Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Datateknologi	Vår 2023 **Åpen** / Konfidensiell
Forfatter(e): Martin Dimmen, Kim Jørgensen, Dominykas Petniunas	
Fagansvarlig: Erlend Tøssebro Veileder(e): Erlend Tøssebro	
Tittel på bacheloroppgaven: Operatørgrensesnitt, topside kommunikasjon for fjernstyrt undervannsfartøy og styring av ROV og Manipulator. Engelsk tittel: Graphical user interface, topside communication for remotely operated underwater vehicle, and control of ROV and Manipulator.	
Studiepoeng: 3 x 20	
Emneord: Graphical User Interface, GUI, UiS Subsea, ROV, Python, Communication	Sidetall: 111 + vedlegg/annet: 1 Stavanger 15. mai 2023

Summary

This bachelor thesis consisted of creating a topside system for a remotely operated vehicle (ROV) that was created by interdisciplinary groups in the student organization UiS Subsea. The ROV was created to satisfy the conditions for the MATE competition, which takes place in Colorado, USA, in early summer.

The topside system required three main implementations to fulfill its purpose, the first being an established connection between the topside and subside using UDP and TCP, the second was adding maneuverability to the ROV and Manipulator with the use of controllers, and the third being the graphical user interface (GUI) to present all the relevant information from the ROV.

The system was modular, which meant it was easy to change and implement functionalities. It was based on last year's topside system but has been changed significantly because of other needs and requirements. It had a good structure, with well-functioning classes.

The project was broken down into sub-tasks to simplify the implementation process and increase efficiency. This approach resulted in the total final product - the topside system. The ROV was controllable, the GUI displayed the necessary information, and the data flow worked well.

The system will be further improved toward the MATE competition, which will not be included in the bachelor as the changes will be made after the bachelor thesis deadline.

Link to the topside system on GitHub:

https://github.com/UiS-Subsea/Bachelor_GUI

Preface

This thesis concludes the final part of the Computer Technology bachelor's degree at the University of Stavanger. We would like to take this opportunity to express our sincere gratitude to all the members at UiS Subsea for their valuable collaboration and support throughout the semester.

We would also like to extend our gratitude to the sponsors of UiS Subsea, the University of Stavanger for their support in facilitating student organizations

We would also like to give a special thanks to our supervisor Erlend Tøssebro for meetings and advice.

Thanks to Stian Myklebust Eiksund for creating our alarm sound.

Abbreviations & Expressions

List of abbreviations or expressions and their accompanying explanations.

Abbreviation - The entire word - Explanation

- **ROV** - Remotely Operated Underwater Vehicle - An unmanned robot able to perform desired tasks
- **GUI** - Graphical User Interface
- **TCP** - Transmission Control Protocol - Communication protocol that ensures stable data transportation. [2.1.2]
- **UDP** - User Datagram Protocol - Communication protocol that ensures fast but unreliable data transportation. [2.1.3]
- **XML** - Extensible Markup Language
- **UI** - User Interface File, XML file created in QT Designer
- **UIC** - User Interface Compiler, module used to convert UI files into Python code
- **Jetson** - Nvidia Jetson - AI computing platform for edge devices with compact hardware modules and real-time AI capabilities. [2.2.3]
- **AI** - Artificial Intelligence

Expression - Explanation

- **Bitarray** - Data structure array that store bits
- **Bytearray** - Data structure array that store bytes
- **Manipulator** - Controllable arm connected to the ROV
- **Topside** - The part of the system above water; also known as the computer
- **Subside** - The part of the system below water; also referred to as the ROV
- **Qwidget** - The QWidget class is the base class of all user interface objects
- **Demultiplexes** - Separating a combined audio or video stream into separate streams for each component.

Contents

Summary **i**

Preface **ii**

Abbreviations & Expressions **iv**

1 Introduction **1**

1.1 Introduction 1

1.1.1 About UiS SubSea 1

1.1.2 Mechanical 3

1.1.3 Electronics 4

1.1.4 Computer Science 6

1.1.5 MATE - Marine Advanced Technology Education . . 7

1.1.6 About ROV Project 19

1.2 GUI - A closer look 25

1.2.1 Overview of what needs to be done 26

2 Background and Theory 27

2.1 Technologies and Protocols 27

2.1.1 Communication 27

2.1.2 Transmission Control Protocol 29

2.1.3 User Datagram Protocol 30

2.1.4 TCP versus UDP 30

2.1.5 Communication Medium 31

2.2 Camera 32

2.2.1 Camera Positioning 33

2.2.2 Camera Lights 33

2.2.3 Remote Connection 34

2.2.4 GStreamer 35

2.3 Development and Working Method 36

2.3.1 Git 36

2.3.2 Github project table 37

2.3.3 Development strategy 37

2.3.4 Meetings 38

2.4 Programming Language 38

2.4.1 Earlier languages 39
2.4.2 Python 39
2.4.3 C# 40
2.4.4 Choosing a language 40
2.5 Frameworks 41
2.5.1 Qt 41
2.5.2 PyQt 41
2.5.3 Pygame 46
2.6 Steering 47
2.6.1 Choice of Controller 47
2.6.2 Testing 48
2.6.3 ROV 49
2.6.4 Manipulator 50
2.7 Operating System 50
2.7.1 Microsoft Windows 11 51
2.7.2 Ubuntu 51
2.7.3 Choosing an operating system 52
2.8 Processing and Threading 52
2.8.1 Process 52
2.8.2 Thread 53

3 Implementation **54**

3.1 Process flow 54

3.2 Data transfer between topside and Nvidia Jetson 59

3.2.1 Control System Circuit Board 61

3.2.2 Sensor Circuit Board 62

3.2.3 Power Circuit Board 63

3.2.4 Communication Circuit Board 64

3.3 Controllers 66

3.3.1 Accessing the controller data 66

3.3.2 Steering data 68

3.3.3 Sending the steering data 71

3.4 Threads and Processes with Threadwatcher 72

3.5 GUI 73

3.5.1 Layout 73

3.5.2 GStreamer 78

3.5.3 Sending data to the GUI 79

3.5.4 Updating the GUI 81

3.5.5 Sending data from GUI 81

4 Testing **82**

4.1 Local testing 82

4.2 ROV testing 84

4.3 Communication testing 85

5 Results **86**

5.1 Water Test 86

5.2 Topside 87

5.2.1 GUI 88

5.2.2 Controllers 88

5.2.3 Data Logging 89

6 Discussion **90**

6.1 Bottleneck 90

6.2 Communication Challenges & Solutions 90

6.3 Controller Challenges & Solutions 91

6.3.1 Mac and Windows to Ubuntu 91

6.3.2 Initializing Controllers 93

6.3.3 Future improvements for steering data 94

6.4 GUI Challenges & Solutions 95

6.4.1 Iterations of the GUI 95

6.4.2 ROV data to GUI 96

6.4.3 Updating Alarms 97
6.5 General Challenges and Solutions 98
6.5.1 Earlier testing/implementation 98
6.5.2 Subside PC to do image processing 98
6.5.3 Time usage 98
6.6 Continuation of the Subsea project 99
6.7 Summarized Future Improvements 99

7 Financial overview & Environmental report 101

7.1 Environmental Report 102
7.1.1 Environmental impact 102

8 Conclusion 104

Attachments 111

A 112

Chapter 1

Introduction

1.1 Introduction

This thesis is part of a larger project of designing and developing a functioning ROV and scientific float for the student organization UiS Subsea and competing in the TAC and MATE challenge in June 2023. This chapter presents the premise of the UiS Subsea organization and the different tasks of every team involved in the project. It also aims to provide sufficient information about the thesis objective, scope, and limitations. Next, the MATE competition and its specific tasks, challenges, and scoring system. It is also an introduction to the background information about the ROV and floats, as well as information about the main task of the thesis.

The introduction chapter is written by the technical leaders of UiS Subsea and is a common chapter for all of the groups.

1.1.1 About UiS SubSea

UiS Subsea is a student organization at the University of Stavanger, engaging students in underwater technology since 2013. The primary goal of this organization is to provide students with the experience of working in a team consisting of different engineering disciplines.

This year, at UiS Subsea, nine bachelor groups are working together to design and build a complete underwater ROV (remotely operated underwater vehicle). This year's ambitions are to build upon and improve from last year's ROV project. The goal is to create a new ROV that is easier to maintain, has more efficient cooling, and can upgrade the software to an AUV (autonomous underwater vehicle). Though it will not be completely autonomous, it should be able to compete in the autonomous challenges from MATE.

There are two groups of mechanical engineering bachelor students on the team. Their responsibility is to design, craft, and construct the chassis, manipulator, and electronic enclosure for the ROV.

Additionally, five groups of electrical engineering students are responsible for various tasks. Their responsibilities include working on sensors, regulation systems, communication, power, and circuits. They are also in charge of establishing communication between the ROV and the topside system.

On the topside, there are two groups of computer engineering students. Their primary responsibilities include sending and receiving commands and data. They also work on displaying the ROV in a graphical user interface (GUI), handling controls, and performing image processing tasks.

Through several years, Subsea has built ROVs and partaken in international competitions, this year: the MATE ROV competition. This provides a basis for more advanced problem-solving and teamwork to create a positive and healthy environment for learning and developing technical skills. UiS Subsea opens up the opportunity for students to collaborate with industrial companies. Several companies are interested in these projects, providing components and other resources through sponsorship deals. To further improve the relations between the organization and the industry, Subsea annually holds an event called 'Subsea dagen', where companies from the sector can participate with their stand and promote themselves. UiS Subsea and the companies gener-

Figure 1.1: UiS Subsea logo

ate a lot of exposure from such an event.

In previous years the organization has suffered from a lack of continuity due to the participants writing their bachelor's, and a handover between current and new bachelor students has been non-existent. This year, the previous leader and second leader of Subsea decided to stay one more year to help guide the students and the overall course of the project. This reduces the learning curve for the new students while also implementing a stream of previous knowledge and experiences to the future project.

This year's board consists of the following roles and leaders:

- **Project manager:** Joar Rodrigues de Miranda
- **Second Project manager:** Thomas Matre
- **Technical leader Electro:** Jesper A. Flatheim
- **Technical leader Computer:** Filip Sølvberg Herrera
- **Technical leader Mechanical:** Haakon Aleksander Schei

1.1.2 Mechanical

Design of ROV frame, electronic enclosure, and Float

Following the product development process, this thesis aims to design and build the frame of the ROV, the electronic housing, and the shell of the float. The primary focus of the design is to mesh together all the individual parts into one functioning ROV while also ensuring the ROV can perform the tasks in the MATE- and TAC ROV competitions.

In light of the present environmental challenges, a secondary focus will be sustainability, and recyclable materials, to minimize the environmental impact of this process. Effective use of DFE can also help reduce cost and production time while increasing product quality. Material choice, structural, flow, FEM, and buoyancy analysis are tasks to be solved here.

Manipulator

The main task is to develop and design a functional manipulator for tasks the ROV completes. The goal is for the mechanical arm to be useful for MATE requirements while being uncomplicated enough for production and maintenance. Creativity and problem-oriented solutions are necessary to complete this task. Deciding degrees of freedom, which mechanical principles to implement, and material choice are some problems that need to be solved.

In addition, cooperation with electrical engineers is essential regarding manipulator compatibility with the rest of the electrical system. Stress, bending, and shear analysis are fundamental for the success of this task.

1.1.3 Electronics

Power module

The power module's primary function is to regulate and distribute the input voltage provided by the topside system. Its role also includes protecting the components from overload and preventing short circuits. Given that the input voltage is 48V, the system must lower the voltage to ensure that each element receives the appropriate amount of power.

Communication ROV

The communication group's main task is to create a standard system so that electrical circuit boards are connected and can communicate with the rest of the system. By utilizing CAN-bus in addition to C-code, the system can efficiently convey signals and commands between each other. In addition, communication between ROV and the topside system has to be solved. Here control signals from topside to ROV, process data, and video feed from ROV to topside need to be processed efficiently with minimal delay.

In addition, the internal design of the electronic enclosure is this group's responsibility.

Regulation system (Control systems)

The primary objective is to design a navigation and regulation system for the ROV. The system's core components include selecting the appropriate thruster configuration and manipulator motor and developing a circuit board in collaboration with the mechanical groups. These components ensure the ROV's physical limitations and characteristics align with its interactions within the environment.

The circuit board will serve as an interface between the motor controller and other circuit boards. The choice of thrusters and motors must comply with competition standards. Additionally, a significant responsibility of this group is to develop a control system. This system will interpret commands and sensor measurements from the topside and other circuit boards, enabling the ROV to operate and maneuver effectively. Ensuring stability, maintaining orientation, and achieving a desired depth position are vital tasks that need to be addressed by the regulation aspect of the system.

Mathematical models and functions will be required to digitally replicate its behavior to accurately simulate the various degrees of freedom of the ROV.

Sensor system:

The main task of the sensor system is to maintain and disperse information from the different sensors and act upon the vital data. These are orientation(IMU), leak, and pressure sensor. IMU retrieves angle data and axis relation and generates data the regulation system uses to control and drive the thrusters. The Leak sensors consist of 3-4 leak probes placed along the inside of the electronics enclosure to detect eventual leaks and aptly react to the information to minimize damage to critical electrical components. The temperature sensor is divided into three identical sensors along the enclosure at critical points. This is to monitor internal temperature and the dispersion of it. The pressure sensor is the only one externally mounted, retrieving changes in water pressure due to depth changes.

FLOAT:

The float is the only component not attached to the main ROV. It is essentially its own AUV, with a pre-programmed flight path and power supply. It gathers vital information about ocean health and the underwater environment.

The competition requires that the float completes two vertical profiles: Sink to the bottom and return to the surface. Afterward, it has to relay the completion time with a ping to the topside system, temperature, and pressure. This will be displayed for the operator to view.

It operates with a buoyancy engine to complete this path.

1.1.4 Computer Science

Image Processing:

The main task of the Image processing group is to complete a subtask for the MATE ROV competition, which is based on processing image data to solve several tasks. These require camera vision with depth perception and autonomous programming. The tasks are the following:

- **Autonomous Docking**
- **3D Modelling of sick coral**

- **Count frogs along a transect line**
- **Monitor/analyze of seaweed growth**

GUI:

The primary objective is to develop a monitoring and control system for the ROV (Remotely Operated Vehicle). To achieve this, we need to implement a system that enables the transmission of commands and control data from the topside to the ROV. This task requires close collaboration with the communication group.

Furthermore, we aim to present all relevant information and video feeds to the user in real-time through a custom graphical user interface (GUI). It is crucial to create a user-friendly GUI that includes control commands, as these aspects significantly contribute to the overall quality and usability of the product.

1.1.5 MATE - Marine Advanced Technology Education

The primary goal of UiS Subsea for this project is to develop competitive products that can participate in and succeed during the MATE competition in June 2023. To achieve this objective, it is crucial to understand the nature and requirements of the MATE challenge thoroughly. This involves identifying the various challenges involved and comprehending the scoring system so that we can optimize our products to score as many points as possible. We will gather the necessary information from the competition's website and manual to ensure the design of satisfactory products. The following information has been obtained from the organizations' websites. [1] [2]

The ROV created by UiS subsea this year follows the specification determined by the international competition, *MATE ROV COMPETITION*. This competition is hosted by organization *MATE.The Marine Advanced Technology Education (MATE) Center* is a partnership av a multitude of American organizations, established in 1997. These partners mainly comprise schools, research institutes, governments, and Marine institutes. This cooperation's primary goal is to improve marine technical education, strengthening the future American workforce for maritime operations.

Figure 1.2: MATE logo

In 2021, *MATE* transferred the responsibility for student activity over to *Marine Advanced Technology Education for Inspiration and Innovation's*, otherwise known as *MATE II*. Their main objective is to motivate students' interest in maritime knowledge, mainly by hosting *MATE* competition every year. They challenge students to implement engineering principles and expertise to solve subsea tasks. UiS Subsea is competing in the EXPLORER class, reserved for students with Higher technical educational backgrounds.

Figure 1.3: MATE II logo

MATE ROV Competition

The information about the competition is retrieved from the competition manual [3].

This year's competition themes are no different from the previous two, highlighting the importance of the United Nations Decade of Ocean Science for Sustainable Development (2021-2030). Their innovation is to increase ocean knowledge and ensure that society implements this knowledge, thus contributing to the UN's Sustainability goals. This year's task is to create an ROV and a scientific float. This year's themes are the facilitation and production of clean energy, surveillance, and tracking of the ocean's biological diversity.

Figure 1.4: MATE Competition logo

Points

Table [1.1] demonstrates the available points' segmentation. Product demonstration is the first part, where UiS Subsea will solve three practical tasks within 15 minutes. If this is achieved, additional points are given, 1 per minute and 0.01 per second. Extra points are given for ROVs below 25 kg and good teamwork under the competition. These practical tasks are meant to test the operational characteristics of the ROV. The secondary segment points are designated for the technical documentation and how the organization portrays itself. The final points are given based on the safety of the ROV and how the relevant dangers have been adequately analyzed and addressed.

Table 1.1: Points structure

Product demonstrations	
Tasks	300 points
Time bonus	10 points
Weight restrictions	10 points
Organization efficiency	10 points
Engineering and communication	
Technical documentation	100 points
Product presentation	100 points
Marketing	50 points
Company specification sheet	20 points
Company responsibility	20 points
Safety	
Review of safety documentation	20 points
Safety inspection	30 points
Safety job analysis	10 points
Total	**680 points**

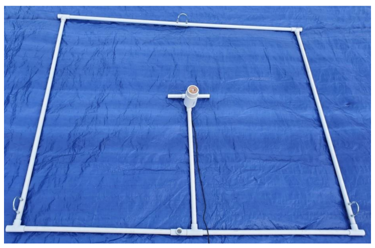
Task 1: (Maritime, renewable energy)

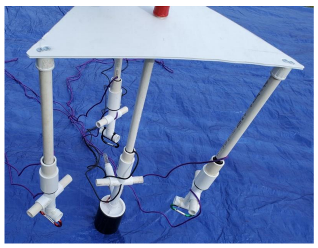
UNs Sustainability goals:
# 7 Clean energy for all
# 12 Responsible consumption and production
The first task is designed to simulate an offshore installation of floating solar panels in an established floating wind farm, removal of biofouling, and

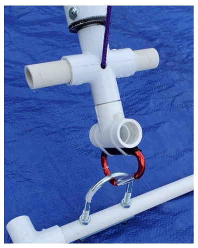
piloting the ROV either Autonomously or manually into an underwater docking station

1.1: Installation of a collection of floating solar panels

- Maneuver the solar panels between 3 existing wind turbines: 10 points.
- Moor 3 moorings to the solar panels: 15 points.
- Remove the lid from power port entry: 5 points
- Connect plug from solar panels: 10 points.


(a) Seabed anchor for installation


(b) Floating solar panel


(c) Hook for mooring

1.2: Remove biofouling from the floating wind turbines

biofouling is simulated either with red PVC pipes connected with Velcro or chenille pipe cleaners twisted together.

- Remove 1-2 units av biofouling: 5 points
- Remove 3-5 units av biofouling: 10 points
- Remove 6 units av biofouling: 15 points

(a) Biological material on structure

(b) Biological material on rope

1.3: Maneuver the ROV into docking station

To be allowed to compete in the competition, the ROV has to fit inside the docking station, with extra points given for the automation of the docking.

- Maneuver autonomously into docking station: 15 points
- Maneuver manually into docking station: 10 points

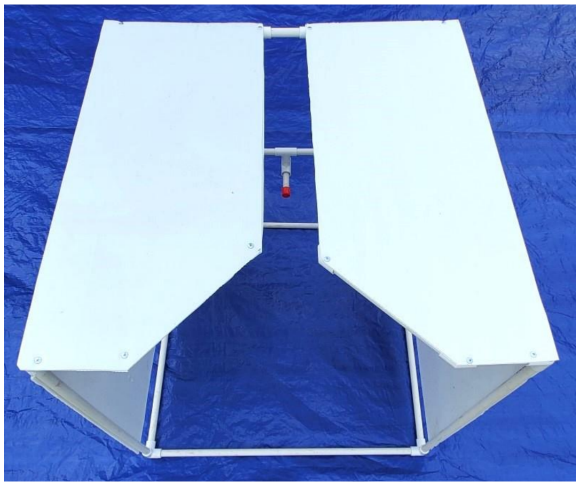


Figure 1.7: ROV Docking station

.

Task 2A: Coral reef and blue carbon

UNs Sustainability goals:
# 13 Stop climate change
# 14 Ocean life
The second task is divided into two parts: 2A and 2B. Part A represents scientific tasks: scanning a coral reef, identifying organisms by utilizing eDNA, exposure to UV light on sick coral reefs, inspection, and installing an environmental mooring system to protect seaweed on the seabed.

2A.1: Measure, model, and identification of disease on coral reef

- Measure diameter on a coral reef: 5 points
- Measure height on a coral reef: 5 points
- Measure area of infection: 5 points
- Make a 3D autonomously: 15 points
- Make a 3D model in CAD manually: 5 points

All measurements must be done within 2cm and can be completed autonomously or manually with reference objects.

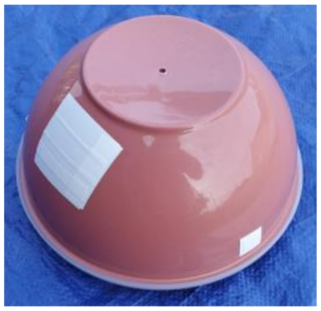
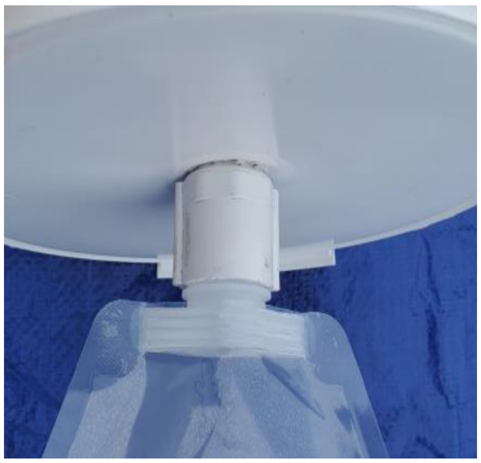
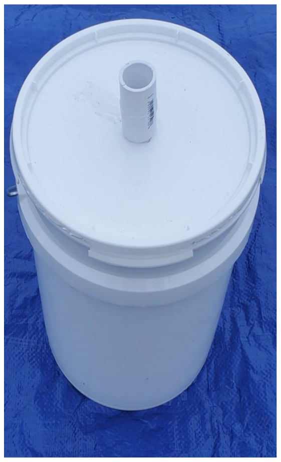


Figure 1.8: Main coral with white spots
.

2A.2: Identify coral reef organisms with eDNA

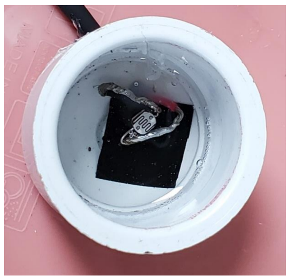
- Extract water sample from the bottle: 10 points
- Identify fish species based upon three samples provided by hosts: 5 points

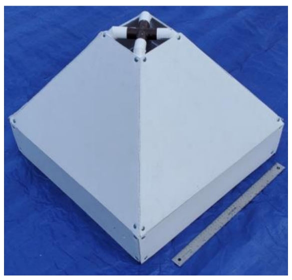

(a) Water bag connection

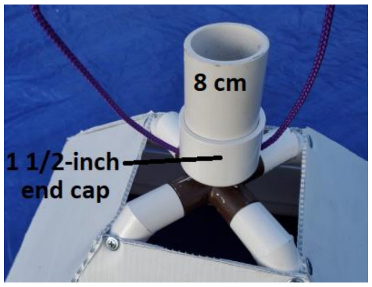

(b) Connection for sample extraction

2A.3: Administrate Rx to infected coral

- Position UV-light above infected area: 5 points
- Activate light and cure: 5 points
- Place tent above coral reef: 10 points
- Place syringe in tent opening: 5 points
- Remove syringe contents inside the tent: 5 points


(a) Photo resistor

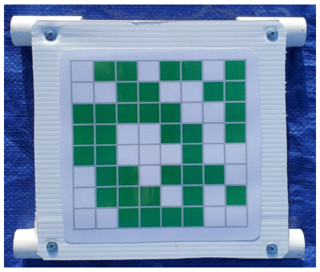

(b) Tent

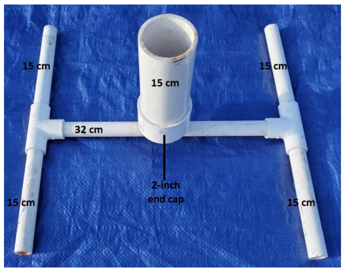


(c) Syringe connection

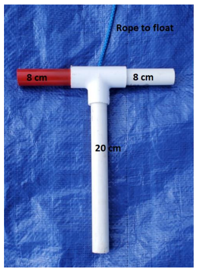
2A.4: Seaweed habitat protection and surveillance

- Identify if seaweed habitat has been rehabilitated, remained unchanged, or worsened, based upon images: 5 points
- Install Eco-Mooring system on the seabed, inside a base, and rotate mooring 720°in the base: 10 points


(a) Seaweed



(b) Eco-Mooring base



(c) Mooring

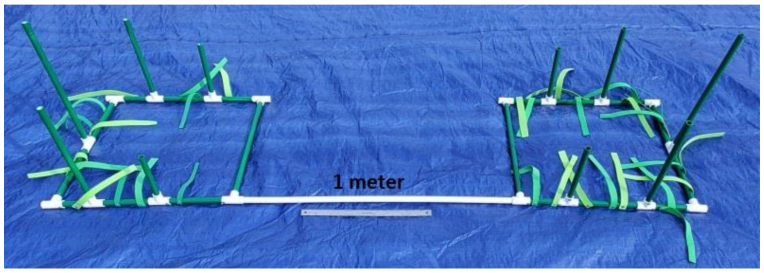
1.1 Introduction

Task 2B: Lakes and rivers

UNs Sustainability goals:
# 13 Stop climate change
# 14 Ocean life
Task 2B primarily focuses on working with freshwater bodies. The objectives of this task are to locate fish, assess whether they are invasive species, and release fry in safe areas. Additionally, there is a requirement to inspect ropes, remove larger objects, follow a designated transect line, count frogs, and install an underwater camera.

2B.1: Re-introduce endangered species of Northern Redbelly Dace fry

- Survey 2 areas, and identify which is safe to place to release the fry: 10 points
- Acclimatize fry to a safe area: 5 points
- Release fry in the safe area: 10 points



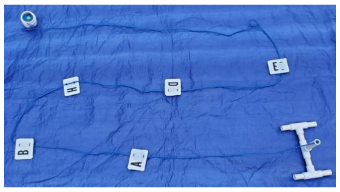
(a) Habitat area

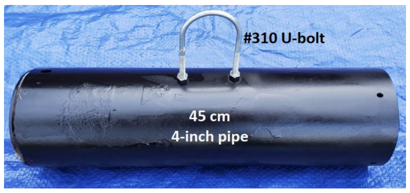
(b) Fry

(c) Current fish species

2B.2: Ensure the health and safety of the Dillion reserve

- Inspect rope for a buoy and display the ten letters attached: 10 points
- Display the documentation of the ROVs lifting capacity: 5 points
- Lift object a maximum of 120 Newton above water: 10 points
- Return object to land: 5 points


(a) Rope with letters

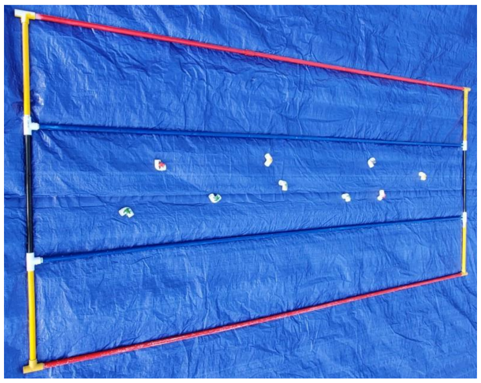


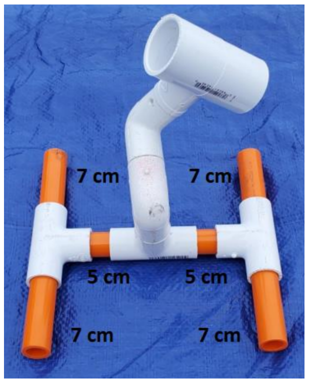
(b) Heavy object which is to be removed

2B.3: Surveillance of endangered Lake Titicaca frogs

- Fly a transect line and maintain the image within the area: 10 points
- Count the number of frogs within an area: 5 points
- Install a camera on a designated area: 5 points

1.1 Introduction


(a) The transect line area that the ROV will fly over



(b) The camera which is to be installed

Task 3: (MATE Floats!)

UNs Sustainability goals:
# 13 Stop climate change
This task represents the construction of a functioning scientific float, which will transmit data when it reaches the ocean surface.[4]

3.1: MATE Floats! 2023

- Design and construct a functioning vertical profile float: 5 points
- Float communicates with land before submersion: 10 points
- Float transmits to land the time of completion after first vertical profile: 10 points
- Vertical profile 2, the float sinks and rises after impact with seabed: 10 points
- Float transmits to land the time of completion after second vertical profile: 10 points
- Vertical profile 1, the float sinks and rises after impact with seabed: 10 points

Restrictions and demands

The competition has certain physical restrictions with size, weight, the operational environment, and electrical limitations. The only vehicle to be utilized is an ROV.

Environment:
The ROV shall be able to operate in fresh, salt, or chloride water in a temperature span of 15 to 30 °Celsius.

Materials:
The ROV shall be able to operate at a minimum four meters depth while being under a maximum of 35 kg.

Tether length:
The tether has to be long enough to operate within an area 10 meters from the edge and 4 meters deep. The topside control system can be up to 3 meters from the pool's edge.

Thrusters:
The thrusters shall be adequately protected and meet IP-20 standards. The thrusters shall be designed to operate underwater.

Electrical:
The organizer provides a power supply of 30A and 48 VDC for ROV. Conversion to lower voltages has to happen inside the ROV. Overload protection on 150% of nominal power usage on the ROV shall also be implemented.

Float:
Batteries utilized shall be of type: AAA, AA, A, A23, C, D, or 9V alkaline batteries. The float shall be protected with a 7.5A fuse. There must be a pressure relief valve with a minimum diameter of 2.5cm.

1.1.6 About ROV Project

ROV history

The creation of the first ROV can be credited to Dimitry Rebikoff, with his invention shown below. [5] The aptly named Poodle was made in 1953, complete with a tethered connection and operated with a topside control panel.

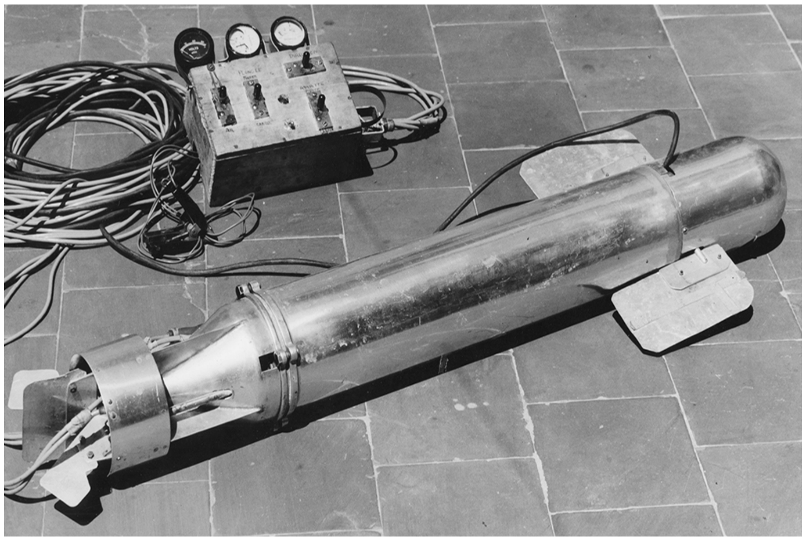


Figure 1.15: The world's first ROV

The subsea sector has come a long way since then, and modern ROVs are quite different. In the 1960s, the U.S. NAVY utilized ROVs as recovery drones for underwater equipment. Within 20 years, there were over 500 ROVs worldwide, mostly in the commercial market, each designed with a specific task and purpose. There are a couple of common components of standard ROVs, being: [5]

1. Thrusters
2. Tether
3. Camera
4. Lights
5. Frame
6. Pilot controls
7. Buoyancy element

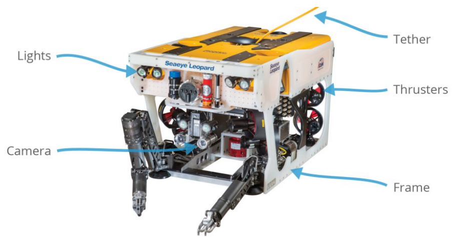


Figure 1.16: Diagram of common components

Newer ROVs are designed based on the given task; some can be: observation, high-speed survey, inspection, trenching, burial, intervention, and construction. Some ROVs can be used for many tasks, while others are limited in design. There are seven main classes of ROVs, from I to VII [6]:

1. Pure observation [7]
2. Observation with payload option [8]
3. Work class vehicles [9]
4. Seabed-Working vehicles [10]
5. Prototype or development vehicles
6. Autonomous underwater vehicles (AUV) [11]
7. High-Speed survey vehicles[12]

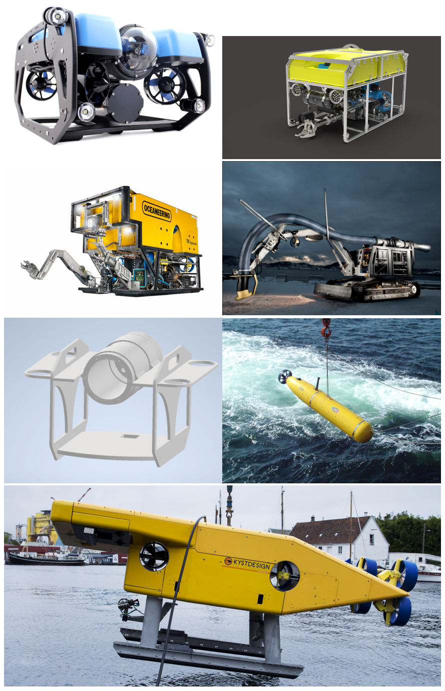


Figure 1.17: The different ROV classes

There are several benefits and limitations to each class. Class I: Pure observation vehicles are physically limited to video observation however highly maneuverable. Generally, they are small vehicles fitted with video cameras, lights, and thrusters. They cannot undertake any other task without considerable modification.

Class II – Observation with payload option has the same capabilities as a pure observation ROV, but usually with additional functionality, such as manipulator, color cameras, other cameras, sonar, and cathodic protection measurement system.

Class III: Work class vehicles are large enough to carry additional sensors and manipulators. They have semi-autonomous capabilities, also known as multiplexing capability, which allows heavier equipment to run without being *hardwired* through an umbilical system. Furthermore, they have enough stability and buoyancy to carry additional detachable equipment without losing functionality. This class is larger than the previously mentioned, with three sub-classes based on power rating:

1. Class III A – Work class vehicles $<$ 100 Hp
2. Class III B – Work class vehicles 100 Hp to 150 Hp
3. Class III C – Work class vehicles $>$150 Hp

Class IV: Seabed working vehicles are utilized, as the name suggests. They maneuver on the seabed by a wheel or belt traction system, thruster propellers, water jet power, or a combination. These vehicles are usually even larger than Class III, with their main purpose being subsea work: dredging, mining, cable and pipeline trenching, excavation, and other subsea construction work.

Class V: Prototype or development vehicles are classified as under development or have not been sufficiently tested. Most special-purpose vehicles or one-off prototypes end up here since any of the previous classes cannot categorize them. Class VI and VII are in this class according to the Norwegian standard of ROVs since they are still under development and only a select few companies produce these vehicles.

YME

This year, the ROV developed and produced a class 2 vehicle with 6 degrees of freedom and a compact design. The goal is to create a vehicle based upon well thought existing solutions while following UN Sustainability goals.

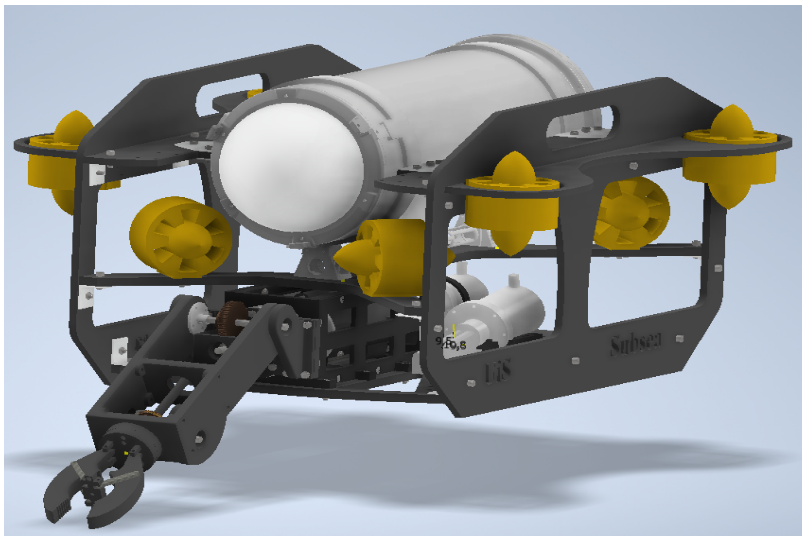


Figure 1.18: 3D model of ROV

YME is controlled by a customized GUI, which communicates via an umbilical cord to the ROV. The project's primary goal is to compete at the MATE ROV competition; thus, the vehicle is designed for this purpose. Additionally, the project participants have set a personal goal to be able to operate at 100m depth. The design is modular, something that makes parts easily replaceable and further developed, both as an ROV and AUV (Autonomous Underwater Vehicle)

Float

Scientific floats have been utilized for a long time, ever since Henry Melson Stommel came up with the idea back in 1955.[13] The main goal of a float was to track and monitor deep drift currents, and the first was manufactured of aluminum with a depth rating of 4500m. Using a buoyancy engine, it changes its depth to pre-programmed heights. A biochemical float uses an array of optical and chemical sensors to gather valuable data at otherwise difficult locations.[14]

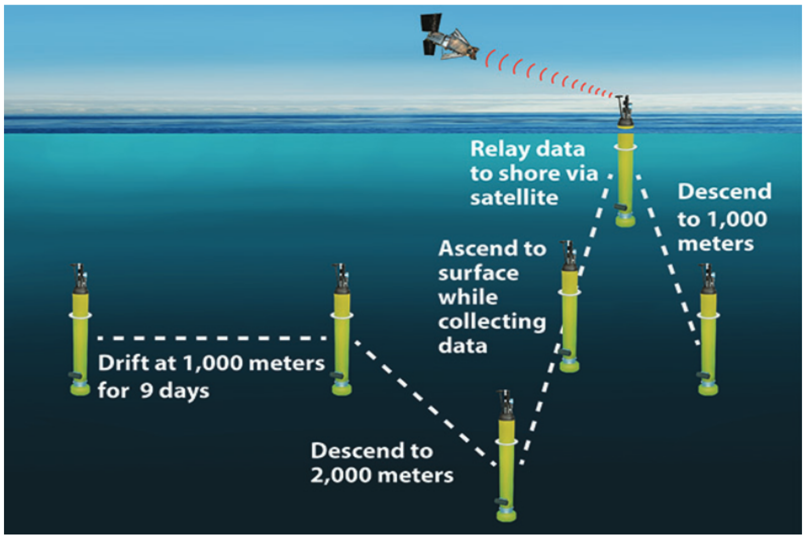

Relay data to shore via satellite
Descend to 1,000 meters
Ascend to surface while collecting data
Drift at 1,000 meters for 9 days
Descend to 2,000 meters

Figure 1.19: Diagram of a float cycle [15]

Figure [1.19] Shows a normal float cycle. Firstly, they descend to a depth of 1000m, drifting for 5-10 days while acquiring valuable data. This is repeated at 2000m and finally ascends to the surface for data transmission. An average float will have a life cycle of about five years once the battery is depleted.

1.2 GUI - A closer look

As stated in Chapter 1.1.4, the main task is to develop a system that monitors and controls the ROV. This involves several sub-tasks to be completed for the main objective to be accomplished. Here is a list of what sub-tasks that need to be done to fulfill the requirements of the main task:

- Establish communication between topside and subside
- Gather steering data from controllers and send this data to subside
- Create a graphical user interface to view sensor data, video feed, and display desired buttons with accompanying functionality

The first sub-task concerns figuring out how to send and receive packets and decide how these packets should look. This requires cooperation with the **Communication** group.

After establishing how the data should be sent and received, steering data can be gathered and built into packets. What functionality the ROV and the Manipulator should have depends on the input from the **Control Systems** group, or rather what UiS Subsea together decides the functionalities should be.

The final part of the main task is to make the GUI itself. This will be used while steering the ROV and is very important to show the necessary information for the pilots operating the ROV and the Manipulator.

The topside system is based on last year's topside implementation but has had several major changes and implementations because of different needs, requirements, preferences, and specifications.

1.2.1 Overview of what needs to be done

Here is a list of planned features to be added to our system:

- Communication
  - Send data from subside to the topside
  - Send data from topside to the ROV
  - Send steering data to the ROV
  - Add logger that oversees every packet sent
- Controllers
  - Gather steering data from controllers
  - Implement steering for the ROV and the Manipulator
  - Make steering effective, user-friendly, and intuitive
  - Drive manually or automatically
- GUI
  - Show important information such as:
    - Temperature of the ROV
    - Temperature of the surroundings
    - Error messages and alarms
    - Thruster throttling values
    - Current and voltage values
  - Turn on/off lights
  - Display camerafeed
  - Turn cameras

Chapter 2

Background and Theory

2.1 Technologies and Protocols

The communication between the topside and the ROV is critical in underwater operations. Underwater operations such as offshore drilling, marine exploration, and research require effective communication between the topside and the ROV. The technologies and protocols used to facilitate this communication are designed to ensure reliable, fast, and secure transmission of data. In this section, we will look at what kinds of assessments were done when choosing the software, the programming language, the joystick, the protocols, and the way communication between the topside and the ROV takes place.

2.1.1 Communication

As part of the bachelor project, establishing a reliable communication channel between the topside and the ROV required an understanding of the necessary steps and protocols

The Open Systems Interconnection (OSI) model is a conceptual model which describes the logical construction of a communication network. The model is divided into seven layers, where each had its own function. The layers are Layer 1-Physical, Layer 2-Data Link, Layer 3-Network, Layer 4-Transport, Layer 5-Session, Layer 6-Presentation, and Layer 7-Application. A more detailed description of each layer and corresponding protocol can be seen in Figure [2.1] beneath. [16]

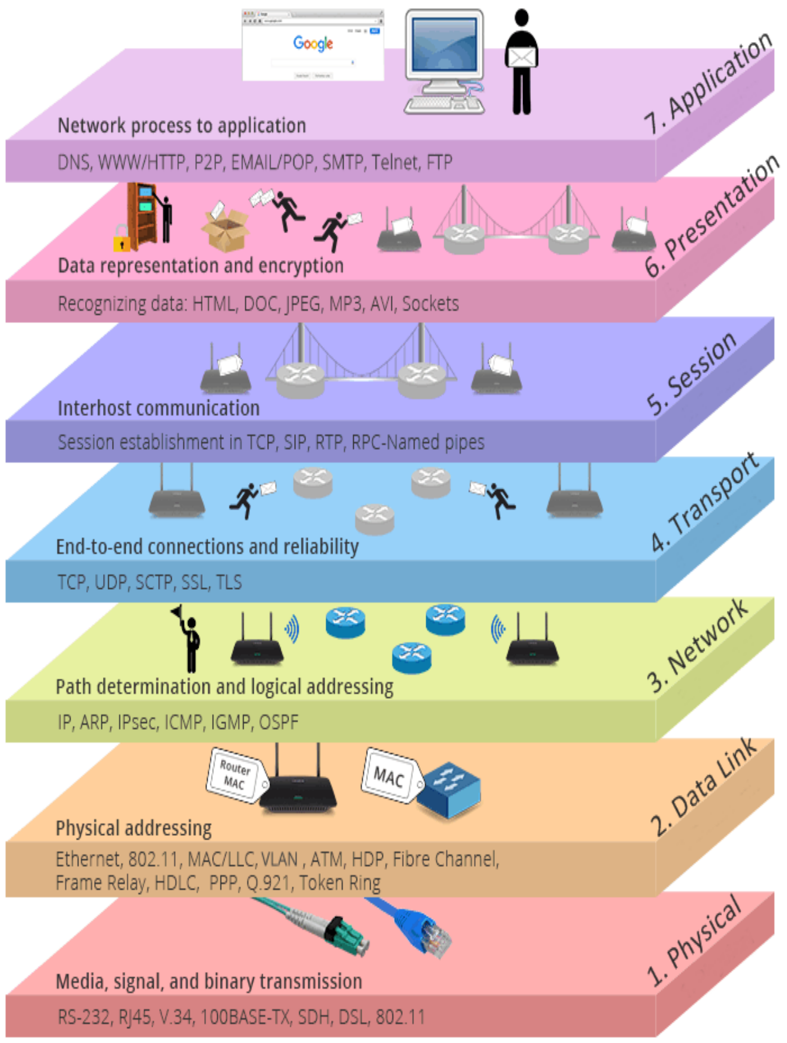

7. Application
Network process to application
DNS, WWW/HTTP, P2P, EMAIL/POP, SMTP, Telnet, FTP
6. Presentation
Data representation and encryption
Recognizing data: HTML, DOC, JPEG, MP3, AVI, Sockets
5. Session
Interhost communication
Session establishment in TCP, SIP, RTP, RPC-Named pipes
4. Transport
End-to-end connections and reliability
TCP, UDP, SCTP, SSL, TLS
3. Network
Path determination and logical addressing
IP, ARP, IPsec, ICMP, IGMP, OSPF
2. Data Link
Physical addressing
Router MAC
MAC
Ethernet, 802.11, MAC/LLC, VLAN , ATM, HDP, Fibre Channel,
Frame Relay, HDLC, PPP, Q.921, Token Ring
1. Physical
Media, signal, and binary transmission
RS-232, RJ45, V.34, 100BASE-TX, SDH, DSL, 802.11

Figure 2.1: OSI layers and corresponding protocols. [16]

As seen in Figure [2.1], Layer 4-Transport, is responsible for end-to-end connections and reliability, meaning that it's one of the layers that will have the biggest impact. [16]

2.1.2 Transmission Control Protocol

TCP is one of the most used protocols in the transport layer. Its main task is to ensure reliable data transmission between end-to-end connections. TCP breaks segments from layer 5, session, into packets and sends them over the network. TCP follows the client-server model to establish the connection. [17]

For reference, the client-server model is a commonly used architecture in computer networks. The client requests a resource from a server, which provides those asked resources. [18]

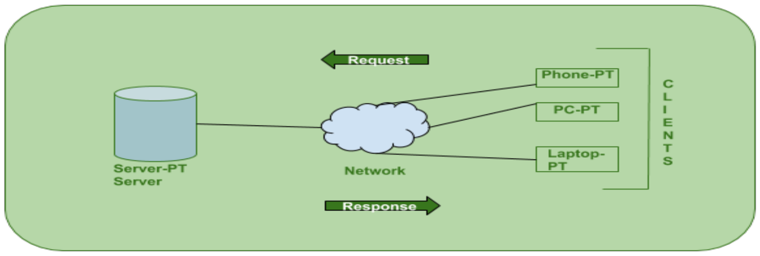


Figure 2.2: Client-server model representation. [18]

TCP has various methods to ensure reliable data transmission. Firstly, it uses a 3-way handshake process where it sends SYN and ACK packets to establish the connection. Secondly, it uses a flow control mechanism to prevent vast amounts of data from overflowing the receiver. Lastly, it uses error detection and correction through the use of checksums. [19]

We used TCP to ensure that the packets that were being sent, were received in the correct sequence and that none of them were missing. There were also discussed other possibilities to use other protocols instead of TCP, that ensured reliable transmission. [19]

2.1.3 User Datagram Protocol

UDP is another transport protocol. Even though it is in the same layer as TCP, it behaves in a completely different way. As stated, TCP is used to ensure the reliable transmission of data, where each packet is being checked, while UDP uses a best-effort mechanism. [20]

The mechanism refers to the protocol's approach of transmitting data without guaranteeing complete delivery of packets, or verification that packets arrive in order or error-free. [20] It is mainly used when speed is of more importance than reliability. Because it does not have to establish a connection using a three-way handshake there is less overhead associated with sending data using UDP.

UDP is often used on applications like streaming media, online video games, and video meetings, where slow speed can lead to poor user experience, and where the loss of packets is of exceptionally small importance. [20]

2.1.4 TCP versus UDP

Unlike last year, this year's communication included both TCP and UDP protocols. TCP protocol uses **connections**, meaning that it sends TCP flags to acknowledge the connection between the topside and ROV. This was used where data is being transported to ensure no loss of important data.

This year's solution implemented as well UDP protocol, which is **connectionless**, to transfer the camera feed from the ROV to the topside. This will not only speed up the process but also drastically decrease the CPU and process capacity needed for data transfer. Figure [2.3] shows how the speed difference between protocols is achieved. [20]

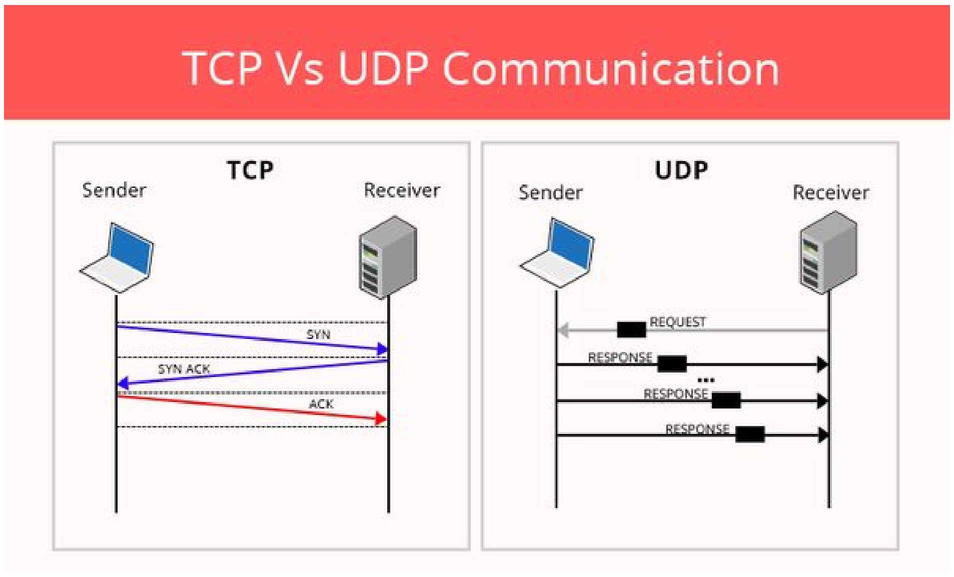


Figure 2.3: Representation of UDP and TCP data transport methods [21]

2.1.5 Communication Medium

In addition to transport protocols, it is important to be aware of the physical layer which covers physical networking media, such as a cable docking connector. The reason for that is that it can affect ROV performance drastically. For example, for the GUI, it was important that the cable had a high enough data transfer rate to be able to see the video live.

When choosing cable, additional calculations are needed, such as electrical properties, functional-, and process characteristics, and characteristic parameters including voltage, data transfer rate, maximum transmission distance, and physical connection media.

There were two options for cable types. The first one was to use fiber cable and the second was to use an Ethernet cable, either CAT5 or CAT6.

To avoid damage to the cable, which can cause loss of connection between the ROV and topside, an Ethernet cable was used, not only because it is more sturdy but because it was cheaper as well. According to "Library

Systems" the fiber price can be approximately 52 percent higher than a CAT 5 cable, and additionally, 70 percent higher to install it since skilled installers are required. [22]

One of the disadvantages of an Ethernet cable is that its range is limited, meaning it can typically only transmit data up to 100 meters without requiring additional equipment, but since we were mainly focusing on the MATE competition, and it takes place in a pool, it was of no importance. Additionally, we will participate in the TAC challenge in Tau which takes place in the ocean, but since the depth will not be exceeding 100 meters, we won't be needing any additional equipment. [22]

2.2 Camera

The ROV camera was chosen with regard to what other groups needed. A stereo camera with HD resolution was necessary for the **Machine Vision** group to perform fish counting, autonomous docking, distance measuring, and 3D modeling. The camera was equipped with tilt capabilities, making ROV steering easier.

The challenging, and one of the most important parts, was to design the camera in a way that it had the ability to withstand harsh underwater conditions like turbidity, high pressure, corrosive saltwater, and currents.

Figure [2.4] beneath shows one of the two cameras that were mounted to the ROV.

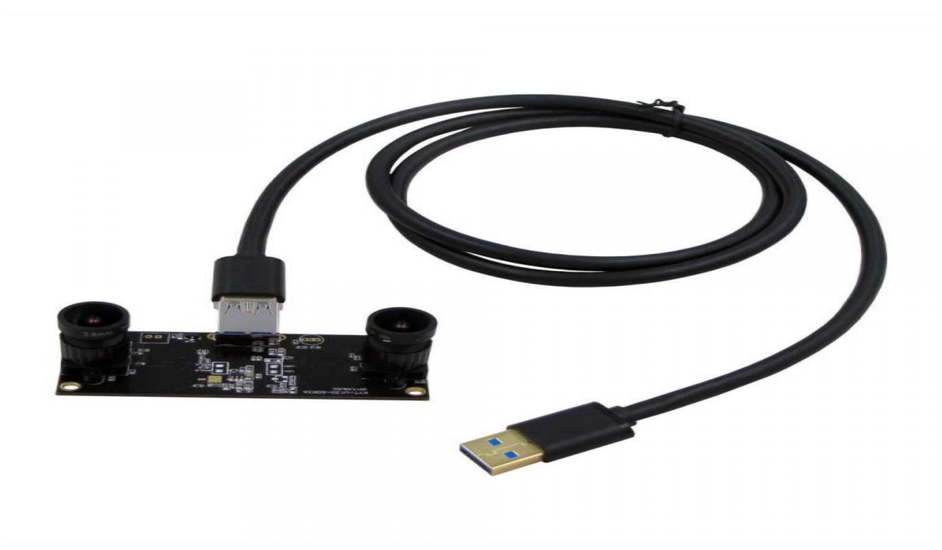


Figure 2.4: One of the ROV cameras [23]

2.2.1 Camera Positioning

The ROV used stereo cameras to have a visual angle around the whole ROV. The cameras were mounted at the front and at the bottom of the ROV. They were inside acrylic domes to make them waterproof.

2.2.2 Camera Lights

To be able to capture high-quality images and video footage deeper in the water, the lights play a crucial role. Therefore, there were LED lights mounted around the cameras. These helped enhance the contrast and clarity of the video feed, allowing the operator to better distinguish objects and have an easier orientation in the water.

2.2.3 Remote Connection

The Nvidia Jetson, see Figure [2.5], is normally used in AI for running multiple neural networks in parallel. However, in our project, it was used as a mini computer that would receive and send commands to the topside and the ROV. [24]

Figure 2.5: Nvidia Jetson Nano b01 4GB. [24]

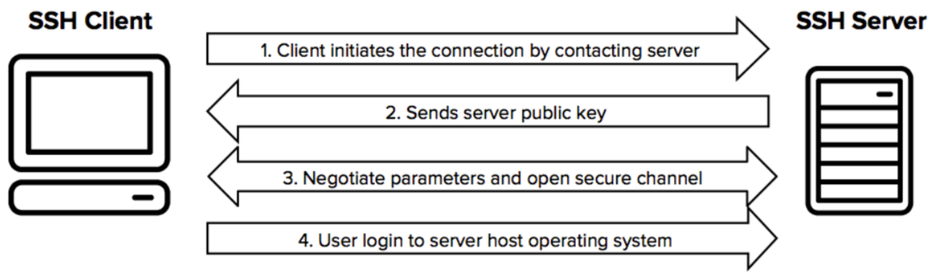

SSH Client
SSH Server
1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system

Figure 2.6: SSH Protocol

To get a connection we used SSH. SSH is a software package that enables secure system administration and file transfers over insecure networks. The

SSH protocol uses encryption to secure the connection between a client and a server. [25]

2.2.4 GStreamer

Figure 2.7: Gstreamer Logo [26]

To get the camera data from the Nvidia Jetson to the topside, **GStreamer** was used. GStreamer is a cross-platform, open-source pipeline-based framework for multimedia. GStreamer has plugins that can be linked and arranged in a pipeline which will then define how the data flows. [27] [28]

Code 2.1: Example on launching camera with GStreamer Pipeline

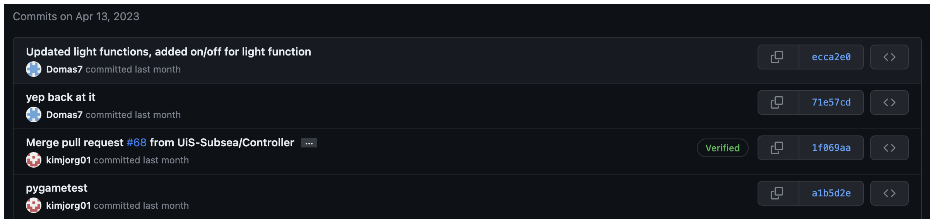
```
gst-launch-1.0 filesrc location=videofile.mpg ! dvddemux ! ...
    mpeg2dec ! xvimagesink
```

- `filesrc` location=`videofile.mpg`: Reads a file and passes it along the pipeline as raw data.
- `dvddemux`: This element **demultiplexes** the data into separate streams, in this case, it demultiplexes a DVD-compliant MPEG-2 stream.
- `mpeg2dec`: This element decodes the MPEG-2 video stream into raw YUV frames that can be displayed.
- `xvimagesink`: This element displays the YUV frames on the X Window System using the XVideo extension.

2.3 Development and Working Method

2.3.1 Git

Git is a version control system that can handle all sorts of projects. This means that users can save different versions of their code with the use of **commits**. Commits make it possible to go back and review previous code snippets, and make it easy to collaborate with others. This also makes it possible to revert any unwanted changes by going back to an earlier version of the project. The possibility of inviting members to contribute simultaneously to the project is a major advantage of Git. [29]



Commits on Apr 13, 2023

Updated light functions, added on/off for light function
Domas7 committed last month — ecca2e0

yep back at it
Domas7 committed last month — 71e57cd

Merge pull request #68 from UiS-Subsea/Controller ...
kimjorg01 committed last month — Verified — 1f069aa

pygametest
kimjorg01 committed last month — a1b5d2e

Figure 2.8: Example of commits merged into the main branch, showing the versatility of them. They all have their own ID which can be used to access each commit individually.

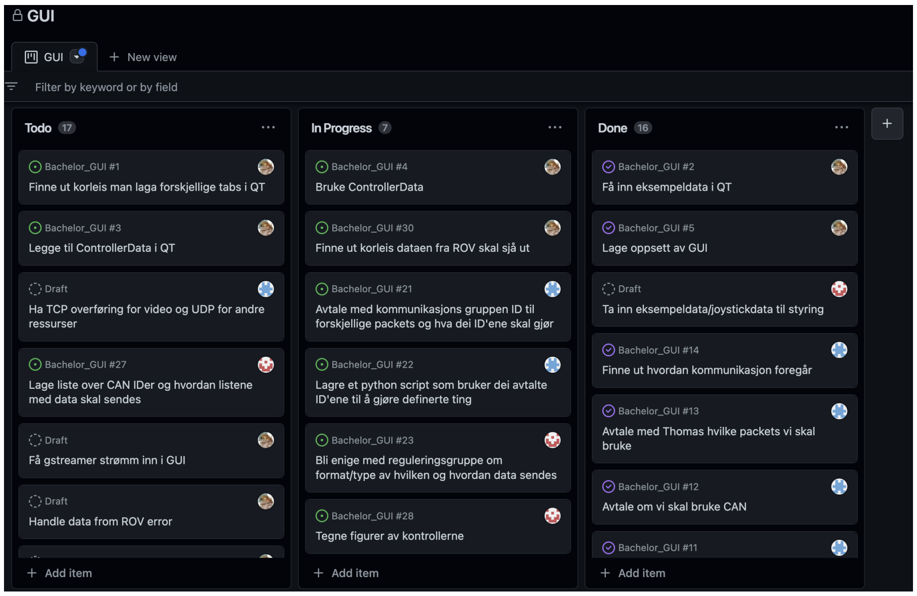
The version control is controllable by **branches**, which allows users to create diverging branches to test and verify new code implementations without jeopardizing the contents of the main branch. The diverging branches can be merged into the main branch to update it with the latest changes.

Gitignore

A **gitignore** file specifies intentionally untracked files that Git should ignore. [30] Simply put, a gitignore file prevents unnecessary files from being committed to the repository. This makes it easier to have an organized repository where few merge conflicts happen. To easily create a gitignore file it is possible to use the website Toptal. Toptal has a function that can automate the creation of chosen gitignore files. [31]

2.3.2 Github project table

GitHub Issues is an efficient tool to organize what to do and who does what in the project. It is a **Kanban** table where all members can add new to-do's and sort them by what needs to be done; **Todo**, what is currently being worked on; **In Progress**, and what is completed; **Done**. [32] [33]



GUI

GUI
New view
Filter by keyword or by field

Todo 17
- Bachelor_GUI #1 — Finne ut korleis man laga forskjellige tabs i QT
- Bachelor_GUI #3 — Legge til ControllerData i QT
- Draft — Ha TCP overføring for video og UDP for andre ressurser
- Bachelor_GUI #27 — Lage liste over CAN IDer og hvordan listene med data skal sendes
- Draft — Få gstreamer strømm inn i GUI
- Draft — Handle data from ROV error
- Add item

In Progress 7
- Bachelor_GUI #4 — Bruke ControllerData
- Bachelor_GUI #30 — Finne ut korleis dataen fra ROV skal sjå ut
- Bachelor_GUI #21 — Avtale med kommunikasjons gruppen ID til forskjellige packets og hva dei ID'ene skal gjør
- Bachelor_GUI #22 — Lagre et python script som bruker dei avtalte ID'ene til å gjøre definerte ting
- Bachelor_GUI #23 — Bli enige med reguleringsgruppe om format/type av hvilken og hvordan data sendes
- Bachelor_GUI #28 — Tegne figurer av kontrollerne
- Add item

Done 16
- Bachelor_GUI #2 — Få inn eksempeldata i QT
- Bachelor_GUI #5 — Lage oppsett av GUI
- Draft — Ta inn eksempeldata/joystickdata til styring
- Bachelor_GUI #14 — Finne ut hvordan kommunikasjon foregår
- Bachelor_GUI #13 — Avtale med Thomas hvilke packets vi skal bruke
- Bachelor_GUI #12 — Avtale om vi skal bruke CAN
- Bachelor_GUI #11
- Add item

Figure 2.9: Github Projects Table

2.3.3 Development strategy

The group consisted of three members, so the communication within the group was good and required minimal coordination to keep up to date. The chosen method of planning and keeping track of progress was by using Github's project table (Ch.2.3.2). A Kanban table can easily get overcrowded by to-do's if the group size is too big, but in this case, it is suitable and efficient. This allowed for splitting up tasks into what state they are in and allocating users to each task.

Further division of work was done by dividing into different areas of the project. As the bachelor thesis consisted of establishing a connection between the topside and the ROV, creating a GUI, and implementing steering for the ROV and manipulator, it was possible to split work into three different areas: communication, GUI, and steering. The result was specialization for each member in their branch of work. Less time was used on everyone understanding everything, and more time was used on getting a deeper understanding of the specialized field, resulting in more efficiency.

2.3.4 Meetings

Good communication is essential in larger projects. Meetings were arranged weekly to summarize the progress of all the groups in UiS Subsea. One member of each group joined these meetings every Monday, giving a short rundown of what was done last week and what needed to be done the current week.

Other meetings were arranged between smaller groups of members. Some groups relied more heavily on each other and could set up meetings to discuss topics related to them. This year, members of UiS Subsea had the opportunity to contact last year's Subsea team, which allowed them to get more help. Meetings with a supervisor were held every other week to give an update on the progress and ask questions.

2.4 Programming Language

To determine the optimal programming language for the project, an evaluation was conducted. The evaluation included an examination of the programming languages previously used in UiS Subsea to create the topside system. We looked at advantages and disadvantages to figure out which language was optimal for our use.

2.4.1 Earlier languages

Year	Earlier programming language	Groups field of study
2014	C# and .NET	Computer Science
2015	C++	Automation and Electronics Design
2016	Python	Computer Science
2017	—	—
2018	Python(with ReactJS and NODEJS)	Automation and Electronics Design
2019	MATLAB	Automation and Electronics Design
2020	—	—
2021	Python (with flask, Gunicorn and Nginx, JS) and GO	Automation and Electronics Design
2022	Python (with QT and PyQT)	Computer Science

Table 2.1: Programming languages used by earlier groups, and which year.

Table [2.1] shows the previous year's usage of programming languages for the UiS Subsea project, and their field of study. Based on the table, C# and Python were the most utilized and looked relevant to consider for the project.

2.4.2 Python

Python is an interpreted, object-oriented, high-level programming language that focuses heavily on readability and fast production. Python has a variety of libraries to use. Another positive element is that almost all engineering students in UiS Subsea have experience with Python from earlier subjects. All this makes Python a good candidate for the project. [34]

Figure 2.10: Python logo [34]

2.4.3 C#

C# is an object-oriented programming language created by Microsoft and designed to be a modern programming language that builds on C++ and C.

Just like Python, C# has a variety of libraries to choose from, many of which were developed in C++ and C, which makes for good documentation. [35] All these features made C# look like a good candidate for the project.

Figure 2.11: C# logo [36]

2.4.4 Choosing a language

When choosing a language, there are a few things to keep in mind: project requirements, availability of libraries and frameworks, community and documentation around the language, and development team skills.

After examining the two programming languages, we found that they fulfilled many of our needs. Both languages met the project requirements and had an extensive range of libraries with active communities and documentation. However, we opted for Python because there was a lack of experience with C# throughout the team. Because everyone understood Python, it would be easier for the groups to assist each other.

2.5 Frameworks

2.5.1 Qt

QT is a versatile framework for creating applications. It can be deployed across multiple platforms including mobile devices, embedded, and desktops. It is written in C++ and is used by leading companies and over 70 industries. Qt is being used to create a variety of different programs, examples of this are Adobe Photoshop Elements, EAGLE CAD IDE, Google Earth, TeamViewer, and many more. In these programs, QT is being used to create graphical interfaces and control the data going in and out. This can also be used to control other systems, which in our case was an ROV. [37] [38]

Figure 2.12: QT-logo [39]

2.5.2 PyQt

PyQt is a set of Python bindings that connects the C++ application framework and the cross-platform interpreted language Python. A binding is an application that translates commands from one language, which in this case is Python to C++ code for QT-specific commands. [40]

Creating a simple application in QT

This is how to create a simple PyQt5 Application. As shown in Code [2.2], a simple PyQt5 application window can be created with just a few lines of code. This is the equivalent of a Hello World! program.

Code 2.2: Pyqt5 example [41]

```python
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel


class SubseaGUI(QMainWindow):
    def __init__(self):
        super().__init__()  # Call the __init__ method of ...
            the QMainWindow class

         # Create a QLabel Widget
        label = QLabel("Hello UiS Subsea")

        # Set the central widget of the Window.
        self.setCentralWidget(label)


app = QApplication(sys.argv)  # Create an instance of the ...
    QApplication class
window = SubseaGUI()  # Create an instance of our ...
    SubseaGUI class
window.show()  # Show the window
app.exec()  # Execute the application
```

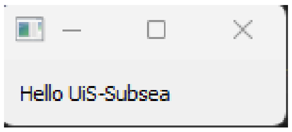

Hello UiS-Subsea

Figure 2.13: PyQt5 Window Example

Figure [2.13] shows how the window looks like in Windows 11. The "**Hello UiS Subsea**" is the Qlabel from the code.

The Event Loop

In a QT application, any interaction with the GUI, such as moving a slider or clicking a button, generates an event that gets added to an event queue. Whenever a waiting event is found, it gets sent to the appropriate event handler, which handles the event and gives control to the **event loop**, which waits for any new events. There is only one event loop in each application, this loop sits in the Qapplication (line 16 in Code 2.2).

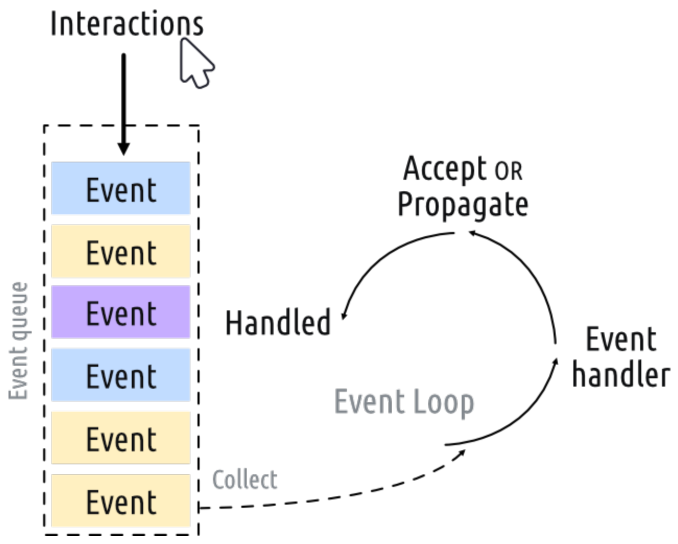
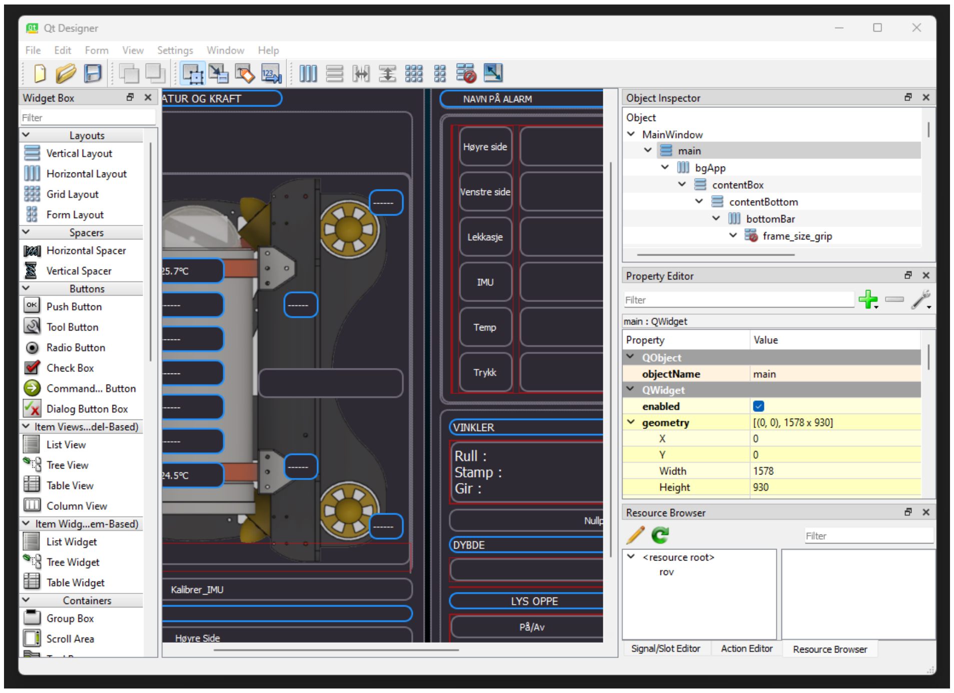


Figure 2.14: Event Loop In PyQt5 [41]

QT Designer

QT designer is one of many tools created by QT. It enables developers to create and modify GUIs. It has a big range of tools and features for creating layouts, using widgets, and customizing everything in the GUI. The result is stored in an **XML-based UI** file that can be used directly in QT designer or converted into a Python module with **UIC**.



Property	Value
QObject	
objectName	main
QWidget	
enabled	✓
geometry	[(0, 0), 1578 x 930]
X	0
Y	0
Width	1578
Height	930

Figure 2.15: QT Designer Window

UI-files

To convert the UI files into Python modules, UIC was used at the beginning of the GUIs Window class. See line 12 in Figure [2.3]. [42]

Code 2.3: How to use UIC to load a UI file

```
1  class Window(QMainWindow):
2      def __init__(
3          self,
4          gui_queue: multiprocessing.Queue,
5          queue_for_rov: multiprocessing.Queue,
6          manual_flag,
7          t_watch: Threadwatcher,
8          id: int,
9          parent=None,
10     ):
11         super().__init__(parent)
```

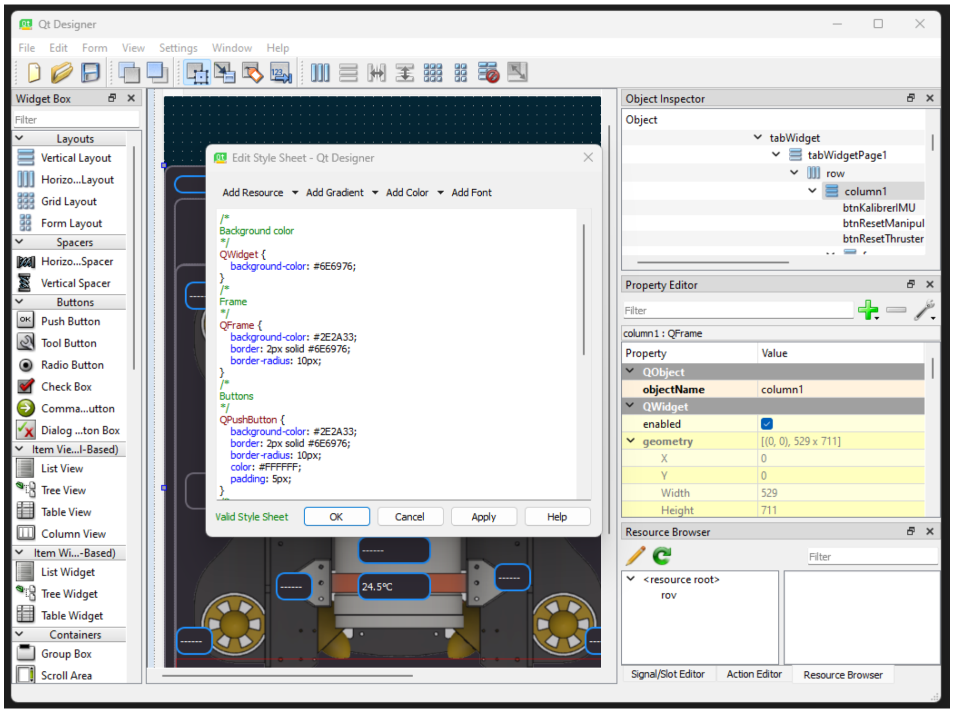
```
        uic.loadUi("gui/mainwindow.ui", self) # Load the ...
            .ui file
```

After importing the UI file, it is possible to reference the widgets in the code.

QT Style Sheets

QT Style Sheets gives us the ability to customize the appearance of our GUI. It has many similarities with CSS. [43] [44]

QT Style Sheets can be used in two ways. One in which you design everything in QT Designer and style the various widgets to change them specifically, or use a Stylesheet file to write it separately from the layout. In this project, designing through QT Designer was used. See Figure [2.16] where the **background-color**, **QFrame** and the **QPushButton** get styled.



```
/*
Background color
*/
QWidget {
   background-color: #6E6976;
}
/*
Frame
*/
QFrame {
   background-color: #2E2A33;
   border: 2px solid #6E6976;
   border-radius: 10px;
}
/*
Buttons
*/
QPushButton {
   background-color: #2E2A33;
   border: 2px solid #6E6976;
   border-radius: 10px;
   color: #FFFFFF;
   padding: 5px;
}
```

column1 : QFrame

Property	Value
QObject	
objectName	column1
QWidget	
enabled	✓
geometry	[(0, 0), 529 x 711]
X	0
Y	0
Width	529
Height	711

Figure 2.16: QT Designer Window with stylesheet

2.5.3 Pygame

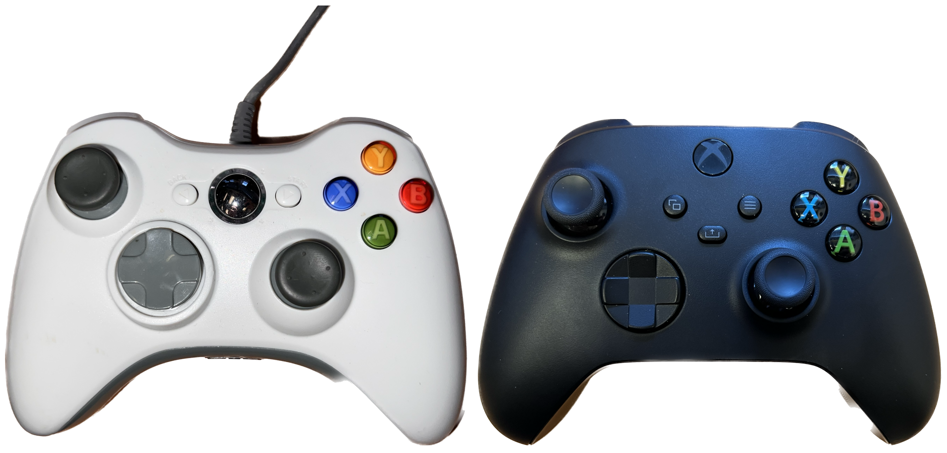
Pygame is designed to create video games using a set of Python modules. Pygame adds more functionality to the already existing SDL library. The Simple DirectMedia Layer is created to provide access to data such as joysticks, mouse, audio, keyboard, and graphics hardware via OpenGL and Direct3D. This warrants the accessing of controller data, which was used to control both the ROV and the Manipulator. [45] [46]

2.6 Steering

This year it was decided to use two separate controllers for steering the ROV and Manipulator. This decision made it possible to implement more functions if needed and allowed for an easier understanding of the controls.

2.6.1 Choice of Controller

Choosing a suitable controller type for the project was important for gathering the correct data that should be sent from the topside to the ROV. When first approaching the decision of choosing what controllers to use, the group started out by researching previous years' implementation of steering in UiS Subsea. This narrowed the choices down to two types of controllers: Xbox 360 controller and Xbox Series X controller.



(a) Xbox 360 controller **(b)** Xbox Series X controller

Figure 2.17: Choice of controller type between **(a)** and **(b)**

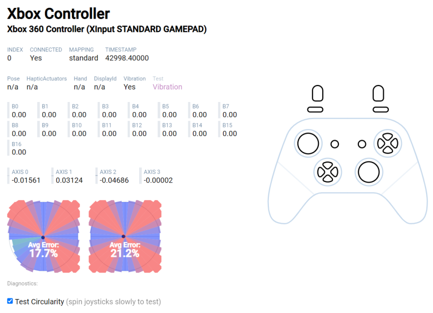
After discussing which option would be best and testing both types of controllers, it was clear that the Xbox Series X controller had advantages over the older Xbox 360 controller. Not only would it be necessary to buy an extra controller of the older type if this was the chosen controller type, but it was also quite imprecise compared to the newer controller type, as seen in Figure [2.18].

Having chosen a controller type, we used some of the budget for our group to purchase two new Xbox Series X controllers. These can be seen as investments, since now they will be used by this year's students in UiS Subsea, and future projects can have them at their disposal.

2.6.2 Testing

To establish an understanding of how precise, which buttons, and what values each input from the controllers gave, a website called **gamepad-tester** was used to display these values. [47]

As seen in Figure [2.18] below, sub-figure **(a)** shows the preciseness of the Xbox 360 controller. Compared to sub-figure **(b)** there is a clear difference in the preciseness of the joysticks favoring the new Xbox Series X controller type.



Xbox Controller
Xbox 360 Controller (XInput STANDARD GAMEPAD)

INDEX	CONNECTED	MAPPING	TIMESTAMP
0	Yes	standard	42998.40000

Pose	HapticActuators	Hand	DisplayId	Vibration	Test
n/a	n/a	n/a	n/a	Yes	Vibration

B0	B1	B2	B3	B4	B5	B6	B7
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

B8	B9	B10	B11	B12	B13	B14	B15
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

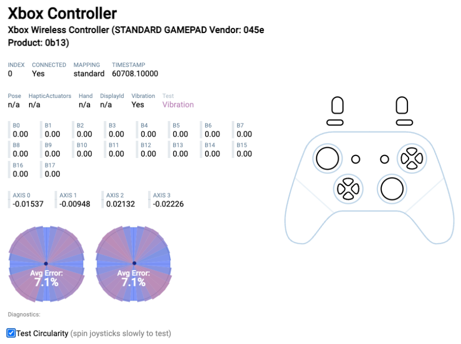
B16
0.00

AXIS 0	AXIS 1	AXIS 2	AXIS 3
-0.01561	0.03124	-0.04686	-0.00002

Diagnostics:

☑ Test Circularity (spin joysticks slowly to test)

(a) Xbox 360 controller test



Xbox Controller
Xbox Wireless Controller (STANDARD GAMEPAD Vendor: 045e Product: 0b13)

INDEX	CONNECTED	MAPPING	TIMESTAMP
0	Yes	standard	60708.10000

Pose	HapticActuators	Hand	DisplayId	Vibration	Test
n/a	n/a	n/a	n/a	Yes	Vibration

B0	B1	B2	B3	B4	B5	B6	B7
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

B8	B9	B10	B11	B12	B13	B14	B15
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

B16	B17
0.00	0.00

AXIS 0	AXIS 1	AXIS 2	AXIS 3
-0.01537	-0.00948	0.02132	-0.02226

Diagnostics:

☑ Test Circularity (spin joysticks slowly to test)

(b) Xbox Series X controller test

Figure 2.18: The two controllers tested

An additional bonus of the new Xbox controllers is that they have the possibility of connecting to the computer wirelessly. This was not needed for this year's project but might be useful for future projects if they want to control the ROV or Manipulator without a cable.

2.6.3 ROV

The ROV has three directional axes: X-, Y-, and Z axis. These axes describe in which direction the ROV should move based on the input from its controller.

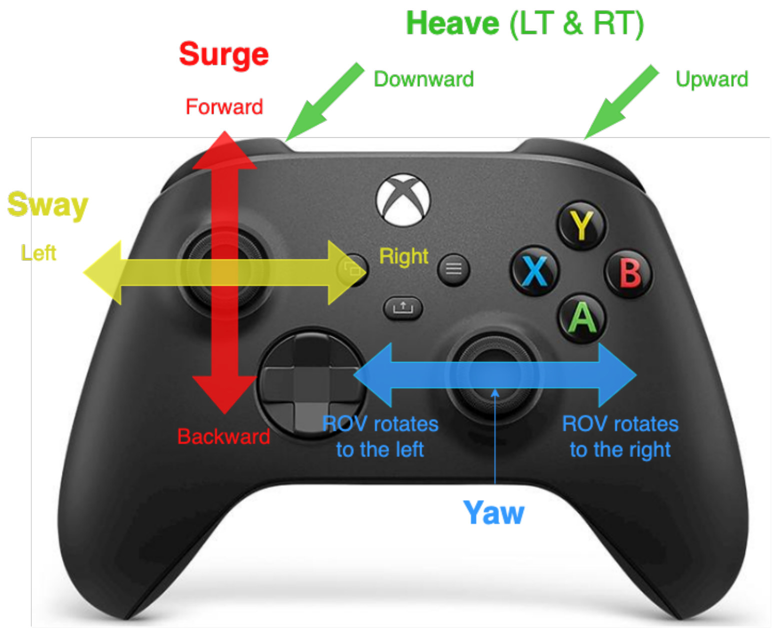


Figure 2.19: ROV controls. Shows what each joystick and trigger responds to as the corresponding directional movement.

Figure [2.19] shows which joysticks and axes were used for controlling the ROV. **Surge** responds to the X axis, which was the axis for moving forward and backward. **Sway** describes the Y axis, which corresponded to moving the ROV left and right. **Heave** was controlled by the left and right triggers, so that movement up and down vertically was possible. Finally, **Yaw** controls the left and right rotation of the ROV.

2.6.4 Manipulator

The Manipulator had its own controller which allowed for more functionality if decided later in the project. The basic features of the Manipulator were to extend and retract (**telescope**), tilt up and down, rotate the claw left or right, and grip and release the claw. The Figure [2.20] below shows which buttons and joysticks respond to these features.

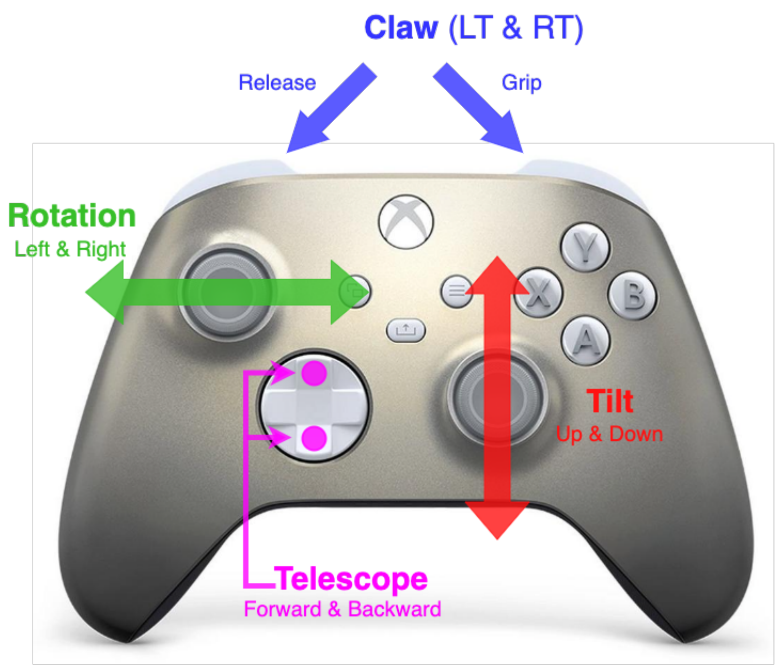


Figure 2.20: Manipulator Controls

2.7 Operating System

Choosing an operating system (OS) is an important part of any project. This is because it can have a lot of impact on compatibility, security, and system performance. An operating system is important because it serves as the bridge between hardware and software. Through our project, there were two main operating systems we looked at. These were Ubuntu and Microsoft Windows 11.

2.7.1 Microsoft Windows 11

Figure 2.21: Windows 11 logo [48]

Windows 11 is an operating system created by Microsoft. It was released in October 2021 and is known for its software ecosystem combined with a user-friendly GUI. [49] It also has **WinUI**, which is the premiere User Interface framework for Windows desktop apps. This is a great tool for creating UIs and could be of great use in a project like this.

Windows 11 also has great support for WSL, Windows Subsystem for Linux, and can be helpful to do tasks that are possible in Linux, but not in Windows. [50]

Some of the features that seemed promising in Windows include:

- Great compatibility: Windows 11 has great compatibility with many programming languages and frameworks like WinUI.
- Popularity and support: Windows 11 is very popular and there is a lot of support to get from communities on the internet and the other groups in UiS Subsea.

2.7.2 Ubuntu

Figure 2.22: Ubuntu logo [51]

Ubuntu is an open-source operating system based on the Linux Kernel. It was developed by a British company called Canonical and a community of other developers.
Ubuntu is known for its flexibility and aims to be secure by default. [52]

Some of the features that seemed promising in Ubuntu include:

- Good Support: Ubuntu has great support for many programs, with many official packages and repositories available. It is also easy to install and manage these tools.
- Pre-installed SSH: Ubuntu comes with pre-installed SSH. This can make it easy to manage another system remotely.

2.7.3 Choosing an operating system

In the end, we ended up running the project on Ubuntu. The Nvidia Jetson was running Ubuntu, and it was better to use the same OS on all computers. Even if Windows 11 had good support for WSL, it was still easier to run and set up everything we needed in Ubuntu.

2.8 Processing and Threading

To effectively execute multiple tasks in our program, we used processes from Python's **multiprocessing** package and threads from Python's **threading** package.

2.8.1 Process

In the context of computing, a process is an instance of a computer program that is being executed. It has its own memory space and is managed by the operating system. [53]

An example of how to start a process in Python can be seen in Code [2.4].

Code 2.4: How to create a process in Python [54]

```python
def f(name):
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

This process prints "hello bob" in a separate process from the main process running. The **Start** method executes the process, while the **join** method makes sure the main process waits for the new process to complete before continuing.

2.8.2 Thread

A thread is a subset of the process. Within an operating system, it is the smallest unit of execution that can be scheduled. Multiple threads can coexist within one process, sharing the process resources, but executing independently. [53]

An example of how to start a thread can be seen in Code [2.5].

Code 2.5: How to create a thread in python [55]

```python
def print_number(number):
    print('Number:', number)

if __name__ == "__main__":
    t = Thread(target=print_number, args=(1,))
    t.start()
```

This program creates a new thread separate from the main program thread. This thread executes and prints the number "1".

Chapter 3

Implementation

3.1 Process flow

This chapter will refer to the sequence of steps that a program follows to execute specific tasks. In our case, we have several files that have to run simultaneously to achieve the desired result. The program's structure and logic will be visualized using flowcharts and diagrams.

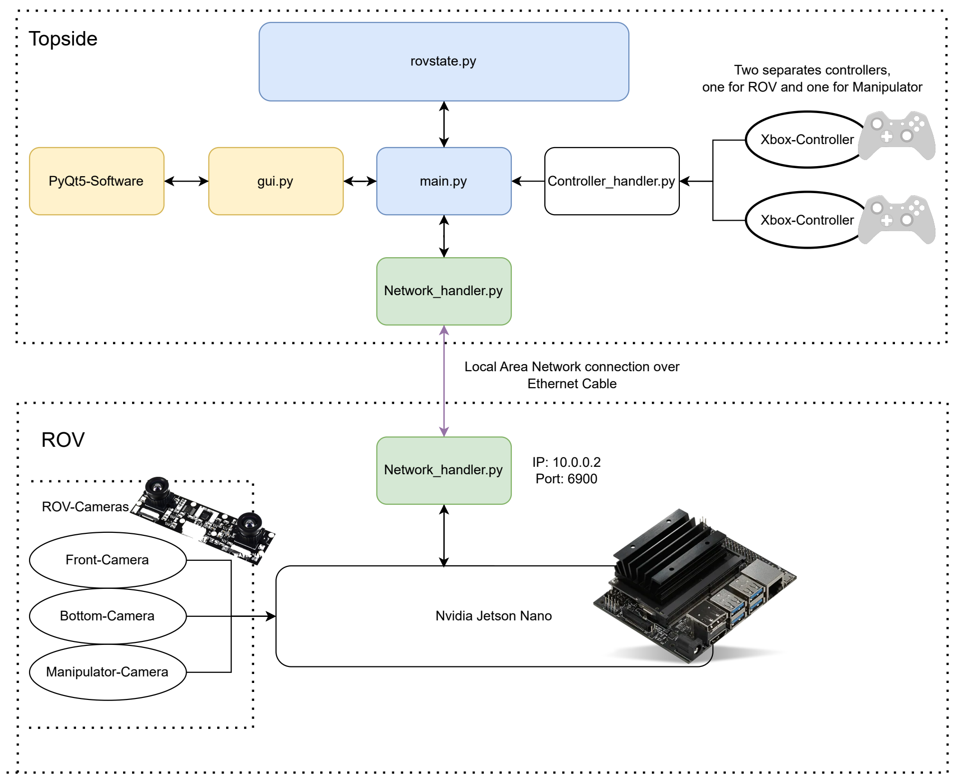


Figure 3.1: Communication structure between topside and ROV.

The figure above shows a representation of the communication structure between the topside system and the ROV. This year, to make piloting easier, we implemented two controllers that pilot the Manipulator and ROV separately.

Rovstate.py is where our main class **Rov_State** is implemented and used by **main.py**. This class holds most of the functionality, such as encoding, decoding, and creating packets.

Network_handler is where the connection between the topside and ROV is implemented. The connection was used to send data from the topside to ROV as a list with different IDs and data for the corresponding ID. When receiving data from the ROV, it is received as a dictionary where the key is the name of the packet and the value refers to a list of data. This ensured that the right packet is being sent and received which helps place the right sensor packet at the correct location. Additionally, sensor data is being sent via the TCP protocol, which ensures no loss when being transported.

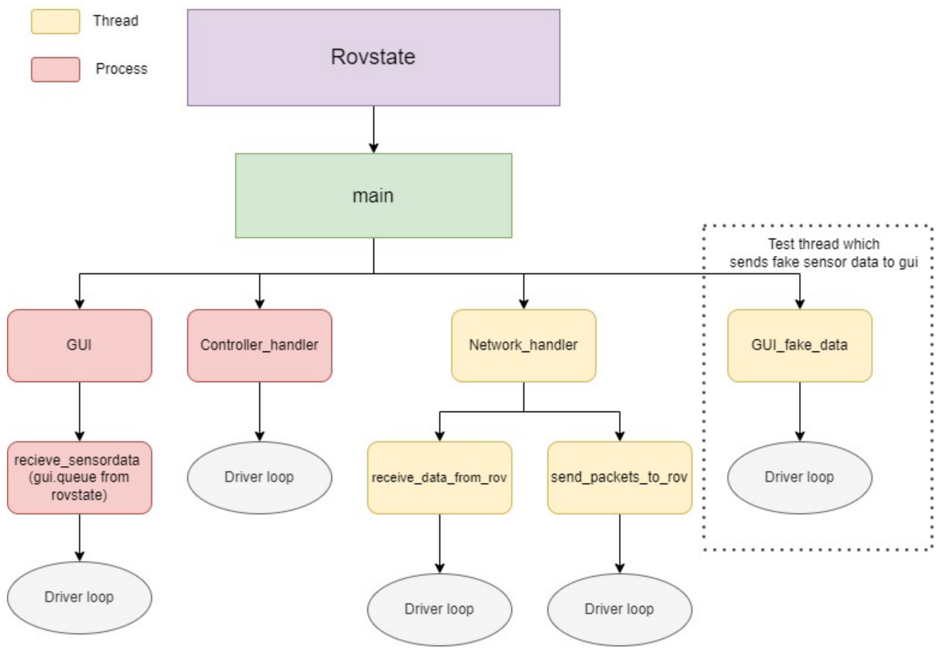


Figure 3.2: Main file structure.

The figure above shows a detailed representation of how threads and processes relate to **main.py**. The whole program is split into several threads, and each thread has its own file. Not only does this help to get a better overview of the structuring, but also significantly increases performance.

The program is started by running **main.py**, which initiates a series of threads and processes. As we worked on multiple tasks simultaneously, we evaluated various options for efficient execution. Through discussion within our, and other groups, we narrowed down the options to using multiprocessing and threads.

When the program is started, it first checks whether the **Network_handler** thread is started (see Code [3.1]). This is simply done by setting the run_network variable to either True or False. The **Network_handler** is responsible for Ethernet connection to the ROV, which uses TCP to transport **heartbeat packets**. Heartbeat packets work as a check function that ensures that there still is a TCP connection between the **Jetson** and the topside [56]. In case the connection drops, a new thread starts and waits for a new connection.

Code 3.1: Initializing Network_handler thread.

```
if run_network:
    network = Network(is_server=False, port=6900,
    bind_addr="0.0.0.0", connect_addr="10.0.0.2")

    rovstate = Rov_state(queue_for_rov, network,
    gui_queue, manual_flag,  t_watch)

    id = t_watch.add_thread()

    rov_state_recv_loop = threading.Thread(target=
    rovstate.receive_data_from_rov, args=(t_watch, id), ...
        daemon=True)

    rov_state_recv_loop.start()

    rov_state_send_loop = threading.Thread(target=
    rovstate.send_packets_to_rov, args=(t_watch, id), ...
        daemon=True)

    rov_state_send_loop.start()
```

Next, independently of whether **Network_handler** is running or not, a new process within the GUI is initiated (see Code [3.2]) that continuously retrieves data by entering a while loop represented as **Driver loop**. This process retrieves data from a queue called **gui_queue** that continuously sends data until it is forcefully closed or until it gets a signal from the Threadwatcher class which indicates that the thread has stopped. The Threadwatcher and its function will be explained in Chapter 3.4.

Code 3.2: Initializing GUI process.

```python
if run_gui:
    id = t_watch.add_thread()

    gui_loop = Process(target=gui.run, args=(gui_queue,
    queue_for_rov, manual_flag,  t_watch, id), daemon=True,
    )

    gui_loop.start()
```

The **Controller_handler** is another process initiated right after the GUI. It checks for connected controller devices and initializes them. It then starts a while loop that continuously retrieves data from the controller(s) and sends it through a queue. Just like the GUI process, the process is continued until forcefully stopped or until it receives a stop signal from the Threadwatcher class.

Code 3.3: Initializing **Controller_handler** process.

```python
if run_get_controllerdata:
    id = t_watch.add_thread()

    controller_process = Process(target=controller.run, args=(
        queue_for_rov, manual_flag, t_watch, id, True, ...
            debug_all), daemon=True)
    controller_process.start()
```

Lastly, **send_fake_sensor_data** is a function that simulates data and sends it to the GUI. This data can be useful for a variety of reasons, such as testing and validating software that processes sensor data, or for experimenting with different scenarios and data inputs. Additionally, by using this test thread, we can develop and test code without requiring access to real data. This saves time, resources and helps to identify potential issues or bugs early in the development process.

Code 3.4: Initializing **run_send_fake_sensordata** thread.

```python
if run_send_fake_sensordata:
    id = t_watch.add_thread()

    datafaker = threading.Thread(target=
    send_fake_sensordata, args=(t_watch, ...
        gui_queue),daemon=True,
    )

    datafaker.start()
```

3.2 Data transfer between topside and Nvidia Jetson

Part of our project was to establish a communication channel between the topside and the **Jetson**. We had two main tasks to perform regarding communication channels. The first was to receive data from the ROV which was then displayed in the GUI, while the second task was to send information packets to the Jetson. The information was sent from the Jetson to one or several circuit boards in the electrical housing of the ROV.

This year it was decided to use a larger number of circuit boards, mainly to reduce the temperature and the risk of overheating. The image and tables below will show every main circuit board that was used. Tables will only show the packets that were being sent from the GUI. The data that was received from the ROV will be shown later in the chapter.

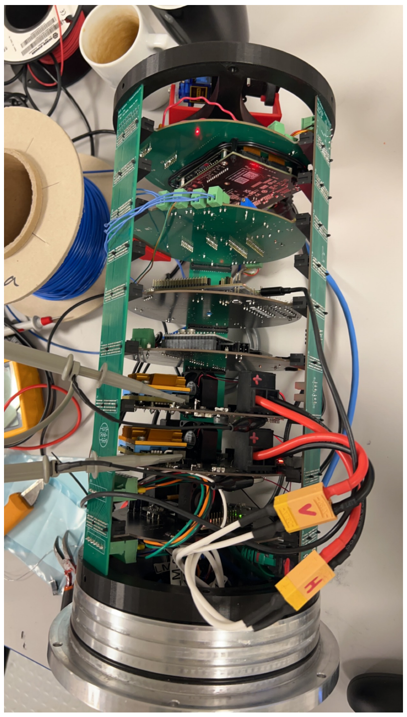


Figure 3.3: Electronic Housing

In Figure [3.3], you can see the electronic housing for this year's project. As shown in the image, the housing contains a total of seven circuit boards, arranged from top to bottom in the following order:

1. **Communication Circuit Board:** Ensures communication between the topside and other circuit boards.
2. **Sensor Board:** Provides sensory information.
3. **Regulation Board:** Handles the ROV's regulation.
4. **Motor Driver Module:** Handles steering signals to the thrust of the motors.
5. **Right 12V Fuse Board:** Responsible for the right side of the power system.

6. **Left 12V Fuse Board:** Responsible for the left side of the power system.

7. **5V Fuse Board:** Main fuse.

3.2.1 Control System Circuit Board

canID	Byte	Bit	Type	Description/Message	Sender	Note
32			Control Word		IDX_Kom	Init
32	B0		bytearray			
32		b0	bit	0: All regulators deactivate		1: ON, 0: OFF
32		b1	bit	1: Activate roll reg.		
32		b2	bit	1: Activate stamp reg.		
32		b3	bit	1: Activate depth reg.		
33				Styredata ROV	IDX_Kom	Styredata ROV
33	B0		int8	X_axis		ROV framover/bakover
33	B1		int8	Y_axis		ROV venstre/høyre
33	B2		int8	Z_axis		ROV opp/ned
33	B3		int8	Rotation		ROV snu venstre/høyre
33	B4					
34			Styredata Manipulator		IDX_Kom	Styredata Manipulator
34	B0	int8	Manipilator IN/OUT			
34	B1	int8	Rotation Left/Right			
34	B2	int8	Tilt Up/Down			
34	B3	int8	Claw Grip/Release			

The presented table describes a communication protocol used in a system that controls the ROV and the Manipulator. The system communicates using messages transmitted over a CAN bus, which is a common communication standard in industrial automation and automotive applications. [57]

The first row in the table specifies the message identifier (CAN ID), which is a unique identifier that sorts different messages on the CAN bus. For the regulation circuit board, three messages are defined with CAN ID 32, 33, and 34. While CAN IDs 33 and 34 are responsible for controlling the ROVs and manipulators' movement and directions, CAN ID 32 is used to control ROV movement without manual interference. This method of control is called regulation and could adjust specific parameters for the ROV to control its roll, pitch, and depth.

A reference for how the movement functionality of the ROV and Manipulator is implemented can be viewed in Code [3.14].

3.2.2 Sensor Circuit Board

canID	Byte	Bit	Type	Description/Message	Sender	Note
66			Control Word		IDX_Kom	
66	B0		Bitarray	Initializing Flag		Set Initializing flag
66		b0		Zeropoint Depth		Resetting Depth
66		b1		Zeropoint Angles		Resetting Angles
66		b2		Calibrate IMU		Calibrating IMU
66		b3		Watertype		0 = Fresh, 1 = Salt

Like in the regulation circuit board table, the sensor board table includes information related to different bits. The bits in the **Control Word** are used to initialize flags, set null points for depth and angles, specify the water type, and calibrate IMU.

Calibrating an IMU involves setting its sensors to the correct initial values and improving a sensor's performance by removing structural errors from the measurements of the sensor. In summary, calibrating an IMU is an important step to ensure the accuracy and reliability of its measurements. [58]

Code 3.5: Building **Calibrate_IMU** packet. This demonstrates one of the packets required to be built and sent to the sensor circuit board.

```python
def calibrate_IMU(self):
    calibrate_IMU_byte = [0] * 8
    calibrate_IMU_byte[0] |= 1 << 2
    print("Kalibrerer IMU")
    values = {"kalibrer_IMU": calibrate_IMU_byte}
    print(("Want to send", 66, calibrate_IMU_byte))
    self.queue.put((9, values))
```

As seen in Code [3.5], the function first initializes a packet by creating a list of eight zeros, which is then used to construct a byte (a sequence of eight bits) to send the IMU. To modify a specific bit in the byte, a **Bitwise OR** operator was used to set the value of the third bit to 1, which is equivalent to **True** in Python.

Next, we create a dictionary called "values" that includes the calibration byte under the key name "kalibrer_IMU". This dictionary is added to a queue and sent to the **Rov_state** class where its ID is specified and added to a list of packets to send.

3.2.3 Power Circuit Board

canID	Byte	Bit	Type	Description/Message	Sender	Note
97				Control Word 5V	IDX_Kom	
97	B0		Bitarray	Function Control		Set Initializing flag
97		b0		Reset Fuse 5V		'True' to Reset
98				Control Word 12V Thruster Supply	IDX_Kom	Thruster = Right Side
98	B0		Bitarray	Function Control		
98		b0		Reset fuse 12V Thruster		
98		b1		Front Light On		
98	B1		uint8	Front Light Dimming SP	IDX_Kom	0-100
98	B0		Bitarray	Function Control		Set Initializing flag
98		b0		Reset Fuse 5V		'True' to Reset
99				Control Word 12V Manipulator Supply	IDX_Kom	Manipulator = Left Side
99	B0		Bitarray	Function Control		
99		b0		Reset fuse 12V Manipulator		
99		b1		Bottom Light On		
99	B1		uint8	Bottom Light Dimming SP	IDX_Kom	0-100

The power circuit board includes three messages with CAN IDs 97, 98, and 99. Each message contains one or more bytes, and each byte has eight bits. The table specifies the type of data for each byte, such as bitarray or uint8, and provides a description of the message or function that it controls.

Message 97 controls the 5V supply to the system and contains a control word in byte B0 that is used to set initializing flags. Bit0 is then used to reset the fuse for the 5V supply.

Similarly, message 98 controls the 12V supply to the thruster and contains a control word in byte B0. Bit0 in Byte0 is used to reset the 12V thruster supply, bit1 is used to turn on/off the front light, and byte1 is a uint8 type that is used to set front light dimming.

Lastly, message 99 has almost the same functionality, except it resets the 12V manipulator supply. It turns on/off the bottom light, and it is used to set the bottom light dimming setpoint.

Code 3.6: Building the **Light_Dimming** packet.

```python
def update_label_and_print_value_down(self, value):
    self.label_percentage_lys_down.setText(f"{value}%")
    set_light_byte = [0] * 8
    set_light_byte[1] = value
    values = {"slider_bottom_light": set_light_byte}
    print((f"Want to send", 99, set_light_byte))
    self.queue.put((18, values))
    print(value)
```

Code [3.6] shows one of the functions the power circuit board is responsible for. The function is a method implemented in the GUI for controlling the brightness of an LED light. It takes a numerical value, representing the percentage of brightness to be set for the light, as input. The byte is then added to the **values** variable and sent the same way as stated in Code [3.5].

3.2.4 Communication Circuit Board

The communication circuit board is a crucial component. It allows for the transmission and reception of information through various channels. In this project, the communication circuit board serves the purpose of transmitting sensor data such as acceleration, temperature, error messages, water depth, and power output.

The following code snippets illustrate the functions used to send a packet to the **Communication** group, which then encoded them for the CAN bus and forwarded them to other circuit boards.

Code 3.7: Initializing depth regulator function

```python
def toggle_dybde_reg(self):
    self.toggle_felles_regulator[0] ^= 1 << 3
    if self.toggle_felles_regulator[0] == (1 << 3):
        print("dybde pÃ¥")
    elif self.toggle_felles_regulator[0] == (0 << 3):
        print("dybde av")
    print(("Want to send", 32, self.toggle_felles_regulator))
    values = {"toggle_rull_reg": self.toggle_felles_regulator}
    self.queue.put((7, values))
```

Code 3.8: Building depth regulation byte.

```python
def build_reset_depth(self):
    if self.data == {}:
        return
    data = [0, 0, 0, 0, 0, 0, 0, 0]

    data[0] = self.data["reset_depth"][0]

    self.packets_to_send.append([66, data])
```

Code 3.9: Setting depth ID

```python
def build_packets(self):
    if self.packet_id == 1 and self.manual_flag.value == 1:
        # self.button_handling()
        self.build_rov_packet()
        self.build_manipulator_packet()
    elif self.packet_id == 2 and self.manual_flag.value == 0:
        self.build_autonom_packet()
    elif self.packet_id == 4:
        self.build_reset_packet()
    elif self.packet_id == 5:
        self.build_reset_thruster_packet()
    elif self.packet_id == 6:
        self.build_reset_manipulator_packet()
    elif self.packet_id == 7:
        self.build_reset_depth()
    elif self.packet_id == 8:
```

The **toggle_dybde_reg** (Code [3.7]) function modifies a variable that controls whether the depth regulator is on or off and enqueues the updated value to be sent over to a communication channel on the **Communication** group's side. **build_reset_depth** creates a byte sequence that resets the ROV's depth to its starting value.

The **build_packets** (Code [3.8]) function determines which type of packet to build based on the value of **packet_id**, and calls **build_reset_depth** if the value is 7. The packets are then added to a list to be sent over the communication channel where they are coded to CAN bus.

The functions also demonstrate the importance of code modularity and encapsulation, making it easier to update or modify specific features without impacting the entire system.

3.3 Controllers

Implementing controller functionality required the use of Pygame (see Ch. 2.5.3) to gather the data needed. To get the controller data, it was needed to first initialize the controllers, then listen for input, and lastly add the **events** to a list, allowing the data to be sent further. This data could then be accessed and used to create packets that are sent from the topside to the ROV.

3.3.1 Accessing the controller data

A class called **Controller** contains all the necessary functions and variables to load controllers, check for input, and add the controller input data into a queue for further use.

wait_for_controller

The first step in using a controller to steer the ROV or Manipulator is to create a function that can initialize the controller(s). Code [3.10] displays how to initialize either one or two controllers using Pygame's built-in joystick initialization:

Code 3.10: wait_for_controller function

```python
def wait_for_controller(self):
    while True:
        try:
            global rov_joystick
            global mani_joystick
            pygame.joystick.init()
            if pygame.joystick.get_count() == 0:
                raise Exception
            if pygame.joystick.get_count() == 1:
                print(f"Found ...
                    {pygame.joystick.get_count()} ...
                    controller. Connecting to ROV Only!")
                rov_joystick = pygame.joystick.Joystick(0)
```

```
12                  print(f"Connected to ...
                        {rov_joystick.get_name()}")
13              if pygame.joystick.get_count() == 2:
14                  print(f"Found ...
                        {pygame.joystick.get_count()} ...
                        controllers. Connecting BOTH!")
15                  rov_joystick = pygame.joystick.Joystick(0)
16                  mani_joystick = pygame.joystick.Joystick(1)
17                  print(f"Connected to ...
                        {rov_joystick.get_name()}")
18                  print(f"Connected to ...
                        {mani_joystick.get_name()}")
19              break
20          except Exception as e:
21              print(e)
22      if pygame.joystick.get_count() == 1:
23          rov_joystick.init()
24      elif pygame.joystick.get_count() == 2:
25          rov_joystick.init()
26          mani_joystick.init()
```

Line 6 in the function above checks if there are any controllers connected to the computer, then loads any registered controllers such that they can be used further in the function, and adds them to a list that can be accessed by calling **pygame.joystick.Joystick(index)**. This makes the registered controllers available for use, but it does not automatically assign a variable to them.

Therefore, by creating the variables **rov_joystick** and **mani_joystick**, and assigning them to their own controller ID, they are distinguishable when trying to access their respective inputs. At the end of **wait_for_controller** (line 22-26) the controllers get individually initialized depending on if there are one or two controllers connected.

get_events_loop

The function **get_events_loop** got called when running the program, and was responsible for checking what events were called. After determining this, it handled the events according to the intended functionality of the corresponding controller inputs.

Code 3.11: get_events_loop function

```python
def get_events_loop(self, t_watch: Threadwatcher, id: int, ...
    debug=False, debug_all=False):

    while t_watch.should_run(id):
        if pygame.joystick.get_count() < 1:
            self.wait_for_controller()

        self.duration = self.clock.tick(20)

        for event in pygame.event.get():
            self.manual_flag.value = 1
            if event.type == DPAD:
                if event.joy == MANIPULATOR_CONTROLLER_ID:
                    self.mani_dpad = event.value
```

This function used **Threadwatcher** to check if it should be running or stopping. The **t_watch.should_run(id)** is a Boolean value that returns **True** if the thread exists. For further explanation of the Threadwatcher class, see Chapter 3.4.

The final action of the function was to add all the inputs from the controller into a queue that could be accessed and further sent from the topside to the ROV.

Code 3.12: Packing all the controller values and sending them to main.py

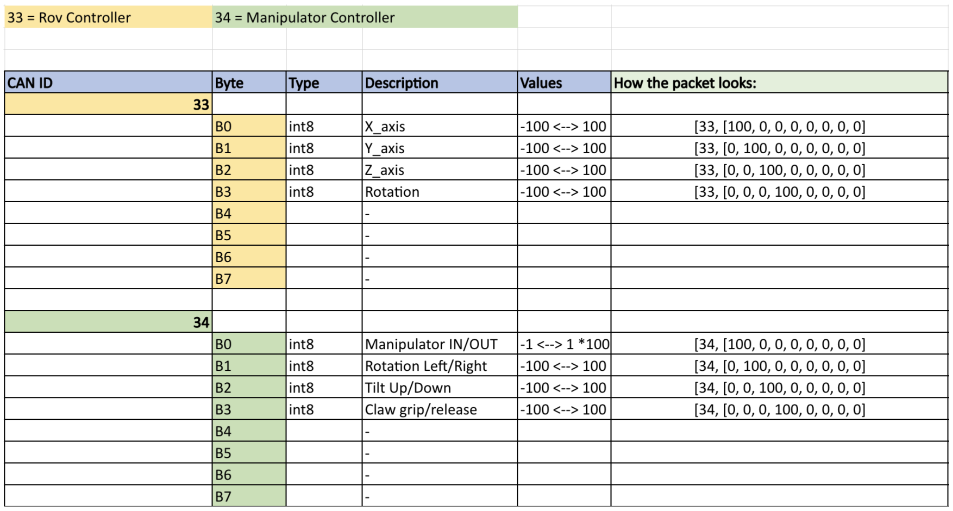
```python
if self.queue_to_rov is not None:
    self.queue_to_rov.put((1, self.pack_controller_values()))
```

3.3.2 Steering data

Most packets in the code were created using a list of eight integers, with the initial value of zero:

```
data = [0,0,0,0,0,0,0,0]
```

Depending on the type of packet to be created, these eight integers could be replaced by wanted data in corresponding indexes of the list. The steering data contained two types of packets: one for the ROV and one for the Manipulator. Figure [3.4] lists the bytes used for each maneuver, and how the packet could look on our end before it was transported and converted to hexadecimal byte values by the **Communication** group.



33 = Rov Controller	34 = Manipulator Controller				
CAN ID	**Byte**	**Type**	**Description**	**Values**	**How the packet looks:**
33					
	B0	int8	X_axis	-100 <--> 100	[33, [100, 0, 0, 0, 0, 0, 0, 0]
	B1	int8	Y_axis	-100 <--> 100	[33, [0, 100, 0, 0, 0, 0, 0, 0]
	B2	int8	Z_axis	-100 <--> 100	[33, [0, 0, 100, 0, 0, 0, 0, 0]
	B3	int8	Rotation	-100 <--> 100	[33, [0, 0, 0, 100, 0, 0, 0, 0]
	B4		-		
	B5		-		
	B6		-		
	B7		-		
34					
	B0	int8	Manipulator IN/OUT	-1 <--> 1 *100	[34, [100, 0, 0, 0, 0, 0, 0, 0]
	B1	int8	Rotation Left/Right	-100 <--> 100	[34, [0, 100, 0, 0, 0, 0, 0, 0]
	B2	int8	Tilt Up/Down	-100 <--> 100	[34, [0, 0, 100, 0, 0, 0, 0, 0]
	B3	int8	Claw grip/release	-100 <--> 100	[34, [0, 0, 0, 100, 0, 0, 0, 0]
	B4		-		
	B5		-		
	B6		-		
	B7		-		

Figure 3.4: CAN ID table of controller IDs and its supposed maneuvers.

Only the four first bytes in both packets were configured to send steering data. This is the result of a combined interest in a simple and effective steering system, which allowed for better control of both the ROV and the Manipulator. It still granted possibilities for further implementation of features, and this will be discussed in more detail in Chapter 6.3.3.

To use the packed values from **get_events_loop**, which is a function in a different file named **Controller_handler**, we used the following function:

Code 3.13: get_from_queue is a function which gets the data from a queue and sets the empty dictionary **self.data** equal to the packet. This is used when creating steering packets.

```python
def get_from_queue(self):
    """Takes data from the queue and sends it to the ...
        correct handler"""
    self.packet_id = -1
    packet = ""
    try:
        self.packet_id, packet = self.queue_for_rov.get()
    except Exception as e:
        return
    self.data = packet
```

ROV

To build the ROV packet, refer to both Figures [3.4] and [2.19], which include the specified controls and joysticks to be used and which indexes that correspond to the maneuvers. The packet created appended the ID for ROV steering data and the actual data, which consisted of joystick motion gathered from **Controller_handler** via a queue. This data was accessed by reading through the **self.data** dictionary containing all packed data, and using the key for ROV joystick values, as shown in Code [3.14]:

Code 3.14: A function that builds a packet to control the ROV.

```python
def build_rov_packet(self):
    if self.data == {}:
        return
    data = [0, 0, 0, 0, 0, 0, 0, 0]

    data[0] = self.data["rov_joysticks"][X_AXIS]
    data[1] = self.data["rov_joysticks"][Y_AXIS]
    data[2] = self.data["rov_joysticks"][Z_AXIS]
    data[3] = -self.data["rov_joysticks"][ROTATION_AXIS]

    self.packets_to_send.append([33, data])
```

Manipulator

The Manipulator packet was very similar to the ROV packet but used data from the controller assigned to the Manipulator instead. Referring to the same Figure, [3.4], but also Figure [2.20], it shows what joysticks and **dpad**-presses that responded to the given maneuvers, and also what indexes in the list of controller data to insert the values into.

Code 3.15: A function that builds a packet to control the Manipulator.

```
def build_manipulator_packet(self):
    if self.data == {}:
        return
    data = [0, 0, 0, 0, 0, 0, 0, 0]

    data[0] = self.data.get("mani_dpad", [0,0])[1]*100
    data[1] = ...
        self.data["mani_joysticks"][MANIPULATOR_ROTATION]
    data[2] = -self.data["mani_joysticks"][MANIPULATOR_TILT]
    data[3] = ...
        self.data["mani_joysticks"][MANIPULATOR_GRAB_RELEASE]

    self.packets_to_send.append([34, data])
```

3.3.3 Sending the steering data

The last step required for being able to actually move and steer the ROV and Manipulator is to send the steering data via the **Communication** group and have the **Control Systems** group process the incoming data and convert this to the physical movement of the motors. The communication part is explained in deeper detail in Chapter 3.1.

3.4 Threads and Processes with Threadwatcher

The **Threadwatcher** class was used to effectively monitor and manage which threads and processes were running and to stop them when needed. See Chapter 2.8 for more info on threads and processes.

An example of Threadwatcher usage can be seen in Code [3.16] where a thread gets added to the Threadwatcher class when we create a thread for the **rov_state_send_loop**. The Threadwatcher is being used the same way whenever we start a process.

Code 3.16: Threadwatcher being used to add and monitor a thread for sending packets to the ROV

```
id = t_watch.add_thread()
rov_state_send_loop = threading.Thread(
    target=rovstate.send_packets_to_rov, args=(t_watch, ...
        id), daemon=True
)
rov_state_send_loop.start()
```

After adding the thread, the Threadwatcher class has a function to check if the thread exists. If it exists, the next part of the Code [3.17] will run:

Code 3.17: Send to ROV

```
def send_packets_to_rov(self, t_watch: Threadwatcher, id):
    while t_watch.should_run(id):
        self.get_from_queue()
        self.build_packets()

        if self.packets_to_send != []:
            self.send_packets()
            self.data = {}
```

To stop one or multiple threads, Threadwatcher has the functions **stop_thread** and **stop_all_threads**.

Code 3.18: Stopping all threads that were added to Threadwatcher

```
def stop_all_threads(self):
    print("ThreadWatch: attempting to stop all threads")
    for i in range(len(self.threads)):
        self.stop_thread(i)
    print(self.threads)
```

3.5 GUI

To make a user-friendly interface for the pilots steering the ROV and Manipulator, certain factors needed to be satisfied. The factors included an easy-to-understand layout, straightforward information display, and styling.

3.5.1 Layout

The figure below shows the final version of the GUI, which went through several iterations. The previous iterations only gave instructions on what the finished GUI should contain because these did not meet all of our requirements for a user-friendly GUI (see Chapter 6.4.1). The final GUI had all the needed information displayed, as well as buttons, sliders, and warning alarms, while still being clear and obvious to new users.

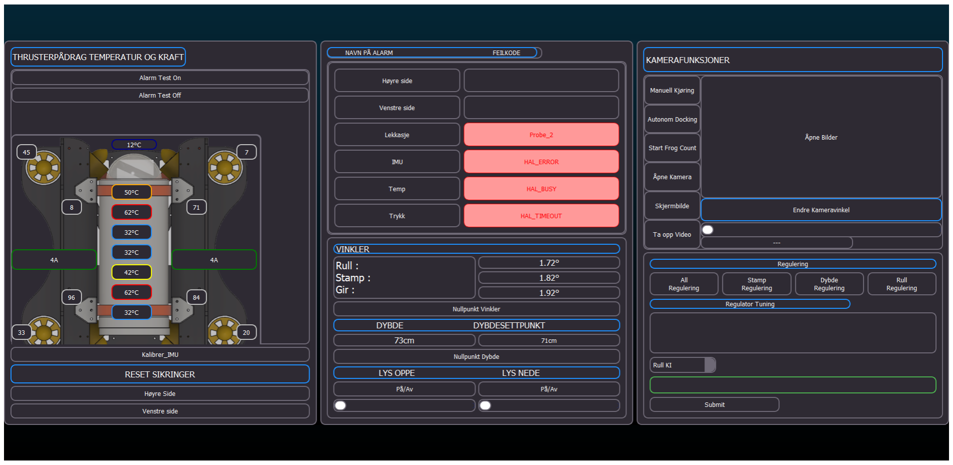


Figure 3.5: The final state of the GUI

Temperature, Thrusters, and Current Consumption

The GUI shows a visual representation of the eight thrusters on the ROV. The thrusters control the movement and maneuverability of the ROV.

There are a total of eight temperature sensors connected to the ROV. One of these monitors the water temperature, and can be seen at the top middle part of Figure [3.6]. The rest of the sensors monitor the internal temperature of the electronic housing.

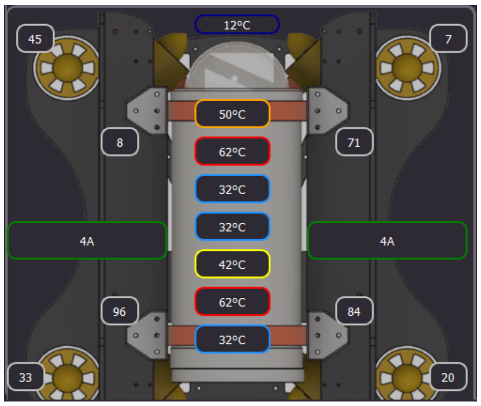


Figure 3.6: Temperature, thrusters, and current consumption. The sensors show a wide variety of temperatures with the use of fake data to present the different colors

The internal sensors start at lightblue, become yellow above 40, orange above 50, and red above 60. Additionally, there were two currents monitored, one on the left side and one on the right side, both in green.

Alarms & Error messages

To integrate a method for showing any alarming events, a box displaying eventual alarms was created. This was used as a reference to verify the integrity of the ROV while being operated. If, for example, a leakage was detected, an error message would pop up. Figure [3.7] depicts the box of alarms in the GUI.

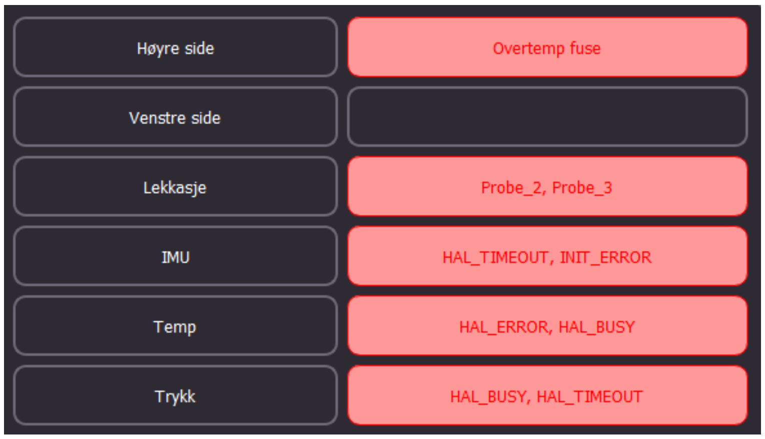


Figure 3.7: "Venstre Side" with no alarm, the rest have detected errors

The table below shows possible errors that could occur in case of an alarm trigger.

IMU Errors	Temp Errors	Trykk Errors	Lekkasje Errors
HAL_ERROR	HAL_ERROR	HAL_ERROR	Probe_1
HAL_BUSY	HAL_BUSY	HAL_BUSY	Probe_2
HAL_TIMEOUT	HAL_TIMEOUT	HAL_TIMEOUT	Probe_3
INIT_ERROR			Probe_4
WHO_AM_I_ERROR			
MEMS_ERROR			
MAG_WHO_AM_I_ERROR			

Table 3.1: Error Codes Table for IMU, Temperature, Pressure, and Leakage Errors

Whenever there is an error, their indices are stored in a set, and their corresponding string error messages get added to a list which then gets shown in the GUI. When the errors are resolved, the sensor indices get removed from the set, and the error messages are removed from the list which then removes the text from the GUI and resets the styling.

Code 3.19: Code for keeping track of alarms

```python
self.currentLekkasjeAlarms = set() #Set for keeping track ...
    of alarms
lekkasjeErrors = [  #Error messages for the leak sensor
    "Probe_1",
    "Probe_2",
    "Probe_3",
    "Probe_4",
]
alarmTextsLekkasje = [] #List with strings to update the ...
    GUI with new alarms
for i in range(len(sensordata[3])):
    if sensordata[3][i] == True:
        self.currentLekkasjeAlarms.add(i)
        alarmTextsLekkasje.append(lekkasjeErrors[i])
        self.play_sound(True)
    elif sensordata[3][i] == False and i in ...
        self.currentLekkasjeAlarms:
        self.currentLekkasjeAlarms.remove(i)
        self.play_sound(False)
if alarmTextsLekkasje:
    labelLekkasjeAlarm.setText(", ".join(alarmTextsLekkasje))
    labelLekkasjeAlarm.setStyleSheet(self.errorGradient)
else:
    labelLekkasjeAlarm.setText("")
    labelLekkasjeAlarm.setStyleSheet("")  # Reset style
```

Buttons & Functions

There were many buttons integrated into the GUI, and all of these had their own function. Functions such as toggling lights, opening cameras, and changing the values of specific regulation parameters, were some of the integrated functions in the GUI.

The buttons added to the GUI were decided by UiS Subsea as a whole to determine what functionality the GUI should contain. The **Control Systems** group needed buttons to support their regulation of the ROV. A camera tilt slider was added to allow the **Communication** groups camera to move. Buttons to open camera video feeds, take screenshots, perform autonomous docking, and more were implemented for the **Machine Vision** group. In general, buttons were added to the GUI as different groups needed them.

Qwidget's full name	CANID	Description
reguleringDropdown	42	Choice of regulation tuning
tuningInput	42	Input til reguleringDropdown
btnRegTuning	42	Submit reguleringDropdown and tuningInput
slider_lys_forward	98	Change light value on front light
label_percentage_lys_forward	Local	Change % of light in the GUI
slider_lys_down	99	Change light value on downlight
label_percentage_lys_down	Local	Change % of light in the GUI
btnBunnLys	99	Turn on or off light for ROV
btnTopLys	98	Turn on or off light for ROV
btnNullpunktVinkler	66	reset roll,pitch and yaw
btnKalibrerIMU	66	Calibrate the IMU
btnManuell	Local	Start manual control of ROV
btnAutonom	Local	Start autonomous parking
btnFrogCount	Local	Start FrogCount
btnOpenCamera	Local	Open Camera
btnTakePic	Local	Take picture and save
btnRecord	Local	Record from camera
showNewWindow	Local	Open window that shows pictures taken
btnRegOn	32	Button for turning on all regulation
btnDybdeOn	32	Button for turning on depth control
btnStampOn	32	Button for turning pitch control
btnRullOn	32	Button for turning on roll control
btnTestSound	Local	Button for testing sound
btnStopSound	Local	Button for stop testing sound
SliderCamVinkel	200	Slider for changing the camera angle

Table 3.2: A table that shows which buttons send data to the ROV, or start functions locally

Above are all of the functions that had to be implemented to support every feature added to the ROV. These functions required buttons, sliders, and input fields in the GUI.

3.5.2 GStreamer

When it was necessary to start the camera before the **Machine Vision** group had implemented its functions, a direct pipeline was established for each of the cameras using GStreamer and Python.

Code 3.20: Example code

```
def create_pipeline(multicast_group, port):
    return Gst.parse_launch(
        f"udpsrc multicast-group={multicast_group} ...
            auto-multicast=true port={port} ! ...
            application/x-rtp, media=video, ...
            clock-rate=90000, encoding-name=H264, ...
            payload=96 ! rtph264depay ! h264parse ! ...
            decodebin ! videoconvert ! autovideosink ...
            sync=false"
    )
```

The function incorporates a multicast group that decides the IP address for our connection and the corresponding port representing the camera that would get connected.

GStreamer is heavily reliant on threads and therefore the cameras ran in their own thread, as seen in Code [3.21]. [28]

Code 3.21: Code to run camera 1

```
camera1_info = ("224.1.1.1", 5000)
    )
thread1 = Thread(target=run_camera_stream, args=camera1_info)
thread1.start()
```

3.5.3 Sending data to the GUI

To allow the GUI to get the data from the ROV, a **multiprocessing.Queue** was used. This schedules a first-in, first-out (FIFO) queue, which means that the items are retrieved in the same order they were inserted. The first items added to the queue will be the first items received, which is perfect for updating a GUI. [59]

Code 3.22: Sending data to GUI

```
def send_sensordata_to_gui(self, data):
    # Sends sensordata to the gui
    self.gui_queue.put(data)
```

Qwidget's full name	CANID	Description
labelIMUAlarm	140	Get Alarm for ROV
labelLekkasjeAlarm	140	Get Alarm for ROV
labelTempAlarm	140	Get Alarm for ROV
labelTrykkAlarm	140	Get Alarm for ROV
labelRull	138	update Roll value
labelStamp	138	Update pitch value
labelGir	138	Update Yaw value
labelDybde	139	Update Depth value
labelTempSensorkort	139	Sensorcard Temperature
labelTempVann	139	Temperature in water
labelHHF	129	Thruster
labelHHB	129	Thruster
labelHVB	129	Thruster
labelHVF	129	Thruster
labelVHF	129	Thruster
labelVHB	129	Thruster
labelVVB	129	Thruster
labelVVF	129	Thruster
labelManipulatorKraft	150	Manipulator12V cards power
labelManpulatorTemp	150	Manipulator12V cards temperature
ManipulatorSikring	150	Status Manipulator12V cards fuse
Thruster12VTemp	151	Thruster12V cards temperature
ThrusterSikring	151	Status Thruster12V cards fuse
ReguleringTemp	130	Regulation cards temperature
MotorTemp	130	EngineControllercards temperature
labelDybdeSettpunkt	130	Start depth for depth regulation
TempKomKontroller	145	CommunicationController Cards Temperature
labelKraft5VTemp	152	Kraft5V cards Temperature
labelKameraVinkel	Local	Label that shows the current camera angle
lastSent	Local	Label that shows the last Regulation message

Table 3.3: A table that shows what in the GUI got values from the ROV

Table [3.3] contains all of the information being sent from subside to the GUI. Every part has its own placement in the GUI, as depicted in Figure [3.5].

3.5.4 Updating the GUI

As mentioned in Chapter 3.1, the data is received as a dictionary. **update_gui_data** gets data from the queue and sends it to the **decideGUIUpdate** function. To decide what data gets updated, the function checks the key from the dictionary and calls the corresponding function.

Code 3.23: Deciding which data to be displayed in the GUI

```python
def update_gui_data(self):
    while not self.gui_queue.empty():
        sensordata = self.gui_queue.get()
        self.decideGUIUpdate(sensordata)

def decideGUIUpdate(self, sensordata):
    self.sensor_update_function = {
        VINKLER: self.guiVinkelUpdate,
        DYBDETEMP: self.dybdeTempUpdate,
        FEILKODE: self.guiFeilKodeUpdate,
        THRUST: self.guiThrustUpdate,
        MANIPULATOR12V: self.guiManipulatorUpdate,
        THRUSTER12V: self.thruster12VUpdate,
        KRAFT5V: self.kraft5VUpdate,
        REGULERINGMOTORTEMP: self.reguleringMotorTempUpdate,
        TEMPKOMKONTROLLER: self.TempKomKontrollerUpdate,
    }
    for key in sensordata.keys():
        if key in self.sensor_update_function:
            self.sensor_update_function[key](sensordata[key])
```

3.5.5 Sending data from GUI

To be able to send commands to the ROV via the GUI, functions creating the related data were connected to their corresponding buttons. Code [3.24] is an example of a button with its equivalent function **calibrate_IMU()**.

Code 3.24: Connecting a button with its function

```python
self.btnKalibrerIMU.clicked.connect(lambda: ...
    self.calibrate_IMU())
```

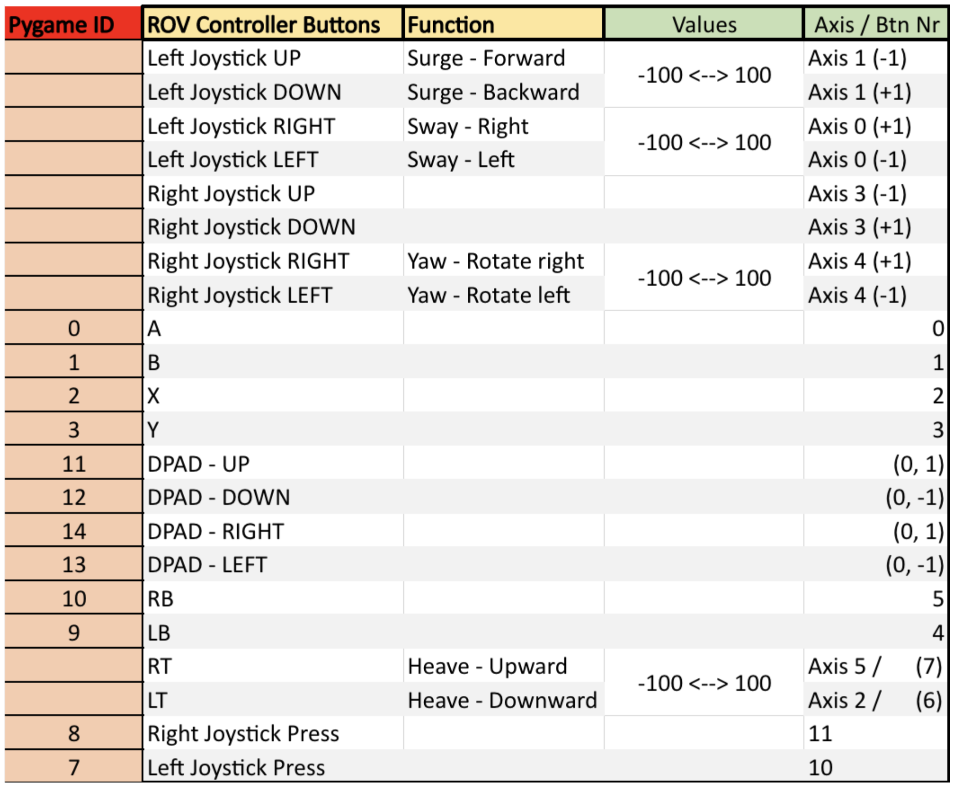
Chapter 4

Testing

Throughout the project, there were constantly things being tested, both local testing of small code implementations and larger tests with every component of the topside and subside connected. All of these tests were crucial for understanding the relations of the system and helped us learn more in-depth how everything worked. With this newly learned experience, we could further improve the code and fix potential bugs.

4.1 Local testing

Most of the local testing was conducted on the group's personal computers and consisted of testing small, specific implementations of the corresponding objectives of the member's distributed tasks. For example, testing the controller steering consisted of identifying all the button and joystick indexes (see Figure [4.1] below), verifying them, and then creating the actual functioning scripts for handling the controller data.



Pygame ID	ROV Controller Buttons	Function	Values	Axis / Btn Nr
	Left Joystick UP	Surge - Forward	-100 <--> 100	Axis 1 (-1)
	Left Joystick DOWN	Surge - Backward		Axis 1 (+1)
	Left Joystick RIGHT	Sway - Right	-100 <--> 100	Axis 0 (+1)
	Left Joystick LEFT	Sway - Left		Axis 0 (-1)
	Right Joystick UP			Axis 3 (-1)
	Right Joystick DOWN			Axis 3 (+1)
	Right Joystick RIGHT	Yaw - Rotate right	-100 <--> 100	Axis 4 (+1)
	Right Joystick LEFT	Yaw - Rotate left		Axis 4 (-1)
0	A			0
1	B			1
2	X			2
3	Y			3
11	DPAD - UP			(0, 1)
12	DPAD - DOWN			(0, -1)
14	DPAD - RIGHT			(0, 1)
13	DPAD - LEFT			(0, -1)
10	RB			5
9	LB			4
	RT	Heave - Upward	-100 <--> 100	Axis 5 / (7)
	LT	Heave - Downward		Axis 2 / (6)
8	Right Joystick Press			11
7	Left Joystick Press			10

Figure 4.1: Identified IDs, Joystick axes, and button numbers

A somewhat equal process was utilized for the other tasks, communication, and GUI, as well. When adding new additions to the system, they were always tested to ensure that it was still operational.

All the local testing resulted in the group having a working system to be further tested with the ROV and all its components, including sending and receiving the required packets and displaying the wanted information in the GUI.

In the end, when most local testing was completed, we could start testing with the other groups. The majority of this testing was done with the **Communication** group, **Control Systems** group, and **Machine Vision** group, but small, consistent testing was done with all groups.

4.2 ROV testing

The first testing with both the topside and the ROV began with using a personal computer. This later changed, as we bought a mini-PC to be used for the project and for the other groups to test at their own schedule.

To start testing, the mini-PC was connected to the ROV, or more specifically the electronic housing. As more circuit boards were completed and added to the electronic housing, they could be individually tested.

The first test conducted was to send and receive packets. This was done in collaboration with the **Communication** group, which had responsibility for the sending and receiving of packets.

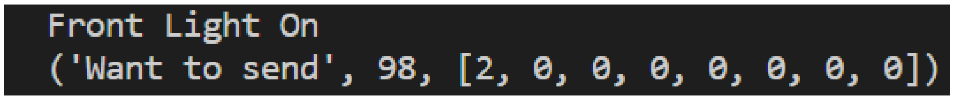
The GUI underwent several design changes, as seen in Chapter 5.2.1, but the displaying of data worked the same. Before all the circuit boards were integrated, a function with fake sensor data was created to see that the GUI worked as intended. The second iteration of GUI testing was done with the **Communication** group, where they sent fake sensor data from subside to be displayed, as in the first iteration of GUI testing.

When the **Control Systems** group integrated their circuit board into the electronic housing, tests were held to verify the controlling of the ROV thrusters. After some testing, the thrusters could be controlled by the ROV controller as planned (see Ch.3.3.2).

Lastly, the Manipulator's functionality was tested with the **Manipulator** and **Control Systems** group. This worked by sending steering data to the correct CAN-ID for Manipulator data, and then checking that the code for handling the steering data worked. The controller input being sent resulted in the movement of the different motors that corresponded to the given controller axes and buttons (see Figure [2.20]).

4.3 Communication testing

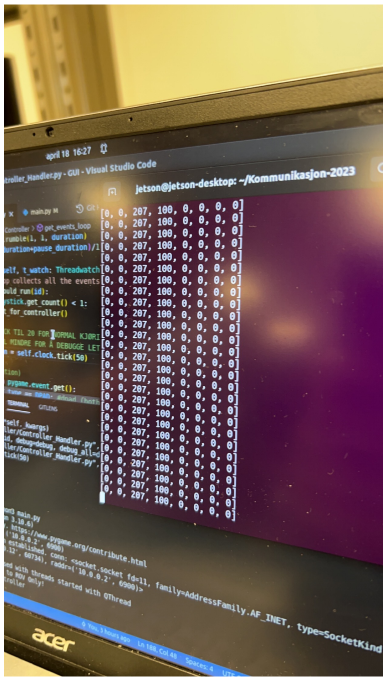
When working with the **Communication** group regarding sending and receiving of packets, several tests were conducted to verify that the packet sending was working as intended. To send one packet, several functions across several files were used, meaning that a good testing procedure was needed in order to verify where any potential error was. The first test was to create a print statement that checked whether the packet had been added to a list of packets or not. Figure [4.2] shows the result of the print statement.



```
Front Light On
('Want to send', 98, [2, 0, 0, 0, 0, 0, 0, 0])
```

Figure 4.2: Packet sent from the GUI

After the packet was added to a list, a second test was conducted which tested whether or not the packet was received on the CAN bus, as seen in Figure [4.3]. By using the**"candump -x any"** command on the CAN bus, it was checked if the sent packet was being converted to hexadecimal bytes. Lastly, different circuit boards were tested to verify the functions they were supposed to perform, such as turning on/off lights, activating different control words, etc.



```
jetson@jetson-desktop: ~/Kommunikasjon-2023
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
[0, 0, 207, 100, 0, 0, 0, 0]
```

Figure 4.3: Packets received on CAN bus

Chapter 5

Results

The results give an indicator of how well the planned integration of the system worked as a whole, but also what aspects worked well individually.

5.1 Water Test

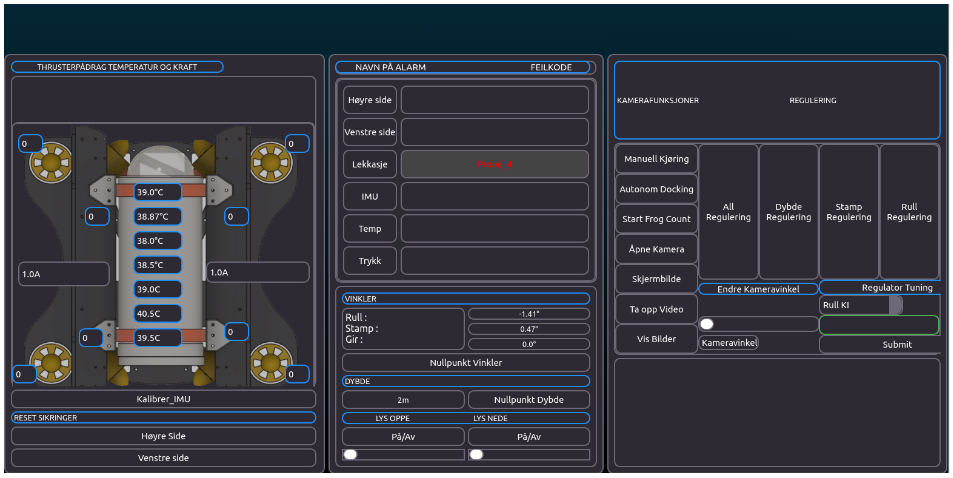
The best way to test the final product was to perform a **water test**. This involved assembling the ROV and dropping it in a pool to test all of its functionalities.

The water test proved successful in showing that all the parts of our system worked as planned. The pilots driving the ROV and controlling the Manipulator managed to easily maneuver it, with all functionalities working. The GUI showed the following:

- Sensor data sent from the ROV to the topside
- Throttle of the thrusters live over an illustration of the corresponding thrusters on the ROV (as seen on the left in Figure [3.5])
- Interactive sliders and parameters that got sent down to the ROV

Another demonstration of the ROV's ease of control was represented in the MATE demonstration video. The video was recorded to apply for the MATE competition by completing a series of tasks required to be able to qualify. The requirements included a time limit of 15 minutes to do the given tasks, and these were completed in less than 9 minutes by the pilots. The MATE demonstration video can be accessed in the conclusion Chapter (Ch.8).

The confirmation of successful alarms in the GUI was shown during the first water test. There was a leakage in the electronic housing due to a gasket not being tight enough, and this was displayed in the GUI as intended.



Lekkasje: Probe_4

39.0°C
38.87°C
38.0°C
38.5°C
39.0C
40.5C
39.5C

1.0A
1.0A

Rull : -1.41°
Stamp : 0.47°
Gir : 0.0°

2m

Figure 5.1: Leakage alarm going off in the GUI.

5.2 Topside

The topside results consist of how our system worked while water testing. This was a good way to check how every part of the system operated compared to our predictions and goals.

5.2.1 GUI

The graphical user interface worked as intended. Everything from running the program to starting the GUI, and finally stopping the program, did so without any errors. The GUI showed all the information that we wanted to display, which means that all the planned features, as listed in Chapter 1.2.1, were implemented and working.

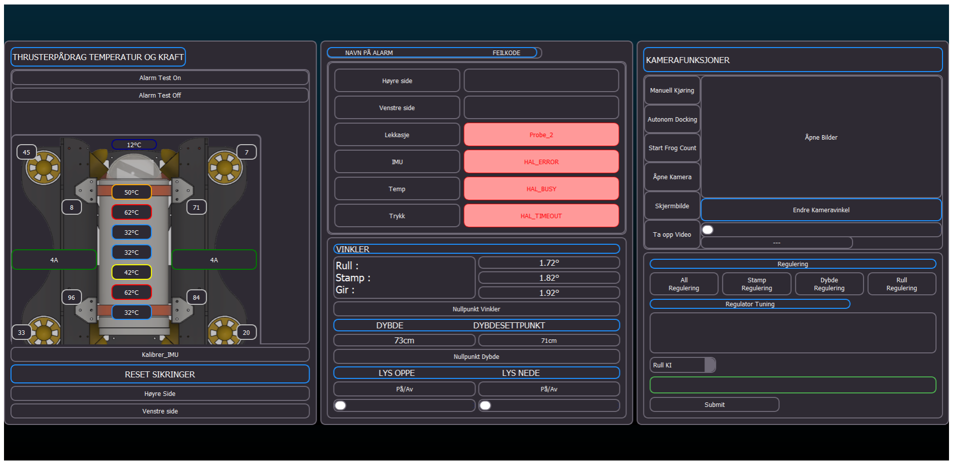


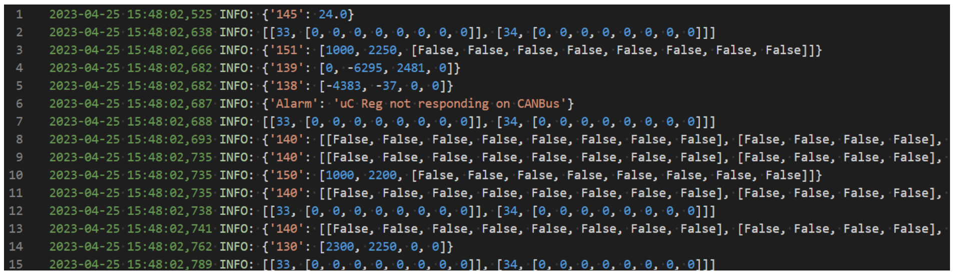
Figure 5.2: The final state of the GUI

5.2.2 Controllers

The steering of the ROV and Manipulator was fluid and intuitive. All functionality worked and gave the pilots an advantage when it came to completing the tasks required to compete in the MATE competition. Some additional features could be added or changed to further improve the controlling of the ROV and Manipulator, and this will be discussed in Chapter 6.3.3.

5.2.3 Data Logging

Logging of data worked well without any issues. The logging proved useful in several scenarios. The log files were mainly used by other groups to check the temperature levels, verify the feedback control system, or oversee packet activity. During the first water test, a small water leak occurred. The logger files were then used to check the timestamp for when the leak occurred and the amount of time it took to react to it.



```
2023-04-25 15:48:02,525 INFO: {'145': 24.0}
2023-04-25 15:48:02,638 INFO: [[33, [0, 0, 0, 0, 0, 0, 0, 0]], [34, [0, 0, 0, 0, 0, 0, 0, 0]]]
2023-04-25 15:48:02,666 INFO: {'151': [1000, 2250, [False, False, False, False, False, False, False, False]]}
2023-04-25 15:48:02,682 INFO: {'139': [0, -6295, 2481, 0]}
2023-04-25 15:48:02,682 INFO: {'138': [-4383, -37, 0, 0]}
2023-04-25 15:48:02,687 INFO: {'Alarm': 'uC Reg not responding on CANBus'}
2023-04-25 15:48:02,688 INFO: [[33, [0, 0, 0, 0, 0, 0, 0, 0]], [34, [0, 0, 0, 0, 0, 0, 0, 0]]]
2023-04-25 15:48:02,693 INFO: {'140': [[False, False, False, False, False, False, False, False], [False, False, False, False],
2023-04-25 15:48:02,735 INFO: {'140': [[False, False, False, False, False, False, False, False], [False, False, False, False],
2023-04-25 15:48:02,735 INFO: {'150': [1000, 2200, [False, False, False, False, False, False, False, False]]}
2023-04-25 15:48:02,735 INFO: {'140': [[False, False, False, False, False, False, False, False], [False, False, False, False],
2023-04-25 15:48:02,738 INFO: [[33, [0, 0, 0, 0, 0, 0, 0, 0]], [34, [0, 0, 0, 0, 0, 0, 0, 0]]]
2023-04-25 15:48:02,741 INFO: {'140': [[False, False, False, False, False, False, False, False], [False, False, False, False],
2023-04-25 15:48:02,762 INFO: {'130': [2300, 2250, 0, 0]}
2023-04-25 15:48:02,789 INFO: [[33, [0, 0, 0, 0, 0, 0, 0, 0]], [34, [0, 0, 0, 0, 0, 0, 0, 0]]]
```

Figure 5.3: Logger file structure

Figure [5.3] shows how the logger structure was set up. Each sent packet has its own timestamp, info level, packet id, and packet content.

Chapter 6

Discussion

6.1 Bottleneck

Certain parts of the project became significant deciders for component testing. The bottleneck consisted mostly of electronic and mechanical parts not being ready on the scheduled dates, which in turn resulted in the topside system not being able to test its functionalities. An easy solution to the bottleneck problem would be to schedule the testing between groups earlier, and in general, keep up good communication between the involved groups. This is not to say communication was bad, but rather that people could be kept more in the loop with each other's projects to better know when components would be ready. This would help mitigate the downtime in the middle of the project, where many of the topside features were ready to be tested, but the components of the ROV were not fully completed yet.

6.2 Communication Challenges & Solutions

A communication challenge encountered was sending packets to the CAN bus. Different solutions were tested, but only one worked. The implemented solution entailed four different functions across three different files just to send one packet.

In addition to the packet implementation in Figures [3.7, 3.8, 3.9], there was another function that uses the **Network_handler.py** file to finally send the packets to the CAN bus, as seen Code [6.1].

Code 6.1: Sending packets to CAN bus

```python
def send_packets_to_rov(self, t_watch: Threadwatcher, id):
    while t_watch.should_run(id):
        self.get_from_queue()

        self.build_packets()

        if self.packets_to_send != []:
            self.send_packets()
            self.data = {}
```

Even though this is a working solution, it could be considered a complex approach, which makes it harder to debug. Additionally, since every function that builds a packet is sent this way, we get several hundred lines of code, which makes it complicated to maintain the code in the long run.

6.3 Controller Challenges & Solutions

When implementing the code responsible for the controllers, there were several complications that appeared. When programming, it is not unusual to encounter errors and unexpected results. It is often about trial and error to achieve the best possible outcome.

6.3.1 Mac and Windows to Ubuntu

Much of the controller-related code was implemented on Mac OS. This turned out to create some challenges when testing between the topside and subside.

Firstly, when testing on Mac OS, there were 15 buttons on both controllers and six joystick axes, including the left and right triggers. The triggers had values between 0-1. This was not the case when testing on the Ubuntu operating system. This system did not recognize the **dpad** (directional pad) as buttons; instead, it represented them as tuples, as shown below:

DPAD UP	DPAD DOWN	DPAD LEFT	DPAD RIGHT
(0,1)	(0,-1)	(-1,0)	(1,0)

This was fixed by checking for dpad input instead of button indexes. The dpad values were packed with the key **mani_dpad** and were used to control the telescope of the Manipulator (see Ch.3.3.2).

In addition, two of the joystick axes were swapped in Ubuntu, compared to Mac OS. When testing the original working code in Ubuntu, some of the controller values returned unexpected outputs, and even outputted values when no joysticks were moved. Before knowing that this was the result of two swapped joystick axes, these results made no sense. After some debugging, it was discovered that joystick axis 2 and 4 were swapped. These correspond to the back left trigger and the left/right movement on the right joystick. The fix to this problem was simply to change the axis numbers affected by the Ubuntu changes.

Another surprising find when testing on Ubuntu was the fact that when connecting the controllers via Bluetooth, the joystick axes were recognized the same way Mac and Windows did, unlike how the cabled connection was recognized. So depending on the type of controller connection to the computer resulted in different values and indexes of some axes. A solution to this was proposed and consisted of creating a variable that could be set before running the program, saying if Bluetooth, Mac, or Windows is running, the joystick axes will swap between 2 and 4 accordingly:

Code 6.2: Boolean to fix swapped axes.

```
if Bluetooth_Mac_Windows_On == True:
    self.switch_2_4_axis = 4
else:
    self.switch_2_4_axis = 2
```

self.switch_2_4_axis was then used in the calculation of the virtual joystick axes (self.rov_joysticks[6] & self.mani_joysticks[6]) that combine the left and right trigger to control the **Heave** for the ROV and the **Claw** for the Manipulator:

Code 6.3: Joystick events with fix implemented.

```python
if event.type == JOYSTICK:
    if event.joy == ROV_CONTROLLER_ID:
        self.rov_joysticks[event.axis] = ...
            self.normalize_joysticks(event)
        self.rov_joysticks[6] = (self.rov_joysticks[5] - ...
            self.rov_joysticks[self.switch_2_4_axis])
    elif event.joy == MANIPULATOR_CONTROLLER_ID:
        self.mani_joysticks[event.axis] = ...
            self.normalize_joysticks(event)
        self.mani_joysticks[6] = (self.mani_joysticks[5] - ...
            self.mani_joysticks[self.switch_2_4_axis])
```

6.3.2 Initializing Controllers

The function responsible for initializing the controllers is an iteration of last year's **GUI group**, which originally made the **Controller** class for their project. This year UiS Subsea decided to use two separate controllers to control the ROV and Manipulator (Ch.2.6). A problem appeared quite early when implementing two-controller support, where it was random which controller was assigned to the ROV and Manipulator. This was because of Pygame's controller initialization. Both controllers get initialized at the same time, which resulted in a random order in the list of initialized controllers. The following table is a representation of this phenomenon:

First example initialize	Second example initialize
[Black_Controller, White_Controller]	[White_Controller, Black_Controller]

When assigning **rov_joystick** and **mani_joystick**, the controller first in the list will become the controller for the ROV, and the second for the Manipulator.

Code 6.4: First controller: ROV. Second controller: Manipulator

```
rov_joystick = pygame.joystick.Joystick(0)
mani_joystick = pygame.joystick.Joystick(1)
```

Since both controllers were of the same type, they had the same ID in Pygame, and this could not be used to differentiate them. There was no easy, fast fix for this issue, but there was a workaround. As long as the controllers were connected via cable, they could be distinguished by using two different cables, one of which should be faster than the other. Doing this made Pygame always initialize the controller with the fastest cable first, allowing for this to consistently be assigned to the ROV.

6.3.3 Future improvements for steering data

Changes and improvements could be made to the way steering data is sent, and how much data is sent. Pygame's **tick** method manages how often input from the controllers is updated.

Code 6.5: get_events_loop registers input depending on the tick-rate.

```
self.duration = self.clock.tick(20)
```

$$(1s/20hz) * 1000ms = 50ms$$

When the tick rate is set to 20, the function will wait until 50ms has gone by before checking for input and adding to the queue of data. Therefore, 20 times per second a packet with controller data will be added to the queue. The downside to this is that even though the list of data is empty, the packet will be sent, and will look like this:

```
data = [0,0,0,0,0,0,0,0]
```

An improvement to this could be to avoid sending empty packets, and rather decide that when no controller packets are sent, there is no movement. The only important thing to note, if this is to be done, is that there needs to be one packet that resets the values in the controller data. If this is not done, the last updated packet will contain some values, not equal to zero, and therefore have the subside keep responding to the latest packet values. A possible way to implement this would be to constrain packet sending to sending only when at least one value in the list of data is not equal to zero, and after releasing joysticks and buttons, reset these values by sending one packet looking like this:

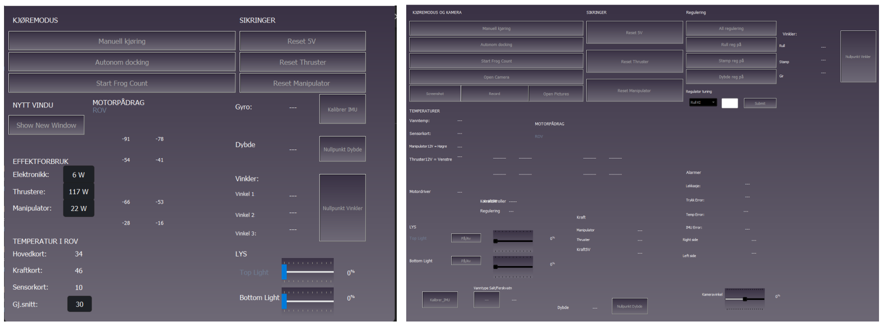
```
data = [0,0,0,0,0,0,0,0]
self.packets_to_send(33, data)
```

At most four of the indexes in the list of controller data contain values, which leaves the last four empty (as seen in Figure [3.4]). For future improvements, it would be possible to use these four empty spaces to send even more data from the topside to subside. This could include ROV maneuvers such as roll, pitch, or thruster throttling. All of these maneuvers were initially proposed to be implemented, but UiS Subsea chose to leave these ROV steering axes for regulation only. This meant that the pilots could not manually shift the ROV in these specific directions, but when applying regulation provided by the **Control Systems** group, these maneuvers could be controlled automatically. The decision to not include these steering functions manually was made to ensure a steady and slow approach to the steering of the ROV for better precision, rather than an unstable and fast approach.

6.4 GUI Challenges & Solutions

6.4.1 Iterations of the GUI

Because of constant new input of the required information to be displayed in the GUI, it went through many major styling and functionality changes. As the project advanced, more and more additions were added to the GUI, and therefore it needed to be modular to allow easy implementation of the new additions. The final iteration of the GUI is the combination of all the additions throughout the project and styling changes for a more user-friendly and self-explaining design. Figure [6.1] shows the first two iterations of the GUI. The final iteration is shown in Figure [5.2] in the results.



Elektronikk: 6 W
Thrustere: 117 W
Manipulator: 22 W

-91 -78
-54 -41
-66 -53
-28 -16

Hovedkort: 34
Kraftkort: 46
Sensorkort: 10
Gj.snitt: 30

Top Light 0%
Bottom Light 0%

(a) 1st iteration **(b)** 2nd iteration

Figure 6.1: Two versions of early GUI implementation

6.4.2 ROV data to GUI

Since the first tests on the GUI were made using dummy data, everything seemed to run fine when we sent the data to the GUI via a pipe. However, when we ran the code connected to the ROV, the GUI only updated the data related to the first received packet, leaving the rest empty. This led us to believe it was a **race condition**. A race condition is when the behavior of a program is affected by the timing and how multiple threads and processes execute their operations. [60] [61]

To fix this, the GUI updating system was changed into a queue. By introducing a queue, the synchronization issues with the pipe got fixed and the GUI ran smoothly. [59]

Playing sound

However, not everything in the GUI was fixed when adding a queue. When alarms first got implemented and went off, everything else stopped updating when the sound was playing. The solution to this was using QT's built-in QMediaPlayer. It operates in asynchronous mode, which means it does not block the main thread when playing sound. [62]

6.4.3 Updating Alarms

In the first implementation of the alarm system, only the most recent error message could be displayed and cleared in the GUI. This was because the system only saved the index of the last error, as shown in line 6 in Code [6.6]. Therefore, if multiple errors happened at once, the system only had the capability to display and clear the latest one. This led to older errors being overlooked until they were the "latest" error.

Code 6.6: Old alarm function

```
1  def guiManipulatorUpdate(self, sensordata):
2      for i in range(3):
3          if sensordata[2][i] == True:
4              labelSikring.setText(str(self.kraftFeilkoder[i]))
5              labelSikring.setStyleSheet(self.errorGradient)
6              self.lastManipulatorAlarm = i
7
8          if sensordata[2][i] == False and i == ...
              self.lastManipulatorAlarm:
9              labelSikring.setText("")
10             self.lastManipulatorAlarm = -1
```

This was fixed by using a **set** to store the indices. A set was primarily used for two reasons: it is unordered and does not allow duplicates. The unordered property is important because the order in which the alarms emerge does not matter, and we can easily remove them when necessary. The restriction on duplicates was also crucial in case multiple instances of the same alarm were accidentally stored. Additionally, a list was used to store multiple error messages as strings that could be shown in the GUI. This was useful because a list shows which order they came in.

6.5 General Challenges and Solutions

6.5.1 Earlier testing/implementation

Working together on the code integration with the **Machine Vision** group turned out to be tougher than expected. This was mostly due to different ways of structuring code and the fact that we had been working separately for too long. To avoid this, it would have been better to set a plan early on how the end project should be and try to combine the codebases earlier.

6.5.2 Subside PC to do image processing

The Nvidia Jetson, which was used in the ROV, is typically used for AI tasks similar to those carried out by the **Machine Vision** group. This would have been a great opportunity to run the image processing tasks subside instead of topside. It would reduce the workload on the main computer and potentially improve the camera feed by reducing the delay.

6.5.3 Time usage

When making the program that controls the ROV and Manipulator, there was extra responsibility added to our group. This responsibility consisted of steering and overlooking the GUI while running the system. This was because we had more experience and knowledge of the related parts, and therefore a lot of time went toward this.

Furthermore, UiS Subsea decided that members of the data groups should steer the ROV for the MATE competition. This was again because of the deeper understanding of the operational mechanics of the ROV. This required time, because practicing for the competition was an important element. The time spent practicing took time away from possibly adding more functionality or working on the thesis in general.

6.6 Continuation of the Subsea project

Every essential part of our system was working, but along the way, we discovered a few ways to improve the GUI and controller features. After the bachelor deadline, we will keep working on our system to optimize it even further. Possible future improvements will be listed in the next chapter (Ch. 6.7).

6.7 Summarized Future Improvements

Here is a shortened version of all of the possible future improvements and implementations for next year:

Communication:

- Simpler implementation of packet sending
- Unit tests to verify whether specific packets were received

Controller:

- Limit the controller data being sent
- Implement more functionality for the controllers

GUI:

- Toggle buttons with visual change
- 3D model of ROV movement
- Alarms displayed over the corresponding area of error
- Live graphs

6.7 Summarized Future Improvements

General:

- Earlier testing/implementation
- Fixing delay on the camera feed
- Subside PC to do image processing

Chapter 7

Financial overview & Environmental report

Financial resources involved in the project

A budget was given to every group in UiS Subsea, and the amount depended on the predicted need for each group as well as previously used financial resources from projects before. Typically, data groups tend not to need a large budget compared to other UiS Subsea groups, because the resources required to complete their part of the project only involve computers for coding and testing.

We had a budget of 2500 kr and ended up using 1498 kr to buy two brand-new controllers. Other than this, we used one monitor, a keyboard, a mouse, and a mini-PC bought this year by UiS Subsea.

The total cost of all data-related resources is listed below. These include every major component used and the price related. Most prices are exactly what was spent, but the prices for the monitor, keyboard, and mouse were gathered from equal products online, and are therefore predicted prices. Most of these resources did not contribute to the use of our own budget but from UiS Subsea's total budget.

Resource	**Price**
Asus Mini-PC	6 582 kr
Kingston SSD 512GB	909 kr
Kingston DDR4 16GB (RAM)	649 kr
Xbox Controllers	1498 kr
Monitor	1200 kr
Keyboard	300 kr
Mouse	200 kr
Speaker	299 kr
Total	**11 637 kr**

Smaller components, such as cables, were not included in the table above because these were owned by UiS Subsea, and have been for a long time. These expenses were therefore saved, as they were only borrowed from ourselves, and will be used in later projects as well.

7.1 Environmental Report

The theme of the MATE competition surrounds the importance of the UN's Sustainability goals (also mentioned in Ch.1.1.5). To support these intentions, it would be ideal not to waste material and resources, and re-use if possible.

7.1.1 Environmental impact

The new controllers and the components of the mini-PC were the only physical things that could have contributed to the impact on the environment. They were bought as an investment so that they can be used by other UiS Subsea projects in the future. Still, we could have used temporary solutions such as privately owned controllers borrowed only for this project and using one of our own PCs to run the system. This could have a smaller positive effect on the environment, but in the long run, the investments might prove efficient to alleviate resource spending in the future.

7.1 Environmental Report

The mini-PC and some of the components were bought separately, which meant that the environmental impact was greater than if they were all bought together.

Our system did not require much computational power to run, and since it was only supposed to run for a limited time to complete certain tasks, this did not have a large impact on the environment.

Chapter 8

Conclusion

To create a topside system, we had certain sub-tasks to complete. These sub-tasks are listed in Chapter 1.2. They gave a sense of direction as to what needed to be done, and further division of these sub-tasks (Ch.1.2.1) gave a more in-depth understanding of each specific task.

The proceeding sections will summarize if and how each sub-task was implemented, and to what extent.

Communication

Both sending and receiving data between the topside and subside worked well. The established connection made data flow fast and reliable. The logger improved our ability to oversee packets, enabling us to identify the potential problem at a more efficient rate.

Steering data

The controller data was accessed, gathered, and sent from the topside to subside without problems. Gathering the controller data worked well by first checking how many controllers were connected to the mini-PC, then, after having access to the input from the controllers, this data was successfully sent to the ROV, which in return made it move.

Graphical User Interface

The GUI went through several design iterations, but we were happy with the final result. The GUI showed all the necessary information: sensor data, video feed that could display any wanted camera feed, and all the buttons and sliders with functionality.

The conclusion to our project is that we completed all the planned tasks, and the topside system we created worked well.

Link for MATE video: **UiS-Subsea MATE Qualification Video**

Link for GUI representation: **GUI Video - UiS Subsea**

Bibliography

[1] N. S. Foundation, *About the mate center*, `https://www.marinetech.org/about/`, 2012.

[2] M. I. for Innovation, *About mate ii*, `https://mateii.org/about-mate-ii/`, 2023.

[3] M. I. for Innovation, *2023 explorer manual final 1 17 2023 withcover*, `https://files.materovcompetition.org/2023/2023_EXPLORER_Manual_FINAL_1_17_2023_withcover.pdf`.

[4] M. I. for Innovation, *Mate floats!* `https://materovcompetition.org/content/mate-floats`, 2023.

[5] R. Jehangir, *What is an underwater rov?* `https://bluerobotics.com/learn/what-is-an-rov/`, 2022 July 18th | Retrieved: 2023 February 9th.

[6] N. T. Centre, "Remotely operated vehicle (rov) services," 2003 October 1st | Retrieved 2023 February 9th.

[7] *Bluerov2*, `https://bluerobotics.com/store/rov/bluerov2/`, Jan. 2023.

[8] R. Solutions, *Underwater rovs*, `https://rovtechsolutions.com/products/underwater-rovs/`, 2019.

[9] OCEANEERING, *Rov systems*, `https://www.oceaneering.com/rov-services/rov-systems/`, Jan. 2023.

[10] *Equipment - Scanmudring %*, nb-NO. [Online]. Available: `https://scanmudring.no/equipment/` (visited on 05/13/2023).

[11] AUVAC, *Bpauv configuration*, `https://auvac.org/34-2/`, 2023.

[12] KYSTDESIGN, *Surveyor rov*, `https://kystdesign.no/rovs/surveyor/`, Jan. 2023.

[13] W. H. O. Institution, *Henry stommel*, https://www.whoi.edu/who-we-are/about-us/people/awards-recognition/henry-melson-stommel-medal/, 2023.

[14] G. O. B. array, *Float technology*, https://www.go-bgc.org/floats, 2023.

[15] Ecomagazine, *Robotic floats provide new look at ocean health and global carbon cycle*, https://www.ecomagazine.com/news/research/robotic-floats-provide-new-look-at-ocean-health-and-global-carbon-cycle, 2023.

[16] , *TCP/IP vs. OSI: What's the Difference Between them? | FS Community*, en, Blog, Nov. 2017. [Online]. Available: https://community.fs.com:7003/blog/tcpip-vs-osi-whats-the-difference-between-the-two-models.html (visited on 05/12/2023).

[17] *What is Transmission Control Protocol (TCP)?* en-us, Section: Computer Networks, Nov. 2021. [Online]. Available: https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/ (visited on 05/12/2023).

[18] *Client-Server Model*, en-us, Section: Web Technologies, Oct. 2019. [Online]. Available: https://www.geeksforgeeks.org/client-server-model/ (visited on 03/10/2023).

[19] *Three-Way Handshake - an overview | ScienceDirect Topics.* [Online]. Available: https://www.sciencedirect.com/topics/computer-science/three-way-handshake (visited on 05/12/2023).

[20] *User Datagram Protocol (UDP)*, en-us, Section: Computer Networks, Sep. 2017. [Online]. Available: https://www.geeksforgeeks.org/user-datagram-protocol-udp/ (visited on 05/12/2023).

[21] N. Jiju, *TCP/IP vs UDP: What's the Difference?* en-US, Dec. 2018. [Online]. Available: https://www.colocationamerica.com/blog/tcp-ip-vs-udp (visited on 05/12/2023).

[22] "An updated cost comparison of CAT 5 and fiber optic cable," en, *Library Systems Newsletter*, vol. 19, no. 10, pp. 75–76, Oct. 1999. [Online]. Available: https://librarytechnology.org/document/5963 (visited on 03/17/2023).

[23] *USB-Hub, USB-Stereo-Cameras, USB-Wifi*, en-US, May 2018. [Online]. Available: https://forum.digilent.com/topic/13106-usb-hub-usb-stereo-cameras-usb-wifi/ (visited on 05/12/2023).

[24] *Jetson Nano Developer Kit | NVIDIA Developer*. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (visited on 05/10/2023).

[25] *What is SSH (Secure Shell)? | SSH Academy*, en. [Online]. Available: https://www.ssh.com/academy/ssh (visited on 04/26/2023).

[26] *GStreamer GitHub mirrors*. [Online]. Available: https://github.com/GStreamer (visited on 05/11/2023).

[27] *What is GStreamer?* [Online]. Available: https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html?gi-language=c (visited on 05/11/2023).

[28] S. Chachra, *All you want, to get started with GStreamer in Python*, en, May 2022. [Online]. Available: https://sahilchachra.medium.com/all-you-want-to-get-started-with-gstreamer-in-python-2276d9ed548e (visited on 05/11/2023).

[29] git, *About - Git*. [Online]. Available: https://git-scm.com/about (visited on 05/08/2023).

[30] *Git - gitignore Documentation*. [Online]. Available: https://git-scm.com/docs/gitignore (visited on 05/08/2023).

[31] *Gitignore.io - Create Useful .gitignore Files For Your Project*. [Online]. Available: https://www.toptal.com/developers/gitignore (visited on 05/08/2023).

[32] GitHub, *GitHub Issues · Project planning for developers*, en. [Online]. Available: https://github.com/features/issues (visited on 05/14/2023).

[33] L. Rachelle, *What Is a Kanban Board?* en-US. [Online]. Available: https://www.planview.com/resources/guide/introduction-to-kanban/what-is-kanban-board/ (visited on 05/14/2023).

[34] *Python (programming language)*, en, Page Version ID: 1140647220, Feb. 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=1140647220 (visited on 03/12/2023).

[35] B. Wagner, *A tour of C# - Overview*, en-us, Feb. 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (visited on 03/10/2023).

[36] *C# / official or unofficial logo · Issue #27 · exercism/meta*, en. [Online]. Available: `https://github.com/exercism/meta/issues/27` (visited on 03/10/2023).

[37] QT, *About Qt - Qt Wiki*, Jul. 2022. [Online]. Available: `https://wiki.qt.io/About_Qt` (visited on 05/08/2023).

[38] A. Solovev, *Using Qt: 10 Famous and Successful Cases | HackerNoon*, en. [Online]. Available: `https://hackernoon.com/using-qt-10-famous-and-successful-cases` (visited on 03/12/2023).

[39] *The Qt Company logo vector download for free*, en, Nov. 2020. [Online]. Available: `https://www.logosvgpng.com/the-qt-company-logo-vector/` (visited on 05/08/2023).

[40] u. Riverbank Computing, *Riverbank Computing | Introduction*. [Online]. Available: `https://riverbankcomputing.com/software/pyqt/` (visited on 03/12/2023).

[41] M. F. L. u. P. G. s. w. PyQt5, *Create your first Python GUI with PyQt5 — A simple Hello world app*, en-us, May 2019. [Online]. Available: `https://www.pythonguis.com/tutorials/creating-your-first-pyqt-window/` (visited on 05/07/2023).

[42] T. Q. C. Ltd, *Using .ui files from Designer or QtCreator with QUiLoader and pyside6-uic - Qt for Python*. [Online]. Available: `https://doc.qt.io/qtforpython-6/tutorials/basictutorial/uifiles.html` (visited on 05/07/2023).

[43] T. Q. C. Ltd, *The Style Sheet Syntax | Qt Widgets 5.15.13*. [Online]. Available: `https://doc.qt.io/qt-5/stylesheet-syntax.html` (visited on 05/08/2023).

[44] T. Q. C. Ltd, *Qt Style Sheets Reference | Qt Widgets 6.5.0*. [Online]. Available: `https://doc.qt.io/qt-6/stylesheet-reference.html` (visited on 05/07/2023).

[45] Pygame, *About - pygame wiki*. [Online]. Available: `https://www.pygame.org/wiki/about` (visited on 04/16/2023).

[46] *Simple DirectMedia Layer - Homepage*. [Online]. Available: `http://www.libsdl.org/` (visited on 04/16/2023).

[47] *Gamepad Tester - Check Controllers and Joysticks Online*. [Online]. Available: `https://gamepad-tester.com/` (visited on 05/12/2023).

[48] *Introducing Windows 11 – Press materials for Windows 11 news announcement.* [Online]. Available: https://news.microsoft.com/june-24-2021/ (visited on 05/08/2023).

[49] D. Kumar Kandakatla, *Windows 11 - release information*, en-us, Apr. 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows/release-health/windows11-release-information (visited on 05/07/2023).

[50] *Install WSL | Microsoft Learn.* [Online]. Available: https://learn.microsoft.com/en-us/windows/wsl/install (visited on 05/08/2023).

[51] *Ubuntu Logo PNG Vector (AI) Free Download*, en. [Online]. Available: https://seeklogo.com/vector-logo/262530/ubuntu (visited on 05/08/2023).

[52] , *About Qt - Qt Wiki.* [Online]. Available: https://wiki.qt.io/About_Qt (visited on 03/12/2023).

[53] *Python Multiprocessing: The Complete Guide*, en-US, Jun. 2022. [Online]. Available: https://superfastpython.com/multiprocessing-in-python/ (visited on 05/14/2023).

[54] *Multiprocessing — Process-based parallelism.* [Online]. Available: https://docs.python.org/3/library/multiprocessing.html (visited on 04/28/2023).

[55] *Threading — Thread-based parallelism.* [Online]. Available: https://docs.python.org/3/library/threading.html (visited on 05/14/2023).

[56] *IBM Documentation*, en-US, Jan. 2023. [Online]. Available: https://www.ibm.com/docs/en/powerha-aix/7.2?topic=heartbeating-over-tcpip-storage-area-networks (visited on 05/12/2023).

[57] *What is CAN bus and why is it so important?* en-US, Oct. 2022. [Online]. Available: https://www.onlogic.com/company/io-hub/what-is-can-bus/ (visited on 05/08/2023).

[58] *Calibration and Characterization of IMUs · VectorNav*, en. [Online]. Available: https://www.vectornav.com/resources/inertial-navigation-primer/specifications--and--error-budgets/specs-imucal (visited on 05/08/2023).

[59] *Multiprocessing Queue in Python*, en-US, May 2022. [Online]. Available: https://superfastpython.com/multiprocessing-queue-in-python/ (visited on 05/11/2023).

[60] baeldung, *What Is a Race Condition? | Baeldung on Computer Science*, en-US, Jul. 2020. [Online]. Available: `https://www.baeldung.com/cs/race-conditions` (visited on 05/11/2023).

[61] *Pipe*, en-US, Sep. 2011. [Online]. Available: `https://www.techopedia.com/definition/3410/pipe` (visited on 05/11/2023).

[62] Q. Company, *QMediaPlayer — Qt for Python*. [Online]. Available: `https://doc.qt.io/qtforpython-5/PySide2/QtMultimedia/QMediaPlayer.html` (visited on 05/13/2023).

[63] OpenAI, *ChatGPT*. [Online]. Available: `https://chat.openai.com` (visited on 05/14/2023).

Attachments A

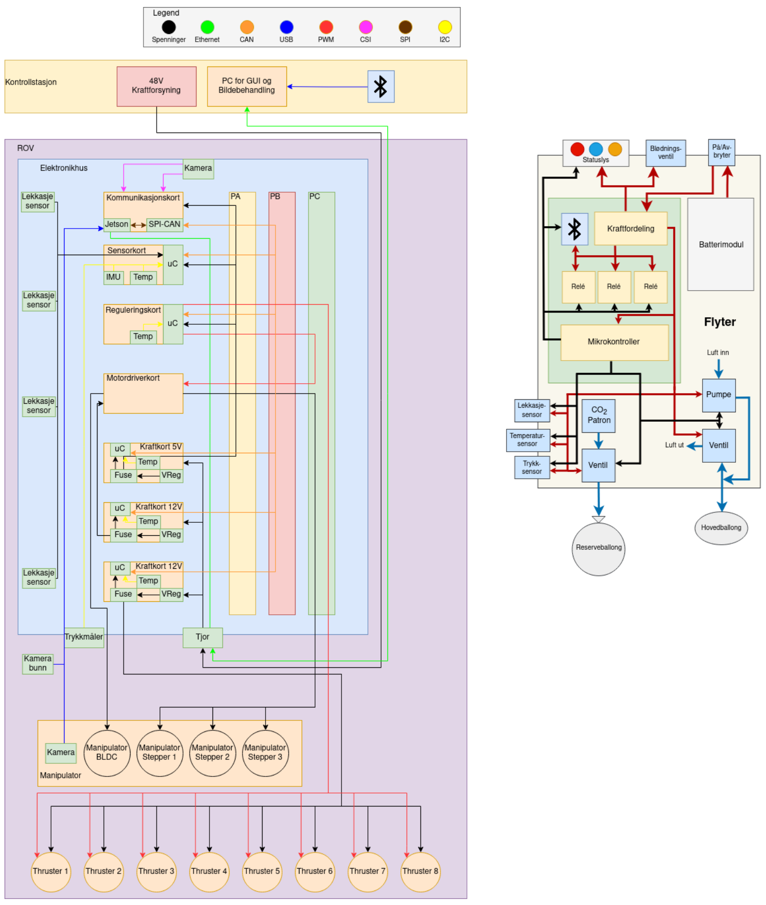


Figure A.1: Executive block diagram of the ROV