Sophus Stokke Fredborg
Jonas Halle
Jørgen Ringdal Sæter
Andreas Øie

Towed ROV

May 2021

Bachelor's thesis

NTNU
Norwegian University of
Science and Technology

Sophus Stokke Fredborg
Jonas Halle
Jørgen Ringdal Sæter
Andreas Øie

Towed ROV

◼ NTNU
Norwegian University of
Science and Technology

NTNU

Kunnskap for en bedre verden

Towed ROV

A BASIS FOR BUILDING A OPEN-SOURCE TOWED ROV FOR SEAFLOOR SURVEYING

Candidates

Sophus Stokke Fredborg

Jonas Halle

Jørgen Ringdal Sæter

Andreas Øie

20 May, 2021

Preface

This bachelor thesis is written by four students from Automation Engineering at NTNU Ålesund. The students in the group come from a number of backgrounds, ranging from automation to sports.

In this thesis, we wanted to renew the Towed ROV-project in terms of rewriting the fundamental software systems and simplify the current hardware. By rewriting the software, we would like to implement a graphical user interface with a database so that researchers could operate the ROV to collect exciting data from the sea through hydrographic surveying, taking pictures or other measures to collect useful sensor data.

The collected data could possibly lead to the discovery of previously unknown fishing gear on the seafloor, which could be useful in the battle against oceanic ghost fishing.

We recommend that the reader have a basic understanding of engineering, software technology, and automation in order to fully comprehend the content of this bachelor thesis.

Acknowledgement

We'd like to express our gratitude to all of the people who helped and supported us during this project, and particularly to:

- Supervisor Ottar L. Osen at NTNU for help and guidance throughout the project.

- Assistant supervisor Robin T. Bye

- Assistant supervisor Øystein Bjelland

- Runde Miljøsenter for their hospitality and cooperation.

- Laboratory engineer Anders Sætersmoen for support with the purchase and lending of equipment.

- The Product and System Design -bachelor group consisting of Endre Myhre and Katherine Galdames for cooperation in the Towed-ROV project.

- Andreas Fredborg for support in relation to hydrodynamics.

- Friends and family who have supported us throughout the project.

Summary

This project aims to improve the continuation of the Towed ROV -project and reestablish a new foundation for future work. The goals include a new software system for operating the ROV, with implemented functions necessary to perform hydrographic surveying. In addition, the project includes the implementation of a Digital Twin to simulate the physical behaviour and the algorithms necessary to regulate above the seafloor.

The results indicate a stable software system with the functionalities required to search and locate objects in the ocean. The system implementation relies on the three-tier architecture, which will help integrate future additions to the project. By testing physical changes in several sea trials, the group have addressed some of the core concerns related to the prototypes hydrodynamical behaviour and the actuator system for the wings.

However, more work is needed to provide a physical prototype capable of truly operating in research environments. Improving the prototype's physical design is needed to reach the depths required to conduct proper seafloor tracking. Furthermore, the design induces unwanted physical behaviour, making it inadequate for hydrographic surveying.

Contents

Abbreviations

SSE Server-sent events, server-side pushing data to client in terms of events

ES EventSource, web content's interface to server-sent events

SR StreamingResponse, special FastAPI response method

ZMQ ZeroMQ, communication networking library

HTTP Hypertext Transfer Protocol

UDP User Datagram Protocol, non connection oriented

TCP/IP Transmission Control Protocol, connection oriented

API Application Programming Interface, activates functions from a remote software

GUI Graphical User Interface

PID Proportional integral derivative controller

IEEE Institute of Electronical and Electronic Engineers

I2C Inter Integrated Circuit

GND Ground in electronical circuits

DOF Degrees of Freedom, number of configurations for a object

ORM Object–relational mapping, converting data between incompatible type systems

PCB Printed Circuit Board

UART Universal asynchronous receiver-transmitter

SONAR Sound Navigation and Ranging

GPIO General-purpose input/output

PoE Power over Ethernet

NMEA National Marine Electronics Association

AOT Angle Of Attack

JSON JavaScript Object Notation, standard data format

IMU Inertial Measurement Unit

Terminology

CRUD Create, Read, Update and Delete (HTTP methods)

Business Logic , term used for the important logic decisions in the software systems

Hydrofoil A wing-shape that creates a large amount of lift in water. separation of concerns

Separation of Concerns , a design principle: used for separating the software programs into own section of responsibilities

Notation

K_p Proportional term of a PID controller

K_i Integral term of a PID controller

K_d Derivative term of a PID controller

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

This project aims to develop a Towed ROV that can be used for monitoring the seafloor and collecting data by a range of sensors. The ROV can be deployed to various tasks like mapping the seafloor, collecting sub-sea sensor data and locate objects.

The project is financed by Handelens Miljøfond/NTNU and collaborates between Runde Miljøsenter and NTNU Ålesund, where the main objective is to locate ghost fishing gear. For decades, ghost fishing had been a significant problem, and according to a study released by WWE [115], fishing waste accounts for at least 10% of marine litter.

The removal of ghost fishing gear is often done by dredging an anchor at low speed by the seafloor. Since the anchor may cause damage, knowing where the gear is located is critical for a safer, effective and accurate operation.

A Towed ROV have several advantages compared to an AUV in terms of cost and distance covered. The Towed ROV does not need a complex navigation system or thrusters to manoeuvre. Instead, it relies on a power supply from the boat, letting the system operate for a longer time.

1.2 Problem Formulation

The project aims to improve and simplify various systems of the ongoing Towed ROV -project. The formulation aims to construct a fundamental system architecture. The emphasized project areas involves minimizing overall cost, increased convenience of usage, and a modular and adaptable base unit.

1.3 Project Requirements

The critical project requirements of this thesis are to rewrite the complete software systems and simplify the current hardware so that future groups can focus on physical design, control, and data processing. Furthermore, we will do some work specifically for the existing prototype, including sea trials. The requirements could be summarized in the following list below.

- Create a new GUI with added functionalities

- Build a new software structure for reliable communication

- Research and buy equipment to perform hydrographic surveying.

- Test software and improve control system on a new prototype

- Evaluate and improve the hydrodynamic behaviour of the ROV.

- Develop a seafloor tracking algorithm

- Improve the Digital Twin.

1.4 Thesis Outline

The structure of the remaining thesis is as follows:

Chapter 2: Theoretical basis Entails the fundamental theory required to understand various aspects of the project.

Chapter 3: Materials Involves the various hardware and software materials used.

Chapter 4: Methodology Entails the strategies for implementing the tests and solutions of the project.

Chapter 5: Results Presents all the test results and the final solution.

Chapter 6: Discussion Shows the various discussions regarding the results as well as personal thoughts about the thesis.

Chapter 7: Conclusion Entails the an overall conclusion from the project in total.

Chapter 2

Theoretical basis

This chapter provides the theoretical foundation for decision-making throughout the project.

2.1 Inertial measurement unit

An IMU is a combined set of sensors used to measure linear and rotational movement and estimate a rigid body's orientation [98]. The orientation is often represented in the three Euler angles, roll, pitch and yaw [60]

2.1.1 Accelerometer

A 3-DOF accelerometer measure linear acceleration in three perpendicular axes relative to the IMU. The measurement of one axes is described in equation 2.1, where $\tilde{a}$ is measured acceleration, a_g is gravitational acceleration, a_l is linear acceleration and n is noise [98].

$$\tilde{a} = a_g + a_l + n \tag{2.1}$$

With three acceleration vectors, one can estimate the angle of the IMU by applying the known gravity vector. Estimating with gravity vector can only work for roll and pitch, as yaw is rotation around the gravity vector. The angles can be calculated by applying atan2. Equation 2.2 for roll and 2.3 for pitch. The accelerometer gives an accurate long term estimation because there are no drift and the center of gravity is a stable reference. However, the measurements are

noisy and are affected by motion giving an inaccurate short term reading.

$$\theta_r = -atan2(-a_x, a_y) \tag{2.2}$$

$$\theta_p = -atan2(-a_z, sign(a_y) \cdot \sqrt{a_x^2 + a_y^2} \tag{2.3}$$

2.1.1.1 Gyroscope

A 3-DOF gyroscope measures the angular velocity of the IMU in the angles of pitch, roll and yaw. Equation 2.4 is a model of a gyro measurement, where $\tilde{\omega}$ is the measured angular velocity, ω is the actual angular velocity, b is bias, and n is noise. The bias is temperature-dependent but can be approximated as a constant, while the noise is additive zero-mean Gaussian noise [98].

$$\tilde{\omega} = \omega + b + n \tag{2.4}$$

2.1.1.2 Magnetometer

A magnetometer measures the magnetic induction and can be used to find the magnetic north and measure magnetic fields. The magnetometer can be used to calculate rotation around the gravity vector. For a system that relies on yaw, it is essential to calibrate the magnetometer. There can be distortions due to metal and electronic objects [98].

2.2 Complementary Filter

The idea of the complementary filter is to combine a high-pass and low-pass filter to remove drift from the gyro and noise from the accelerometer. The filter is described mathematically in equation 2.5, where θ is the Euler angle, θ_a is the Euler angle from the accelerometer and magnetometer fusion and $\tilde{\omega}$ is the angular velocity. α is the filter coefficient and is in the range of $0 \leq \alpha \leq 1$ [98].

$$\theta^t = \alpha \cdot (\theta^(t - 1) + \tilde{\omega} \cdot \Delta t) + (1 - \alpha) \cdot \theta_a \tag{2.5}$$

2.3 Hydrographic Surveying

Hydrographic surveying is the science of collecting depth data about the features of the seafloor[3].

2.3.1 Echo Sounding Thechnology

An echo sounder is a sound-based sensor type primarily used in finding distance but sometimes used to collect data from the surface of a solid [69]. Some of the most common uses of echo sounder technology include surveying the seafloor, use as a fish finder or surveying the open sea [9].

2.3.1.1 Single-beam sonar

Single-beam sonar transmits a single wave to measure the distance to, for example, the seafloor [62]. Most depth sonars, are single-beam echo sounders. A single-beam echo sounder calculates the distance to the seafloor with regards to time with the formula 2.6:

$$depth = \frac{S_w \cdot \Delta t}{2} \tag{2.6}$$

2.3.2 Surveying with sonar technology

Surveying using sonar technology is today a common practice. There are many different technologies based on this, including side-scan sonar and multi-beam-sonar.

2.3.2.1 Side-scan sonar

Side-scan sonar is a sonar system used to collect data and image the seafloor using a high-frequency sonar transducer to send fan-shaped soundwaves in a thin line towards the ground and record the return wave's time, then using that to create a picture of the seafloor. With side-scanning transducers, the image's resolution is determined by the frequency of the soundwaves and the, Traditionally soundwaves between 100Khz and 700 kHz are used. Higher frequency gives a better resolution on the image but reduces the range of the transducer [95]. What makes

side-scan sonars unique from other sonar surveying technologies is their ability to build a high-resolution image of the seafloor [64].

2.3.3 Hyper-Spectral imaging

Hyper-spectral imaging (HSI) is a new type of imagery technology that works similarly to multibeam-sonar, but instead of using sound, the hyper-spectral imaging uses light. An HSI sensor transmits a light beam onto a surface and measures using a camera that can collect the data from different parts of the electromagnetic spectrum [66]. Since different kinds of materials reflect different parts of the electromagnetic spectrum, a large amount of data about a surface can be collected this way [59].

HSI has recently started to be used to map the seafloor [68]; it provides more information and higher accuracy and precision than traditional seafloor mapping technologies but covers a smaller area at an increased price.

2.4 Software architecture

When creating new software systems, it is common to use a set of design patterns, and best practices are to allow for scalability. The subsections below show various popular architectures.

2.4.1 Three-Tier Architecture

A three-tier architecture [28] is a common software architecture for creating logical and scalable software solutions. The architecture is divided into three tiers: the presentation tier, the application tier and the data tier.

- **The presentation tier** is the tier where the data is displayed to the user, so-called the view.

- **The application tier** is the tier where the business logic is located. This tier considers all the systems that handle the communication between the various entities in the software systems.

- **The data tier** is the tier where information gathered by the software system is stored and controlled, commonly referred to as the "database tier".
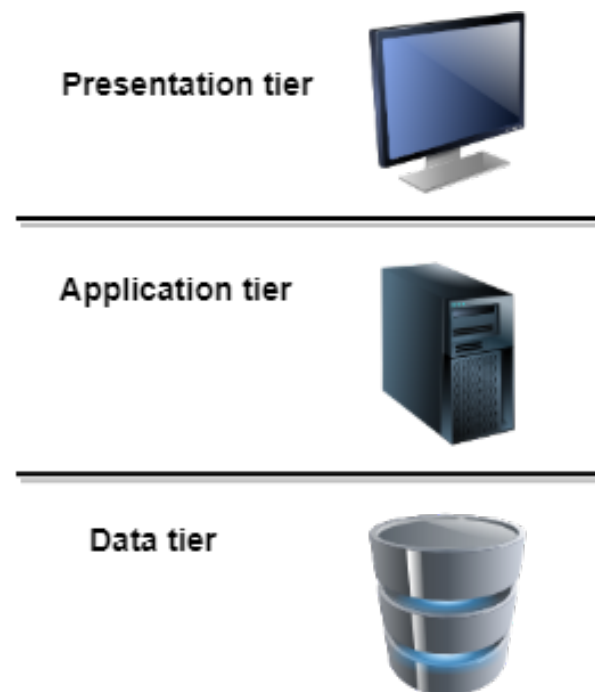
Figure 2.1: The three-tier architecture

A three-tier architecture helps to distribute responsibility in the software systems. Dividing responsibility decreases coupling and increases cohesion as the separation of concerns is divided into tiers. Each tier has its function and responsibility.

2.5 Database

In order to maintain persistent data throughout an application, one would often need to write the data from memory into disk allocated space. A persistence data storage would be a database. A database can be described as a collection of information ordered in a structural and representative way [49].

2.5.1 Relational Database

A relational database [70] is a type of database, which relies on a structure where data is put into relations to each other.

SQL is an example of a relational database. This type organizes its data in tables with rows and columns to describe the content of the information. In order to make the data accessible, the SQL database contains a few features to make it easier to access the data logically. Below are a few key concepts in a SQL database [106].

2.5.2 Primary Key

A primary key [111] is the main column in a row, which uniquely defines that row, which means that there cannot exist identical types of its sort.

2.5.3 Foreign Key

The foreign key [110] is used as an identifier in a 'child' to its 'parent'. In order to create a relationship between tables, one would often want the 'parent' table to contain some context information not described in its table but a separate one. In the separate table, one could then insert a foreign key - pointing to its parent.

2.5.4 Relationship

This section briefly explains the different ways of defining a relationship between tables and how they are designed. There are three different kinds of relationships; one-to-one, one-to-

many, and many-to-many [71].

2.5.4.1   One-To-One

In a one-to-one relationship [71], we only have a single connection between two tables. A single connection would mean that one column in one table points to a column in another table in database terms. For instance: if we had a table called Adults, where one row would represent one adult, we could have had another table called DriverLicences, where each row represented a driver licence. Then, one could have a one-to-one relationship between those two rows since there only exist one driver licence for that adult.

2.5.4.2   One-To-Many

One-to-many [71] is more of the more common types of relationship. A single connection/row could have a relationship to other rows in other tables in general terms. If we expand the previous example, an adult/mother could have many children, but not the other way around.

2.5.4.3   Many-To-Many

In a many-to-many relationship, [71] things can be designed a bit more freely. If you think of tables of A and B, where A is authors and B are books, we can say that an author can write many books, but a book could have been written by multiple authors.

2.6   Application Programming Interface

The application programming interface, generally known as API [81], is a software intermediary which connects other software units.

2.6.1   Representational state transfer

A REST-API [112] is an API interface for the web that uses the HTTP protocol. It utilizes the HTTP request methods for different functionality to various endpoints. In most situations, web applications primarily use the HTTP methods: GET, POST, PUT and DELETE.

2.6.1.1 HTTP methods

Below, we will clarify the four most commonly used HTTP methods [112].

- **GET** - action to retrieve data from the server.

- **POST** - action used to create/store some data in the database.

- **PUT** - action to update a data in the database.

- **DELETE** - action to remove data from resource.

2.6.1.2 Endpoint

Endpoints are essentially URL paths that are open for communicating from external sources [85]. These endpoints or paths tell what resource lies behind them. For instance: **http://127.0.0.1:8000/users/**[1] is an endpoint. Where all calls made to this endpoint would trigger an action depending on the HTTP method used.

2.6.1.3 Path parameters

A path parameters [85] is described as additional path direction in an API endpoint. Where usually an item in a database has an ID, we could use the ID as a path parameters to endpoint request to narrow the search. For instance, using an endpoint of users, we could write the following; **/users/{id}** , where the curly braces denote the path parameter.

2.6.1.4 Query parameters

A query parameter [85] is used to describe how an action should be executed in the HTTP method. An HTTP method could be a GET call where a query parameter could contain variables to filter the search. For instance, **/users/?age=25** , would return use all users with the age 25.

---

[1]Throughout the thesis, the host and port prefix will no longer be used when referring to endpoints. Thereby **http://127.0.0.1:8000/users/** is equal to **/users/**

2.6.1.5 StreamingResponse

A StreamingResponse [75] is a FastAPI [74] specific method of taking an generator/iterator and streams the response body. A StreamingResponse could be used inside endpoints to return large amounts of data as a stream of bytes.

2.6.1.6 Server-Sent Events

Server-Sent Events [88] is a type of data transmission technology that uses server pushing to send data from the server to the client. The client receives automatic updates via a single HTTP connection. In contrast to REST which uses the request/reply pattern, server-sent events are solely initiated from the server. Therefore, the client would only need to listen for incoming events to specific topics to receive continuous streams of data generated by what is called a *EventSourceResponse*.

2.7 Communication Protocols

2.7.1 TCP / IP

The transmission control protocol (TCP) is a networking protocol above Internet protocol(IP) which ensures reliable network communications [114]. For TCP to be reliable, it provides a mechanism for lost, out of order, or corrupt packages, which are all problems that can occur in network communication. All packages in TCP includes a sequence number and an acknowledgement number, and the receiver must acknowledge all sent packages. It is common to implement a separate protocol over TCP, the application-specific protocol between a server and a client. HTTP is an example of such a protocol.

2.7.2 Serial communication

Serial communication is a group of communication protocols where data is sent one byte at a time instead of parallel communication. Serial communication is the most common type of communication protocol and includes well-known protocols like USB, Ethernet, I^2C [5] In

serial communication, a system is split into senders and receivers. There are three main ways of handling data flow [21]:

- **Simplex** means that a connected device can be a sender or a receiver. Simplex is one-way communication, and there is no possibility for acknowledgements or communication back and forth. If a device is a sender, it cannot receive data, and if it is a receiver, it cannot send data. A common example of simplex communication is radio communication.

- **Half-Duplex**, sender and receiver can be active at the same time. Therefore the receiver can send data back to the sender, like acknowledgements, but cannot transmit data simultaneously.

- **Full-Duplex**, both sender and receiver can send and receive data at the same time. A typical example of full-duplex communication is the Ethernet protocol.

2.7.2.1 NMEA 0183

NMEA-0183 is a serial communication protocol. The protocol is simplex based and can therefore only have one sender designated as the Talker in the protocol, but can have many listeners, with a baud rate of 4800 Mbps [1].

The protocol transmits serial data as "ASCII" encoded strings. A data packet starts with the '$' character [2]. After the start character, a five-character identifier follows, where the first two characters define the Talker ID, and the next three defines the sensor type [4]. Following is a variable-length field array containing sensor data; each field is denoted by the ',' character and the packet ends with a checksum denoted by the '*' character. The checksum is calculated as an XOR of the provided data. To calculate it, parse out the start ($) and the stop signal (*) of the NMEA sentence and perform an xor calculation of each element:

---

[2]the '!' character is sometimes used as well, especially in the NMEA-0183HS version

NMEA 0183 Structure									
Start Character	Sensor type	Data type	value 1	value 2	value 3	value 4	value 5	value 6	Check-sum
$	SD	DBT	10	f	10	m	10	F	F*29
	sonar	depth data	depth	unit	depth	unit	depth	unit	XOR

Table 2.1: NMEA sentence structure.

2.7.3 I^{2}C

The Inter-Integrated Circuit (I^{2}C) protocol is used to allow multiple integrated circuits (chips) to communicate with each other. I^{2}C is made for communication over a short distance. This distance can vary due to noise, I^{2}C clock speed etc. It also allows for multiple masters and slaves [82].

2.7.4 Software UART

Software UART is a software replication of the serial UART allowing UART communication on hardware not supporting Serial UART. Software UART is possible through "bit-banging (creating a series of pulses in software rather than in hardware). Bit-banging is more processor consuming and not as precise as serial UART [61].

2.8 ZeroMQ

ZMQ is an open-source, cross-platform asynchronous messaging library [45] developed by many contributors. The library focuses on abstracting the low-level socket communication into common messaging patterns such as pub/sub or request/reply, on mentioning a few. The messaging patterns are based upon various types of transports such as TCP, UDP and IPC. The libraries API has support for many different program languages such as Python, JavaScript, C++, to mention a few.

2.8.1 Request–reply

Request–reply [44] is a common basic pattern supported by ZMQ. This protocol connects a set of clients to a set of services. This pattern, in particular, is used to task for a certain task to be distributed, much comparable to the common REST pattern. A *requester* requests an action to be made, while the *receiver* receives the requests and replies with an response.

2.8.2 Publisher–subscriber

Publisher–subscriber [43] is a common basic pattern supported by ZMQ. This protocol can connect a publisher to a set of subscribers. This pattern is pipelined to distribute data through a stream purely. The data is optionally categorized into topics where so that subscribers can connect and listen to different topics.

2.9 Digital Twin

A digital twin is a representation of a system or physical object in a digital simulation. This digital representation takes in real-time data about a physical object or system as input and computed a predicated output of the behaviour of the physical object or system. A digital twin can serve as a prototype testing different versions and iterations before a physical version is built. A digital twin can also be used alongside a physical system to get a visualization of a system which would not be possible without the digital twin or to test function before they are implemented in the existing system [83].

2.10 Pulse Width Modulation

PWM is a way to seemingly create an analogue signal with a digital signal to control an analogue device [41]. PWM is possible is to apply power in pulses to the output instead of constant power (digital signal only high or low).

The output voltage will be the average voltage, which is the percentage of the pulse length in

a given period; see equation 2.7. If the digital signal (5 volts) is on 50 % of a period, the average voltage out is 2.5 volt.

$$\text{Duty Cycle} \cdot \text{High voltage level} = \text{Average voltage} \tag{2.7}$$

2.11 Geographical terms

2.11.1 Haversine formula

The Haversine formula can calculate the distance between two points on a circle along the surface of the circle using the latitude and longitude provided [97]. Given the two points in latitude and longitude, the distance d can be calculated as the inverse of the haversine h. When calculating the distance, the radius r of the circle is important. The haversine is defined as $h = \dfrac{d}{r}$. d can be calculated as follows [78]:

- **lat1** = latitude of point 1

- **lon1** = longitude of point 1

- **lat2** = latitude of point 2

- **lon2** = longitude of point 2

- **R** = 6371e3; // metres

ϕ *is latitude,* δ *is longitude.* **R** *is earth's radius (mean radius = 6,371km).* $\delta\phi$ *is the change between latitudes while the* $\Delta\delta$ *is change in between longitudes with* ϕ *and* δ *described in radians.*

$$a = \sin^2(\frac{\Delta\phi}{2}) + \cos\Phi_1 \cdot \cos\Phi_2 \cdot \sin^2(\frac{\Delta\delta}{2}) \tag{2.8}$$

Thereafter, we use the change of latitude and longitude into 2.8, which gives use a, where a denotes the square of half the chord length between the points.

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)}) \tag{2.9}$$

or the alternative:

$$c = 2 * \arcsin \sqrt{a} \tag{2.10}$$

where c from either 2.9 or 2.10 describes the angular distance in radians.

$$result = R \cdot c \tag{2.11}$$

Which together gives us the distance between the two coordinates in metres (as the angular distance in radians is rescaled with the earth's radius R).

2.11.2 Earth's radius

When calculating distances using the Haversine function on earth, it is crucial to consider that the earth's radius is not constant. The shape of the earth is not that of a globe but that of an oblate spheroid. The radius varies from around $6378 Km$ at the equator to $6357 Km$ at the poles [94]. Due to the various in radius, the distance travelled by an object along the earth's surface between two equally distant points might be different depending on the latitude of those points. Therefore the radius of the earth needs to be calculated at the location that the calculation will take place with the formula 2.11.2.

$$r(\theta) = \sqrt{\frac{(a^2 \cdot \cos\theta)^2 + (b^2 \cdot \sin\theta)^2}{(a \cdot \cos\theta)^2 + (b \cdot \sin\theta)^2}} \tag{2.12}$$

2.12 Hydrodynamics

Hydrodynamics is the science of the flow of fluid and how it affects its surroundings. In hydrodynamics, a fluid is viewed as its flow patterns; fluids flow at different speeds at different locations. Sometimes the flow of fluids is parallel, but fluid flows are more often chaotic and frequently form whirlpools [42].

2.12.1 Forces on a body in water
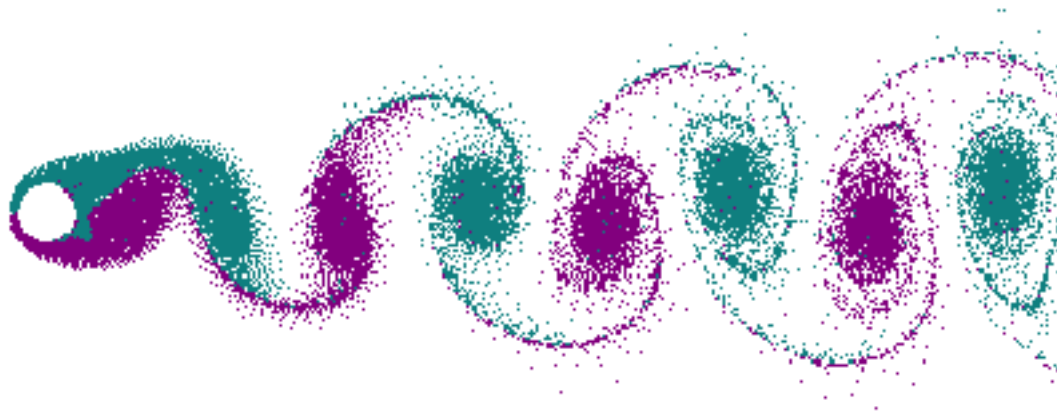
Figure 2.2: Simulated vortex shedding, by Cesareo de La Rosa Siqueira [113].

2.12.1.1 Drag and inertia

The dampening primarily comes from drag and inertia. The drag forces can be described as friction between the object and the fluid. Inertia forces can be described as the forces related to the acceleration of the mass of the water that the body would displace. The drag and the inertial forces act on the body in a direction opposite to its movement [89].

Drag forces are dependent on the shape of the body, as well as its surface roughness. Inertia forces are dependent on the volume.

2.12.1.2 Turbulence: shapes and vortex shedding

Turbulence adds complexity to the problem. Turbulence is, in essence, the absence of laminar flow. The most significant effect of turbulence on an ROV comes from vortex shedding or separation of flow caused by sharp edges, abrupt corners, or other non-aerodynamic features on the body [42]. These forces can have a self-amplifying effect if the separation of flow occurs at regular intervals between different features or to alternating sides of a shape where the flow can pass on both sides. This effect is called vortex shedding [22].

2.12.2 Stability and buoyancy

For a fully submerged rigid body, the center of gravity (CG) will always lay directly below the center of buoyancy(CB). If the center of gravity is shifted, the rigid object will rotate, so the statement above is fulfilled [63]. For a rigid body, this is shown in Figure 2.3. Adding positive buoyancy to the center-top and more weight to the center-bottom, the stability of a rigid body will improve because more force has to be applied to rotate a rigid body [63].
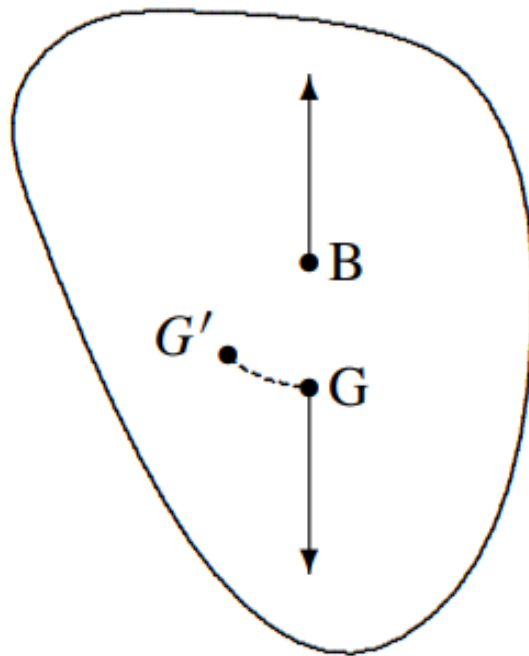
Figure 2.3: A fully submerged rigid body [63]

2.12.3 Hydrodynamics and Towed vehicles

The Hydrodynamics of a Towed ROV can often be complex, but the principles are simple. A Towed ROV needs to be stable in the water, with minimal unintended changes in pitch, roll, yaw or position. The combined hydrodynamic and hydrostatic forces acting on the body, as well as the orientation of the center of buoyancy and the center of mass, determine the ROV's stability [29]. The forces are determined by the ROV's speed and shape, fluid properties, buoyancy, and mass. The hydrodynamic and hydrostatics are dominant at high speed, while at low speed, buoyancy and gravity play a more critical role [31]. When an object moves relative to the water, the object's shape will create turbulence, but sharper edges will create more vector shedding and therefore have a more pronounced effect on the body, as seen in Figure 2.4.
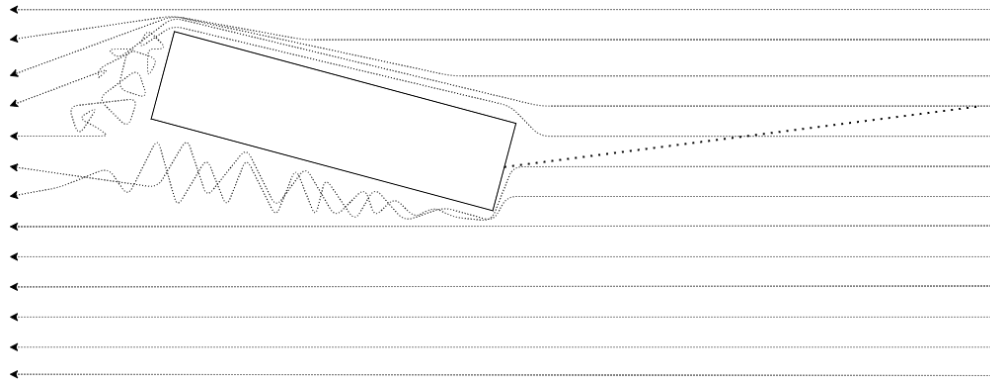
Figure 2.4: A Towed object disturbs the water flow around it

2.12.4 Hydrofoil

Foil is a wing or flipper that produces both a pressure and a sucking force on the wing as it moves through a fluid [72]. The laminar flow around the wing creates these forces. As seen in Figure 2.5 the flow around the wing is compressed on one side, while on the other side creating a sucking force. These disturbances in flow around the wing create an imbalance in pressure above and below the wing, resulting in a force in a different direction compared to the flow. This force is called lift.
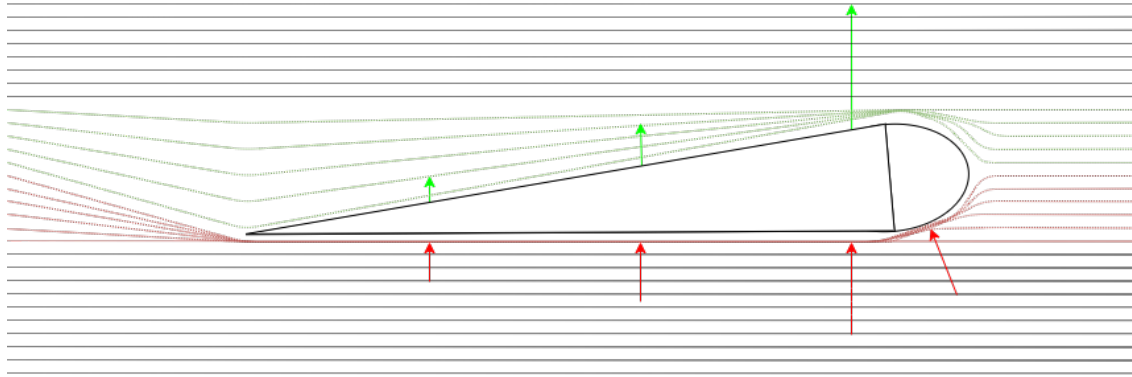
Figure 2.5: Waterflow is displaced around a wing, creating pressure and sucking forces.

2.12.4.1 Stall

In hydrodynamics, the stall is the loss of lift force due to the wing's angle of attack (AOT) [72]. If the AOT is too steep, the turbulence created by the wing disturbs the standard flow patterns. In this case, the wing loses its pressure vs sucking imbalance and instead operates more like a brake; This will still generate a force on the wing due to the pressure of water being displaced, but this force is often smaller than the forces created by a hydrofoil [80].

Chapter 3

Materials

3.1 ROV prototype

The prototype used in this project was built in a previous project, Figure 3.1. The body is made of aluminium with a total weight of 34 kg and a positive buoyancy in the water of 120N. The depth is controlled by linear stepper motors rotating a wing on each side.

The ROV has been tested with a 90-meter long tether cable and were controlled by a PID with a range of 8-12 meters of depth. The electronics can be accessed through a flange at the bottom. Each wing is rotated by a linear stepper motor moving a gear rack which rotates a gear, rotating the wing. Side plates are designed to make the ROV more stable but have yet to be tested. The tether cable is attached to a cable anchor and secured with a bolt; the cable anchor is adjustable. There are penetrates on the back of the ROV for entering cables. The NACA 0015 profile inspired wings will give no lift when the angle of attack is 0 degrees. In the front, there is a camera housing with a plastic dome. There are mounted one light on the side of the camera.
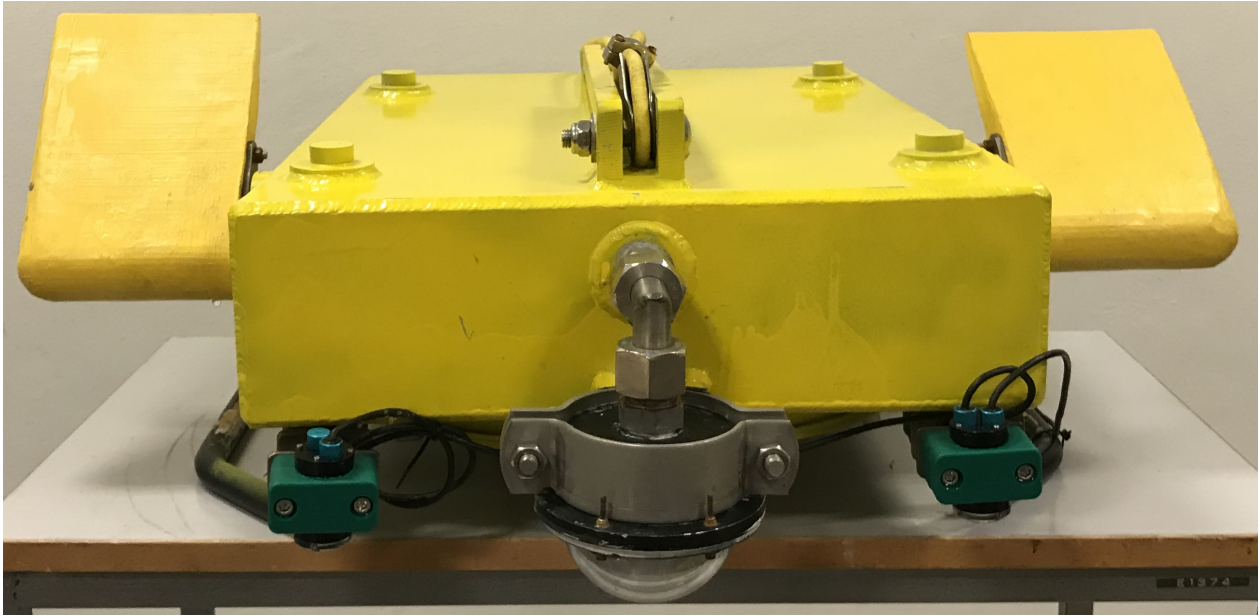
Figure 3.1: ROV prototype

3.2 Hardware

3.2.1 ROV

- SparkFun RedBoard

- Arduino Mega 2560

- Raspberry Pi 4

- Adafruit 9DOF IMU

- Bar30 depth/pressure sensor

- Ping sonar altimer/echo-sounder

- RS PRO 42DBL10 linear stepper motors x2

- Pololu md20b stepper driver x2

- Fathom-X Tether interface board

- I2C level converter (Blue robotics)

- Micro servo motor

- Low-Light HD USB (Blue robotics)

- Lumen Subsea Light (Blue robotics)

- Custom PCB card, 70-18V input, 5V and 12V output

- Food oil, non electrical conductive [73]

3.2.2 Surface Unit

- Raspberry PI 4

- Aimar DST800 echo-sounder

- Adafruit ultimate GPS

- Optocoupler

- Fathom-X Tether interface board

- Ethernet switch

3.2.3 Power and cables

- 5 x 12 volt battery

- Tether cable (90m)

- Tether cable (150m)

3.3 Metals and Plastics

- Acrylic sheet 6mm

- Polyvinyl chloride/polypropylene pipes

- Perforated fixing strap

3.4 Tools

- Oscilloscope

- 3D Printer

- Laser cutter

- Multimeter

3.5 Software, libraries and frameworks

3.5.1 Programming Languages

3.5.1.1 Python

The Python programming language [99] is a high-level, general-purpose interpreted language. The language has been selected due to the simple interface for building, testing and communicating with embedded systems. It is based on a dynamically typed style and handles garbage collection automatically. The simplicity makes it so interacting with various hardware components through third-party libraries such as pyserial 3.5.3.1 improves test efficiency significantly.

3.5.1.2 JavaScript

JavaScript is a high-level programming language mainly used in web development. It is included in every browser and is responsible for making more advanced and dynamic web pages. NodeJS is a runtime for JavaScript, which makes it possible to run JavaScript on the server. It uses the V8 engine developed by Google, the same JavaScript engine used in the Chrome browser [102].

3.5.1.3 SQL

Structured Query Language is a standardized language used by relational databases for accessing the database. Because SQL exists in various dialects, it is a good fit for an open-source project

like this. For many languages, including Python, SQL provides a variety of implementation options. Python will be used to implement SQL because it is the primary language for the project's software development [106].

3.5.1.4 C / C++ Arduino

Arduino C/C++ is a C/C++ extension for the Arduino microcontrollers. It implements the C/C++ programming language but adds constants, structures, sketches and functions to make it easier to use the abilities of the Arduino microcontroller. Since it is based on the C++ programming language, various C++ libraries can be used as well [14],[15].

3.5.2 Software utilities

3.5.2.1 Microsoft Teams

Microsoft Teams, a common popular multi-communication platform offering workspace chat and videoconferencing, file storage, and application integration [101].

3.5.2.2 Discord

Discord, a Multi-communication platform offering voice and text communication through chat rooms and voice chat channels where the users can also share their screen recordings, files and other documents [109].

3.5.2.3 Visual Studio Code

Visual Studio Code is a lightweight multi-languages supportive code-editor [108].

3.5.2.4 PyCharm

Pycharm is an IDE for the Python programming language. It provides integrated functionalities, handles virtual environments automatically and comes with built-in developer tools [104].

3.5.2.5 AGX Dynamics

AGX Dynamics is a physics engine that can simulate rigid bodies, collisions, wire control, aero and hydrodynamics [7]. It supports Python, C# and C++. AGX can work with modular terrain that can be actively modified by physics during runtime.

With the possibility to import 3D models and use pre-built hydrodynamics to simulate the Towed-ROV behaviour in water. AGX also have a simulation of wires, working with hydrodynamic and reports the tension and forces in the wire.

3.5.2.6 Arduino IDE

The Arduino IDE is used to write software to the Arduino Microcontrollers. It supports and is written in C / C++ [16].

3.5.2.7 Solidworks

Solidworks is a 3D CAD design software used to simulate drag in ROV [24].

3.5.2.8 Simens NX

Simens NX Is a 3D modeling software, used to model simplified ROV models for simulation [84].

3.5.2.9 Virtual Serial Port Driver

This software allows a computer to create virtual COM ports for sending serial messages within the computer [2]. It can create pair of COM ports where one end is one application communicates with another application connected to the other COM port in the pair.

3.5.2.10 Autodesk Fusion 360

Autodesk Fusion 360, 3D CAD design software used for 3d modeling [17].

3.5.2.11 Cura

Cura, free slicing software for 3D printing from Ultimaker [96].

3.5.2.12 Fritzing

Fritzing, Open-source CAD software for designing electronics hardware (Schematic) [40].

3.5.2.13 VNC Viewer

VNC Viewer, a graphical desktop-sharing system used to control Raspberry pi over Ethernet [77].

3.5.2.14 iPerf

Software analyzing tool for measuring maximum achievable bandwidth on IP networks [27].

3.5.2.15 Matlab

Programming and numeric computing platform and is used in this project for analyzing and presenting data from sea-trials and simulations [93].

3.5.2.16 Swagger UI

Swagger UI [86] is a user interface that allows for REST-API visualizing and testing (bundled with FastAPI).

3.5.2.17 TablePlus

TablePlus is a GUI database management tool to help visualize and interact with data more efficiently [92]

3.5.2.18 Draw.io

Draw.io, an diagram drawing software, used for drawing flowcharts, UML diagrams etc [13].

3.5.2.19 DeepView FV

DeepView FV is a free software to display side-scan sonar images (.dvs-files) created by DeepVision AB [8].

3.5.3   Libraries and Toolkits

3.5.3.1   Python

- **OpenCV** is a computer vision library related to image related real-time operations [103].

- **PyZMQ** , the official python bindings for the ZMQ messaging library [51].

- **SQLAlchemy** is a tookit and ORM which helps with SQL database interactions [20].

- **Pydantic** is a data validation and management tool [47].

- **Uvicorn** a ASGI server implementation [67].

- **pyserial** is a library for accessing and handling communication over serial ports using Python [65].

- **PyNMEA2** a package for parsing nmea0183 data from bytes to strings[30]

- **Numpy** , a scientific computing library [23].

- **Adafruit_gps** a library provided by the producers of the Adafruit GPS which makes it easier to work with the GPS dat [57].

- **Multiprocessing**, a package used for managing multiprocessing [39] concurrency with tools like:

    - **Process**, represent activity that is run in a terminal/process [33].

    - **Queue**, a queue used for information exchanged safely between multiple processes [34].

    - **Event**, a flag-object used as a boolean flag [32].

- **Threading**, a package used for managing threads [38] concurrency with tools like:

    - **Thread**, represent activity that is run in parallel with the main process [35].

    - **Queue**, a queue used for information exchanged safely between multiple threads [37].

    - **Event**, a flag-object used as a boolean flag [36].

3.5.3.2 JavaScript

- **React-Router** is a component library which helps for navigating between pages (URLs) [76].

- **Axios** is a modern HTTP API web frontend client [18].

- **Chakra UI** , an modular component library for developing better user experiences [11].

- **Highcharts** is a multi-platform charting library for displaying data [48].

- **Leaflet** , an open-source, lightweight library used for interactive maps [12].

3.5.3.3 C / C++

- **Zeromq**, the original ZMQ messaging library in windows x86 architecture [46].

- **EthernetSonarAPI**, interface library from DeepVision (x86 arc) (see Appendix [E]).

3.5.3.4 Arduino

- **Blue Robotics ping-arduino**, for communication with the Ping sonar [53].

- **Blue Robotics MS5837**, for communication with the depth/pressure sensor [54].

- **Adafruit 9dof**, for estimating orientation [10].

3.5.4 Frameworks

3.5.4.1 GUI

- **Electron** is a open-source software framework which enables the creation of desktop GUIs by the use of modern web technologies [100].

- **React** is a web frontend framework in Javascript for building user interfaces applications [105].

- **NodeJS** is a cross-platform runtime environment for running Javascript outside a web browser [25].

- **CRA** , create-react-app, a cross-platform bootstrap for rapid React development [55].

3.5.4.2   API

- **FastAPI** is a modern, high performance web framework for building APIs in Python [74].

Chapter 4

Methodology

4.1 Project Organization

The group consist of four bachelor students with a similar background. To ensure structure and responsibilities amongst the members in the group, three disciplinarian positions were initiated. We divided positions into one group lead, one software lead whilst the third secretary role was rotated.

The group leader has the responsibility to ensure cooperation, time management and delegating various tasks. The leader also has the responsibility to clear up any conflicts that may arise. In terms of time management, the group leader is responsible for organizing internal meetings with the group and external meetings with the management group.

The software leader has the responsibility to ensure a standard code style for the programming languages used and the responsibility to make top-level decisions to ensure the functionality needed for the software solutions is matched.

The secretary has the responsibility to take notes during meetings and maintain the structure of how the various documents for report, meetings and discussions are written and stored in Microsoft Teams 3.5.2.1.

Internal meetings were scheduled anywhere between 1-2 week, depending on the workload and issues during the project. In the early stages of the project, the meetings consisted of testing our various implementation strategies of the hardware and software systems. Later meetings consisted of discussing the weekly progress made to the systems and various required technical improvements.

External meetings were scheduled from anywhere between 2-3 weeks, depending on the project's progress and various complications. In the early stages of the project, the meetings consisted of seeking advice and discussing implementation strategies with the management group. Later meetings consisted of discussing the current system implementation and the results made from testing. Discussions focused on the performance and behaviour of the ROV in terms of physical sea trials as well as corresponding simulated scenarios.

For further details regarding the Project Organization, please see Appendix [A.1].

4.2 Deviation Protocol

As this thesis is written under the influence of the ongoing COVID-19 pandemic, we could plan and implement a day-to-day workflow that suited everyone. In the early stages of the project, the R-ratio[1] in the local county was not particular higher. Therefore we were allowed to work in the school without any disturbances.

Halfway through the thesis, the R-ratio spiked in the local county leading to more strict social policy. The strict policy did not directly affect the group. Some of us were able to work in the labs doing hardware testing or similar, whilst others could stay at their apartment working on software specific requirements. Other problems like illness or complete lockdown, as mentioned in Appendix [A.1], also did not considerably affect the group.

---

[1]The R-ratio is a method of assessing the ability of a coronavirus or any other disease to spread.

4.3 Communication

Communication is an integral part of the project; using inadequate communication protocols or implementations would severely impact every other system in the project. In order to properly scale a project, a fundamental communication protocol is required to connect various systems.

4.3.1 System to System

ZMQ has been utilized to handle the socket communication between the various software systems. The base patterns used throughout the applications are based upon the publisher/subscriber 2.8.2 and the request/reply 2.8.1 patterns as previously described. Because different software systems are written in different languages, there arose a demand for cross-language communication. As previously noted in 2.8, ZMQ is highly supportive through a wide range of languages.

4.3.1.1 Validating ZMQ

Since we rely on a messaging library to carry messages across sockets between various devices, it is necessary to test whether they are trusted. Connections, disconnects and reconnection are three key junction points for secure communication. The test setup should first consist of a one-to-one connection in a small environment without other devices before scaling it up to a complete software solution.

4.3.2 Payload structure

As the communication across various languages is implemented, it was decided to follow a strict JSON only communication protocol. Each JSON response consists of three types of payload names; sensor_data, settings and commands. As seen in the following subsections; 4.3.2.1, 4.3.2.2 and 4.3.2.3 the structure is identical but the varies in payload name and thereby in payload data content.

4.3.2.1   Sensor data

Example of a JSON payload containing two sensor data values.

```
data = {
    "payload_name": "sensor_data",
    "payload_data": [
        {
            "name": "pressure",
            "value": 3.2
        },
        {
            "name": "roll",
            "value": 23.1
        }
    ]
}
```

4.3.2.2   Settings

Example of a JSON payload containing a settings config.

```
data = {
    "payload_name": "settings",
    "payload_data": [
        {
            "name" : "Oxygen",
            "origin": "ArduinoSensor",
            "port": "A2"
        }
    ]
}
```

4.3.2.3 Commands

Example of a JSON payload containing an ROV command.

```
data = {
    "payload_name": "commands",
    "payload_data": [
        {
            "name": "pid_depth_p",
            "value": 5
        }
    ]
}
```

4.3.3 ROV systems

The purpose of the RPi located in the ROV is mainly to collect data from Arduino and pass this on to the REST-API in the onshore computer and send commands the other way. It uses serial communication to receive the data from the Arduino, opening up one thread for each serial connection. The whole structure of classes is shown in Figure 4.1.
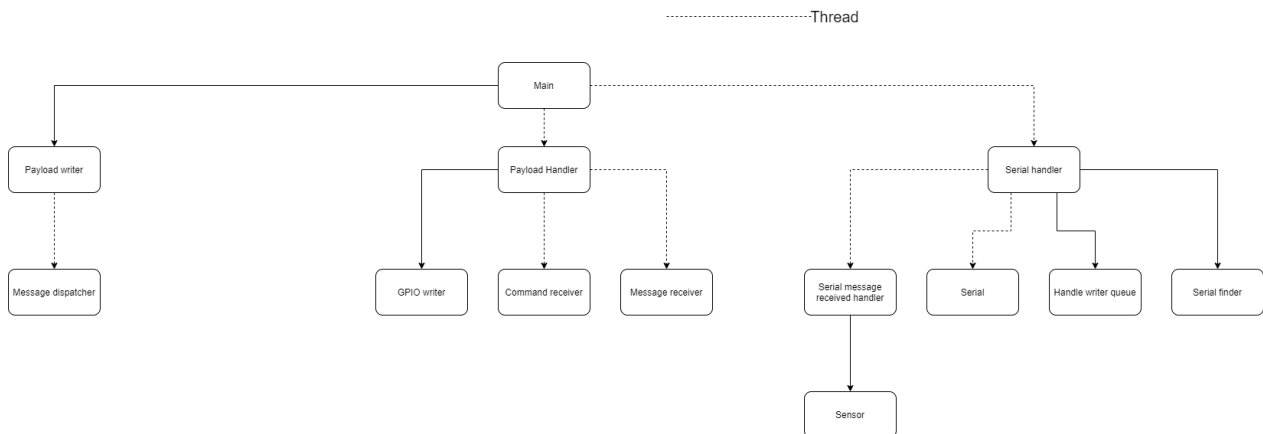
Figure 4.1: ROV-RPi structure

4.3.3.1 Serial communication

The old software had a good way of detecting which Arduino was connected to a specific serial port, and this will be used. However, it did not have the stability and robustness that is required for this project. For this project, we need a stable connection that can handle a high amount of data, second the procedure of finding a serial port needs improvement. Third, be able to search for serial ports at any time (not only at boot).

Establishing connection

To ensure that the RPi finds the Arduinos each time, a good connection has to be made. Firstly an upgrade in the baud rate so that it can handle the data flow. The Arduino, with the task of collecting sensor data and Arduino handling control of the stepper motors, use 57600 baud rate. Throughout the rest of the thesis, these will be referred to as ArduinoSensor and ArduinoStepper. The frequency of data sent from the ArduinoSensor is considerable. To not let this disturb the process of finding the Arduinos, a delay of 2 seconds is added in the setup.

The procedure of finding the Arduinos and at the correct serial port is shown in Figure 4.2. It starts by getting all the serial port available (works for both Linux or Windows-based system). Then it will start to open each port at a baud rate of 57600. If it gets an expected message from one of the Arduinos, it will put the serial port in a list with the correct baud rate. If it does not get a correct message at the 57600, it tries a baud rate of 115200; the amount of baud rate to try is easily expandable. The process is continued until all serial ports are checked. All ports found with an Arduino connected will be opened in a thread for reading and sending data.
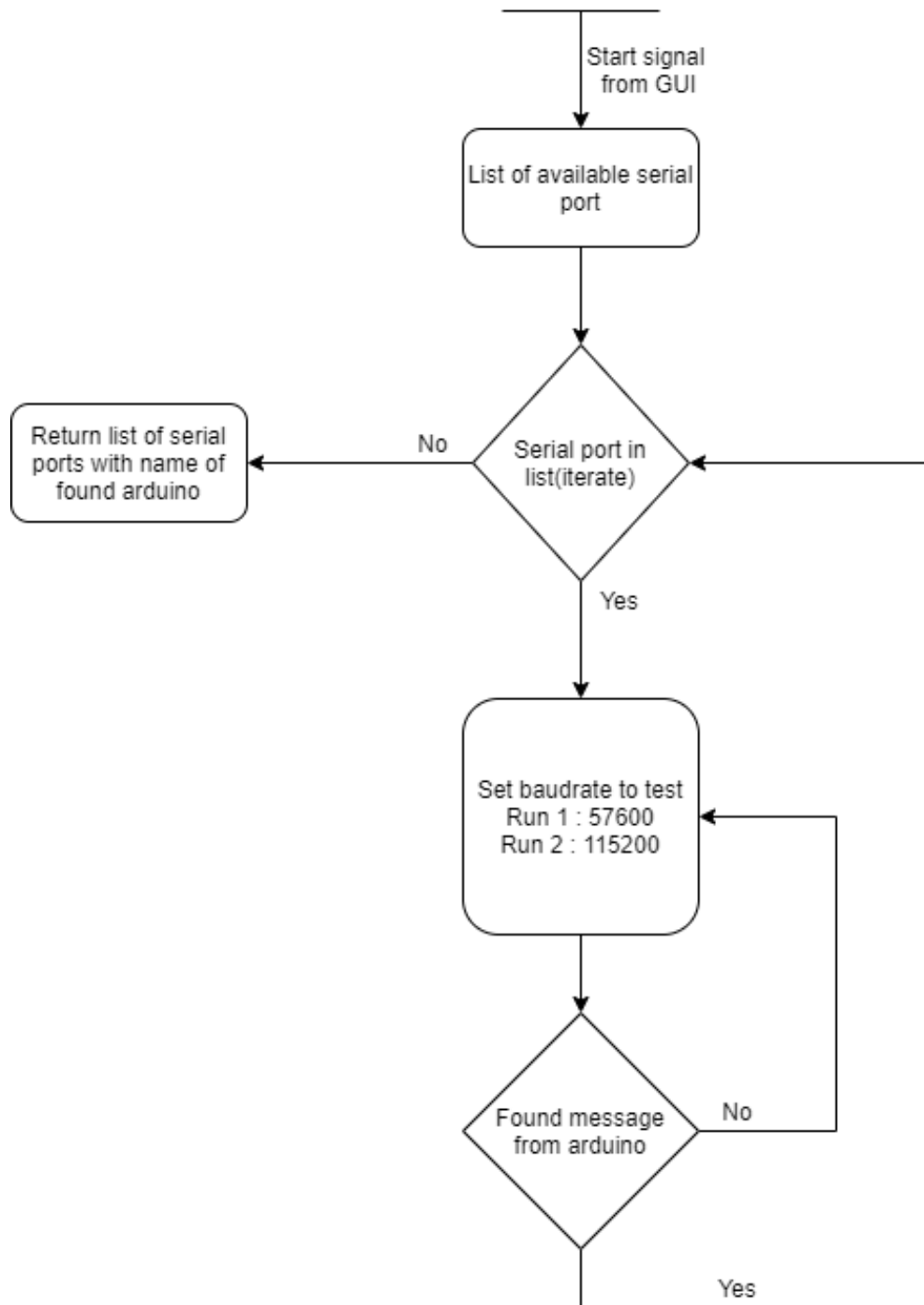
Figure 4.2: Serial finder flow

Serial reader function

Initially, using the standard pySerial 3.5.3.1 library from Python to communicate with the Arduinos worked fine in the project development. However, as the project side increased and the

system frequency speed increased, it was noticed that the standard **readLine()** -method from the library was becoming a bottleneck for the system's performance when using it a thread. Therefore an own serial read -method was made; this was inspired by pySerial own serial thread library. This function is shown in appendix [K].

Searching for serial ports

Even with a good connection, there is a possibility of not finding all the Arduino due to different reason. I could not have been connected at boot or could have been disconnected. Therefore, the possibility of searching for a serial port is added. Simply it closes all serial threads and starts the procedure of finding the serial port again; this will reduce the amount of time the RPi has to reboot and potentially extend the operating time.

4.3.3.2   Handling data from ZMQ to serial port

Classes to reading and sending serial and using ZMQ are threaded. Therefore a queue is used. Queue is, as explained in 3.5.3.1 is thread-safe, and it is an excellent way to send data between threads.

For sensor data from serial to **MessageDispatcher** (ZMQ publisher class) a list of sensor objects is used. Each sensor object has a name and a value. Although the list is thread-safe, the data in the list are not. There is no guarantee that if we change the value of a sensor in one thread, it will change if we change it in another thread simultaneously. However, in this case, it is not necessary to change the value multiple places. The reason for using a list over queue for the sensor data is that the GUI gets an update from the ROV with a sensor data payload 4.3.2.1 every 100 ms, and then the rate of incoming data from the sensors is far greater. As we always want the latest data, we constantly have to take a sensor(object) out of the queue, and it is important to read from the queue more often than data is written to the queue. Using a list, the latest values are only read every 100ms and sends it to be published with ZMQ.

In contrast, queue is used when handling commands and responses since every command/response has to reach its destination.

4.4 Structuring the Software application

As mentioned in the project requirements, it essential to have a functional, operational and reliable graphical user interface. Ease of use, adaptability, support of a live video feed, sensor data, and the ability to control the ROV are significant. As the ROV is essentially a data collecting sensor platform, the amount of data and the selection of sensor types can vary from time to time. Therefore, the new software systems must be built with a solid architecture to support the scalability of the project long term.

In order to correctly build scalable software for the bachelor project, a three-tier architecture was chosen 2.4.1. The advantages of this architecture lie in the separation of concerns. Each tier can independently be developed without affecting the other tiers. This architecture presents an increase in development time, higher modularity, and looser coupling of code. Figure 4.3 below, shows the organization inside the three-tier architecture.
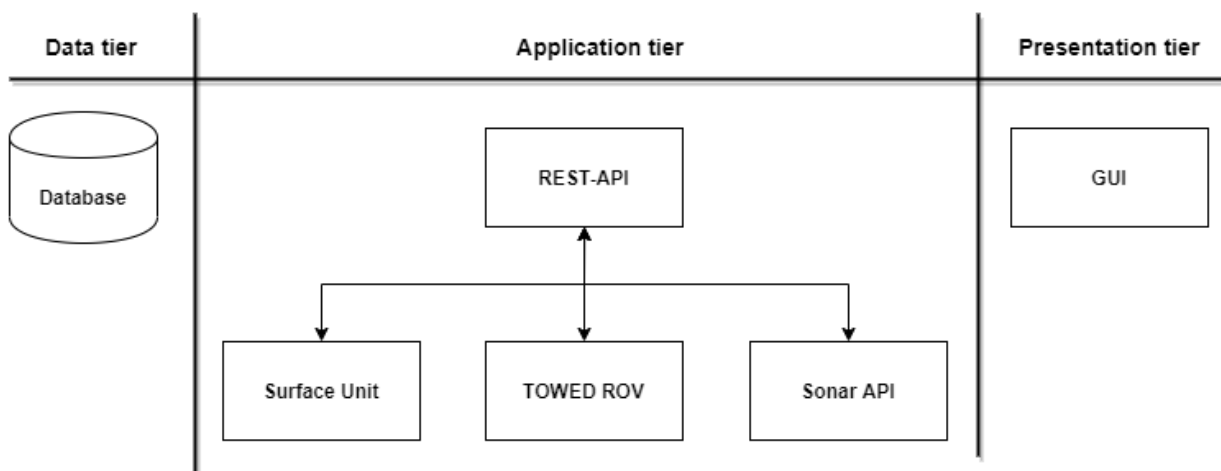
Figure 4.3: Organizing the three-tier organization

4.4.1 Presentation tier

This tier represents what the client will see when operating the system, more commonly known as the Frontend / GUI. The GUI is implemented using React 3.5.4.1. In order to create a desktop application, Electron 3.5.4.1 is used to create an executable file, wrapping the web application inside of it, capable of running on cross-platforms.

Selecting of Graphical User Interface

It was decided to use React and Electron 3.5.4.1 for implementing the presentation tier, referred to as the GUI. Many desktop GUIs nowadays is implemented in various languages and frameworks such as, for instance, Tkinter, PyQT in Python programming language or Swing, JavaFX in Java programming language. Using React, we will gain the benefits of its popularity and comprehensive software community. Combining React with Electron, we essentially create a web application inside a desktop application that can run off any platform (Windows / Mac / Linux).

Figure 4.4 below shows the apparent trend of a few UI frameworks and their popularity according to Stack Overflow Trends [56].
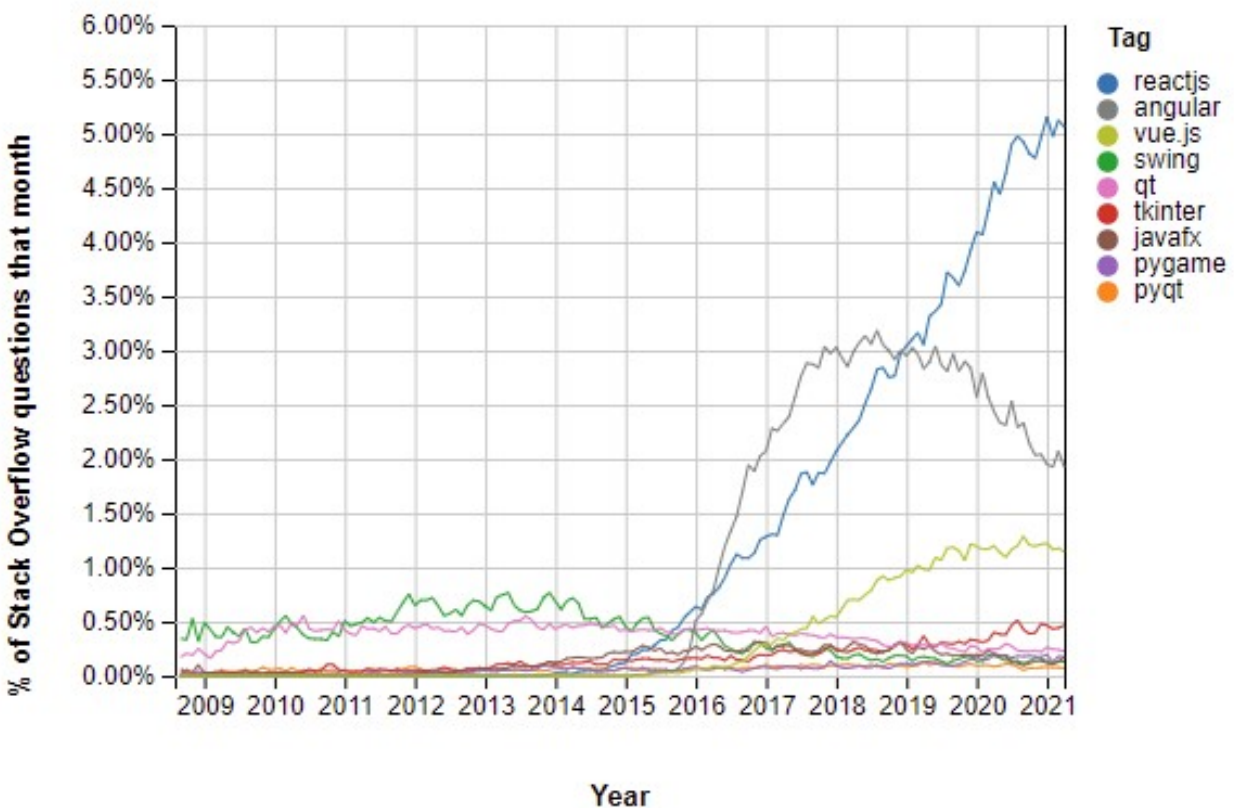
Figure 4.4: StackOverflow: A generated plot showing technologies trends over time

4.4.1.1 Setup

In order to correctly set up the GUI, the web application is bootstrapped with Create-React-App 3.5.4.1 tool to initialize starting template files for development. As the files are created, they need to be converted from a standard browser to a desktop environment. Using the electron-builder from Electron, we can package and build the React template files into an application capable of running cross-platform as a desktop application. The design aspect of the React application was made with the Chakra UI 3.5.3.2 component library to build a faster and more accessible interface design making it easier and more intuitive to use for the operator.

As mentioned, the React application resides in the Electron environment. In order to convert the web application into a desktop application, the steps in Figure 4.5 (on windows in this case) can be followed to create an executable desktop application. A more detailed approach can be found in the README file in the GUI source code.
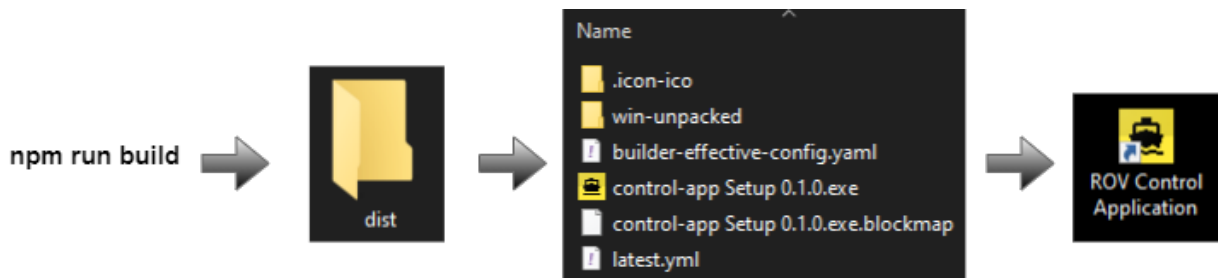
Figure 4.5: Windows build: from web app to desktop app

4.4.1.2 Structure

The web application uses HashRouter[2] from the React-Router (3.5.3.2) library in order to manoeuvre through the various pages safely. Using JavaScript libraries such as Axios (3.5.3.2) and Event-Streams (2.6.1.6), the GUI can communicate with the REST-API. The sections below will briefly explain the planned functionality of various pages inside the GUI.

---

[2]A <Router> that uses the hash portion of the URL (i.e. window.location.hash) to keep your UI in sync with the URL.

Home page

The Home page represents the landing page of the application, which is the first site upon startup. From here, one could navigate to other parts of the application.

Settings page

Inside the Settings page, all the settings for setting up the sensors at the ROV is initialized. As the project requirement previously stated, an adaptable and modular system is needed. Therefore, ahead of starting any exploration with the Towed ROV, one would often want to adjust the various equipment inside the ROV. For instance, creating, removing, disable- or enabling various sensors. Before each run, the operator should double-check that the ROV's added hardware is correctly wired; then, when the device boots up, go to the Settings option and change the settings preferences to match the hardware setup.

Dashboard page

In the Dashboard page, the main graphical user interface resides. The dashboard has the options to control, in real-time, the various aspects of the ROV systems and monitor the live sensor data from the various sensor components, the live video feed from either the video camera or the side-scan sonar. The operator also can create a 'Session' to collect data from a given geographical mapping mission.

In terms of monitoring the system's performance, the sensor data is presented in an orderly manner while watching various metrics in a Chart display to view its changes as the ROV is in action. As a measure of control and secure operation ability, the operator can also view the sent commands and their respective responses inside the toolbox. Since the ROV is often not visible from the boat, it is challenging to notice whether the commands are actually in progress. A solution to this problem is to use confirmation responses to be safe, knowing the system works as intended. All commands and responses are marked as either successful or failed.

Map page
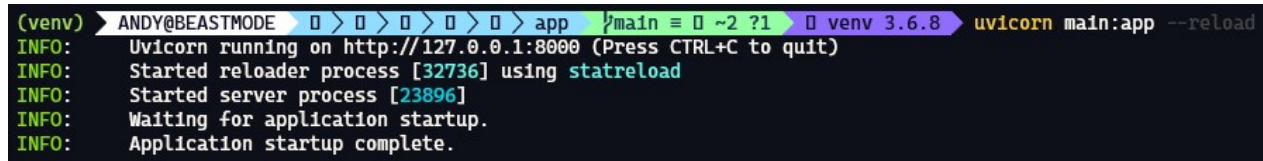
The Map page allows the operator to view all session recording and the stored sensor data in a Google Maps-lookalike setup. As the latitude and longitude are used to create waypoints in the Map, the operator can click at every saved waypoint to view the various sensor data collected from that exact location and a camera/sonar image visualizing the view at the bottom of the sea.

4.4.2   Application tier

The application tier is where all the business logic is created. This part of the application is part of the backend, where the information from various entities/hardware components is collected and processed. The backend is built using the FastAPI web framework to create a REST-API, which provides HTTP support for communicating with the frontend and ZMQ / TCP/IP connections to communicate with the other ZMQ entities (Surface Unit, ROV, Sonar API) of the application tier.

Setup

As the FastAPI is a web framework, specifically an ASGI framework, it requires an ASGI server implementation to run frameworks. Using Uvicorn 3.5.3.1, we can host FastAPI as a server running at localhost with port 8000.


```
(venv)  ANDY@BEASTMODE  □ › □ › □ › □ › □ › app   ᚉmain ≡ □ ~2 ?1  □ venv 3.6.8  uvicorn main:app --reload
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [32736] using statreload
INFO:     Started server process [23896]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Figure 4.6: REST-API Startup

The REST-API start-up code shown in the code example (from Appendix [G]) to the right shows the steps necessary to initiate the structure of the REST-API. It starts by creating a FastAPI object; this is the REST-API itself which resides all the connections. After that, we define a list of allowed remote URLs allowed to access the REST-API (i.e., GUI runs at localhost with port 3000) and attach it to the API object as allowed app middleware. In the bottom line, we include the api_router, which holds the routes of all the endpoints 2.6.1.2 available in the REST-API.

```
app = FastAPI()
origins = [
    "http://localhost:3000",
    "localhost:3000",
]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)
app.include_router(api_router)
```

Structure

The application tier is structured as a standard REST-API, consisting of endpoints for interactions and a few separate processes that handle all communication with the other entities in the software systems, as shown in Figure 4.7. The different endpoints, shown in the REST-API documentation found in Appendix [D], are used by the GUI to distribute and retrieve data from the database through HTTP methods.
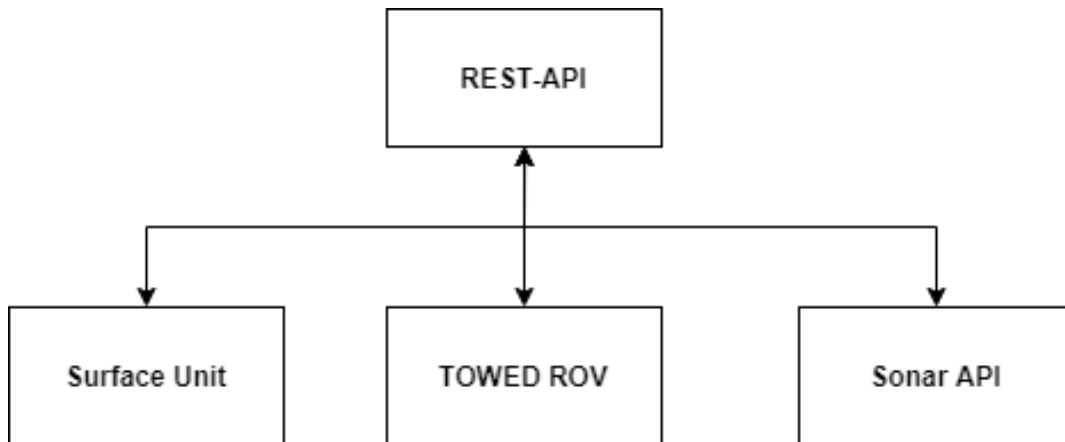
Figure 4.7: Entities in the application tier

Implementation: HTTP methods

The GUI uses HTTP methods to communicate with the REST-API to make a request to the database. Figure 4.8 shows how the base methodology of how the transaction method between the GUI and the REST-API is done. In short, a payload request is sent from the GUI to the REST-API. The REST-API looks at the payload request and makes decisions based upon the endpoint, path parameter and query parameter. Using these filter methods, the algorithms inside the API fetches items from the SQL database as requested and returns a response accordingly to the request made. The following examples are: POST, GET, PUT (update) and DELETE.
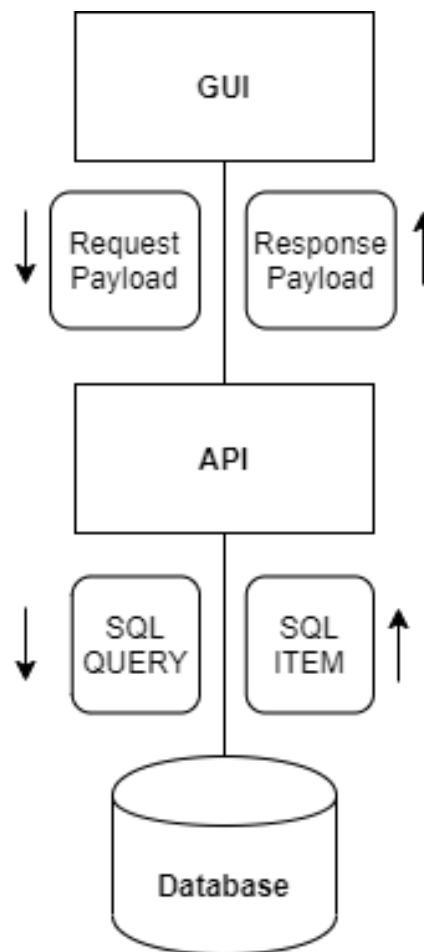
Figure 4.8: HTTP-transaction

POST

1   /waypoints/

2

Waypoint

```
{
  "session_id": "my_test_session",
  "latitude": 62.472169,
  "longitude": 6.234030,
  "sensors": [
      {
          "name": "temperature"
          "value": 4.5
      },
      {
          "name": "depth"
          "value": 84.8
      },
      {
          "name": "pressure"
          "value": 8480
      }
  ]
}
```
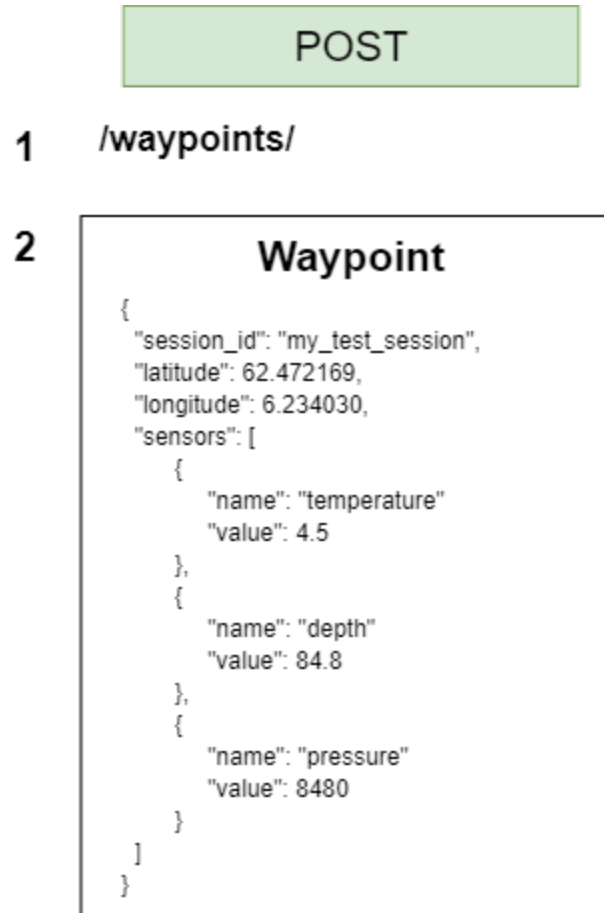
Figure 4.9: HTTP-POST example: add to database

1. Signifies the endpoint the data is pointed towards.

2. Signifies the payload data the API receives when the GUI creates a waypoint
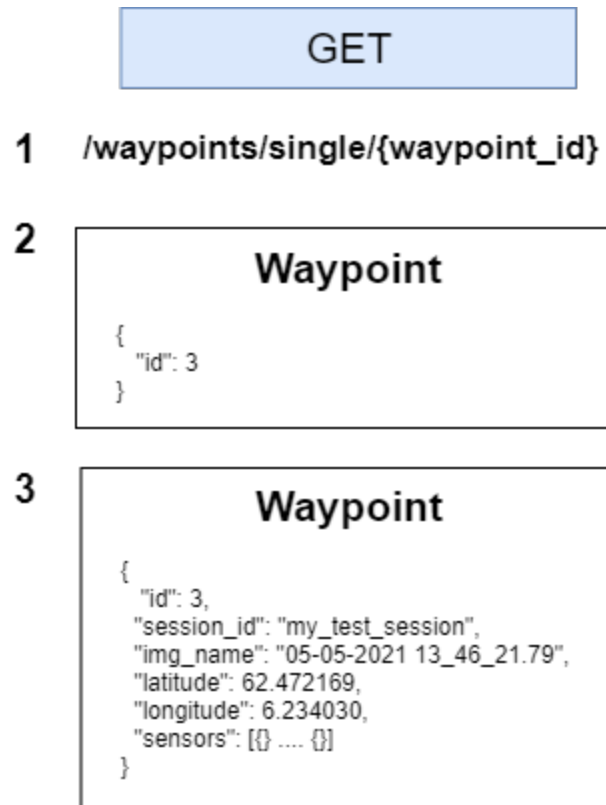
Figure 4.10: HTTP-GET example: retrieve from database

1. Signifies the endpoint of the request. The GET-request calls the endpoint 'waypoints' with a path and query parameter, which is in the waypoint_id.

2. Is the GET-**request** payload which is sent to the REST-API. It receives only a single key:value pair, which is enough to filter the search.

3. Is the GET-**response** payload which the GUI receives given the waypoint_id requested. Since, a *waypoint* has a relationship with a *sensor* by the One-to-Many 2.5.4.2 structure, the waypoint item uses its foreign key 2.5.3 to retrieve a a list of sensors and attaches it to the total payload.
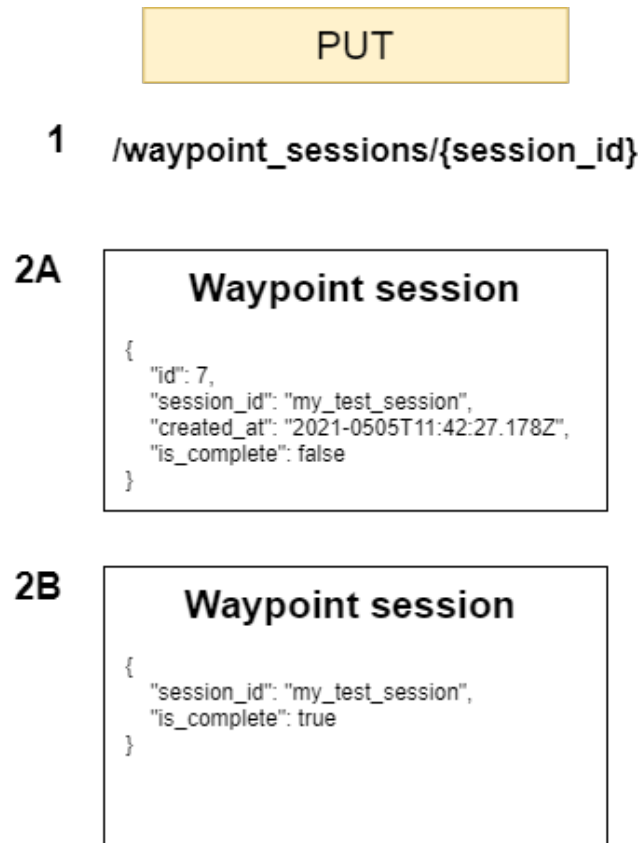
Figure 4.11: HTTP-PUT example: update in database

1. Signifies the endpoint the request is made to, with a query parameter *session_id*.

2. Is the item which is stored in the database with the *session_id* before its updated.

3. Is the payload required to update the item in the database.
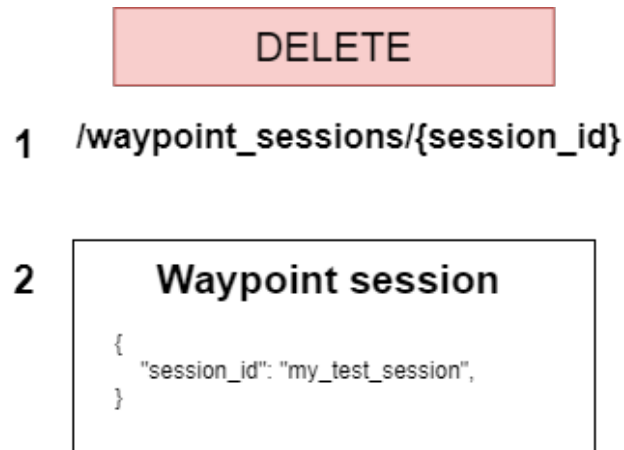
Figure 4.12: HTTP-DELETE example: deleting from database

1. Signifies the endpoint the request is made to with and query parameter *session_id*.

2. Is the payload required to delete the waypoint session from the database. Since session_id is the primary key 2.5.2, there can only exist exactly one item with that name in the database. As a waypoint also has a relationship with other sensors, a waypoint deletion will also trigger the foreign key to delete any sensors if a relationship exists.

Implementation: External communication

As shown in Figure 4.7 above, the REST-API has three external connections. All three connections run on separate processes 3.5.3.1 to prevent any interference with the internal REST-API thread-pool reserved for HTTP requests.

As the messages are sent and received between the various ZMQ entities, they are all forwarded by the use of multiprocessing queues 3.5.3.1 as a communication channels between processes. In Figure 4.13 below, we can see an example of how each of the three connections communicates through message queues.
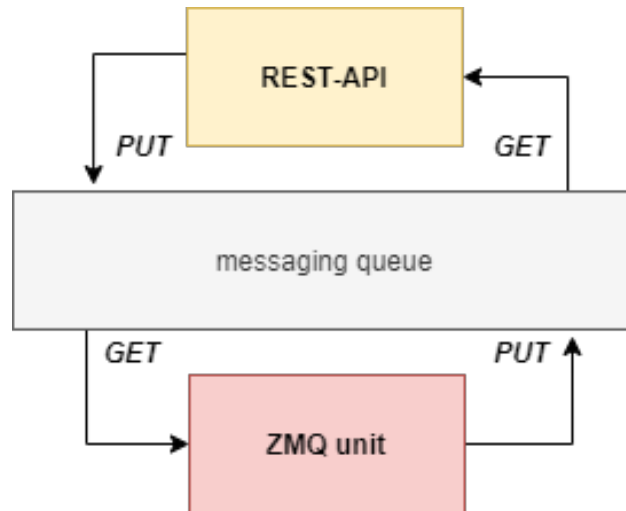
Figure 4.13: Using a shared communication queue.

ZMQ unit could be of type publisher, subscriber, requester or replier

All external connections have predefined transmitting frequencies to ensure predictable behaviour by not overflowing the REST-API with data. The ROV should send the sensor data 10 times per second and video data 30 times per second. The Surface Unit should send the sensor data 5 times per second. The sonar data from the Sonar API should be sent anywhere from 30-60 times per second, depending on size.

Throughout the following subsections, **SensorPublisher** is referred to as a class that implements a ZMQ Publisher socket, and **SensorSubscriber** is referred to as a class that implements a ZMQ Subscriber socket. Below, we will briefly explain the concepts and interactions between the REST-API and the connected processes.

Implementation: REST-API $\Longleftrightarrow$ Surface Unit

The relationship the REST-API has with the Surface Unit is purely a ZMQ-protocol. The pattern is based upon publisher/subscriber. When the system starts up, the REST-API creates a **SensorSubscriber**, which connects to a Raspberry Pi's IP and PORT inside the Surface Unit and listens for any published messages. As the messages are received, the **SensorSubscriber** adds them into the messaging queue. The REST-API can then collect data from the queue and forward it where it is needed (as visualized in Figure 4.13 above).

Implementation: REST-API ⟺ ROV

The connection to the ROV consists of two ZMQ sockets and one pure TCP/IP socket.
The first ZMQ socket follows the publisher/subscriber pattern. Like the Surface Unit, a Sensor-Publisher sends a stream of sensor data from various sensors collected from different Arduinos and Raspberry Pi into the socket where the SensorSubscriber in the REST-API side continuously reads from and adds to the messaging queue.

The second ZMQ socket follows the request & reply pattern, which is used to send commands from the REST-API to the ROV. By using this pattern, each command sent is expecting a response back as a command confirmation.

The third socket for communicating with the ROV is a pure TCP/IP server, and the client used to send live video stream data from the ROV's camera. Inside the ROV, running a TCP camera server allows incoming TCP clients to connect and receive image sent from the mounted camera. As images are received on the client-side, they are added into a messaging queue and forwarded into the REST-API.

Implementation: REST-API ⟺ Sonar API

From the REST-API to the Sonar API, a ZMQ publisher/subscriber pattern is used for receiving data from the sonar. As the Sonar API receives sensor data from the side-scan sonar through a TCP connection, the data is formatted and given to a SensorPublisher. The SensorSubscriber, on the other hand, gets the sonar data, examines it, and then adds it to a message queue that is delivered to the REST-API.

Implementation: Special HTTP methods

In order to support the demands for the live streaming of sensor data and video, unique methods are needed in the REST-API. As standard HTTP methods are based upon a request-response pattern, it will cause various issues if the GUI is required to ask for data every N-times per second. The solution to this problem relies on using unique methods such as FastAPI's built-in

StreamingResponse and EventSourceResponse.

A **StreamingResponse**, explained in detail in 2.6.1.5, is used for sending a continuous stream of images converted into bytes. Taking advantage of the OpenCV-library 3.5.3.1, we can perform simple image processing techniques to change between readable image and images in bytes whenever we want to save the image to disk or show it in the GUI.

A **EventSourceResponse**, explained in detail in 2.6.1.6, is used for sending sensor data collected from the various ZMQ entities as an event-stream. The data is collected through multiprocessing queues, formatted into one JSON and sent as an EventSourceResponse to the GUI.

StreamingResponse and EventSourceResponse are considered one-way patterns in which the data travels only from A to B.

4.4.3 Data tier

In the data tier, all the information gathered by the software systems is stored. The type of database chosen is the SQL [106] database, or more specifically SQLite [107], which implements the SQL database engine. Also, SQLite is open-source, which is a requirement for the project's development. The SQLite databases, due to their simplicity and reliability, provide fast response times for data storage. The SQLite database file is self-contained in a single .db -file. Therefore it can easily be shared amongst users and can even be e-mailed to other colleagues.

Setup

The tables defined in the SQLite database file are created by the models and schemas defined inside the REST-API. By using models, we can design decided what fields and data type the tables should contain. Schemas provide a sort of cookie-cutter-shape, which request data has to match to be verified and added to the tables. Inside the REST-API, the database is defined from a URL-string seen from the *SQLALCHEMY_DATABASE_URL* below. To set up the database, it is customary to use a create_engine method from the Python SQL library, SQLAlchemy [20]. This

method accepts a URL-string defining the database type, SQLite in this case, and creates an engine object. The engine object is then passed on to the *sessionmaker* which can then invoke the database and create a *SessionLocal*-object. The *SessionLocal* object is then used throughout the REST-API each time a request is needed for querying the database.

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker


SQLALCHEMY_DATABASE_URL = "sqlite:///./sql_app.db"


engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)


SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

Structure

The structure of the SQL database is based upon tables that contain rows and columns. Each row in a table represents one unit of data where each column's various fields are represented in each column. With these principles, the tables can be represented and controlled using predefined models and schemas for each table. Using a model, one can define the fields and types of data that should be stored in the table. Using schemas, one easily controls the input and output data each time data is created, retrieved, updated, or deleted. The Pydantic 3.5.3.1 Python library provides a base model from which we can build schemas. Incoming untrusted data can be passed to a schema. After parsing and validating the data with Pydantic, the resultant schema instance will conform to the field types defined on the schema.

Implementation

Using the principles earlier described in 2.5 about database theory, we landed on using a struc-
tured and logical way of collecting the sensor data. Figure 4.14 below shows the relationship
method used for defining the relationships between each table. For each waypoint session,
there can exist many waypoints. For each waypoint, there can exist many sensors connected
to it.
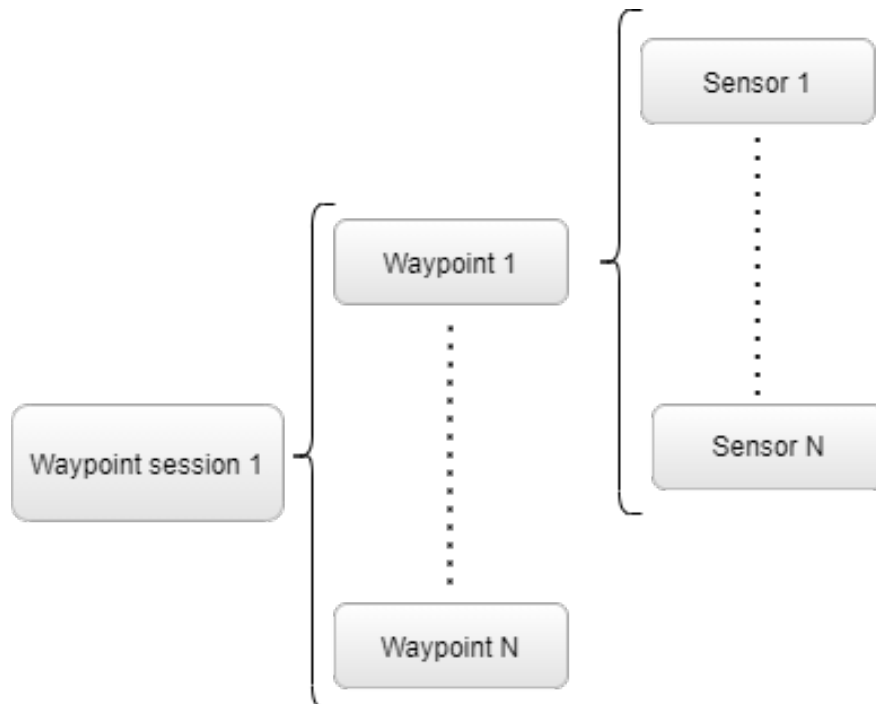
Figure 4.14: SQL: Relationships

Example of 'One-to-Many' → 'One-to-Many'

The following Figures below show how we have designed the various field names and field data types for each of the four different types of tables used in this bachelor project.

Settings						
field	id	name	enabled	origin	role	port
type	int	string	boolean	string	string	string

Figure 4.15: SQL Model: settings

Sensors				
field	id	name	value	wp_id
type	int	string	float	int

Figure 4.16: SQL Model: sensors

Waypoints					
field	id	session_id	img_name	latitude	longitude
type	int	string	string	float	float

Figure 4.17: SQL Model: waypoints

WaypointSessions				
field	id	session_id	created_at	is_complete
type	int	string	datetime	boolean

Figure 4.18: SQL Model: waypointsessions

4.5 The Surface Unit systems

The surface unit consists of an echo sounder and GPS connected to an RPi and uses serial communication, either over USB or UART. Both sensors have their data parsed using the NMEA 0183 protocol. The Surface Unit RPi reads the echo sounder sensor and the GPS sensor in the boat and publishes it as a ZMQ publisher 2.8.2. When designing the system, the main goal was the ability to handle multiple different sensors with serial connections with a single microcontroller.

The program uses different threads to read from the sensors, handle logic and publish using a ZMQ Publisher. It saves data to the thread-safe **StorageBox** class and publishes data over ZMQ as seen in the software structure 4.19.

The system also calculates the distance travelled using the change in the GPS coordinates. When a set distance is calculated, it publishes it as a command from the ZMQ server. The set distance is used in the seafloor tracker calculation 4.7.5.
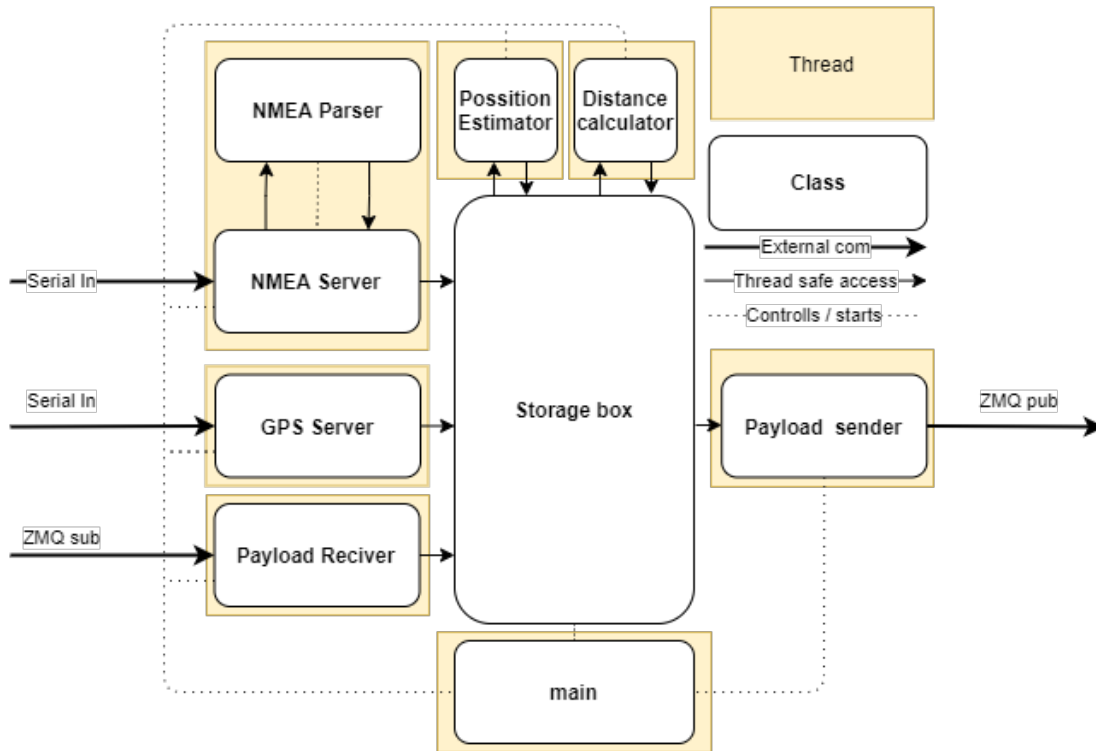
Figure 4.19: Surface Unit class structure

4.5.1   Handling NMEA data

The RPi reads NMEA signals 2.7.2.1 over serial communication. The data must be parsed, and the checksum must be verified before being transmitted to the StorageBox.

When parsing the data, all values are stored, even Null values. Because it is relevant to the system if a sensor does not send correct data. NMEA data packets are converted to a human-readable representation. for example, the NMEA id "DPT" changed to "depth_beneath_boat".

For the depth measurements, it was necessary to change all Null value from the echo sounder to -1 to make it easier for the database to handle the data. The system, therefore, supports adding conditions to specific NMEA Sensor IDs.

NMEA structures

There are many NMEA structures, and when trying to find the structure behind an identifier like "STDPT", you often come across contradictory or incomplete data. It is, therefore, essential to make sure that the structure you are using is correct. For ease of use, the surface unit has a large selection of NMEA sentence structures saved, but these should always be controlled when adding a new sensor. If the system does not have a stored structure, the system uses the default values the sensor identifier and "value_#" to check the structure. The structure should then be compared with the datasheet from the sensor provided.

Parsing NMEA

When working with the NMEA data over serial, there is a need for controlling the data that arrives. To do this, the NMEA structure ends with a checksum. The pynmea2 library calculates this checksum. An example of NMEA checksum calculation is provided in Appendix [I].

When parsing a message, the data should be delivered as a byte array. However, there might be useless data around the actual NMEA sentence, or the data might be corrupt. Therefore the data is stripped by removing any values before the start character and after the checksum. The pynmea2 3.5.3.1 package can decode the message. From there, the sentence can be renamed using a collection of key-value pairs with NMEA names and names for the ROV system, manipulated and published to the rest of the system.

4.5.2   Implementing the Echo Sounder

The topside echo sounder uses standard NMEA 0183 communication and operates at 12 V. It is connected to the RPi through an optocoupler. This coupler converts the 12V signal to a 5V signal to protect the RPi from the high voltage of the echo sounder. The echo sounder transmits the depth beneath the transducer in both meters, feet and fathoms, the speed of the echo sounder, water temperature and other redundant information [91]. The echo sounder does not receive data from the RPi, and the communication is simplex 2.7.2, with the echo sounder acting as a talker and the RPi acting as a listener. The echo sounder transmits the following NMEA sentence structures:

- **SDDBT**-(Depth Sounder: depth below transducer) for depth

- **SDDPT**-(Depth of water) also for depth.

- **VWVHW**-(Velocity sensor Water: Water speed and heading) for speed.

- **VWVLW**-(Velocity sensor Water: Distance Traveled through Water) for Distance.

- **YXMTW**-(Transducer: Mean Temperature Water) for temperature

[90] Only the depth below the transducer and the water temperature is sent to the REST-API. The echo sounder also supports the NMEA 2000 protocol.

4.5.3 GPS implementation

The GPS is an Adafruit GPS [58] that transmits data over serial. It is connected with a $TX\backslash RX$ Serial to USB converter to the USB of the RPi. Since there is a lot of data transmitted by the GPS, the Adafruit data from the GPS is transmitted in multiple NMEA sentences containing different GPS data, from latitude and longitude to the number of available GPS satellites and the unit's speed. Only the latitude, longitude and speed are sent to the REST-API. The GPS uses these NMEA sentences:

- **GPGGA**-(Global Positioning System Fix Data), latitude, longitude, different general GPS data like satellite number, speed and time.

- **GPRMC**-(Recommended minimum specific GPS data) latitude, longitude, speed and heading.

- **GPGSA**-(GPS DOP and active satellites), information for the satellites it can connect to.

- **GPGSV**-(GPS Satellites in view), the number of and data about satellites in view.

- **GPVTG**-(Track Made Good and Ground Speed.) Information about the speed and track of the GPS.

Although the GPS uses the NMEA 0183 protocol, its communication is duplex since it can receive commands. For example, When the system starts, the GPS can receive a setup command. The GPS server uses the Adafruit GPS library 3.5.3.1.

Position estimation for GPS data

When collecting sensor data for scientific or environmental research, the system needed to have an accurate position for the data. Underwater GPS or other position estimation hardware is often quite expensive. Therefore, the group wanted to create a robust and easy to implement software solution using sensor fusion and established physical theory connected to towed objects. Calculations for the arc of a tow cable underwater are highly complex, [79] and not well known. Therefore a different solution had to be implemented.

Using the length of the cable and the depth of the ROV to compute the distance behind the boat using Pythagoras' equation 4.1 is a straightforward approach to obtain an estimate of where the real ROV is. With a cable of around 200 meters and a depth of 50 meters, this will give a distance of $\sqrt{(200m)^2 - (50m)^2} = 193.649m$. This distance can then be combined with the GPS heading to create a vector for the ROV position.

$$\text{length from ROV to boat} = \sqrt{\text{length of cable}^2 - \text{depth of rov}^2} \tag{4.1}$$

The Surface Unit collects GPS data, latitude, longitude and heading, and calculates an approximate GPS position of the ROV by simplifying the calculation to one degree of latitude or longitude represents a 111Km distance [9].

Calculating travelled distance

The system also calculates the distance using the changes in the GPS position. The distance calculator uses the Haversine function to calculate distance travelled 2.10 and calculates the radius of the earth of the location of the ROV using the formula 2.11.2.

4.6 Hydrographic surveying

One of the project's goals was that the ROV should have the possibility of doing hydrographic surveying. There are a large number of available technologies with different advantages and disadvantages.

4.6.1 Points of consideration

When choosing a seafloor mapping technology, we can present the following points of consideration:

- **The data needs**

  The data needs depend on what kind of data is needed to survey and are the most important factor. For example, a side scan sonar can produce a high-resolution image over a large area but does not collect topographic data or information about the seafloor composition.

- **Price**

  Price is an important and limiting factor. The price of a system might entirely disqualify an otherwise perfect solution. It should always be minimized as long as the requirements are reached.

- **Range**

  The area that a survey unit can cover can be important or redundant depending on the application. The range can even be detrimental because it often comes at the cost of accuracy, price or both.

- **Adaptability**

  If the surveying equipment is to be implemented into a specific or existing system, it needs to be adaptable and have software integration possibilities.

- **Bandwidth needs**

  When survey data is collected, it needs to be saved in the ROV or sent to a computer for further processing or plotting. The system bandwidth needs to be able to handle data coming from the survey equipment while not throttling the communication system.

Evaluating the price categories

Different companies making hydrographic surveying equipment were contacted. After talking to different system providers, it was decided that we would use a side-scan sonar. Side-scan sonar systems come in a wide price range and can be classified into three categories.

- **Hobby level** systems are mostly used for fishing and hobby experiments. The systems are cheap and usually cost around 700€ for the transducer systems, and has a decent resolution. However, these kinds of systems are difficult to adapt to existing software solutions, and they often only have a depth rating of around 10 meters. The systems are almost always integrated incorporate systems with ready to deploy displays, software and hardware. These supplementary systems often increase the cost by thousands of euros.

- **Medium level** systems offer increased adaptability and often have custom made specifications like housing or depth rating to fit your system. In addition, these systems often provide an API. With a provided API, it is possible to implement a sonar image into our GUI. The price of these systems is from around 1 500 € to around 20 000 €.

- **High-end systems** are the most accurate, with great depth rating, resolution and range and adaptability. However, the price matches, from 20 000€ to over 100 000€.

Primary goals

When choosing a sonar, we considered these points to be the main needs of the ROV project:

- Runde Miljøsenter wanted to find lost fishing equipment. Therefore resolution and range were relatively important.

- The system's price should be as low as possible while achieving the other requirements.

- The adaptability needs to be good to implement the sonar into our system easily.

- The required bandwidth of the sonar needs to be acceptable to the bandwidth of the tether system on the ROV. The Slepe-ROV has a bandwidth of around 10/100 over ethernet, which the sonar system would need to share with other systems in the ROV.

For the Towed ROV, the high-end systems are too expensive to consider. Therefore the choice was between the hobby level and the medium level. As we reached out to a few engineers from Garmin, we were told that these systems would not support being implemented into our system. In addition, the depth rating of these systems where low and therefore would not support the depths we wanted to reach. Therefore using these kinds of systems would require modification for them to fit the ROV system.

This would include making sure that the transducer where depth rated for at least 50 meters and decoding the image sent from the electronics to implement it with our API. Therefore, it was decided that the hobby level systems were not a good fit for the ROV system.

Main alternatives

In the end, we seriously considered three different companies, each with multiple systems and available customization's.

- **Imagenex:**
  Has the smallest size and weight and the widest range. The Imagenex sonar uses serial communication, but they are also the most expensive [50].

- **DeepVision:**
  The electronics of the DeepVision sonar can be potted, which would increase the adaptability for future systems. This would also present the possibility for connectors instead of penetrators for cable entry into the ROV body. Making it easier to add and remove the transducers in the future and decrease the amount of watertight space required for future ROV bodies. This would reduce the chance that they could be damaged during transportation. DeepVision also Provides a cross-platform software API for integrating into our

system. The Deep vision OEM module sends data using ethernet and is powered by POE [26].

- **StarFish:**

  Includes software API for ease of integration. The electronics box is large and would take a lot of space in the ROV body. But it can provide pre-made fastening brackets, making it easier to mount the system. StarFish also have a "single unit" system where the entire system could be attached to the ROV with a few screws. But the electronics Uses USB communication and would need to be coded into the RPi system in the ROV [87].

Both DeepVision and Starfish provided unique advantages. Deepvision with their potted and smaller electronics and possibilities for connectors, and StarFish with their simple bracket system and lower price. In the end, the group went with the DeepVision system, mainly because of the advantages of connectors with the potted electronics, the small size of the electronics, and the ease of integration.

4.6.2   Implementation

The plan for the Side-Scan Sonar was only to implement it to our software system and get it up and running. Therefore it was built a small electronic box to keep water out while testing, also brackets were made; these can be glued onto the ROVs flange. Connection of the sonar to the electronics module and Ethernet can be found in Appendix [F]. After a recommendation from the manufacturer, these were mounted, so they were crossing each other as shown in Figure 4.20. For both sonars, penetrators were added and sealed with epoxy. More about that in section 4.7.1.
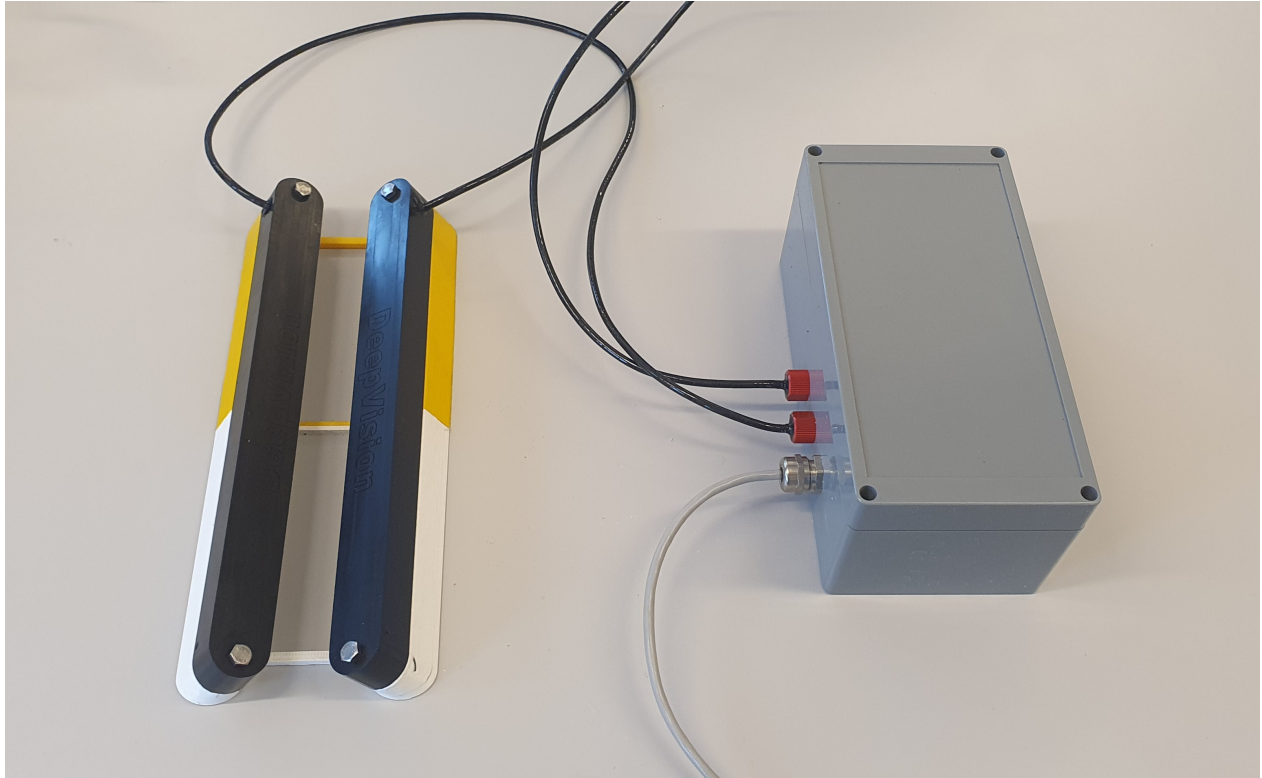
Figure 4.20: Side-Scan Sonar setup

4.6.3   Sonar API

The provided API for the side-scan sonar provides TCP implementation for connecting and receiving raw data from the sonar hardware. Optionally when initializing the main.cpp in Appendix [H], a DSV file writer can activated in order continuously store sonar recordings.

4.6.3.1   Testing

In order to rely on the test results and the operation ability of the side-scan sonar, it is necessary to test its validity through generating ground truth scans.

Before using the sonar in ROV operations, tests were done to validate if our software interpreted the sonar data similar to Deep-View FV. The comparison and display of sonar data were measured and shown below. The methodology to conduct these results was previously described in 4.6.3.1. Figure 4.21 shows the ground truth standstill image from the DeepView-software and Figure 4.24, 4.23 and 4.22 shows the corresponding images in our own software. The scan duration is approximately 45 seconds.

Figure 4.22: Dashboard VIDEO: the last section of the groundtruth scan
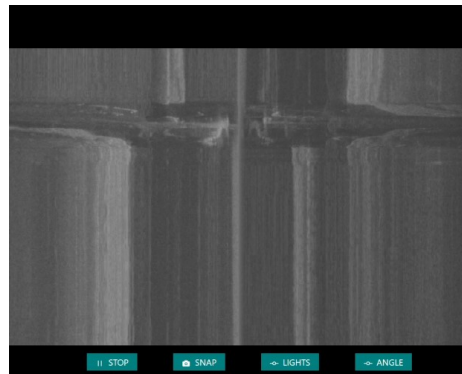
Figure 4.23: Dashboard VIDEO: the middle section of the groundtruth scan

Figure 4.21: Deepview FV: groundtruth scan
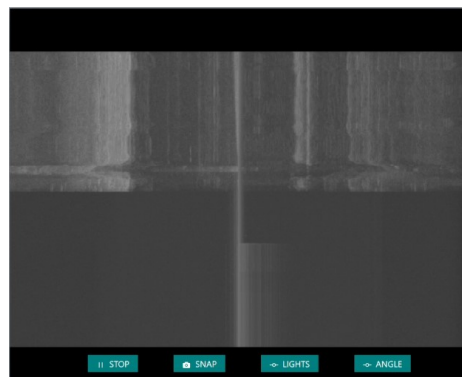
Figure 4.24: Dashboard VIDEO: the lower section of the groundtruth scan

4.6.3.2 Important settings for the Side-Scan Sonar

Some of the most important settings in the API is listed below.

- **Range**, the range of the side-scan sonar is between 10 and 500 meters. The range is in the width of the beam, not in-depth below the sonar. The depth below the sonar should be 10% of the range.

- **nSamples**, is the number of samples on each side, and with range this is what affects the resolution of the sonar image, $\frac{Range}{nSamples}$.

- **Frequency**, the side-scan sonar can operate on both 340 kHz and 680 kHz; the higher frequency will give a better-detailed image (340 kHz, 1.5 cm resolution) (680 kHz, 1 cm resolution) while the lower frequency will provide a better range (340 kHz, 15 to 200 meters range) (680 kHz, 10 to 100 meters range).

4.7 ROV Implementations

4.7.1 Selecting between Penetrator and Connector

The ordered tether cables (150 and 300 meters) comes with penetrators attached. Only apply thread seal and mount it to the ROV. The disadvantage of penetrators is that the cable cannot be swapped fast, and when carrying the heavy ROV, the cable is often in the way and makes the moving harder.

The alternative to this is using connectors, making it possible to switch cables fast and disconnect the cable when moving. Mounting the connectors take hours of work, but the upside with them is more significant than the downside; connectors was therefore added to the two new tether cables. Initially, we wanted connectors for our side-scan sonar, but since the electronic box for the side-scan was not potted, we could not place it outside of the ROV and send signals via Ethernet into the ROV. Therefore the electronic box had to stay inside the ROV and two cables with a high-frequency signal going inside the ROV; due to recommendation from the manufacturer that the connector could make noise on the signal, the connector was not chosen.

4.7.1.1 Implementation

Soldering the connector requires some experience with soldering; this is because the wires had to be as short as possible to get the connector more waterproof. When soldering the connector, it is also essential to keep in mind the temperature on the soldering iron and the amount of time used to solder the connection. The isolation on each wire tends to melt fast. Figure 4.25 shows the soldered connector. The colour code for the pin is shown in Figure 4.26, which corresponds to the male side of the connector, which comes pre-soldered.
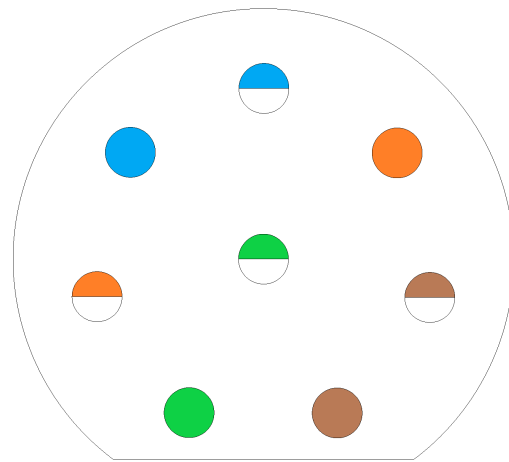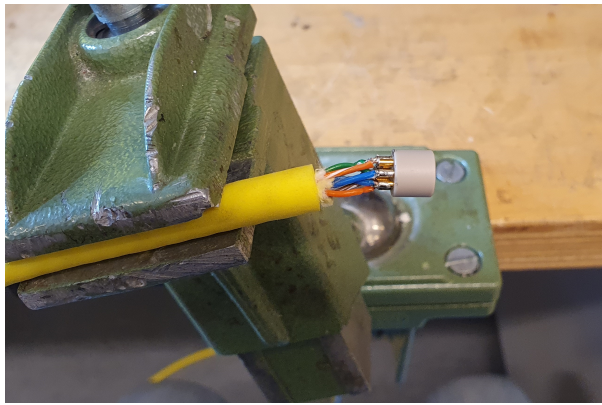
Figure 4.26: Female connector to ROV

The epoxy "3M™ Scotch-Weld™ uretanlim 620NS" is used for sealing, and it is chosen for its flexibility and softness. Harder epoxies used in the earlier iteration of this projects tend to crack. The epoxy was heated with a heat gun to get a smoother finish and make the epoxy more fluid. A needle was used to poke air bubbles out. Figure 4.27 shown the sealed penetrators.
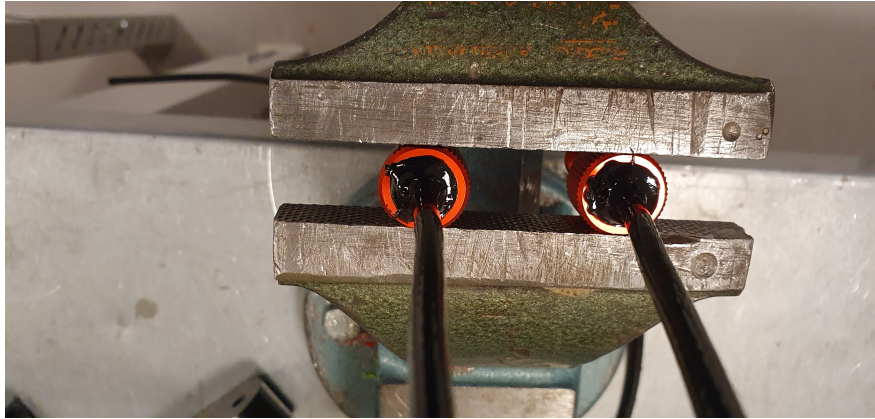
Figure 4.27: Penetrators sealed with epoxy

4.7.2 Sensor modularity

One of the potential use of the ROV is to collect sensor data with a different type of sensor attached to the ROV. Sensors can be expensive, or they are just needed for a short time. By making the sensor system modular, these can be easily attached and removed for other projects. These sensors will primarily be connected to the one ArduinoSensor. Considering the number of protocols and possible connection method of a sensor, not all can be considered. A selection of easily accessible ways to read sensor data is made and shown in 4.1.

Type	Pin
Analog	A0
Analog	A1
Analog	A2
Analog	A3
Digital	D2
Digital	D3
Software UART	11, 12

Table 4.1: Modular sensors

As described in the Settings page 4.4.1.2, adding a new sensor to the system requires it to be connected and adequately sealed physically. Then a name for the sensor, its origin (ex: ArduinoSensor) and port it is attached to is typed in the GUI. The GUI will make the REST-API cre-

ate and send a payload (Payload A1/A2, Figure 4.28) to the Raspberry Pi, where it is forwarded to its destination. The Raspberry Pi will also store the variable name for filtering out erroneous sensor data. When it arrives at its destination, the given port will check its availability and start sending data with the given variable name. The new sensor data will be added to payload B along with all other sensor data. This process is showed in Figure 4.28.
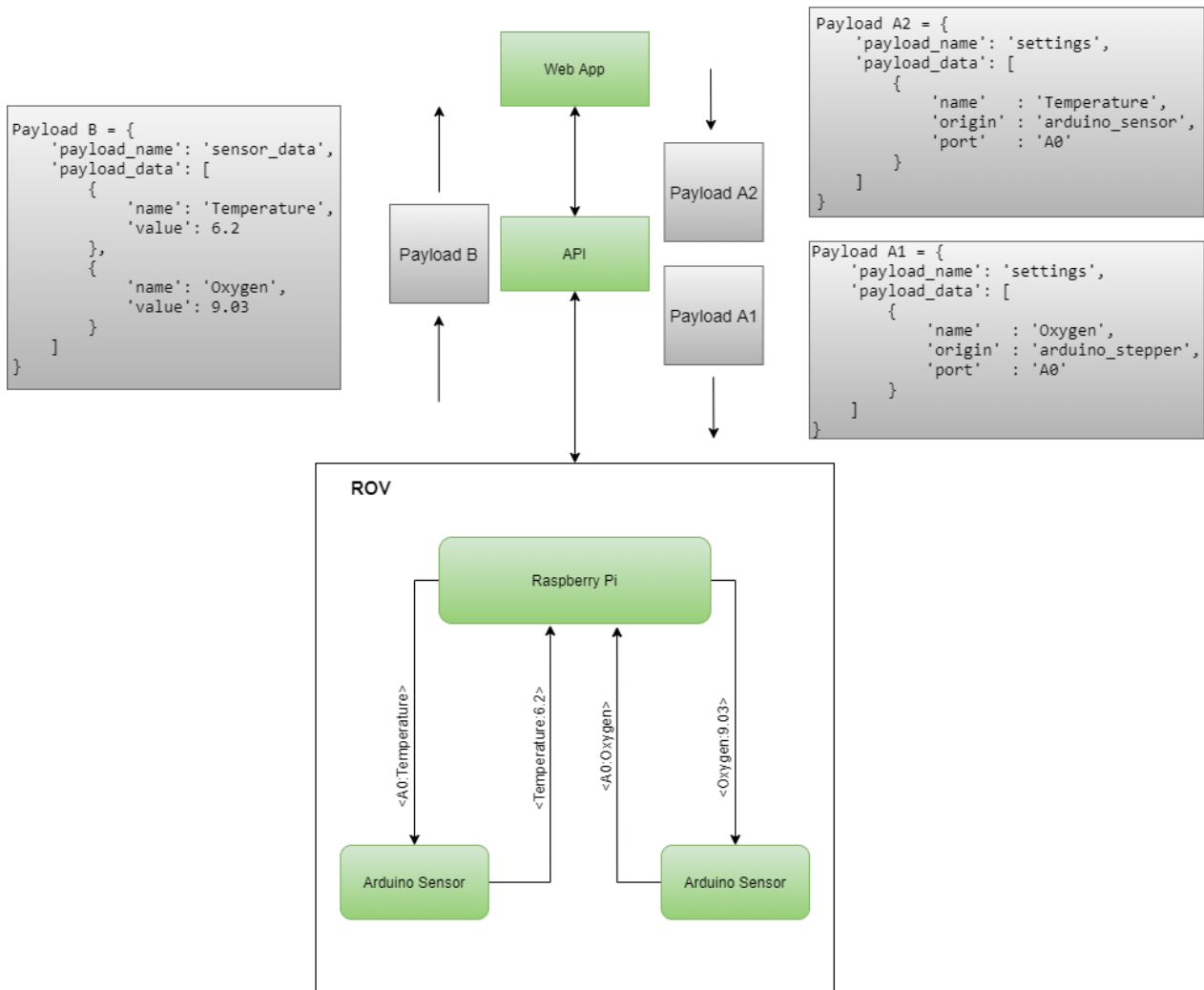
Figure 4.28: Modular sensor flow

4.7.3 Lights

From the previous ROV only one light was working, and before testing it properly, we found water leaks inside. Therefore four new lights were purchase and mounted. With four light sources, the lighting should be more even for the video, and in total, the four light can produce 6000 lu-

mens [52]. It should consume around 60 watts with maximum brightness, meaning at 12 volts, the current should be about five amps. Although the lights are rated for 10-48 volt input power, the voltage it will be connected to is 12 volt. The reason for that is the variance in the voltage from the batteries, which should be around 48 volts. The lights are depth rated for 500 meters and are using a PWM signal to control the brightness. Brightness can be dimmed from 1100 μS (off) to 1900 μS (brightest). The ground from 12 volt needs to be connected to the ground on RPi to control the light; else, it will be unstable due to grounds with different potentials. The PWM signal needs to be connected to a pull-down resistor to the ground to avoid floating output signal at startup.

Figure 4.29: Lumen Subsea Lights R2 [52]

4.7.4   IMU

The IMU is used to estimate the angular position of the ROV. The angular position is required for controlling the trim angle and is essential when evaluating the ROVs behaviour.

Choice of IMU and Sensor Fusion algorithm

The IMU from the previous project had to be replaced as the USB connector fell off, and the IMU is out of production. The Adafruit 9-DOF was chosen as it is easily available and is accurate enough for its purpose.

First implementation

The first algorithm used in the project were Adafruits 9-DOF library for Arduino. The measurements were noisy, and after some research, it was discovered that the library only used the accelerometer and magnetometer to estimate the angles. The 9-DOF library reads sensor values from the accelerometer and magnetomer and fuses the senors by applying equations 4.2, 4.3 and 4.4. where θ is the Euler angle for roll, pitch and yaw, a is accelerometer reading and m is magnetometer reading.

$$\theta_r = atan2\left(\frac{a_y}{a_z}\right) \tag{4.2}$$

$$\theta_p = atan\left(\frac{a_x}{a_y\sin(\theta_r) + a_z\cos(\theta_r)}\right) \tag{4.3}$$

$$\theta_y = atan2\left(\frac{m_z\cdot\sin(\theta_r) - m_y\cdot\cos(\theta_r)}{m_x\cdot\cos(\theta_p) + m_y\cdot\sin(\theta_p)\cdot\sin(\theta_r) + m_z\cdot\sin(\theta_p)\cdot\cos(\theta_r)}\right) \tag{4.4}$$

Second implementation

To get a better orientation estimate, the gyroscope is added. To get the orientation, the measured value is integrated. As there is bias in the measurements, it is essential to calibrate the gyro. As therein practical use always will be some drift and the initial state is unknown for the gyroscope, it is fused with the accelerometer and magnetometer.

Calibrating the gyroscope

The purpose of calibrating the gyroscope is to remove the bias in the measurement 2.1.1.1. Calibrating can be done by printing the mean value of 500 readings 2.4 and subtract that value from each reading.

Complementary Filter

The complementary filter is implemented in Arduino and is updated every 20ms. The discrete implementation can be viewed in listing 4.1. The complimentary is tuned by changing the value of alpha, alpha is limited to $[0 \leq \alpha \leq 1]$. Where a high alpha value will increase the contribution from the gyro, while a low alpha value will increase the contribution of the accelerometer.

Listing 4.1: Complementary filter

```
float complementary_filter(float alpha, float prev_angle, float gyro_value, float
    accel_value, float dt) {
  float angle = alpha * (prev_angle + gyro_value * dt) + (1 - alpha) * accel_value;
  return angle;
}
```

4.7.5   Seafloor tracking

When monitoring the seafloor, it is crucial to keep a consistent distance from the seafloor to ensure that the pictures from the side-scan sensor and the camera are as straightforward as possible. At the same time, the setpoint should not be changed too often, as moving to a new depth will cause some instability in the ROV that can affect the recordings. Another aspect is collision avoidance to prevent damaging the ROV.

Test class

A test class is made to evaluate the results of the algorithm. A semi-random seafloor is generated with parameters for length, minimum depth, maximum depth and noise in measurements. While the ROVs depth is simulated with a simple linear function to see how the algorithm handled different scenarios.

Implementation

The algorithm relies on updates from the echo-sounder and GPS mounted on the boat for es-timating the depth of the seafloor and the systems movement. As the echo sounder is a single beam, the boat needs to move in a straight line. The measured depth of the ROV is also im-plemented for finding the optimal set point and to send an alarm if the incline is too steep for the ROV to avoid a collision. As the system behaviour may vary based on physical changes, and different mapping sensors require different distances, there are three parameters for adjusting the algorithms' distance and sensitivity.

- Desired distance:

  The desired distance from the sea floor.

- Minimum distance:

  Minimum allowed distance from the sea floor

- Distance to ignore:

  The minimum deviation for changing set point.

There is a method implemented for setting the parameters while operating the ROV. There is also a method for changing the array of future set points, as the distance between the boat and ROV can vary according to depth as calculated in 4.5.3.

To keep track of the ROVs movement relative to the depth measurements, a recording of sonar values is sent down every 10th meter from the Surface Unit. The sonar values are then sent to a cost function finding the optimal distance according to the measurements and parameters. The new optimal distance is added to an array, including every optimal distance between the ROV and boat. This array is then evaluated by a method deciding on a new set point based on the parameter values. Figure 4.30 gives an overview of selecting a new set point.

10 meters of
sonar values

Cost function finds a legal
value.

No

Alarm to GUI

Yes

set points[0] is deleted
set points[end] = new value

Rov depth - min_sp
<
max incline limit

No

Yes

current sp-min sp
>
desired distance - min distance
or
depth rov - min sp
>
desired distance - min distance

Yes

set points array
=
lowest value

No

dist to ignore
<
depth ROV-mean val
>
depth ROV - current sp

Yes

set points[0]
=
mean value

No

abs(set point-mean)
>
distance to ignore

No

set points[0]
=
current sp

Yes

mean value
<
current set point

Yes

set points[0]
=
mean value

No

mean value - C_SP
<
desired distance - min distance

Yes

No

set points[0]
=
min sp

Figure 4.30: overview of the selection of a new set point

4.8   Method for Sea Trial

As the results from the sea trial were evaluated as we were working, there were several changes in the ROVs design to change the behaviour of the ROV.

4.8.1   Physical changes

Physical changes of the ROV in this project.

Figure 4.31: CAD model of ROV

4.8.1.1   Adjusting buoyancy and stability

To make the ROV more stable, the center of gravity and buoyancy should be in the same x,y coordinate, so the torque is kept to a minimum. The center of buoyancy should also be placed over the center of gravity, this will give the same effect, but it will only occur when the ROV is rotated, forcing the ROV back to the wanted position 2.12.2. To allow the ROV to go deeper, the ROVs total buoyancy should be closer to zero. To decrease the positive buoyancy, the ROV is filled with oil. As the ROV can fit 30 litres, buoyancy would become negative, and there is a need for compensation. Two pipes with a diameter of 160mm are therefore sealed and mounted on top of the ROV. The total buoyancy of the ROV is found by subtracting the total weight from the net buoyancy. The center of buoyancy can be adjusted by moving the pipes along the ROV. Together all of these factors will increase the stability of the ROV.

4.8.1.2 Side plates and Spoiler

To make the rov more stable, side plates were made to prevent the ROV from rolling. The side plates are cut out of an acrylic plate. An adjustable spoiler was 3D printed, designed and made by the cooperating group working on a new ROV design. The side plates are glued to the ROV with tec7.

4.8.1.3 Tail fin

A one-meter long acrylic plate, stiffened with a din rail, is added for pitch stability. This plate is mounted with screws to another plate glued to the ROV body. The glued plate is mounted with brackets and screws to the sideplates. The fin will act as a damper and remove some of the disturbance in the pitch.

4.8.1.4 Design of new wings

New and different wings for testing is made of acrylic plastic and a 3D printed bracket for mounting. Using the laser cutter and 3D printer makes the time to create new wings much faster than the old ones, made out of a hollow 3D printed part filled with two-component polyurethane and added a bracket for mounting. This process is time-consuming but allows for wings with a 3D profile; if there is an advantage of using the old NACA profile wing over the new 2D wing is to be tested. The idea behind the new wing is to have the centre of rotation in the middle, so the force pushing on the upper side of the wing is equal to the force pushing on the lower side of the wing. Therefore, the only force needed to rotate the wing is only the displacement of water and the force loss in the transmission.

4.8.2 Flow simulation of wings

To validate the new wing assumption , a flow simulation is made in Solidworks flow simulation. The simulation is done with seawater ($1025kg/m^3$) with a flow in X-direction of $1.5m/s$. Since our CAD models are made in Fusion 360, models are converted to STEP-file and then imported.

4.8.3 Troubleshooting

Whenever issues arose, various strategies have been used to troubleshoot hardware areas.

4.8.3.1 Measure DC voltage using oscilloscope

For measuring the DC voltage on the custom PCB board, an oscilloscope and a measuring probe with the GND is needed. The GND lead will be attached to the negative potential, while the probe to the positive potential, in our case 5/12V. The oscilloscope will read the positive voltage from the supply and shown in Figure 4.32, switching the position of the will give us the negative voltage. Adjust the ground reference by pressing the GND button on the oscilloscope and rotation the position knob. The GND position is adjusted to -5 volts, so when measuring 5 volts, it should appear in the centre of the screen.
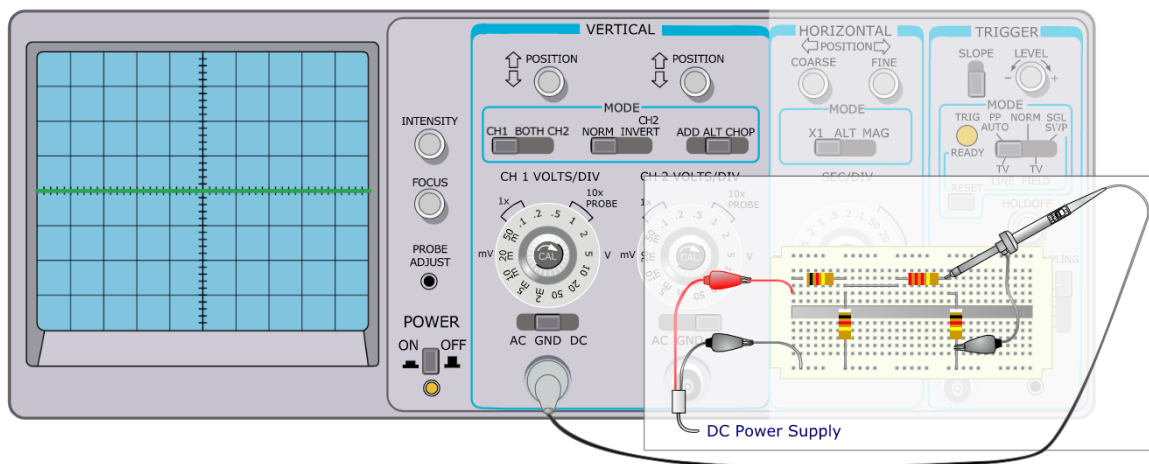
Figure 4.32: Measuring DC voltage with oscilloscope [19]

4.8.3.2 Testing conductivity of the oil

As we had various problems with the electronics in the ROV we tested the conductivity by using an isolation tester. The oil was moved to a bucket to prevent damage to the electronics. The test voltage was set at 1000V and the resistance were measured to 148MΩ when there were a couple of millimeters between the test leads.

4.8.3.3   Validating Tether performance

Stable network connectivity is required in order to develop a control application for the ROV. Since the physical connection between the ROV and the Surface Unit is a tether cable, it is necessary to validate the performance of the cable and its ethernet speed throughput. Given the modularity of the ROV, it will carry a large number of sensors, thereby enforcing a solid data-transfer throughput. Using bandwidth testing software such as iPerf 3.5.2.14 one can measure network performance.

4.9   Digital Twin

The simulation in AGX Dynamics is a further development of the simulation made by a current group member in a previous course. The simulation mainly lacked two things; the ROV's behaviour in the simulation did not correspond with the ROV's behaviour at sea and could not be connected to our software to simulate functions and features. On the other hand, the previous project had already added a functioning Towed system with simulated cable, rov body with wings and a boat that could tow it.

The AGX simulation can simulate simple hydrodynamic effects on the ROV. The depth of the ROV is controlled by the hydrodynamics on the wings and the ROV body. Simplified versions of the CAD models represent the ROV body and the boat. The sonars in the boat and the ROV are simulated using an AGX height field, the speed of the system is measured in relation to water, and the depth/pressure sensor is modelled using the position of the ROV in the z-direction. When creating simulation software, parameters in different places are changed frequently. Because of this, it is practical to add a python module with constants for values like hydrodynamic parameters, wire length, depth of the sea, size and location of the simulation models.

4.9.1 Seafloor

Building seafloor

The previous simulation had no actual seafloor simulation. To do a proper simulation of, for example, depth control, this was needed. To create a seafloor, a black and white image was generated representing different heights and depths of the seafloor; see Figure 4.33. This image will be used by AGX to generate a height field, a plane that only varies in the z-direction and is constant in x and y. When the water and seabed have been added to the simulation, it should look something like Figure 4.34.
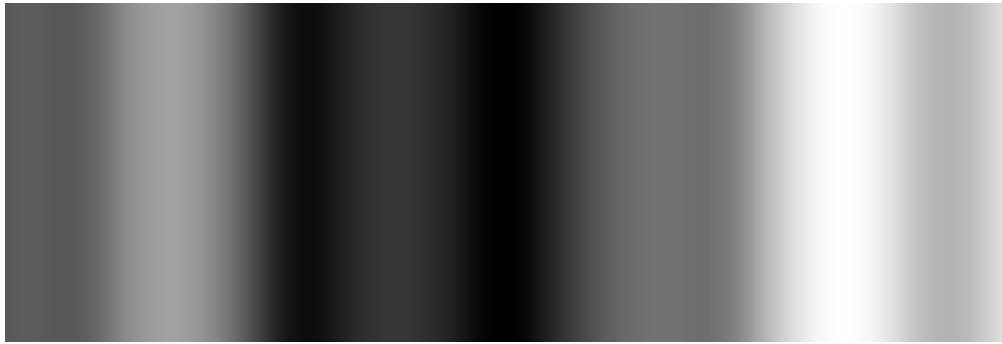
Figure 4.33: An example of an image used to generate seafloor in the simulation
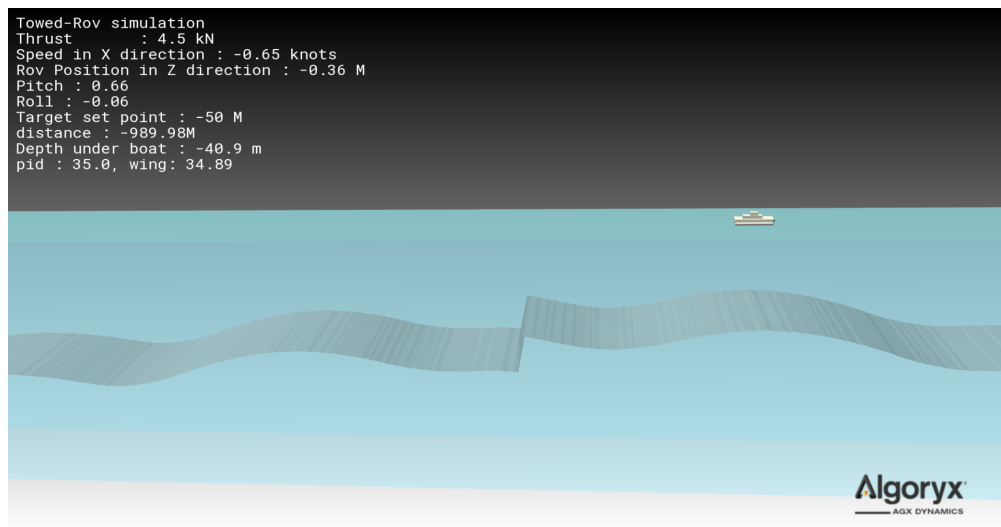
Figure 4.34: image of the simulated water, with a seabed and the Boat in the left.

4.9.2   Building model for simulation

Importing models

When building the model for hydrodynamic simulation, it is essential to simplify the model given to the simulator; this will increase the performance and the accuracy of the system because the simulation has to calculate the forces on every polygon in the model. The ROV model from the previous project was a severely downscaled version of the original CAD file, but not a new file. Because of this, the model was full of holes, weird angles and overlapping polygons. It was, therefore, necessary to create a new simulation model. When building this model, ignore complex and small parts like bolts, screws and small pipes, anything that does not affect the hydrodynamics in a significant way. After the simplified body is built, it should be exported as an *.OBJ* file. This file is made of triangles representing the polygons in the model. Using the AGXMeshReader class, the Models' mesh file can be imported to AGX as an AGXMesh object that can be used to build a Geometry object. The Geometry class specifies a material, the weight, shape of a component in AGX.

Internal models

AGX has some basics shapes that it provides internally. The simplest of these shapes are optimised for hydrodynamic calculations [6]. When adding the floating elements on the ROV in the simulation, the internal Cylinder class was used.

Assemblies

An assembly consists of several different rigid bodies and event handlers and constraints. When adapting the assembly from the previous project changing out parts was a change since all the position parameters were individualized for that group's specific model. Therefore the movement system for the wings and the positioning system was modified so that they work of the dimensions of the new model. The positions of the wings, for example, are now given by the dimensions of the ROV, as is the tanks that have been added. In Figure 4.35 you can see our new assembly of the ROV body.
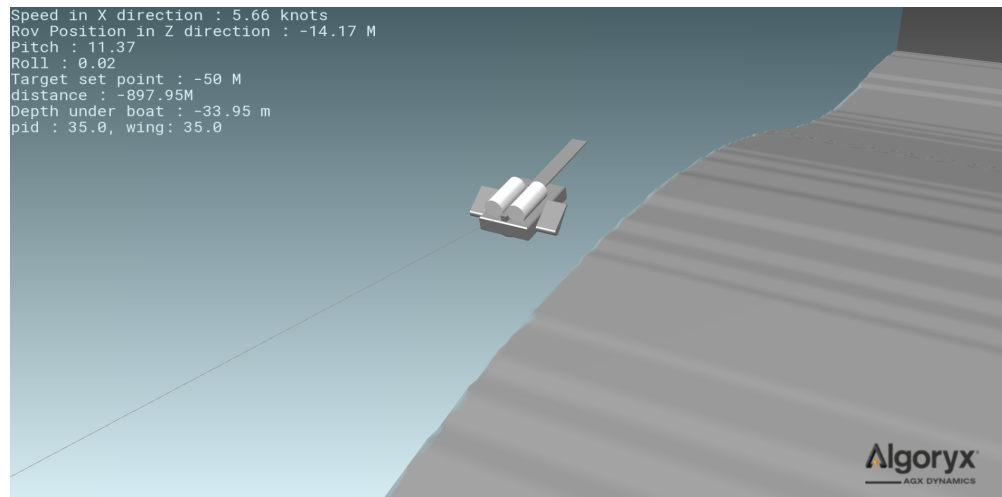
Figure 4.35: The simplified ROV body, for simulation

4.9.3   Sensors and Stepper motors

To simulate the sensors and Arduinos in the system, Event listeners are used. Event listeners in agx are classes that execute their methods when certain events happen. The Arduinos written in Arduino could be implemented in two way. Either translate the code to Python or use a Python wrapper for C++. Either way, both the Arduino code had to be edited to receive sensor data from ROV and control the actuator. The functionality left in the Arduino is mostly just the syntax to make it work in Python, and since the group had little experience using a Python wrapper, translating the code was therefore chosen. The PID library from the real ArduinoStepper has previously been rewritten into Python.

Step Event

The ArduinoStepper, ArduinoSensor and the boat_sensor classes are instances of StepEventListeners. Once every step in the simulation, these execute their pre() and post() methods. Using the pre method, we can receive data, update controller outputs and send or receive sensor data. Using the post method, we can set visualisation and log data. The pre and post methods are predefined by AGX to include a "time" variable; this variable represents the "in simulation" time. When building a sensor for Digital Twin in AGX using this variable, the Arduinos and boat_sensor is made to act in the same time cycles as it would in the actual world. For example:

The echo sounder in the surface unit updates once every half-second, so it should only act when the condition $(round(time, 2)\%0.5 == 0)$ is true. The example below shows a simplified example of how the StepperArduino could use the pre-function to control the wings of the simulation. It is worth mentioning that the virtual Arduino needs a loop to send its start message "SensorArduino:0" or "StepperArduino:0". The reason for the loop is because the virtual Arduino does not reset every time a new serial connection is established, which is the opposite of the real Arduino, making it able to send one message every reset.

4.9.3.0.1 Simulating Echo sounders

To simulate the echo sounders in the actual system, the Height field seafloor is used. To get the boat's depth, input the x and y position of the boat as projected on the Height field in the $getHeight(x, y)$ method.

4.9.4 Communication

When building the digital twin, the goal is that the rest of the system should not notice that it is not a physical ROV it is controlling. Therefore, the simulation needs to use the standard communication method. When simulating locally on a computer, use ZMQ 2.8 with localhost, and serial communication with regular ports using software like the one described in 3.5.2.9.

NMEA over serial

When the digital twin sends messages to the Surface unit, the message needs to be in the NMEA style. Since the data required to run the system consists of depth, longitude and latitude, we construct sentences for these. format a string in the NMEA style seen in 2.1 but without the "$" and * " characters. Then add the depth as in meter as a parameter, calculate the checksum, add the "$" and "*" chars to the start and end of the string, then add the checksum and encode the message.

4.9.5   Simulating system on computer

An AGX simulation incorporates these objects and calculates the change in the system over a time unit. A higher time unit will cause a loss in stability and accuracy but will drastically improve performance. If the step time increases over around $0.01sec$ the simulation will fail even in the most stable condition without complex calculations. With complex simulations like this system, 0.005 seems to be a decent value. When simulating the system, remember to start the AGX simulation and the surface unit code first since this will limit the search ports for the serial finder in the ROV software 4.3.3.1 to the simulated system.

4.9.6   Behavior

The Simulated ROV can be tested by switching different models, simulation parameters[3]. To make it easier to check with different systems, a parameter Python Module was added. Most parameters can be set from this module, like PID parameters, speed, water and seafloor dimensions, ROV and wing scale, and model file names. This streamlines testing with different system parameters.

4.9.6.1   Hydrodynamic parameters

After creating the wire and the shapes of the models, the models need to implement hydrodynamic parameters. These parameters depend on the physics of the model. These parameters can be estimated using computer simulation.

Caluclating the Drag coefficient

By default, the pressure, viscous drag and lift coefficients are set to 0.6, 0.1 and 0.01, respectively. They can be adjusted to simulate our ROV drag coefficient. For this, we are using Solidwork's flow simulation. The simulation uses a flow of water at the velocity we normally run the ROV in, 3.5 knots. Simulation of the force acting on the ROV with the given velocity will give us the total

---

[3]for example hydrodynamic parameters, max wing angle, or speed.

drag coefficient. To calculate the drag coefficient, we use the formula below.

$$C_D = \frac{F_D * 2}{A * \rho * V^2} \tag{4.5}$$

where

F_D is the drag force

C_D is the drag coefficient

A is the reference area

ρ is the density of the fluid

V is the flow velocity relative to the object

This gives us the result in table 4.2

Goal Name	Unit	Value	Averaged Value	Minimum Value	Maximum Value
GG Force (X)	[N]	71.18001305	72.00554579	71.10082164	72.82404687
Equation Goal	[N]	0.771599057	0.780547922	0.770740614	0.789420562

Table 4.2: Drag coefficient

A total drag coefficient of 0.771599057. The velocity flow around the ROV at 3.5 knots are shown in Figure 4.36
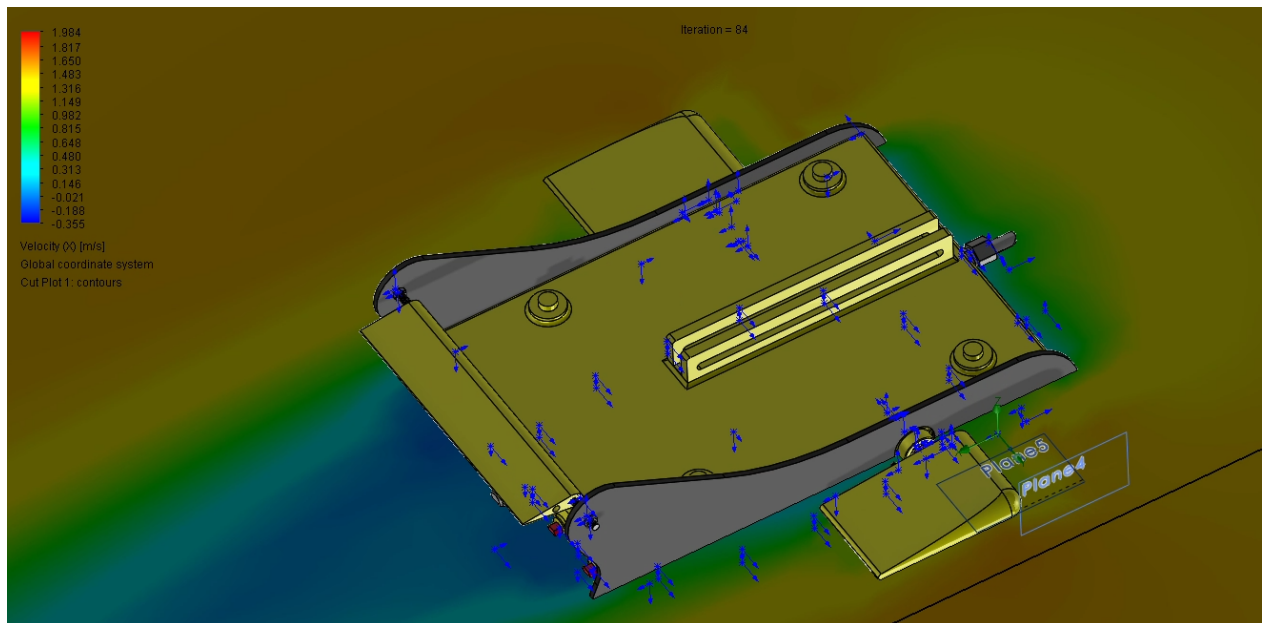
Figure 4.36: Velocity flow around ROV

Chapter 5

Results

The final results of the proposed implementation and the results of various studies are presented in this chapter. The first sections present the final implementation of the project's elements, while the later sections demonstrate the outcomes of various tests and sea trials.

5.1 Software solutions

This section explains the final details of the software system and how its implemented.

5.1.1 Architecture

The final software solution is divided into six separate systems, which builds the project's final architecture. In Figure 5.1 below, the flowchart shows how the communication protocols working together to send and receive data. The final software architecture follows the three-tier architecture as explained in 2.4.1 and shown in 4.3.
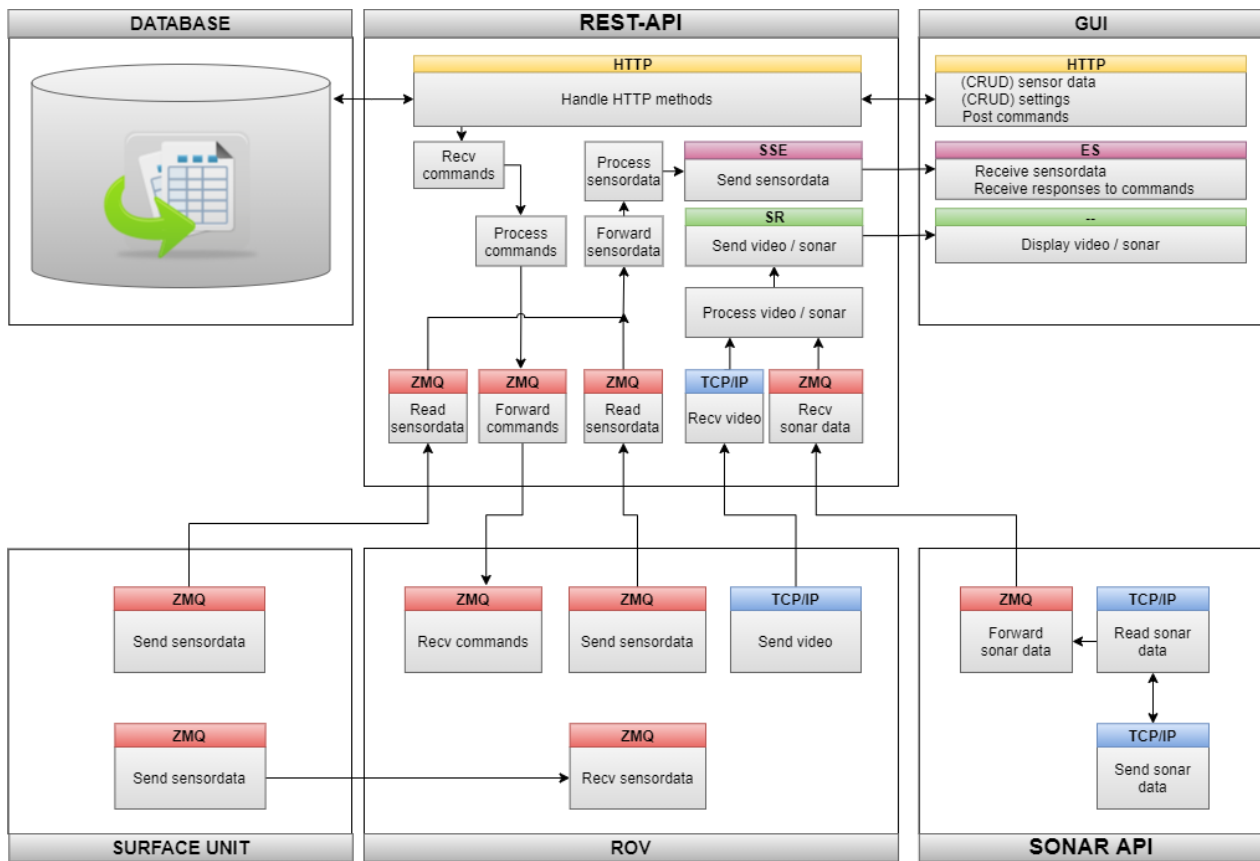
Figure 5.1: Data communication

5.1.2 Graphical User Interface

The Figures below shows the final desktop application developed, referred to as the GUI. The desktop application consists of four pages available to the operator. The initial loading page also called the landing page, is seen in Figure 5.2. The next following sections below will show the final design and functionality in the various pages of the application.
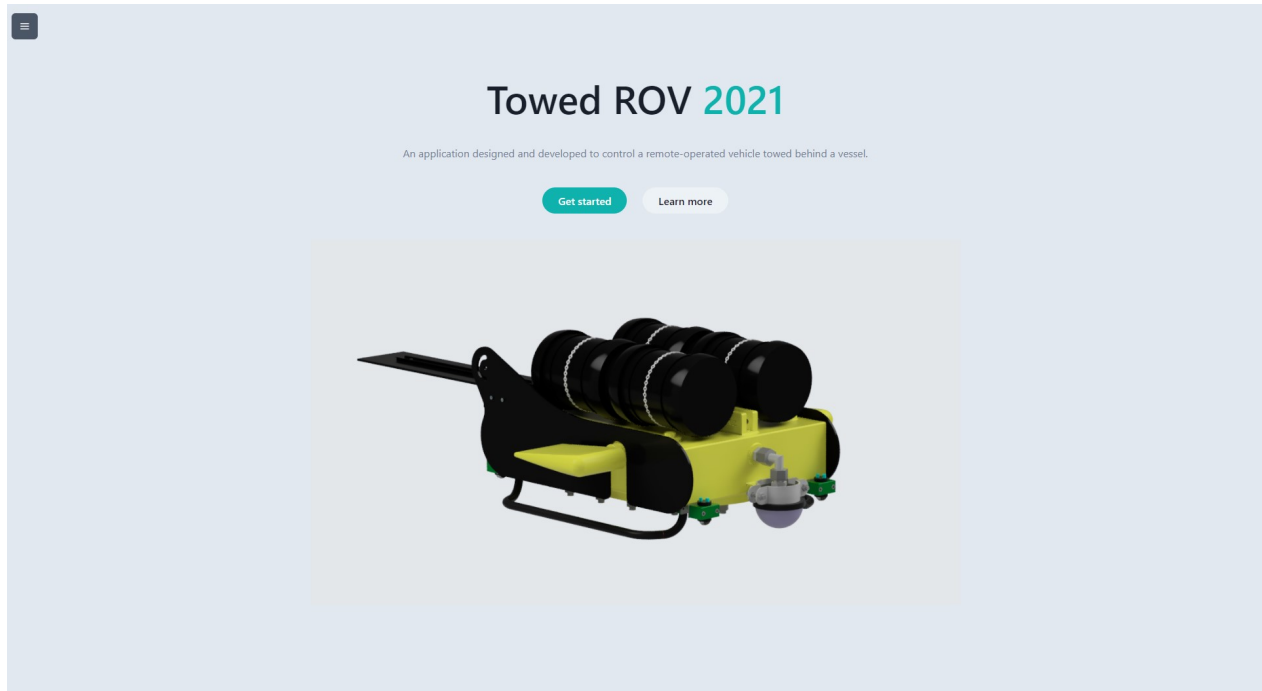
Figure 5.2: Landing page

5.1.2.1 Pagination

As briefly mentioned in the structure section of Presentation-tier 4.4.1.2, the application uses 'Route' as a common way of defining a page. Below is the base structure of the GUI, which uses various pages for displaying specific elements of the system as components. Each component was explained in details in the structure section of the Presentation-tier.

Code examples from Appendix [J] shows the outline of the React GUI,

```
function App() {
  return (
    <HashRouter>
      <SettingsProvider>
        <NavIcon />
        <Switch>
          <Route path="/" exact component={Home} />
          <Route path="/settings" component={Settings} />
          <Route path="/dashboard" component={Dashboard} />
          <Route path="/map" component={Map} />
          <Route component={NotFoundPage} />
        </Switch>
      </SettingsProvider>
    </HashRouter>
  );
}
```

The respective routes are accessible at the 'Hamburger-menu"-icon in the top-left corner and available on every page. On click, a navigation bar is presented to the user, which helps navigate between pages as shown in Figure 5.3 - starting from left to right: Settings, Home, Dashboard, Map and Dark-Mode toggle.
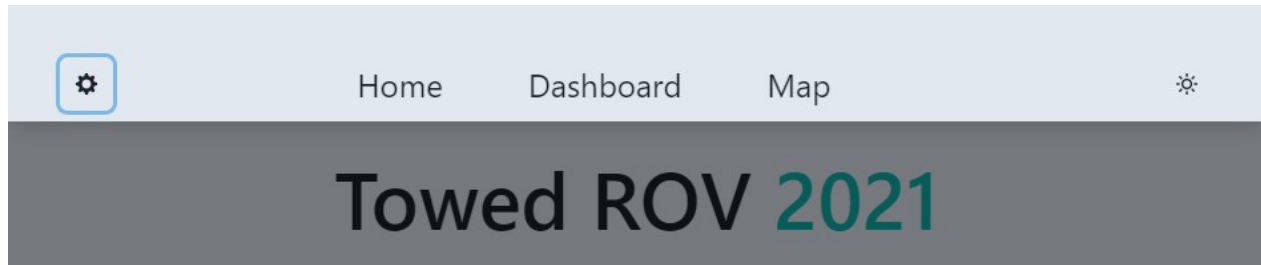
Figure 5.3: The navigation bar

5.1.2.2 Modularity

Before starting any sea operations, the operator can adjust the settings to modify the additional modular sensors added to the ROV. As the ROV platform has a certain amount of base sensors, the **Settings** page provides the user to use their specific sensors. In the **Settings** page, the user can view the stored modular sensors saved in the database. When loading the page, a list of sensors is shown (as seen in Figure 5.4). As the user leaves the pages, the settings are stored in a local context manager, so when the operator enters the **Dashboard** page, he will have the option to send the settings to the ROV. The ROV will then receive the settings and create the specified sensors at their respective locations. Explained more in details in section 4.7.2.
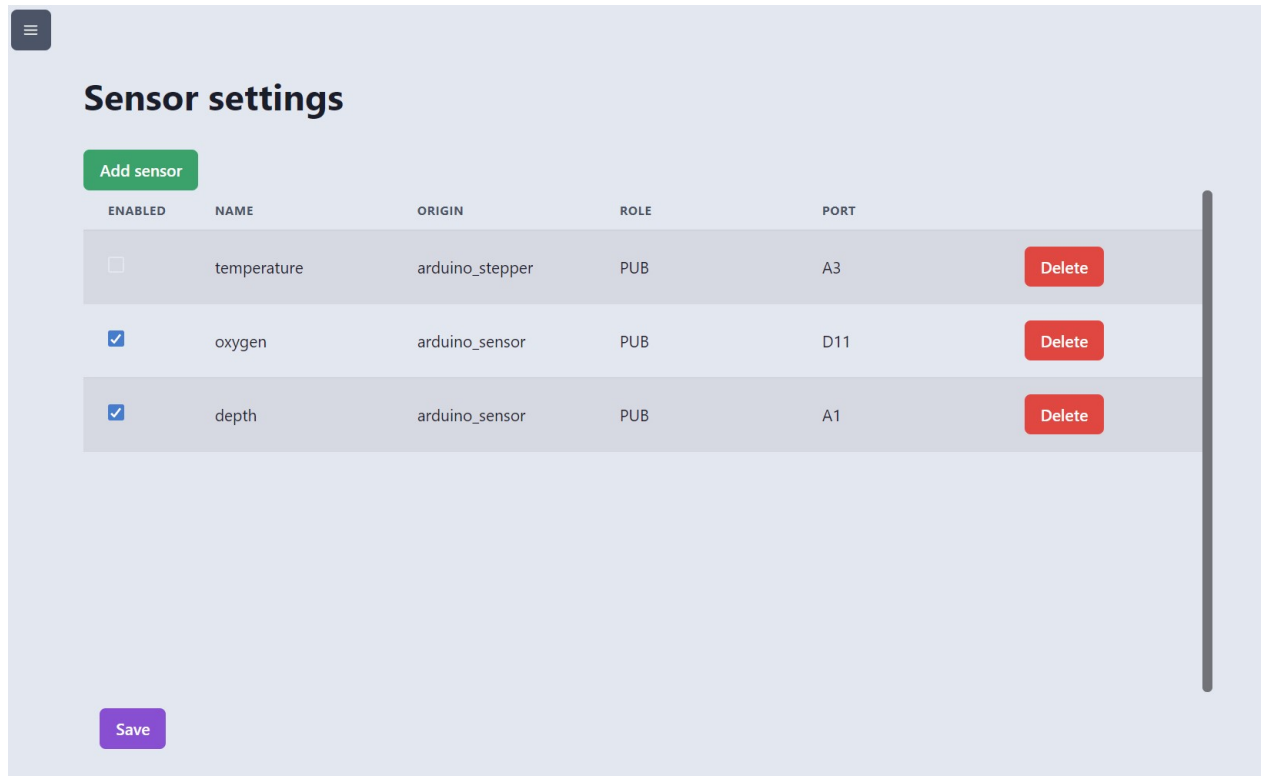
Figure 5.4: Settings: the saved modular sensors (Edit-mode is activated)

To create a sensor, the operator can click the Edit button to go into editor mode. Once the operator is inside the editor mode (as seen in Figure 5.4), the operator can add or remove new sensors and toggle whether they should be enabled or not. Modification made to the sensors is immediately updated in the database. Therefore the data is always similar to what is shown to the operator. When creating a sensor, as shown in Figure 5.5, the sensor's name is required, an origin (the hardware the sensor is connected to), and the specified port connected. Optionally, the system also accepts a specified role, i.e.: publisher.
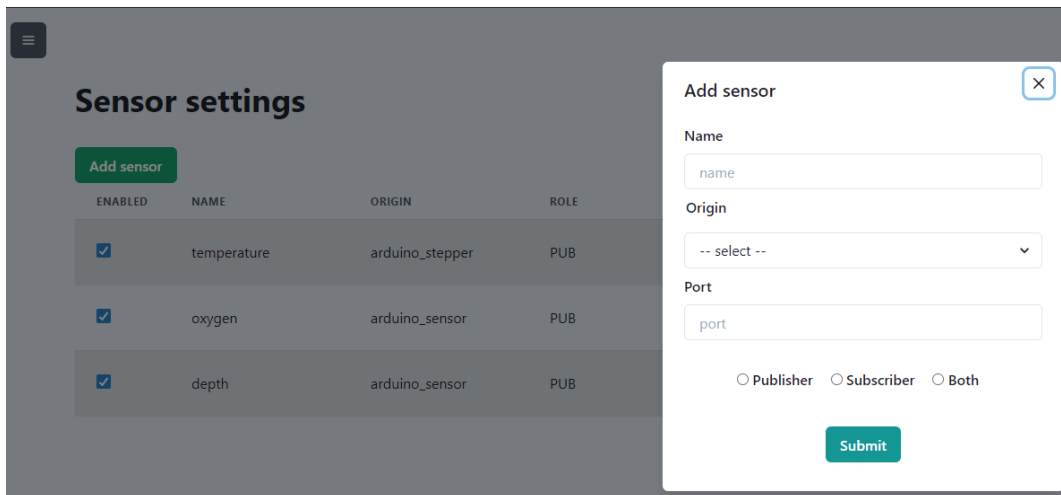
Figure 5.5: Settings: adding a new sensor

Lastly, we can observe with the updated sensor settings in Figure 5.4, where we have sent and received the two new sensors, showing in the metrics in Figure 5.6 below.
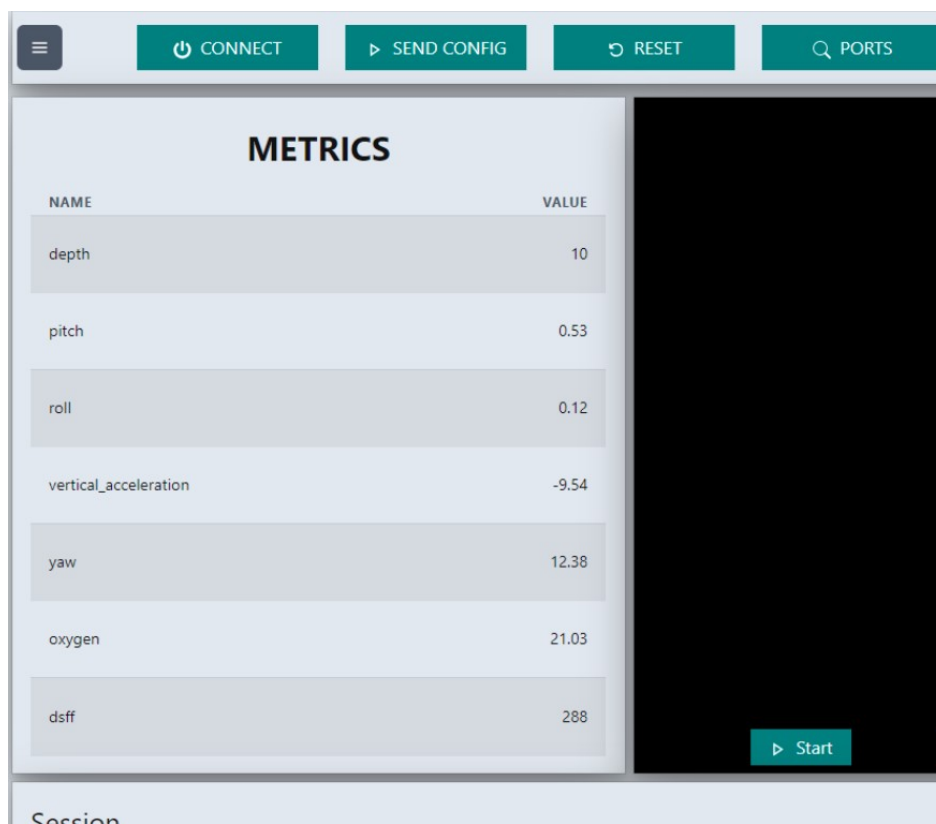
Figure 5.6: Dashboard: with newly added sensors

5.1.2.3 Dashboard

Figure 5.7 below shows the main component of the desktop application, **Dashboard** page.

This is where the operator has control of the ROV during operations.

The Dashboard consists of a top-level system control bar to initiate various system commands such as *CONNECT*, *SEND CONFIG* (sends sensorsettings), *LOGGING* (starts CSV-logger) to mention a few.

On the upper left side, the operator has a live feed of all the metrics measured in the system. This metric box involves data from all the connected sensors in the system.

The centre black box is the video window[1] where the operator can control various aspects related to viewing the video from either the camera or the sonar.

The upper right side displays a command box that accepts operator commands to control sensor values in the system. Above the input field, the user can view any sent command as well as their responses.

The bottom left section allows the operator to record "Sessions" whenever he wants to record the sensor data and video data to the database.

The bottom right section helps the operator visualize the sensor data in a historical plot, where the specific sensor values are saved up to the last 60 seconds (i.e., depth).

---

[1]**OFFLINE IMAGE**: camera and side-scan sonar were offline due to hardware complications
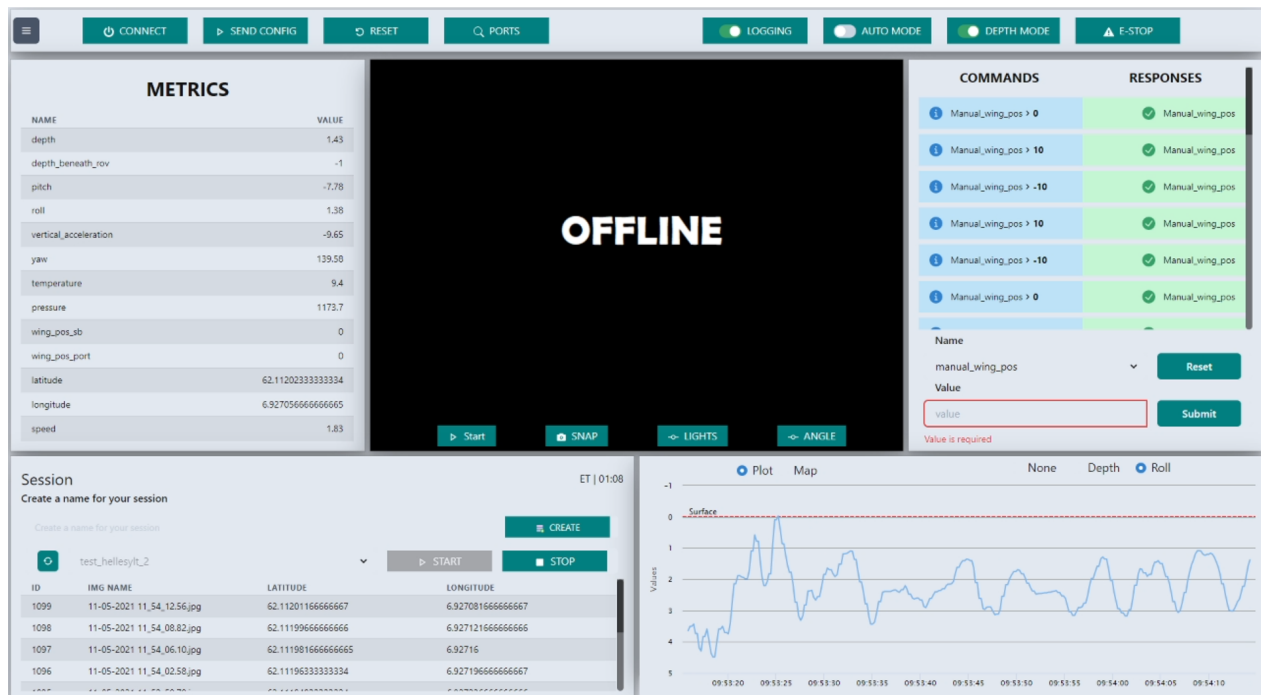
Figure 5.7: The Dashboard page during a live ROV operation

Paragraphs below this section will show the final workflow results from the most relevant sections of the **Dashboard** page's functionality.

Displaying a videofeed

In Figure 5.8 below, the operator can toggle between viewing the video-feed[2] generated by the camera attached to the front of the ROV or the side-scan sonar attached on the external body. Using either of the two, the operator can also use the *SNAP* options to take a snapshot of the current image in view if he will discover something interesting. The snapshot-functionality is an extra option to save image data if something interesting is spotted and the operator is not running a waypoint session.

The two buttons *LIGHTS* and *ANGLE* is mainly used for adjusting the brightness of the lights attached to the ROV front-side and the view angle of the camera mount.

---

[2]**OFFLINE IMAGE**: camera and side-scan sonar were offline due to hardware complications
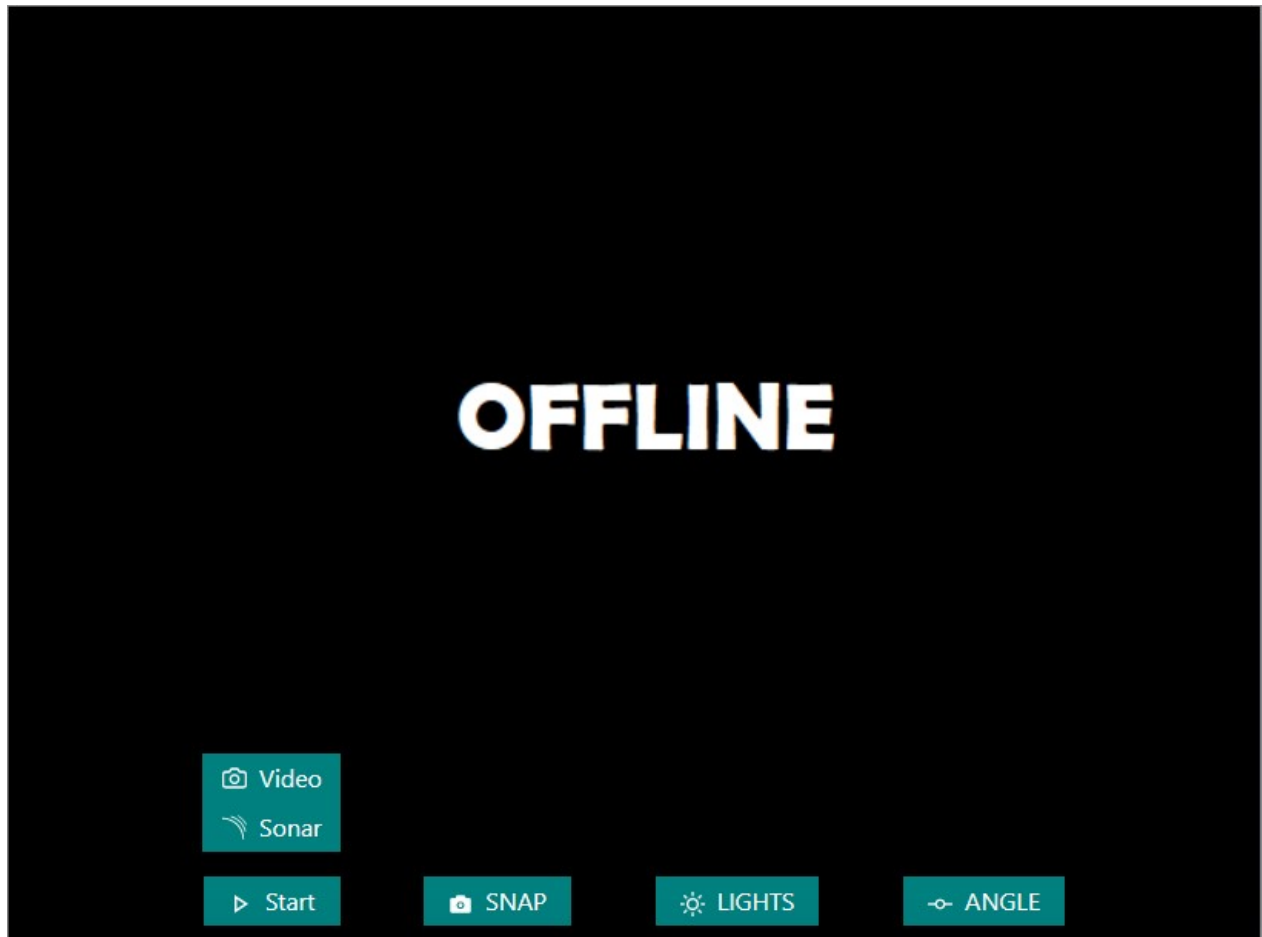
Figure 5.8: Dashboard video display

Controlling the ROV

At all times, the operator will have complete control of what commands he has sent to the ROV and their corresponding response status. Below in Figure 5.9, we can on the left-hand side see various commands sent to the ROV, and on the right-hand side see whether or not the ROV responded successfully. The operator can choose a name from a predefined set of controllable ROV options and enter a value that makes sense.

Figure 5.9: Real-time commands and responses during ROV operations

Creating a session

This paragraph entails the final workflow of the session functionality in the **Dashboard** page. Figure 5.10 below shows the communication across entities in the system and the internal methods called inside each entity.
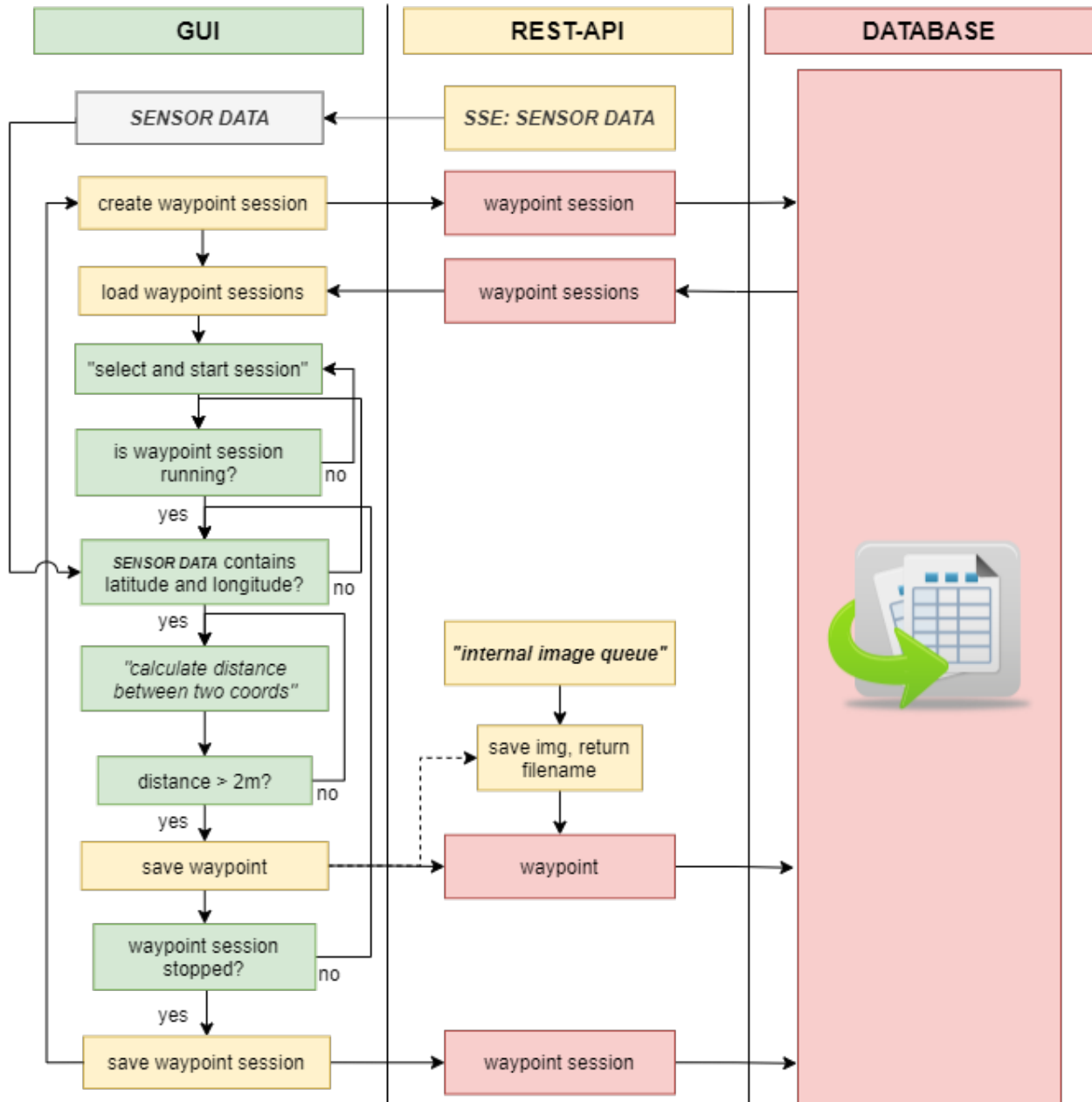
Figure 5.10: Session functionality

The operator starts by creating a name for his waypoint sessions. After that, he loads all cre-

ated missions from the database and selects one of his choices. The operator can then begin the waypoint session, where sensor data is automatically fed into the waypoint session. As incoming sensor data is updated, checks are made to validate whether it contains GPS coordinates or not.

The main algorithm for deciding the save intervals is the calculation of the distance between two coordinates. Inspired by the Haversine method, explained in detail in 2.11.1, the algorithm developed checks the current and previous GPS coordinates, compares and saves whenever the distance has exceeded its threshold (in this case 2-meters).

When the system reads a valid distance, it then automatically makes a waypoint POST requests to the REST-API. The REST-API queries the internal image queue in the application tier for an image and saves it an image bank. Upon saving, a filename reference is returned where the corresponding image is equal to the sensor data location. The filename reference is attached to the waypoint and saved in the database. The response of the POST request is the returned complete waypoint which is finally logged in the session interface for the operator to see, as can be seen below in Figure 5.11.

Figure 5.11: Ongoing waypoint sessions during ROV operations

5.1.2.4 Map

Figure 5.12 shows the mapping component of the desktop application, the **Map** page. In the **Map** page, the operator can load the results made after ROV operations and view the recorded data stored in the database.

The left-hand side of **Map** page consists of an interactive map that the operator can navigate. On the right-hand side, the operator can view and load completed waypoint sessions in the menu. The buttons to *DISPLAY*, *RESET* and *DELETE* can be used to control the data from the database.
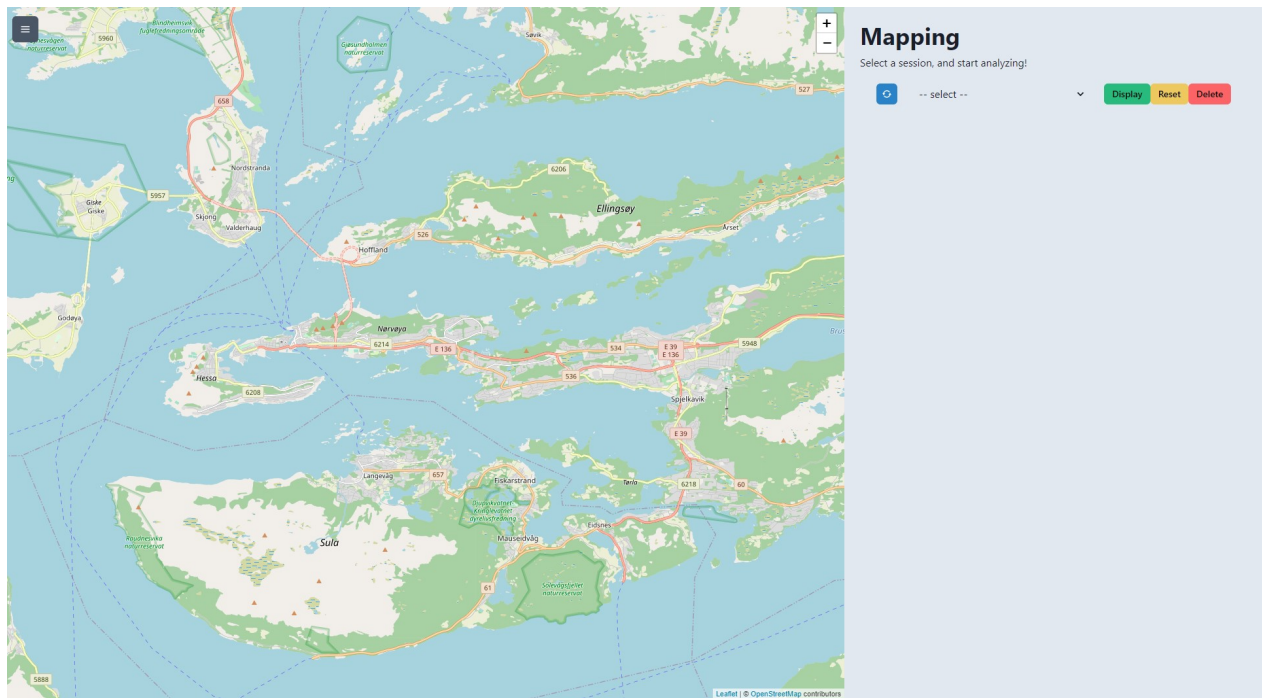
Figure 5.12: The **Map** page

Analyzing sea operations

Figure 5.13 below shows the final workflow of analyzing the recorded sensor data and images after ROV operations. As briefly stated above, the operator can click *REFRESH* and view any completed waypoint sessions by its "session_id" stored in the database. Upon selection (ex: "session_id" = "My Session 1337"), the operator can click *DISPLAY* to load all the stored waypoints which has a relationship 2.5.4 with the session_id selected. The loaded waypoints will automatically be drawn onto the map at their respective GPS coordinates. The operator can then freely click any waypoint, and get a summary of all the sensor data and image saved in that exact location, as can be seen on the left side of Figure 5.13.
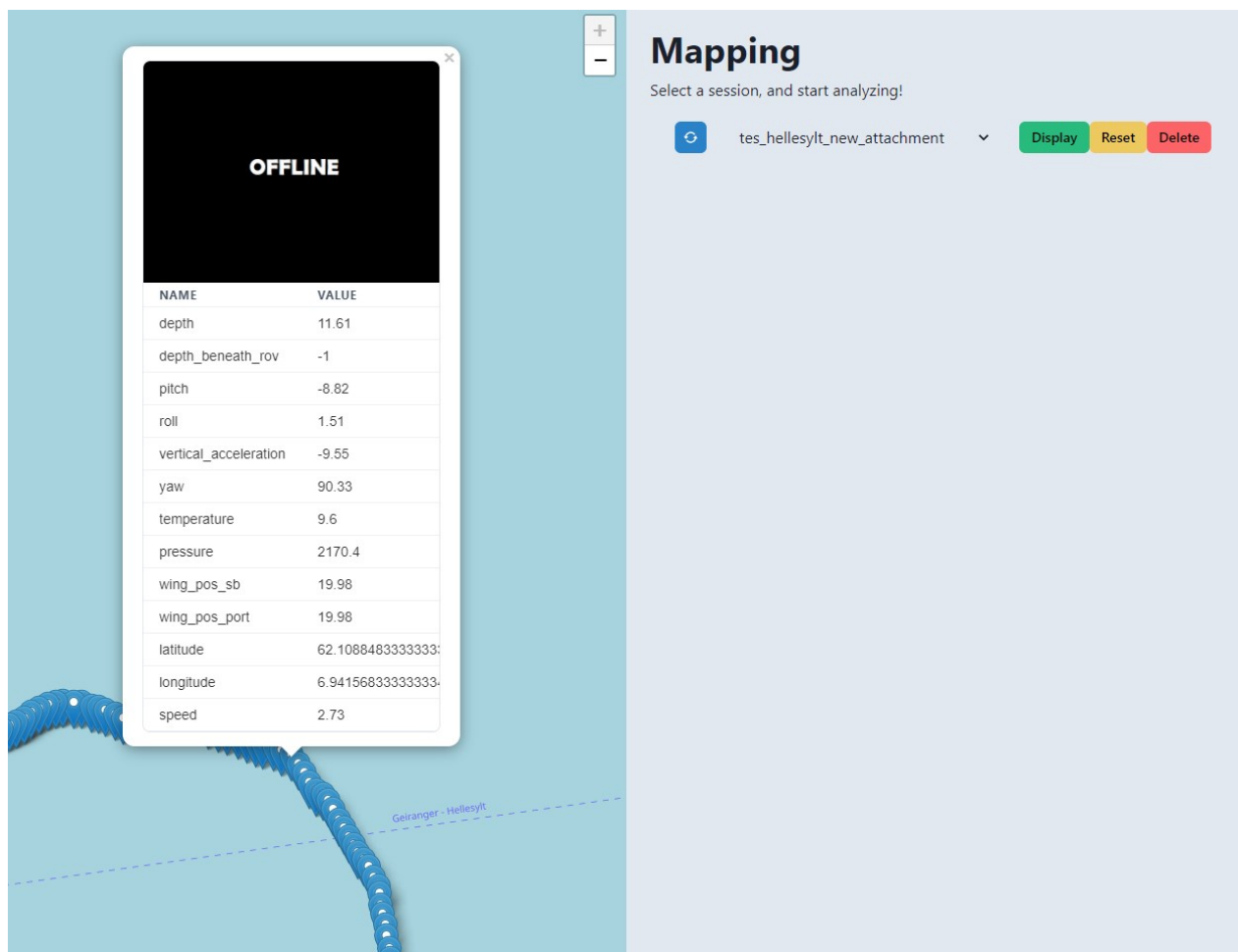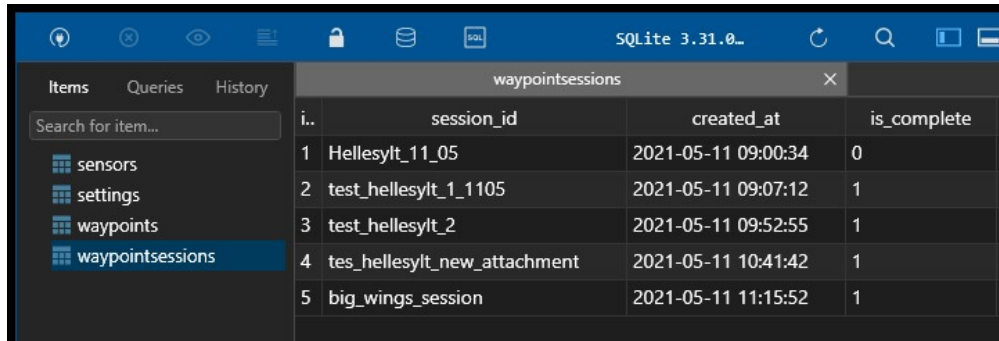
Figure 5.13: Analyzing a single waypoint

5.1.3 REST-API

The final endpoints accessible through the API is documented in Appendix [D]. In Figure 5.14 below, we can observe an excerpt from a real waypoint session using the technique referred to in 5.1.2.3.

1. We can initially observe a POST-request is made to create a waypoint session, followed by a GET-request to retrieve all available waypoint sessions.

2. Furthermore, we see a chain of POST-requests to save sensor data and imagery.

3. Lastly, we observer the completion of the waypoint session as it is updated with a PUT request.

Figure 5.14: Example of a full waypoint sessions.

5.1.4 Database

The final layout of the SQL database can be shown in the following images. The data is systematically connected using SQL relationships between the tables to create the best possible and scalable solution for easily expanding the database.

Figure 5.15: Using TablePlus 3.5.2.17, we can view a items in the **Waypointsessions** -table

Figure 5.16: Using TablePlus 3.5.2.17, we can view a few items in the **Waypoints** -table

Figure 5.17: Using TablePlus 3.5.2.17, we can view a few items in the **Sensors** -table

5.2   Software performance

Below in Tables 5.1 and 5.2, we can observe the speed performance of the loading operations during map analyzing. The tables show the time elapsed from when the call to the REST-API was made and the display speed inside the **Map** page.

		GUI	REST-API
Session ID	**Total waypoints**	**Time (ms)**	**Time (ms)**
test_hellesylt_1_1105	1076	2619.785000104457	35.924673080444336
test_hellesylt_2	768	1618.4950000606477	6.981372833251953
tes_hellesylt_new_attachment	565	1521.4599999599159	8.955717086791992
big_wings_session	63	158.7549999821931	2.0253658294677734

Table 5.1: Response time during analyzing: REST-API and GUI

Converted into averages,

	GUI	API
Session ID	**waypoint / ms**	**waypoints / ms**
test_hellesylt_1_1105	0.410720727	29.95155997
test_hellesylt_2	0.474514904	110.007017
tes_hellesylt_new_attachment	0.371353831	63.0881921
big_wings_session	0.396837895	31.1054917
–	–	–
avg: waypoints / milliseconds	0.4134	58.5380
avg: waypoints / second	413.35684	58538

Table 5.2: Average time during analyzing: REST-API and GUI

We can observe that the GUI can load on average 413 waypoints per second.

We can observe that the REST-API can fetch on average 58538 waypoints per second from the database.

In the Figure 5.18, 5.19, 5.20 and 5.21 below we observe the corresponding waypoint sessions with their *Session ID* as used above.
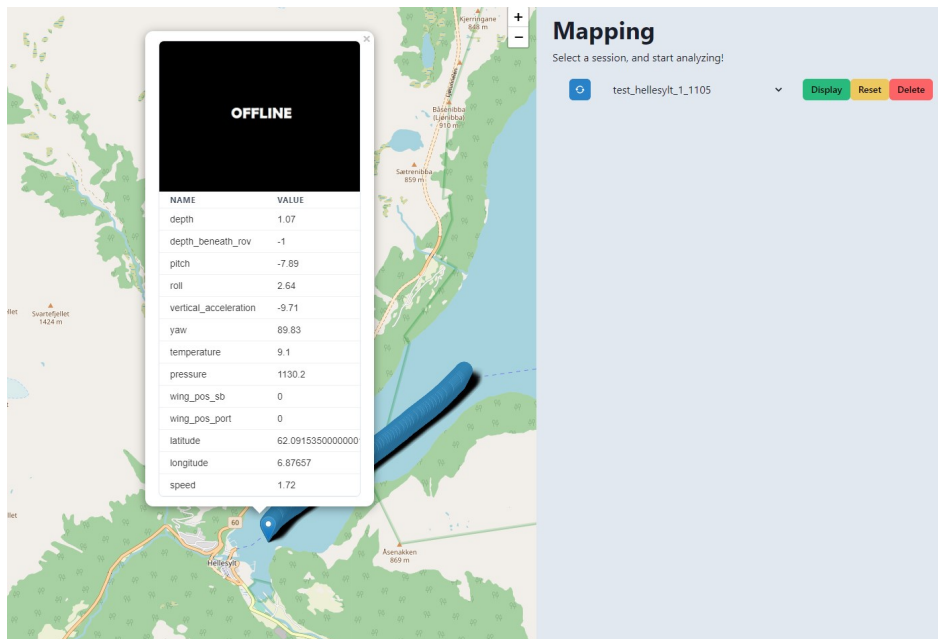
Figure 5.18: Session ID: test_hellesylt_1_1105
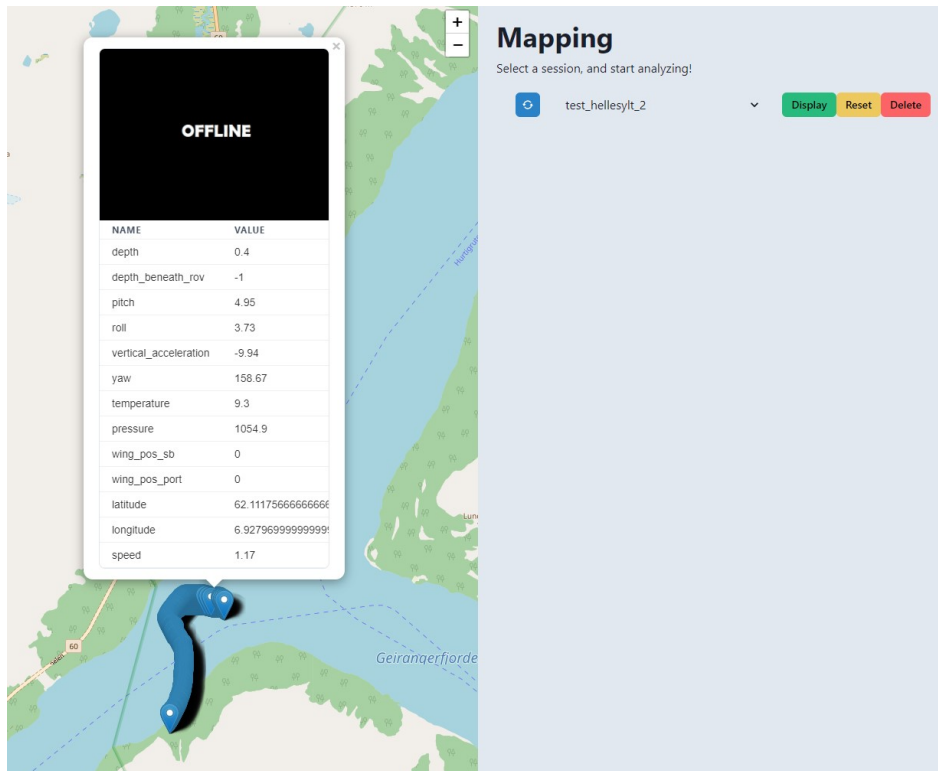
Figure 5.19: Session ID: test_hellesylt_2
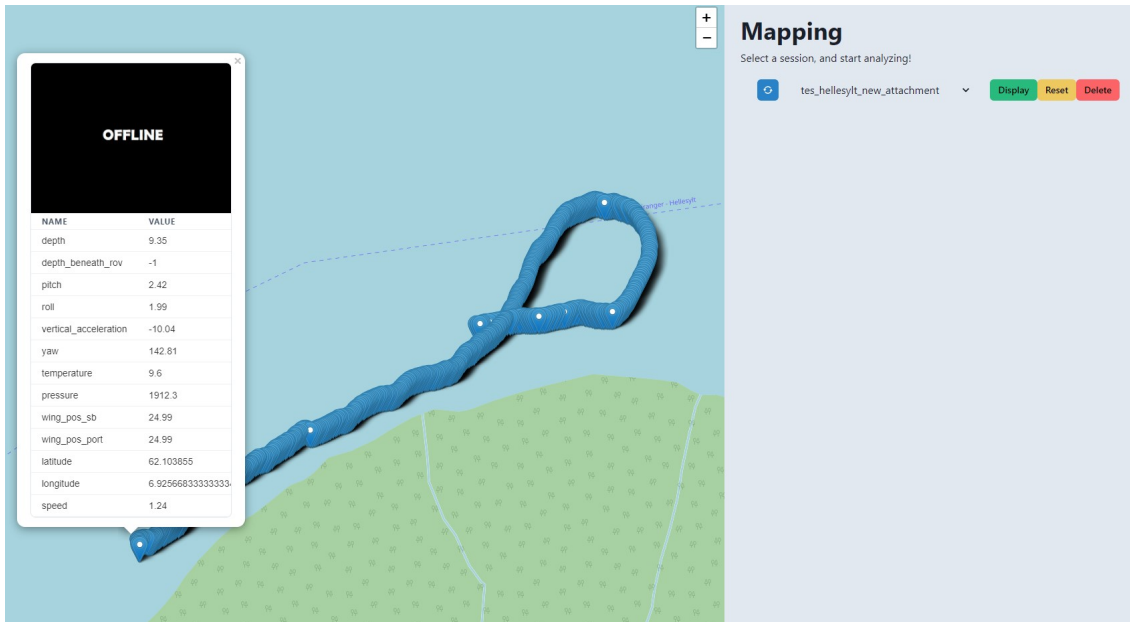
Figure 5.20: Session ID: tes_hellesylt_new_attachment
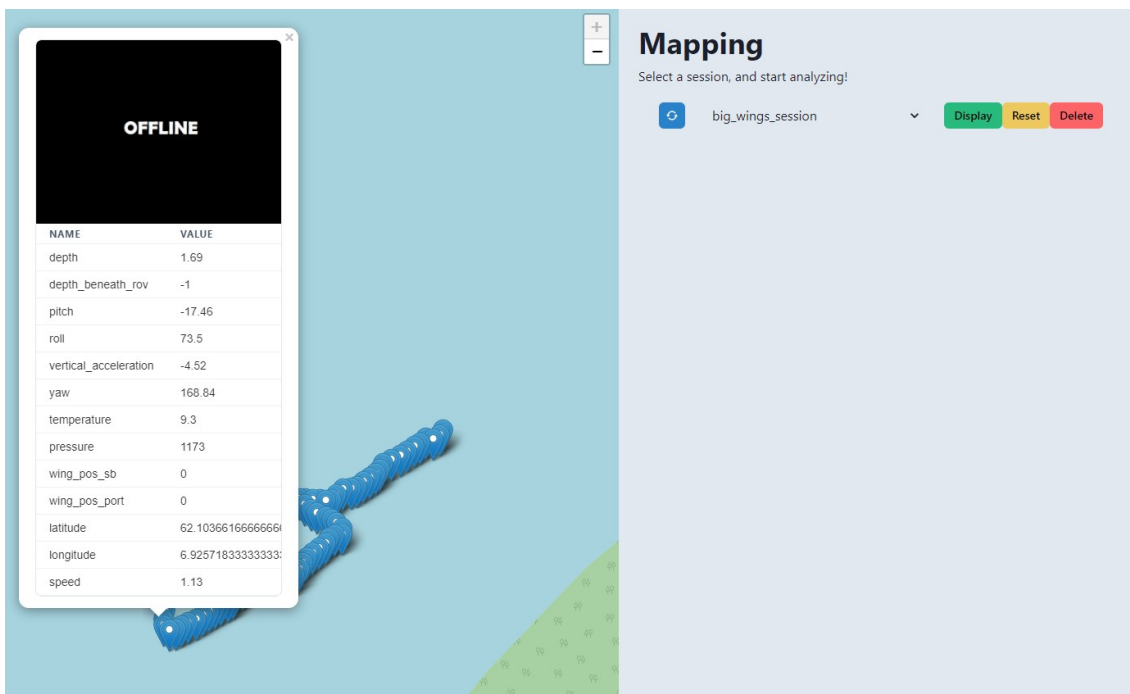
Figure 5.21: Session ID: big_wings_session

5.3 Communication results

5.3.1 Tether

As all data and communication go through a tether cable between the ROV to surface, it was necessary to test its capacity to develop software, considering Ethernet speed safely. The test shown in Figure 5.22 shows an average bandwidth of 75 Mbits/sec using the 90 meter cable.

```
PS C:\Users\andre\Downloads\iperf-3.1.3-win64> .\iperf3.exe -c 192.168.0.22
Connecting to host 192.168.0.22, port 5201
[  4] local 192.168.0.20 port 60056 connected to 192.168.0.22 port 5201
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.00   sec  8.62 MBytes  72.3 Mbits/sec
[  4]   1.00-2.02   sec  8.75 MBytes  72.3 Mbits/sec
[  4]   2.02-3.01   sec  8.75 MBytes  73.5 Mbits/sec
[  4]   3.01-4.02   sec  8.88 MBytes  74.2 Mbits/sec
[  4]   4.02-5.01   sec  9.00 MBytes  75.9 Mbits/sec
[  4]   5.01-6.01   sec  9.12 MBytes  76.9 Mbits/sec
[  4]   6.01-7.00   sec  8.88 MBytes  74.9 Mbits/sec
[  4]   7.00-8.01   sec  9.25 MBytes  77.2 Mbits/sec
[  4]   8.01-9.01   sec  8.88 MBytes  74.5 Mbits/sec
[  4]   9.01-10.00  sec  8.88 MBytes  74.5 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-10.00  sec  89.0 MBytes  74.6 Mbits/sec                  sender
[  4]   0.00-10.00  sec  89.0 MBytes  74.6 Mbits/sec                  receiver

iperf Done.
```

Figure 5.22: Test 1

5.4 Digital Twin

One of the goals for this project was a digital twin simulation that had the same results as the real-world systems. The digital twin can act as a simulation of the system's inputs and can be operated as if it was an ROV from the GUI without making changes to the software systems of the project. Figure 5.24 shows see that AGX transmits live data to the GUI, and in Figure 5.23 we can see the set-point being set in the GUI, travelling through the ROV RPi System and being set in the simulation, and then the new set point being returned to the GUI.
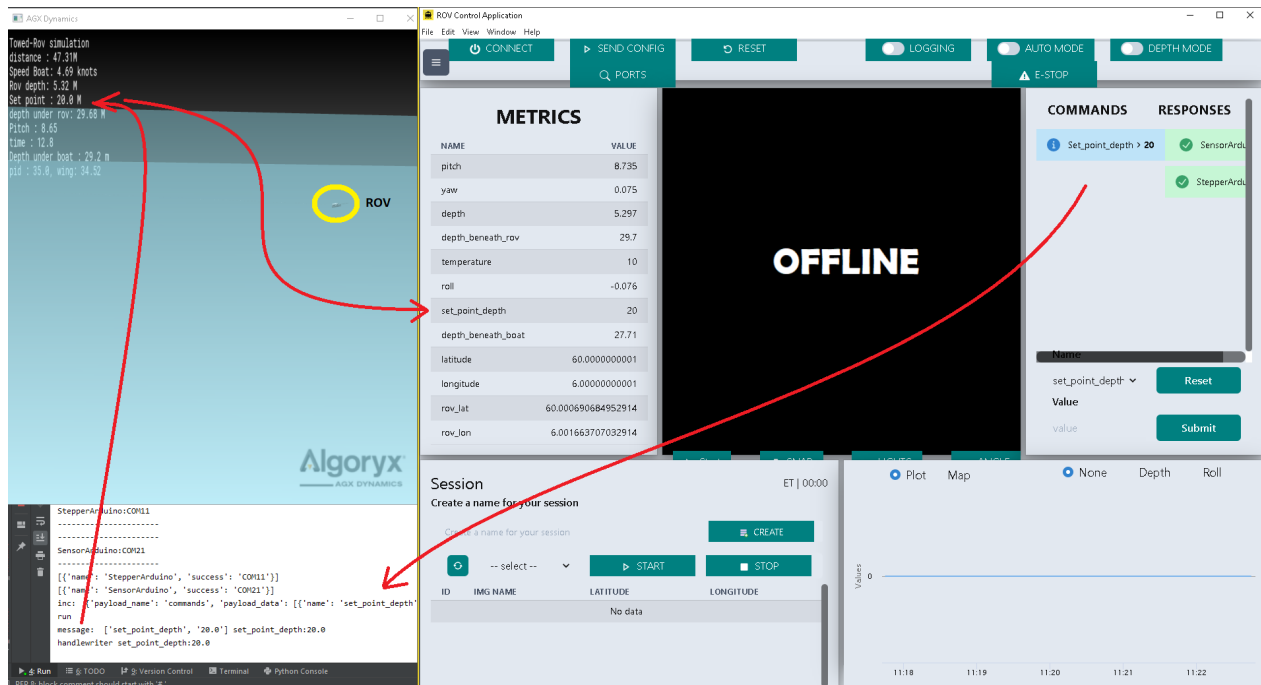
Figure 5.23: GUI commands the AGX Digital Twin
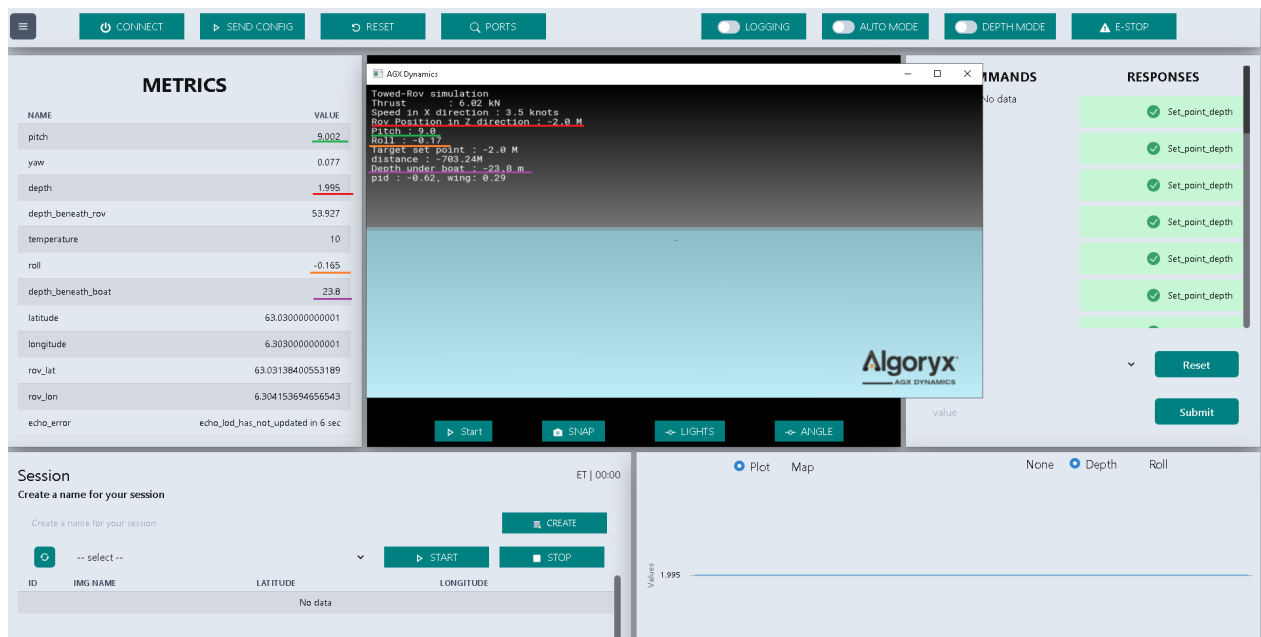
Figure 5.24: AGX and GUI communicates the same data

5.4.1 AGX and Hydrodynamics

5.4.1.1 ROV control

We can observe how pitch and positions change a lot smoother in the simulation see 5.26 when compared to the physical ROV seen in Figure 5.45. Figure 5.25 shows that the simulated ROV can reach a depth of 45 meters with a cable of 200 meters. However, as the depth increases, the vertical speed decreases, especially after 25 meters.
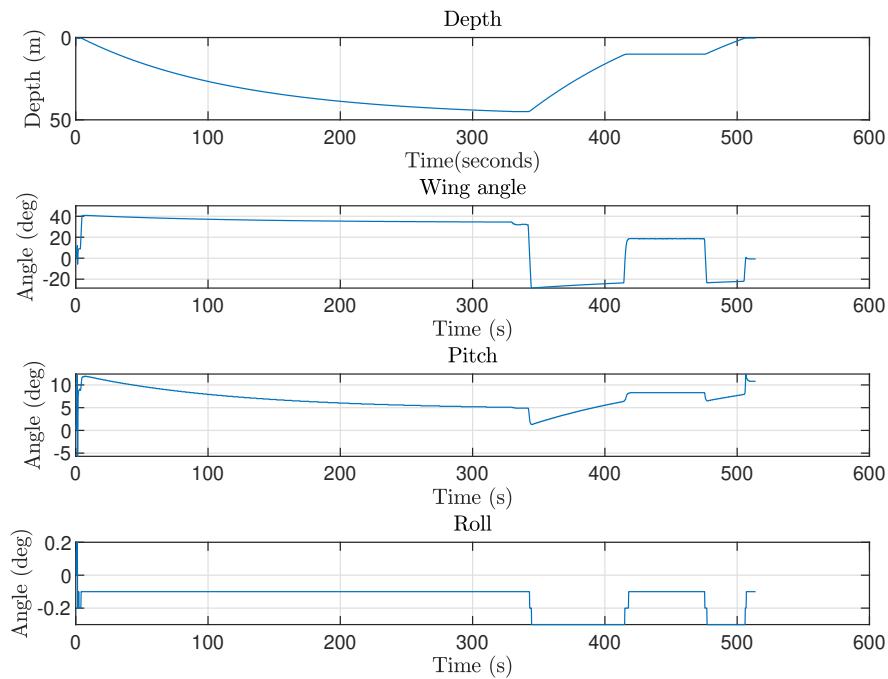
Figure 5.25: Pitch and wing angles from AGX with longer cable.
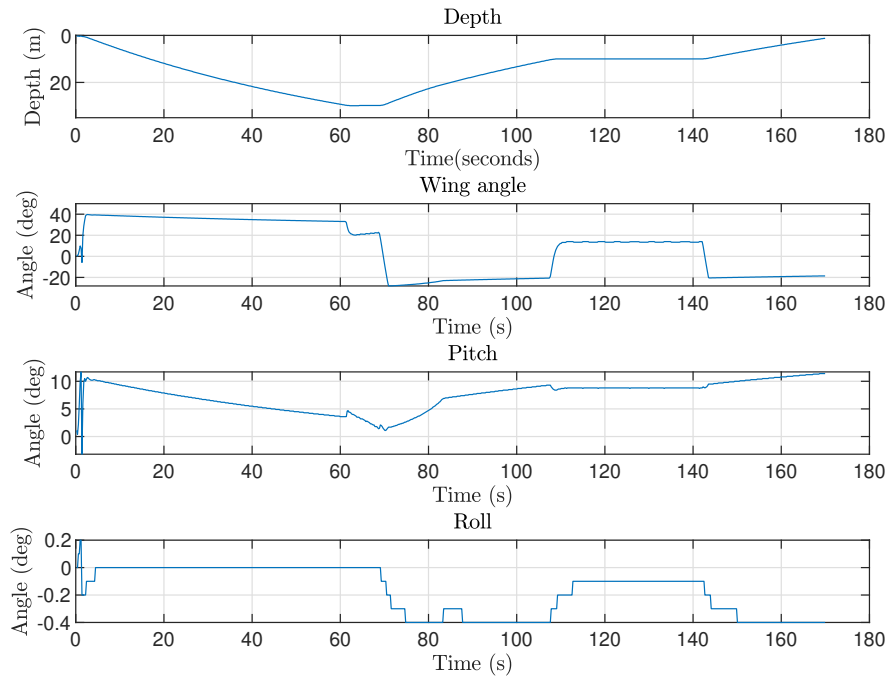
Figure 5.26: Pitch and wing angles from AGX, when tested with large wings

Using the simulation, it is easy to make changes to the software RPi in the Surface Unit, or RPi in the ROV or the REST-API, and then validate the success of the changes without testing the ROV in the sea. The system can also simulate the systems with different wings and shapes to see how this will affect the control of the ROV. For example, see differences in vertical speed between the Figure 5.25 with a 200-meter long cable and standard wings and the Figure 5.26 with 100-meter cable and larger wings. The Wings have a significant effect on the manoeuvrability of the ROV since even with the shorter cable; the wings reach a depth of 25 meters in 60 seconds, while the other simulation uses 100 seconds to reach the same depth.

5.4.2   Simulation speed

In Figure 5.27 we can see the plot of real time against inn simulation time. The simulation speed can increase or decrease depending on the simulation parameters, also increase if plotting or recording is on.
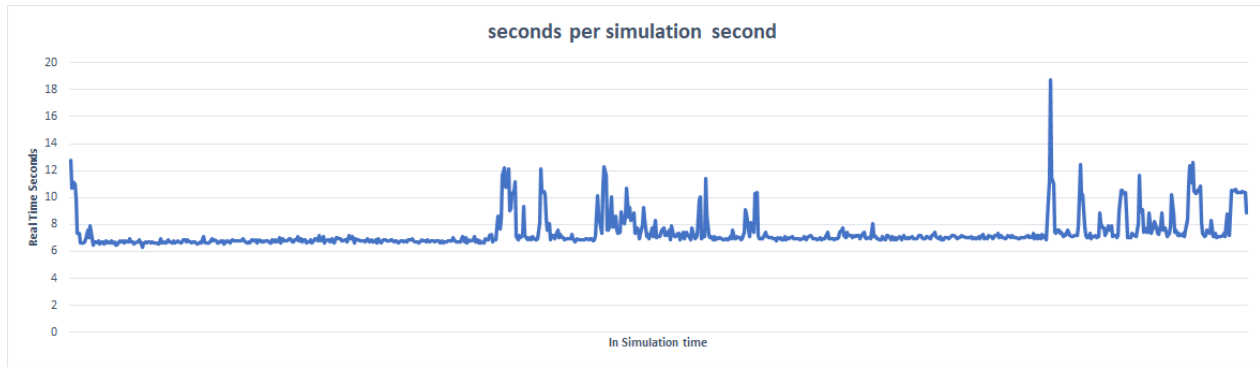
Figure 5.27: The simulation time in sec per sec for each step in a simulation run.

5.5 IMU sensor fusion

We can observe that the complementary filter is working as intended by keeping the long-term effects of the accelerometer and the gyro's short-term effects as the theory suggested 2.2. This led to measurement with less noise, with good response and no drift. Figure 5.28 compares different tuning values of the filter, where alpha as zero is the raw accelerometer, and alpha 1 is the raw gyro. The results indicate that an increased gyro slows down the response while giving a smoother measurement. Figure 5.29 compares alpha value of 0.85 with the raw accelerometer data.
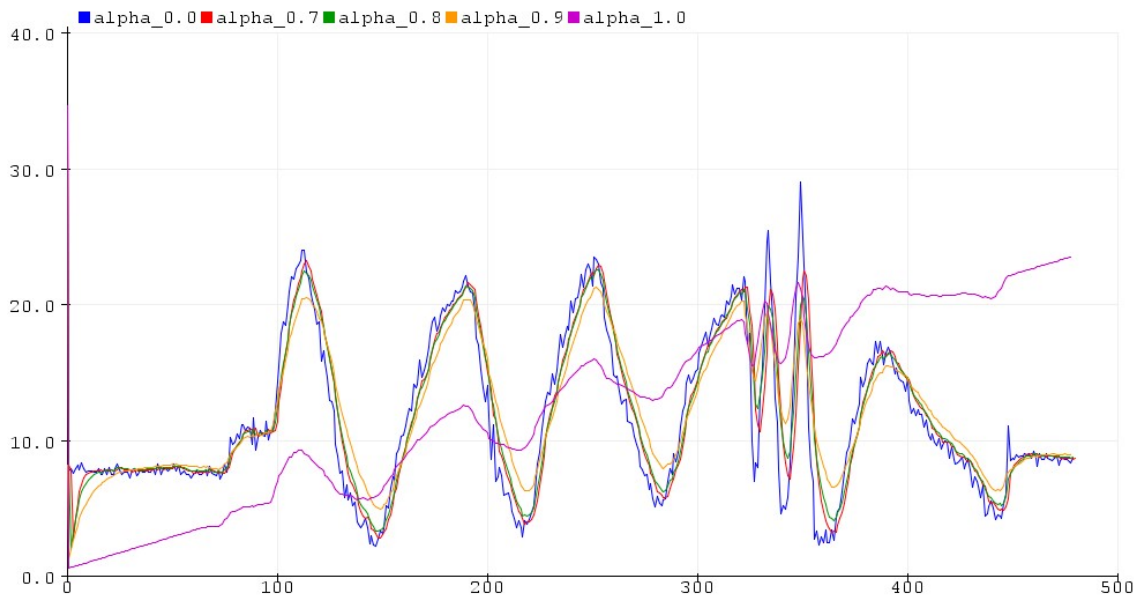
Figure 5.28: Comparing alpha values
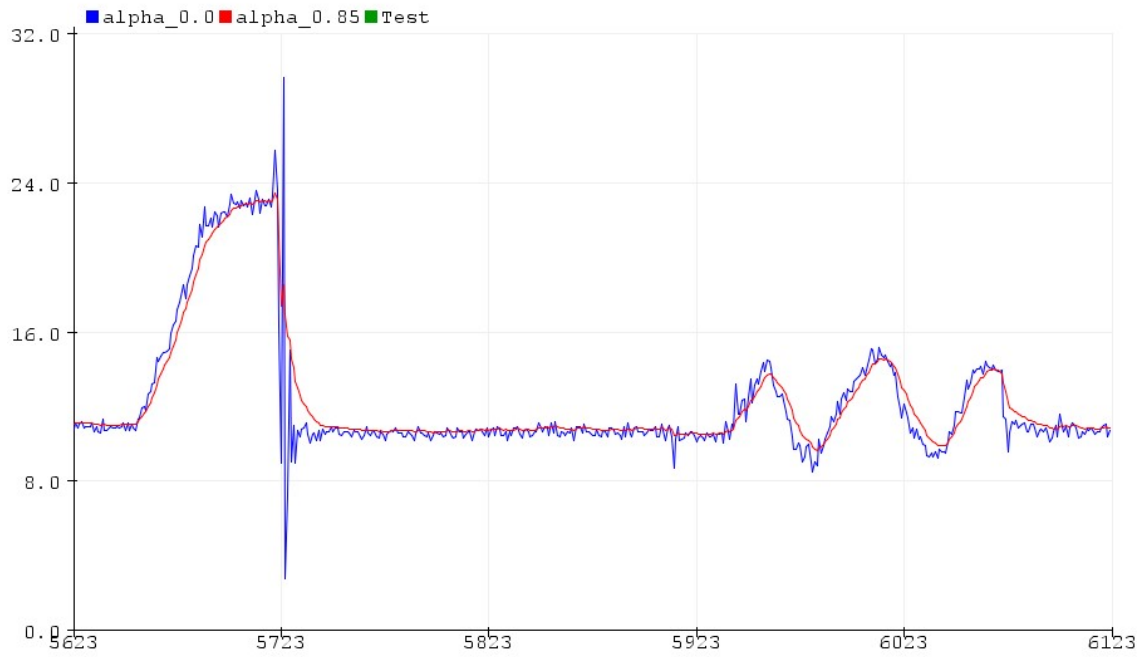
Figure 5.29: Accelerometer vs Complementary Filter

5.6 Seafloor Tracking

The seafloor tracking was only tested in the test script and the digital twin. The results displayed are the first run with the respective parameter values. The seafloor settings and parameters are listed in a table for every figure. The parameters were selected for displaying different scenarios. The seafloor is semi-randomly generated as described in 4.7.5. Notice that the estimations are based on the length of the cable and not the entire plot. For details regarding the implementation of seafloor tracker, please see Appendix [L].

Seafloor		Parameters	
Maximum depth	80	Desired distance	40
Minimum depth	30	Minimum distance	25
Length	6000	Distance to ignore	10
Samples/m	1	Length cable	300

Table 5.3: Seafloor tracking test 1
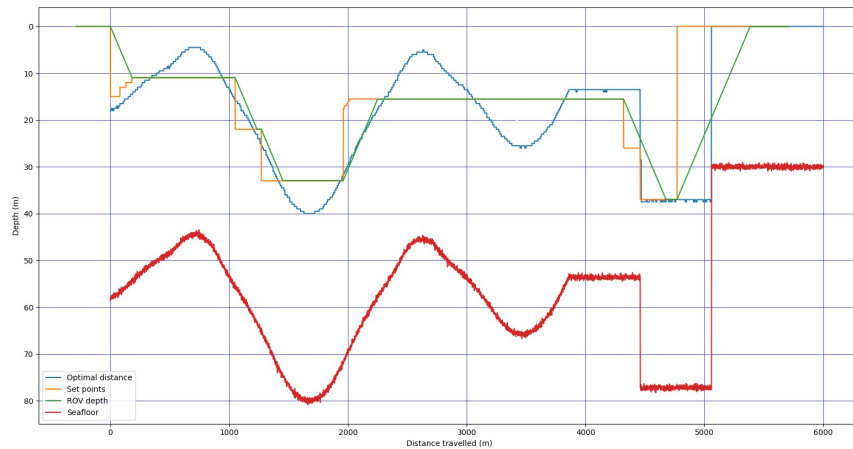
Figure 5.30: Seafloor tracking test 1

The parameters 5.3 were used in Figure 5.30. The ROV does not descend before 300 meters of data is collected. While the ROV is descending, the mean value rises with the seafloor. This displays the adaption of ROV data when deciding on the next set point. The ROV keeps the same distance until approximately 1100 meters before changing the set point. At 1100 meters, the distance between the current set point and the mean value becomes greater than the distance to ignore the parameter. At 2000 meters, the system notices that the distance between the current

set point and the lowest set point is greater than the minimum distance, meaning the system will ascend as soon as possible. It is making sure that there will not be a collision. The last part of the seafloor at 5000 meters is to simulate a sudden cliff. A new set point is chosen as soon as the cliff is detected. There is no calculation for estimating if the ROV is fast enough to avoid detection, as there were not enough data for the behaviour of the real system.

Seafloor		Parameters	
Maximum depth	80	Desired distance	20
Minimum depth	30	Minimum distance	15
Length	3000	Distance to ignore	5
Samples/m	1	Length cable	100

Table 5.4: Seafloor tracking test 2
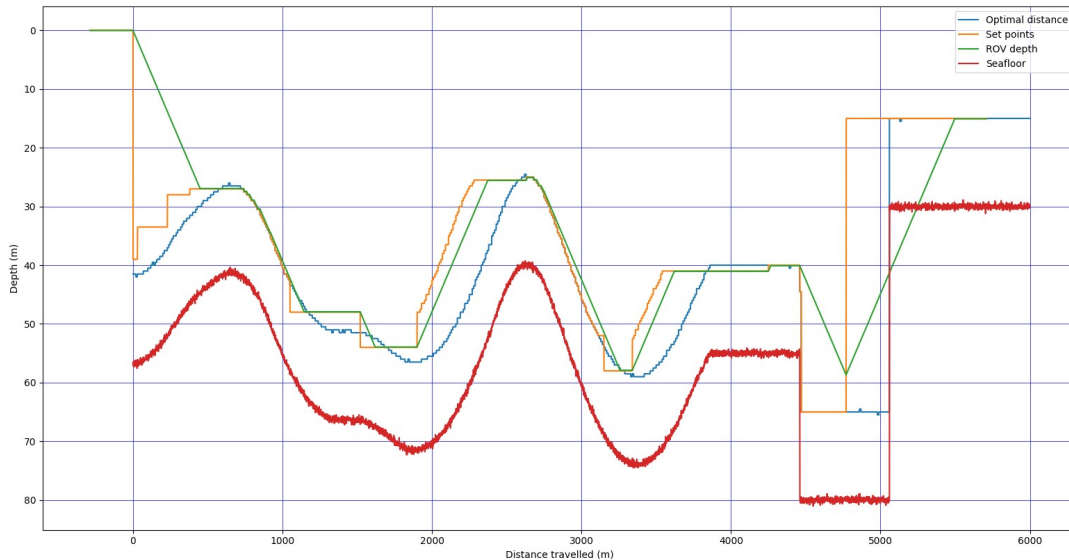
Figure 5.31: Seafloor tracking test 2

The purpose of the next test 5.31 were to see how the algorithm handled a more aggressive parameter setting 5.4. The response is more active, and the ROV follows the seafloor tighter than in the previous test. The alarm for too steep incline was not added, as it should be designed based on the performance of an ROV able to work in a range big enough for implementing seafloor tracking.

Seafloor		Parameters	
Maximum depth	40	Desired distance	20
Minimum depth	30	Minimum distance	12
Length	2000	Distance to ignore	8
Samples/m	1	Length cable	300

Table 5.5: Seafloor tracking test 3
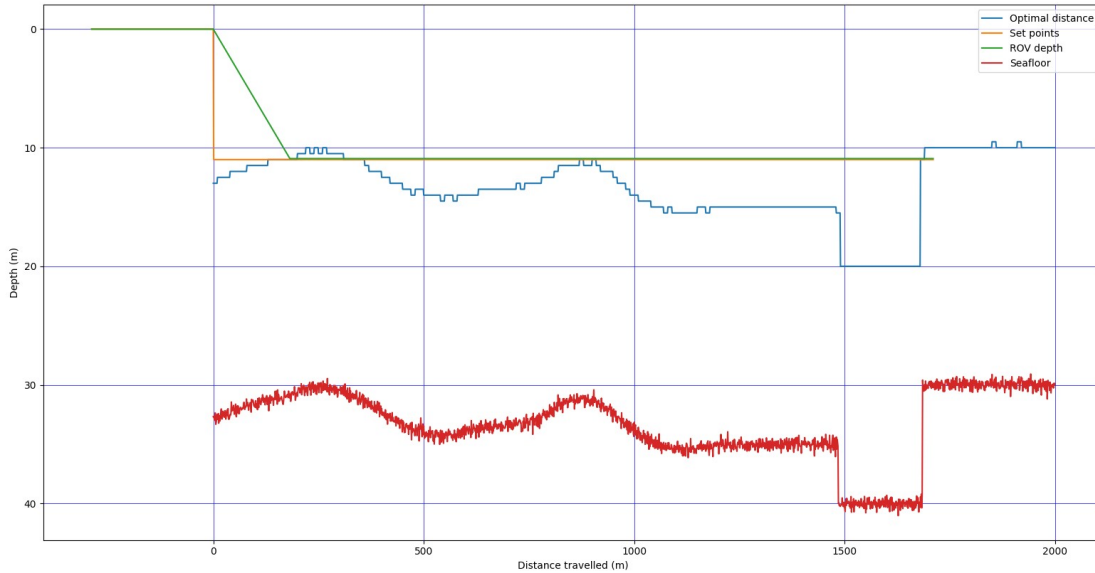
Figure 5.32: Seafloor tracking test 3

The purpose of the test was to see the response 5.31 to a less aggressive slope with a parameter setting 5.5 that could fit the seafloor. The set point is kept at the same value, although the seafloor is 40 meters at 1500 meters as the sonars have already picked up the steep incline.

Seafloor		Parameters	
Maximum depth	90	Desired distance	40
Minimum depth	10	Minimum distance	20
Length	6000	Distance to ignore	12
Samples/m	1	Length cable	100

Table 5.6: Seafloor tracking test 4
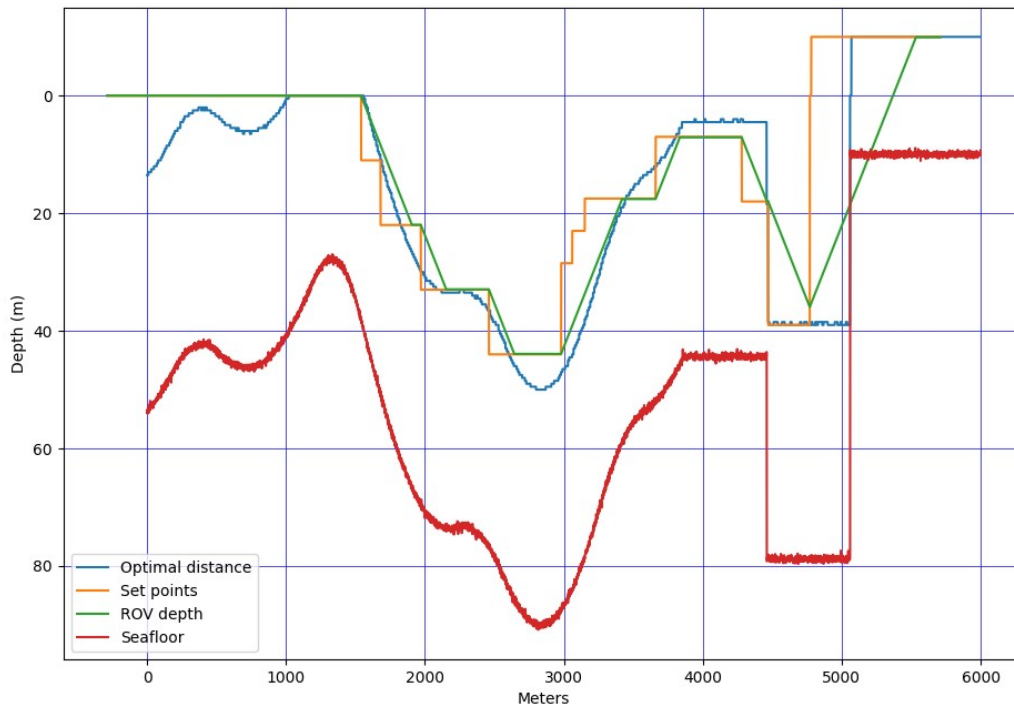
Figure 5.33: Seafloor tracking test 4

The last test, Table 5.6 and Figure 5.33, was to confirm that the alarm went off when the cost function could not find a legal depth. This is illustrated by setting every new set point to a negative depth. In the real system, an alarm should warn the driver to stop the boat.

5.7 Surface Unit Software

The final software in the surface unit consists of serial NMEA readers 4.5.1, a ZMQ subscriber and publisher 2.8, position estimation 4.5.3 and a distance calculator 4.5.3. It parses NMEA data, calculates the position of the ROV in GPS coordinates and sends the sensor data to the ROV for the seafloor tracking algorithm.

5.7.1 NMEA Parsing

The parsing of the NMEA data works as intended, with an extensive database of different NEMA sentences stored. As the data is parsed and sent to the system. After the parsing is done, the data is sent to the storage box and handled as standard data.

5.8 Electronics

The number of intended completed sea trials has been drastically reduced due to unforeseen problem with electronics, mostly due to 12 volts DC supplies breaking down. The last 12 volt DC supply broke down and stopped last sea trial, with a measured resistance measuring between the 12 volts positive, and GND showed a zero ohm resistance. Other 12 volt supplies have not broken down in the same manner, showing no shortage. Figure 5.34 shows a 12 volt supply with dirt on the components. In addition to the battery, the stepper drivers and several Raspberry Pi's have broken down.
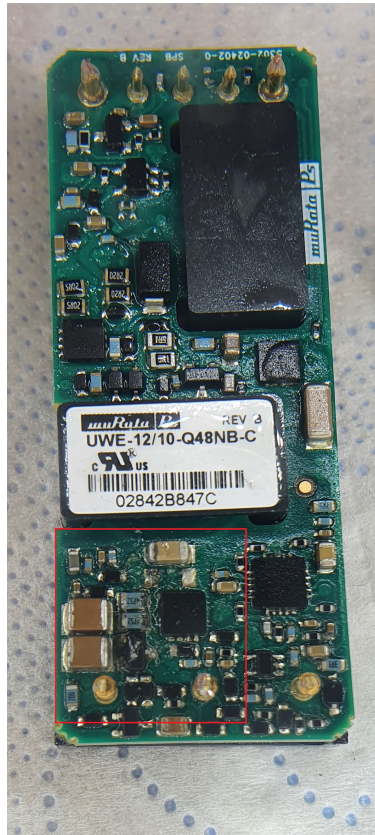
Figure 5.34: 12 volts DC supply with dirt on components

5.8.1   5 V DC-DC converter

Due to the problem with part breaking down, a new PCB card was made with all new parts.
While testing the new PCB, the green status light of the 5 volt supply was blinking—both with
and without load. Voltage was measured with a voltmeter and found a voltage varying between
4.75 - 4.98 volts without load. Adding an Arduino UNO as a load, the voltage was down to 3.80 -
4.21 volts. To get a more accurate measurement reading, this was hooked up to an oscilloscope
as explained in section 4.8.3.1. As shown in Figure 5.35 the yellow reading is the TEN 25-2411WI
5 V DC supply, and the blue is a 3-40 V to 1.25-35 V DC supply using the lm2596 integrated
circuit. These readings are from now on referred to as CH1 and CH2, respectively. Both CH1
and CH2 had the same input voltage of 12 volts. CH1 shows a minimum peak to maximum peak
voltage at 1.24 volts, while CH2 has a peak-peak voltage at 0.2 volts.
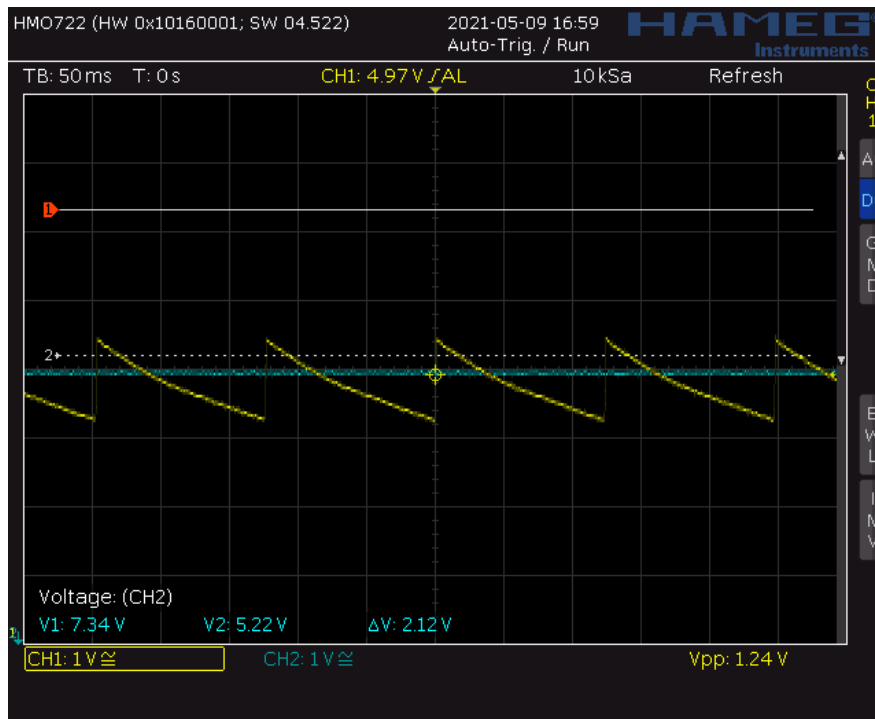
Figure 5.35: 5V supply without load

Adding an Arduino as a load on both 5-volt supplies in Figure 5.36 shows that CH2 still has a stable 5 volt while CH1's voltage had dropped significantly with a peak-peak of 2.64 volts. Tests show that this voltage drop was too significant for the Arduino to run.
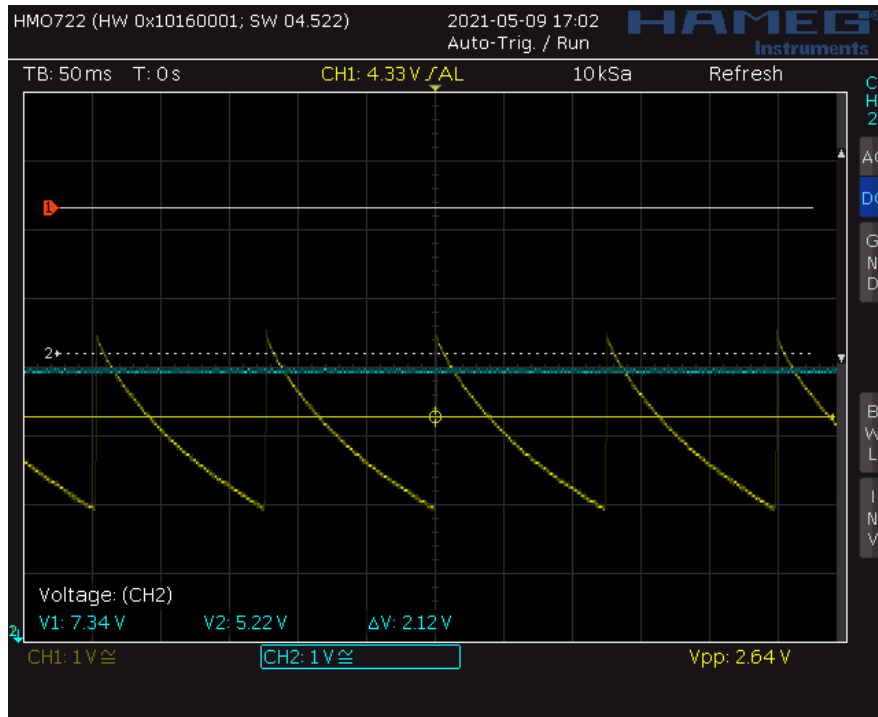
Figure 5.36: 5V supply with load

The old TEN 25-2411WI from the old PCB board was also tested as shown in Figure 5.37. Here the 5 V is stable and within limits for use. Checking the DC-converter documentation needs a minimum load of 10 % of max load, in other words, 50 mA.

The LED from the old PCB board draws 90 mA, and therefore this problem does not occur on the old one. The earlier report does not highlight this problem or give any information about the reasonably high draw of an LED.
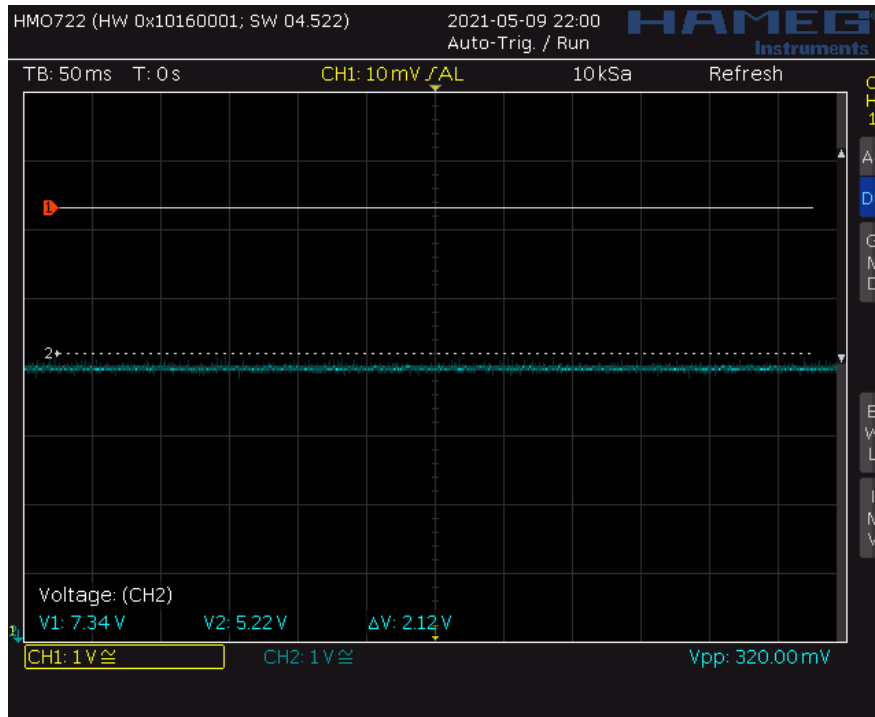
Figure 5.37: Old 5 volt supply

5.9 Camera and lights

The camera has been tested on two different occasions, on at Runde and one at the Voldsdalsvågen Marina.

The left picture in Figure 5.38 is taken at night at nearly full brightness (5250 lumens). Note that the focus on the camera is set for longer distance (10 meters, which is the distance to the seafloor at this point) than the seaweed observed in the image. The seaweed is at 0.5-1 meters of distance from the camera.

The right picture in Figure 5.38 in the same conditions, but the object observed at a larger distance apart from the camera, fading out at around 3 meters distance.
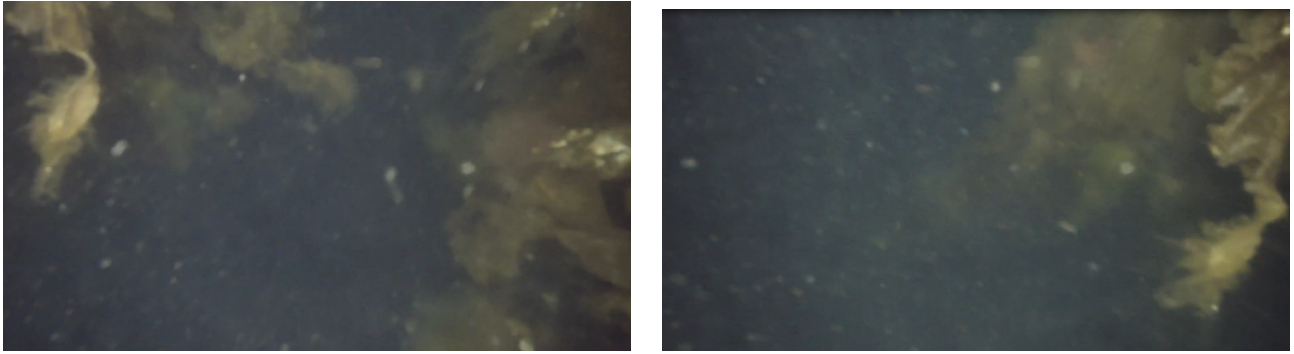
Figure 5.38: Left: Camera test 1 Right: Camera test 2

The picture in Figure 5.39 is taken at Runde in daylight, with 1500 lumen from the lights. Distance to the seafloor is 11 meters, and the focus is not adjusted.

Figure 5.39: Camera test at Runde

5.10 Side-Scan Sonar

Our most successfully test with the side-scan sonar is taken at Valderøya; these are presented in this section.

In Figure 5.40, 3 square objects are shown, the same image live from the GUI is displayed in Figure 5.41. The depth of this image is around 10-15 meters, and the sonar operates with a range of 50 meters. The sonar has a resolution of 2000 pixel on the x-axis. Note that after the breakwater, a turn was made, and significant waves at the test. The disturbance from the waves is shown as shivering in the sonar image.
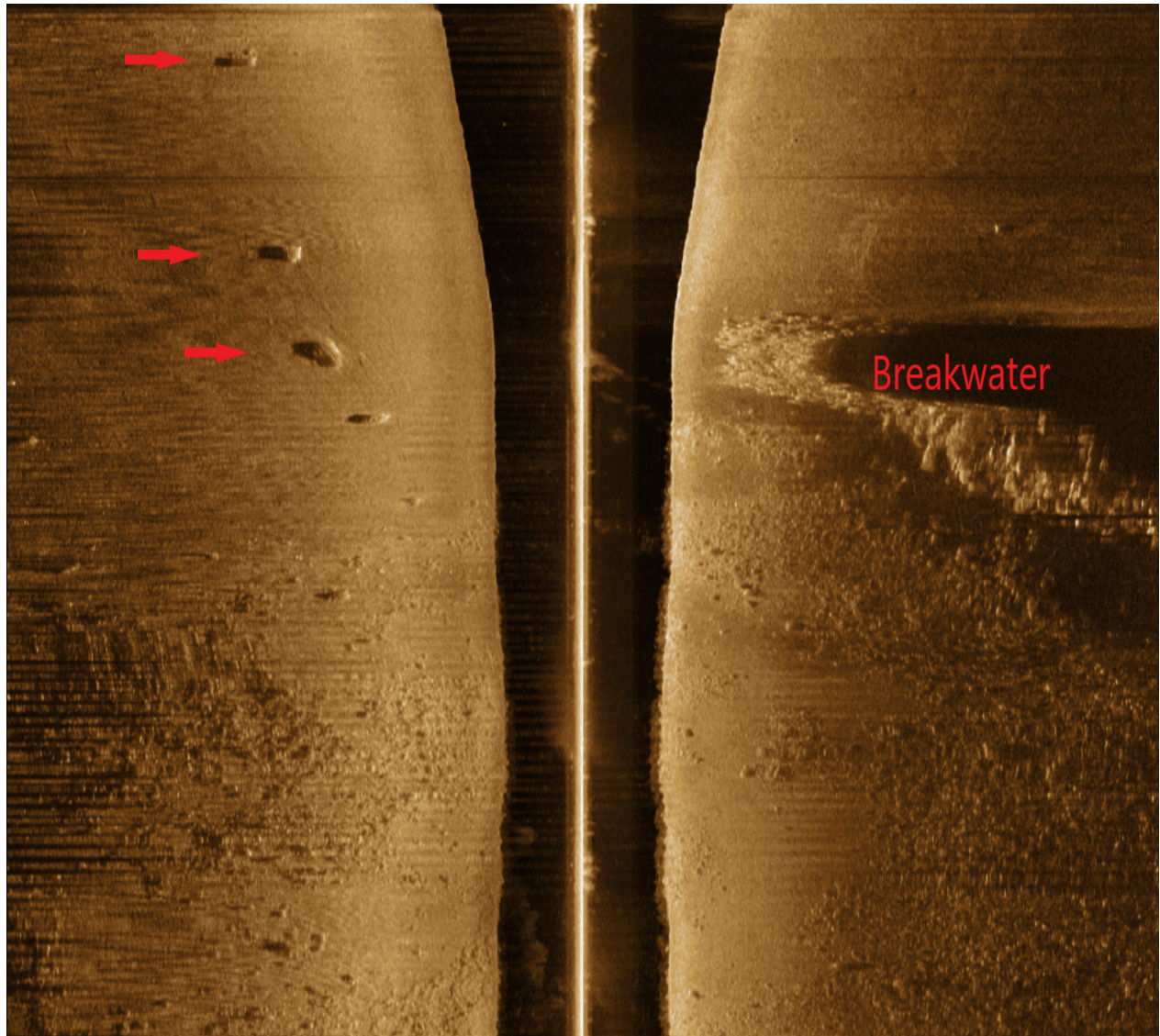
Figure 5.40: Sonar image taken at Valderøya displayed using DeepView FV 3.5.2.19
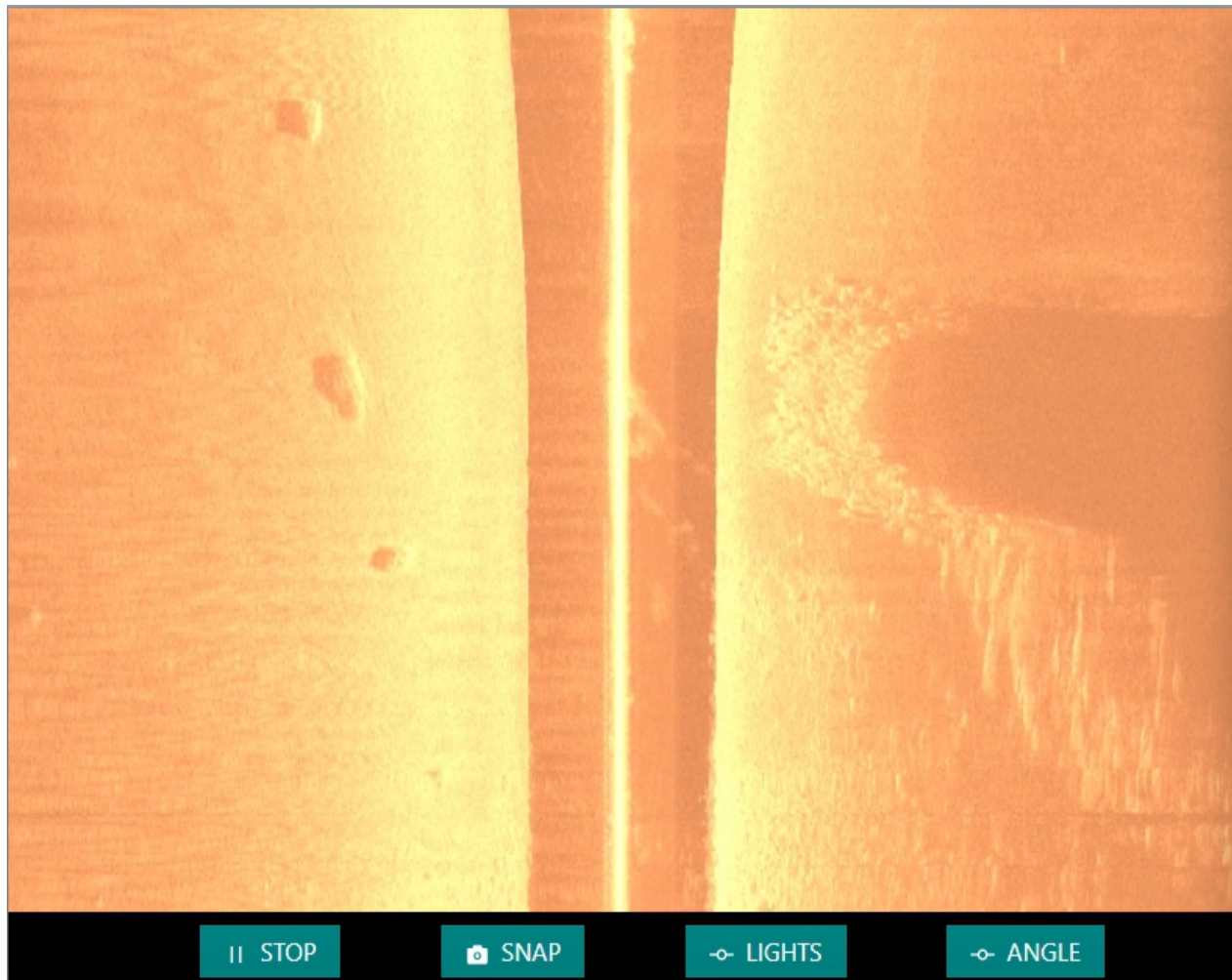
Figure 5.41: Sonar image taken at Valderøya displayed using the GUI

In Figure 5.42 the height of the object is estimated to 0.8 meters. The width of the object was found to be 1.8 meters

Shadow End Height:0.8 m
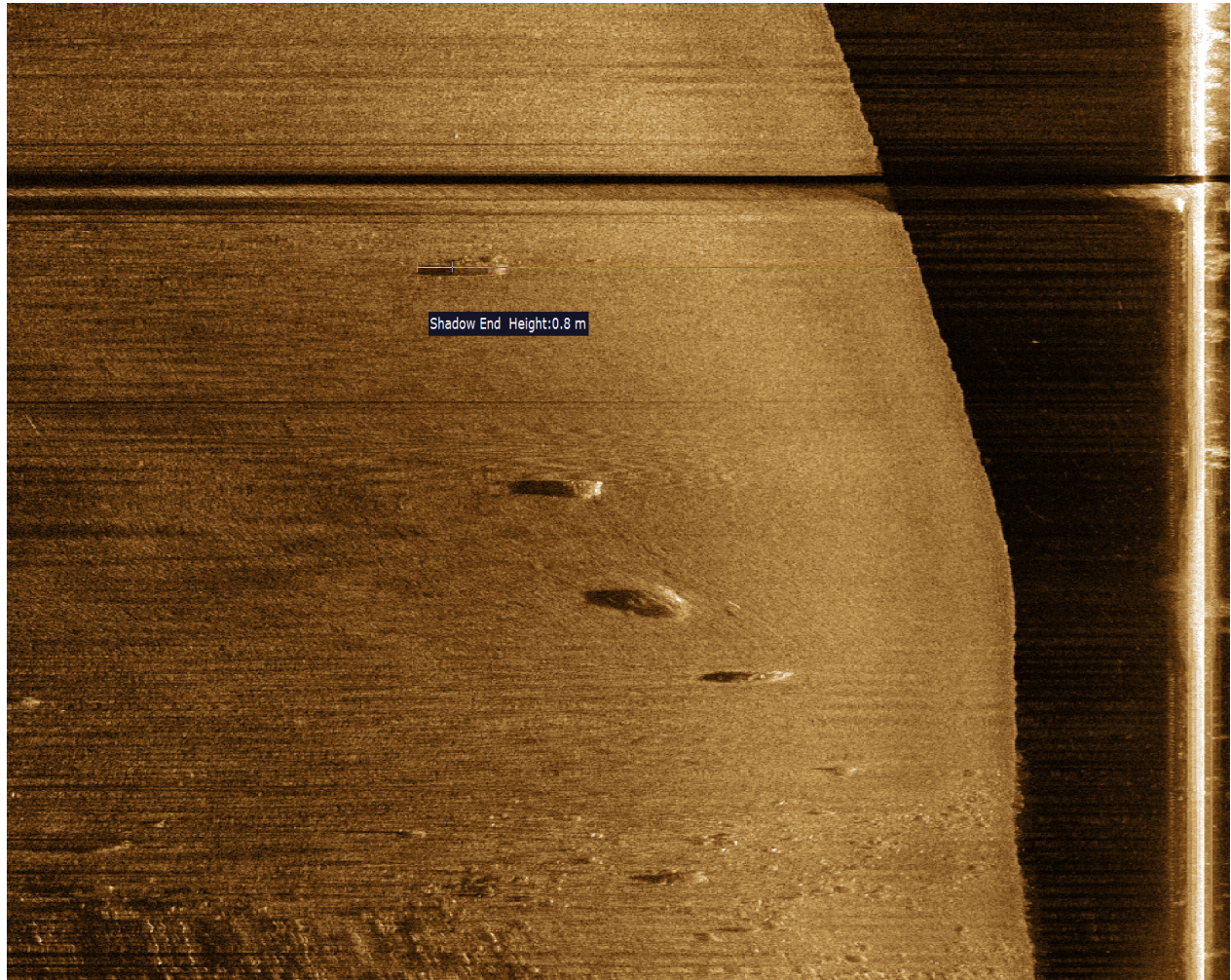
Figure 5.42: Sonar image taken at Valderøya

Figure 5.43 shows a far larger object. The sonar image is captured with 1000 pixels in the X-axis and a range of 50 meters while the depth is 45 meters.
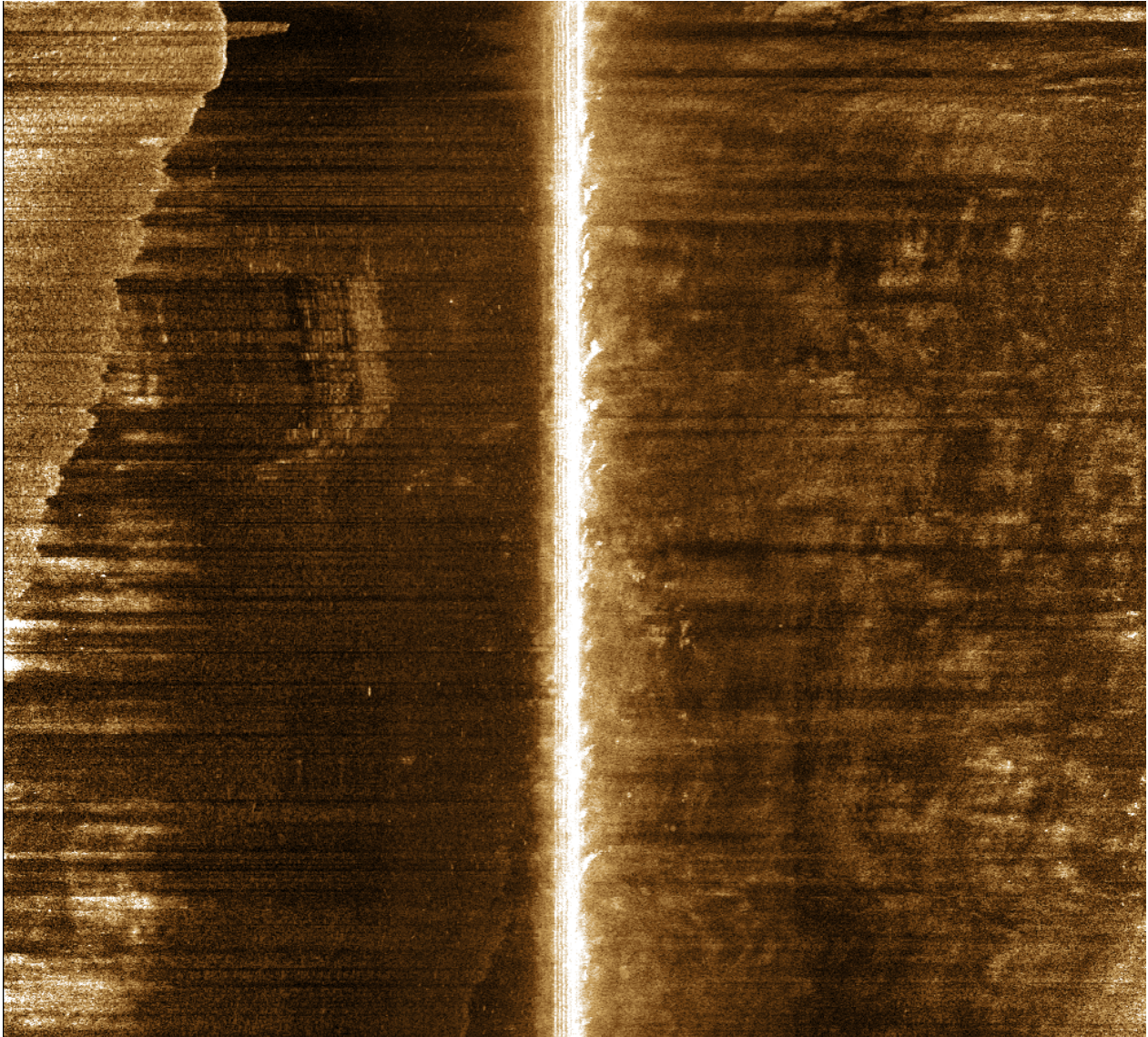
Figure 5.43: Sonar image taken at Hellesylt

5.11 Sea Trials

To improve the ROV prototype and address eventual weaknesses, a set of sea trials were conducted. The sea trials are numbered with two digits(a.b), where a represents each day, while b represents each test conducted that day.

5.11.1   Data presented

In the first sea trial, a timestamp was not yet added to the GUI. The time given in the plot is estimated by comparing the time the wings used to move 20 degrees. The data collected from the first five sea trials contains an error for every 100-150 sample with a value of 1. The reason and solution to the problem are mentioned in 4.3.3.1. These values are filtered out before the data were plotted to make the data more readable. Another thing to be noted is that changes in the sea current and waves can interfere with the ROVs behaviour and reduce the repeatability.

5.11.2   Sea trial: 1

The purpose of the tests was to see the effect of a spoiler and the change in behaviour with a reduced pitch. The wing angle is compensated in terms of pitch, and the angle presented is relative to the angle of attack.

Sea trial 1.1

By comparing Figure 5.44 with Figure 5.45 the pitch is reduced by 10-20 degrees. The depth is also clearly reduced, as the ROV settles at 5 meters. The Roll is considerably higher, and the ROV kept rolling around after a short period.
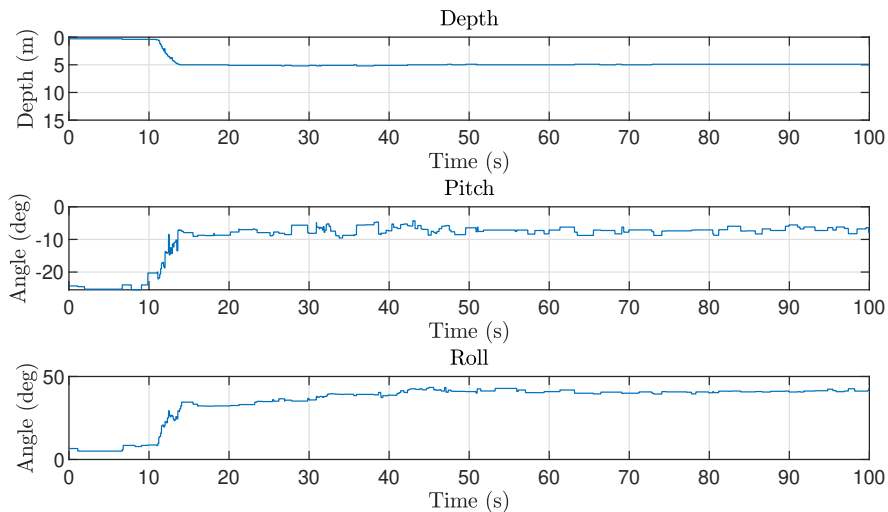
Figure 5.44: Sea trial 1.1

Sea trial 1.2

The side plates were removed to see if the ROV became more stable in the next test run 5.45; the spoiler was also removed since it was attached to the side plates. The roll is almost reduced to zero degrees, and the ROV did not roll around. On the other hand, when the wings are rotated, the roll becomes more unstable. Another observation is that pitch angle is increased when the spoiler is removed and seems to vary with either depth, wing angle or a combination of both. The operating range is very limited as the minimum, and the maximum depth is 14 and 16.5 meters.
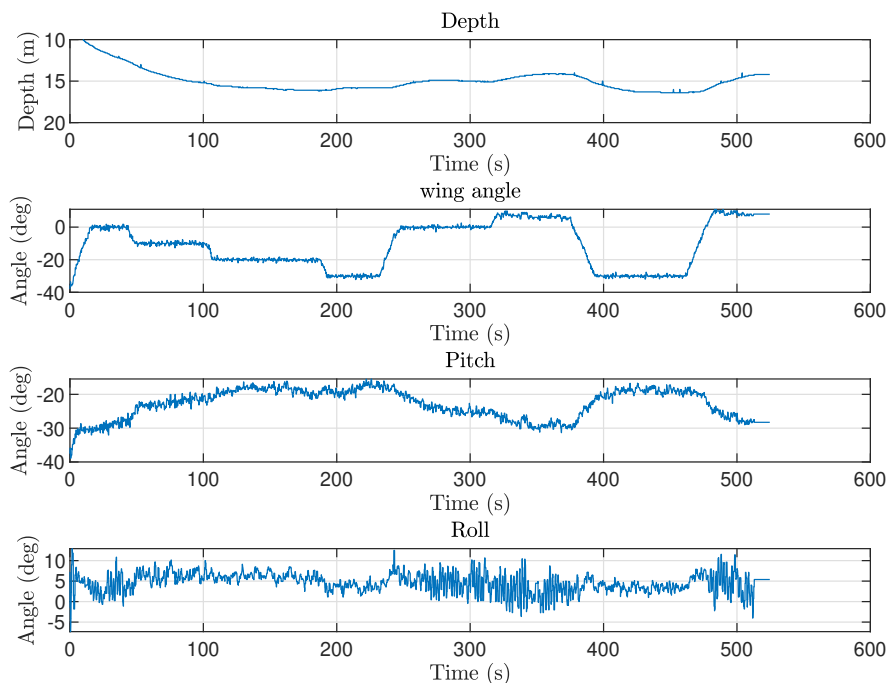
Figure 5.45: Sea trial 1.2

Hypothesis after test 1

1. The ROV rolled around due to currents attacking the ROV from the side. This force is relatively small, and the main reason is the placement of the center of mass and center of buoyancy in combination with a significant positive buoyancy of 120N.

2. The wings contributes more with an increased angle, leading us to believe that the stall effect 2.12.4.1 does not occur at 15 degrees.

5.11.3 Sea trial 2

The purpose of the second test was to see if the ROV became more stable by doing some modifications to increase the stability. This was done with an improvised solution where the ROV was filled with oil. The added weight was compensated with extruded polystyrene insulation and a 100mm pipe from a previous project. A short test was conducted to verify that everything was working before mounting the wings. After the test, it was discovered that the ROV was descending. The ROV stopped at 68 meters before we pulled it back up. There were no signs of water intrusion in the ROV.

5.11.4 Sea trial 3

To keep the ROV floating there were done some physical changes 4.8.1.1. By adding some weight on top of the ROV, the new positive buoyancy was approximated to be close to 20N. The side plates were also mounted but without the spoiler.

By inspecting Figure 5.46 it is clear that the pipes have increased the oscillation in pitch. At the same time, the pitch is reduced when comparing with the first sea trial without spoiler. However, there is more disturbance as the pitch is oscillating with an amplitude up to 10 degrees. The unstable pitch is most likely making the roll less stable as well, as there are more disturbances due to more turbulence (2.12.1). The depth is almost the same as experienced in sea trial 1.2 (5.45), indicating that the change in pitch and buoyancy cancelled each other.
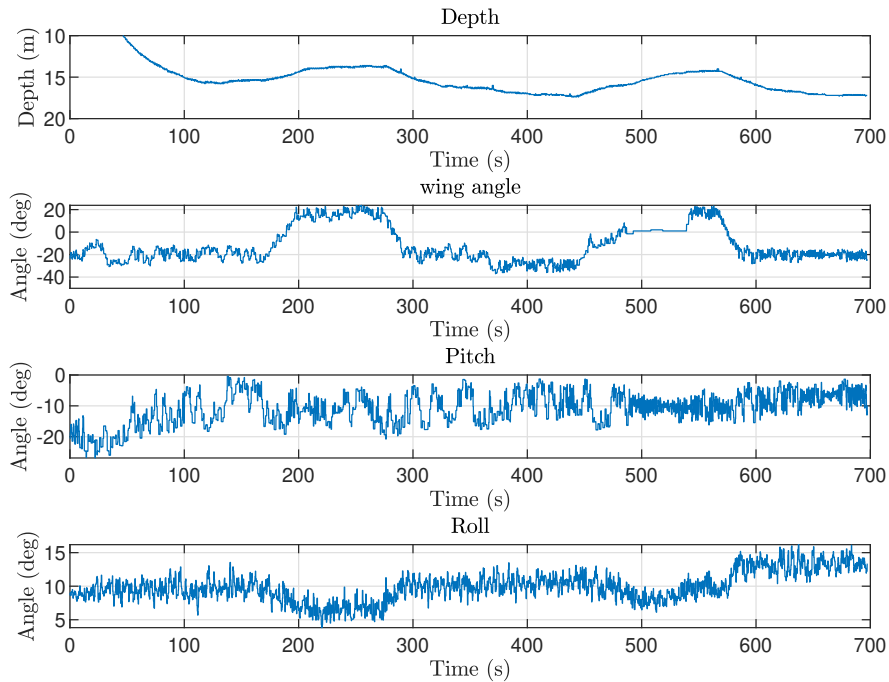
Figure 5.46: Sea trial 3

Hypothesis after test 3

1. The unstable pitch also affects the roll.

2. Wings with an increased areal will increase the range of the ROV.

3. A plastic fin can damp the oscillating pitch.

5.11.5 Sea trial 4

To increase the depth and the range, a new 150m tether cable was bought, and a new set of wings were made 4.8.2. To reduce the oscillating pitch, a tail fin was made 4.8.1.3.

The test in Figure 5.47 started without wings to see if the depth increased with a longer cable. The logging was unfortunately stopped by mistake, but the depth settled at about 16 meters. The depth is about the same as in previous tests. The tail fin worked as intended and decreased the oscillating pitch. The pitch is also changing with the depth, indicating that the angle of the tether determines the pitch. The roll is also more stable, the amplitude of the oscillation seems

to be the same, but the frequency is reduced.

We lost connection with the ROV and stopped the boat while reconnecting. After a while, we saw that the tether had a steep incline from the boat. It turned out that the ROV had taken another dive to the seafloor, this time approximately 100-120 meters based on the approximately 30 meters of cable that were pulled in with no resistance. The ROV had taken some damage, as the end capsules on the pipes had collapsed, and there was a hole in the camera bulb. Fortunately, the camera section and the ROV is sealed with epoxy, and when we were back on land, there had been no water intrusion, and the system booted. We believe the increasing roll angle indicates that one of the pipes were filled with water.

A few days later, we discovered that the connector was partly shorted and measured the resistance to 40 ohms when performing a test.
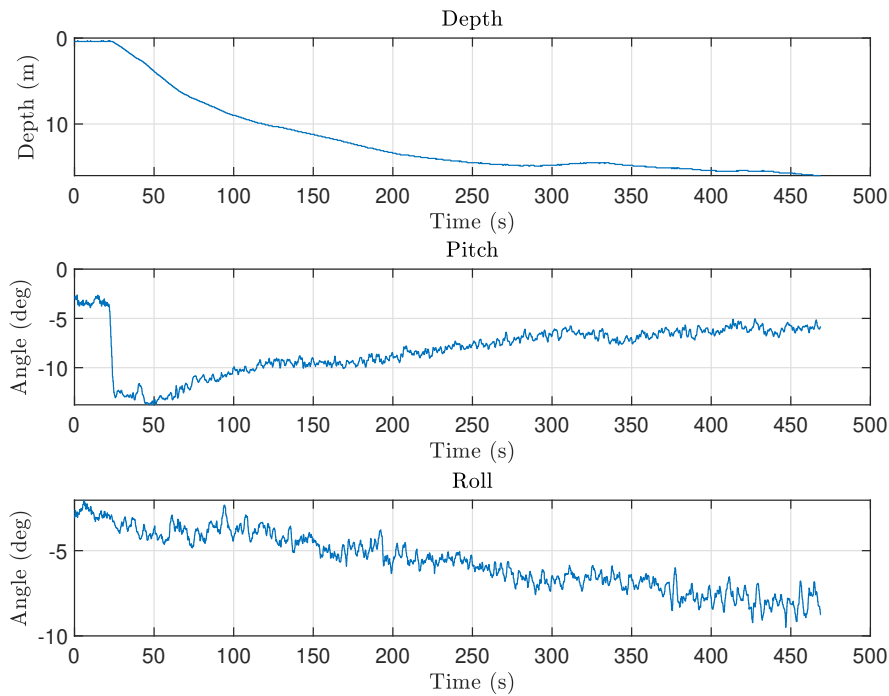
Figure 5.47: Sea trial 4

5.11.6   Sea trial 5

Because of the damaged connector, the original cable was mounted back on the ROV. To get better orientation readings, the IMU code was changed 4.7.4. Also, a new set of wings were made to improve the ROVs range in terms of depth 4.8.2. The purpose of the day was to see how much the new wings would improve the depth range. After a short duration in the first test, one of the wings was forced down to a minimum position. By inspecting a video taken of the ROV with a GoPro camera, we saw that this incident gave a yaw rotation but did not inflict roll that much. The reason for this is most likely a steep wing angle of 80 degrees, where the rotation of the wing is stopped as it hits the ROV body. The sea trial was then aborted as the 12 volts DC/DC converter stopped working. The roll and yaw are not plotted as there was a bug in the Arduino code.
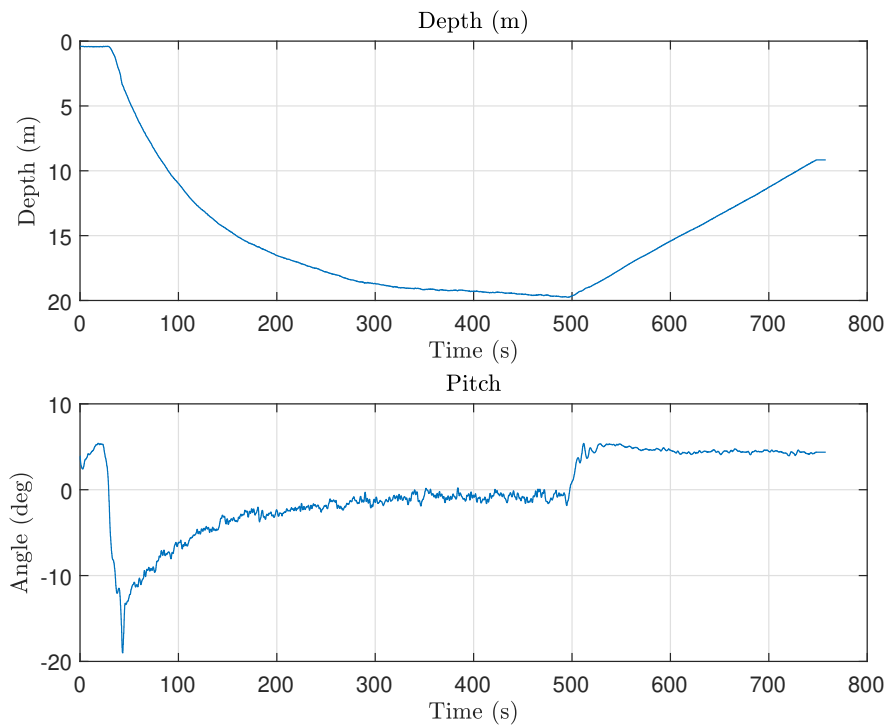
Figure 5.48: Sea trial 5

Hypothesis after test 5

1. The wings provide a significant torque on the shaft as the water follows the wing.

5.11.7   Flow simulation wings

As the wings did not work as expected, a flow simulation was executed (4.8.2). Figure 5.49 shows the simulated flow around the wing. Arrows are represented by a colour where green is slowest; this is fading into yellow and then orange with increased flow. We can observe that the result shows a slower velocity flow on the lower part of the wing. On the upper part of the wing, the velocity is increasing. It also shows that the water angle is more direct on the lower part than the upper.
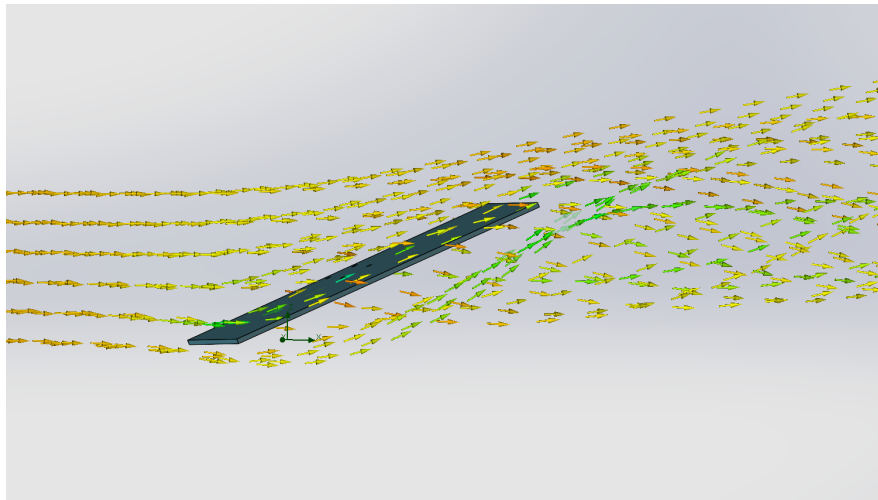
Figure 5.49: Water flow around new wing

5.11.8 Weight test

Figure 5.50: Left: Drag force from ROV at 1.3 m/s Right: Drag force from ROV at 1.7 m/s

A weight test was conducted to get an estimated drag force. To find the drag force on the ROVs tether cable, a mechanical weight was attached between the boat and the tether. The test results (Figure 5.50) indicate that a slight increase in speed gives a drastic change in drag. As the drag increases from 15kg to 25kg when the speed is increased from 1.3 to 1.8 m/s.

5.11.9 Sea trial 6

Every wing angle in sea trial 6 is relative to the ROV and not the angle of attack. The cable is 90 meters. The purpose of the day was to see how different attachment points for the cable affected the ROV and to find the effect of an increased wing size at different angles. The attachment of the gear to the shaft was glued with epoxy and tested to hold more torque than the motor could provide, by holding back the wings while running the steppers.

Sea trial 6.1

The purpose of sea trial 6.1 was to see how the ROV behaved with the old set of wings compared to the original prototype tested in sea trial 5.11.2, as there had been several physical modifi-

cations. Figure 5.51 compares the wing angle with the depth of the ROV. The range is slightly increased from the original prototype and has a span of 8 meters. The response can be inspected in Figure 5.52, and it takes close to 100 seconds to go from minimum to maximum depth. Another thing worth noticing is that increasing the wing angles absolute value from 20 to 25 degrees increases the range by 3-4 meters in total.
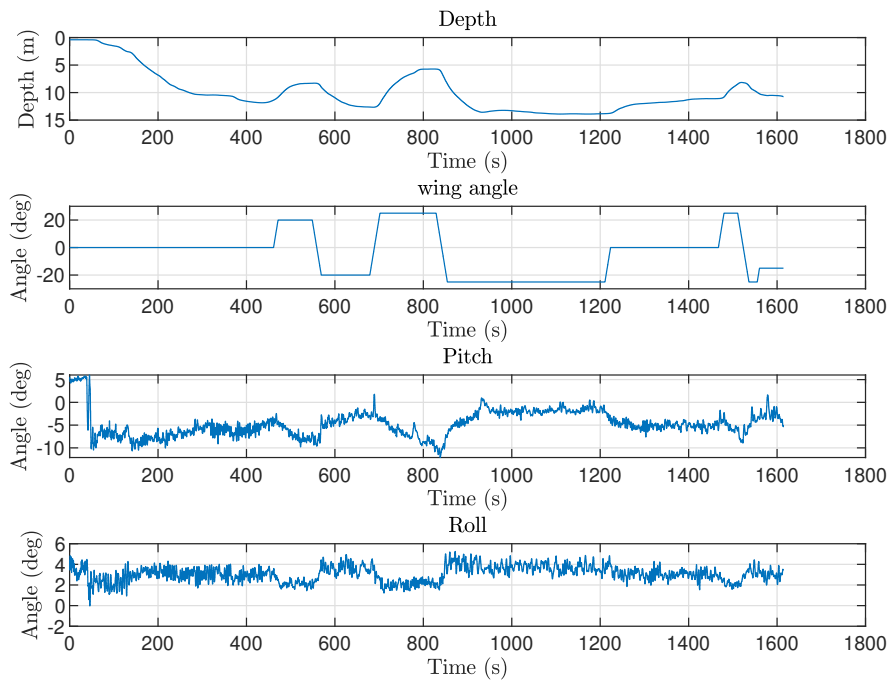
Figure 5.51: Sea trial 6.1
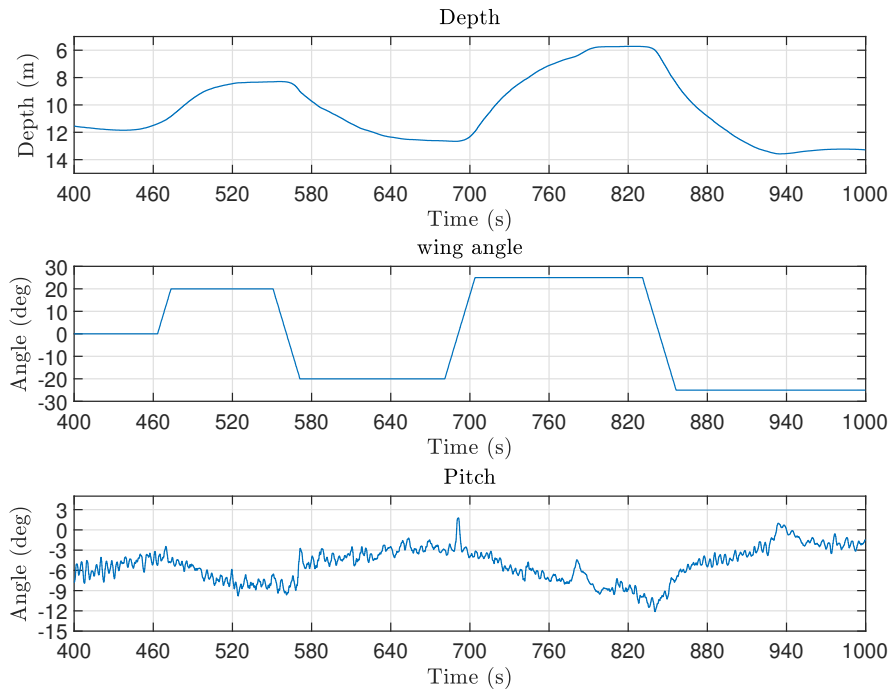
Figure 5.52: Sea trial 6.1

Sea trial 6.2

The purpose of sea trial 6.2 was to see how a different attachment point affected the ROVs behaviour by moving the attachment from the front of the ROV to be directly between the two wings. Figure 5.53 shows the results from the trial, and it is clear that the pitch angle was reduced. This is as expected as the force from the cable is moved closer to the center of rotation. Another observation is that the pitch is more stable; the same theory can most likely explain this as the attachment point is at the same x coordinate as the ROV. The increased pitch seems to let the ROV go deeper. The span of the range seems to be quite similar. Figure 5.54 is the same plot with a limited time scale. There is no particular change in depth when the wing angle is decreased from 20 to -10 degrees. The reason is most likely a speed reduction. When the test was carried out, the waves were challenging, so keeping a constant speed was difficult. The drop in speed compared with the wing angle is displayed in Figure 5.55. An interesting observation is that changing the wing angle from -20 to -25 degrees relative to the angle of attack significantly impacts the depth.
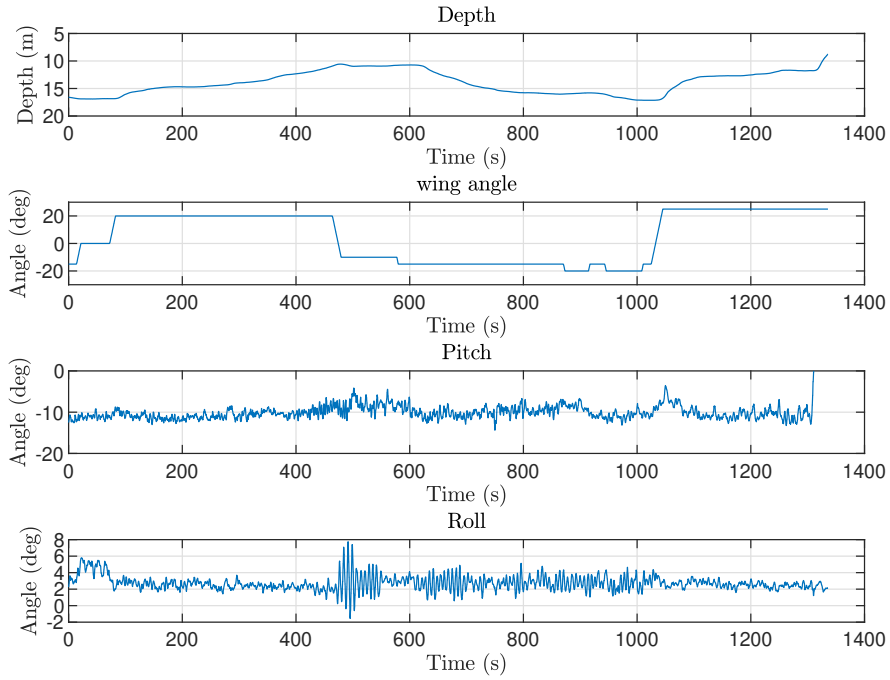
Figure 5.53: Sea trial 6.2
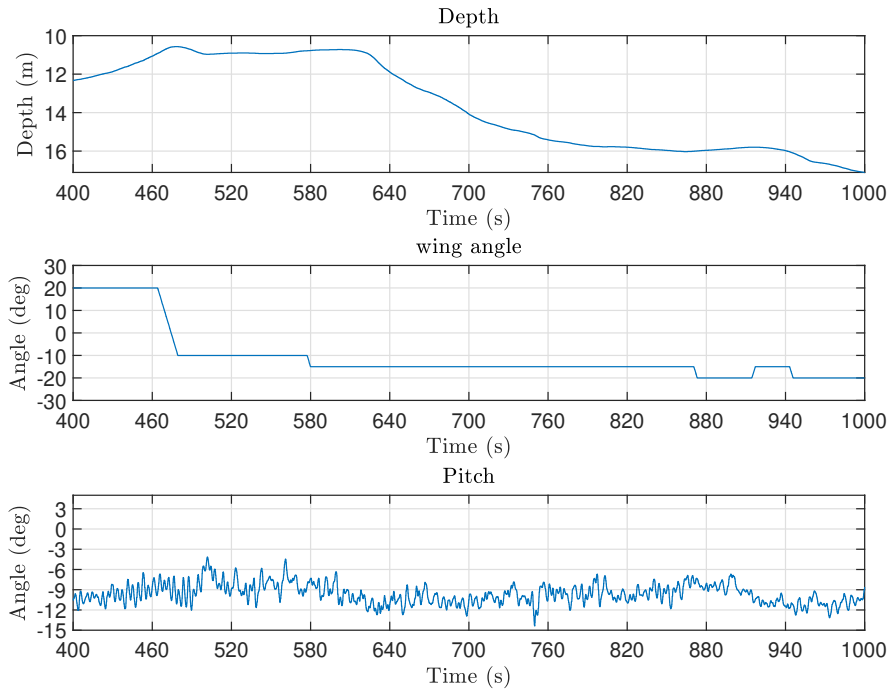
Figure 5.54: Sea trial 6.2
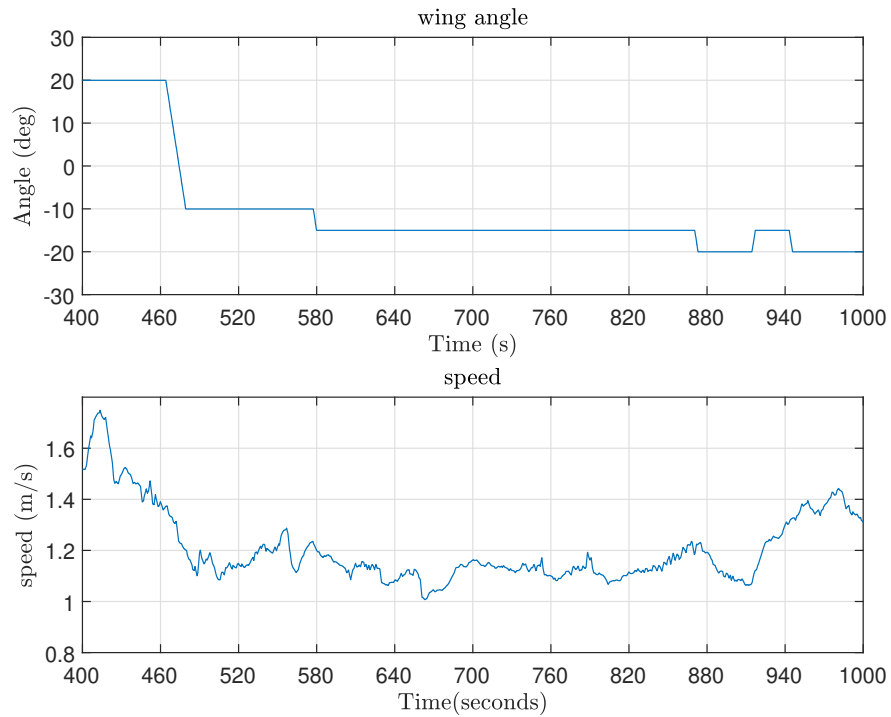
Figure 5.55: Sea trial 6.2

Sea trial 6.3

When trying to do a new test with the new wing type, the shaft slipped almost momentarily. To see how the wings affected the ROVs behaviour, they were locked at approximately -25 degrees. By inspecting Figure 5.56 it is clear that the response increase dramatically. The ROV descends 30 meters in about 100 seconds. The pitch seems to stabilize as the depth settles. The angles of the wings were not exactly the same and are most likely the reason for the increased roll.
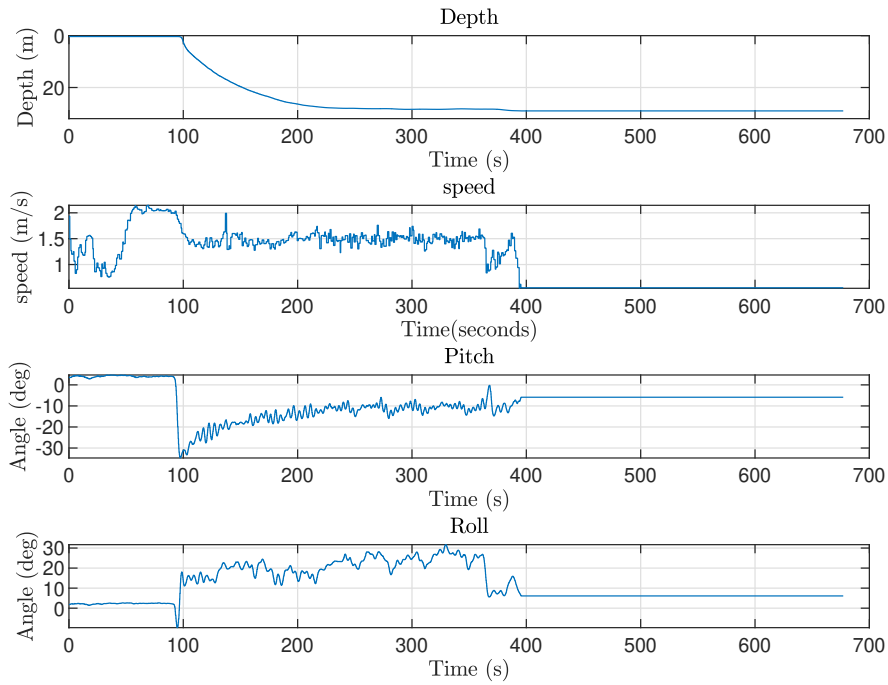
Figure 5.56: Sea trial 6.3

Sea trial 6.4

The wing angle were then changed to 25 degrees and is displayed by Figure 5.57. Even though the pitch is increased to -15 degrees the wings keeps the ROV at the surface,
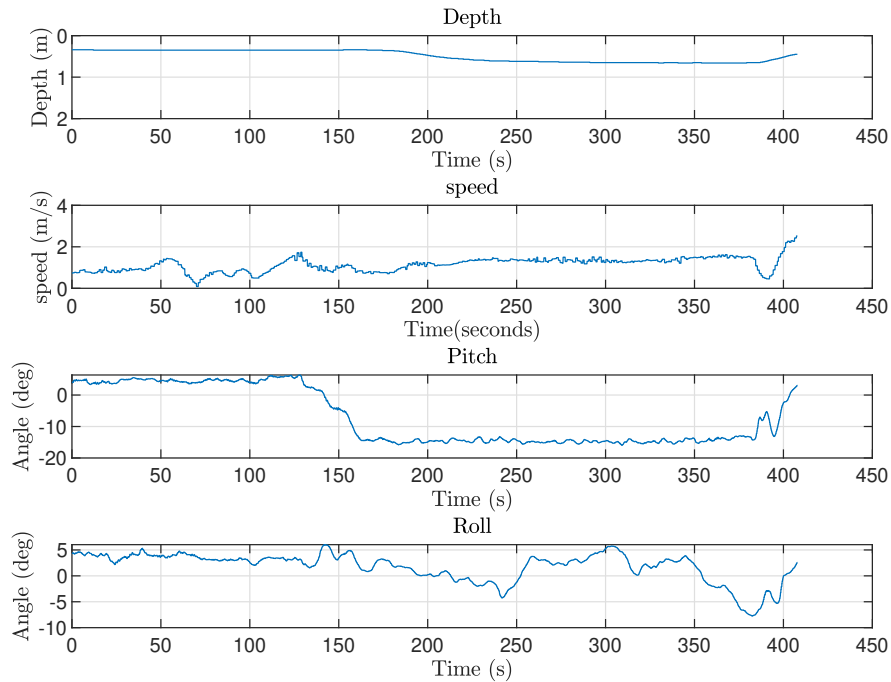
Figure 5.57: Sea trial 6.4

Sea trial 6.5

Finally, the wings were tested at an angle close to zero degrees. The ROV settles at a depth of 20 meters, as shown in Figure 5.58. The unstable pitch and high roll may result from an uneven attachment of the wing angle, as the ROV seems to be stable in the previous tests. The angle of attack on the wings oscillates between -5 and -15 degrees. This indicates that the ROV can operate at a range from 0-30 meters with the right set of wings and a towing length of 90 meters. The effective wing angle is
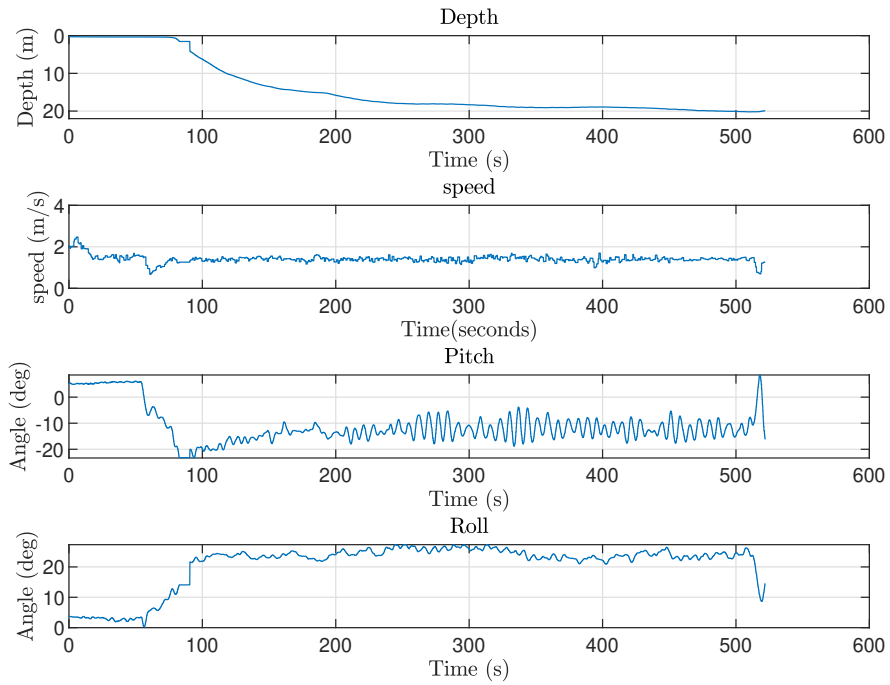
Figure 5.58: Sea trial 6.5

The last test of the day were with the same wings and angle but the attachment point were moved to the front 5.59. Again reducing the pitch and depth.
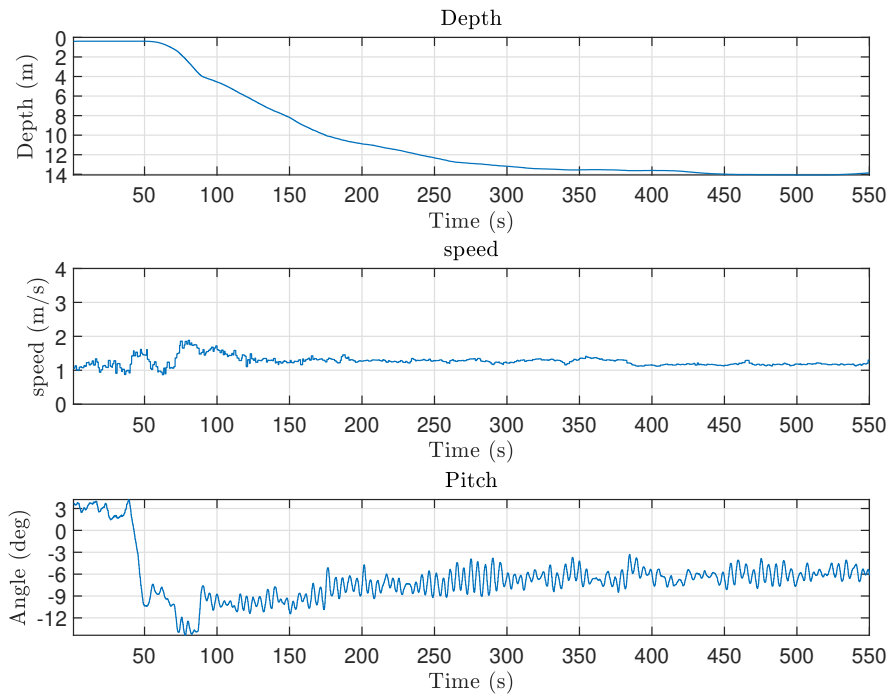
Figure 5.59: Sea trial 6.6

Figure 5.60 is a picture taken of the ROV during the last sea trial.

Figure 5.60: The final look of the Towed ROV

Chapter 6

Discussion

This chapter explains the reasoning behind the achieved results aswell as how we have the experienced the workflow throughout project.

6.1 Technical Results

This section includes a review of the technical outcomes. We will go through how the final product turned out and some feedback on whether it was successful or might have been done better.

6.1.1 ROV prototype

In the earlier stages of the project the plan was to use the prototype to test the software, and then later on implement the new software to a prototype with better physical performance. Due to corona the work on the new prototype stalled and we had to rely on the current prototype. This led us to perform several tests to improve the existing ROVs design and collect data and experience that could be useful when designing a prototype in the future.

The method of testing one physical change at a time was effective in terms of identifying how each component changed the ROVs behaviour. Most of the results were as expected, as it relies on well-known theory. However, the impact of these changes can be helpful when designing a new prototype.

Adjusting the centre of gravity and centre of buoyancy, the ROV is now clearly more stable in terms of the roll. It also allowed us to move the attachment point for the tether to align with the wings. However, the pipes installed for keeping a positive buoyancy increased the disturbance drastically.

The pitch was also reduced as the pipes were mounted over the tether attachment. The pipes giving a spoiler effect have also reduced the pitch (see Figure 5.60). However, it came at the cost of instability as the ROV became vulnerable to oscillations.

The oscillation amplitude was reduced by implementing a tail to dampen the pitch in pitch and possibly remove some disturbance by leading the water flow away from the ROV. We believe that the tail fin should be considered in future design, as it provides a notable effect. It is also easy and cheap in production, so producing a set of them to allows for easy testing to figure out which length is best suited.

As the stability was increased, it is now possible to move the attachment point closer to the wings. The new attachment gave a higher pitch angle, as the force from the tether were moved further back to be aligned with the wings. However, when testing with the new wing type, the pitch seemed to be more consistent at different depths than the previous wings' result. Even if the change in depth was considerably more extensive.

We discovered that the wing designed provided a significant torque on the shaft by testing the new wings. The torque was not something we expected, as the wing were symmetrical. The flow simulation seems to verify the results. The wing design was a success in reaching 30 meters of depth, while they also were able to keep the ROV at the surface when the angle was shifted. Based on our result, we believe that further test with unsymmetrical wing has to be executed. The reasoning is to create a wing with a torque equally distributed around the rotation point at max wing angle. Allowing for bigger wings with less torque from the motor.

Regarding the different tests done with both the wings, it seems like the lift force is peaking

at relatively high angles. This might be because of turbulence around the ROV body disturbs the flow around it, therefore the hydrofoil 2.12.4 effect from the wings is not in effect. This means that the only force on the wings is the "breaking" effect from moving water out of the way of the wing.

As the shaft kept slipping off the gear, there is a need for a change in the transmission system. We also believe that the motors should be replaced, as the torque provided by the motor seems to be lower than the torque on the shaft given by the wings.

Another concern is the drag force measured on the cable as it is in the limit on what the tether can hold according to the provider. The high force will also provide a lot of disturbance around the ROV, contributing to vibrations and oscillation.

There are still a few things we appreciate with the current ROV. First, the ROV have proven to be very solid, as it has been exposed to high pressure and have probably taken a hit when landing on the seafloor . Another feature is the flange making it easy to access the electronics, which has proven to be significant in this project.

As we have experienced a lot of trouble during the sea trials, we would suggest that when designing a new ROV, it can be practical to only integrate the pressure sensor and IMU, to see how the prototype behaves. The rest of the components should be implemented when the design of the prototype is proven. The sensor box can be potted, as there are no moving parts, and this will allow the design team to only focus on the hydrodynamics, before finding other practical solutions.

6.1.2 Side-Scan Sonar

6.1.2.1 Quality

From the results of the side-scan sonar in section 5.10, three square objects were found outside the marina at Valderøya. As we were told, there were crab traps at the location where the

side-scan sonar detected some objects, measurements seemingly close to the actual size. We are relatively sure that the deviance in the estimated size is caused by a disturbance in terms of waves and probably some uncertainties in the measurements.

Running the side-scan sonar at steady water with little disturbance and at adequate depth, we believe that the sonar we have selected is good enough for detecting crab/lobster trap. However, before attaching it to the ROV body, adjustments to the ROV have to be made to stabilize the pitch, roll and yaw to a constant value.

6.1.2.2 Stability

All previously shown sonar images have the same minor disturbance on the image. The disturbance is most likely due to the waves around the sonar disturbing its physical position in the water. As soon as the sonar is held still, the quality of the image improves. Therefore it is ill-considered to mount the side-scan sonar onto the ROV. The result from the Sea trials shown shows a disturbance in pitch which is too violent for the sonar.

As shown in Section 5.10, the stability of the sonar image goes quickly down when there is a slight movement in the sonar. Looking at the ROVs movement, it will create a sonar image that is wildly shaking, and further improvement of the stability of the ROV have to be done before a good sonar image can be received from the side-scan sonar.

The result from Hellesylt shows a larger object (Figure 5.43), most likely a boat that sank a few years ago. The image is embossed of disturbance from waves and a range that is too small. Despite the wrong range, it is pretty sensitive to waves when the depth is 45 meters, which means that the image is less affected by disturbances from the change in position when keeping a close distance to the seafloor.

6.1.3 Seafloor Tracking

The seafloor tracking was included in the project to meet future requirements. The ROV is intended to keep a consistent distance when monitoring the seafloor and have the ability to steer clear of obstacles.

The results from the test indicate that the tracking algorithm works for a wide range of seafloors, and the system's behaviour changed when the parameters were adjusted. However, as we could not test it on the current prototype, the prototype designed by the cooperating bachelor group from Product and System Design was not finished. Therefore, seafloor tracking is not verified in practical use.

6.1.4 Software solutions

A great advantages by following the three-tier architecture methodology from the start was that we could develop separate code bases which didn't collide with each other.

Graphical User Interface

The code developed was written from scratch, and the final size of the turned out to be sufficiently large; there is always room for improvement given the time-span the software systems were developed.

The choice of using React as our GUI was something that we have appreciated. At the beginning of the project, we considered various GUI implementation, and as our project requirements were set - it became more evident to use React. Primary because modern web technologies are on the rise in popularity and provide great functionality for this project's use case. As we have seen through using Leaflet 3.5.3.2, we can integrate sensor data with interactive maps in an elegant way.

A small notice regarding a feature in the Dashboard page is the bottom-right box which contains a live Chart and a live Map. The live-Map can only be used whenever the operator is connected

to the internet as Leaflet map uses a cloud API to fetch image-map tiles. Using the Mapping functionality offline would cause the image-map tiles not to update. However, having the live Map-functionality is rarely used in the first place and might be removed as viewing the history plot provides much more value to the operator.

In section 5.10, when comparing images shown in the GUI 5.41 with the ones displayed using Deepvision FV 5.40, we can see that Deepvision FV uses some post-process image manipulations to enhance the details in the sonar image. There have not been any measures to replicate the same image manipulation techniques, but we suspect this should be possible as we use the same raw sonar data. While the image in the GUI is not as good, it can show live images. Deepvision FV cannot display a live image from the sonar, which is a significant drawback when looking for a crab or lobster trap. Further improvements to the sonar image in the GUI should be made. The same quality image should be achievable by testing various image processing techniques to emphasize the graduation between the pixels. In order to fit the sonar image in the GUI, it should also be noted that it has to be re-scaled to match the image size of the video camera (640x480). This resizing may lead to a smaller image and may cause various details to be filtered out.

REST-API

Since we wanted to store data in a database, a REST-API had to be implemented as a connection point between the database and the GUI.
The REST-API follows the request-reply pattern, where you want the REST-API to be so-called 'stateless'. It should not be required to store/remember any state in the server but rather on the client. Regarding being 'stateless', the REST-API state is being violated to a small degree since the endpoints **/sensors** and **/videos** does store some information through global variables. A solution to this could be by creating a new separate API to handle the stream of live data and the serving of larges files, whilst the current REST-API could be purely used interactions with the database.

Database

The experience with using SQL as our database turned out to work very well. As its structured now, the **waypoints** is connected to **waypointsessions** through the use of *session_id,* where it could have been used a relationship instead (ex: one-to-many). The many-to-many relationship was not used throughout as it did not fit in any logical way. Even though the types of sensor data was changing a lot during development, it was relatively easy to change field names and field types of the SQL tables.

6.1.5   Digital Twin

The simulation works as well as one could expect, given the limitations of the AGX software. It can test different parts of the ROV and even proved helpful to us when the limitations of the physical ROV prevented us from testing some of the ROV systems and the communication between them. The simulation can be slow; when simulating it, the time to calculate each second can be as high as 20 seconds per second if recording the simulation. The simulation is highly dependent on the computer running the simulation, but it can present some challenges for using the software.

Communication with system

The communication between the Digital Twin and the rest of the system works as intended 5.4. It communicates as the sensors in the surface unit, ROV and the motors for the wings in the ROV. This means that for use as a platform for testing software, it fulfils its purpose in this aspect despite its differences from a real system.

6.1.5.1   Differences between the physical systems and Digital Twin

When building the Digital Twin using the AGX software, we could not recreate the instability we experienced with the physical ROV. This is because of the limitations of the AGX software. AGX does not consider the physics of flow 2.12.4 and turbulence 2.12.1.2 in its calculation 3.5.2.5 [6]. Therefore turbulence and laminar flow does not affect the simulation in the same way it affects the physical ROV.

During the testing of the physical ROV, we found that the system's stability regarding pitch was relatively low. Especially in the earlier sea trails before we added the tail (see 5.11.2, 5.11.4 and 5.11.4). In addition, the wing's ability to control the depth of the ROV was relatively limited. When simulating the system, the stability of the ROV seemed to be completely different compared to that of the physical system as seen in 5.25 and 5.44.

This is because of how hydrodynamics is calculated in the physics simulation. When AGX Dynamics calculates hydrodynamics, it considers the effect of water as a constant field. The disturbances caused by the ROV on the water are not considered [6].

This means that the AGX Dynamics simulation cannot calculate any turbulence and other disturbing forces. Therefore, it is important to build a ROV body that minimises turbulence if the simulation is to be accurate. By having sharp edges and edges perpendicular to the towing direction, the drag and turbulence of the ROV are increased 2.12.1, so these should be minimised to make the simulation more accurate and make the ROV perform better.

The original version of the ROV had large perpendicular angles relative to the towed direction and had sharp edges. These sharp edges and large surfaces will cause Vortex sheering behind the ROV 2.12.1.2, which might be what caused the instability of the physical systems.
Because of these reasons, it might be advantageous to instead use a simulation software more suited to this kind of simulation.

6.1.6 Communication

To retrieve the camera feed from the ROV, we used a TCP server/client pattern that worked quite well. Since we wanted to save the images from the ROV, we did not want to create validation checks before saving the image to the database. Therefore we used TCP from the start, which allowed for a reliable transfer of images without any problems. As we tested various resolution, we found that 640x480 pixels were enough to see what was going on. The bandwidth using this resolution was approximately 10-15% of the 74.6 Mbits/s network bandwidth (from Figure

5.3.1), which allows for more sensor data to be sent from the ROV. The FPS difference between using TCP and UDP was identical, capping on around 30 FPS. The camera TCP server in the ROV allowed only one connection by the TCP client in the REST-API. Whenever a sudden crash happened due to unforeseen actions, the TCP server just discarded the current client and allowed new camera TCP clients to connect without crashing.

As we planned the project initially, we started using ZMQ, which we slowly scaled and developed into the software system for handling communication between the REST-API and the three entities: Surface Unit, ROV and the Sonar API. As most of the data was coming from the ZMQ Publishers to ZMQ Subscribers inside the REST-API, we could easily collect the data through multiprocessing queues that worked quite well. When allowing the sensor data publishing frequency to freewheel from the ROV, various problems occurred, which was the primary explanation for predefining the publishing frequency on the various systems.

Another positive contribution of using ZMQ is its popularity and reputation. Creating a software system that relies on this protocol has turned out to work without any significant problems. By not trying to reinvent the wheel by creating our implementation of low-level TCP servers and clients, we based our implementations on top of ZMQ. This measure allowed the group to focus on the actual functionality of the software systems.

6.1.7 Camera and lights

From the result (Section 5.9) of the camera test at Runde, we can assume that an adjusted focus for this depth will provide a good photo of the seafloor with enough lighting. Daylight is essential. The lack of daylight is clearly shown when the camera is tested at night in Voldsdalsvågen Marina. Even though the lights are at full brightness, the length of vision is not as far as in daylight at lower brightness. In total, the lights should be enough for getting decent images as long as the daylight gives a considerable amount of light.

For the camera, on the other hand, we are not so sure that it is sufficient enough. We take out the factor of the quality that all image is out of focus. At the night test, the camera does not

perform any better than our own eyes. Therefore we do not believe that the camera is suited for the greater depths, where the amount of daylight is reduced. On the other hand, for this project, future groups should be able to cope with the quality of the current camera.

6.2 Project accomplishments

This section includes a discussion of how the project was carried out. We will talk about how the project was designed and the numerous experiences we gained, and the challenges we faced.

6.2.1 Distribution of work

The group members have to a certain degree, different experience when it comes to hardware and software. In order to ensure a logically sound project plan, the tasks to implement the various software and hardware systems have been delegated by experience to a certain degree. Where one participant may have been more invested in the hardware and electronic aspects of the project, while another was more engaged in the software design, we have all learned from each other and advanced in our areas of expertise.

6.2.2 Unforeseen consequences

The subsections below contain the segments responsible for causing the delays and issues during this project.

6.2.2.1 Electronics

We mentioned in sections 5.8, that multiple sea trials have been ended with electronics problem, mostly the 12 volt supply. All of them was after the ROV was filled with oil. It is to believe that the constant struggle with electronics is due to the oil. In section 3.2.1 it was mention that the food oil used is non-conduction; this is also tested in section 4.8.3.2 and confirmed. While the oil itself is non-conductive, we believe that the oil could have transported particles of electrically conductive material. From section 5.8, dirt is attached to the 12 volt supply, strengthening this assumption.

Other devices that had broken down was the RPi and stepper driver. Common for all is un-potted electronics where the components are closely mounted together, and sometimes these started working again. In contrast, the Arduino, which has components at a greater distance, had not had the same issue.

Due to different problem appearing and checking if the oil is conductive, we have not drawn the assumption stated above early enough to test with fully potted electronics. Troubleshooting has been challenging due to problem appearing and disappearing before finding a fault; this has thrown us off the assumption of particles in the oil causing problem.

6.2.2.2 Control system

As the prototype has yet to operate in greater range of depth, and the production of a new pro-totype was not finished, there was no need to change the control system. We believe that a new control system is not needed before the existing PID controller is tested, not to be sufficient.

6.2.2.3 Position estimation

A good amount of research has been put into estimation the ROV position. Due to problem at Sea trail workload have been shifted and the estimation got little attention. Therefore only an estimation using the Pythagoras theorem was used. The extensive drag from ROV will make the cable go in a straight line to the ROV, improving this estimation while driving straight forward.

6.2.3 Improvements

As we've created a working system, its still room for improvement. To further increase the ro-bustness and functionality of the project, the following suggestions could be favoured:

6.2.3.1 Hardware

- Electronics should be potted to avoid a possible faulty devices.

- Improve the ROVs hydrodynamical behaviour,

- Test types of wings to find the optimal size and shape.

- Change transmission system and motor to handle more torque.

6.2.3.2   Software

- Enhance the side-scan sonar image with image processing.

- Use Machine Learning to detect crab/lobster trap or similar.

- Implement a offline-version of Leaflet whenever internet is unavailable.

- Switch to interactive map tiles with higher zoom capabilities.

- Switching from AGX dynamics to an other simulation software more suited to simulating hydrodynamics.

Chapter 7

Conclusions

The purpose of this years Towed ROV project was to develop a new software system with new and improved functionalities. With the addition of the new software and a reduced and improved hardware configuration, future groups can focus on developing new parts of the Towed ROV instead of facing fundamental software issues.

In light of the project requirements outlined in the preliminary report, it is apparent that the project results represent the thesis's main focus areas. The ROV can be operated by the researchers using a new GUI with supporting software applications such as a REST-API and database to acquire informative data from the sea through hydrographic surveying with side-scan sonar and other useful sensors.

Implementing a new controller and a mathematical model of the ROV was not fulfilled, as the focus was on improving the ROVs hydro-dynamical behaviour and addressing issues that need to be changed for the project to proceed.

Further work on the Digital Twin was done to implement the new software and test the seafloor tracking algorithm. The test indicates that the Digital Twin can be used to test changes regarding the software. However, the limited hydrodynamic simulation cannot properly represent the ROV, as the physical system is exposed to significantly more disturbance.

All things considered, the group believes the final product in this project is a decent proof of concept. Furthermore, with further improvement of the prototype, the Towed ROV -project can be used to identify ghost fishing equipment. The project has given the group much experience in terms of planning and execution. Many problems and unexpected difficulties have arisen in recent months. These problems were solved using each other's experience and abilities whilst also learning new things on the way.

Bibliography

[1] National Marine Electronics Association - NMEA 0183. URL https://www.nmea.org/content/STANDARDS/NMEA_0183_Standard. publisher: National Marine Electronics Association.

[2] Virtual serial port driver. URL https://www.virtual-serial-port.org/. Accessed: 2021-05-07.

[3] Hydrographic surveying - methods, applications and uses, Oct 2016. URL https://theconstructor.org/surveying/hydrographic-surveying-methods-uses/13838/#:~:text=Hydrographicsurveyingorbathymetricsurveying,marineconstructions,offshoredrillingetc.

[4] NMEA FAQ | NMEA Products, October 2019. URL https://nmea.gr/2019/10/20/nmea-faq/.

[5] Serial communication, 2021. URL https://www.ibm.com/docs/en/aix/7.2?topic=communications-serial-communication.

[6] Algoryx Simulation AB. 30. hydro- and aerodynamics. URL https://www.algoryx.se/documentation/complete/agx/tags/latest/UserManual/source/hydro__and_aerodynamics.html.

[7] Algoryx Simulation AB. Agx, 2021. URL https://www.algoryx.se/.

[8] DeepVision AB. Sonar software: Deepview fv, 2018. URL https://deepvision.se/download/sonar-software/.

[9] US Naval Academy. Approximate metric equivalents for degrees, minutes, and seconds. URL `https://www.usna.edu/Users/oceano/pguth/md_help/geology_course/marine_survey/sonar_instruments.htm`. accessed 14-May-2021.

[10] Adafruit. Adafruit 9dof library, 2020. URL `https://github.com/adafruit/Adafruit_9DOF`.

[11] Segun Adebayo. Chakra ui, 2019. URL `https://chakra-ui.com/`.

[12] Vladimir Agafonkin. Leaflet, 2019. URL `https://leafletjs.com/`.

[13] Gaudenz Alder. Drawio, 2016. URL `https://app.diagrams.net/`.

[14] Arduino. Language reference, . URL `https://www.arduino.cc/reference/en/`. Accessed 13 May 2021.

[15] Arduino. Arduino libraries, . URL `https://www.arduino.cc/en/Hacking/Libraries`. Accessed 13 May 2021.

[16] Arduino. Arduino, 2021. URL `https://www.arduino.cc/`.

[17] Autodesk. Fusion 360, 2021. URL `https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR`.

[18] Axios. Axios, 2014. URL `https://github.com/axios/axios`.

[19] Terry Bartelt. Oscilloscope, dc voltage measurements. URL `https://www.wisc-online.com/learn/career-clusters/stem/ace3403/oscilloscope-dc-voltage-measurements`. Accessed: 2021-05-10.

[20] Michael Bayer. Sqlalchemy. In Amy Brown and Greg Wilson, editors, The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks. aosabook.org, 2012. URL `http://aosabook.org/en/sqlalchemy.html`.

[21] Jim Blom. Serial communication. URL `https://learn.sparkfun.com/tutorials/serial-communication/all`.

[22] The Editors of Encyclopaedia Britannica. Turbulent flow, 2020. URL https://www.britannica.com/science/turbulent-flow. Accessed 13 May 2021.

[23] The SciPy community. What is numpy?, 2021. URL https://numpy.org/doc/stable/user/whatisnumpy.html.

[24] Dassault Systèmes SolidWorks Corporation. Solidworks, 2021. URL https://www.solidworks.com/.

[25] Ryan Dahl. Node.js, 2009. URL https://en.wikipedia.org/wiki/Node.js.

[26] Deepvision. URL https://deepvision.se/.

[27] Bruce A. Mah Jeff Poskanzer Kaustubh Prabhu Dugan, Seth Elliott. The ultimate speed test tool for tcp, udp and sctp, 2021. URL https://iperf.fr/.

[28] IBM Cloud Education. Three-tier architecture, 2020. URL https://www.ibm.com/cloud/learn/three-tier-architecture.

[29] Ole Alexander Eidsvik. Identification of hydrodynamic parameters for remotely operated vehicles. URL https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2350869. locations of importance:'2.25(Vortex Shredding)','3.1.4(Damping)'.

[30] Tom Flanagan. pynmea2. URL https://github.com/Knio/pynmea2.

[31] Thor I. Fossen. Handbook of Marine Craft Hydrodynamics and Motion Control. JOHN WILEY and SONS, Ltd., John Wiley and Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom, 2011.

[32] Python Software Foundation. Event, 2021. URL https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Event.

[33] Python Software Foundation. Process, 2021. URL https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Process.

[34] Python Software Foundation. Queue, 2021. URL https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Queue.

[35] Python Software Foundation. Thread, 2021. URL https://docs.python.org/3/library/threading.html?highlight=threading#thread-objects.

[36] Python Software Foundation. Event, 2021. URL https://docs.python.org/3/library/threading.html?highlight=threading#event-objects.

[37] Python Software Foundation. Queue, 2021. URL https://docs.python.org/3/library/queue.html#module-queue.

[38] Python Software Foundation. Threading, 2021. URL https://docs.python.org/3/library/threading.html?highlight=threading#module-threading.

[39] Python Software Foundation. Multiprocessing, 2021. URL https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing.

[40] Fritzing. Fritzing, 2021. URL https://fritzing.org/.

[41] Janet Heath. Pwm: Pulse width modulation: What is it and how does it work?, 2017. URL https://www.analogictips.com/pulse-width-modulation-pwm/.

[42] Bjørn Pedersen Helmut Ormestad, Øyvind Grøn. Hydrodynamikk. URL https://snl.no/hydrodynamikk. Accessed 14. may 2021.

[43] Pieter Hintjens. Chapter 5 - advanced pub-sub patterns, 2012. URL https://zguide.zeromq.org/docs/chapter5/.

[44] Pieter Hintjens. Chapter 3 - advanced request-reply patterns, 2012. URL https://zguide.zeromq.org/docs/chapter3/.

[45] Pieter Hintjens. Ømq - the guide, 2012. URL https://zguide.zeromq.org/docs/chapter1/.

[46] Pieter Hintjens. Zeromq - github, 2021. URL https://github.com/zeromq/libzmq.

[47] https://github.com/samuelcolvin/pydantic/. Pydantic, 2017. URL https://github.com/samuelcolvin/pydantic/.

[48] https://www.highcharts.com/. Highcharts, 2009. URL https://www.highcharts.com/.

[49] Adam Hughes. Database, 2019. URL https://searchdatamanagement.techtarget.com/definition/database.

[50] Imagenex. Sidescan sonar kit. URL https://imagenex.com/products/sidescan-sonar-kit. accessed=.

[51] iMatix. Pyzmq, 2010. URL https://github.com/zeromq/pyzmq.

[52] Blue Robotics Inc. Lumen subsea lights. URL https://bluerobotics.com/store/thrusters/lights/lumen-sets-r2-rp/. Accessed: 2021-05-07.

[53] Blue Robotics Inc. Ping-arduino, 2019. URL https://github.com/bluerobotics/ping-arduino.

[54] Blue Robotics Inc. Bluerobotics ms5837 library, 2021. URL https://github.com/bluerobotics/BlueRobotics_MS5837_Library.

[55] Facebook Inc. Create react app, 2013. URL https://create-react-app.dev/docs/getting-started.

[56] Stack Exchange Inc. Stack overflow trends, 2021. URL https://insights.stackoverflow.com/trends.

[57] Adafruit Industries. Adafruit circuitpython gps. URL https://github.com/adafruit/Adafruit_CircuitPython_GPS.

[58] Adafruit Industries. Adafruit ultimate gps, 2021. URL https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf.

[59] Autun Purser Yann Marcon Martin Ludvigsen Steinar L. Ellefmo Geir Johnsen Ines Dumke, Stein M. Nornes and Fredrik Søreide. First hyperspectral imaging survey of the deep seafloor: High-resolution mapping of manganese nodules. Remote Sensing of Environment, 209:19–30, 2018. ISSN 0034-4257. doi: https://doi.org/10.1016/j.rse.2018.02.024. URL https://www.sciencedirect.com/science/article/pii/S0034425718300300.

[60] S. Widnall J. Peraire. Lecture l29 - 3d rigid body dynamics, 16.07 dynamics fall 2009, 2009. URL https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-07-dynamics-fall-2009/lecture-notes/MIT16_07F09_Lec29.pdf.

[61] JIMBLOM. Serial communication, 2012. URL https://learn.sparkfun.com/tutorials/serial-communication/uarts.

[62] Norvald Kjerstad and Norvald Kjerstad (NTNU). ekkolodd. URL https://snl.no/ekkolodd.

[63] Benny Lautrup. Physics of Continuous Matter, Exotic and Everyday Phenomena in the Macroscopic World. Institute of Physics Publishing, Temple Back, Bristol BS1 6BE, UK, 2005.

[64] Huibert-Jan Lekkerkerk. Technology in focus: Insides of side-scan sonar, Apr 2021. URL https://www.hydro-international.com/content/article/insides-of-side-scan-sonar.

[65] Chris Liechti. Welcome to pyserial's documentation, 2015. URL https://pythonhosted.org/pyserial/.

[66] Bohan Liu, Zhaojun Liu, Shaojie Men, Yongfu Li, Zhongjun Ding, Jiahao He, and Zhigang Zhao. Underwater hyperspectral imaging technology and its applications for detecting and mapping the seafloor: A review. Sensors, 20(17), 2020. ISSN 1424-8220. doi: 10.3390/s20174962. URL https://www.mdpi.com/1424-8220/20/17/4962.

[67] Encode OSS Ltd. Uvicorn, 2017-2021. URL https://www.uvicorn.org/.

[68] Martin Ludvigsen, Geir Johnsen, Petter Lagstad, Asgeir Sørensen, and Oyvind Odegard. Scientific operations combining rov and auv in the trondheim fjord. volume 48, pages 1–7, 06 2013. ISBN 978-1-4799-0000-8. doi: 10.1109/OCEANS-Bergen.2013.6608194.

[69] University of Rhode Island and Inner Space Center. Echosounder, Jan 2019. URL https://dosits.org/galleries/technology-gallery/observing-the-sea-floor/echosounder/.

[70] Oracle. What a relational database is, 2021. URL https://www.oracle.com/database/what-is-a-relational-database/.

[71] Visual Paradigm. What is entity relationship diagram, 2020. URL https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/.

[72] Teguh Putranto and Aries Sulisetyono. Lift-drag coefficient and form factor analyses of hydrofoil due to the shape and angle of attack. International Journal of Applied Engineering Research, 12:11152–11156, 01 2017.

[73] Imane Hassanain Souad Ismaili Alaoui Meryem Belgharza Fadwa Elmakhoukhi El Habib El Azzouzi Mohamed Alaoui El Belghiti Rajae Rochdi, Khouloud Lakari. Study of the electrical properties of vegetable oils as an alternative to mineral insulating oils. Advances in Environmental Biology, pages 1–4, 2014.

[74] Sebastián Ramírez. Fastapi, 2018. URL https://fastapi.tiangolo.com/.

[75] Sebastián Ramírez. Fastapi: Streamingresponse, 2018. URL https://fastapi.tiangolo.com/advanced/custom-response/?h=streaming#streamingresponse.

[76] ReactTraining. React-router, 2021. URL https://reactrouter.com/.

[77] RealVNC. Realvnc, 2021. URL https://www.realvnc.com/en/.

[78] C. C. Robusto. The cosine-haversine formula. (USA). Mathematical Association of America, 64(1):38–40, 1957. doi: http://www.jstor.org/stable/pdf/2309088.pdf.

[79] Álvaro Rodríguez Luis, José Antonio Armesto, Raúl Guanche, Carlos Barrera, and César Vidal. Simulation of marine towing cable dynamics using a finite elements method. Journal of Marine Science and Engineering, 8(2), 2020. ISSN 2077-1312. doi: 10.3390/jmse8020140. URL https://www.mdpi.com/2077-1312/8/2/140.

[80] Tina Rosado. Hydrofoils. URL https://web.mit.edu/2.972/www/reports/hydrofoil/hydrofoil.html. Accessed 15. may 2021.

[81] Eirik Rossen. Api, 2020. URL https://snl.no/API.

[82] SFUptownMaker. I2c, 2013. URL https://learn.sparkfun.com/tutorials/i2c/all.

[83] Keith Shaw and Josh Fruhlinger. What is a digital twin and why it's important to iot. Network World, 2019. doi: https://www.networkworld.com/article/3280225/what-is-digital-twin-technology-and-why-it-matters.html.

[84] Siemens. Siemens nx, 2021. URL https://www.plm.automation.siemens.com/global/en/products/nx/.

[85] SmartBear Software. Paths and operations, 2021. URL https://swagger.io/docs/specification/paths-and-operations/.

[86] SmartBear Software. Swagger ui, 2021. URL https://swagger.io/tools/swagger-ui/.

[87] StarFish. Starfish sidescan sonars. URL https://www.blueprintsubsea.com/starfish/index.php.

[88] Sysid. Server sent events, 2021. URL https://sysid.github.io/sse/.

[89] A. H. Techet. Vortex induced vibrations, 2005. URL https://ocw.mit.edu/courses/mechanical-engineering/2-22-design-principles-for-ocean-vehicles-13-42-spring-2005/readings/lec20_viv1.pdf.

[90] Airmar technology Corporation. Dst800. URL https://www.airmar.com/images/uploads/brochures/DST800.pdf.

[91] Airmar technology Corporation. Owner's guide and installation instructions, 2020. URL https://www.airmar.com/uploads/InstallGuide/17-435-01.pdf.

[92] Raccoon Thai and Henry Pham. Tableplus, 2017. URL https://docs.tableplus.com/.

[93] Inc The Mathworks. Matlab, 2021. URL https://www.mathworks.com/products/matlab.html.

[94] Timur. Earth radius by latitude (wgs 84), 2018. URL https://planetcalc.com/7721/.

[95] Tritech. Learn more about side scan sonars. URL https://www.tritech.co.uk/uploaded_files/SideScanSonars.pdf.

[96] Ultimaker. Ultimaker cura, 2021. URL https://ultimaker.com/software/ultimaker-cura.

[97] Chris Veness. The haversine formula, 2020. URL https://www.movable-type.co.uk/scripts/latlong.html.

[98] Gordon Wetzstein. Lecture 9 ee 267 virtual reality, February 2013. URL https://stanford.edu/class/ee267/lectures/lecture9.pdf.

[99] Wikipedia contributors. Python (programming language), 1991. URL https://en.wikipedia.org/wiki/Python_(programming_language).

[100] Wikipedia contributors. Electron, 2013. URL https://en.wikipedia.org/wiki/Electron_(software_framework).

[101] Wikipedia contributors. Microsoft teams, 2017. URL https://en.wikipedia.org/wiki/Microsoft_Teams.

[102] Wikipedia contributors. Javascript, 2021. URL https://en.wikipedia.org/wiki/JavaScript.

[103] Wikipedia contributors. Opencv, 2021. URL https://en.wikipedia.org/wiki/OpenCV.

[104] Wikipedia contributors. Pycharm, 2021. URL https://en.wikipedia.org/wiki/PyCharm.

[105] Wikipedia contributors. React, 2021. URL https://en.wikipedia.org/wiki/React_(JavaScript_library).

[106] Wikipedia contributors. Structured query language, 2021. URL https://en.wikipedia.org/wiki/SQL.

[107] Wikipedia contributors. Sqlite, 2021. URL https://en.wikipedia.org/wiki/SQLite.

[108] Wikipedia contributors. Visual studio code — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=1023706400. [Online; accessed 19-May-2021].

[109] Wikipedia contributors. Discord (software), 2021. URL https://en.wikipedia.org/wiki/Discord_(software).

[110] Wikipedia contributors. Foreign key, 2021. URL https://en.wikipedia.org/wiki/Foreign_key.

[111] Wikipedia contributors. Primary key, 2021. URL https://en.wikipedia.org/wiki/Primary_key.

[112] Wikipedia contributors. Representational state transfer, 2021. URL https://en.wikipedia.org/wiki/Representational_state_transfer.

[113] Wikipedia contributors, 22 March 2005. URL https://en.wikipedia.org/wiki/File:Vortex-street-animation.gif. Animation representing the two-dimensional flow patterns behind a rounded obstacle, known as a Von Kármán vortex street.

[114] Wikipedia®. Internet protocol suite, 2021. URL https://en.wikipedia.org/wiki/Internet_protocol_suite.

[115] WWF. New report from wwf says abandoned fishing gear an "immortal menace" which must be central in the fight against plastic pollution, 2021. URL https://wwf.panda.org/wwf_news/press_releases/?983716/New-report-from-WWF-says-abandoned-fishing-gear-an-immortal-menace-which-must-be-c

Appendices

A   Reports

A.1   Preproject report

Towed ROV

PRELIMINARY REPORT

Candidates

Sophus Stokke Fredborg

Jonas Halle

Jørgen Ringdal Sæter

Andreas Øie

Advisors

Runde Miljøsenter

Ottar Laurits Osen

Ronb Trulsen Bye

20 january, 2021

Summary

This report concerns the development of a modular and open-source towed ROV. It will have the ability to connect a wide range of different sensors and has a camera and supports sonar or hyperspectral imaging. With a live feed of the collected data and control of the ROV during operations, and an intuitive GUI.

This project aims to create a cheap and easy to use alternative to commercial ROV's for science and data collection, the group has been contacted by Runge who want to use the ROV to look for nets that are ghost fishing. Another focus of this project is to generalise and create software for further development of towed ROV's so that groups can focus on developing new parts of the ROV instead of making the regulation, GUI or Communication between the ROV and topside.

Contents

Chapter 1

Introduction

1.1 Background and choice of project

The towed-ROV was first built in 2018 for a bachelor project and was inspired by The Acrobat from Sea Science Inc. The ROV was later modified by two consecutive bachelors and a mechatronics project the fall of 2020. Two of the group members for this project worked on the ROV in the mechatronics course. The project is given by NTNU Ålesund with Runde miljøsenter as a partner.

There are several reasons for the choice of this project. First, we believe that there is a great potential in a low budget open-source ROV, as it is able to cover large distances collecting data and observing the seafloor. The project also has numerous challenges regarding control, operating the ROV, handling of data and physics.

The thesis problem is to develop software for operating any towed ROV with the same hardware structure and collect sensor and camera data. Develop a control system for the current ROV, making it possible to hold a constant depth and track the seafloor. We will also continue the work on a digital twin.

Chapter 2

Concepts

ROV Remotely operated vehicle

PID Proportional integral derivative controller

GUI Graphical User Interface, makes it possible to interact with a computer

API Application Programming Interface, activates functions from a remote software

Chapter 3

Project organization

3.1 Project group

The group consists of four bachelor students at NTNU Ålesund: Sophus Stokke Fredborg, Jonas Halle, Jørgen Ringdal Sæter and Andreas Øie.

3.1.1 Assignments for the project group - organization

Group lead:

- Jonas Halle

Software lead:

- Andreas Øie

Secretary:

- Rotates

3.1.2 Assignments for project leader

Responsibilities

- Group cooperation

- Time management

- Task delegation

Tasks

- Mediating conflicts

- Organising meetings

3.1.3   Assignments for secretary

Responsibilities

- Logging meetings.

- Structure files.

Tasks

- Logging meetings.

- Add and sort files used in the project.

3.1.4   Assignments for software leader

Responsibilities

- Code style

- General responsibility for project software

Tasks

- Control the code style

- Decide on a code style

- Code review

3.1.5   Assignments for all members

Responsibilities

- Logging progress.

- Working the allotted time.

- Being available during work hours.

Tasks

- Logging progress

- Supporting tasks delegated by the project leader or software leader.

- Report

3.2   Management group

The management group consists of Ottar L. Osen, Robin T. Bye and Øystein Bjelland.

Chapter 4

Agreements

4.1 Agreement with the client

4.2 Workspace and resources

Workspaces:

- The NTNU Ålesund Lab

- Working remotely from home

- Group / class rooms at NTNU Ålesund

Resources:

- Discord and Facebook for group communications.

- Computer with access to the internet and peer-reviewed articles through the NTNU network

- The NTNU Ålesund Library

- Monetary support form NTNU and Runde Miljøsenter

Persons:

- Professors and Teachers at NTNU.

- Engineers and scientist at multiple companies

Data security and confidential information:

- The project is open source so all data and information will be made available to the public domain.

4.3 Group agreements

4.3.1 Work hours and attitude

Base work time is weekdays from 9 to 15, with a minimum weekly time of at least 40 hours, If someone cannot work these hours, they must notify the group and try to work on the weekend or evenings instead. If group members break this agreement repeatedly, and talking to them does not help, there will be consequences.

When working as an engineer, time management is essential. Engineers often have much independence in approaching their work, and it is therefore extra important to keep a good workflow. Having a structured work week might be extra important when normal work is impeded, for example during a global pandemic, when most people work from home. Besides, there is a need to be flexible when an engineer works because you might be working with people in different time zones. Engineers often work flexi-time, centred around a mandatory base time, but the total amount of work in a week must be greater than the sum of the base time.

This bachelor uses this flex-time approach with a base time of 9 to 15 for 30 hours a week, but an expected work time of 40 hours each week.

4.3.2 Cooperation

For reaching our objectives, every member of the group must contribute. For this reason, we will have a weekly meeting to update each other on the last weeks' work and to discuss the primary goals for the next week. This is done to make sure that everyone does their part, and it also makes it possible to restructure tasks if something consumes more time than expected.

Almost every engineer will have to work with other people at some point, and in these cases, cooperation is fundamental. For a project to go as well as possible, every member of the team must contribute. When there are several people involved, it is not possible to have control of every operation. For this reason, communication is essential. Every project has its problems

and challenges, and changes are almost guaranteed. If everyone tries to solve these problems by them self, there will most likely be a lot of wasted time. Also, if a change is made, and another team member is not informed, this person might have to redo some of the work which is already done.

By introducing a platform where every member can share their progress and problems, it is a lot easier to keep track of the ongoing project status, making it possible to solve problems together when necessary. Our platform is a weekly meeting, shared documents and continues interaction. The structure of the platform should be configured to each project. This will probably also make the atmosphere better, as everyone is aware of the project status, and can contribute to solve potential challenges and suggest changes if necessary.

If this platform is to work as intended, everyone must show respect for each other's work and skills. There will always be some people in a project that is more resourceful than others. If the most resourceful members have an approach that allows the rest of the team to share ideas and ask for suggestions, everyone will benefit. However, if the most resourceful members are arrogant, the team will most likely suffer as the group chemistry will be hard to maintain. Another aspect of having an environment where everyone feels safe is that when someone makes a mistake, they are more likely to tell the others. Being aware of mistakes early can be crucial for the end product to be completed in time, without unnecessary expenses after the project is completed.

4.3.3   Industry 4.0

All group members have the course Industry 4.0, and the week's lectures in Industry 4.0 are scheduled, most of the base work time expires. The base time will not be move, but group members should work outside the base hours to keep up with the planned progress. Monday meeting will be rescheduled to another day if its conflict with Industry 4.0.

Chapter 5

Project description

5.1   Thesis problem

Our objective in this bachelor is to develop a functioning ROV. The ROV will be Towed behind a boat while collecting pictures, Sonar data, or other sensor data along the seafloor. The ROV will have depth control regulation using two flippers connected to the sides. It will know the distance between itself and the seafloor and regulate to keep the distance consistent. The ROV will send the data it collects to a data bank on the boat. The ROV should also be controllable from the GUI.

The GUI should also have a live-feed of the data, and the ability to "mark" data of interest live during the operation.

The ROV should be able to handle different kinds of sensors and should be modular so that researchers can connect the sensors they need to the ROV.

The bachelor is open source and has four main focuses:

- **Minimising cost** The system should be cheap to produce so that it can be used even with a smaller budget.

- **Modular design** With a modular design where the users can add or remove the sensors or equipment they do not need, further reducing the system's cost.

- **Adaptable base system** Part of the project is creating a base system that can be adapted for new projects in the future.

- **Ease of use** The system should be relatively easy to use so that people not familiar with automation can operate it.

5.2   Project requirements

- Ability to support side scan sonar or Hyperspectral imaging

- A functioning topside GUI with a live feed of sensor data, pictures and sonar data, and live control of the ROV.

- An Algorithm that stores the Sensor data from the ROV to a database in a structured and logical form

- Depth control with the ROV down to a depth of 20 meters. Or as deep as our current hardware allows for

- An algorithm that estimates test he distances from the seafloor to the ROV.

- Control algorithm that can keep a constant distance from the seafloor by using echo sounders.

- A camera with lights on the ROV

- Ability to mark places of interest in the data log during operation, for easy reference later.

5.3   Development methodology

In this project, we want first to develop an understanding of the systems we use. Therefore, we start with writing a theory base for our systems and contacting experts to find the information we need. At the same time, we need to order components to arrive in time for a practical start.

We will then create the software and prototypes necessary for testing the ROV; after testing, we will make any necessary modifications and test again until this is no longer necessary. After this, we will finalize the project and thesis.

We choose this method because we can test the system; this will then give us a more accurate system.

5.4 Collecting information

Done

- Experience from previous projects

- Read up on previous similar projects

- Investigated different technologies and software libraries to solve various problems.

To Do

- Investigate how various technologies could be implemented in a efficient way.

- Research on side scan sonar and hyper-spectral cameras for monitoring the sea floor

- Gather information for the theoretical base of the project.

- Read articles concerning useful methods concerning the project

- Find resources for making testing, information gathering and programming easier. [1]

5.5 Risk analysis

The project can be separated into two parts regarding the success chance. We believe that developing the software structure for operating and storing the data is realisable; the same applies to the depth controller and a decent model. However, for testing the seafloor tracking, we depend on the precision of the echo sounder, and the physical attributes of the ROV. As for now, the only boat available is in Hellesylt, where the fjord is too deep for testing seafloor tracking with the current ROV.

If there are problems with the hardware or the ROV structure, we believe we can prove the seafloor tracking algorithm by simulation.

The risk for damage on people and material is listed in table **??** and described by table 5.1.

---

[1] Such as local boat rental, knowledgeable people and companies that are willing to help

5.5.1 Illness issues

If one or more of the group members becomes ill due to either Covid-19 or something else, this might affect the ability to finalize the project depending on the event's time and duration. Suppose a member becomes ill or can only operate under strongly reduced capacity for three or more days. In that case, the group need to call a meeting to see if it is necessary to redistribute the workload. If the situation does not get better, the group will schedule a meeting with the control group as fast as possible.

5.5.2 Covid-19

If the University gets closed down, the group will relocate its base to one of the persons in the group apartment. The group have most likely the tools to complete the thesis.

The group looks a the possibility of a curfew as little, but if it becomes a reality, the group will not complete the thesis according to plan since a sea-trial is not possible. Therefore the group will have a focus to improve the simulation made in AGX Dynamics.

Overall, only a restriction against being together outside will stop the thesis going according to plan.

5.5.3 Risc table

Risk Analysis		Probability				
		Rare	Unlikey	Possible	Likely	Certian
Consequence	Negligible	A1	B1	C1	D1	E1
	Marginal	A2	B2	C2	D2	E2
	Critical	A3	B3	C3	D3	E3
	Catastrophic	A4	B4	C4	D4	E4

Table 5.1: Table to evaluate the risk of an event

TYPE	RISK
Leakage ROV	A3
Leakage camera	B2
Powerloss	A1
Cable winds up in engine	A4
Man over board	A2
Boat engine failure	A2
Component failure	B1
Incorrect connections	A2
Battery short ciruiting	A3

5.6 Main work activities

Id	Name	User	Estimated Hours
1	Model identification	Jonas Halle	155
1.1	Research MI		70
1.2	Collect data		20
1.3	Calculate		50
1.4	Test		15
2	Controller:	Jonas Halle	110
2.1	Research Con		20
2.2	Develop Con		40
2.3	Simulation from model		10
2.4	Test on ROV		40
3	Hardware	JørgenS	12
3.1	Test current lights		8
3.2	Create/assemble spoiler		4
4	Sea floor imaging:	sophus stokke Fredborg	190
4.1	Decide on Side step sonar or hyperspectral camera		24
4.2	Order scanning hardware		24
4.3	Develop scanning software		110
4.4	Mount scanning hardware		12
4.5	Test scanning hardware		20
5	echo sounders:	sophus stokke Fredborg	104
5.1	Develop echo sounder ROV software		40
5.2	Develop echo sounder boat software		40
5.3	Test echo sounder ROV		12
5.4	Test echo sounder Boat		12
6	Report:	all	1066
6.1	Plan		40
6.2	preproject		80
6.3	rework of pre-project after review		16
6.4	Theory		100
6.5	Documetation		60
6.6	Description		100
6.7	Discussion		150
6.8	Summary		120
6.9	Conclusion		100
6.10	Preface		100
6.11	Review		200
7	Sea floor tracking	sophus stokke Fredborg	138
7.1	Research SFT		12
7.2	Develop algorithm		110
7.3	Test algorithm		16
8	Digital twin	JørgenS	200
8.1	Further work in AGX		200
9	Possisjon estimasjon:	Jonas Halle	35
9.1	Choose method		10
9.2	Build system		15
9.3	Test		10
10	GUI:	Andreas Øie	260
10.1	GUI Base software		200
10.2	GUI - ROV		30
10.3	GUI - Database		30

11	Communication:	Andreas Øie	45
11.1	Establish commuinication protocols		5
11.2	UDP Networking		10
11.3	TCP Networking		10
11.4	Communication between GUI and ROV		20
12	Data handeling:	Andreas Øie	120
12.1	API (Server)		60
12.2	Choose database/imagebank protocoll		5
12.3	Database		10
12.4	Imagebase		5
12.5	Impl datastoring (sensor / video)		20
12.6	Impl 'Snapshot'-saving		10
12.7	Impl sessions for start/stop datastoring		10
13	Software ROV:	JørgenS	70
13.1	Image processing		20
13.2	Command handling		20
13.3	Data stream		20
13.4	Sensor software		10
14	Sea trial:	all	300
14.1	Sea trial		300
Total time	per member:	work per workday:	total time
		701.25	9.35	2805

Figure 5.1: current project plan with a total of: 2805 hours

5.7 Project schedule

5.7.1 Main project plan

As the project is of considerable size, it will naturally have many different important main topics. However, through boiling the table provided in 5.1 the key topics could be shortened down to the following key subjects;

Control system

There are several factors involved when designing the control system for sea floor tracking. We will separate this into three modules. The first module is system identification. To have a model of the system is very useful when designing a controller. For the sea floor tracking, we must also develop an algorithm that can adjust the depth set point as the sea floor changes.

Software

Regarding the software used to connect the different systems in the project, programming is a key element when it comes to operability of the ROV. Using modern existing web technologies combined with open source libraries one would be able to interact with visual effects such as mapping of images inside the graphical user interface combined with control options for sending signals through low-level networking protocols to the various systems in the project pipeline.

Report

The Thesis is the most crucial part of the entire project. It will explain the challenges of the bachelor, and explain how the group approached solving them, contain discussions on the possible future improvements and the bachelor as a whole. The report will end with a conclusion and a review of the project results.

Simulation and visualisation

As sea trials relies on good weather and is time consuming we want to design a digital twin. Since the system is quite complex and nonlinear it will be difficult to design an exact copy. However, if we can find a system model that is reliable, and use the AGX platform, it should be possible to create a twin that is useful for testing new design features and optimasing the control.

5.7.2   Project control assets

- Instagantt, project task scheduling and tracking

- Github repository for code development

- Microsoft Teams, filesharing and storage

- Meeting platforms: Zoom, Discord

5.7.3   Development assets

The development tools needed to accomplice this project are listed below.

- Programming utilities: Visual Studio Code, PyCharm, Arduino IDE

- Autodesk Fusion 360, CAD software

- PrusaSlicer, Slicing software for 3D printers

- Autodesk Eagle, Schematic/PCB drawing software

- Matlab, to calculate an experimental model

5.7.4   Internal evaluating control

Every Monday, we will have an internal meeting to discuss the present and future work. Each group member will also write a log each day, and this will help explain why a task is or is not a schedule. When progress is made in a task, this will be updated in the Gantt diagram by the end of the day.

5.8   Decision making process

The group will be available for each other and open for discussion during work hours. If there is a disagreement after discussing the matter, there will be a vote. If the vote is inconclusive, the person responsible for the subject will be granted a double vote. If there is no responsible person, the final word will be given to the team leader.

Chapter 6

Documentation

When documentation is found, the routines for the group are to store the source as pdf if possible and note when and why it was used. This applies to all types of documentation (research, equipment etc.).

If a test has been completed a report from the test will be made. Containing the result, why it did/did not work as expected and possible improvements found in the test.

The group has a Microsoft Teams group where all of the report and documentation will be stored. For code, Github will be used. The physical equipment will be stored in assigned project areas in the NTNU's workshops.

Chapter 7

Scheduled meetings and reports

7.1 Meetings

7.1.1 Meetings with control group

The group has scheduled a meeting with all the group members every other week together with the supervisor. Before every meeting, the group will discuss what questions they might have and what kind of comments or adjustments they might have to the project. They will then present a summary of what they have accomplished and where they are in relations to their plan.

7.1.2 Internal meetings

A meeting is scheduled every Monday at 09:00 AM CET. The meeting will consist of updating the project plan, reading the weekly personal activity logs, discuss the current activities and possible solutions for further work. During the meeting, a summary of the main topics will be noted.

7.2 Periodical reports

7.2.1 Progress report, including milestone

The group will be conducting daily logs, describing the daily activities performed. Instead of weekly reports, daily logs is performed instead. The daily logs will build the basis of discussion prior to the internal meetings.

Chapter 8

Planned deviation management

We plan to have a meeting every week. If a task is not completed within the allotted time, the person responsible for this task must present their progress and where they are stuck/what demanded more time than estimated. The group will then decide what actions are needed to complete the task. This can be actions such as: giving more time to the task, increasing the number of people that work on the task or brainstorming solutions.

If we disagree on how to solve a problem, the group will gather to discuss the issue. If we cannot reach an agreement, the group votes on their preferred solution, in the case of a tie, the person responsible for this problem has the deciding vote.

If a personal issue should arise, anyone in the group can call for a meeting to discuss the issue. The project leader also has a mediating role in this kind of situations. If the Project leader is involved in the conflict, and no one else can mediate, the group might need an external mediator.

Chapter 9

Equipment requirements for project execution

For this thesis, we will reuse most of the components of the existing ROV. However, some new equipment is needed.

Equipment needed for project execution:

What:	Why:	Price:
Side-scan echo sounder or hyperspectral camera	To map the seafloor	
Custom PCB board	New PCB board that takes less space than old, which now have multiple features that are currently not in use	200,-
Teensy 4.0	To change out all arduinos in ROV to one Has more I2c and serial port available, if new equipment are added	200,-
Subsea lights	The current ROV has only one working light, test and research has to be done to find the right amount of light to get a quality picture	1500,-
Paint	The ROV needs a new coat of paint both aesthetic and to protect the aluminium	200,-
Oil	Oil to used fill the ROV insides to reduce buoyancy and leakage	200,-

Appendices

A Gantt diagram

A.2 Status reports

ID301702 Hovedprosjekt	Project	Number of meeting this period 1).	Firma - Oppdragsgiver	Side
	Towed ROV	1 planned	NTNU/Runde Miljøsenter	1 av 18
Progress report	Period/week(s)	Number of hours this period. (from log) Approx per person. 120	Prosjektgruppe (navn) Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	Dato
	3			12.01.2021

Main goal/purpose for this periods work

Finish preliminary report.

Planned activities this period

- Work on preliminary report.
- Start research on GUI, side-scan sonar and experimental transfer function of system.
- Start redoing the code in the ROV.

Actually, conducted activities this period

- Finished preliminary project.
- Tested ROV condition, parts, software, etc.
- Made new PCB card (smaller) for DC supply.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

-

Main experience from this period

-

Main purpose/focus next period

- Continuing research.
- Make a plan for stable communication between devices.
- Test dataset in MATLAB to find transferer function.
- Mount ROV back together to test it a marina.

Planned activities next period

- Further investigate Jacobian (velocity control) -> implement it!

Other

Wish/need for counselling

- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

Main goal/purpose for this periods work

- Check library for telemetry and commands communication.
- Test dataset in matlab to find transfer function.
- Further research of echo sounder.
- Mount ROV back together and ready for small test at a dock.

Planned activities this period

Actually, conducted activities this period

- We have opted for ZMQ for telementry and commands.
- The datasat has been tested in matlab, with decent results. TF, SS and NLARX models have similiar results
- Software for echo sounder is almost finished.
- Motors is mounted, some connections are still lacking. 1 hour of work remains.
- Produced new cnc milled washers for the motors.
- Template for GUI
- Tested video stream from API to GUI
- Research video stream RPI
- Decided how the data will be sent
- Fixed up the hardware in the suitcase
-

Description of/ justification for potential deviation between planned and real activities

- The GA part was not planned in this period, but after discussing the possibilities of using such an algorithm to solve the inverse kinematics we got a little eager and wanted to see how it could be done. Implementation was easier than expected, and after implementing the algorithm on the 2 DOF Scara (not counting the end effectors rotation, translation and gripper, as this is not a function of the haptic's position) we wanted to test it on the more difficult 3DOF RRR arm. After updating the forward kinematics, this one also worked.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- By seeing how well the GA performs (just as good as the geometric approach) we will continue working on it.
- To make control the models easier we want to make the working space squared instead of the shape of a heart.
- To prepare for the velocity control, we want to make an option for the haptic to snap back to [0,0,0] when user interaction is aborted.

Main experience from this period

- In the process of implementing the genetic algorithm, we had some problems, the Scara robot worked nicely right from the beginning, but the 5DOF KUKA (only 3DOF was tried implemented) we also have in the simulation had some problems. At first we thought it had to do with the extra DOF, but it turned out to be the forward kinematics. This shows how important it is to have accurate forward kinematics.
- We also had some problems with twitching when using the GA, but this was solved by preserving the population between calculations (it is reset when changing models, and algorithm).

Main purpose/focus next period

- Velocity control

Planned activities next period

- Further investigate Jacobian (velocity control) -> implement it!
- Mount ROV and complete a on shore test.
- Make suitcase ready for sea trial.
- Work the main features RPI code.
- Make a payload structure.
- Work on video stream

Other

Wish/need for counselling

- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

ID301702 **Hovedprosjekt**	Project Towed ROV	Number of meeting this period 1). 1 planned	Firma - Oppdragsgiver NTNU/Runde Miljøsenter	Side 4 av 18
Progress report	Period/week(s) 1	Number of hours this period. (from log) Approx per person. 50	Prosjektgruppe (navn) Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	Dato 11.02.2021

Main goal/purpose for this periods work

- Mount ROV and complete a on shore test Jonas/Sophus
- Make suitcase ready for sea trial. Sophus
- Finish the main features RPI code. Jørgen
- Complete payload structure. Andreas
- Server/client video stream. Andreas

Planned activities this period

- Refactor Arduino(controller) Jonas
- Code input signal for collecting I/O data. (different sequences) Jonas/sophus

Actually, conducted activities this period

- The controller is tested. Need to change a stepper driver. (1 HR of work missing)
- Could not connect the ROV from the GUI. Have to look in to this. (12 hours of work missing including testing)
- Suitcase is ready for sea trial.
- Most of the RPI code is finished, still need to do some testing. Might be some features where we still have not found a solution. (20 hours missing)
- Payload structure is finished. (might need some updates further on.)
- Server/client video stream is finished.

Description of/ justification for potential deviation between planned and real activities

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

Main purpose/focus next period

- Test ROV and make it ready for sea trial.
- Finish the code for the RPI in the ROV.
- Develop the API.

Planned activities next period

- Further investigate Jacobian (velocity control) -> implement it!

Other

Wish/need for counselling

- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

Main goal/purpose for this periods work

- Test ROV and make it ready for sea trial.
- Finish the code for the RPI in the ROV
- Develop the API

Planned activities this period

- Mount ROV and complete a on shore test Jonas/Sophus (12 h)
- Test echo sounders by the dock. Sophus/Jonas
- Start working on sea floor tracking. Sophus/Jonas
- Start working on position estimation. Jonas/Sophus
- Finish the main features RPI code. Jørgen (20h)
- Develop the API. Andreas
- Start working on the GUI. Andreas

Actually, conducted activities this period

Everything is tested except of the camera and lights. It worked this fall so it should not be a problem.

The RPI code is almost finished. There are a few things down the road that we will solve later.

The real time aspect of the API is finished.

An algorithm for seafloor tracking is tested with different signals in MATLAB. Seams to work quite well.

Have done some research on position estimation.

Description of/ justification for potential deviation between planned and real activities

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

Main purpose/focus next period

Comment code (Jørgen).

AGX simulation (Jørgen).

Communication ROV (Andreas).

Continue work on GUI and API (Andreas).

Implement seafloor tracking algorithm in Python(Jonas).

Planned activities next period

- Further investigate Jacobian (velocity control) -> implement it!

Other

Wish/need for counselling

- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg Jørgen Ringdal Sæter Andreas Øie

Main goal/purpose for this periods work

    Comment code (Jørgen)

    AGX simulation (Jørgen)

    Communication ROV (Andreas)

    Continue work on GUI and API (Andreas)

    Implement seafloor tracking algorithm in Python(Jonas)

Planned activities this period

Actually conducted activities this period

Started with AGX. Have done a flow simulation in solid works.

Sensor data flow from ROV to API to GUI.

Seafloor tracking algorithm is implemented in Python.

Still have trouble getting in contact with various echo sounder providers.

Description of/ justification for potential deviation between planned and real activities

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

Main purpose/focus next period

    Test ROV by shore. (There are some issues with sensor transmitting)

    Flow simulation, implement new model. AGX.

    GUI design with implemented functions.

    Seafloor and position estimation.

Planned activities next period

-     Further investigate Jacobian (velocity control) -> implement it!

Other

Wish/need for counselling

-     Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

Main goal/purpose for this periods work

Test ROV by shore. (There are some issues with sensor transmitting)

Flow simulation, implement new model. AGX.

Gui design with implemented functions.

Seafloor and position estimation.

Planned activities this period

Side scan sonar – have decided

Have added settings to GUI for sensor handling with enable/disable.

Some work on AGX.

Some work on RPI.

Added new features to seafloor tracking for optimal change of set point.

Description of/ justification for potential deviation between planned and real activities

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

Main purpose/focus next period

- Research for lending a boat.
- Find problems ROV.
- By shore trial
- Commands added to GUI.
- AGX -> Refactoring, further work,

Planned activities next period

- Further investigate Jacobian (velocity control) -> implement it!

Other

Wish/need for counselling

- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

ID301702 **Hovedprosjekt**	Project	Number of meeting this period 1).	Firma - Oppdragsgiver	Side
	Towed ROV	1 planned	NTNU/Runde Miljøsenter	9 av 18
Progress report	Period/week(s)	Number of hours this period. (from	Prosjektgruppe (navn)	Dato
	1	log) Approx per person. 50	Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	03.03.2021

Main goal/purpose for this periods work

- Research for lending a boat. 4.3
- Find problems ROV 4.3 electronics(sensor values etc)
- By shore trial 5.3
- Commands added to GUI 9.3
- AGX -> Refactoring, further work,

Planned activities this period

Actually conducted activities this period

- Boat (Runde/hellesylt).
- The bar30 sensor was broken and caused troubles for the echosounder as well. New sensor and i2c adapter fixed the problem.
- As we made an agreement of sea trial with Runde, on shore trial was not necessary.
- Commands is added to GUI.
- AGX is postponed, as the focus is turned towards testing.

Description of/ justification for potential deviation between planned and real activities

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

Main purpose/focus next period

Planned activities next period

- Test at Runde Miljøsenter.
- Further work in new GUI.
- Test that everything is ready for the first sea trial.

Other

Wish/need for counselling

- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

ID301702 **Hovedprosjekt**	Project	Number of meeting this period 1).	Firma - Oppdragsgiver	Side
	Towed ROV	1 planned	NTNU/Runde Miljøsenter	10 av 18
Progress report	Period/week(s)	Number of hours this period. (from	Prosjektgruppe (navn)	Dato
	2	log) Approx per person. 120	Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	09.03.2021

Main goal/purpose for this periods work

3-DAY Workshop w/ MVP Test @ Runde Miljøsenter

Test at Runde:

- Echo sounders

- I/O response

- Camera/lights

Create a model using the system identification toolbox by adding I/O data of the system.

Further work on sea floor tracking by applying echo data and behaviour

Improve software depending on eventual problems in testing.

Update after test:

- We had several problems during the test period at Runde. Raspberry Pi, SD card, IMU and the old software were all causes for the problems. Water leakage?

- We have decided to stop using the old version as there have been a lot of bugs, so we need to finish the new beta version of the software.

- We want to finish this this week so we can do further testing at Hellesylt next week.

Planned activities this period

Actually conducted activities this period:

- The new software is now up and running, there are some minor bugs that should be fixed, but all over the system seems to be more stable and reliable than the previous system.

- The side plates forced the ROV to flip around. They were removed for now in order to fix this issue.

- The spoiler took away some pitch as expected, decreasing the depth from about 14 meters to 4 meter. Seems that the pitch angle is the main actuator for the system at the moment.

Description of/ justification for potential deviation between planned and real activities

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

Main purpose/focus next period

Planned activities next period

Easter:

Fix bugs software

Filter IMU

Work on the report

Test echosounder boat

ID301702 **Hovedprosjekt**	Project	Number of meeting this period 1).	Firma - Oppdragsgiver	Side
	Towed ROV	1 planned	NTNU/Runde Miljøsenter	11 av 18
Progress report	Period/week(s)	Number of hours this period. (from	Prosjektgruppe (navn)	Dato
	2	log) Approx per person. 120	Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	09.03.2021

Further work AGX
Some changes GUI
Other
Wish/need for counselling
-     Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

Main goal/purpose for this periods work

Planned activities this period

- Fix bugs software
- Filter IMU
- Work on the report
- Test echosounder boat
- Further work AGX
- Some changes GUI

Actually conducted activities this period:

- Bugs regarding commands is fixed.
- Vertical acceleration relative to gravity is implemented.
- There are committed some work on the report, however not as much as we planned.
- Connecting the system as a digital twin is now working.
- Redesign 3d model for simulation.
- GUI, refactor, added charts, started on GUI v2.0(database etc).
- Seafloor tracking tested in simulation.
- Mounted two new tether cable on spools.
- New sideplates/wings cut out in acrylic plastic.

Description of/ justification for potential deviation between planned and real activities

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

- Simplification to the 3D model of the ROV must be done to get a simulation that is fast enough to work with.

Main purpose/focus next period

Planned activities next period

- Test rov with oil, and different wings.
- Implement seafloor tracking.
- Improve rov behaviour in agx simulation.
- Start working on side scan sonar API.
- Make filter for roll, pitch.
- Implement session storage in gui.

Other

Wish/need for counselling

- Nothing in particular

ID301702 **Hovedprosjekt**	Project Towed ROV	Number of meeting this period 1). 1 planned	Firma - Oppdragsgiver NTNU/Runde Miljøsenter	Side 13 av 18
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx per person. 90	Prosjektgruppe (navn) Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	Dato 22.03.2021

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg Jørgen Ringdal Sæter Andreas Øie

ID301702 **Hovedprosjekt**	Project	Number of meeting this period 1).	Firma - Oppdragsgiver	Side
	Towed ROV	1 planned	NTNU/Runde Miljøsenter	14 av 18
Progress report	Period/week(s)	Number of hours this period. (from	Prosjektgruppe (navn)	Dato
	2	log) Approx per person. 100	Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	05.04.2021

Main goal/purpose for this periods work

Planned activities this period

- Test ROV with oil, and different wings
- Implement seafloor tracking
- Improve ROV behaviour in AGX simulation
- Start working on Side Scan Sonar API
- Make filter for roll and pitch
- Implement Session-based data-storage in APP / GUI

Actually conducted activities this period:

AGX:

- The digital simulation has better performance.
- Added controller to AGX
- Echo sounder added to AGX.
- Increasing wing size, and setting the hydrodynamic parameters to very specific values had a positive effect on the ROV Control.

Side Scan Sonar API:

- Side Scan Sensor is up and running.
- Added support for sending from Sonar API to our Python API
- Tested side sonar (needs more)

IMU

- Added low pass filter to IMU

GUI

- Implemented "Session" for storing data in batches (sensor data w/lat/long & image)
- Added support for showing live-feed from Sonar to GUI

ROV

- The ROV has been filled with oil and pipes have been installed to get a low positive buoyance.

The pitch is reduced but the depth remains the same. We believe the reduced positive buoyancy is the reason.

Description of/ justification for potential deviation between planned and real activities

When performing the sea trial, we discovered that the bolts for mounting them was not properly attached to the bracket. We have created new brackets.

There have been some issues with electronics. The usb between RPI Arduino, might have caused some problems. Steppers moves for a short duration when opening serial port.

Seafloor tracking is ready, but not implemented. Will, most likely, only be tested in AGX.

AGX performance has slowed progress. Communication issues with DeepVision caused us to get the API software from them much later than expected and waste a lot of time to get it.

ID301702 **Hovedprosjekt**	Project Towed ROV	Number of meeting this period 1). 1 planned	Firma - Oppdragsgiver NTNU/Runde Miljøsenter	Side 15 av 18
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx per person. 100	Prosjektgruppe (navn) Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	Dato 05.04.2021

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

- Sonar needs a stable mount, and is tested separately from the ROV for this reason.

Main experience from this period

- A lot of late nights with problems with the electronics, caused a lot of delay. A late arrival of the sonar API caused a lot of delay, we had to contact deepVision multiple times and ask for the API many times before we actually got it.

Main purpose/focus next period

Report, Side-Scan Sonar, Regulation in ROV with AGX. Sea Trails. Seafloor tracking

Planned activities next period

- Test ROV with oil, and different wings.
- Implement seafloor tracking.
- Improve ROV behaviour in agx simulation.
- Start working on side scan sonar API.
- Make filter for roll, pitch.
- Implement session storage in GUI.

Other

Wish/need for counselling

- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg Jørgen Ringdal Sæter Andreas Øie

ID301702 Hovedprosjekt Progress report	Project Towed ROV	Number of meeting this period 1). 1 planned	Firma - Oppdragsgiver NTNU/Runde Miljøsenter	Side 16 av 18
	Period/week(s) 2	Number of hours this period. (from log) Approx per person. 140	Prosjektgruppe (navn) Sophus Fredborg Stokke Jonas Halle Jørgen Ringdal Sæter Andreas Øie	Dato 19.04.2021

Main goal/purpose for this periods work

Planned activities this period

- Test ROV with oil and different wings.
- Implement seafloor tracking.
- Improve ROV behaviour in AGX simulation.
- Start working on Sonar API.
- Make filter for roll, pitch.
- Implement session storage in GUI.

Actually conducted activities this period:

AGX:

- Communicates between GUI, RPi code for ROV, and AGX, the ROV controls depth and gets values back from the RPi code in the ROV (digital twin).
- Does not go very deep, around 10 m
- Added the tanks and ability to scale the system with just setting the ROV scale, so that wings and tanks move position and so that the task scale with the rest of the ROV. The wing scale left open to control intentionally.
-

Side Scan Sonar:

- There were waves making it difficult to hold the sonar stable, however we believe we spotted a sunken ship in Hellesylt.
- The colours are changed in the stream, making it easier to spot objects.

ROV

- Added a tail to the ROV. The pitch became more stable.
- There was a leakage in the pipes, resulting in a sunken ROV. The ROV went down to approximately 120 meters. The camera bulb and pipes took damage. Everything else seems to be fine.
- The ROV is fixed and ready for new sea trials.
- A filter is implemented for roll, pitch and vertical acceleration.
- The echo sounder on the boat is now working and tested.

GUI

- Session storage is implemented.
- Modular design is implemented.

Description of/ justification for potential deviation between planned and real activities

- The sea trials had to be aborted and postponed due to the accident.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

- The ROV body can handle great depths, but the arrangement with the pipes is a weak spot.

Main purpose/focus next period
- Report, Side-Scan Sonar, Regulation in ROV with AGX. Sea Trails. Seafloor tracking
Planned activities next period
Other
Wish/need for counselling
- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

Main goal/purpose for this periods work

Planned activities this period

- Report

- Work on side-scan sonar.

- Regulation in ROV with AGX

- Sea trials

Actually conducted activities this period:

- Some result from ROV testing

- Camera tested at marina at night.

- Got result from side scan sonar.

- Seafloor tracking added to ROV code.

- GUI finished.

- Filter tuned for pitch and roll.

Description of/ justification for potential deviation between planned and real activities

- The sea trials had to be aborted and postponed due to the accident.

Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report

Main experience from this period

- Deepvision forgot to send us the API for the side-scan sonar, have take a long time to get a hold of it.

Main purpose/focus next period

- Report, Side-Scan Sonar, Seafloor tracking in simulation.

Planned activities next period

Other

Wish/need for counselling

- Nothing in particular

Approval/signature group leader	Signature other group participants
Jonas Halle	Sophus Stokke Fredborg
	Jørgen Ringdal Sæter
	Andreas Øie

B   Elschematic

Arduino Mega

RESET
AREF
IOREF

A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15

SDA
SCL

D38
D39
D40
D41
D42
D43
D44
D45
D46
D47
D48
D49
D50
D51
D52
D53

3V3   5V   VIN

D13 PWM
D12 PWM
D11 PWM
D10 PWM
D9 PWM
D8 PWM
D7 PWM
D6 PWM
D5 PWM
D4 PWM
D2 PWM
D3 PWM
D1 TX0
D0 RX0

D14/TX3
D15/RX3
D16 PWM/TX2
D17 PWM/RX2
D18/TX1
D19/RX1

Arduino
Mega
2560
(Rev3)

D22
D23
D24
D25
D26
D27
D28
D29
D30
D31
D32
D33
D34
D35
D36
D37

GND

Echo sounder

PING-SONAR
1 TX   5V 4
2 Rx   GND 3

Pressure sensor

MS5837-30BA1
1 SDA   5V 4
2 SCL   GND 3

Adafruit 9dof IMU

IMU
1 SDA   5V 4
2 SCL   GND 3

C   Gantt diagram

Bachelor - TowedROV

Read-only view, generated on 19 May 2021

Instagantt

ACTIVITIES	ASSIGNEE	EH	START	DUE	%
Model identification		16h	13Jan	10Mar	49%
1 Research MI	Jonas malle	30h	13Jan	12Feb	100%
2 Collect data	Jonas malle	30h	08Feb	01Mar	0%
3 Calculate	Jonas malle	50h	01Mar	05Mar	0%
4 Test	Jonas malle	150	08Mar	10Mar	0%
Controller:		110h	01Feb	16Apr	49%
6 Research Controller	Jonas malle	20h	01Feb	01Mar	100%
7 Develop Controller	Jonas malle	80h	10Mar	29Mar	0%
8 Simulation from model	Jonas malle	10h	10Mar	29Mar	0%
9 Test on ROV	Jonas malle	40h	01Apr	16Apr	0%
Hardware		17h	12Jan	08Feb	100%
11 Test motors	Jonas malle	3h	12Jan	14Jan	100%
12 Test current lights	Jonas malle	4h	14Jan	21Jan	100%
13 Create/assemble spoiler	Jonas malle	4h	18Jan	08Feb	100%
Sea floor imaging:		190h	20Jan	11May	89%
15 Decide on Side scan sonar ...	sophus stokke F...	30h	20Jan	22Mar	100%
16 Order scanning hardware	sophus stokke F...	20h	22Mar	30Mar	100%
17 Develop scanning software	sophus stokke F...	110h	08Feb	05May	100%
18 Mount scanning hardware	sophus stokke F...	12h	06Apr	05May	30%
19 Test scanning hardware	sophus stokke F...	20h	27Apr	11May	100%
Echo Sounders:		120h	22Jan	19Apr	100%
21 Develop echo sounder ROV ...	sophus stokke F...	80h	22Jan	16Feb	100%
22 Develop echo sounder boat ...	sophus stokke F...	80h	22Jan	16Feb	100%
23 Research	sophus stokke F...	20h	22Jan	29Jan	100%
24 Test echo sounder ROV	sophus stokke F...	50h	11Feb	16Feb	100%
25 Test echo sounder Boat	sophus stokke F...	12h	12Feb	19Apr	100%
Report:		1960h	13Jan	14May	100%
27 Plan	Vic all	40h	13Jan	27Jan	100%
28 preproject	Vic all	80h	14Jan	29Jan	100%
29 rework of pre-project after r...	Vic all	50h	25Jan	27Jan	100%
30 Theory	Vic all	50h	26Jan	16Mar	100%
31 Documentation	Vic all	40h	01Feb	26Apr	100%
32 Description	Vic all	150h	12Apr	30Apr	100%
33 Discussion	Vic all	150h	19Apr	07May	100%
34 Summary	Vic all	150h	26Apr	07May	100%
35 Conclusion	Vic all	100h	06May	07May	100%
36 Preface	Vic all	100h	10May	12May	100%
37 Review	Vic all	200h	12May	14May	100%
Sea floor tracking		130h	15Feb	30Apr	100%
39 Research SFT	Jonas malle	30h	15Feb	11Mar	100%
40 Develop algorithm	Jonas malle	110h	16Mar	26Apr	100%
41 Test algorithm	Jonas malle	60h	05Apr	30Apr	100%
Digital twin		260h	01Jan	29Apr	100%
43 Further work in AGX	Jørgen4	200h	01Jan	11Mar	100%
44 Simulate propper ROV beha...	Jørgen4	60h	02Apr	26Apr	100%
45 Connect AGX with system fo...	Jørgen4	60h	02Apr	26Apr	100%
Position estimation:		45h	01Feb	16Apr	100%
47 Choose method	sophus stokke F...	50h	01Feb	15Feb	100%
48 Build system	sophus stokke F...	10h	15Feb	31Mar	100%
49 Test	sophus stokke F...	50h	01Apr	16Apr	100%
GUI - Development:		150h	29Jan	30Mar	100%
51 GUI Development version	Andreas ller	150h	29Jan	30Mar	100%
GUI - Production:		80h	02Apr	16Apr	100%
53 GUI Production version	Andreas ller	80h	02Apr	16Apr	100%
Communication:		45h	11Jan	16Apr	100%
55 Establish communication p...	Andreas ller	5h	11Jan	24Jan	100%
56 UDP Networking	Andreas ller	80h	18Jan	15Mar	100%
57 TCP Networking	Andreas ller	80h	18Jan	15Mar	100%
58 Communication between G...	Andreas ller	200h	05Feb	16Apr	100%
Data management:		130h	18Jan	15Apr	100%
60 API (Server)	Andreas ller	180h	18Jan	15Apr	100%
61 Choose database/imageba...	Andreas ller	5h	26Jan	27Jan	100%
62 Database	Andreas ller	80h	27Jan	16Apr	100%
63 Impl datastoring (sensor / vi...	Andreas ller	80h	08Feb	15Mar	100%
64 Impl Snapshot-saving	Andreas ller	50h	22Feb	15Mar	100%
65 Impl sessions for start/stop ...	Andreas ller	50h	22Feb	15Mar	100%
66 Imagebase	Andreas ller	50h	05Apr	15Apr	100%
67 Add API Endpoints		5h	06Apr	15Apr	100%
68 Create Session options in GUI		5h	06Apr	15Apr	100%
Software ROV:		140h	11Jan	30Apr	100%
70 Image processing	Jørgen5	80h	11Jan	16Feb	100%
71 Command handling	Jørgen5	80h	11Jan	01Feb	100%
72 Data stream	Jørgen5	80h	11Jan	01Feb	100%
73 Sensor software	Jørgen5	50h	26Jan	16Feb	100%
74 Modify code for new features	Jørgen5	60h	02Feb	30Apr	100%
Sea trial:		200h	16Mar	30Apr	100%
76 Sea trial	Vic all	200h	16Mar	30Apr	100%

x

D   REST-API Documentation

FastAPI `0.1.0` `OAS3`

/openapi.json

videos ⌄

POST	**/videos/preference** Video Preference

GET	**/videos/snap** Trigger Image Snapshot

GET	**/videos/live** Live Video Feed

GET	**/videos/{img_name}** Get Img From Database

commands ⌄

POST	**/commands/** Send Command To Rov

sensors ⌄

GET	**/sensors/toggle_recording** Toggle Csv Recorder

GET	**/sensors/live** Live Sensor Data Feed

waypoints ⌄

GET	**/waypoints/single/{waypoint_id}** Get Waypoint

GET	**/waypoints/** Get Multiple Waypoints

POST	**/waypoints/** Create Waypoint

GET	**/waypoints/{session_id}** Get Waypoints By Session Id

DELETE	**/waypoints/{session_id}** Delete Waypoints By Session Id

waypoint_sessions ⌄

| GET | /waypoint_sessions/ Get Multiple Waypoint Sessions |

| POST | /waypoint_sessions/ Create Waypoint Session |

| GET | /waypoint_sessions/completed Get Completed Waypoint Sessions |

| GET | /waypoint_sessions/uncompleted Get Uncompleted Waypoint Sessions |

| GET | /waypoint_sessions/{session_id} Get Waypoint Session |

| PUT | /waypoint_sessions/{session_id} Update Waypoint Session |

| DELETE | /waypoint_sessions/{session_id} Delete Waypoint Session By Session Id |

settings ⌄

| GET | /settings/ Get Settings |

| POST | /settings/ Create Setting |

| GET | /settings/{id} Get Setting |

| DELETE | /settings/{id} Delete Setting |

| PUT | /settings/{sensor_id} Update Sensor Enabled |

Schemas ⌄

AbstractWaypoint ❯

Command ❯

HTTPValidationError >

Sensor >

Setting >

SettingCreate >

SettingUpdate >

ValidationError >

VideoPreference >

Waypoint >

WaypointSession >

WaypointSessionCreate >

WaypointSessionUpdate >

E OSMEthernet Software API

DeepVision

OSMEthernet Software API

Research & Development
Sonar Systems
Linköping, Sweden

Table of Contents

Introduction

This document provides an introduction to the multi platform C++ API for the DeepVision OSMEthernet. The example included is not designed to be a complete solution, but rather to show the basics of interfacing and setting up the DeepVision OSMEthernet module using the OSMEthernet Software API.

The API is designed to interface DeepVision OSMEthernet sonars and to store sonar data in the .dvs format to be used in the DeepVision DeepView software.


```
debian@beaglebone:~/API$ make
makefile:26: .depend: No such file or directory
rm -f ./.depend
g++ -g -ggdb -ffunction-sections -O0 -std=c++14 -DDV_LINUX -MM EthernetSonarExample.cpp EthernetSonarAPI.cpp DVSFileWriter.cpp DSSPParser.cpp>>./.depend;
g++  -g -ggdb -ffunction-sections -O0 -std=c++14 -DDV_LINUX  -c -o EthernetSonarExample.o EthernetSonarExample.cpp
g++  -g -ggdb -ffunction-sections -O0 -std=c++14 -DDV_LINUX  -c -o EthernetSonarAPI.o EthernetSonarAPI.cpp
g++  -g -ggdb -ffunction-sections -O0 -std=c++14 -DDV_LINUX  -c -o DVSFileWriter.o DVSFileWriter.cpp
g++  -g -ggdb -ffunction-sections -O0 -std=c++14 -DDV_LINUX  -c -o DSSPParser.o DSSPParser.cpp
g++  -o EthernetSonarExample EthernetSonarExample.o EthernetSonarAPI.o DVSFileWriter.o DSSPParser.o
debian@beaglebone:~/API$
```

Figure 1: Example of ExampleSonarAPI being run.

API Contents

The API consists of one example file, EthernetSonarExample.cpp, the main API file, EthernetSonarAPI.cpp, the DSSPParser.cpp, the DVSFileWriter.cpp and a makefile.

The important classes the functions needed to use them are described below.

The EthernetSonarExample File

The EthernetSonarExample file calls all functions needed to record a .dvs file. It calls the CSonarInterface::FindSonar(std::vector<std::string> * ip, std::vector<int> * ports) function to find all connected sonars. If the IPs and ports of the sonar modules are known, the sonar can be initiated directly using the IP and port in the constructor.

The EthernetSonarAPI Class

The CSonarInterface class is the main class of the OSMEthernet Software API. The important public functions of the class are described below:

CSonarInterface(std::string ip, int port):
Class constructor, takes the IP and port of the sonar to be connected.

static bool FindSonar(std::vector<std::string> * ip, std::vector<int> * ports)
Static function for finding all sonars connected to the system. Adds IP and port of each sonar found to the vectors with the corresponding names. Returns true if one or more sonar was found.

void DSSP_SetPulseDual(
    U32 nPeriods0,
    float StartFreq0,
    float DeltaFreq0.
    U32 nPeriods1,
    float StartFreq1,
    float DeltaFreq1)
Function for setting the characteristics of the sonar pulse of a two channel sonar. All parameters ending with a "0" are applied to channel one, which defaults to the left channel of side scan sonar modules.
The nPeriodsX is the number of output periods of each ping.
The StartFreqX is the starting frequency of the ping in **Hz**.
The DeltaFreqX is the delta frequency of the ping in **Hz**.
The end frequency can be calculated as EndFreq = StartFreq + DeltaFreq. It is recommended to use approximately 10% of the central frequency as delta frequency for the standard DeepVision transducers. For side scan sonar applications, the settings of the two channels are usually identical.

New settings will take effect when a new recording is started, and will not affect an active recording.

void DSSP_SetSampling(
      unsigned int nSamples,
      bool CH0Active,
      bool CH1Active,
      bool onePing)

Function for controlling the output of the sonar module.

The nSamples sets the number of output samples per ping and side, and thereby also the resolution. The resolution can be calculated as: $resolution = range / nSamples$ where both range and resolution are given in meters. The value of nSamples are typically 500 to 1000.

When CH0Active is true, the sonar module outputs data from channel one, which is normally the left channel on a side scan sonar module.

When CH1Active is true, the sonar module outputs data from channel two, which is normally the right channel on a side scan sonar module.

When onePing is true, the sonar will ping once for each *StartRec(float range)* function sent, if onePing is false the sonar will ping continuously until the *StopRec()* function is sent. In both cases the *StopRec()* function is needed to stop the recording.

New settings will take effect when a new recording is started, and will not affect an active recording.

void StartRec(float range)

Starts a recording with the given range in meters. The range of the recording is limited to a minimum of 10 meters and a maximum of 500 meters. The resolution of the recording is given by the value of *nSamples* in the *DSSP_SetSampling()* function and the range, and can be calculated as: $resolution = range / nSamples$.

When onePing is used each ping has to be initialised by *StartRec()*.

int GetData(char* buffer, int buffersize)

The *GetData* function receives sonar data over TCP and puts it into *buffer*. The *buffersize* argument is the size of *buffer*. The function returns the number of received bytes.

void StopRec()

Stops the active recording and closes the .dvs file.

int   m_type
int   m_model

The two integers *m_type* and *m_model* stores the type of the connected sonar. They can be used to differentiate between different sonars on a system running DHCP.

The CDSSPParser Class

The CDSSPParser class parses streamed data from DeepVision OEM sonars.

Bool Add(char b)
Used to add sonar stream data to the parser. Returns *true* when a complete sonar package has been correctly added.

void GetChannelData(char*& data0, int* size0, char*& data1, int* size1)
Function to retrieve data from the parser after the *bool CDSSPParser::Add(char b)* function has returned true. The data of each channel will be stored in data0 and data1 respectively and the size of each channel in size0 and size1.

The CDVSFileWriter Class

The CDVSFileWriter class is used to create .dvs files from sonar data which can be viewed and edited in the DeepVision DeepView software.

bool Create(const char* fileName,
                    bool left,
                    bool right,
                    float res,
                    int nSamples);
Function to create and define a new .dvs file. The *fileName* argument sets the name and path of the file, the *left* and *right* arguments indicates if the left and right channels of the recording were active. The *res* and *nSamples* arguments corresponds to the resolution and nSamples as described in *StartRec()* of the *EthernetSonarAPI* class (*resolution = range / nSamples*).

void AddPingData(double lat,
                        double lon,
                        float speed,
                        float heading,
                        BYTE* pLeftData,
                        int nLeft,
                        BYTE* pRightData,
                        int nRight)
The *AddPingData* function adds ping data to the file that has been defined by the *Create()* function. If the ping is to be georeferenced the lat, lon, speed and heading arguments need to be supplied. The ping data is added to the *pLeftData* and *pRightData* arguments respectively and *nLeft* and *nRight* are the length of the two buffers in number of bytes.

void CreateDemoFile(const char* fileName)
The *CreateDemoFile* function will create a demo file in the .dvs format and is intended as an example of both how to use the other functions in the class and to show the resulting file format.

F   OSMEthernet Connector

Main Connector

The main connector is a Ampmodu Mod II 2x4-way male pin header connector. Pitch is 2.54 mm. A suitable mating connector is TE Connectivity part number 280365.

The OSM Ethernet Sonar Module can use mixed power and data, thus eliminating the need to for V1 and V2.
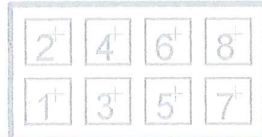
Main Connector

PIN No.	Name	Description	Color
1	RX-	Receive Data-	Green
2	RX+	Receive Data+	Green/white
3	TX-	Transmit Data-	Orange
4	TX+	Transmit Data+	Orange/white
5	V1*	Supply voltage 12 V 500mA. Min 11 V. Max 48 V. Protected to 60V peak.	Blue
6	V1*	Supply voltage 12 V 500mA. Min 11 V. Max 48 V. Protected to 60V peak.	Blue/white
7	V2*	Ground	Brown
8	V2*	Ground	Brown/white

* Pin 5-6 are internally connected and pin 7-8 are internally connected

Connectors

Sonar Transducer Connector

The sonar transducer connector is a Ampmodu Mod II 6-way male pin header connector. Pitch is 2.54 mm. A suitable mating connector is TE Connectivity part number 280360.

1	2	3	4	5	6

Sonar Transducer Connector

1	T1+	Transducer positive channel 1.	Brown
2	T1-	Transducer negative channel 1.	White
3	Gnd1	Ground/Shield channel 1.	Shield
4	Gnd2	Ground/Shield channel 2.	Shield
5	T2-	Transducer negative channel 2.	White
6	T2+	Transducer positive channel 2.	Brown

G   Excerpt from a source code: REST-API

G.1   Startup code

```python
from fastapi import FastAPI
from starlette.middleware.cors import CORSMiddleware


from api.api import api_router
from db.session import engine
from models import setting


setting.Base.metadata.create_all(bind=engine)


app = FastAPI()


origins = [
    "http://localhost:3000",
    "localhost:3000",
]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)
app.include_router(api_router)
```

H Excerpt from a source code: Sonar API

H.1 Startup code

```cpp
#include "EthernetSonarAPI.h"

#include "DVSFileWriter.h"

#include "DSSPParser.h"

#include <fstream>

#include "zmq.h"

#include "windows.h"

#include <iostream>

#include <string>


#define ZMQ_EXPORT __declspec(dllexport)


#define BUFFERSIZE 2048


using namespace std;


int main() {

  std::vector<int> ports;
  std::vector<std::string> IPs;


  CEthernetSonarAPI::FindSonar(&IPs, &ports);


  for (unsigned int i = 0; i < IPs.size(); i++) {
    std::cout << "Sonar with IP " << IPs[i] << " found." << std::endl;
  }
  if (IPs.size() == 0) {
    return -1;
  }
```

```cpp
void* context = zmq_ctx_new();

void* requester = zmq_socket(context, ZMQ_PUB);

int rc = zmq_bind(requester, "tcp://127.0.0.1:5555");


CEthernetSonarAPI Sonar(IPs[0], ports[0]);


CDVSFileWriter DVSFile;
CDSSPParser parser;


float range = 20;                    // Sonar range in meters
int nSamples = 500;                    // Number of samples per active side and ping


int nPeriods = 32;                    // Number of periods of transmitted pulse
float startFreq = 320000;            // Starting frequency of transmitted pulse
float deltaFreq = 40000;             // Delta frequency of transmitted pulse
bool leftActive = true;              // true if left side is to be used
bool rightActive = true;             // true if right side is to be used
float resolution = (float)(range * 1.0) / nSamples; // Resolurion of the
    resulting image in meters


Sonar.DSSP_SetPulseDual(nPeriods, startFreq, deltaFreq, nPeriods, startFreq,
    deltaFreq);


Sonar.DSSP_SetSampling(nSamples, leftActive, rightActive, false);


Sonar.StartRec(range);


char sonarData[BUFFERSIZE];


int receivedSamples = Sonar.GetData(sonarData, BUFFERSIZE);
while (receivedSamples <= 0) {
```

```cpp
    Sonar.StartRec(range);
#ifdef WIN32
    Sleep(50);
#elif LINUX
    usleep(50000);
#endif
    receivedSamples = Sonar.GetData(sonarData, BUFFERSIZE);
#ifdef WIN32
    Sleep(50);
#elif LINUX
    usleep(50000);
#endif
  }

  std::cout << "Sonar connected successfully." << std::endl;

  // Create your 'test' file here
  DVSFile.Create("my_test_file.dvs", leftActive, rightActive, resolution,
      nSamples);

  int pings = 500;
  int n = 0;


  while (pings > 0) {
    receivedSamples = Sonar.GetData(sonarData, BUFFERSIZE);
    if (receivedSamples > 0) {
      for (int i = 0; i < receivedSamples; i++) {
        if (parser.Add(sonarData[i])) {
          char* data0;
          char* data1;
          int size0 = 0;
```

```cpp
        int size1 = 0;

        parser.GetChannelData(data0, &size0, data1, &size1);



        DVSFile.AddPingData(0.0, 0.0, 0.0, 0.0, data0, size0, data1, size1);


        string result;


        for (int i = 0; i < size0; i++) {
          result += to_string((int)data0[i]) + " ";
        }
        for (int i = 0; i < size1; i++) {
          result += to_string((int)data1[i]) + " ";
        }
        const char* valueChar = result.c_str();
        int rcb = zmq_send(requester, valueChar, strlen(valueChar), 0);


        std::cout << "Size0: " << size0 << ", size1: " << size1 << ", n=" << n++
            << std::endl;


        pings--;
      }
    }


  }
 }
 zmq_close(requester);
 zmq_ctx_destroy(context);
 return 0;
}
```

I   Excerpt from a source code: Surface Unit

```python
def calc_checksum(message:str,startchar,endchar)->str:
    start = message.index(startchar)
    stop = message.index(endchar)
    checksum = 0
    for c in message[start:stop]:
            checksum ^= ord(c)
    return hex(checksum)[2:].upper()
```

J   Excerpt from a source code: GUI

J.1   Startup code

```javascript
import React from "react";
import { Switch, Route, HashRouter } from "react-router-dom";

import { SettingsProvider } from "./components/SettingsProvider";
import NotFoundPage from "./pages/NotFoundPage";
import NavIcon from "./components/NavIcon";
import Dashboard from "./pages/Dashboard";
import Settings from "./pages/Settings";
import Home from "./pages/Home";
import Map from "./pages/Map";

function App() {
  return (
    <HashRouter>
      <SettingsProvider>
        <NavIcon />
```

```jsx
      <Switch>

        <Route path="/" exact component={Home} />

        <Route path="/settings" component={Settings} />

        <Route path="/dashboard" component={Dashboard} />

        <Route path="/map" component={Map} />

        <Route component={NotFoundPage} />

      </Switch>

    </SettingsProvider>

  </HashRouter>

 );
}


export default App;
```

K   Excerpt from a source code: Serial from Towed-ROV

```python
import serial

import queue

from threading import Thread

from time import time


ENCODING = 'utf-8'

TERMINATOR = ':'

START_CHAR = '<'

END_CHAR = '>'

NEW_LINE = '\n'

START = b'<'

STOP = b'>'



class SerialWriterReader(Thread):
```

```python
    """
    Serial reader and writer running as a thread to read and write data to and
        from a serial port.
    """
    def __init__(self, output_queue, input_queue, com_port,
                 baud_rate, from_arduino_to_arduino_queue):
        Thread.__init__(self)
        self.from_arduino_to_arduino_queue = from_arduino_to_arduino_queue
        self.output_queue = output_queue
        self.input_queue = input_queue
        self.com_port = com_port
        self.baud_rate = baud_rate
        self.serial_port = serial.Serial(
            port=self.com_port,
            baudrate=self.baud_rate,
            parity=serial.PARITY_NONE,
            stopbits=serial.STOPBITS_ONE,
            bytesize=serial.EIGHTBITS,
            timeout=0)
        self.stop = False
        self.in_packet = bytearray()
        self.packet = bytearray()
        self.last_output = ''
        self.FROM_ARDUINO_TO_ARDUINO = ['depth', 'roll', 'pitch']


    def run(self):...


    def __write_serial_data(self, message):...


    def __read_incoming_data(self):
        """
        Reads from serial port, iterates over each byte to find the start byte "<"
```

```python
            and add each follow byte to a variable of bytes until the byte received
                equal the stop byte ">".
            The bytes will be decoded and a string of received data will be returned.
            :return: message(String) read from serial port
            """
            data_received = []
            message_received = ""
            try:
                data = self.serial_port.read(self.serial_port.in_waiting or 1)
                for byte in serial.iterbytes(data):
                    if byte == START:
                        self.in_packet = True
                    elif byte == STOP:
                        self.in_packet = False
                        data_received.append(self.__handle_packet(bytes(self.packet)))
                        del self.packet[:]
                    elif self.in_packet:
                        self.packet.extend(byte)
                    else:
                        pass

            except (Exception) as e:
                print(e, "serial1")
            return data_received


    def __handle_packet(self, data):
        """

        Decodes data and removes charaters
        @param data: Bytes received for the serial port
        @return: the decoded data as a string.
        """
        message_received = data.decode(ENCODING). \
```

```python
        replace(START_CHAR, ""). \
        replace(END_CHAR, ""). \
        replace(" ", "")
    return message_received


    def stop_thread(self):...
```

L   Excerpt from a source code: SeafloorTracker

```python
"""
Seafloor tracking class
block of x sea floor measurements are sent to class
measurements are sent to a cost function that finds a sp for a distance of 10
    meters
when a new set point is created, the set_point is sent to the optimal path
    algorithm, this algorithm returns index(0) as the new set_point
"""
import numpy as np
from threading import Thread
from time import sleep




class SeafloorTracker(Thread):
    def __init__(self, length_rope, desired_distance, min_dist, dist_to_skip,
                depth_of_rov, depth_beneath_boat, new_set_point_event,
                    set_point_queue):
        Thread.__init__(self)
        self.length_rope = length_rope
        self.desired_distance = desired_distance
        self.min_dist = min_dist
```

```python
    if desired_distance - min_dist >= dist_to_skip:
        self.dist_to_skip = dist_to_skip
    else:
        self.dist_to_skip = desired_distance - min_dist
    self.set_points = np.zeros([round(length_rope / 10)])
    self.depths_of_rov = depth_of_rov
    self.depths_beneath_boat = depth_beneath_boat
    self.new_set_point_event = new_set_point_event
    self.set_point_queue = set_point_queue
    self.array_count = 0


def run(self):
    while True:
        if self.new_set_point_event.is_set():
            depths_beneath_rov = np.array(self.depths_beneath_boat.queue)
            depths_of_rov = self.depths_of_rov
            with self.depths_beneath_boat.mutex:
                self.depths_beneath_boat.queue.clear()
            print("ok")
            self.set_point_queue.put(self.get_set_point(depths_beneath_rov,
                depths_of_rov))
            self.new_set_point_event.clear()


def get_set_point(self, sonar_values, depth_rov):
    """[Get a new set point based on the current set point and future depths]
    Args:
        sonar_values ([float]): [numpy 1D array with recorded sonar values]
        depth_rov ([float]): [The last measured ROV depth]
    Returns:
        [float]: [The new depth set point for the ROV]
    """
    current_set_point = self.set_points[0]
```

```python
        new_set_point, alarm = self.__cost_function(sonar_values)
        self.set_points = np.delete(self.set_points, 0)
        self.set_points = np.append(self.set_points, new_set_point)
        if self.set_points_full:
            self.set_points = self.__find_opt_sp(self.set_points,
                current_set_point, depth_rov, self.desired_distance,
                                        self.min_dist, self.dist_to_skip)
        elif self.array_count >= len(self.set_points) - 1:
            self.set_points_full = True
        else:
            self.array_count = self.array_count + 1
        return self.set_points[0]



    def set_paramter_values(self, length_rope=None, desired_distance=None,
        min_dist=None, dist_to_skip=None):
        """[summary]
        Args:
            length_rope ([int], optional): [The length of the towing cable].
                Defaults to None.
            desired_distance ([int], optional): [The ROVs desired distance from the
                seafloor]. Defaults to None.
            min_dist ([int], optional): [The minimum distance the ROV can have to
                the seafloor]. Defaults to None.
            dist_to_skip ([int], optional): [The distance the ROV can have to the
                new set point before changing]. Defaults to None.
        """
        if length_rope is not None and length_rope != self.length_rope:
            new_size = round(length_rope / 10)
            self.set_points = self.__change_matrix_size(self.set_points,
                len(self.set_points), new_size)
        if desired_distance is not None:
```

```python
        self.desired_distance = desired_distance
    if min_dist is not None and min_dist < self.desired_distance:
        self.min_dist = min_dist
    if dist_to_skip is not None:
        self.dist_to_skip = dist_to_skip
        if desired_distance - min_dist <= self.dist_to_skip:
            self.dist_to_skip = dist_to_skip
        else:
            self.dist_to_skip = desired_distance - min_dist


def __cost_function(self, sonar_values):
    """[Run the echo sounder data through a cost function to find the optimal
        distance]
    Args:
        sonar_values ([float np array 1D]): [The recorded sonar values]
    Returns:
        [float]: [The optimal distance from the seafloor]
    """

    new_sp = 0
    alarm_flag = False
    max_set_point = round(min(sonar_values) - self.min_dist)
    if max_set_point >= 0:
        legal_set_points = np.arange(0, max_set_point, 0.5)
        cost = np.zeros([len(legal_set_points)])
        for index, set_point in enumerate(legal_set_points):
            for sonar_value in sonar_values:
                cost[index] += abs(sonar_value - set_point -
                    self.desired_distance)
        min_cost = np.amin(cost)
        min_cost_idx = np.array(np.argmax(cost == min_cost))
        new_sp = legal_set_points[min_cost_idx]
```

```python
        else:
            alarm_flag = True
        return new_sp, alarm_flag


    def __find_opt_sp(self, set_points, current_sp, depth_rov, desired_distance,
        min_dist, dist_to_skip):
        """[Evaluetes the set points array to find an optimal path]
        Args:
            set_points ([type]): [description]
            current_sp ([type]): [description]
            depth_rov ([type]): [description]
        Returns:
            [type]: [description]
        """
        set_points_mean = round(np.mean(set_points))


        # If the ROV is on colision course every set point is set to a safe
            distance giving it time to go up
        if current_sp - set_points[-1] > desired_distance - min_dist or depth_rov -
            set_points[
            -1] > desired_distance - min_dist:
            set_points_min = min(set_points)
            set_points = np.empty(len(set_points))
            set_points.fill(set_points_min)
        elif current_sp - min(set_points) > desired_distance - min_dist or
            depth_rov - min(
                set_points) > desired_distance - min_dist:
            set_points[0] = min(set_points)
        # Rov closer to mean than current sp
        elif dist_to_skip > abs(depth_rov - set_points_mean) < abs(depth_rov -
            current_sp):
            set_points[0] = set_points_mean
```

```python
        # if the distance between current sp and mean sp is greater than dist to
            ignore
        elif abs(current_sp - set_points_mean) > dist_to_skip:


            # if mean is less than current the ROV goes up
            if set_points_mean < current_sp:
                set_points[0] = set_points_mean
                # if the mean value is greater than current set point, and it wont
                    reduce the depth if it have to go back up.
            elif set_points_mean - set_points[0] < desired_distance - min_dist and
                set_points_mean - min(
                    set_points) < desired_distance - min_dist:
                set_points[0] = set_points_mean
            else:
                set_points[0] = min(set_points)


        else:
            set_points[0] = current_sp
        return set_points


    def __change_matrix_size(self, matrix_to_change, size, new_size):
        difference = new_size - size
        if difference >= 1:
            idx = np.full((1, difference), matrix_to_change[-1])
            new_matrix = np.append(matrix_to_change, idx)
        elif difference <= -1:
            idx = np.arange(abs(difference))
            new_matrix = np.delete(matrix_to_change, idx)
        return new_matrix



if __name__ == "__main__":
```

```
q1 = queue.Queue()

q2 = queue.Queue()

q3 = queue.Queue()

q4 = Queue()

q1.put(15.01)

q1.put(15.1)

q1.put(15.2)


q2.put(30.01)

q2.put(30.9)

q2.put(30.8)

test = SeafloorTracker(300, 20, 9, 6, 10, q2, q3, q4)

test.start()


sleep(3)

q3.put(True)
```

M   Demonstration video

The demonstration video can be viewed in full on YouTube.

https://www.youtube.com/watch?v=K7ho2UsRcDo

N   Source Code

The source code can be viewed in full and is available on Github.

https://github.com/Towed-ROV