithms such as  $A^*$ [53][54], Dijkstra's Algorithm[46] and rapidly exploring random tree algorithm. Some of these algorithms require cost functions to distinguish which edge should be prioritized over another. Such cost functions are case-specific, but for ocean voyages, weather and current forecasts may influence how desirable a route may be[53]. Forecasts can be vital as they can provide time-efficient traversals across large distances[55]. Furthermore, safety distances from obstacles, other vessels, and shore should also be taken into account[46].

#### **$A^*$ and Dijkstra's Algorithm:**

These algorithms are among the more well-known algorithms used to find the optimal path. The main difference between the two is that  $A^*$  uses heuristics to guide its search area. Improved performance can be achieved with this heuristic[56]. However, Dijkstra's algorithm will examine every possible path until the specified goal is reached[57].

#### **2.6.4 Haversine - Distance between two point on a sphere**

The Haversine formula is commonly used to determine the distance between two points on a sphere[58][59][60]. In navigation, the Haversine formula is a special case derived from a general formula in trigonometry of spheres. This formula makes it possible to calculate the distance between two coordinates on Earth. However, not entirely correct due to the Earth's ellipsoidal shape, but quite close. The radius along the Earth's poles is approximately 6399.594 km and approximately 6335.439 km at the equator. Thus, the haversine formula cannot guarantee an estimation better than 0.5%.

#### **Haversine formula:**

Decimal degrees, usually written in angles, must be multiplied with  $\frac{\pi}{180}$  to transform it to radians.

$\theta$  is the central angle between two points on a sphere.  $d$  is the spherical distance between the two points.  $r$  is the radius of the sphere.

$$\theta = \frac{d}{r} \tag{2}$$

*The haversine of  $\theta$  :*

$$h = hav(\theta) \tag{3}$$

*The haversine function :*

$$hav(\theta) = \sin^2\left(\frac{\theta}{2}\right) \tag{4}$$

---

With direct use of coordinates in decimal radians, *latitude* :  $\varphi$ , *longitude* :  $\lambda$ .

Where  $(\varphi_1, \lambda_1)$  is the first coordinate, and  $(\varphi_2, \lambda_2)$  is the second:

$$hav(\theta) = hav(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot hav(\lambda_2 - \lambda_1) \quad (5)$$

Or to avoid using *cosines* which can cause degradation at small angles:

$$hav(\theta) = hav(\varphi_2 - \varphi_1) + (1 - hav(\varphi_1 - \varphi_2) - hav(\varphi_1 + \varphi_2)) \cdot hav(\lambda_2 - \lambda_1) \quad (6)$$

To get the distance,  $d$ , between two points where  $r$  is the sphere's radius.

$$d = 2 \cdot r \arcsin(\sqrt{h}) \quad (7)$$

Note:  $0 \leq h \leq 1$ , floating point errors can make  $h > 1$  which must be hindered.

### 3 Approach

This section presents an approach for developing and testing a system for an autonomous sailboat. When developing a complex system, organizing and documenting the work can help keep track of the progress as the system evolves. This section will introduce and describe techniques and tools to support development. Further, the section proposes a distributed system for an autonomous sailboat consisting of on-land services, sensors, communication, and actuators. Software and algorithms for the system, enabling autonomy, are outlined. Lastly, the section discusses the theory and techniques for testing such systems.

#### 3.1 Organizing and building a sailboat

While working on larger projects in a team, a good structure are important to keep track of the project's progress. There are a plethora of helpful digital tools that can assist teams in projects to improve the results. Such digital services can, for example, share changes in a task's status, visualize essential deadlines, and provide an overview of the project between members. Thus, making it easier for each member to be up to date on every aspect of the project. Tracking minutes spent on different aspects of the development process will also help the team optimally distribute available work hours and ensure the workload between members is somewhat equal.

Documentation can help describe the history of the project's development. It should include information on how to use it, how it works, what is complete, and what is lacking. The more complex a solution is, the more critical the documentation can be, as associates may forget crucial details. Further, documentation can make specific tasks more approachable as new team members are onboarded into the project.

Prioritization of tasks can be helpful to have some way to distinguish them. Critical assignments that are delayed could have tasks that succeed them and therefore can be worked on in the meantime. This way, essential features may be integrated, and progress made, even when there are obstacles during the development process.

---

### 3.1.1 Workflow

Reviewing team members' work and documentation helps to understand and keep up with the progress made during development, increases learning output, and reduces the chance of errors. This workflow can be achieved by creating a pipeline with organized phases each task needs to complete before being considered finished. Examples of such phrases could be: to do, in-progress, in-review, and done. When working on a more extensive software system with simultaneous development on different parts of the system, keeping track of changes and versions is crucial. When complexity increases, the ability to review changes, revert to previous versions, and handle merge conflicts between changes in the same document are valuable tools. It also enables multiple developers to work on different features and versions of the same system before combining the individual parts as they get finalized. The usage of digital tools to improve workflow is highly recommended.

## 3.2 On-land services

Some services do not necessarily have to be located on the sailboat. Moving resource-intensive and mission-critical tasks from the sailboat to land can help improve performance and reliability. This section describes how to achieve such improvements by introducing intermediary servers and an HMI.

### 3.2.1 Intermediary on-land server

An on-land server should be used for two-way communication with the boat and store data for further investigation and analysis. The data can prove practical to spot undesired behaviors or inadequacies. Additionally, a server could reduce the workload on the boat's main computing unit (MCU), making it possible to be smaller and more power-efficient. A data storage on land has a larger capacity and is a safer place than on a boat, which can be exposed to salt water or other hazards. When the server acts as an intermediary between clients and boats, limitations like connection speed, power consumption, and accessibility are reduced.

In order to reduce the load on the boat's power-efficient computers, a proxy pattern could be a suitable design pattern where one utilizes an intermediate computer. The pattern is supposed to create an interface layer, adding more logic to the desired data and reducing stress on the client. The server receives data from the boat and broadcasts the latest update to all connected users. Changes from the end-user are sent to the server and redirected to the boat's computer when necessary. Typical chores on the server could be reading, analyzing, storing, and distributing data from the boat.

Making data flow generic increases extensibility at the cost of security. It allows any data to be received, processed, and labeled, thus making it easier for new component data to be integrated without having to change anything server-sided. However, mistakes or intentional exploits can compromise the server when no data control exists.

According to the literature study, Section 2.2.2, cloud computing allows for scalability based on workload, meaning the more users, the more resources are assigned. Further, cloud services can scale down resource usage when not needed. Such technology allows a server solution to be more future-proof as it supports adding more boats and clients. For these reasons, it is suggested to utilize cloud computing.

---

### 3.2.2 HMI for monitoring and control

The purpose of the HMI is to visualize data provided by the intermediate server to enable an operator to monitor and control the boat when it operates autonomously, Section 2.1.3. In developing the HMI, prioritizing accessibility above high performance is desirable, as performance is not an issue with the current level of technology. However, the client-side application can be platform-specific in search of higher performance, thus restricting the usage of such an application to future associates and users as not everyone has access to specific platforms. The user interface should be intuitive for the end-user and secure enough to prevent the user from performing exceptionally system-breaking actions. The communication solution with the server should be similar to the boat since any slower will reduce the real-time aspect of the data flow, and any faster would be redundant. The complexity of the communication could also be kept at a minimum by utilizing similar technology. A client-side application should have some control over the boat to allow remote control by an operator. Either by instructions, direct control, or other means. The end-user should be able to provide the path for the boat through the HMI. However, another approach could be a computer-generated path as mentioned in Section 2.6.3. Even way both methods should ensure a safe voyage.

The following tables list functional requirements recommended for an HMI to achieve monitoring and control.

Table 2: Functional requirements authentication

---

<b>Authentication requirements</b>	
FRA1	There should be a clear login screen for a user
FRA2	A message should be visible when authentication errors occur
FRA3	The user should be able to register and create an account

---

Table 3: Functional requirements for the map

---

<b>Map requirements</b>	
FRM1	There should be an indicator of the boat's current position and its target
FRM2	Waypoints should be visible on the map when created
FRM3	A path editor should be visible when editing a path
FRM4	Current path should be visible and allow for editing
FRM5	A data file for the path should be possible to download
FRM6	A valid data file for a path should be able to be uploaded and activated as the current path
FRM7	New paths should be generated and sent to the boat and respond with an indication of success
FRM8	Changes to the path should be visible to all clients

---

Table 4: Functional requirements for the data visualization

---

<b>Data visualization requirement</b>	
FRD1	Visualization should present sensor data from the boat
FRD2	The visualization should be clear and intuitive
FRD3	Relevant data representation should include current and target values

---

---

Table 5: Functional requirements for the command system

---

<b>Command system requirements</b>	
FRC1	Clients should be presented with a list of available commands
FRC2	Each command should indicate what they do
FRC3	Input fields should be visible and have a label for their purpose
FRC4	A description of the command should be visible for the user

---

Table 6: Functional requirements for data and parameters

---

<b>Data and parameter requirements</b>	
FRDP1	Clients should be able to have a visualization of all raw data and parameters
FRDP2	Data from sensors should not be editable
FRDP3	Parameters should be allowed by the user to be edited, but with safety restrictions

---

Table 7: Functional requirements for the notification system

---

<b>Notification system requirements</b>	
FRN1	Popup notifications should be visible for the user
FRN2	Notifications should be updated with the latest messages from the sailboat
FRN3	Indication of the time the notification made should be available
FRN4	The notification center should have a history list of earlier notifications
FRN5	A client should be able to turn off the notification popup

---

### 3.3 Sensors and communication technology

The sailboat must have a MCU that can communicate to on land server and services, act on sensor data, execute commands and report errors. These requirements make a certain computing power necessary, as algorithms and numerous concurrent tasks run in multiple threads can be a heavy load on the central processing unit. An overview of the proposed architecture can be seen in Figure 18.



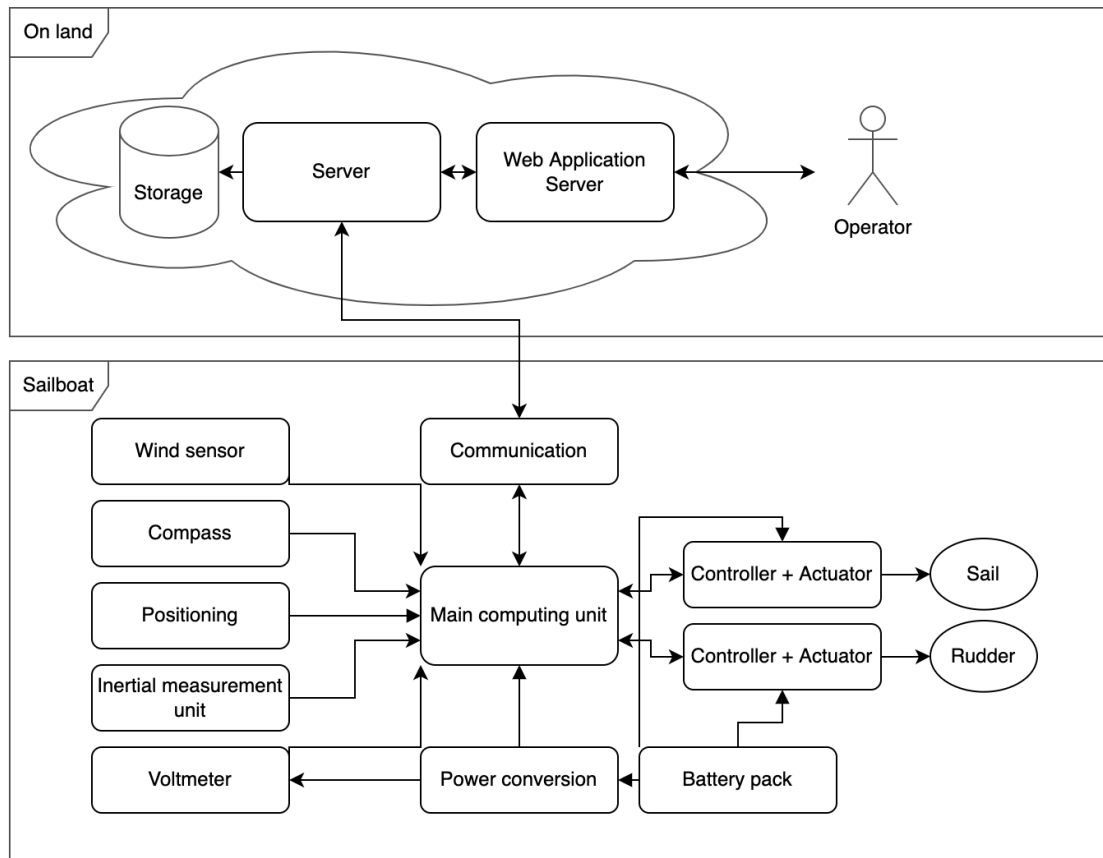


Figure 18: Proposed general hardware architecture

### 3.3.1 Sensors

Numerous sensors are required to create an autonomous sailboat that can perceive the environment and its internal state. A wind sensor must provide wind speed and direction, as it is necessary for sailing. global positioning system (GPS) allows the MCU to determine a precise position of the sailboat's location, velocity, bearing, and distance to a target position. Further, a compass is needed to know the boat's heading and an IMU to relay information about acceleration and gyroscopic forces acting upon the boat. Lastly, a voltmeter could provide information about the state of charge, power consumption, and when the system can not be able to operate due to a lack of energy.

In order to know the rudder and sail position, some encoders are vital. It should be able to reflect its absolute position digitally as it is essential for controlling the boat. Some encoders provide absolute positions. Others may require calibration when powering on and a solution for keeping track of the position.

### 3.3.2 Communication technology

According to the literature study, Section 2.1.1, communication technology on the MCU should allow real-time data flow for up-to-date information as required by the digital twin. Further, communication is needed to collect information and monitor the boat remotely. During testing, such communication would be beneficial when continuous adjustments are necessary. While the



---

The forces on the rudder will mainly come from the boat's forward velocity through the water. Other forces like currents are not included in the calculation. The equation for forces from an area  $A$  moving in a fluid with density  $\rho$  and velocity  $v$  is

$$F = \frac{1}{2}\rho v^2 AC_d \quad (9)$$

where  $C_d = 1.28$  for a flat plane.

With a 90-degree rudder deflection, the torque exerted on the rudder shaft by the forces from the water is given as:

$$T_r = F_r x \quad (10)$$

The torque from the actuator on the rudder shaft is given by:

$$T_a = F_a r_1 \quad (11)$$

Actuator torque should have a safety margin of 2 to ensure reliable operation when the sailboat is within design conditions. The actuator must be weather and salt waterproof and provide an interface for controlling it. Inputs are power and move in/out signals. Feedback from the actuator includes the current position and all the way in/out signal. In/out signals are necessary to calibrate the actuator and prevent it from overextending.

### 3.4.2 Motor for moving the wing sail

A motor inside the sail provides clockwise and counterclockwise rotation around the mast when a computer sends a corresponding signal. The computer can determine absolute position with calibration and continuous feedback from the sail. When the sail has reached its target, it holds the position until a new target is given. A worm gear increases the motor's torque and mechanically prevents the sail from moving without the motor spinning. The motor must be able to turn the sail into the wind even under maximum design load. Equation 9 is used to calculate the force from the wind hitting the sail with a 90-degree angle of attack and 0 degrees of roll. Torque on the sail is given by the center of force on the area from the trailing edge to the mast, subtracted from the center of force from the mast to the leading edge of the sail.

$$T_w = \frac{A_t}{A} F_w x_t - \frac{A_l}{A} F_w x_l \quad (12)$$

Motor torque needed can be calculated from gearing ratio  $i$ , gear drive efficiency  $\eta$ , torque from wind hitting the sail  $T_w$ , and a safety margin of 2, resulting in the following equation:

$$T_m = \frac{2T_w}{\eta i} \quad (13)$$

The equation does not account for other losses in the system that might further increase the torque requirement. The worm gear mechanism, motor, and encoder are mounted inside the sail to protect it from weather and seawater during operation, thus removing the need for waterproof components.

---

### **3.4.3 Power supply**

A battery pack with sufficient capacity should be used to power the sailboat. This portable power source removes the need for a cable running to the boat during tests and will be necessary for missions on the open sea. Power generation might be needed for more extended missions but is not covered in this thesis. Components like computers and actuators often require different supply voltages. Power converters can supply different voltage levels from a single source. Power is a critical system component, so it should be monitored using a voltmeter to prevent blackout during a mission.

## **3.5 Software development for autonomy**

When creating a complex software system, designing an architecture that can support current and future features is difficult but essential. Making changes to an implemented architecture can require considerable rework. When implementing a reasonable architectural plan can help prioritize and provide an overview. This section proposes a general software architecture separated into two main parts Figure 20.

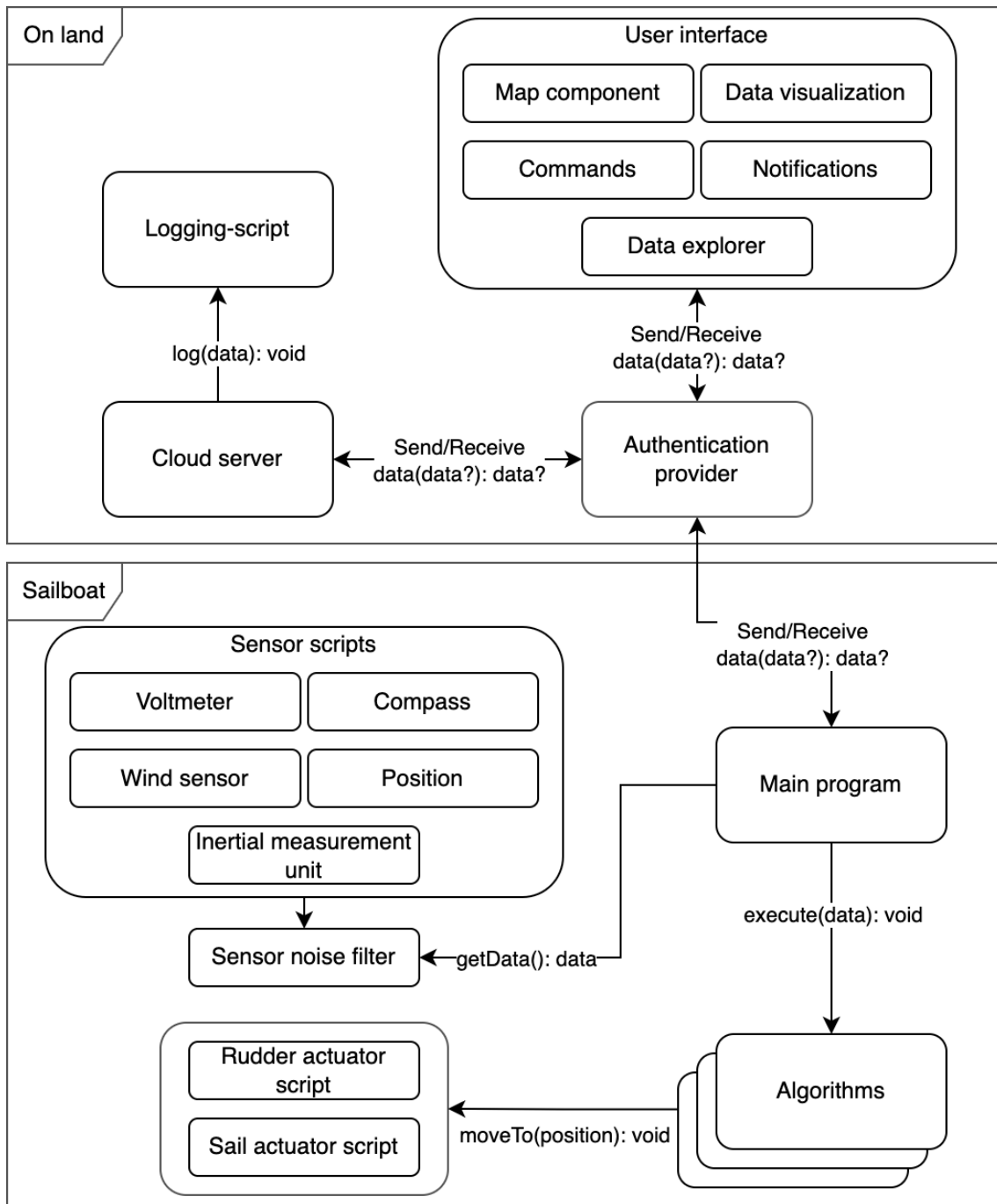


Figure 20: Proposed general software architecture

The goal of the architecture is to decouple components and tasks and keep the software as modular and readable as possible. Dividing software into subgroups after component classes helps with reaching this architectural goal. Almost every software program needs a starting point, usually known as the main program, which is the boat's core in this context. A star topology design pattern is suitable, assigning the main program to act like a hub[61]. The hub should do routine tasks as primary assignments and centralize communication and decision-making. Essential routines include updating the boat's state, recalculating actuator positions, and communicating with the on-land server to share state and receive updates or tasks. These duties are examples that the main program can execute in intervals.

---

Each task and routine should be separated, descriptive, and have minimal data flow to aim for extensibility. By doing so, modifying or adding a new algorithm is more straightforward. Furthermore, to improve modifiability, constant values that change the system's behavior should be adjustable by external users and not locked to a released version of the software. As a hub, the main program can act accordingly with the data's context.

### **3.5.1 Data processing, noise filtering**

As mentioned in the literature study under section Kalman Filter, a noise filter may be advantageous when processing data. Sensor measurements can be inaccurate, and significant fluctuations may result in extreme algorithm behaviors in worst-case scenarios. More reliable sensor measurements can improve the overall performance of the algorithms. If the data is accurate enough, a filter is redundant and introduces unnecessary delays when trying to perceive the environment.

### **3.5.2 Security through authentication**

Security becomes important with unrestricted data flow to the server and a high degree of control over the sailboat. Every client wanting to connect to the system must undergo an authorization process where the server validates identity before access is granted. This authentication includes the sailboat, operators, and developers. Traffic between the server and authorized clients is encrypted to prevent attacks on the data sent and received.

## **3.6 Algorithms required for autonomous sailing**

During autopilot, the MCU must be able to have the latest data from the sensors and adjusted parameters from the on-land server to execute algorithms. Such algorithms must be able to tell what the sailboat needs to do to get closer to the desired goal provided by the operator.

One of the minimal requirements to sail is to be able to sail in all directions. From the literature study, Section 2.6.2, a decoupled approach to sailing control may be a good option. The rudder should always strive to keep a steady course towards the target position. On the other hand, the sail should always try to optimize its position to gain as much force from the wind as possible.

To be able to sail upwind, a solution according to Section 2.5.2 is beating. Given the speed required for tacking, the algorithm should ensure the sail minimizes the wind resistance while passing the no-sail zone. The no-sail zone's size depends on the boat's physical specifications. Further considerations should be made about how far off the straight line between the current and target positions the sailboat should travel. According to Section 2.5.2, the least amount of tacking will result in less travel time. However, different strategies may be more suitable depending on factors like weather forecast, distance and traffic routes.

## **3.7 System testing**

Tests can validate whether something works to a certain degree, and various bugs can be uncovered. Different types of tests are designed for different kinds of verifications.

For smaller, isolated tests, unit testing helps check expected behavior, for example, whether the

---

sensor's output is correct or proper for a set of requirements. For quality assurance, unit testing is essential. However, some errors may not be discovered, and integration and complete system issues are not exposed.

An integration test allows for multiple modules to be tested together. The goal is to verify that input from one system gives the proper output in another. Hence, a validation of the interoperability between chosen components. Integration testing is a decent solution for systematically bringing together a system and exposing errors related to interfacing. However, with complex system integrations, finding the origin of the defect can be challenging.

At a late stage of development, a testing technique called the acceptance test can be executed. The technique validates the whole system and whether it fulfills a set of requirements from stakeholders or others. Acceptance tests focus more on user needs rather than just technical implementations.

---

## 4 Implementation

Based on relevant literature study and steps outlined in the approach, an implementation process was conducted and is presented in this section. It introduces standard tools for organizing large projects with two or more members. Then the development and deployment of a cloud-based server are described, followed by another on-land service in the form of a web application. The selection, development, and implementation of sensors, actuators, and electronics required for an autonomous sailboat are presented in detail. Computer and microcontrollers with custom software are introduced to read sensors and control actuators. The software implements a set of algorithms that uses the input to calculate output to achieve autonomy. Lastly, an authentication system is implemented, and an overview of the whole system is given.

### 4.1 Documentation and workflow

Several digital tools were introduced to improve workflow. Trello is a web-based list application with a Kanban style[62]. It helped organize tasks, thoughts, and issues. Furthermore, keeping track of their statuses and the project was more manageable for every member involved. A beneficial feature was also a labeling system, allowing for quick identification of severity and category. A section of the Trello board can be seen in Figure 21.

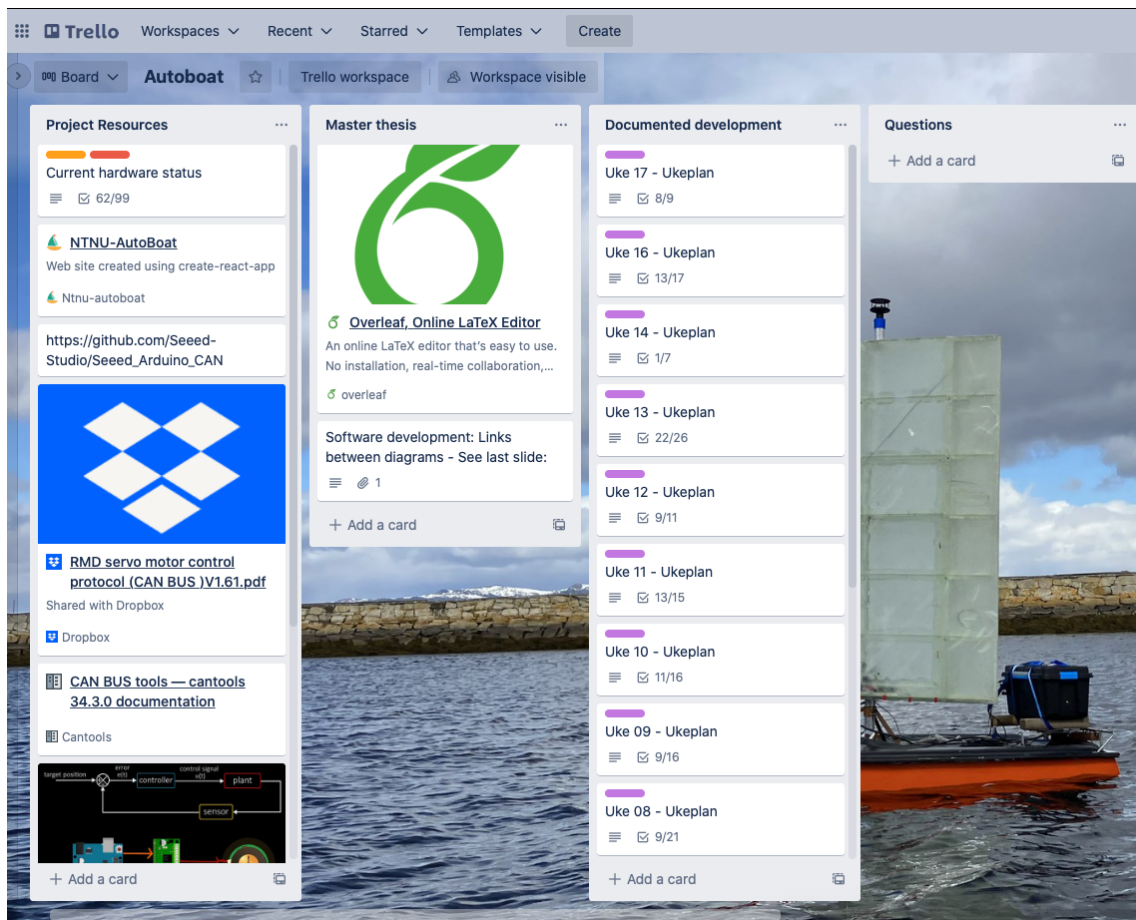


Figure 21: A section of the Trello board

Hosting and version control of source code was provided by GitHub[63], one of the most popu-



---

lar providers of such services. It makes history and description of each iterative software change available for the entire project. As seen in Figure 22, an organization was created to structure components into repositories. With the project’s comprehensive software requirements, an organization helps keep all the code in one place.

The software development workflow in 3.1.1 was followed rigorously. Such a workflow provides code safety and easier bug fixing, allowing every developer to understand how the system works together. Much time went into documentation, explaining how each software works and how to download and execute each program. Some of the documentation, called "README," can be seen in Appendix Section C.

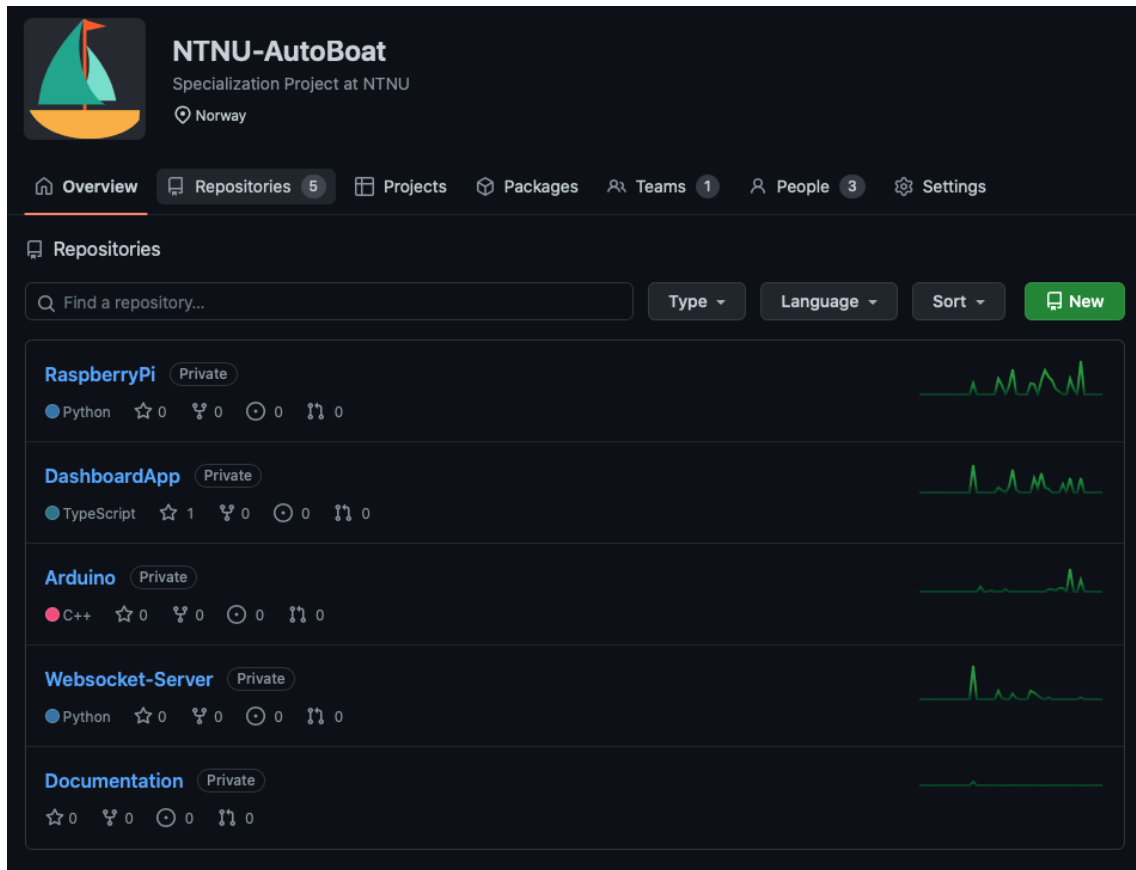


Figure 22: GitHub Organization[64], Note: The organization is private, but can be accessed by request to authors

## 4.2 On land server and cloud computing

The on-land server was based on a server developed during the pre-study of this thesis[1]. The server implements a WebSocket server to communicate with clients. It was hosted on a local machine which was not sustainable for the future of this project, so a move to an external server was mandatory. Cloud Computing mentioned in 2.2.2 was chosen as a solution due to the benefits of accessibility, scalability, and the numerous integrated services by the providers of the platforms. Since Google Firebase was already hosting a dashboard from the pre-study, Google Cloud Platform (GCP) was the chosen cloud service provider.

---

### 4.2.1 Server hosting service

A service provided by GCP called Cloud Run was used to host the on-land server. The service was cheap, scalable based on demand, and allowed multiple simultaneous requests. Additionally, as the service was reasonably new, Google advised the use of Cloud Run to host containerized applications.

### 4.2.2 Containerizing and building

In order to containerize the program, a Docker image is created by Google when deploying the server. A build file was developed to improve the workflow that instructed Google's Cloud Build service on how to create the container image and host the server. The continuous deployment function is executed automatically each time a new feature is implemented and uploaded to the version control provider, GitHub.

Table 8: Container instance specifications

Container Capacity	
CPU allocation	Only allocate during requests
CPU	1
Memory	256MiB
Concurrency	80
Request timeout	300 seconds
Execution environment	1st. generation

### 4.2.3 Run time optimizations

Because Cloud Run reduces server costs during idle time, a goal was to allow the server to shut down without breaking the system. This required the WebSocket connection to have a way of initializing and storing the latest values before being terminated. The solution was to let the server save the latest data once every minute if any updates were received. When the server restarted, it would initialize with the last state from the previous server instance.

### 4.2.4 Logging and cloud storage

A logging feature was developed during the pre-study project to store values received as a comma-separated values (CSV) file on the server. An object storage service from GCP called Cloud Storage was utilized to keep this functionality. Once every minute, the server will update the corresponding file for the day with the latest data transactions accumulated over the last minute. The service will be accessible to every developer at a meager cost for the project's entire lifetime.

## 4.3 Dashboard

The development of a dashboard is a continuation of the pre-study of this thesis[1]. As a hosting service, Firebase Hosting is unchanged. Further, the continuous deployment solution with GitHub

---

remains intact for rapidly updating the hosted dashboard. Several functional requirements were listed in Section 3.2.2. This section highlights new features implemented.

#### 4.3.1 Path planning

In order to give the boat a route, an easy-to-use path planner was developed. The feature required a custom data structure to achieve desired features, adjustability, and performance. A path consists of numerous waypoints connected. After the boat has reached a waypoint, the next in line will be the next target for the boat to follow. A waypoint consists of the following data:

Table 9: The data in a waypoint object

Waypoint Object	
Id	<i>uid</i>
Position	<i>lat, long</i>
HasVisisted	<i>boolean</i>
Next	<i>uid</i>
Previous	<i>uid</i>

This data structure represents a chain in a linked list. A linked list is performant when removing and adding new elements to the path. The first waypoint of the path will have no data in the "previous" field, as it is the first element of the list. The same is true for the "next" field of the last waypoint of the path, as it is the last element of the list.

The entire path can be downloaded as a JSON file, and a similar file could be uploaded to import a saved path. The current path of the boat can be viewed on the home page map and adjusted with an editing tool. Adding, deleting, and reordering the waypoints are possible. It is possible to send a new path to the boat, and every other user using the dashboard will have a visual update of the path on the map.

#### 4.3.2 Notification center

Incoming error codes are visualized as a notification on the dashboard. It displays a message explaining the context of the error code and can be closed by clicking the X. The background color represents the severity of the error, with four levels ranging from normal to high. Furthermore, a timestamp can be viewed on each error message to know when the message was sent. The error codes and messages are stored in the source code as a JSON file.

#### 4.3.3 Instruments and the voltmeter

The instruments page received a new instrument called a voltmeter. It shows the user a measurement of the current-voltage level in the battery pack. Further, instruments showing the current heading, rudder, and sail position, received a green indicator showing the target position of each of the named instruments to reflect what state the algorithm wants to reach.

---

#### 4.3.4 Commando system

With the need to control the boat, a flexible command system were developed. In order to keep it up to date, a Firestore database from Google stores all the available commands. As more commands are added to Firestore, the dashboard will be updated and allow execution of the new commands. To have a predictable format, a commando object have these properties:

Table 10: Properties of the command object

<b>Command Object</b>		<b>Parameter Object</b>	
Computer	<i>string</i>	Name	<i>string</i>
Component	<i>string</i>	Order in array	<i>number</i>
Command	<i>string</i>		
Description	<i>string</i>		
Parameters	<i>objects</i>		

When the computer has tried to execute a command, the command system allows for sending a new command. If the command has not been executed, the operator can remove the command with a button.

#### 4.3.5 Data viewer

A page to view and edit JSON structures was added to visualize raw data. It included tools to modify data so parameters could be tuned. Rules were set in place in order to keep the system from breaking. In order to edit a variable, the data type must be the same, or the update will be rejected.

### 4.4 Sensor technologies and solutions

Sensors are necessary for the boat to perceive its environment. This section will describe the sensor technologies and solutions that are implemented in this project.

#### 4.4.1 Sensor to determine the angular position of the sail

There are different methods to measure the angular position of a rotating object using rotary encoders. The sail has two locations that spin with a fixed center of rotation. Those are the actuator shaft and worm gear, and the mast itself. The actuator is already equipped with a rotary encoder which an internal PID controller uses to rotate the motor to the desired position. However, data from this encoder is hard to access, and it lacks a reference point for the actual sail angle. To rely on an internal encoder also limits which actuators can be fitted to the sail.

An external encoder should therefore be implemented to provide a reference point and ensure the correct operation of the sail. As mentioned earlier, there are two possible locations for an encoder. The first location on the encoder shaft has limited space available due to the narrow profile of the sail. The second location is to measure the sail mast directly. The length and diameter of the mast make it challenging to mount an off-the-shelf encoder without an intricate mechanical interface.

---

The chosen solution was to turn the entire mast into an encoder by designing custom parts around it.

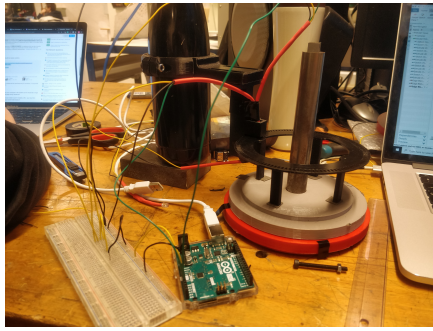
Several ideas and solutions were considered when designing the custom rotary encoder. One suggestion was an imitation of bar codes on the surface of the mast. An electric photo sensor would cast light onto the bar code and read the reflected light to determine if it were black or white. The software would use these signals to register changes in the sail's angle.

Another suggestion was using an infrared (IR) beam intersected by a disc with holes. This solution would work the same way as the bar code but instead read the start and end of IR transmission as changes in the sail's angle. This solution was selected in favor of the bar code as IR emitters and receivers are reliable and easily available. Figure 23a shows early testing of this method.

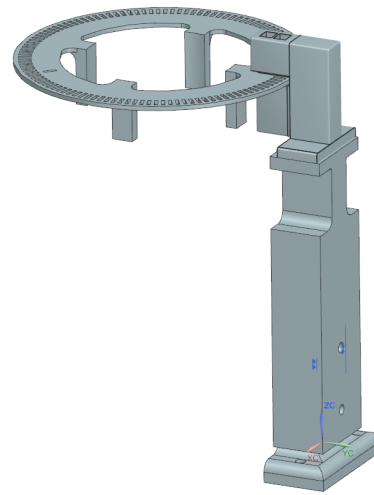
During the development of the disc, many parameters were adjusted with performance and simplicity in mind. The first disc was 3D modeled and printed with orange PLA plastic. Precision levels were initially set to 1 degree, which meant 180 holes around the disc. Early tests showed that reflection was an issue leading to a change of PLA color to black.

The first IR receiver tested was a TL1838, regularly used for reading IR signals from wireless controllers. However, it would only read IR specific pulses, not a continuous beam. Another prototype used a regular light diode and a photoresistor. However, too much delay in the photoresistor led to inaccuracy. After more research, IR were given another chance using a receiver called BPV10NF. The new IR receiver sent a voltage that could be measured when exposed to IR light. It was quite sensitive to interruptions and, therefore, suitable for the task. Figure 23b and Figure 23c is the computer aided design (CAD) model and printed prototype mounted on the sail for testing.

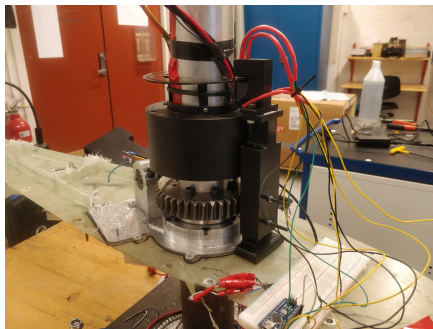
Finding the right-sized holes and edges to get a reliable readout required much testing. The final disc changed from 180 holes to 120 holes, each 1.0mm wide, resulting in a resolution of 1.5 degrees. The disc also had a second row with only one hole used to make a point of reference. A second IR beam was integrated to read this position. Different iterations of the disc are displayed in Figure 23d.



(a) Early development and testing of encoder



(b) CAD model of encoder assembly



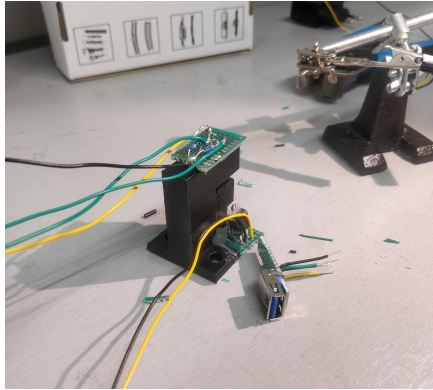
(c) First prototype mounted on sail



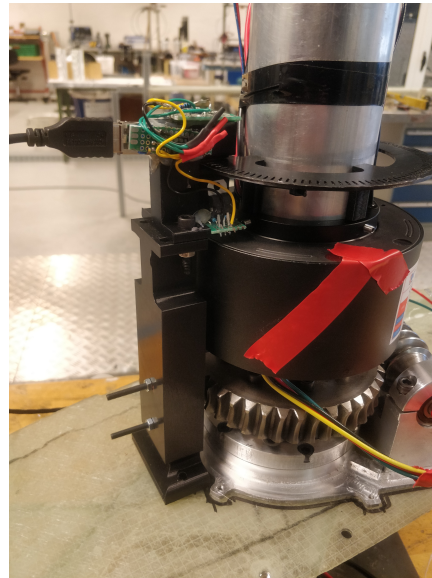
(d) Iterations of encoded disc design

Figure 23: Development of rotary encoder

A USB connector was repurposed to make it easy to connect the encoder, as seen in Figure 24a. A power supply supplied 5v from the 24v battery pack to the receiver, while a voltage regulator in the encoder stepped it down to 3v for the emitters. Further, some trial and error were needed to find the perfect disc diameter to fit onto the mast. The final version are depicted in Figure 24b.



(a) Encoder reading head universal serial bus (USB) interface



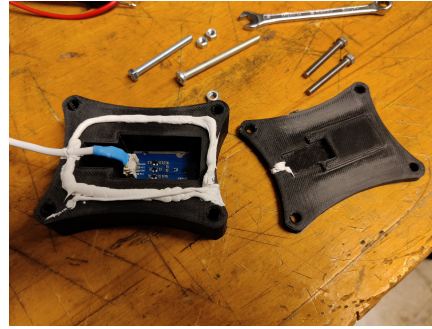
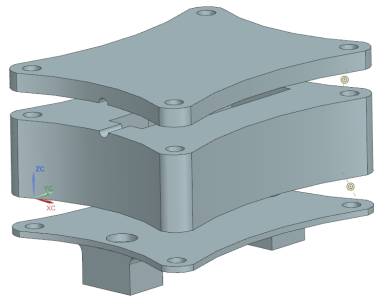
(b) Finished version of rotary encoder

Figure 24: Rotary encoder for reading sail angle

#### 4.4.2 Sensors for measuring heading, attitude, and location

A IMU consists of a gyroscope, accelerometer, and magnetometer. It can measure pitch, roll, and heading. From the pre-study project, the IMU was connected to the Arduino Uno through inter-integrated circuit (I2C). With further progress, it was decided to move it to the Arduino Nano. The reason was the dependency between the rudder and the boat's heading. Having the IMU connected to the same computer which controlled the rudder reduced some complexity and improved performance. To ensure the IMU were reliably connected, an I2C socket was soldered to the circuit board connected to the Arduino Nano.

To move the IMU away from magnetic and electromagnetic noise a box was designed using CAD (see Figure 25a) and 3D printed. After installing the IMU, the box was sealed shut to be water-tight (see Figure 25b). Additional four bolts were used to secure the lid and attach it to the boat mount (see Figure 25d). A long cable and water-tight connector provided much freedom when deciding placement for the IMU.



(a) CAD model of IMU box with lid and boat mount (b) IMU installed in box and being sealed with silicone



(c) Mount attached to sailboat (d) IMU mounted on sail boat

Figure 25: Details of IMU

Before using the IMU, a calibration was necessary. The resulting offset values in x, y, and z were measured as -7, -7, and -42. A calibration mode was implemented to update these numbers if necessary. Furthermore, a magnetic deviation must be included to have the compass point toward the true north. These values can be found online for any location. At the time of writing, the magnetic deviation in Trondheim is 4.29.

x	y	z	magnetic deviation
-7	-7	-42	4.29

The raw heading value from the IMU was noisy and unreliable, posing the need for a signal filter. After the literature study in Section 2.4, integration of a Kalman Filter was done and resulted in drastically improved precision. The Kalman Filter is lightweight in both processing load and memory usage, which are favorable to a small microcontroller like Arduinos. A library offering a Kalman Filter for Arduinos was used to simplify the process [65]. Three parameters are needed to initialize a Kalman Filter:

- Measurement Uncertainty - How much do we expect our measurement to vary (MU)
- Estimation Uncertainty - The adjusted value can be initialized with the same value as above (EU)
- Process Variance - Usually a small number between 0.001 and 1. It reflects how fast the measurement moves. A higher value adjusts the estimation faster, making it more sensitive to new measurements. (PV)



---

During testing, suitable parameters were 2, 2, and 0.1 in the order above.

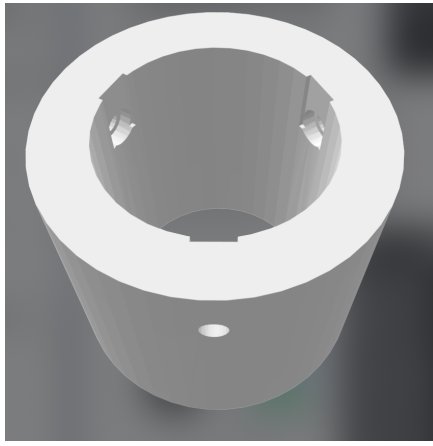
MU	EU	PV
2	2	0.1

A GPS sensor called SiRF Star IV by GlobalSat was connected to the Raspberry Pi through USB to measure position. Initially, it was configured to communicate with a SiRF Binary Protocol, but after some adjustments, it delivered data through NMEA0183, a standard in marine data communication. The recognized NMEA sentence transmitted is called GPRMC, which includes a position in longitude and latitude, speed over ground, and course over ground. Data transmitted from the GPS are read and processed by the Raspberry Pi in its own thread.

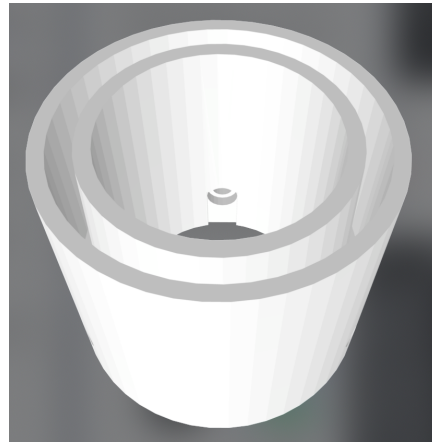
#### 4.4.3 Sensor to measure wind direction and speed

A wind sensor was mounted on top of the mast using a custom 3D printed adapter, designed with a press fit for 70mm aluminum extrusions. The signal cable can pass through the adapter and connect with the wind sensor, which can then be inserted and fastened with three screws. The wind sensor uses an RS232 interface over a CAT6 cable and is connected to a conversion board on the Raspberry Pi inside the electronics box. New data is sent from the wind sensor every second, and the raw data is processed using the EWMA filter introduced in Section 2.4.2. Under low wind conditions or gusty winds, the measured wind direction and speed can change rapidly, possibly causing the relatively slow turning sail to adjust constantly without ever reaching its target. The EWMA filter helps smooth out these rapid changes and returns a mean value. Wind direction is almost constant at a larger scale while many small and rapid changes might occur locally, making a low value of  $B = 0.1$  favorable to suppress new values. Values that can loop around, like wind direction, from  $359^\circ$  to  $0^\circ$  must be converted from polar coordinate to a complex number in the rectangular form to be used with an EWMA filter. After calculating a complex  $\bar{S}_t$  it can be converted back to polar coordinate and displayed as wind direction.

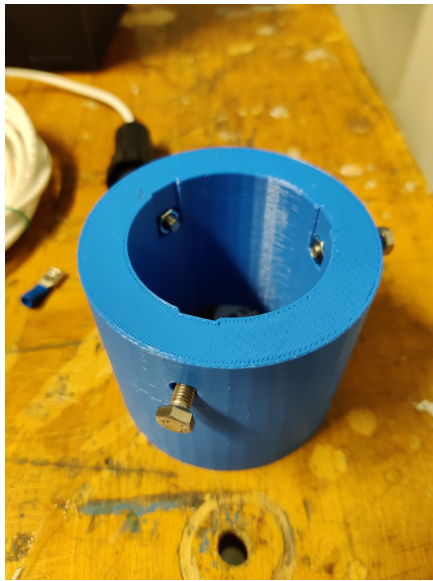
The filter process of wind speed is identical to wind direction, except there is no need to convert it to a complex number. Weight  $B = 0.5$  was chosen as the wind speed changes much faster, and these changes should be reflected.



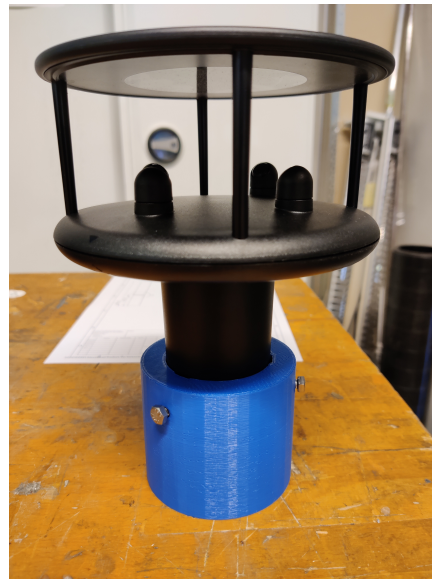
(a) Top view of CAD model of wind sensor adapter



(b) Bottom view of CAD model of wind sensor adapter



(c) 3D printed and assembled wind sensor adapter



(d) Wind sensor and adapter

Figure 26: Wind sensor mast adapter

## 4.5 Actuators and electronics

Actuators and other electronics enable mechatronic systems to change their state based on perceived input. This section will first introduce the selection and implementation of actuators for the sail and the rudder. The work and calculations done to mount them are explained and documented. This section includes some electronics to act as an interface between the actuator and microcontrollers. Lastly, the section present the remaining electronics and their placement.

### 4.5.1 Sail actuator

The sail wing foil and rotation mechanism has previously been designed, produced, and documented in master theses by Dyrseth [38] and Gauden [66]. This thesis covers further development of this sail, including wiring, electronics, motor, and mast extension.

Table 11: Properties of wing foil[38]

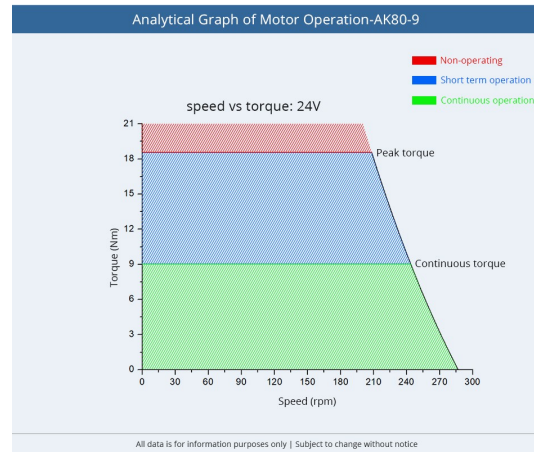
Properties of wing foil	
Cruise speed	1.5m/s
Max wind speed	20m/s
Height	2400mm
Length	1200mm
Area	2.88m <sup>2</sup>
Lift force	0 – 54N
Drag force	0 – 1.4N
Optimal AoA	6 – 11°

The rotation mechanism uses a worm gear with a 40:1 ratio and an 80mm diameter motor mount. Previously a Gyems RMD-X8 Pro servo motor was used to drive the worm gear. Due to quality and reliability issues, they had to be replaced. It was decided to use the opportunity to find new and better motors. In order to minimize rework, the goal was to find motors similar to the old ones but of higher build quality. Using Equation 12, the maximum torque from the wind on the sail is calculated to be  $T_w = 272.1Nm$ . According to the thesis by Gauden[66],  $\eta = 0.46$  and  $i = 40$  for the current worm gear. Using these values and Equation 13 the required motor torque with a safety margin of 2 is  $T_a = 29.571$  and  $T_a = 14.786$  without the safety margin.

Based on these requirements AK80-9 servo motor from CubeMars was selected. It could be fitted to the existing mounting hardware with only minor changes to the brackets, and it runs on 24v DC with communication over controller area network (CAN). One drawback is the rated torque of 9Nm and peak torque of 18Nm as shown in Figure 27b. This rated torque does not meet the design requirements addressed in the discussion section.



(a) AK80-9 servo motor from CubeMars



(b) Torque curve of AK80-9 servo motor

Figure 27: Pictures from CubeMars' website [67]

Power and signal from the electronics box to the servo motor are carried through an outdoor grade five conductor 1.5mm<sup>2</sup> cable from Biltrema[68] rated for up to 20 amps. The motor is controlled using a CAN interface, which carries commands sent from an Arduino with a CAN-shield.

The mounting point of the sail on the boat is a tripod that holds the mast. The bottom part of

the mast goes through the hull of the boat. Furthermore, a keel is attached to the end of the mast. Upon closer inspection, it turned out the mast was too short to reach the underside of the hull nor long enough to be extended. A new and longer piece was manufactured based on original drawings to replace the short piece.



(a) Turning new mast stub on a lathe



(b) First dry test of sail

Figure 28: Manufacturing and testing of new mast stub

#### 4.5.2 Rudder actuator and mount

The rudder used in this project was previously made by Brandal [69] to serve as a prototype for further development. Specifications for the rudder are as follows:

Table 12: Properties of rudder[69]

Properties of rudder	
Cruise speed	$1.5m/s$
Cord length	$330mm$
Span length	$500mm$
Area	$0.165m^2$
Range of motion	$120^\circ$

Calculating torque on the rudder from the water can be done using the same equation as from the sail, only changing  $\rho = 1024kg/m^3$  for sea water and  $A = 0.165m^2$ . A rudder deflection of 90 degrees was used for simplicity. It will result in a higher max torque than the max design deflection of 60 degrees and add to the safety margin. With Equation 10, the maximum torque from the water on the rudder is calculated to be  $T_w = 80.29Nm$ . This maximum torque is with a safety factor of 2.

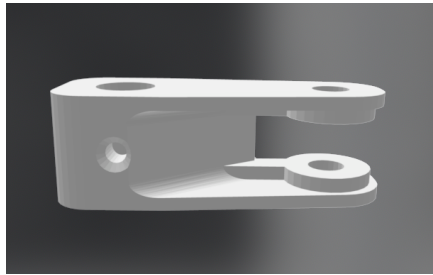
For reasons discussed in Brandal's thesis[69] and Section 3.4.1, a Linak LA33 linear actuator was acquired. Based on the specifications of this actuator, a lever arm and mounting mechanism were designed.

Table 13: Linak LA33 specifications

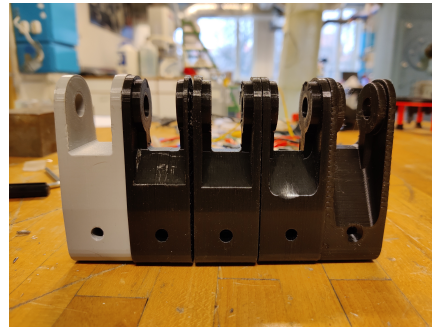
<b>Linak LA33</b>	
Load	2250N
Speed	21mm/s
Stroke length	100mm
Overall length	260mm
Stroke tolerance	$\pm 2mm$
Voltage	24v

The length of the lever arm can be calculated by solving the two equations mentioned in Section 3.4.1. Given the actuator length and desired range of motion the equations yields  $r_1 = 57.99mm$  and  $r_2 = 308.6mm$ . From Equation 11 the torque provided by the actuator is  $T_a = 130.5Nm$ . This torque is well above the designed load and should not be a concern.

A simple lever arm with length  $r_1 = 57.99mm$  was designed using CAD and 3D printed. The design was iterated until the prototype was performing without issues. The CAD model and different iterations can be seen in Figure 29. The actuator mount was created with scrap metal and some bolts, only intended for shorter test trips. It was mounted on an existing bracket on the boat, with a length of  $r_2 = 308.6mm$  from the rudder shaft to the actuator's rear mounting point. The lever arm and actuator mount were not analyzed for strength or stiffness as they were only intended to work for a few tests with the old catamaran hull.



(a) CAD model of lever arm



(b) Different iterations of lever arm



(c) Actuator mount on boat



(d) Actuator mounted on boat with rudder

Figure 29: Development of lever arm and actuator mount

An interface consisting of an h-bridge, voltage dividers, and shunt resistor ammeter is necessary to control the 24v actuator using a 5v Arduino Nano. Figure 30 shows the h-bridge, which receives

---

a 5v signal from the Arduino and switches two 24v outputs connected to the linear actuator. One input extends the actuator arm while the other retracts the arm.

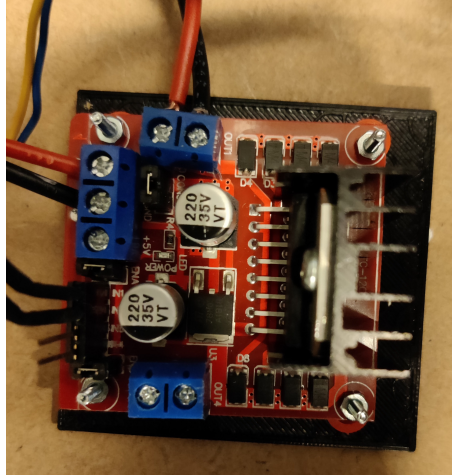


Figure 30: H-bridge to control rudder actuator

In order to read end stop signals, which are 24v, two voltage dividers were made to step it down. Following the schematics from Figure 31a the output voltage of the circuit can be calculated using this equation:

$$U_{out} = \frac{U_{in} \times R_2}{R_1 + R_2} \quad (14)$$

Given resistors  $R_1 = 100k\Omega$  and  $R_2 = 20k\Omega$ , the result is  $U_{out} = \frac{24v \times 20\Omega}{100\Omega + 20\Omega} = 4v$ . When one of the signals are high, the Arduino reads 4v, otherwise 0v. The actuator is equipped with an internal potentiometer which provides analog position feedback as a current  $I \in [4 - 20mA]$ . This feedback can be measured on an Arduino using a shunt resistor ammeter shown in Figure 31b. The shunt resistor  $R_1$  is found by applying Ohms law,

$$U = RI \quad (15)$$

in reverse for the expected voltage and current:  $R_1 = \frac{U}{I} = \frac{5v}{0.020A} = 250\Omega$ . A  $220\Omega$  resistor was selected to avoid accidentally exposing the Arduino to over voltage. This yields a measured voltage of  $U = 220\Omega \times 4mA = 0.88v$  when the actuator is retracted, and  $U = 220\Omega \times 20mA = 4.40v$  when it is fully extended. These values are updated when a rudder calibration sequence is executed to account for any slight variations in accuracy.

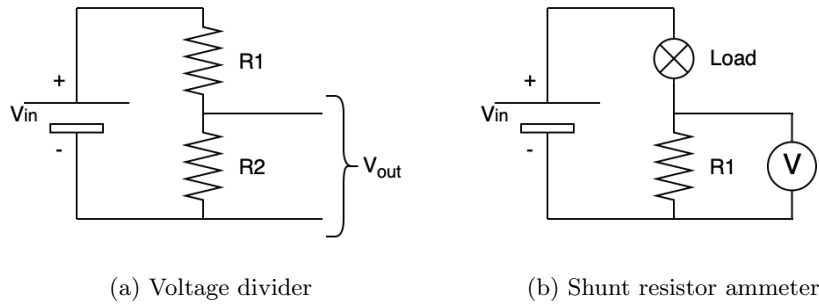


Figure 31: Circuit diagrams of voltage divider and ammeter

### 4.5.3 Electronics box

All the sailboat electronics are housed in or connected to the electronics box. It features six waterproof connectors to the external sensors and actuators. This connector setup simplifies transport and assembly of the boat as every component can be moved and mounted separately from each other and quickly connected when everything is ready. A layout of the connectors can be seen in Figure 32a. It must be followed closely as connecting a component to the wrong port can cause permanent damage.

After the first dry test of the sail, it became apparent that there was not enough clearance underneath the sail for the electronics box. The only suitable location was at the very back, out of range of the sail. In the same area, the actuator for the rudder took much space. Therefore, the box was mounted above the actuator. The mount was made from two aluminum strips bent at a 90-degree angle. Two pieces of wood were screwed to the aluminum to get the box above the actuator, as shown in Figure 32b. The mount worked well during the sea trial. However, the placement resulted in the boat having a constant pitch up.

Inside the box, four lead-acid batteries with a nominal voltage of 12v are glued to the bottom. Two pairs of batteries connected in serial are connected in parallel to output 24v. Figure 32c shows the batteries through cutouts on the bottom shelf. The shelf also contains two transformers, an Arduino Nano, an h-bridge, and a voltmeter. The transformer is a standard phone charger made for 12v car sockets. They are made to work on 12 or 24v and transform it to 5v. One is used to power the Raspberry Pi, while the other powers the sail encoder. It was chosen as a cost-effective, easily accessible, and reliable way to convert power from 24 to 5v.

The voltmeter is similar to the voltage dividers used for the actuator and steps down from 24v to 4v. It is connected to the batteries and the Arduino Nano. Four 3D printed pillars support the second shelf. This level contains the Raspberry Pi, Arduino Uno, GPS and 4G mobile modem. See Figure 32d.

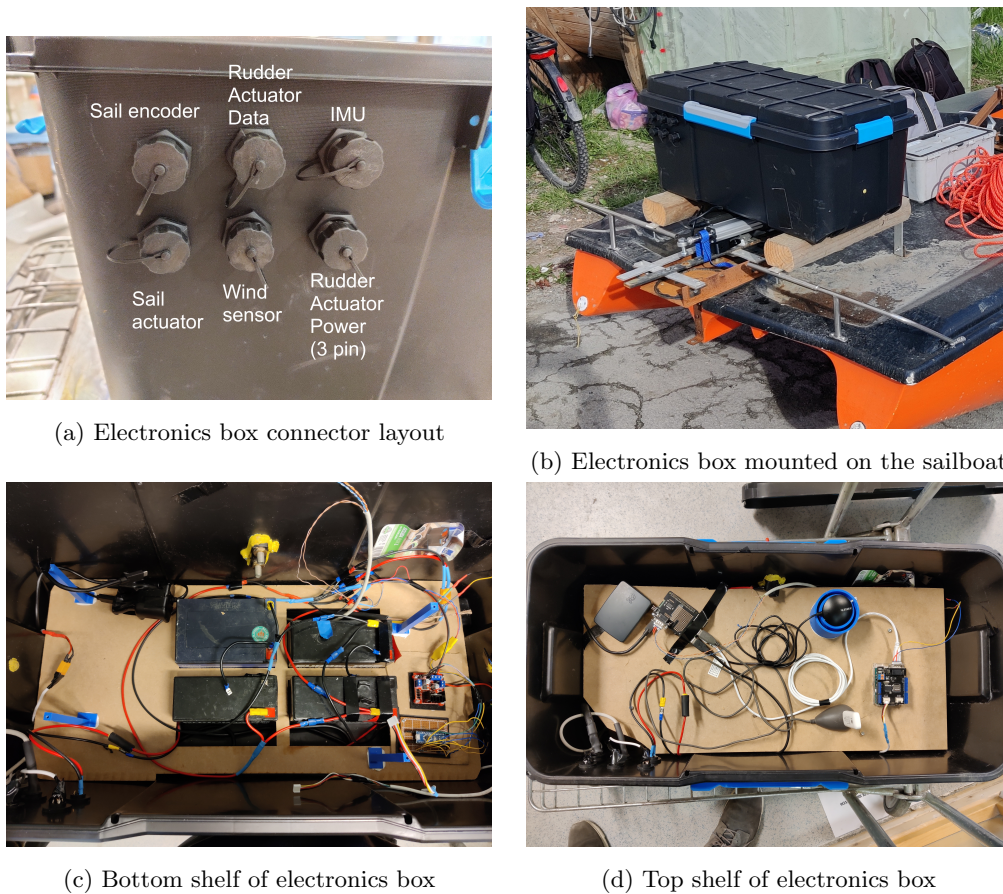


Figure 32: Details of electronics box

## 4.6 Computer and software implementation

The following sections will outline how computers communicate, read and process data. Further, some highlighted challenges during development and their solutions are presented. Lastly, structures and strategies are described where certain qualities are desired.

### 4.6.1 Microcomputer - Raspberry Pi

One of the challenges with using the USB-interface to communicate between components was the inconsistencies of which ports were registered on which component. As a result, a port scanner was required to identify each component. When the main program on the Raspberry Pi starts, it will execute a Python script to list ports with vendor names of the electric component connected. A solution was to register which vendor name belonged to that component and identify each with a vendor name. It required storing the vendor names as environment variables to have a way of mapping each component to its corresponding vendor name.

However, a problem was discovered when a second Arduino was introduced. There was no way of differentiating them because of identical vendor names—this hindrance required further programming on both communications ends. The solution was to allow the Raspberry Pi to ask every port that suggested they were an Arduino about who they were. A hard-coded name was stored on each Arduino so that they could respond to the Raspberry Pi. As a result, the Raspberry Pi could



---

now identify each Arduino without requiring manual configurations.

Nevertheless, it was not the only communication problem between the MCU and the microcontrollers. Serial communication through USB was not as straightforward as envisioned. When connecting the Arduino, there were inconsistencies as they did not always reset successfully or filled the USB-buffer with data. These inconsistencies confused the Raspberry Pi, as it did not know the start or end of a message. Therefore, a new communication solution added a start and end marker for messages, namely < and >. These message markers were added to both communications ends, using the same logic for sending and reading messages between the computers. The stability of message transmission improved and worked sufficiently enough for this project.

Error messages are not initially available through the entire system. Working blindly, not knowing where something had crashed, proved difficult. A Python class was created to handle errors. It would collect error codes, give them timestamps and send them to the server. The effectiveness of this error handler was beneficial. As a result, it was also extended to report successful actions and notable changes. For example, it reports when the Raspberry Pi is connected, when it shuts down, and when a component is connected.

A command structure was implemented to have external control over the components. A set of instructions could be sent to the Raspberry Pi, and it would execute the command or pass it to the correct recipient. This system allowed the user to give instructions, adjust parameters, and reset and calibrate components. The command message is a string and is structured as follows: `Computer!Component;Command:Param1,Param2`.

For the simplicity of new users, a shell script was developed that guides the user through the process of adding environment variables. It also includes steps for registering every variable required to execute the program with every electronic component connected.

All state and parameter data are stored inside dictionaries on the Raspberry Pi. The datatype inside varies and sometimes includes nested dictionaries. In order to have an easier time finding values inside these dictionaries, utility functions had to be implemented. A get and a set function allow for more straightforward usage of dictionaries as they can easily break a system and be pretty verbose when extracting values. They work similarly to how Javascript handles objects. A dictionary can be sent into the function. The second parameter is a string that works like a path to find the correct value if it exists. The string contains keys, separated by dots if they are nested dictionaries. If the key does not exist, the program will not break, returning the "None" data type instead. The set function can be used similarly but requires a value to be inserted.

Further stability measures are in place to reduce the chance of the entire system breaking. Try except-blocks are used to have a way to report errors when they happen and where. When the script is terminated, the main thread will send the signal to every other thread and wait for them to finish terminating. Afterward, a message is sent to the server that the Raspberry Pi is shutting down gracefully before terminating the entire program.

One of the program's final inclusions was a thread that would handle navigation. A user could set the system on autopilot with a target position or path that system should try to follow. A further description will be presented in section 4.7.

Figure 33 is a sequence diagram of the Raspberry Pi software running on the boat. The main script called `run.py` is executed to start the program. It starts by initiating sensors, which is done by creating a new object of every sensor class. Each object will then perform the necessary identification and setup of its respective sensor. This step also includes initiating the error handler,

Ngrok server, and navigation object. After waiting for the internet connection, the IP address and authentication token are retrieved. Then a process for each object created in the first step is dispatched in its separate thread. Multi-threading allows each connected sensor to communicate and update independently of the other sensors, resulting in more flexibility and better reliability. If a sensor throws error codes, the thread could be restarted without affecting the other threads.

Next, the program connects to the on-land server using the authentication token, receives STATE from the server, and then enters the main loop. In the main loop, the threaded processes invoke with their objects, read sensor data, calculate navigation values, and send commands. At the same time, data are sent to and from the server.

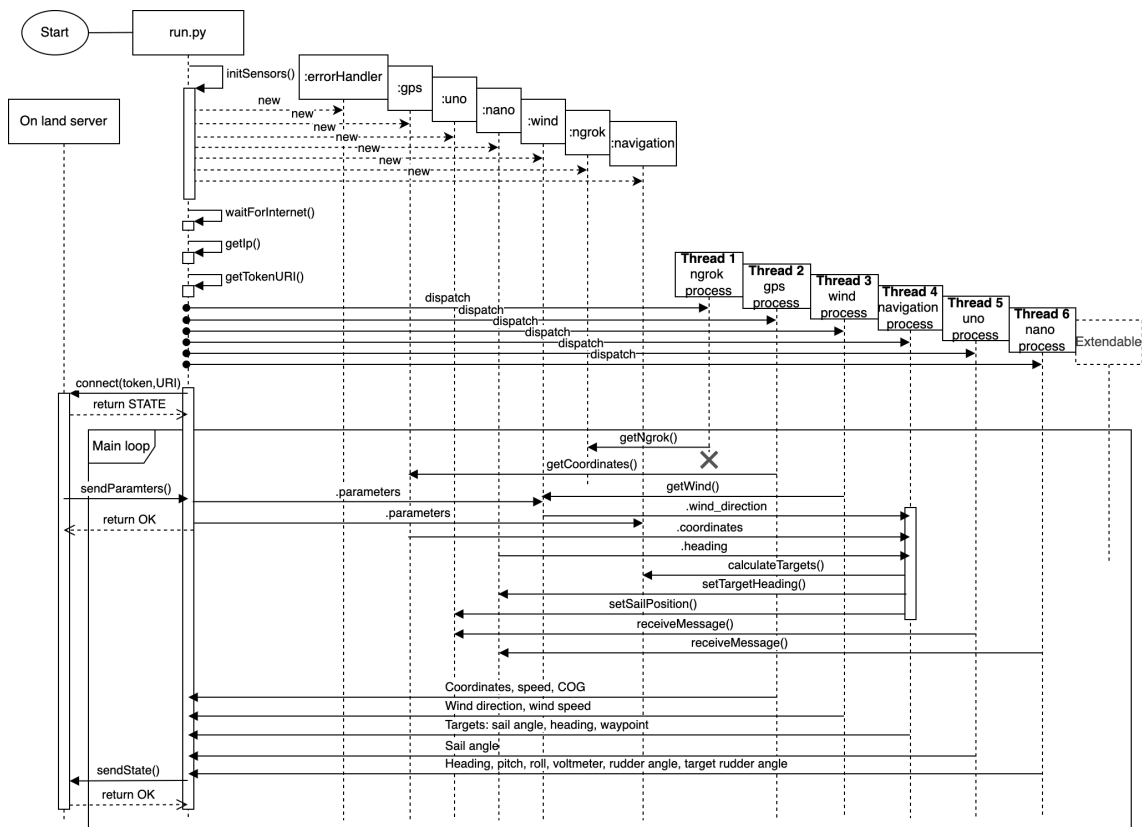


Figure 33: Sequence diagram of sailboat system

#### 4.6.2 Microcontrollers - Arduinos

With the increased complexity in the code on the Arduinos, moving from the traditional Arduino setup was necessary to have a more organized code structure. PlatformIO is an open-source, cross-platform solution for embedded development that includes many of the same features as Arduino but extends upon them. One of the features allows for remote updates to the Arduinos. The Raspberry Pi compiles the programs and uploads the new builds to the corresponding Arduinos. Additionally, splitting code into multiple files ensured better decoupling, which has been beneficial during the development. Modules could now be moved between Arduinos with little to no configuration as code for each component is separated into files.

Every component with its individual files resulted in easier maintenance, configuration, and debugging. As the structure between the component classes was similar, some code could be reused and have a throughout coherence. For example, reading data, changing parameters, and executing



---

## 4.7 Sailing algorithm

When creating a sail algorithm, some limitations must be in place because the algorithm's complexity could be endless. The minimum requirements were set to be:

- Calculate true wind speed and direction
- Calculate distance between current and target position
- Calculate bearing
- Recognize the no sail zone
- Calculate correct sail angle according to AoA
- Efficient downwind sailing called drag mode.
- Solution for upwind sailing

### 4.7.1 True wind calculations

As mentioned in the literature study, there is a difference between apparent wind and true wind. In order to calculate the true wind, certain variables must be known, including apparent wind speed and direction and the boat's velocity. The following variables are  $A$ : apparent wind speed,  $V$ : Boat's velocity, and  $\beta$ : Apparent wind angle.  $W$ : True wind speed, found using the formula:

$$W = \sqrt{A^2 + V^2 - 2AV \cos \beta} \quad (16)$$

Finally, to calculate the true wind direction,  $\alpha$ , with the equation:

$$\alpha = \arccos\left(\frac{A \cos \beta - V}{W}\right) + \lambda \bmod 2\pi \quad (17)$$

where  $\lambda$  is the heading of the boat in radians.

### 4.7.2 Bearing and the no-sail zone

In order to calculate the boat's bearing,  $\theta$ , two coordinates are required, the boat's position and the target's coordinate.

With direct use of coordinates in decimal degrees, *latitude* :  $\varphi$ , *longitude* :  $\lambda$ . Where  $(\varphi_1, \lambda_1)$  is the boat's location, and  $(\varphi_2, \lambda_2)$  is the target's position:

$$\theta = \arctan\left(\frac{\sin(\lambda_2 - \lambda_1) \cdot \cos(\varphi_2)}{\cos(\varphi_1) \cdot \sin(\varphi_2) - \sin(\varphi_1) \cdot \cos(\varphi_2) \cdot \cos(\lambda_2 - \lambda_1)}\right) \quad (18)$$

where  $\theta \in [-\pi, \pi]$ .

The calculated bearing is the target heading that the rudder aims to reach. As the boat drifts off course, the rudder will compensate to make the boat head towards the target. Since the IMU is

directly connected to the same microcontroller as the rudder, all the computer needs is the target heading to function properly for this algorithm.

To check the bearing is inside the no-sail zone, the following equation calculates a delta,  $\delta$ , which tells how far the bearing is of the true wind direction:

$$\delta = ((\theta_b - \theta_w + \frac{4\pi}{3}) \bmod 2\pi) - \pi \quad (19)$$

where  $\theta_b$  and  $\theta_w \in [0, 2\pi]$  are the bearing and true wind angle respectively.

If the  $\delta \leq \beta$ , where  $\beta$  is an angle from the wind where the boat cannot sail, the boat is inside the no-sail zone. For the algorithm tested in this project, the  $\beta$  was set to  $\frac{\pi}{6}$ , which equals  $30^\circ$ . An illustration of the no-sail zone can be seen as black lines in Figure 36.

### 4.7.3 Haversine distance

In the theory section 2.6.4, a presentation of the Haversine Formula was made. As mentioned, the calculation made by the formula for finding the distance is not correct but precise enough for this algorithm. Therefore, the formula was implemented to calculate the distance between the boat's and target's positions.

### 4.7.4 Optimal sail angle

In order to optimize sailing performance, a theoretical study was done in Section 2.5.4 on fixed wing sails and their optimal angles. As seen in Figure 35 by Dyrseth, an optimal AoA would be between  $6-10^\circ$ . During testing, the AoA were set to  $10^\circ$ .

Using the sail as a drag generating surface when sailing downwind, presented in Section 2.5.3, was implemented. Even though this may be less efficient than beating downwind, the technique was worth exploring since beating would be tested during upwind sailing. The requirement for this mode to be activated was set to  $140^\circ \geq \delta$ , where  $\delta$  is the absolute difference between the bearing and the wind direction. This was to make the window a little smaller than what was used in the study in Section 2.5.3.

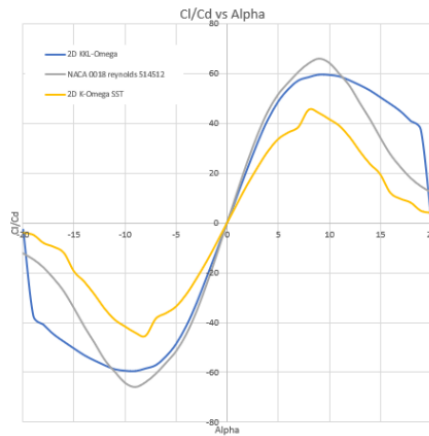


Figure 35:  $\frac{\text{Coefficient of lift}}{\text{Coefficient of drag}}$  vs  $\alpha$ , from article [38]

#### 4.7.5 Midway point and beating for upwind sailing

After calculations of the no-sail zone mentioned in section 4.7.2, a possibility was to begin beating. In order to find a new quick waypoint during the beating, a formula was used. The equation is used to find a new coordinate at a certain distance and angle of the wind based on the  $\delta$  in section 4.7.2.

The new bearing in radians,  $\theta_b$ , to the midway point where found by the following equation:

$$\theta_b = (\theta_w + \frac{\delta}{|\delta|} \cdot \beta + \beta_+) \bmod 2\pi \quad (20)$$

where  $\beta$  is mentioned in section 4.7.2, and  $\beta_+$  is a buffer for this beating limit.

The sign of the  $\delta$  decides which "side" of the no-sail zone would be closest to the target. A buffer,  $\beta_+$ , which in the algorithm was set to  $10^\circ$  was meant to reduce the chance of the boat's movement turning the midway point inside the no-sail zone immediately. An example in Figure 36 illustrates the logic explained above.

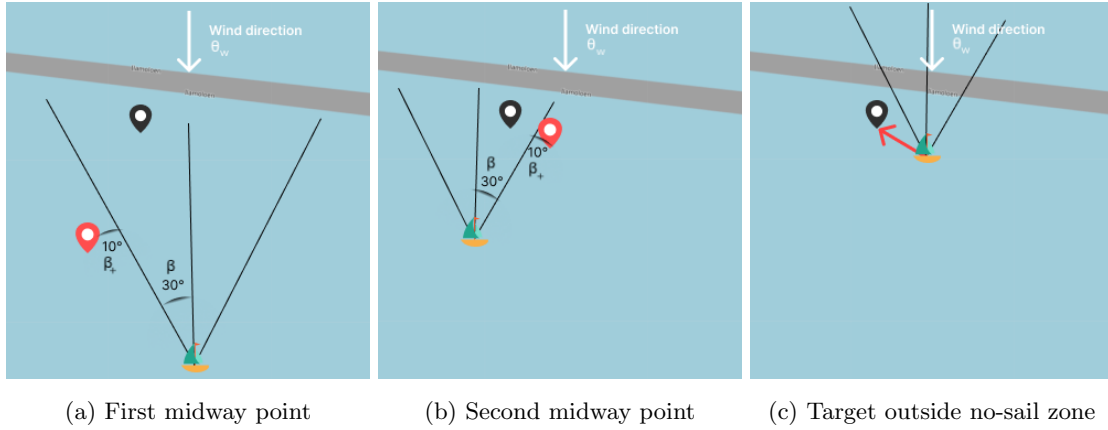


Figure 36: Tacking sequences, black lines indicate edges of the no-sail zone. Red marker is the midway point, and the black marker are the target.  $\beta = 30^\circ$ ,  $\beta_+ = 10^\circ$

To calculate a new coordinate, given a the new bearing  $\theta_b$  in radians, distance  $d$  and a start coordinate (*latitude, longitude*)( $\varphi_s, \lambda_s$ ):

New latitude  $\varphi$  :

$$\varphi = \arcsin(\sin(\varphi_s) \cdot \cos(\frac{d}{r}) + \cos(\varphi_s) \cdot \sin(\frac{d}{r}) \cdot \cos(\theta)) \quad (21)$$

New longitude  $\lambda$  :

$$\lambda = \lambda_s + \arctan\left(\frac{\sin(\theta) \cdot \sin(\frac{d}{r}) \cdot \cos(\varphi_s)}{\cos(\frac{d}{r}) - \sin(\varphi_s) \cdot \sin(\varphi)}\right) \quad (22)$$

where the new coordinates are in decimal radians.

---

#### 4.7.6 Adjustable parameters

In preparation for testing, adjustable parameters were added alongside default values. Due to the difficulty of performing a field test of the algorithm, it was deemed crucial to be able to test numerous configurations during testing.

For the sail algorithm, the following parameters were adjustable:

- Update frequency of the algorithm
- AoA on the sail
- Drag mode activation limit
- Size of no-sail zone,  $\beta$
- Beating buffer,  $\beta_+$
- Midway point's distance from boat's location

#### 4.8 Security through authentication

With the server publicly on Google Cloud Platform, more security was necessary to have a sufficient level of protection. A proposed System design by the WebSockets library documentation[70] in Python allowed for improved security when a WebSocket server is publicly available. Firstly, an authentication solution for users needed to be established. With the integration of Google and Firebase as hosting and database services, it was pretty natural to introduce Firebase's authentication service.

A simple login page was integrated as a startup page for the dashboard. Authentication could be possible with a pre-registered user. However, it would only give the user access to the website's features and no data. The user could not receive and visualize data until a WebSocket connection was established with the server. In order to allow access to the server, a generated JSON web token (JWT) delivered by Firebase must be sent first to the server to check its validity against Firebase. If the server could recognize a user ID registered in the current Google project, authentication would be successful, and a connection would be established.

The Raspberry Pi also needs to authenticate itself to Firebase and the server to set up a connection. A pre-registered user must be created in Firebase Authentication for the Raspberry Pi to be able to log in. As mentioned in 4.6.1, a shell script can be executed to fill in every environmental variable required to connect to the server.

#### 4.9 Integrating whole system

Figure 37 illustrates the integration of sailboat electronics. The link between components used and the interface is marked with arrows and text. Together with the boat's hardware like sail, rudder, and hull, a mechatronic system capable of autonomous sailing is presented.

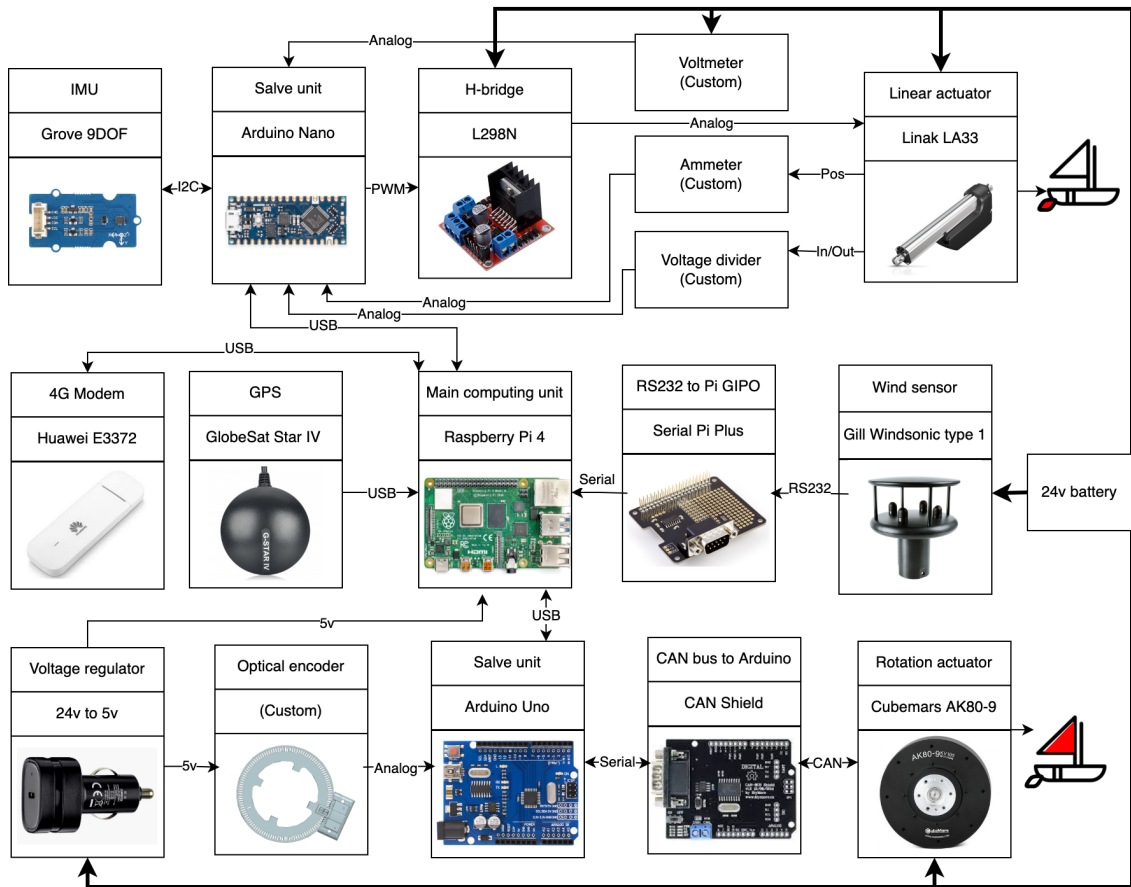


Figure 37: Hardware architecture diagram

Figure 38 shows the distributed system's software architecture. The sailboat communicates with the cloud server. Using a dashboard connected to the same server, an operator can observe, control and review data from the sailboat. Table 14 lists the number of code lines written, divided into each system. They are indications of the amount of software development needed in order to achieve comparable levels of features.

Table 14: Lines of code across system

System	Lines of code
On land server	414
Dashboard	4022
Raspberry Pi	1398
Arduino Uno	754
Arduino Nano	1011
<b>Total</b>	<b>7599</b>

*Table does not include libraries*

In total, this distributed system is designed to enable the operation of autonomous sailboats, which will be put to the test in the next chapter.



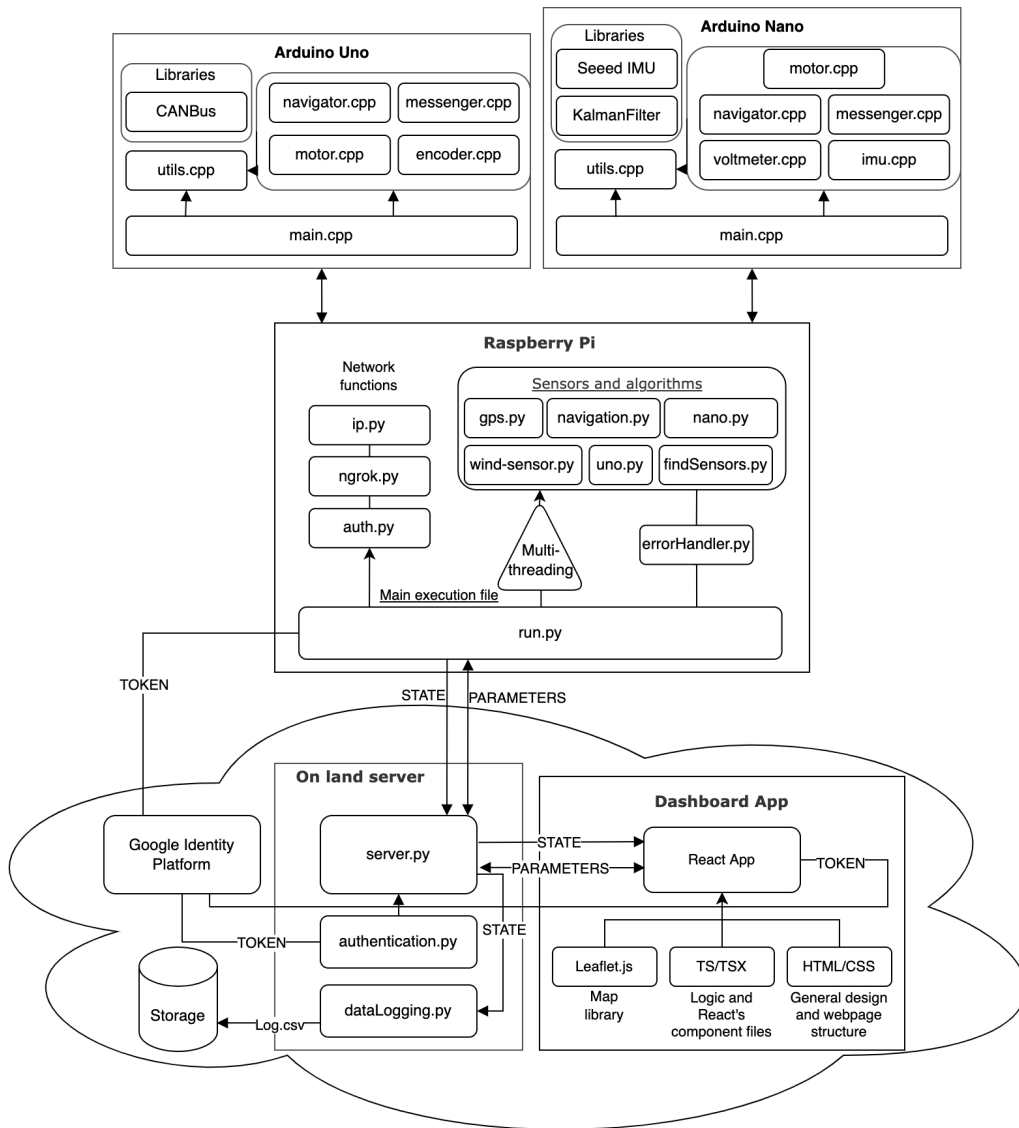


Figure 38: Software architecture diagram

f

---

## 5 Tests and results

Following the last step in Section 3 a system validation test should be performed and results reviewed. This section will present an extensive sea trial on the system from Section 4 and is an overarching test throughout the section. Further, separated parts of the system will be analyzed and discussed in detail.

First, this section will outline the on-land server's performance and evaluate interoperability between the sailboat's computers. Afterward, it will review acceptance tests on the client application to see whether it accomplishes the functional requirements in Section 3.2.2. Then, it presents system unit tests of the sail, rudder, and noise filters, and outcomes are studied. After that, the algorithm's performance during different tasks at the sea trial is reviewed. Lastly, the integrated system's potency and test results are summarized.

### 5.1 On sea trial

On April 26th, a sea trial was conducted at Skansen guest harbor in Trondheim from 10 am until 4 pm. The boat experienced apparent wind speeds ranging from  $0.5m/s$  to  $11.44m/s$ , with an average of  $4.26m/s$ , and temperatures below  $5^{\circ}C$ . It also included two hail bursts and strong wind gusts. Figure 39 are the recorded wind speed by the sailboat for the entire test. In the first part, the wind speed was decreasing and almost disappearing, making it difficult to sail. It was followed by strong gusts of wind, halting the test for a brief period. The last part of the test involved a more stable and optimal wind speed for performing tests.

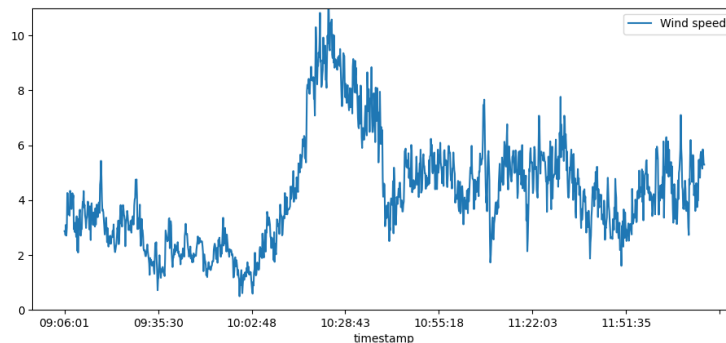


Figure 39: Wind speeds recorded by the sailboat during the entire test

The sailboat was transported with a private car and the university's trailer from NTNU's building, Verkstedteknisk.



Figure 40: Sailboat on the trailer

Figure 41 shows for the first time all the equipment mounted on the catamaran at the same time.



Figure 41: In-progress assemble of the sailboat

The sea trial was a milestone for the project and essential to validating the system. The only way to perform a full system test was to deploy it outside in water and wind. Such a test was necessary to gather valuable data and experience.

### 5.1.1 Experiment set up

The boat was transported to the harbor and assembled on the dock. As seen in Figure 41, in order to mount the keel, a good technique was to balance the boat on its side after mounting the sail.

---

The boat as a whole is quite challenging to put together, and it is recommended to be at least three people.

With everything mounted, the boat was carried to the edge and carefully lifted to an upright position over the water. Two ropes tied to each end were used to control the descent into the water from the dock. When seaborne, the electronic box was securely mounted and all cables connected.

A small motorized support boat provided by co-supervisor Dr. Echtermeyer was used to move the sailboat when necessary.

## 5.2 Server and communication

The following subsections analyze the cloud server’s performance and interaction between components in the system. Data about the state of the cloud server are collected from the Google Cloud Platform, while data on interoperability of the boat’s computers are derived from sensor logs.

### 5.2.1 On land server

From Figure 42, the blue line indicates active instances, and the green indicates idle instances. As the test was done throughout most of the day, the server instance was in an active state until around 3 pm.

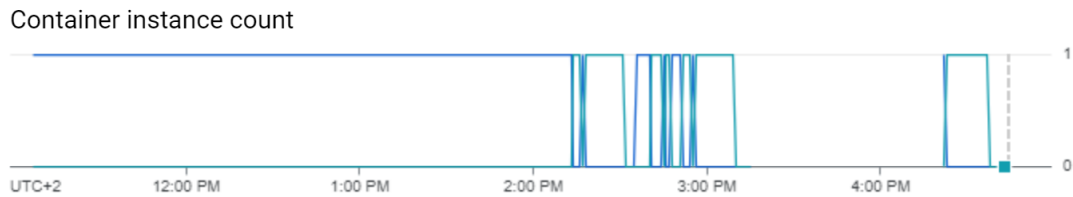


Figure 42: Container instances

A request occurs when a client or the boat tries to connect to the server. After connection, any consequential data transmission through a WebSocket connection does not count as a request. However, reconnects do. The request count during the sea trial can be seen in Figure 43. The monitor calculated a maximum of three requests each minute ( $0.05/s \cdot 60$ ).



Figure 43: Request count

The cloud server treats a WebSocket connection as a long-running HTTP request, hence the reason for the long request latencies reported in Figure 44. In the figure, a small red line indicates that the longest request latency was 8.69 minutes. As a request timeout kicks in, a reconnection happens

---

every five minutes, but it can be adjusted up to 60 minutes. Google recommends that a connection is not open longer than 15 minutes before initiating a reconnect.

#### Request latencies

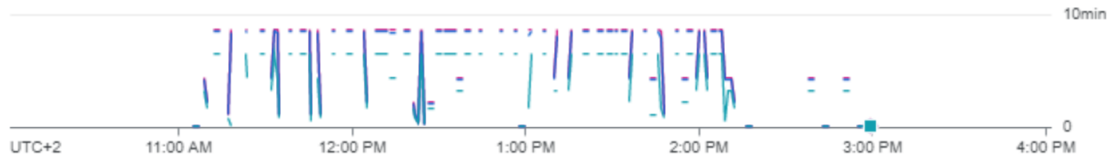


Figure 44: Request latencies

As seen in Table 8, the instances use a 1st generation CPU provided by Google. These CPUs offer 1 GHz of processing speed, which according to Figure 45 is more than enough. At its peak, the CPU only utilized 6% of its available resources.

#### Container CPU utilization



Figure 45: CPU utilization

Memory usage did not exceed its limit of 256MiB. At most, the container used 38% of available memory. However, as seen in Figure 46, the usage increases from the beginning of the test until about 2:30 pm. The drop might be due to a break in the test, which allowed the server to terminate and reset. As for the build-up, one reason may be a local log file for the server connection, which keeps track of everyone that tries to establish a connection.

#### Container memory utilization



Figure 46: Memory utilization

For the extensive usage of Google Cloud Platform during the test, the approximate cost of their services was 2.51 NOK. Figure 47 shows a blue line indicating the number of instances per second that were billed during the sea trial.

#### Billable container instance time



Figure 47: Billable container time

---

### 5.2.2 Interoperability between Raspberry Pi and the Arduinos

No problems were observed on the whole sea test regarding the MCU. It never crashed, had no connection problems, and was overall very reliable. Authentication was tested with and without a security token, and the Raspberry Pi was correctly allowed and prohibited from connecting, respectively.

There were, however, some challenges with one of the Arduinos. The one responsible for reading the IMU and controlling the rudder proved quite vulnerable. One of the reasons may be the unforeseen types of data provided by the Raspberry Pi that the Arduino did not know how to process, resulting in unnotified crashes. A temporary solution to this problem was to reset the Arduino remotely through the Raspberry Pi, providing enough operating time to test several aspects of the system. However, from Table 15, the impact of the downtime was quite severe during the test. 25% downtime made some of the data collected hard to analyze, as the sailboat sometimes executed tasks with inaccurate data.

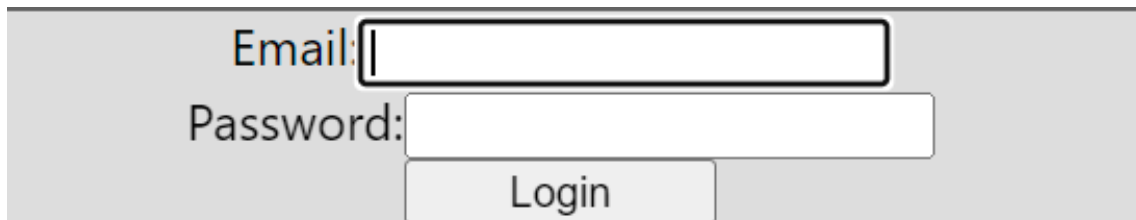
Table 15: Downtime on the Arduino Nano

Downtime measurements	
Shortest	14s
Longest	834s
Average	230s
Occurrences	12
Total downtime	25%
Total downtime (seconds)	2764s
Total uptime	75%
Total uptime (seconds)	8330s

### 5.3 Acceptance test of the client-side application

A client-side application was created to help the operators during the development and testing of the boat. This subsection will go through all the functional requirements from Section 3.2.2 and their current status. Furthermore, it will review how the dashboard performed in the on-sea trial.

#### 5.3.1 Authentication requirements



The image shows a login form on a light gray background. It consists of three main elements: a label 'Email:' followed by a white rectangular input field with a black border; a label 'Password:' followed by a white rectangular input field with a black border; and a 'Login' button below the password field, which is a light gray rectangle with rounded corners and the text 'Login' in a dark gray font.

Figure 48: Authentication form

---

Following Table 2, FRA1 is implemented. It follows a typical login sequence as seen in Figure 48. However, neither FRA2 nor FRA3 are completed as functionalities. Finalizing these requirements may not take much time, but it was not prioritized during the development. FRA3 became an unnecessary functionality as project members are the only users that need access to the service. The security was increased further by only allowing pre-registered accounts through Firebase Admin to log in.

### 5.3.2 Map requirements

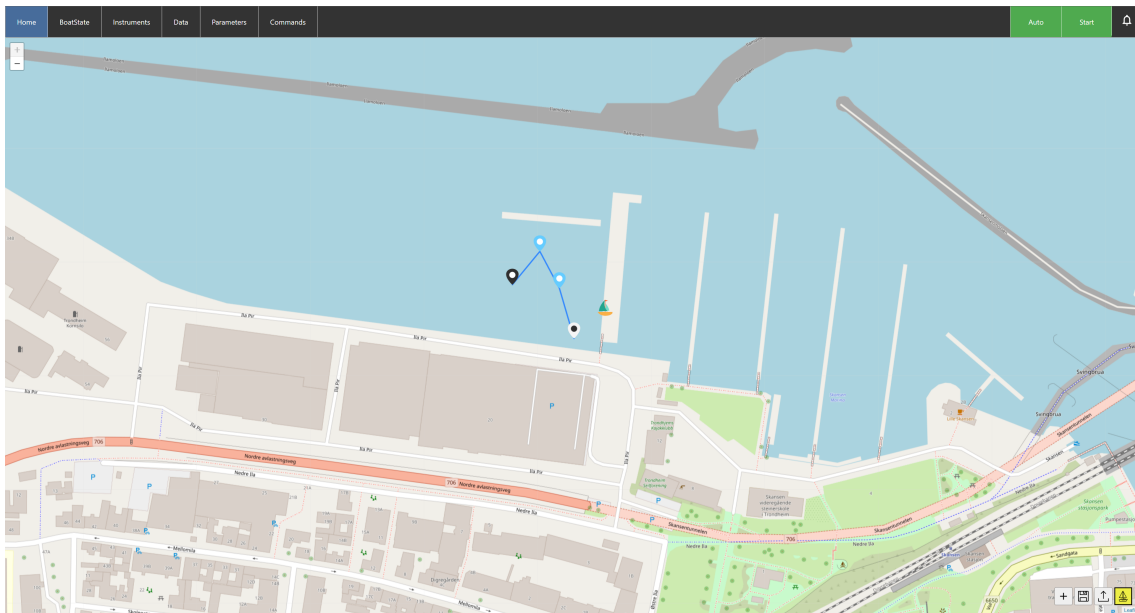


Figure 49: Map with a path

Map functionalities from Table 3 have mostly been implemented in the dashboard. FRM1 is complete as seen in Figure 49. The position is visible on the map as a sailboat icon and is updated in real-time as the location changes. Additionally, the targets can be seen as markers on the map; see Figure 49. A white marker indicates the first waypoint of a path, and a black is the last. Blue markers are intermediate waypoints along the path. As each waypoint is reached, the marker changes color to green. If a midway point is created, it is a red marker as seen in Figure 77, thus also completing FRM2.

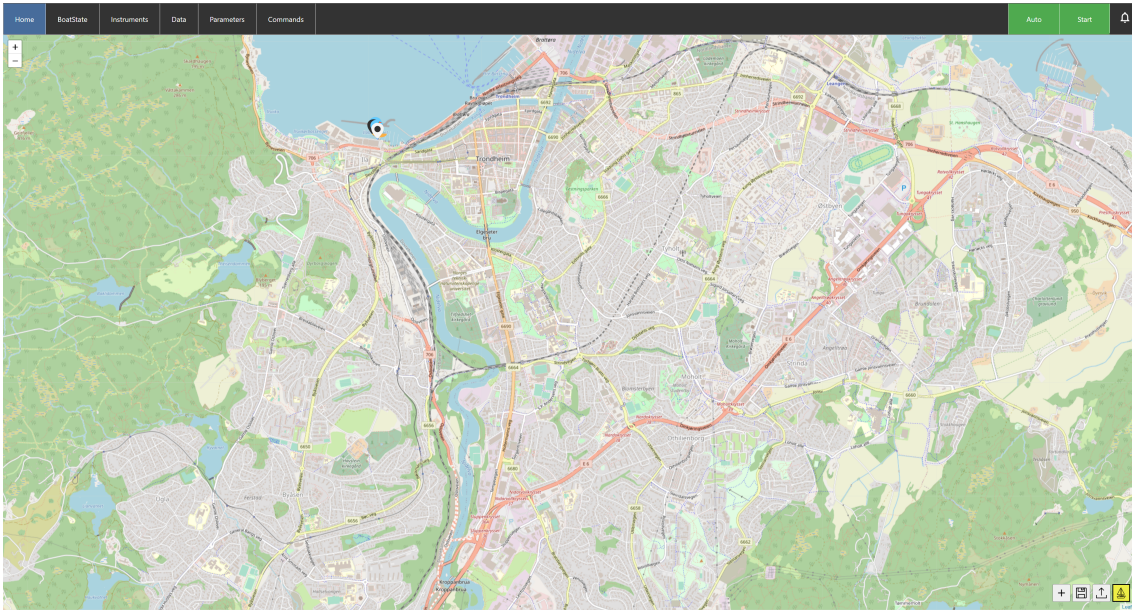


Figure 50: Homepage with the map

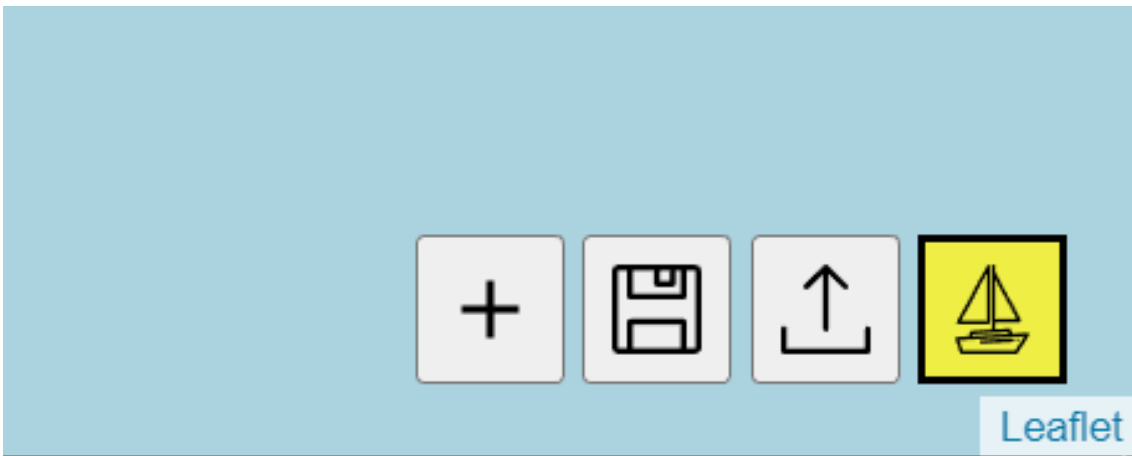


Figure 51: From left to right: (1) New path, (2) download current path, (3) upload a path file, (4) edit boat's current path.

The buttons visible in the bottom right corner of Figure 50 are used for path editing. Figure 51 shows a closeup view with a caption explaining what each button does. The rightmost button, button (4), offers the functionality required from FRM4. Further, FRM5 and FRM6 are completed by the functionality of buttons (2) and (3), respectively.



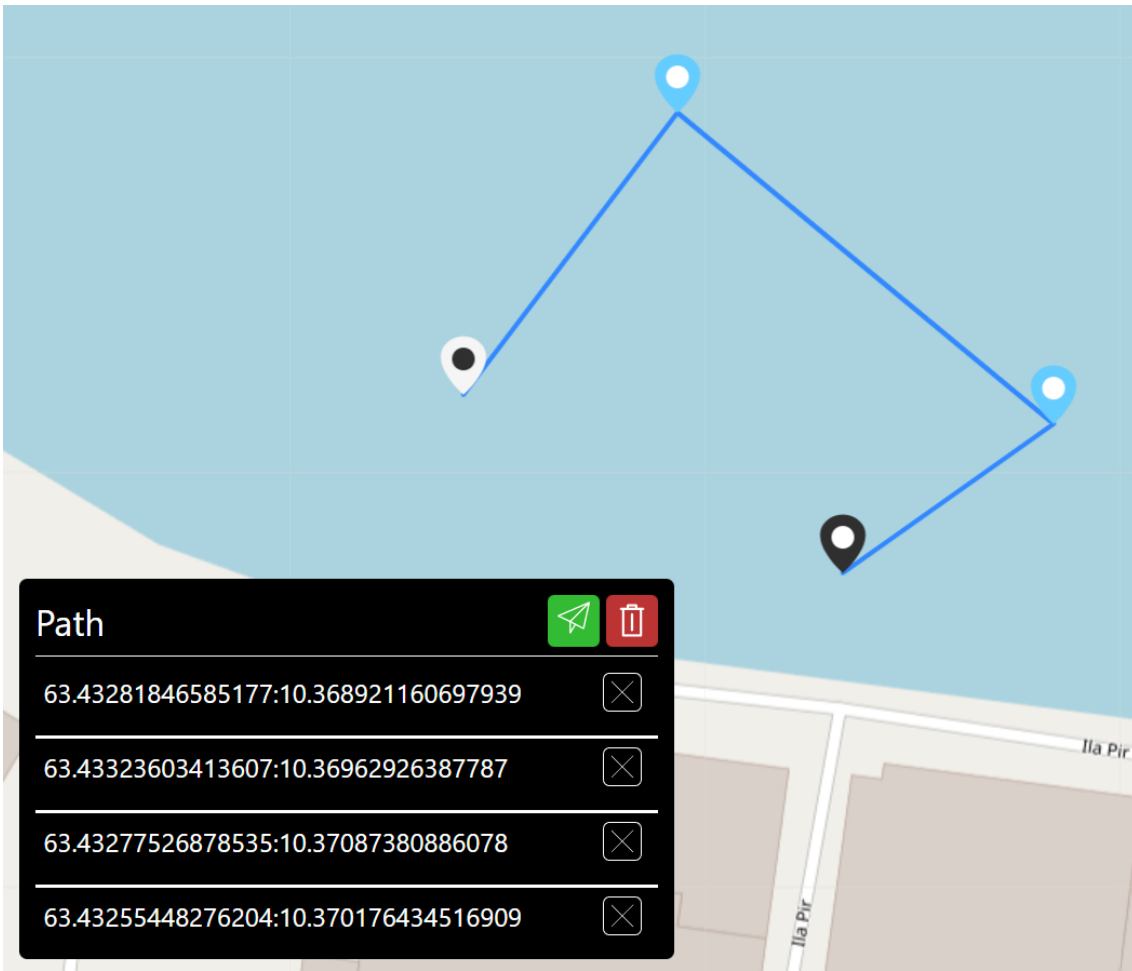


Figure 52: Path planner with a list of waypoints

New markers are placed when clicking on the map while in editing mode. Further, a black box at the bottom left corner will be visible, as seen in Figure 52. The box includes all the waypoints in the currently edited path. They ordered from the first to the last waypoint of the path. The list elements are draggable to change the order of the waypoints to the path easily. To the right of each element is a button that removes the corresponding waypoint. The top right buttons inside the black box, see Figure 53, allow the user to send the boat the new path and delete the edited path. Thus FRM3, FRM7, and FRM8 are finalized.

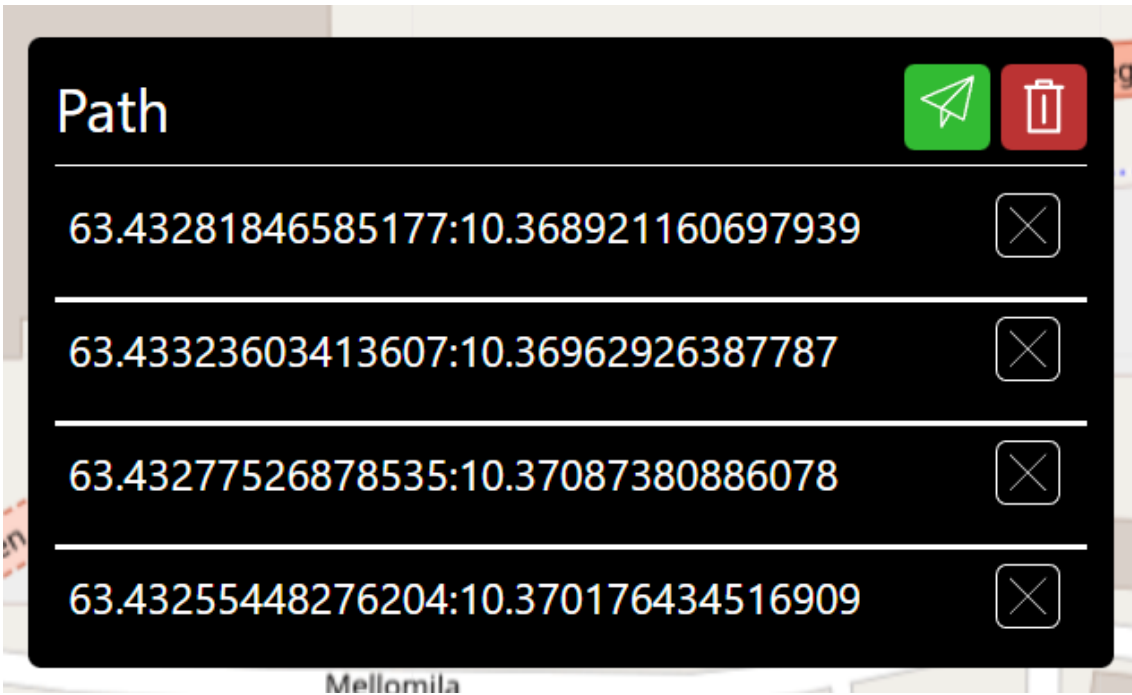


Figure 53: Path coordinates in order

### 5.3.3 Data visualization requirements

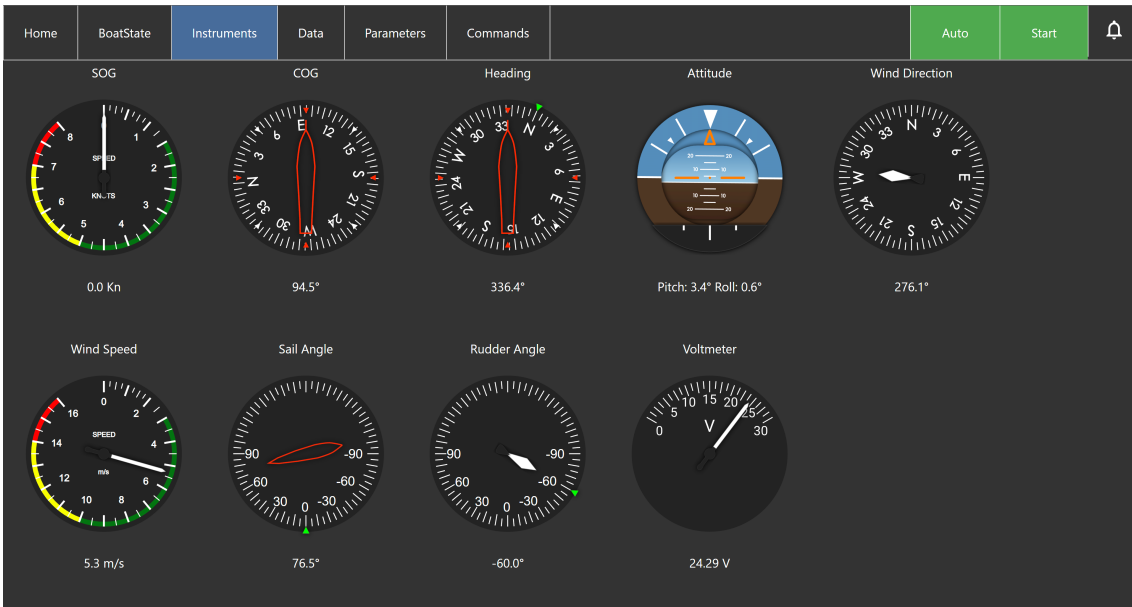


Figure 54: Instrument page

Next is the data visualization page. From Table 4, FRD1 works as seen in Figure 54 where the dials shows data from the sailboat. The green indicator on the instruments for heading, sail, and rudder angle is the target for each corresponding component, which was part of the requirement for FRD3. Lastly, FRD2 is considered complete with the usage of common instruments from boat and airplane panels.

### 5.3.4 Command system requirements

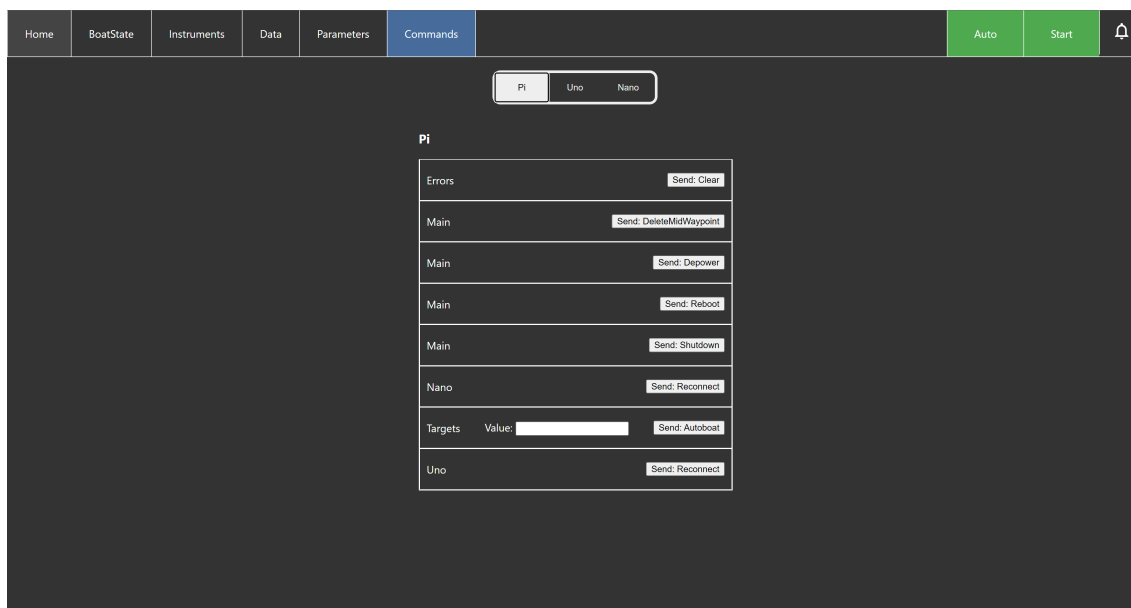


Figure 55: Command page

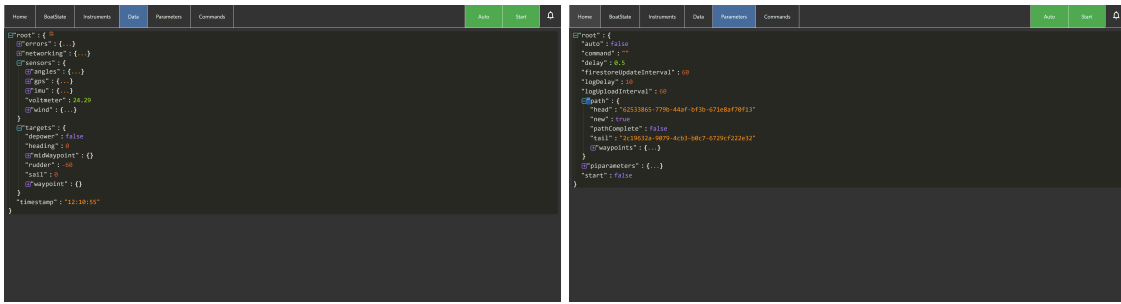
Figure 55 shows the page with the commands available. In order to satisfy requirement FRC1 from Table 5, available commands for each computer or controller on the boat are listed. When a computer is selected in the tab menu, commands for the corresponding computer are listed. The leftmost text of an element in the list shows the component's name to which the command belongs. The name of the command is inside the button for sending it. If the command requires inputs, a text field with a label is visible left of the text field. Lastly, as seen in Figure 56, a description is available when hovering over the list element. Hence, FRC1, FRC2, FRC3, and FRC4 have successfully been implemented. A complete list of commands can be seen in Section B.



Figure 56: Command description

### 5.3.5 Data and parameter requirements

FRDP1 from Table 6 is complete since visualisation of the data and parameters are implemented, as can be seen in Figure 57. In the data page, Figure 57a, the data is not editable, but they are updated in real-time. However, the parameters page, Figure 57b, allows for editing, except for changing the datatype. Doing so will result in an error message being displayed to the user. Therefore, both FRDP2 and FRDP3 are completed.



(a) Data page

(b) Parameters page

Figure 57: Pages visualising data and parameters

### 5.3.6 Notification system requirements

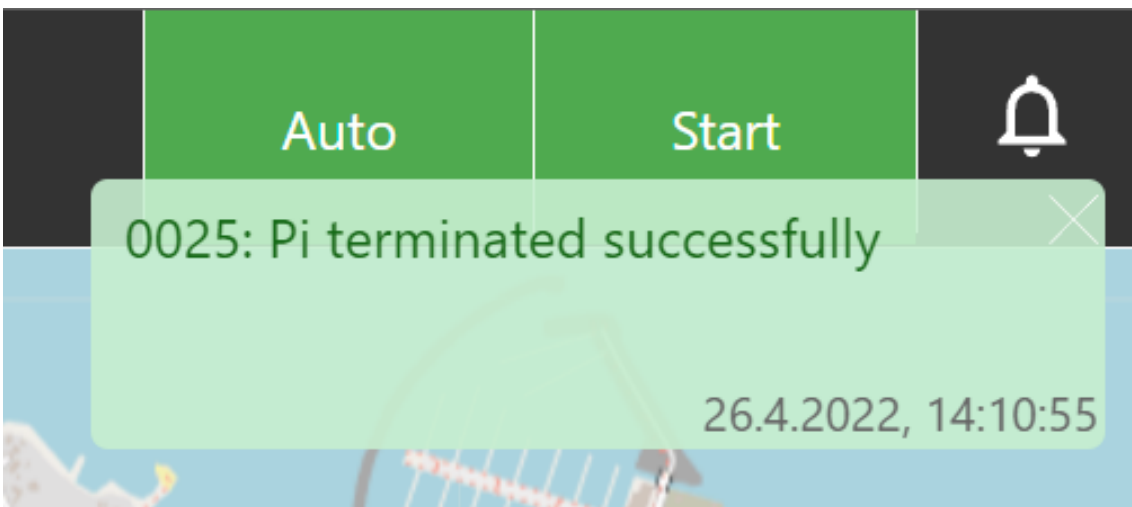


Figure 58: Notification message

When a notification pops up, it will slide into the screen from the upper right corner, as seen in Figure 58. The popup message includes a code number, description, and the date and timestamp at the bottom right corner. These functionalities were the requirements FRN1, FRN2, and FRN3.

As seen in Figure 59, a notification center is implemented, completing FRN4. Additionally, at the top of the notification center, a toggle switch is available for turning on and off the popup messages, the last functional requirement in Table 7.

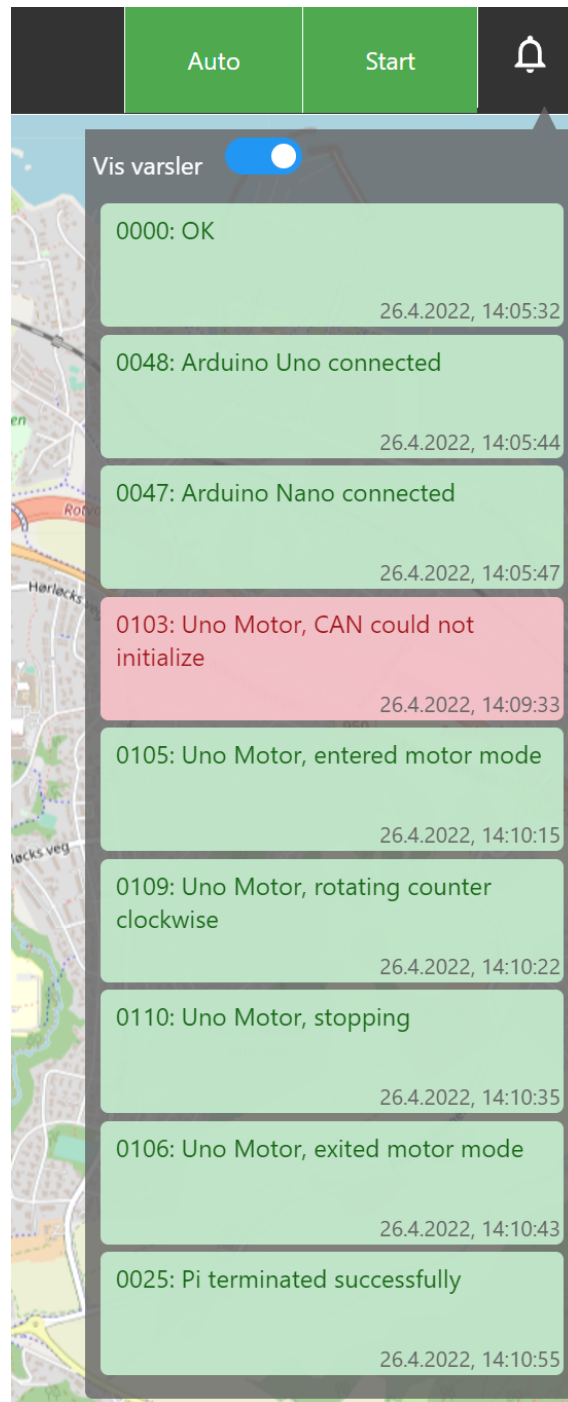


Figure 59: Notification center

## 5.4 System unit tests

Unit tests in controlled environments were conducted for both the sail and rudder system and are presented in the coming sections. Following is an analysis of their performance during the sea trial. Lastly, an investigation of the noise filters' performance is reviewed.

---

### 5.4.1 System test of the sail

#### Test introduction

The importance of a functional sail system was vital. For testing the necessary functions, a list of questions that needed to be answered was as follows:

- Does the sail rotate adequately?
- Can the encoder report the correct position?
- Is the z-index identifiable?
- Does the actuator and encoder cooperate to indicate the sail position?
- Can the sail move to the target position?

#### Experiment set up

During controlled laboratory testing, a portion of the mast was mounted to a heavy base plate to prevent the mast from falling over. The portion of the mast included the worm gear, rotary actuator, encoder, and the bottom plate of the sail, see Figure 60.

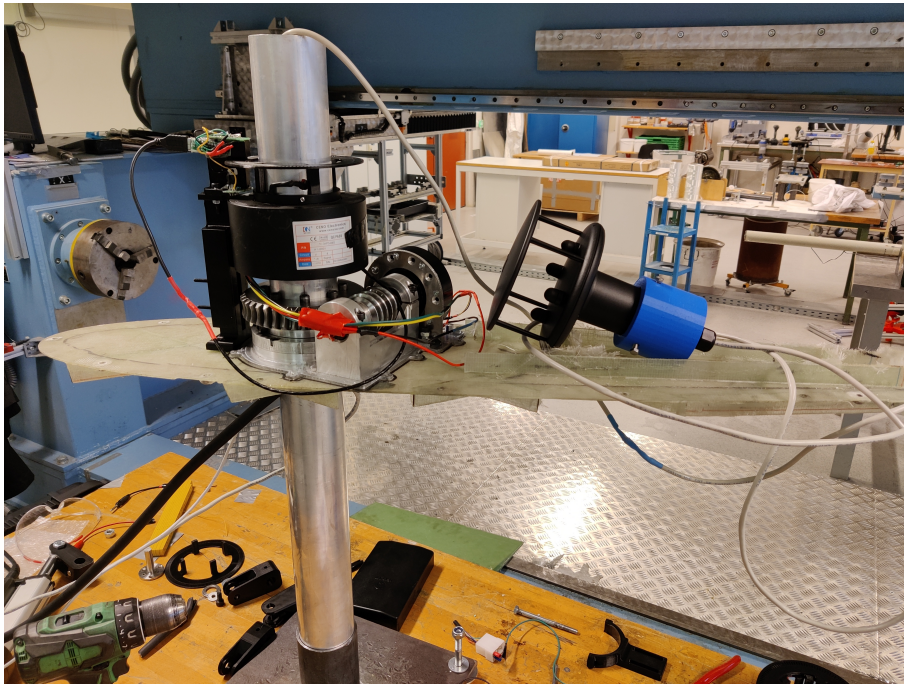


Figure 60: Set up of the sail control system

#### Results

The actuator is able to rotate the sail clockwise and counterclockwise without issues. A full rotation is achieved in 56 seconds resulting in  $6.43^\circ/s$  with no load. The encoder consistently changes and reports the position of the sail, and the accuracy is verified using a few known positions around its rotation. The z-index, or the reference point, was found during every test.

Cooperation from the motor to whether the encoder value should increment or decrement during changes showed satisfactory results. Depending on the motor's rotational orientation, the value incremented or decremented accordingly.

---

When the navigation program for the sail receives a target position, the system adjusts the sail to the correct position with a precision of  $1.5^\circ$ , limited by the optical rotary encoder. When calibration mode is initiated, the sail rotates to find the reference point and updates values.

### Sea trial results

Figure 61 shows the target sail angle and actual sail angle in the same plot. As new targets are received, the sail adjusts to reach the target. The plot shows that the sail can mostly keep up with the target. The average deviation was calculated to be approximately  $1.576^\circ$ , most likely caused by the  $1.5^\circ$  precision of the encoder. The accuracy should not be an issue with optimal AoA ranging from  $6 - 11^\circ$ .

An issue was discovered when the boat was moored during strong winds. It was observed from the dashboard instruments that the reported sail angle increased, despite the sail not moving in reality. The most likely cause is that some slack in the gearbox allows the sail to rock back and forth in strong wind. This movement is enough for the optical encoder to believe it is rotating. A simple solution to this issue could be to turn off the optical encoder when the actuator is not rotating.

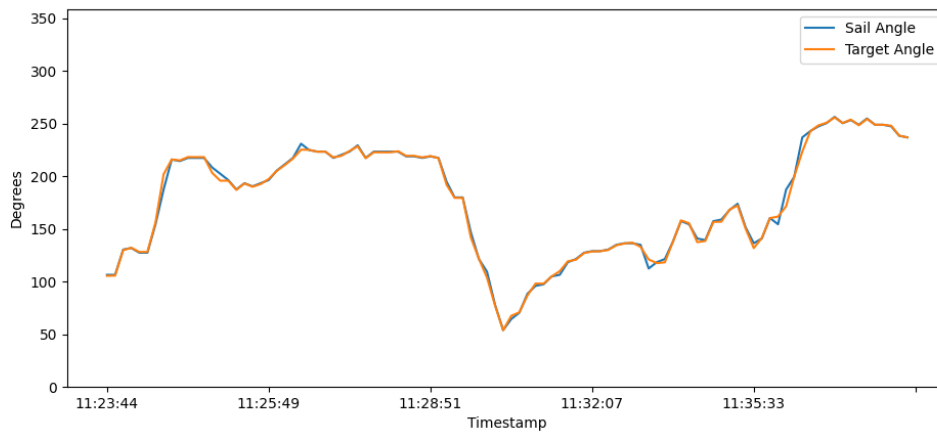


Figure 61: Target sail angle, and actual sail angle (adjusted  $180^\circ$  to improve readability)

Figure 62 compares the sail angle to the apparent wind direction. Depending on the apparent wind being  $w \in [0^\circ, 180^\circ]$  or  $w \in (180^\circ, 359^\circ]$ , the system turns the starboard or port side of the sail into the wind, respectively. The sail turns into the wind with an AoA of  $10^\circ$ , as can be seen by the deviation between the two graphs in Figure 62.

On average, the offset of the sail angle compared to the wind direction was calculated to be approximately  $9.898^\circ$  in Figure 62.

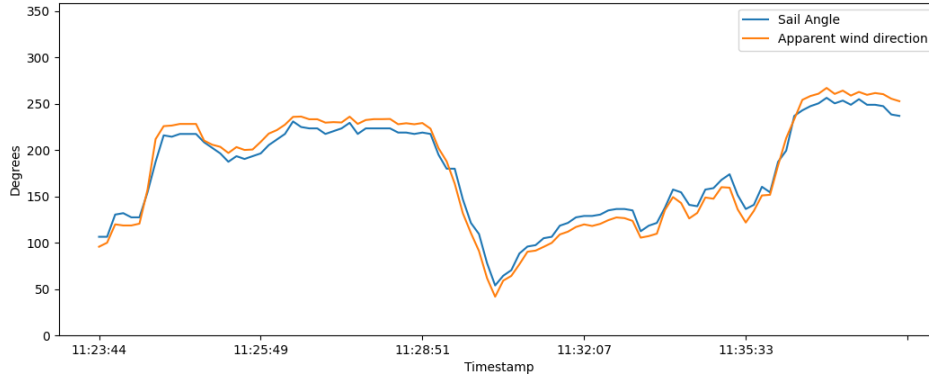


Figure 62: Sail angle and apparent wind direction (adjusted 180° to improve readability)

## 5.4.2 System test of the rudder

### Test introduction

As mentioned earlier, the rudder is responsible for changing the boat's heading. The list that follows are quality signs that were desired to map the current state of the rudder system:

- Does the rudder move sufficiently fast?
- What level of precision can be obtained?
- Does the rudder construction offer the estimated range of motion of  $\pm 60^\circ$ ?

### Experiment set up

Several controlled rudder tests were conducted on the workbench during the development. Figure 63 is from a range of motion test where a protractor was printed out on a piece of paper and placed at the center of the rudder to verify its position.



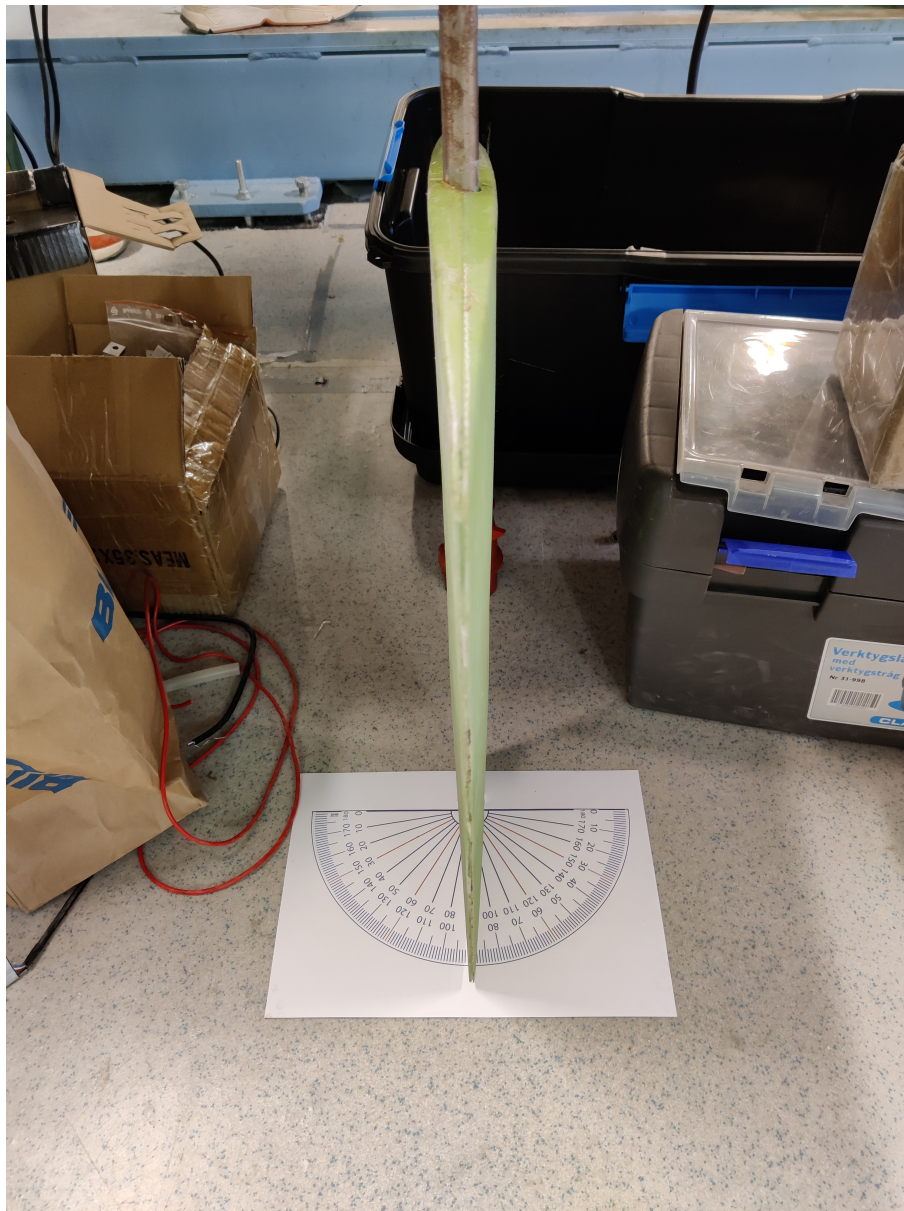


Figure 63: Set up of the rudder and the actuator

## Results

After several tests, it was determined that the rudder moves at approximately  $24.5^\circ/s$  and reaches a range of motion of  $120^\circ$ . Total execution time from one end to the other takes around 4.8 seconds. Considering the speed of the sailboat, the speed of the rudder was not a concern. The protractor also verified the accuracy of the position feedback, which was higher than the resolution of the test setup. Despite getting precise position feedback from the actuator, the implemented rudder actuation mechanism had an accuracy of  $8^\circ$ . During initial testing, a behavior was discovered where the actuator needed time to accelerate from a standstill and decelerate when moving. The acceleration phase increased the lowest possible duration of a move signal the Arduino could send and still observe a change in actuator position. After the move signal ended, the actuator needed time to decelerate, which extended the distance traveled. The result was a constant of  $8^\circ$  added to every position change of the actuator.

---

## Sea trial result

As seen in Figure 64, the rudder angle follows the target angle decently well. The average deviation between the target and set rudder angles was approximately  $4.33^\circ$ . Since the rudder moves toward the target angle when the deviation exceeds  $8^\circ$ , and the new rudder angle stops close to the target angle, the calculated average seems to hold with that logic.

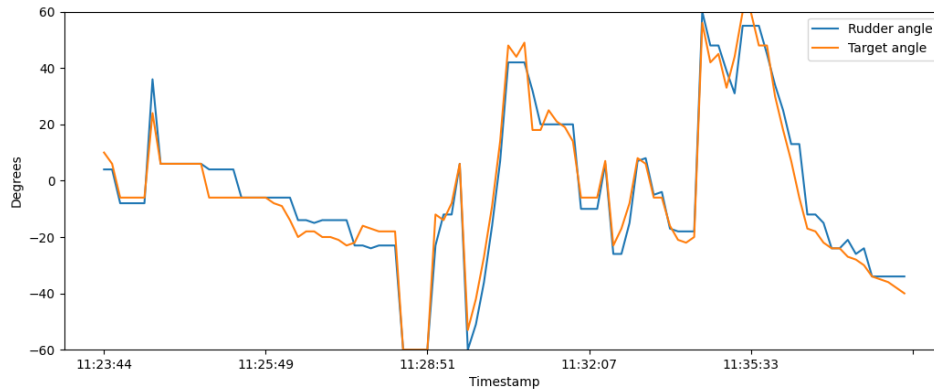


Figure 64: Rudder and target angle

### 5.4.3 Noise filter test

#### Test introduction

- Does the IMU filter smooth out the data without removing important information?
- Does the wind sensor filter smooth out the data without removing important information?
- Does data from the two sensors correlate, indicating their accurateness?

#### Experiment set up

The test of the wind sensor was carried out during the sea trial, exposing the sensors to real-world conditions. A 15-minute interval from the sea trial was selected based on a fluctuating but non-zero wind speed and all sensors communicating as expected. Y-axis values are adjusted to prevent the graph from going from  $359^\circ$  to  $0^\circ$  when the boat rotates passed north. The scale is still the same, but the heading is inaccurate with respect to the real world.

#### Results

Figure 65 shows the wind speed in the given time window. The EWMA wind speed filter has a  $B = 0.5$  meaning new values dominate the mean. The result is a graph with a high variance but, at the same time, more true to the varying wind conditions. The wind speed is not used in the sailing algorithm, so this is not a concern. It shows that the wind speed in this time window was usually between 3 and 7  $m/s$ , which are good conditions for the sailboat.

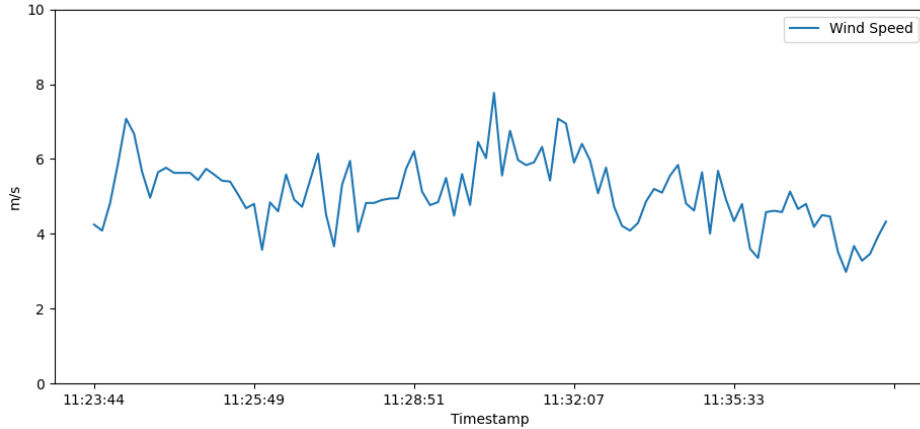


Figure 65: Wind speed

Figure 66 is a plot of the apparent wind direction reported by the wind sensor and the boat's heading reported by the IMU. The two graphs should mirror each other about the horizontal axis, assuming a constant actual wind direction. If the boat rotates clockwise, the heading will increase while the apparent wind angle will decrease, and vice versa. The graphs in Figure 66 are close to being a mirror of each other, indicating accurate sensor readings. The slight variations might be due to changes in the wind. The two graphs do not appear to be shifted in time, indicating that the polling rate and delay introduced by the two different filters are equal among the sensors. It does not, however, measure the total delay between the real world and the values provided by the sensors.

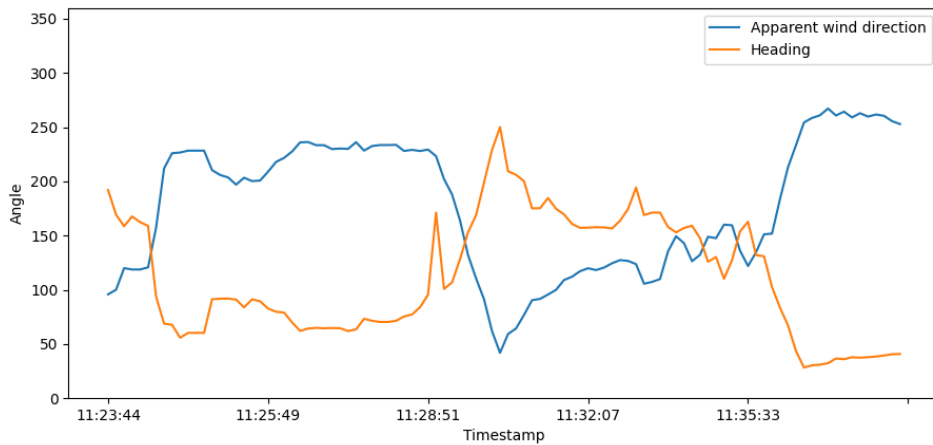


Figure 66: Apparent wind direction and heading (adjusted 180° to improve readability)

Figure 67 is a plot of the true wind direction, which is the apparent wind direction adjusted for the boat's velocity and heading. It shows a quite consistent wind direction of around 100° with some local variations. The slight variation is expected as the wind direction in an area is close to constant but with deviations caused by wind gusts, nearby buildings, and other obstructions. The only exception is the spike at 11.28.51 AM, caused by a jump in heading seen in Figure 66. It is clear from the heading graph that the spike should not be there and indicates that the IMU

---

Kalman Filter is not correctly adjusted.

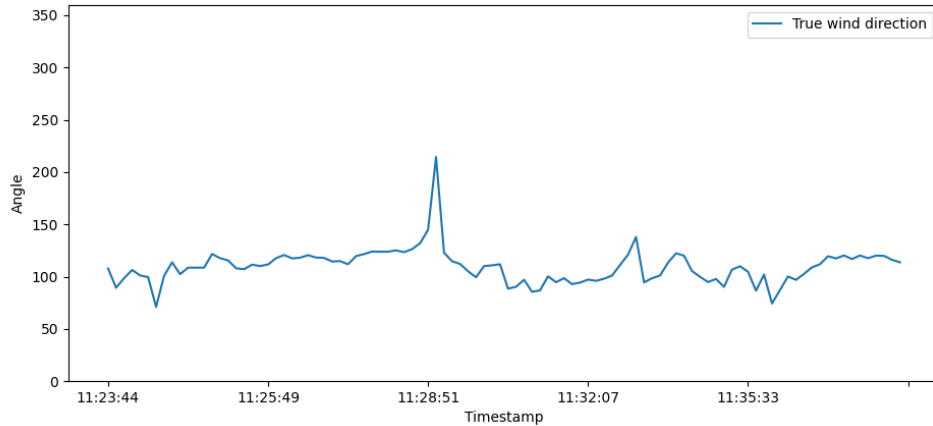


Figure 67: True wind direction (adjusted 180° to improve readability)

## 5.5 Algorithm performance test

During the sea trial, the boat's autonomy was tested close to real situations. The following sections will review the algorithm's capabilities to solve well-known tasks for an autonomous sailboat, such as beam reach and downwind sailing. Lastly, tests of the algorithm's performance during beating mode and the challenging sailing maneuver, tacking, are analyzed.

### 5.5.1 Beam reach

The goal of the test was to measure the performance when sailing beam reached. It is when the wind direction is perpendicular to the sailboat's heading.

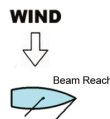


Figure 68: Beam reach illustration

GPS data from the test are visualized in Figure 69. It shows the boat sailing from north to south, close to the dock. The wind was blowing from approximately west to east. In Figure 70, one can identify that the apparent wind angle is steadily between 50 – 100° of the boat's heading. Further, the sail angle is consistent with the rule of an AoA of about 10°.

Figure 71 illustrates the wind speed in  $m/s$  and the Speed over Ground (SoG) in  $knots$ . According to the graph, the maximum speed reached during beam reach were  $0.7Kn$ . With further improvements, a higher SoG should be attainable.

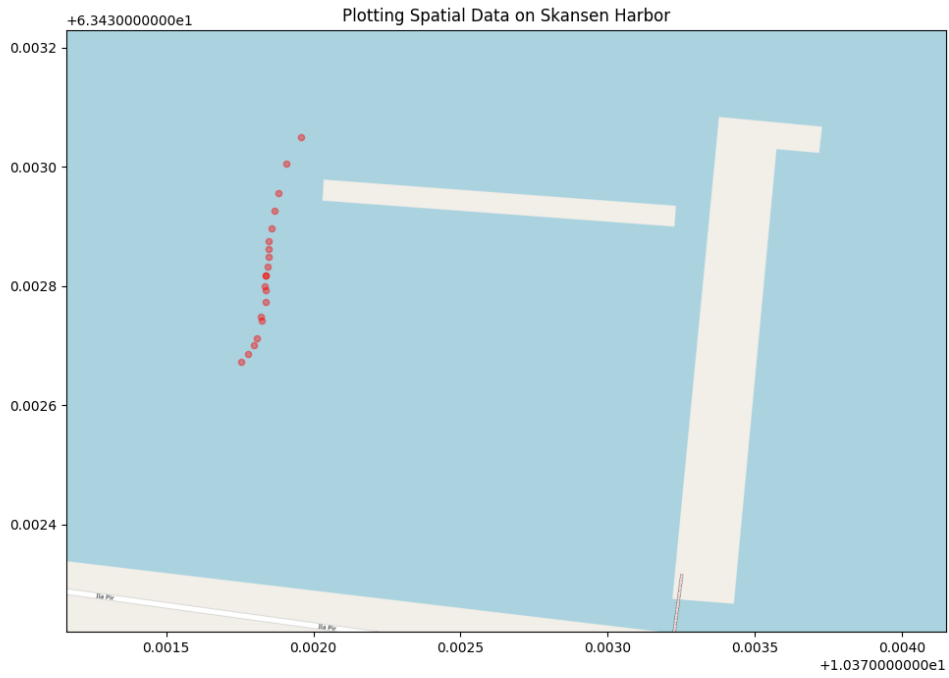


Figure 69: Beam reach map

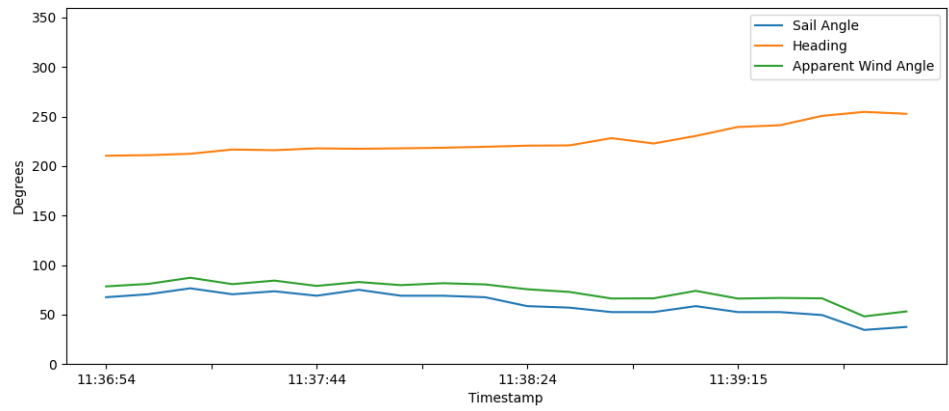


Figure 70: Beam reach angles

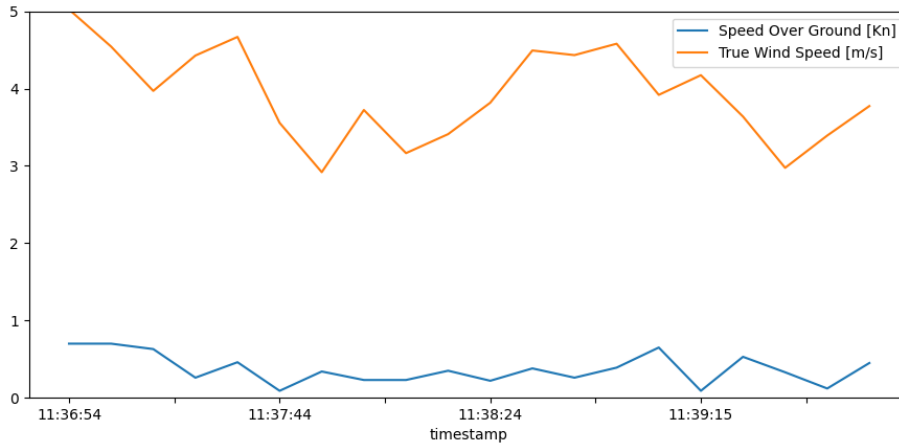


Figure 71: Beam reach speeds

### 5.5.2 Downwind sailing

The downwind sailing test was performed during the sea trial. While far out in the harbor, a waypoint was set at the shore from where it initially started. The goal was to see if the boat could sail to a waypoint with a tailwind.

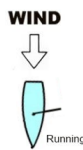


Figure 72: Downwind illustration

- Is the boat able to sail with wind coming from the back?
- Does the sail enter drag mode?



Figure 73: Downwind

Figure 74 is a plot of the sail angle, heading, and apparent wind angle as the sailboat turns clockwise about  $100^\circ$  heading towards the dock, creating the path seen in Figure 76. The conditions for downwind sailing mode are met when the apparent wind direction is within the range  $\beta \in [135^\circ, 220^\circ]$ . As the green graph in Figure 74 drops below  $225^\circ$ , the gap between the green and the blue graph increases from  $10^\circ$  to  $90^\circ$  indicating that the sail has a  $90^\circ$  angle of attack. At this point, the sail is no longer acting as a lifting surface but more of a parachute. At about the same time, the IMU stops communicating, a problem that will be discussed later. However, this does not affect the boat too much as it is already on a steady course.

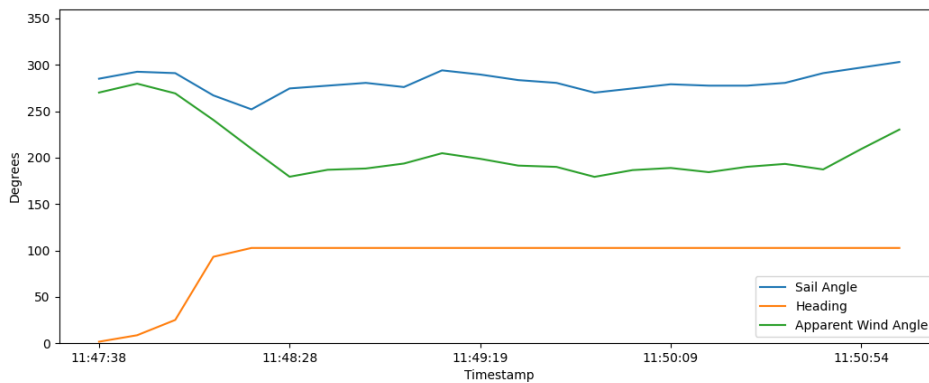


Figure 74: Down wind sailing mode angles

---

The blue graph in Figure 75 shows how the boat accelerates as it enters downwind sailing mode with a velocity settling around  $1 - 1.5Kn$ . When using the sail in drag mode, the boat will not be able to sail faster than the wind. In this case, the speed is quite a bit lower, likely due to drag forces on the hull from the water, and that  $90^\circ$  AoA does not result in the maximum coefficient of drag for the unsymmetrical convex-shaped side profile of the sail. The boat did not sail directly downwind either, as shown by the green graph in Figure 74 sometimes being closer to  $200^\circ$  rather than  $180^\circ$ . The wind speed is given in m/s, while the boat speed is recorded in knots, making the difference between boat and wind speed even bigger. More time to accelerate would not have helped either, as Figure 75 shows that the boat is decelerating when the wind speed drops.

However, these speeds were relatively high compared to speeds reached in the rest of the test. The boat was also able to make it back to the dock, making the test a success.

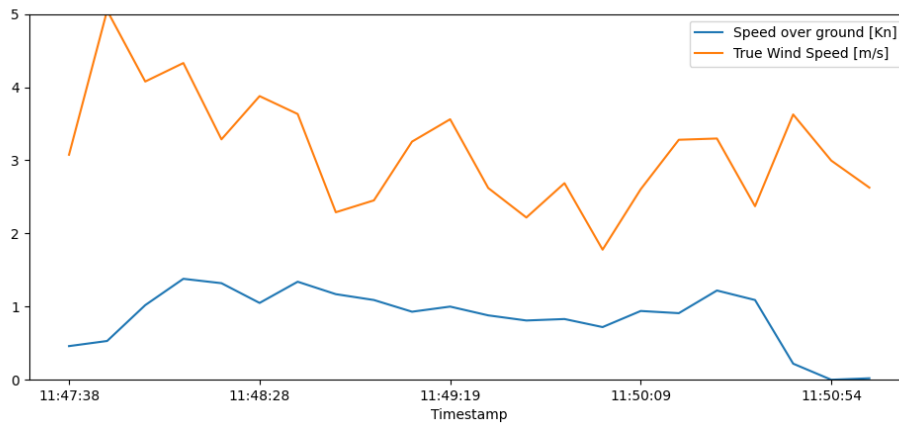


Figure 75: Down wind sailing test speeds



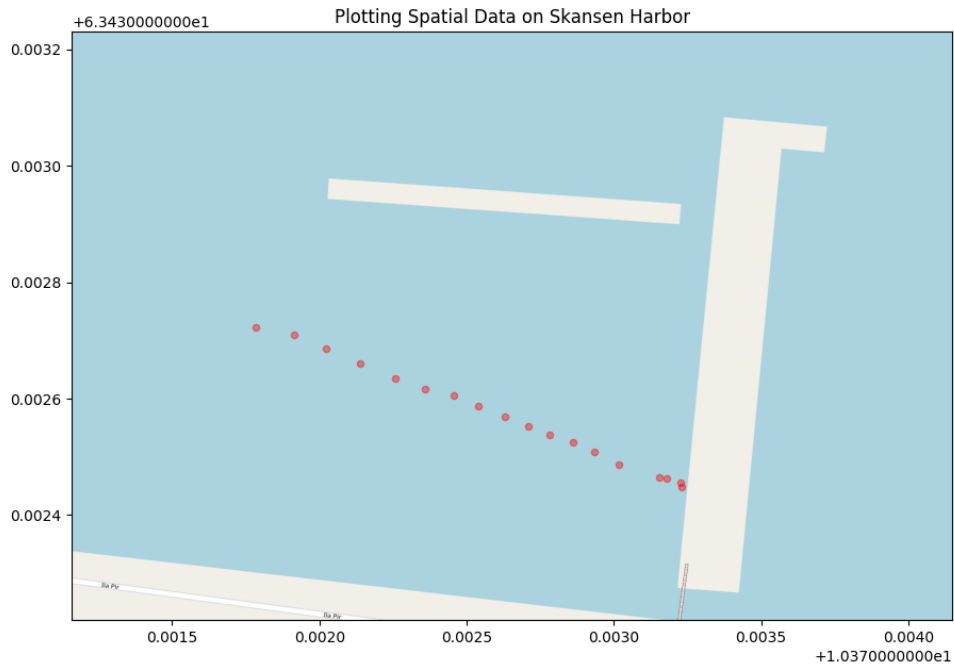


Figure 76: Path traveled in down wind sailing mode

### 5.5.3 Beating sailing mode and tacking maneuvers

During the sea trial, the boat's starting position at the harbor was set in a direction against the wind to force beating mode. The boat should detect this and calculate a midway point outside the no-sail zone, and a red marker should appear on the map as seen in Figure 77. After the boat reached the midway point, it would check if the goal was outside the no-sail zone, or a new midway point would be created on the opposite side of the no-sail zone, therefore requiring the boat to tack.

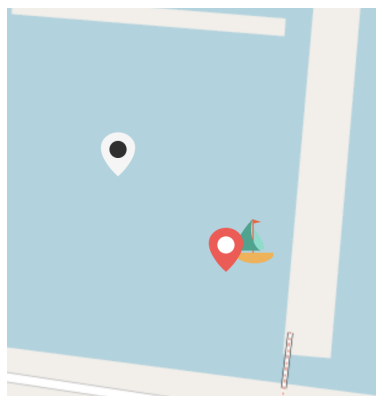


Figure 77: Sailboat and a mid waypoint

It quickly became apparent that the boat struggled to perform tacking maneuvers. Figure 78 shows a scenario from the test where the target heading crosses apparent and true wind direction, making the boat initiate a tacking maneuver. The heading graph shows the boat stops turning

---

when approaching the apparent wind, preventing the boat from completing the maneuver. The main challenge was for the boat to have enough speed to enter the no-sail zone and exit on the other side. When entering the no-sail zone, the sail algorithm would still try to add an angle of attack to the sail, sometimes resulting in the boat going backward. Instead, the sail should neutralize itself when entering this zone to minimize forces from the headwind. Combined with a higher speed requirement and a more confident rudder movement when tacking should help mitigate some of the tacking problems.

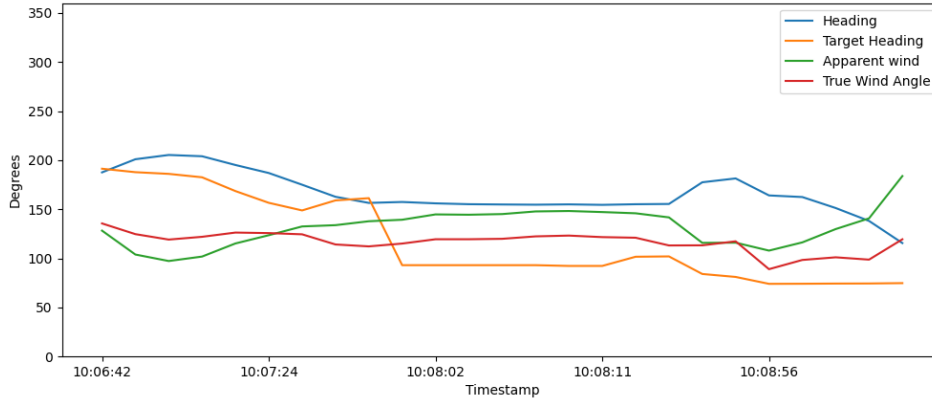


Figure 78: Sailboat struggling to tack (adjusted 180° to improve readability)

When sailing in beating mode, the boat is designed to sail close-hauled to the wind and tack when a specific deviation from the course is reached. Figure 79 are data from the trial when the boat was in beating mode, attempting to sail closed hauled 60° of from headwind. However, as the graphs show, the heading, target heading, and true wind were equal, meaning the boat experienced a direct headwind and could not sail. The issue was that the target heading was calculated using apparent wind direction, not true wind. As seen in Figure 79, there is a 60° difference between target heading and apparent wind direction, not true wind direction as it should have been. In Figure 79 this resulted in heading and true wind being equal, meaning the boat experienced direct headwind and could not sail. The mistake significantly reduced the performance of the beating sailing mode unless the boat was sailing north, aligning apparent and true wind direction. It is likely that without this mistake, the boat would have performed much better.

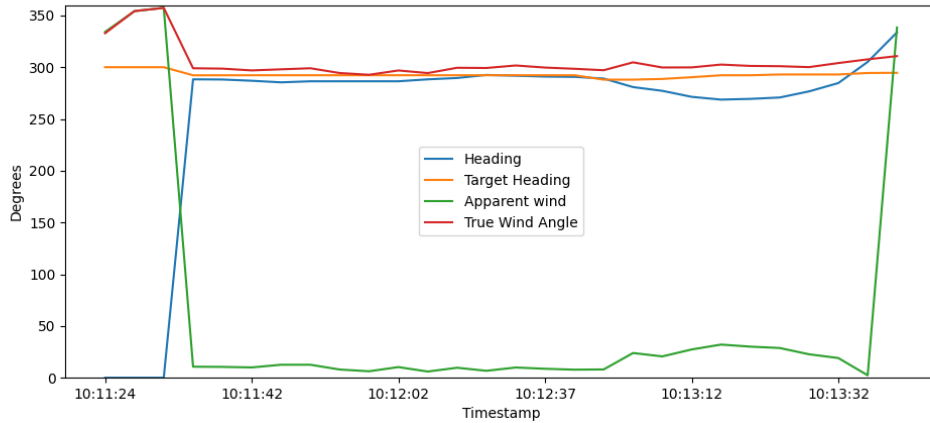


Figure 79: Beating calculation error

## 5.6 Integrated system review

During testing, the affordable components proved sufficient for the task at hand. Where problems occurred, a software solution was enough to rectify their inadequacy. The different subsystems of the mechatronic system could cooperate and act as a single coherent system. The subsystems appeared transparent when a new path was created and sent to the boat, not revealing how the data was represented or where it was stored. The dashboard was accessed from several devices, including personal computers and mobile phones. They were all able to communicate with the other parts of the system. However, the user interface itself did not adapt correctly to smaller displays. The unexpected disconnections of the Arduino Nano did not affect the rest of the system, but little feedback was given to the operator, and manual actions were required to restore the connection. The system should handle such events autonomously without requiring an operator to discover and fix them. Despite these issues, the prototype could perceive its environment and act upon it, although not always correctly. Still, valuable test data was collected and analyzed, creating a solid foundation for future development.

---

## 6 Conclusion

An approach to developing and validating an autonomous sailboat has been proposed and implemented. The approach presents a distributed system with an on-land server, an operator dashboard, and a mechatronic system. It facilitates an autonomous sailboat's development, testing, and operation. The proposed solution was implemented, validated, and documented in this thesis. Comprehensive development in mechanics, electronics, and computer science was fundamental to creating a complete and integrated system. As part of the validation process, a full-scale sea trial was conducted, exposing the system to a real-world environment. Analysis of the test results gave confidence in the system's potential to reach the long-term goal of a fully autonomous sailboat for monitoring and collecting oceanographic data.

The use of an on-land cloud server proved valuable. It maintained a stable connection to multiple clients simultaneously, providing real-time capabilities to the system. The server was also responsible for logging and collecting essential data from the field test. Performance figures from the test showed that the cloud service was more than capable of handling the load. Most of the tasks assigned to the cloud server were offloaded from the sailboat's main computer freeing up its resources and lowering the hardware requirement. The cloud services can be scaled and integrated easily, and ownership can be transferred to new candidates working on the project with the help of a cloud platform.

A dashboard was developed to provide operators and stakeholders a HMI. It was helpful during development and testing, displaying raw data, sensor values, and error codes in a way that was easy to comprehend. The tools for creating paths, sending commands, and monitoring the boat state were crucial to reaching the achievements in the sea trial.

An approach for a mechatronic system to enable autonomous sailing was designed, built, and tested. It included sensors, actuators, hardware, electronics, and software necessary to perceive its surroundings, communicate with a server, and change its state accordingly. During the sea trial, the sailboat system proved capable of fully autonomous sailing. However, some weaknesses of the system were uncovered. Based on this, future recommendations and suggestions are presented.

- The software issues identified in Section 5 should be rectified, and a new sea trial should be conducted to set a new baseline.
- More advanced sailing algorithms should be developed, improving path planning, maneuvering, power consumption, and situational awareness.
- The actuator moving the sail is, according to the calculations, not strong enough to meet design requirements and should be replaced with a stronger version before conducting more extreme tests.

---

## Bibliography

- [1] Adrian S Pleyrn and Magnus W Ølstad. *How to build a platform to support development of an autonomous sailboat?* 2021.
- [2] Maulshree Singh et al. ‘Digital Twin: Origin to Future’. In: *Applied System Innovation* 4.2 (2021), p. 36. DOI: 10.3390/asi4020036.
- [3] Michael Grieves and John Vickers. ‘Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems’. In: *Transdisciplinary Perspectives on Complex Systems* (Aug. 2016), pp. 85–113. DOI: 10.1007/978-3-319-38756-7\_4.
- [4] Allison Maloney. *The difference between a simulation and a digital twin*. MindSphere, Oct. 2019. URL: <https://blogs.sw.siemens.com/mindsphere/the-difference-between-a-simulation-and-a-digital-twin/#:~> (visited on 6th May 2022).
- [5] *Modern Manufacturing’s triple play: Digital Twins, analytics, IOT*. URL: [https://www.sas.com/en\\_us/insights/articles/big-data/modern-manufacturing-s-triple-play-digital-twins-analytics-iot.html](https://www.sas.com/en_us/insights/articles/big-data/modern-manufacturing-s-triple-play-digital-twins-analytics-iot.html).
- [6] Andreas Birk and Max Pflingsthor. ‘A HMI supporting adjustable autonomy of rescue robots’. In: *RoboCup 2005: Robot Soccer World Cup IX* (2006), pp. 255–266. DOI: 10.1007/11780519\_23.
- [7] Morgan William Arthur Trench. ‘Developing a Single Page Web Application to Monitor and Control a Solar Powered Autonomous Boat’. In: ().
- [8] R Divya et al. ‘Autonomous Sailing Boat’. In: *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*. Springer, 2021.
- [9] Andouglas GS Júnior et al. ‘N-BOAT: An autonomous robotic sailboat’. In: *2013 Latin American Robotics Symposium and Competition*. IEEE. 2013, pp. 24–29.
- [10] Sarah Sharples and John R. Wilson. *Evaluation of human work*. Taylor and Francis, an imprint of CRC Press, 2015.
- [11] Shaolong Yang et al. ‘Generic and Flexible Unmanned Sailboat for Innovative Education and World Robotic Sailing Championship’. In: *Frontiers in Robotics and AI* 8 (2021), p. 27.
- [12] Maarten Van Steen and Andrew S Tanenbaum. *Distributed systems*. Maarten van Steen Leiden, The Netherlands, 2017.
- [13] Michael A Murphy and Sebastien Goasguen. ‘Virtual Organization Clusters: Self-provisioned clouds on the grid’. In: *Future Generation Computer Systems* 26.8 (2010), pp. 1271–1281.
- [14] Luis M. Vaquero et al. ‘A Break in the Clouds: Towards a Cloud Definition’. In: *SIGCOMM Comput. Commun. Rev.* 39.1 (Dec. 2009), pp. 50–55. ISSN: 0146-4833. DOI: 10.1145/1496091.1496100. URL: <https://doi.org/10.1145/1496091.1496100>.
- [15] Seyyed Mohsen Hashemi and Amid Khatibi Bardsiri. ‘Cloud computing vs. grid computing’. In: *ARP journal of systems and software* 2.5 (2012), pp. 188–194.
- [16] Stefan Poslad. *Ubiquitous computing: smart devices, environments and interactions*. John Wiley & Sons, 2011.
- [17] J. Castán et al. ‘Improving Vehicular Mobility in Urban Traffic Using Ubiquitous Computing’. In: *Journal of Computer and Communications* 04 (Jan. 2016), pp. 57–62. DOI: 10.4236/jcc.2016.410006.
- [18] M. Satyanarayanan. ‘Pervasive computing: vision and challenges’. In: *IEEE Personal Communications* 8.4 (2001), pp. 10–17. DOI: 10.1109/98.943998.

- 
- [19] Gordon S. Blair et al. *Interoperability in Complex Distributed Systems*. Ed. by Marco Bernardo and Valérie Issarny. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.
- [20] W3C. *Web Services Architecture*. URL: <https://www.w3.org/TR/ws-arch/> (visited on 27th May 2022).
- [21] W3C. *OWL-S: Semantic Markup for Web Services*. URL: <https://www.w3.org/Submission/OWL-S/> (visited on 27th May 2022).
- [22] Florian Böhm and Axel Schulte. ‘UAV Autonomy Research - Challenges and Advantages of a Fully Distributed System Architecture’. In: *International Telemetering Conference Proceedings* (2012). ISSN: 0884-5123. URL: <http://hdl.handle.net/10150/581826>.
- [23] Gerardo Pardo-Castellote. *OMG data-distribution service (DDS): Architectural overview*. Tech. rep. REAL-TIME INNOVATIONS INC SUNNYVALE CA, 2004.
- [24] Joseph M Schlesselman, Gerardo Pardo-Castellote and Bert Farabaugh. ‘OMG data-distribution service (DDS): architectural update’. In: *IEEE MILCOM 2004. Military Communications Conference, 2004*. Vol. 2. IEEE. 2004, pp. 961–967.
- [25] Chen Hong and Dianxi Shi. ‘A Cloud-Based Control System Architecture for Multi-UAV’. In: (2018). DOI: 10.1145/3265639.3265652. URL: <https://doi.org/10.1145/3265639.3265652>.
- [26] N. Kyura and H. Oho. ‘Mechatronics-an industrial perspective’. In: *IEEE/ASME Transactions on Mechatronics* 1.1 (1996), pp. 10–15. DOI: 10.1109/3516.491405.
- [27] D.M. Auslander. ‘What is mechatronics?’ In: *IEEE/ASME Transactions on Mechatronics* 1.1 (1996), pp. 5–9. DOI: 10.1109/3516.491404.
- [28] T.-R. Hsu. ‘Mechatronics. An overview’. In: *IEEE Transactions on Components, Packaging, and Manufacturing Technology: Part C* 20.1 (1997), pp. 4–7. DOI: 10.1109/3476.585138.
- [29] Alfian Ma’arif et al. ‘Kalman filter for noise reducer on sensor readings’. In: *Signal and Image Processing Letters* 1.2 (2019), pp. 50–61.
- [30] Xiaozheng Lai et al. ‘IOT implementation of Kalman filter to improve accuracy of air quality monitoring and prediction’. In: *Applied Sciences* 9.9 (2019), p. 1831. DOI: 10.3390/app9091831.
- [31] Alex Becker. *Online kalman filter tutorial*. URL: <https://www.kalmanfilter.net/default.aspx>.
- [32] Charles C Holt. ‘Forecasting seasonals and trends by exponentially weighted moving averages’. In: *International journal of forecasting* 20.1 (2004), pp. 5–10.
- [33] *Anatomy of a sailboat*. URL: <https://www.sailrite.com/anatomy-of-a-sailboat>.
- [34] *Point of sail*. May 2022. URL: [https://en.wikipedia.org/wiki/Point\\_of\\_sail](https://en.wikipedia.org/wiki/Point_of_sail).
- [35] Roland Stelzer and Tobias Pröll. ‘Autonomous sailboat navigation for short course racing’. In: *Robotics and Autonomous Systems* 56.7 (2008), pp. 604–614. ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2007.10.004>. URL: <https://www.sciencedirect.com/science/article/pii/S0921889007001480>.
- [36] Patrick F Rynne and Karl D von Ellenrieder. ‘Development and preliminary experimental validation of a wind-and solar-powered autonomous surface vehicle’. In: *IEEE Journal of Oceanic Engineering* 35.4 (2010), pp. 971–983.
- [37] J Otto Scherer. ‘Aerodynamics of high-performance wing sails’. In: *Marine Technology and SNAME News* 11.03 (1974), pp. 270–276.
- [38] Maria Brudeseth Dyrseth. ‘Development, Design and Production of a Wingsail for an Autonomous Surface Vessel’. In: (2020). URL: <https://hdl.handle.net/11250/2781743>.
-

- 
- [39] Claes Tretow. ‘Design of a free-rotating wing sail for an autonomous sailboat’. In: 2017.
- [40] Paul Miller et al. ‘An Alternative Wing Sail Concept for Small Autonomous Sailing Craft’. In: 2018.
- [41] *Wind amp; Solar Powered Autonomous Vehicles*. URL: <https://www.saildrone.com/technology/vehicles>.
- [42] *Unmanned Surface Vessel*. URL: <http://www.sailbuoy.no/>.
- [43] URL: <http://airfoiltools.com/airfoil/details?airfoil=naca0018-il>.
- [44] URL: <http://airfoiltools.com/airfoil/details?airfoil=n0009sm-il>.
- [45] Hiroyuki FURUKAWA et al. ‘Performance of wing sail with multi element by two-dimensional wind tunnel investigations’. In: *Journal of Fluid Science and Technology* 10.2 (2015). DOI: 10.1299/jfst.2015jfst0019.
- [46] Hadi Saoud et al. ‘Routing and course control of an autonomous sailboat’. In: *2015 European Conference on Mobile Robots (ECMR)* (2015). DOI: 10.1109/ecmr.2015.7324218.
- [47] Zhenyu Yu, Xiping Bao and Kenzo Nonami. ‘Course keeping control of an autonomous boat using low cost sensors’. In: *Journal of System Design and Dynamics* 2.1 (2008), pp. 389–400.
- [48] Corentin Jegat. *Ensta Bretagne*. URL: [https://www.ensta-bretagne.fr/jaulin/rapport2019\\_jeguat.pdf](https://www.ensta-bretagne.fr/jaulin/rapport2019_jeguat.pdf).
- [49] Thomas Augenstein et al. ‘Using a controlled sail and tail to steer an autonomous sailboat’. In: *World Robotic Sailing championship and International Robotic Sailing Conference*. Springer, 2016, pp. 91–103.
- [50] Scott A Bortoff. ‘Path planning for UAVs’. In: *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*. Vol. 1. 6. IEEE, 2000, pp. 364–368.
- [51] Ryo Takei et al. ‘A practical path-planning algorithm for a simple car: a Hamilton-Jacobi approach’. In: *Proceedings of the 2010 American control conference*. IEEE, 2010, pp. 6175–6180.
- [52] Alex Nash, Sven Koenig and Craig Tovey. ‘Lazy Theta\*: Any-angle path planning and path length analysis in 3D’. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 24. 1. 2010, pp. 147–154.
- [53] Johannes Langbein, Roland Stelzer and Thom Frühwirth. ‘A rule-based approach to long-term routing for autonomous sailboats’. In: *Robotic Sailing* (2011), pp. 195–204. DOI: 10.1007/978-3-642-22836-0\_14.
- [54] Mingshu Du et al. ‘Study of Long-term Route Planning for Autonomous Sailboat’. In: *Proceedings of the International Robotic Sailing Conference*. 2018.
- [55] Jorge Cabrera-Gómez et al. ‘Optimization-based weather routing for sailboats’. In: *Robotic Sailing 2012* (2013), pp. 23–33. DOI: 10.1007/978-3-642-33084-1\_3.
- [56] *A\* search algorithm*. May 2022. URL: [https://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm).
- [57] *Dijkstra’s algorithm*. May 2022. URL: [https://en.wikipedia.org/wiki/Dijkstra%5C%27s\\_algorithm](https://en.wikipedia.org/wiki/Dijkstra%5C%27s_algorithm).
- [58] Nitin R Chopde and Mangesh Nichat. ‘Landmark based shortest path detection by using A\* and Haversine formula’. In: *International Journal of Innovative Research in Computer and Communication Engineering* 1.2 (2013), pp. 298–302.
- [59] Zainal Arifin, Muhammad Rivani Ibrahim and Heliza Rahmania Hatta. ‘Nearest tourism site searching using Haversine method’. In: *2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*. IEEE, 2016, pp. 293–296.
-

- 
- [60] M Basyir et al. ‘Determination of nearest emergency service office using haversine formula based on android platform’. In: *EMITTER International Journal of Engineering Technology* 5.2 (2017), pp. 270–278.
- [61] S Keliwar. ‘A Secondary Study Examining the Effectiveness of Network Topologies: The Case of Ring, Bus, and Star Topologies’. In: *International Journal of Communication and Computer Technologies* 8.2 (2020), pp. 5–7.
- [62] *Trello helps teams move work forward*. URL: <https://trello.com/>.
- [63] *Where the world builds software*. URL: <https://github.com/>.
- [64] *NTNU-Autoboat, GitHub Organization*. URL: <https://github.com/NTNU-AutoBoat>.
- [65] *Simple Kalman Filter*. 2022. URL: <https://github.com/denyssene/SimpleKalmanFilter>.
- [66] Sverre Gauden. ‘Development of an Internally Actuated GFRP Rigid Wing Sail for an Autonomous Surface Vesse’. In: (2021). URL: <https://hdl.handle.net/11250/2787204>.
- [67] CubeMars. *AK80-9*. URL: <https://store.cubemars.com/goods.php?id=982> (visited on 11th May 2022).
- [68] Biltema. *Gummikabel RDOE*. URL: <https://www.biltema.no/bygg/elinstallasjoner/installasjonskabler/gummikabel-rdoe-2000017922> (visited on 11th May 2022).
- [69] Almar Vreim Brandal. ‘Development of a Modular Polyethylene Pipe Hull and GFRP Rudder System for an Autonomus Surface Vessel’. In: (2021). URL: <https://hdl.handle.net/11250/2787221>.
- [70] *Websocket system design for authentication*. URL: <https://websockets.readthedocs.io/en/latest/topics/authentication.html#system-design>.



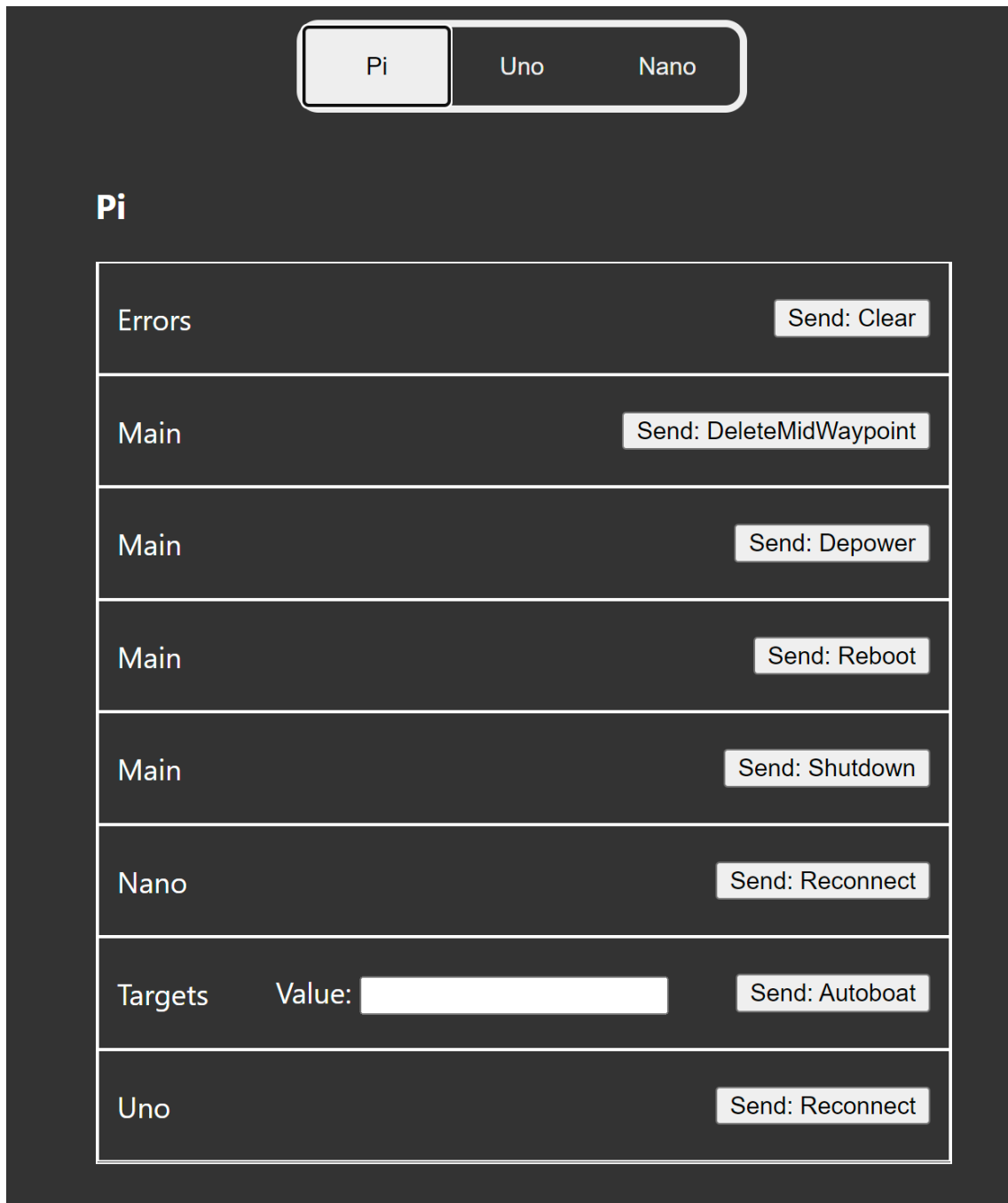
---

# Appendix

## A Source code

For now, the GitHub repositories are private. However, the source code can be requested by contacting one of the authors of this master thesis.

## B Command system



Pi

Uno

Nano

## Uno

Encoder

Value:

Send: Frequency

Encoder

Angle:

Send: SetZAngle

Main

Send: Reboot

Main

Send: Whoami

Messenger

Value:

Send: Frequency

Messenger

Value:

Send: Timeout

Motor

Send: Back

Motor

Send: Enter

Motor

Send: Exit

Motor

Send: SetZero

Motor

Send: Start

Motor

Send: Stop

Navigator

Send: GoToZero

Navigator

Angle:

Send: SetPosition

Pi

Uno

Nano

## Nano

IMU		Send: Calibrate
IMU	Value: <input type="text"/>	Send: Frequency
IMU		Send: Reboot
Main		Send: Reboot
Main		Send: Whoami
Messenger	Value: <input type="text"/>	Send: Frequency
Messenger	Value: <input type="text"/>	Send: Timeout
Motor		Send: Calibrate
Motor		Send: CalibrationValues
Motor		Send: DefaultValues
Motor	Value: <input type="text"/>	Send: Frequency
Motor		Send: In
Motor		Send: Out
Motor	Offset: <input type="text"/>	Send: SetOffset
Motor	Position: <input type="text"/>	Send: SetPosition
Navigator	Length: <input type="text" value="91"/>	Send: SetImpulseLength
Navigator	Value: <input type="text"/>	Send: SetPrecision

---

## C Readmes from GitHub repositories

### C.1 RaspberryPi readme

The README start on the next page...

---

# RaspberryPi

---



## Setup

---

This setup will go through how to set up the Raspberry Pi to automatically send its IP address to the websocketserver on startup.

### Prerequisites

- [Python 3.9](#)
- [Websockets](#)
- [PySerial](#)
- [Firebase](#)
- [Pyudev](#)

### Python Libraries:

- [Asyncio](#)
- [Urllib](#)
- [Json](#)
- [Time](#)
- [Sys](#)
- [Os](#)
- [Math](#)
- [Socket](#)
- [Signal](#)
- [Threading](#)
- [Functools](#)

## How to set up a Raspberry Pi 4

### Installing Raspberry Pi OS

To begin, you would need the [Raspberry Pi Imager](#) to download and install the Raspberry Pi OS to an SD card.

After running the program, the first step is to select an Operating System. In this project, the **Raspberry Pi OS Lite (32-bit)** is sufficient enough, since the Pi won't be connected to a screen a

---

full desktop environment would be redundant.

Note! You can probably find the image in the section *Raspberry Pi OS (other)*.

There has been released a 64-bit version, but has not yet been tested (as of 14.02.2022).

The next step is to select the correct *Storage*. This should be an SD card with at least 16 GB available space. Finally click the **Write**-button to begin installing the operating system.

Before ejecting the SD card, you would need to decide for a solution to communicate with the Raspberry Pi.

### Connecting to the Raspberry Pi

In the beginning, SSH on a local network should good enough until the setup is complete.

To enable SSH on the Pi, simply create a file named `ssh`, without any extensions or data in the root directory of the SD card.

For connecting the Pi to a local network, you could use an ethernet cable or Wi-Fi. Click on [this link](#) to set up WiFi in the **Raspberry Pi Imager**.

Note! Sharing local network from a phone can work, but some challenges has been seen when using an iPhone.

Open a *Terminal* and write the following:

```
ssh hostname@pi-local-ip

# The hostname is by default: pi
# To find out the local ip, you would need to find a list of all the connected
# clients on the local network.

# You will now be prompted to save a "fingerprint",
# enter yes and proceed to enter the password.

# By default, the password should be: raspberry
```

Note! When you are asked to write commands in the next subsections, they are supposed to be commands for the Raspberry Pi, after having a SSH connection.

### Configuring The Pi

After connecting to the Pi, execute the following command to configure the Pi:

```
sudo raspi-config
```

---

In this menu, you can change multiple settings on the Raspberry Pi. A missing piece of information that can be important, are *Localisation Options*. Proceed to set these configurations to your specifications.

### Authentication problems

For some strange reason, you may eventually not be permitted to connect the Raspberry Pi. A solution for this problem, may be fixed by following [this guide](#).

Note! This must be done before receiving *permission denied*, thus should be done immediately.

### Installing Git and adding a SSH key

To install [Git](#), type the following.

```
sudo apt update
sudo apt install git

git --version
# Should output the latest Git version if installed correctly
```

To generate the SSH key and add it to this project repository, follow [this guide](#).

### Adding Pip, a package installer for Python

Open a *Terminal*, and write:

```
python --version # Start by checking the python version
# The output would hopefully be 3.9+, else you would need to update the Python version

sudo apt update
sudo apt install python3-pip

sudo apt install python3-pip

pip3 --version # Should output a pip version.
```

```
chmod u+x job.sh
```

### Download project

This section will show you how to clone this git repository. Type the following lines in the Terminal:

```
cd /to-your-desired-directory
git clone https://github.com/NTNU-AutoBoat/RaspberryPi.git
cd /RaspberryPi
```

---

## Sensor setup

Sensors are normally connected to the Pi through USB / Serial. However USB port locations (like: `/dev/ttyUSB0` ) may change from time to time. The code therefore searches for vendor name when identifying USB connected sensors. These vendor names must be defined as environment variables which can be accessed by the program. To automatically identify vendor names, the included script `findSensors.py` can be used. The current wind sensor on the other hand is connectet through a RS232 serial board. This does not report a vendor name, and instead the specific location must be manually identified and added. Follow the next steps to identify vendor names and add them to envvars including the wind sensor port.

1. Open a new terminal window and type:

```
python findSensors.py
```

2. The script will print vendor names and their USB location. Copy the vendor names for the next step
3. While still in the terminal, type:

### setup.sh instead.

```
sudo nano ~/.profile
```

2. Scroll down to the bottom, add the following lines, and fill in the names from the steps above:

```
export GPS_VENDOR = # GPS VENDOR NAME. "Prolific_Technology_Inc." as of 24.01.22
export IMU_VENDOR = # IMU VENDOR NAME. "Arduino__www.arduino.cc_" as of 24.01.22
export WIND_PORT = # WIND SENSOR PORT. "/dev/ttyS0" as of 24.01.22
```

3. Press `Ctrl` + `X` to exit.

Press `Y` to save.

Press `Enter` to close.

4. Reboot the Pi using the command:

```
sudo reboot
```

5. Print all environment variables by typing:



---

```
printenv
```

6. Verify that the new variables exist.

## Add user credentials

Authentication to access the websocket server is handled by Google's [Identity Platform](#). User credentials for authentication must be manually configured on the Raspberry Pi following these steps:

1. Create a new user in Firebase
  - i. Open [Firebase Console](#) and select the project.
  - ii. Navigate to *Authentication* using the left side panel.
  - iii. Click on *Add user*, fill in a new email and password and click *Add user*.
  - iv. Then navigate to *Project settings* by clicking on the cogwheel in the left side panel.
  - v. Scroll down to *Your apps and SDK setup and configuration*.
  - vi. The information under `const firebaseConfig` will be used in the next step.

2. Add credentials as environment variables to the Pi:

- i. Open a new terminal window and type

**setup.sh** instead.

```
sudo nano ~/.profile
```

- ii. Scroll down to the bottom, add the following lines, and fill in the information from the steps above:

```
export GOOGLE_MAIL= # EMAIL
export GOOGLE_API_KEY= # WEB API KEY
export GOOGLE_PASSWORD= # PASSWORD
export GOOGLE_AUTH_DOMAIN = # AUTH DOMAIN
export GOOGLE_STORAGE_BUCKET = # STORAGE BUCKET
export WEBSOCKET_SERVER_URI = # See websocket server readme
export NGROK_LOG_PATH= # PATH TO ENGROK LOG FILE e.g. "/home/pi/ngrok.log"
```

- iii. Press `Ctrl` + `X` to exit.

Press `Y` to save.

Press `Enter` to close.

- iv. Reboot the Pi using the command:

---

```
sudo reboot
```

v. Print all environment variables by typing:

```
printenv
```

vi. Verify that the new variables exist.

## Add to crontab

Next we need to make sure the script runs every time the Raspberry Pi starts up. To do this we add it to the crontab.

To edit the crontab type:

```
crontab -e
```

If you have multiple text editors installed, the system prompts you to select an editor to update the cron task list with. Use the number in the brackets to choose `Nano`.

Scroll down to the bottom of the page and type:

```
@reboot . $HOME/.profile; PATH_TO_PYTHON PATH_TO_REPO_FROM_GITHUB/run.py
```

For example: `@reboot . /home/pi/.profile; /usr/local/bin/python3.9 /home/pi/RaspberryPi/run.py`

To save and exit the editor:

Press `Ctrl` + `X` to exit.

Press `Y` to save.

Press `Enter` to close.

The script will now run on every boot and message internal and external ip to the server.

## The great wall of Eduroam

It can be quite a hassle to connect the Raspberry Pi to [eduroam](#). By following the guide below, you will hopefully succeed.

In this project, we used Raspberry Pi OS (Raspbian) version 11 (*Bullseye*) and therefore had to do some downgrading on the `wpa_supplicant` to get everything to work. This may not be an issue in the later versions, but could be a solution if it doesn't work.

---

Connect to your Raspberry Pi through SSH, and follow the steps below.

Note! Make sure the Raspberry Pi has an internet connection before following these steps.

## Downgrading wpa\_supplicant to Raspbian Stretch (version 9)

```
sudo apt remove wpasupplicant -y # Removes the latest version of wpa_supplicant

sudo mv -f /etc/apt/sources.list /etc/apt/sources.list.bak # Creates a backup of the P

sudo bash -c "echo 'deb http://raspbian.raspberrypi.org/raspbian/ stretch main contrib
# Adds an older source list

sudo apt-get update # Updates
sudo apt-get install wpasupplicant -y # Installs the older version of wpa_supplicant
sudo apt-mark hold wpasupplicant # Makes sure to not update wpa_supplicant later

sudo cp -f /etc/apt/sources.list.bak /etc/apt/sources.list # Returns the source list t

sudo apt-get update # Final update to the rest of the softwares
```

## University's CA certification

- Head over to [this link](#) to install the eduroam installer.
- Pick the correct university that you have access to.
- A small text under the big button, should say *Chose another installer to download*, click on it.
- Pick Linux.

You should receive a `.py` -file with your credentials. Scroll down to the line beginning with:

```
Config.CA = ""-----BEGIN CERTIFICATE-----
```

Copy all of the certifications including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`. Make sure to copy all of them, there may be multiple certifications following each other.

Go back to a *Terminal* connected to the Raspberry Pi:

```
cd /usr/share/ca-certificates/
sudo nano eduroam.pem
```

Copy and paste the the certificates into this file.

Press `Ctrl + X` to exit.

---

Press `Y` to save.

Press `Enter` to close.

## Configuring the wpa\_supplicant

Inside the *Terminal*:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

```
network={
    ssid="eduroam"
    scan_ssid=1
    key_mgmt=WPA-EAP
    eap=PEAP
    identity="student-email"
    anonymous_identity="student-email"
    password="student-password"
    ca_cert="/usr/share/ca-certificates/eduroam.pem"
    phase1="peaplabel=auto peapver=auto"
    phase2="auth=MSCHAPV2"
}
# The student email and password you use to connect to eduroam can be written here.
```

```
# You can try to hash your password doing the following (Not sure if it works)
```

```
read -s input ; echo -n $input | iconv -t utf16le | openssl md4
```

```
# You should receive a hashed version of your password.
```

```
# The password field can then be:
```

```
# password=hash:<your-hashed-password-here-without-less/greater-signs>
```

## Finally

You should now be ready to connect. Type the following to try to connect without rebooting:

```
wpa_cli -i wlan0 reconfigure
```

To check if you are connected:

```
iwconfig
```

Check if the Pi has internet connection, the bug we had prevented us from connecting to the web, thus we had to downgrade.

---

If the Raspberry Pi is connected to **eduroam** while you are at another internet connection, you can still reach the Pi by using a VPN (if your University provide one). You still need to find out the Pi's local IP.

This is a guide inspired by [this article](#).

## ngrok and 4G dongle

---

To allow external connection to the Raspberry Pi, we used a 4G Dongle by [Huawei](#). The carrier provider need to allow external connection trough the ip-address for it to work. In this project, we used [Telia](#) as our provider.

The usual APN they use is: `internet.netcom.no` . This APN does however not allow us to create any direct connection to the Raspberry Pi with e.g. `ssh` .

There is another APN: `vpn.telia.no` . It may allow easier two-way connections.

If the 4G dongle's light indicator is constant and blue, you should have a connection. With the standard configuration, you should not need to any changes.

The standard settings uses the first APN option. Therefor the project includes another solution for external connections to the Raspberry Pi by using [ngrok](#).

### Setup ngrok

To download and unzip ngrok:

```
sudo wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-arm.zip
unzip ngrok-stable-linux-arm.zip
```

Ngrok should now be unzipped to the home directory of the Raspberry Pi `/home/pi/` . The next steps require you to have a [Ngrok account](#).

After registration you need to copy **Your Authtoken** to proceed.

Paste in your authtoken at: `__Your Authtoken__`.

```
cd /to-your-ngrok-path
./ngrok authtoken __Your Authtoken__
```

The next step will create a [TCP](#) tunnel for connecting externally to the Raspberry Pi. Ngrok will create an URL with a port number. To make this available for the rest of the program, you can log it with this simple command:

---

```
./ngrok tcp 22 --log=stdout > /home/pi/ngrok.log &
```

**NOTE** The text after the last ">" and before the "&" is the path to where ngrok will write the log file. The python script, `ngrok.py`, will read the URL from a file at a specific path. This path could be changed, but must be identical in the commando above and in the `ngrok.py` file to work.

It is worth noting that the TCP tunnel will close if the terminal window is terminated. Using a cron scheduler like [crontab](#), then you won't need to worry about it. If you are using a regular terminal, e.g. through SSH, you could use this command to add it to the jobs list and run in the background.

```
./ngrok tcp 22 > /dev/null &

jobs -l          # To List all the jobs

disown -h %n     # Where n is the process index, not the ID!

exit            # To close the terminal
```

*< /dev/null is used to instantly send EOF to the program, so that it doesn't wait for input (/dev/null, the null device, is a special file that discards all data written to it, but reports that the write operation succeeded, and provides no data to any process that reads from it, yielding EOF immediately). & is a special type of command separator used to background the preceding process. -Some StackOverflow user*

## GPS

---

This program is built and tested to work with GlobalSat G-Star IV USB GPS Reciever (BU-353S4). However some setup might be required to make it work. Following sections will explain how to read, write, and change the GPS reciever.

### Communicate with the reciever

First we need a way to communicate with the reciever. Follow the steps below to read raw data from the reciever: (We recommend to follow these steps on a regular computer)

- Connect the reciever to one of the USB ports on your computer.
- Download and install [CoolTerm](#) for you OS.
- Once installed, open CoolTerm and select *options* from the menubar.
- Click on the dropdown menu next to *Port* to select the port where you plugged in the GPS reciever. If you only have one USB device connected it will most likley be only one option.
- Click on the dropdown menu next to *Baudrate* and choose *4800*.
- Leave the other settings unchanged and select *OK*.

- 
- Press *Connect* from the menubar to start reading from the device

If the output only consists of unreadable symbols the device is in the wrong mode. See the section [Switch mode](#).

If the output is readable, look for these two lines:

```
$GPGLL,X,X,X,X,X,X,X,X*XX
```

```
$GPRMC,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X*XX
```

If one or both of them are missing, see the section [Reprogram output](#) to fix it.

To increase performance unnecessary output can be disabled and the frequency of output can be adjusted. This is also described in the section [Reprogram output](#).

## Switch mode

**NOTE:** Please read [Communicate with the reciever](#) before continuing with this section.

The GPS reciever must be in NMEA0183 mode, not SiRF binary mode. Follow the steps below to enable NMEA mode:

- Open CoolTerm and connect to the reciever as described in [Communicate with the reciever](#)
- From the top bar click on the dropdown menu *Connection* and select *Send String*
- Select *HEX* and paste the string below into the text field

1 sec GGA, 1 sec RMC, baud rate 4800.

```
A0 A2 00 18 81 02 00 00 01 01 00 00 00 00 01 01 00 00 00 00 00 00 00 00 12 C0 01
```

Check out this link to read more about SiRF mode: [SiRF Binary Protocol Reference Manual](#)

[NMEA Reference Manual](#)

## Reprogram output

**NOTE:** Please read [Communicate with the reciever](#) before continuing with this section.

The reciever is capable of outputting many different NMEA messages. This project need GLL and RMC messegas to work properly. To verify

---

## C.2 Dashboard app readme

The README start on the next page...



---


# DashboardApp

---

This repository contains the dashboard application's source code that allows monitoring and control for the autonomous sailboat.

## Demo

---

demo\_image

<https://ntnu-autoboat.web.app/>

## Setup

---

This section will guide you through setting up the web application by downloading, installing, and executing.

### Prerequisites

- [Node.Js](#)

### Javascript Libraries:

- [React](#)
- [Leaflet](#)
- [React Leaflet](#)
- [Threejs](#)
- [Firebase](#)
- [dnd-kit](#)
- [uuid](#)

### Download project

This section will show you how to clone this git repository. Type the following lines in the Terminal:

```
cd /to-your-desired-directory
git clone https://github.com/NTNU-AutoBoat/DashboardApp.git
cd /DashboardApp
```

### Running the React application

---

This section will guide you in starting up the web application locally. To be able to do so, you would need a package manager like [NPM](#), which is installed through [Node.Js](#), or [Yarn](#).

To install the packages, do the following in the Terminal:

```
cd /to-project-directory
```

**NPM:**

```
npm install
```

**Yarn:**

```
yarn install
```

After installing every package listed in `package.json`, you can now start the application by typing the following:

**NPM:**

```
npm start
```

**Yarn:**

```
yarn start
```

React-script should start hosting the application locally at `localhost:3000`

## How to use

---

The home page requires authentication available on Firebase and can be received from the project leader. After authentication, you are presented with an interactive [Leaflet](#) map with last location of the boat and a path with waypoints. The interactive map can be used to create and update paths which are sent to the boat, or downloaded as a JSON file on your local computer. Further, an instrument page is available for watching dials with animations, reflecting the latest and target information from the server. A data and parameters page are available and can present raw data from the [server](#). Data are not editable, however, parameters are under some restrictions. Notifications may popup at the upper right corner. The notification center are available by clicking on bell, which provides a history view of previous notifications and a button to mute messages. Finally, you can toggle a button at the upper right corner for activating rapid updating if the [Raspberry Pi](#) is online, and start a autopilot mode.

---

### C.3 WebSocket server readme

The README start on the next page...

---

# Websocket-Server

---

## Setup

---

This setup will go through how to set up the WebSocket server on a local machine and continuous integration with GitHub (optional).

### Prerequisites

- [Python 3.9](#)
- [Websockets](#)

### Python Libraries:

- [Asyncio](#)
- [Urllib](#)
- [Json](#)
- [Time](#)
- [Sys](#)
- [Os](#)
- [Csv](#)
- [Logging](#)
- [Signal](#)

### Download project

This section will show you how to clone this git repository. Type the following lines in the Terminal:

```
cd /to-your-desired-directory
git clone https://github.com/NTNU-AutoBoat/Websocket-Server.git
cd /Websocket-server
```

### Start server

Ensure that you are using python 3.9 or higher by typing:

```
python --version
```

**NOTE:** You might need to type `python3` instead of `python`.

---

With the right version of python, you can start the server by typing:

```
python server.py
```

The server will be hosted on `0.0.0.0` using port `8080`. To change this you can open `server.py` in your favorite editor or type:

```
sudo nano server.py
```

Scroll down to the line where the existing address is and change it.

```
async with websockets.serve(counter, *ADDRESS*, *PORT*, logger=logger):
```

**NOTE:** It is in `line: 112`. Last updated: *10.06.2022*.

Press `Ctrl` + `X` to exit.

Press `Y` to save.

Press `Enter` to close.

## Google Cloud Platform

The server has transitioned to Google Cloud Platform. As of now, it runs in a container as a *Cloud Run* service. When a developer pushes to the main branch, a trigger will be sent to *Cloud Build*, where the container will be built and automatically uploaded to *Cloud Run*. All traffic will be routed to the latest revision of the server.

Logs will be periodically uploaded to the *Cloud Storage* service provided by Google Cloud Platform.

Further, for reading the latest state of the server, check out the *Firestore Database* which is updated regularly.

**NOTE:** Multiple builds will accumulate, which eventually increases the costs of the service. These can be deleted in the *Container Registry* section.

---

#### C.4 Arduino readme

The README start on the next page...

---

# Arduino

---

## Setup

---

This setup will go through how to download, compile and upload an Arduino program.

### Prerequisites

- [PlatformIO](#)

#### C++ Libraries:

- [Seeed IMU](#)
- [Seeed CAN](#)
- [SimpleKalmanFilter](#)

### Download project

This section will show you how to clone this git repository. Type the following lines in the Terminal:

```
cd /to-your-desired-directory
git clone https://github.com/NTNU-AutoBoat/Arduino.git
cd /Arduino
```

### How to set up an Arduino

The following guide will explain the steps to download the repository, compile code for an Arduino Uno, and upload it.

To begin, you would need the software [PlatformIO](#).

#### Compile and upload code for the Arduino Uno

Connect the Arduino to your computer.

Enter the directory for the code of the Arduino Uno by typing the following into a terminal:

```
cd /to-your-project-directory/Arduino
cd /Uno
```

The project is pre-registered to look for the correct microcontroller that fits the code.

---

To compile and upload the code with the help of [PlatformIO CLI](#), type the following into a terminal of your choice:

```
pio run --target upload
```

The code will begin to compile, and upload the firmware devices specified inside the `platformio.ini`. A success indication will be at the end of the message prompts.

**NOTE:** The sequence of tasks are identical for the Arduino Nano, but requires to be directed into the `Nano` directory before executing the last command.



