Tony Paulsen
Petter Henriksen

Development of ROV for aquaculture inspection platform

Bachelor's thesis in Automation
May 2022

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Kunnskap for en bedre verden

Tony Paulsen
Petter Henriksen

Development of ROV for aquaculture inspection platform

Bachelor's thesis in Automation
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

NTNU
Kunnskap for en bedre verden

Development of ROV for aquaculture inspection platform

Tony Paulsen

Petter Henriksen

May 2022

IELEA2920 / BACHELOR THESIS

Department of ICT and Natural Sciences

Norwegian University of Science and Technology

Supervisor 1: Ottar Laurits Osen

Supervisor 2: Lars Christian Gansel

Preface

This bachelor thesis is written by two students from automation engineering at NTNU Ålesund. The students in the group have similar backgrounds within technology, however some variation in subjects studied during the degree.

For this thesis, we wanted to create a new Remotely Operated Underwater Vehicle, ROV, for the Aquaculture Inspection Platform, AIP. We wanted to expand upon previous groups projects, by continuing solutions that worked well, such as the thrust-vector configuration and the general architecture of electronics. We rewrote the entire software, by doing this we had better control of all the subsystems of the ROV.

The main functionality that we added for our ROV was the higher depth rating, the ROV can dive and work as intended at 60 meters+ below surface. The ROV is much lighter and mobile than previous iterations. The ROV also has more measurement capabilities, that are highly relevant for aquaculture inspection. A maneuvering assisting system was implemented, which prevents the ROV from colliding with underwater structures, that in turn supports the operator of the ROV in challenging underwater areas.

We recommend that the reader of this report has a basic understanding of mechanical, electrical and software engineering to be able to fully understand the content of this bachelor thesis.

Acknowledgement

We would like to thank everyone who helped us during the project and especially:

- Our mentors Ottar L. Osen and Lars Gansel for guidance throughout the project.
- Family and friends for supporting us throughout the project.
- Laboratory engineers Anders Sætermoen and Øyvind Andre Hanken for helping with ordering parts and supplying tools.

Summary and Conclusions

This projects aims to create a new and improved prototype for a ROV that should later be integrated into the aquaculture inspection platform. Our development used experiences and good solutions from previous ROV projects, this included the general physical structure and electrical architecture. The goals of the project was to create a brand new software system, that among other things performs communication between the sub-systems in an efficient way. Additionally, functionality as collision avoidance systems was integrated. For physical improvements, the system should be lighter, more modular and able to dive to at least 60 meters depths.

The results suggests that the software was stable during operation of the ROV. By completing gradually more demanding tests, first from under controlled environments and later with sea trials, we adjusted the systems to work under variable conditions. Most of the implemented functionalities worked as intended, however the video-streaming implementation did not deliver as originally hoped, however potential solutions to these problems were suggested. In summary, the developed ROV has good solutions for most of the new functionality. But still, there is some key features that are not working as optimally as originally intended.

Contents

Terminology

PID Proportional integral derivative controller

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

API Application Programming Interface, activates functions from a remote software

TCP Transmission Control Protocol, connection oriented transmission protocol of information.

UDP User Datagram Protocol, non connection based transmission protocol of information.

IP Internet Protocol is a "best effort" delivery protocol

Notation

C Degrees Celsius

M Meters

V Volt

DC Direct current

AC Alternating current

A Ampere

F Farad

Kg Kilogram

ACK Acknowledgement message

GND Ground in electronic circuits

Abbreviations

IEEE Institute of Electrical and Electronic Engineers

I2C Inter-Integrated Circuit

Gnd Ground in electronic circuits

DOF Degrees of Freedom, number of unique directions an object can move

GPIO General purpose input/output

RPi Raspberry Pi

SONAR Sound Navigation and Ranging

JSON Sound Navigation and Ranging

ESC Electronic speed controller

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

This project aims to create a brand new prototype for a remotely operated underwater vehicle, *ROV*, for the Aquaculture Inspection Platform, *AIP*. The ROV is made to be used as a part of a larger system consisting of an unmanned surface vehicle and winch. The AIP will be remotely controlled by a operator, where it will be directed to the area that needs inspection. When it reaches its destination the winch lowers the ROV into the water. The ROV will provide the operator with a visual feed from a camera and a selection of data from multiple sensors mounted on it.

1.2 Problem formulation

This project aims to develop a brand new prototype ROV. The goal is to create a improved version of the existing prototype. The main focus will be on creating a ROV that is lighter, rated for higher depths, easier to use, more modular and with more sensors than the previous design. This prototype will also serve as a base for future development on *AIP*.

1.3 Objectives

The main requirement for this project was to make a functional prototype. The prototype needed to carry a few different sensors, a camera and it needed to be able to dive 60 meters below surface. Since the ROV needed to be built from the ground up a new software had to be created from scratch,

1. Build new ROV body
2. Implement Aanderaa sensor and sonar
3. Create a brand new GUI with more functions
4. Implement new camera functions
5. Create and integrate a collision avoidance system
6. Test the ROV in the sea and try to dive to 60m

1.4 Structure of the Report

The rest of the report is structured as follows.

Chapter 2 - Theoretical basis: Chapter two gives an introduction to the theoretical background for all aspects of the project.

Chapter 3 - Materials: Contains a description of the materials and software that were used in the project.

Chapter 4 - Methodology: Goes over the methods and solutions used for the project.

Chapter 5 - Results: This chapter goes over the test results.

Chapter 6 - Discussion: Various discussions about the results and the groups thoughts on the project.

Chapter 7 - Conclusions: This chapter present an overall conclusion for the project.

Chapter 2

Theoretical basis

This chapter goes over the theoretical background of the project.

2.1 Physics

2.1.1 Buoyancy

Buoyancy is a force that a liquid exerts on a object that it is immersed in. It is calculated with the formula $F_b = -pgV$ where F_b is the force of the buoyancy, p is the fluid density, g is acceleration due to gravity and V is fluid volume. For objects floating, sunken and in gases as well as liquids Archimedes principle can be stated as such: "Any object, wholly and partially immersed in a fluid, is buoyed up by a force equal to the weight of the fluid displaced by the object". If the force is positive the object floats because it is lighter than the fluid it is displacing and if it is negative the the object sinks due to it being heavier than the liquid that's being displaced [43].

2.2 Communication protocols

Communication protocols are descriptions for how digital information should be formatted and transported between devices [41]. These protocols are required to efficiently and reliably send information in computer systems [51]. There is a multitude of unique protocols, all with advantages and disadvantages when compared to one another. The majority of the sections under will briefly explain the working principle and key advantages for common communications protocols often used in industrial automation. However, firstly, a conceptual model for better understanding of how the protocols are implemented, known as the OSI-model, is explained.

2.2.1 OSI model

The *Open Systems Interconnection* model is a conceptual and systematic way of structuring the communication functions in computer systems [52]. Every layer in the model performs a function for the neighbouring layers. In total the model consists of 7 layers, as seen in Figure 2.2.

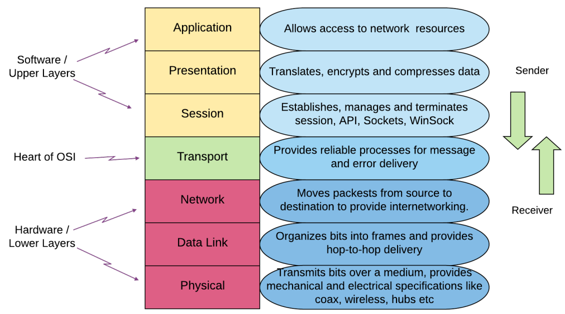


Figure 2.1: Angle of list.

The OSI model is a helpful tool for recognising what function specific protocols perform. That said, many of the protocols can be in multiple layers simultaneously, for example *ethernet* which is in both the physical and data link layer [53].

2.2.2 TCP

The *transmission control protocol* operates in the transport layer of the OSI-model. TCP is widely known as a reliable method of exchanging information, as it is reliant upon verifying packets [22]. In simple terms this is performed by sending *ACK* acknowledgment packets in response to received packets. If the sender of the original data packets did not receive *ACK*-packets, new packets are resent. This ensures that lost packets will be replaced, and no data is lost [22].

This protocol is widely used in many applications, and most programming languages and devices has an easy integration of TCP. Additionally the protocol is easily scalable and open-source

[29]. However, TCP has some drawbacks that is important to be aware of. As it is dependent upon constantly checking packets, and responding with *ACK* messages, a lot of bandwidth is used. Typically in applications that use a lot of data, this can be a bottleneck.

2.2.3 UDP

The *User Datagram Protocol* operates in the transport layer of the OSI-model. UDP is often considered an alternative to TCP, in certain applications [31]. In contrast to TCP, which sends packets and listens for responses. UDP sends datagrams and does not listen for acknowledgement messages that informs if the data has arrived correctly. UDP therefore does not have any guarantee that data arrives at the destination, this method is considered as *best-effort communications* [31].

For applications that require low usage of bandwidth and quick processing time, UDP is often superior to TCP. As the datagrams in UDP uses less bytes in overhead, in opposition to packets in TCP. This means that there is less information to process in relationship to relevant data. This can be very important in systems that are comprised of controllers and regulation logic.

2.2.4 Serial

Serial communication is a type of communication protocols that sends data one bit at a time, instead of sending data over multiple wires at the same time **??**. Serial communication is a widely used communication for small and simple components that do not require a lot of bandwidth. Some of the common serial protocols are USB, I2C and two-wire ethernet connections **??**.

2.2.5 I2C

The *Inter-Integrated Circuit* protocol is a type of serial communication and is commonly used for short-distance communication in simple circuits [23]. The protocol uses four wires, two of the wires are used for power supply, and the remaining for transmitting and receiving data. The communication wires are called *Serial data* (SDA) and *Serial clock* (SCL). The SCL signal transmits a clock signal that is used to synchronize and confirm the data bits sent by the SDA line [23].

Advantages of the I2C protocol is that it is relatively simple to program and set up, cost-efficiency and good error handling capabilities. However, main disadvantages is that speed is limited, as it is a half-duplex protocol. In addition, the protocol can not handle EMI when cable lengths are long.

2.3 Camera

2.3.1 Machine vision

Machine Vision is a term for all technology and methods used to extract information from a image. The task is automated and it can be used to get all sorts of data. Machine Vision can be applied to a single image, a set of images and videos since each frame of a video is a single image. There are many use cases for this technology like for example on assembly lines to filter out products that are not up to a set standard, it can also be used to for guidance systems in robots and it can also be used for monitoring people as a part of a security system [49].

2.3.2 Resolution and FPS

There are many things to take into consideration when working with cameras two of the most important are FPS and Resolution. FPS stands for frames per second and as the name suggests it tells how many images the camera captures in 1 second [47]. Resolution is a term that tells us how many pixels a image consists of. It is usually expressed as "width x height" so for example a 4K image has a resolution of 3840x2160 pixels. A image with a higher resolution will be able to display more details but the file will be bigger and therefore take up more data storage [44].

2.4 Aquaculture quality

2.4.1 Conductivity

Conductivity is a measure for how good a substance is able to conduct electricity. For liquids and electrolyte solutions, the SI unit *siemens per meter* is used [56]. This characteristic is highly relevant for aquaculture conditions, as it used to calculate values such as salinity. Additionally it can also used to find how much, and which types of dissolved elements the water contains [56].

2.4.2 Salinity

The amount of dissolved salt in a body of water, is known as salinity [57]. Salinity is either measured in gram/litre or gram/kg. Salinity in aquaculture applications is highly relevant as it an important factor in determining water quality and gives an estimation of different substances in water [57][30]. Specifically salinity can affect density of water, therefore water with salt concentrations will sink, and obstruct water flow, which could result in poor circulation of water in areas with high numbers of fish or other marine life [30].

Calculation

When calculating salinity, the term practical salinity is used, and it is measured in the dimensionless quantity g/kg. Practical salinity is an approximation of salt dissolved in water, however it is not interchangeable with absolute salinity, which is the true salinity level [57]. The equation for practical solution is dependant on temperature, pressure and conductivity.

2.5 Sonar

Sonar, *Sound navigation and ranging,* is a technique that uses sound propagation to measure distance, navigate and detect objects [55]. Sonar can be used both in air and in water, however usage in air is very limited as speed of sound is slow and gives inaccurate results, and since the development of superior technology such as radar gives better results [59] it is rarely used on land.

The working principle of sonar is based on that if speed of sound in water is known and time between outgoing generated sound signals and incoming reflected sound signals is controlled. The distance can be calculated with equation 2.1. Where d is the distance from measured object, $sigTimeRec$ and $sigTimeSent$ is the time when the sound signal was received and sent, respectively. Finally, $speedSound$ is the speed of sound under water.

$$d = \frac{(sigTimeRec - sigTimeSent) \cdot speedSound}{2} \tag{2.1}$$

Another important parameter of sonar technology is the *target-strength* of objects. This characteristic is used to determine the size, shape and type of objects [40][54]. The target strength is evaluated with equation 2.2. TS is the value for how much signal is measured based on how much signal was sent out, and is measured in decibels. δ_{bs} is the back-scattering cross-section. Where back-scattering is a measure of how much signal is reflected back from its origin [54]. Which helps determine the type of objects, as for example seabed reflects sound energy differently in comparison to fish.

$$TS = 10 \cdot \log(\frac{\delta_{bs}}{4\pi}) \tag{2.2}$$

In addition to the applications mentioned above, back-scatter is also relevant when compensating for false negatives and signal noise. Often sound signals will travel in many different paths to- and from an object, as is illustrated in Figure 2.2. The figure shows three unique paths for the signal to travel, although often there can be additional paths. One of these can arise if there is a quick temperature change in seawater based on depth, the signal sound can *bounce* off the warmer water level [55]. When signals take alternative paths, the time between sending and receiving is artificially inflated, and will incorrectly indicate that measured objects are further away than in reality. Back-scatter is used to combat this issue by comparing sound signals strength and type, and determining which returns signal gives the most realistic value.

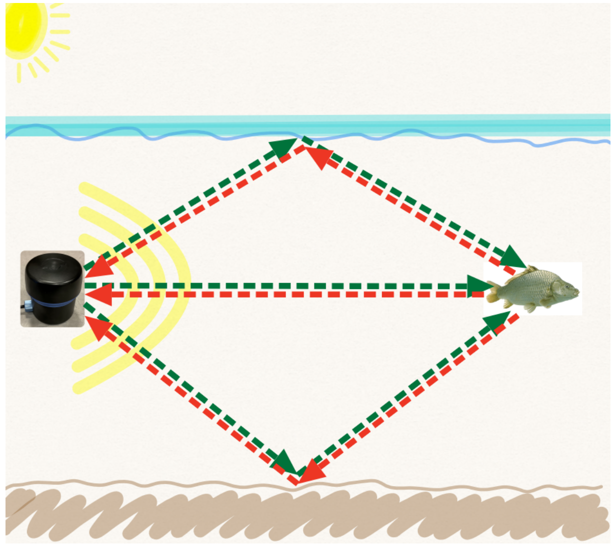


Figure 2.2: Sonar paths illustrated when detecting an underwater object.

2.5.1 Passive sonar

Passive is one of two main categories of sonar types. This sonar type exclusively listens for ambient noise, and interprets the signals to find usable information. However passive sonars often have clear limitations in term of performance. The sonar will often struggle with separating relevant noise information, such as other ships, with its own vessel's sound sources. This type of sonar is often used when the vessel that is trying to locate objects, does not want to reveal its own position, as in military applications. Or when the sonar signal can cause disturbances to underwater life, as in aquaculture applications.

2.5.2 Active sonar

Active sonar has the added capability of transmitting sound pulses itself. This is the most typical type of sonar, and is superior when locating objects quickly and often. This sonar often has a rotational build, where it rotates around its z-axis and outputting and reading sound signals.

2.6 Electrical

2.6.1 EMI

Electromagnetic interference is a disturbance on an electrical circuit by electromagnetic induction, electrostatic coupling or conduction [46]. Such disturbances can reduce the performance of a circuit, or in worst case, completely shut down the functionality. Often communication circuits are the most susceptible to such disturbances, as small changes in voltage levels can decide the value of a signal. EMI can arise from natural sources, such as solar flares and lightning. However, interference usually comes from other electrical components like frequency drives and transformers [19].

2.6.2 EMC

Electromagnetic compatibility describes the ability of electrical equipment and systems to function in EMI environments [45]. EMC includes the generation, propagation and reception of electromagnetic interference. Most electrical systems utilise a combination of the before-mentioned methods to function properly. A common technique to reduce generation and reception is to use cables with shields that are grounded at one side [32]. The shield will work as a drain for electric fields.

2.6.3 Power transmission

Transmission over long distances results in power loss. The main reason for this is that the resistance in the cables increases the longer the cables are. If an application is using a constant amount of current, but the length of cable is increased, the resistance will increase. And by Ohm's law, the voltage over the cable will increase, see equation 2.3.

$$V_{cable} = I \cdot R_{cable} \tag{2.3}$$

As the voltage drop over the cable increases, the voltage over the application components have to drop, as Kirchoff's voltage law states, see equation 2.4.

$$V_{application} = V_{source} - V_{cable} \tag{2.4}$$

Chapter 3

Materials

In this chapter you will get a overview of all the components and software that were used for this project.

3.1 Components

Raspberry Pi 4

The Raspberry Pi is a small computer made by the Raspberry Pi Foundation and is often used in robotics. Its around the size of a credit card and can run a lot of different Linux based operating systems. The model we are using for this project has 8GB of RAM and a 1.5 GHz Quad core processor. The I/O consists of 2 USB3 ports, 2 USB2 ports, 1 Gigabit ethernet port and 2 micro HDMI ports. It also has 40 GPIO pins that can be used to control everything from motors to LEDs [38].

Bar30 Depth/Pressure Sensor

The Bar30 is a waterproof pressure sensor made by BlueRobotics. The sensor itself is a Measurement Specialities MS5837-30BA and it can measure up to 30 bar with 0.2 mbar resolution. The sensor also measures temperature with a accuracy of +- 1 degree Celsius. It communicates using I2C and the operating voltage ranges from 3,3V to 5,5V [7].

Celsius Fast-Response Temp Sensor

The Celsius Fast-Response is a waterproof temperature sensor made by BlueRobotics. The sensor itself is a Measurement Specialities TSYS01 and it can measure temperature with 0,1°C resolution. The sensor communicates using I2C and the operating voltages ranges from 3,3V to 5,5V. It also has a fast response time with 1 second with water flow and 2 seconds without [16].

SOS Leak Sensor

The SOS Leak Sensor is a sensor made by BlueRobotics which uses sponge tipped probes to detect water leaks. The sensors operating voltage ranges from 3,3V to 5V [12].

T200 Thruster

The T200 is a underwater thruster made by BlueRobotics. It uses a brushless electric motor housed in a body made of durable polycarbonate plastic. The motor is three phase and requires a electronic speed controller to run it. The operating voltage ranges from 7 Volts to 20 Volts and at full power it can produce 65.83 N of forward thrust and 49.37 N of reverse thrust. The thruster is widely used in underwater robotics and can be run with a microcontroller like the Arudino and the Raspberry Pi [15].

STC-MCA503USB

The STC-MCA503USB is a small USB 3.0 camera made by Omron Sentech. It has a 5 megapixel sensor and a resolution of 2592 x 1944, it can record video at up to 14 fps. The camera uses the C-mount standard for lenses, it gets its power from a USB 3.1 cable which it also uses for communication. The camera has specific drivers made by Omron which are needed to make the camera work [39].

Fathom-X Tether Board

The Fathom-X is a product made by BlueRobotics. It allows you to run a Ethernet connection on only one pair in a standard CAT cable. It works by having one board on either end of the cable, it can be powered by USB or a 7-28V input. It has a max practical bandwidth of 80Mbps and it only consumes 5W of power. It is also fairly small which is advantageous when working in tight spaces [10].

ICR18650 battery

The ICR18650 is a Lithium Ion battery sold by Biltema. The battery produces 3,7V and can output up to 5,9A. It is rechargeable, and it has a max charging current of 4A and a max charging voltage of 4,2V [6].

MEAN WELL RSDW60F-15 DC/DC converter 60W 15V 5A

The RSDW60F-15 is a small DC to DC converter made by MEAN WELL. It can output 15V and a maximum of 4A. It can take a wide range of input voltages ranging from 9V to 36V. The output voltage can also be adjusted up or down by 10 percent so it means the converter can output voltages ranging from 13.5V to 16.5V. The converter has a max power output of 60W and can operate in temperatures from -40 C to 85 C. It also has built in safety systems against short circuits, overload, over voltage, over temperature and input under voltage lock out [21].

Murata UWE-12/6-Q48NB-C

The UWE-12/6-Q48NB-C is a a small isolated DC to DC converter made by Murata Power solutions. It outputs 12V and up to 6A, it can take inputs ranging from 18V to 75V. The converter can operate in temperatures from -40 C to +85 C [34].

Arduino UNO

The Arduino UNO is a microcontroller made by Arduino. It has a lot of different input and output pins both analog and digital. Some of the digtial pins can also make a PMW signals. The Arduino also has a power jack and a reset button. It has a recommended input voltage ranging from 7-12V but it can run on 6-20V but this is not recommended [2].

Basic ESC

The Basic ESC(Electronic Speed Controller) is a motor controller made by BlueRobotics, it is a improved version of a ESC made by BLHeli. The ESC allows you to control any three-phase brushless motors. It runs on 7-26V and can consume up to 30A [9].

Ping 360 Sonar

The Ping 360 Sonar is a mechanical sonar made by BlueRobotics. It runs on 11-25V and consumes at max 5W. The sonar is rated for depths of up to 300m and has a max range of 50m. It communicates via USB, Ethernet or RS-485 [14].

Lumen Subsea Light

The Lumen Subsea Light is a LED light made by BlueRobotics. It has a waterproof housing which is rated for depths of up to 500m. The LED has a peak brightness of 1500 lumen. It is dimmable and can be controlled by using a micro controller by sending PWM signals. It runs on 7-48V and has a peak current draw of Vin(Input Voltage)/15 [13].

Aanderaa Conductivity Sensor

The Aanderaa Conductivity Sensor 5819 is a small sensor made by Aanderaa. It measures temperature, conductivity and depth to calculate a estimate of the salinity in water. The sensor runs on 5-14V and consumes a maximum of 100mA. For communication it uses AiCaP, CANbus, RS-232 and RS-422 [1].

BlueRobotics Watertight Enclosure

The enclosure is made by BlueRobotics and is rated for 100m depths. It consists of a acrylic tube with two end caps that use double o-rings to ensure a water tight seal. For our configuration we used one domed acrylic end and one aluminum end with 14 holes [11].

3.2 Software

3.2.1 Pycharm

PyCharm is a program made by JetBrains and it is used to write Python code. The program has a lot of features which makes writing python code more intuitive [28].

3.2.2 CLion

CLion is a program made by JetBrains and it is used to write C and C++ code. The program has a lot of features which makes writing C and C++ code more intuitive [27].

3.2.3 Arduino IDE

Arduino IDE is a software made for programming Arudino microcomputers. IDE uses C and C++ programming language with a few modifications, the software is available on Windows, macOS and Linux [58].

3.2.4 Fusion 360

Fusion 360 is a 3D modeling CAD(Computer-aided design) software made by Autodesk, which can be used for designing and engineering products. Fusion is a cloud based software [5].

3.2.5 PC Schematic

PC Schematic is a program made for drawing electrical schematic diagrams [37].

3.2.6 Gantt

A Gantt chart is a popular way of illustrating a project schedule, it makes it easier to get a overview of what is actually being done in a project [48].

3.2.7 Raspberry PI OS

Raspberry PI OS is a GNU/Linux based operating system made by the Raspberry Pi Foundation. It is specifically made for use on all Raspberry Pi computers [50].

3.2.8 Cura

Cura is a free software made by Ultimaker allowing you to 3D print your 3D models with a Ultimaker 3D printer [42].

Chapter 4

Methodology

4.1 Project Organisation

The group consists of two bachelor students with a similar background. Both students study electrical engineering with a speciality of automation. However the students have taken some different subjects leading up to the bachelor thesis, this gives a broader knowledge basis for the project. Both members was assigned different positions within the group to ensure structure and good cooperation. The positions were project leader and secretary.

The project leader's responsibility was to ensure good time management and divide up the work tasks in a reasonable way. To ensure this the project leader had to update the Gantt-diagram regularly. In addition the project leader was tasked with organizing periodic meetings with the control-group, producing progress-reports and meeting notices every two weeks.

The secretaries responsibility was to keep a structured overview of the general progress. Additional tasks included reservation of eventual meeting location for the periodic meetings. And writing and distributing minutes of meetings after meetings to ensure good documentation of tasks and feedback.

4.2 Function testing equipment

To ensure that the components that were ordered were working properly and in a way that we had planned, it was determined that we were going to prepare provisional connections and testing scripts. These systems were simplified as much as possible, and further expanded with more advanced functions that we needed for our purposes. The sub-chapters below describes in more detail about how function testing for each respective components were performed.

4.2.1 Sonar

The initial test of the sonar was done by supplying the sonar with 12VDC externally, and connecting the USB-adapter into a computer. BlueRobotics the supplier of this sonar has created a program, *Ping-Viewer* [18], to easily test parameters and sensor readings. This program was downloaded and used. By watching the sonar circular and waterfall plots, we could see that the sonar was apparently working as intended. We adjusted parameters as length and gradient step length, to find first draft settings for the collision avoidance algorithm. Additionally the sonar was lowered into a small tub of water to get more realistic sensor values, however even at the shortest scan range, the sonar could not display any reasonable values. As the minimum scan range is $0.75\,[m]$, and the tub had dimensions of $0.5 \times 0.4\,[m]$.

After we had confirmation from previous tests that the sonar was functioning properly, implementation of the functionality in the RPi was the next step. The scanning imaging sonar has open-source libraries that has the general functionality to operate the sonar and read raw data. These libraries are available in multiple programming languages as Python, Arduino and C++. For our purposes the functionality of the testing script will be implemented in a Python script, as the finished main program will be run a Python system in our Raspberry Pi.

The before-mentioned libraries for Python was downloaded from the sonar suppliers official GitHub page [35] to the Raspberry Pi from source using the built in Bash terminal in the RPi. The commands were as follows:

```
$ git clone --single-branch --branch deployment
```

```
https://github.com/bluerobotics/ping-python.git
$ cd ping-python
$ python3 setup.py install --pi
```

The performed commands downloads the required libraries in a path that the assigned python interpreter can access. Specifically this is done in the final command by using *python3* in place of *python*. This is important as the solution we want to implement is dependant on using python version 3.7 or newer as the RPi has python 2.7 as the standard interpreter.

With these libraries it easy to communicate with the sonar from the python script. The sonar is initialized as an object, and writing new commands and reading values are done with class methods already implemented in the libraries. In conjunction with the libraries, the first logic was heavily inspired by another example found on GitHub by CentraleNantesRobotics [33]. This example sends new angles periodically, and reads the sonar reflections by using *OpenCV* plots and displays the readings in a circular plot. With this implementation we had sufficient working code examples to expand into the algorithms needed for our desired functionality in later implementations.

4.2.2 Camera

The Omron Sentech camera was first tested with Omron specific software, *StCamSWare*. This initial testing was simple, as it only required downloading of the camera driver, and the software application. Using this software, parameters as light sensitivity could easily be adjusted.

After confirmation that the camera worked as specified, we began implementation of reading the video feed in Python. Final system requirements requires us to open the camera in a Python file running on our RPi. However, from downloading the camera drivers for the windows PC previously, it was quicker to attempt using Python on the Windows computer. Launching the camera and displaying images was done using the *OpenCV* Python library. In Python we experimented with different compression techniques to reduce the image files from the camera, as communication from the RPi to the surface computer running the GUI was limited to 80 MBps [10].

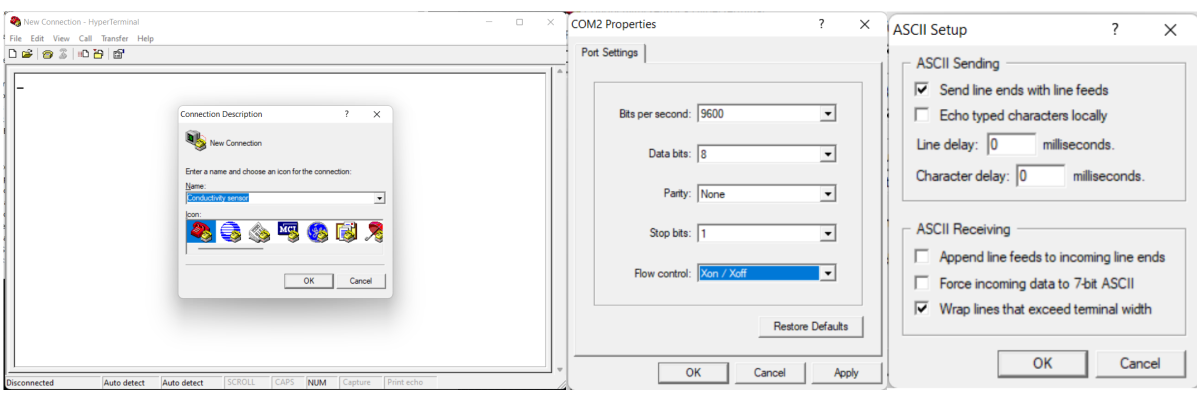
4.2.3 Combination sensor

To get sufficient knowledge regarding interacting with the conductivity sensor, we used the start-up guide provided in the sensors data-sheet. The guide instructed us to use a *terminal-emulator* tool to be able to interact with the sensor over serial communication, we used a free tutorial of *HyperTerminal* [24]. Furthermore the communication parameters were specified as seen in table 4.1.

Parameter	Value
Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	Xon/Xoff
ASCII sending	Active

Table 4.1: RS232 serial connection settings for communication with conductivity sensor from Aanderaa

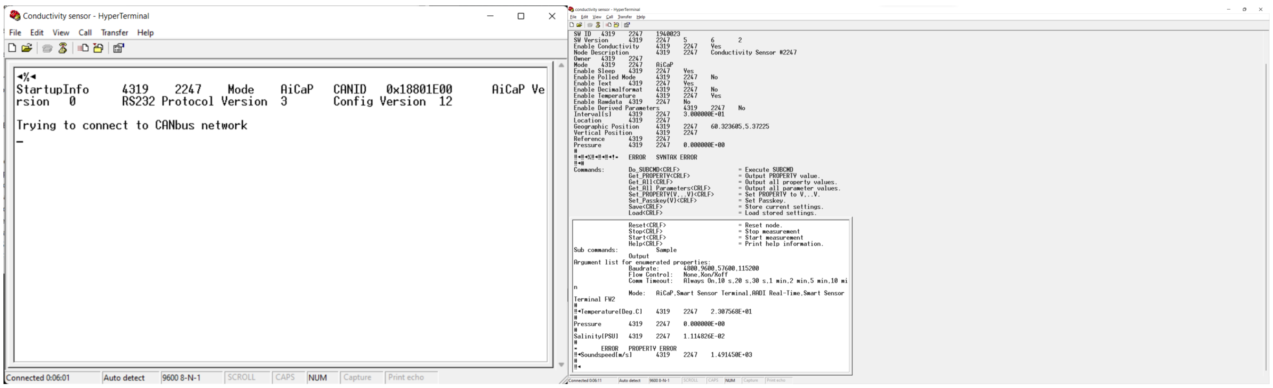
When launching *HyperTerminal* an initial prompt from the software is given as seen in Figure 4.1a, here we select an instance name that is used for saving the adjusted settings. Next the serial settings are specified for the communication, see Figure 4.1b and Figure 4.1c, which is done according to Table 4.1.



(a) Configuration: sensor type and name (b) General serial settings (c) ASCII settings

Figure 4.1: Initialization of serial communication with Aanderaa conductivity sensor using HyperTerminal software.

After the initial set-up is completed, the user interface menu is is displayed, as seen in Figure 4.2a. We can then send commands to the sensor by writing in ASCII. The possible commands are listed in the data sheet []. From activating different modes and sensor data gathering, a good understanding of how the conductivity sensor works was obtained. Figure 4.2b shows some responses from the sensor resulting from commands sent. From experimenting with multiple commands and reading resulting commands, we gained knowledge of how to structure the main program in Python at a later point.



(a) Initial serial communication interface

(b) User interface showing inputs and outputs from user and sensor, respectively

Figure 4.2: Command line interface with Aanderaa conductivity sensor over serial using HyperTerminal software.

4.2.4 Thrusters, I2C sensors and safety sensors

The thrusters and ESCs were first tested one by one on land to ensure that all the components were functioning as they should. After that the thrusters were tested in a water to measure and confirm the current draw at different engine speeds. During this testing we also found the limitations at which we could run them during the different stages of ROV operation. Furthermore from this testing, we tuned how long the Arduino should attempt to initialize the ESC, before running other functions like serial communications.

The sensors that communicate with I2C, the pressure sensor and the temperature sensor, were tested in a separate C++ script with the Arduino extension. This script was found on the BlueRobotics Github page [26][25]. As both sensors measures different values, we did not have to change the I2C address of either sensor. Finally the moisture detection sensor was simply tested by connecting the input to a digital pin, initializing the pin as input, and checking if the sensor gave the correct value when water was detected.

4.3 Collision avoidance system

An integral part of the maneuverability control of the ROV is the usage of sonar data to create a system that actively prohibits the ROV to run into obstacles. As real life conditions can be unpredictable, the system was designed with focus on simplifying operations for the user, but still be easy to deactivate if not working as intended.

The collision avoidance system was coded as an object, initialized from class *InterlockingSystem* which is located within python file *Interlocking.py*. In listing 4.1 it shown how the initialization of the system is performed. For the explanation under the code within the class *InterlockingSystem* will not be discussed as it is consisting of many code lines, but can be seen in detail within Appendix G.

```
if __name__ == "__main__":
    ils = InterlockingSystem()
```

Listing 4.1: Python Raspberry Pi: Initializing object for controlling collision avoidance system.

The main logic for using the interlocking system is referred to using the *main.py* script by using object methods, this creates abstraction where the main logic is hidden away, making the code easy to read and troubleshoot. In the continuously running while loop within *main.py* it is first checked if operator is requesting reset of all interlocked zones, see listing 4.2.

```
while 1:
    if config.forceReset:
        print("Operator is forcing reset of all interlocked zones")
        ils.resetAllZones()
```

Listing 4.2: Python Raspberry Pi: If user requests reset of interlocked zones

Next, within the while loop referred to in 4.2, a check is performed to see if any objects in the scanned angle is located. If the object is detected, the currently scanned zone is interlocked by calling a class method, this logic is shown within listing 4.3. Additionally this is shown in the GUI by highlighting run zones as red.

```
if ils.findObject(config.data_lst):
    ils.setInterlockZone(ils.findZone(config.angle), config.angle)
```

Listing 4.3: Python Raspberry Pi: Checking if object is detected an interlocks zones accordingly.

As the ROV will under normal operations be able to rotate freely even if there is interlocking for linear movements, and objects could move away from ROV, logic to automatically reset interlocked zones were implemented. The general logic for this is shown in listing 4.4. If the sonar has rotated an entire and reads a previous interlocked value as *no-object*, this interlocking is removed.

```
if ils.checkIfResetPermitted(config.angle):
    ils.resetInterlockZone(ils.findZone(config.angle))
```

Listing 4.4: Python Raspberry Pi: Checking if a previously locked zones should be reset.

4.4 Graphical user interface

The user interface is the only way for the operator can interface with the ROV during normal operation. The GUI was implemented with simplicity in mind, where a minimalist approach was taken to ensure that little training was needed to operate the ROV safely.

To design the general layout of buttons, sensor values and other components *Qt designer* [20] was used. This software enabled us to easily decide the position, size and types of objects for our GUI. When all the objects are placed correctly, the software enables us to export the graphical design file, into a Python file. This Python file contains a class which contains all the previously designed objects. From the code within the class, graphical parameters can be adjusted, as for example color and position if the initial design was not satisfactory.

However, for our purposes we mostly needed access to the class methods that changes if buttons the GUI are manipulated. In this way the logic needed for the system could be designed. To create a tidy program for the surface PC, it was decided to split the GUI functionality into three files. *config.py* contains the global variables, *interfacing.py* contains the generated class from the designer software. Finally, *main.py* handles the logic between the communication and GUI.

4.5 Communication

For the system to work as intended, the components have to be able to share information as sensor data and commands, efficiently. The total system consists of three main components that handles computation and running algorithms. These components are the Raspberry Pi, Arduino Uno and a personal computer running the graphical user interface at surface level. Additionally, the more advanced sensors, like the sonar and conductance sensor also requires more sophisticated communication, as opposed to more straightforward analog or discrete sensors. In Figure 4.3 an overview of the total system communication is visualized. In the following sub-chapters the methodology of implementing communication between the different devices are explained.

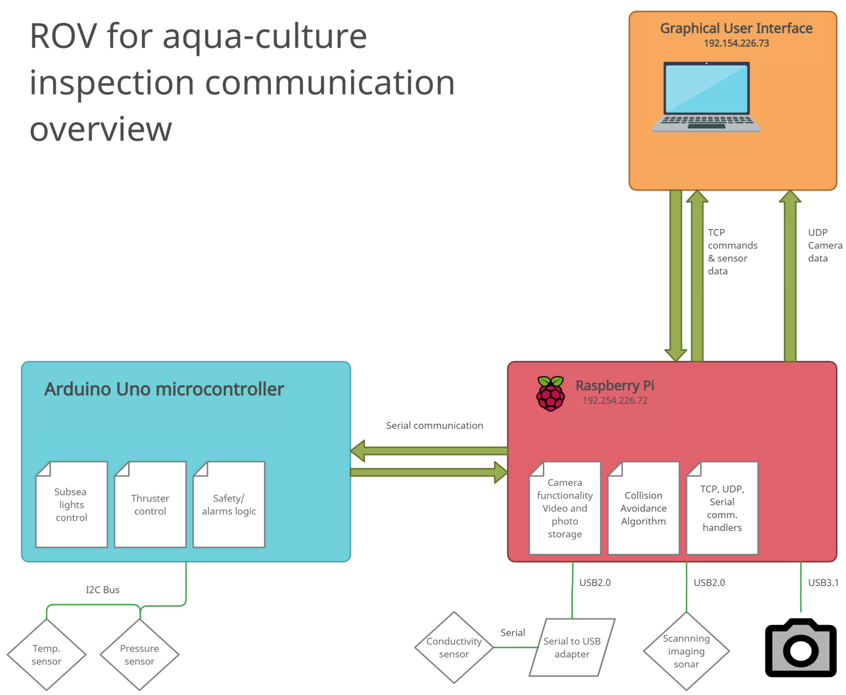


Figure 4.3: Displaying communication between all devices for the ROV system.

4.5.1 Temperature & pressure sensors - Raspberry Pi

Both the temperature and pressure sensors communicate over I2C protocol. I2C communication is implemented by supplying 3.3VDC to both sensors, and connecting two additional wires in parallel that carry the data. The data wires consists of an SDA and SCL signal connected to pin A4 and A5 on the Arduino Uno, respectively [3]. To use I2C communication on Arduino we need to import functionality from the Wire library and specific libraries for each sensor, see listing 4.5.

```
#include <Wire.h>
#include "MS5837.h"
#include "TSYS01.h"
```

Listing 4.5: C++ Arduino Uno: Importing I2C communiction functionality from Wire library.

In the Arduino setup function, the I2C bus is activated, and both I2C sensor objects are created, see listing 4.6 for an excerpt of the Arduino code.

```
void setup() {
    Wire.begin();

    pressSensor.init();
    tempSensor.init();
}
```

Listing 4.6: C++ Arduino Uno: Setup phase of Arduino initializing I2C and sensor objects.

Finally to read the ambient pressure and temperature, built-in class methods are executed on the objects. These values are saved to variables on the Arduino, that can later be communicated or performed actions on, see listing 4.7.

```
void loop() {
    tempSensor.read();
    pressSensor.read();

    temp = tempSensor.temperature();
    depth = pressSensor.depth();
}
```

Listing 4.7: C++ Arduino Uno: Reading and storing data over I2C communication.

4.5.2 Arduino Uno - Raspberry Pi

The Arduino and RPi sends and receives data using serial communication. The communication is implemented in a way that data is only sent when necessary, which in turn reduces bandwidth and computing power usage. The Arduino Uno sends data continuously with a set interval, as new sensor values from temperature and pressure is needed in the GUI. But the RPi only sends data to the Arduino when new commands or control parameters have been set.

On the Arduino side, the serial communication is initially started with defining one parameter, baud-rate, see listing 4.8. The baud-rate has to correspond to settings in the RPi. No additional configuration has to be done, as the default is eight data bits, no parity and one stop bit [4]. These settings are favorable for our transmission requirements, and are easily replicated on the corresponding RPi end.

```
void setup() {
  Serial.begin(9600);
}
```

Listing 4.8: C++ Arduino Uno: Initializing communication.

The Arduino software consists of multiple files to organize functionality. The serial communication functions are located in the *communications.cpp* file, and has two functions, *sendToRaspberry* and *receiveFromRaspberry*, which is shown in listing 4.10 and 4.12, respectively. Data sent from Arduino consists of sensor values temperature, depth and a boolean value of if there is detected any leaks. These are sent every 30th iteration of the Arduino loop function, as can be seen in listing 4.9. This was decided as the Arduino executes code quickly in relation to the Python script in the RPi, and the measured values are slow processes.

```
void loop() {
  // Every 30 program iteration the Arduino sends data to Raspberry
  if (i > 30) {
    sendToRaspberry(temp, depth, leakStatus);
    i = 0;
}
```

```
i++;
```

Listing 4.9: C++ Arduino Uno: Logic in *main.cpp* that determines how often data is transmitted to RPi.

The data is passed to a function that parses the data into a JSON object, which makes it easy to access when received in the RPi. The JSON object is then serialized to a bytes object and sent to the RPi, see listing 4.10.

```
void sendToRaspberry(float arg1, float arg2, bool arg3) {
  outDoc["Temp"] = roundNum(arg1, 1);
  outDoc["Depth"] = arg2;
  outDoc["Leak"] = arg3;

  // Format the data to serial
  serializeJson(outDoc, Serial);

  // Sending to Raspberry Pi
  Serial.println();
}
```

Listing 4.10: C++ Arduino Uno: Function in *communications.cpp* that takes in arguments to be sent over Serial to RPi.

For every loop in the main Arduino program, the program checks if there is data in the serial input buffer. As the communication continuously sends data when the programs are active, a loss of communication can be coded. If the loop is iterated four times and no data is found, all motors are commanded to stop. This is to prevent the ROV from running if the communication is broken, as new commands would not be detected, see listing 4.11 for an excerpt of the Arduino main function. Otherwise if data is available, the data is unpacked in *reveiveFromRaspberry* function shown in listing 4.12.

```
void loop() {
  if (Serial.available()) {
    receiveFromRaspberry();
    missedPackets = 0;
  } else {
    missedPackets++;
```

```
    if (missedPackets > 3) {
      fullStop();
    }
  }
}
```

Listing 4.11: C++ Arduino Uno: Initializing communication.

The function shown in listing 4.12 deserializes the data received from the RPi, and calls the appropriate control functions with the new data. Initially, data is read into a string variable until it reaches the end of a command, which is registered when a newline character is detected. The input data is then stored in a JSON document, for easy accessibility. Finally all the received values are used as arguments for functions that control the movement and light strength.

```
void receiveFromRaspberry() {
  bool z1lock; bool z2lock; bool z3lock; bool z4lock;
  bool z5lock; bool z6lock; bool z7lock; bool z8lock;

  String  payload;
  payload = Serial.readStringUntil( '\n' );
  StaticJsonDocument<512> doc;
  deserializeJson(doc, payload);

  setLights(doc["light"]);

  z1lock = doc["locked"][0];
  ...
  z8lock = doc["locked"][7];

  setMotorSpeeds(doc["runZone"], z1lock, z2lock, z3lock,
  z4lock, z5lock, z6lock, z7lock, z8lock);
}
```

Listing 4.12: C++ Arduino Uno: Initializing communication.

For the RPi side of the serial communication with Arduino, the communication is executed in a separate thread from the main program, see listing 4.13. This is done partly to make the main system program, which is the Python scripts in the RPi, to be able to perform other actions when communication is not active, or from other causes. Additionally, to increase program execution speed.

```
SerialThread = threading.Thread(target=serialCom)
SerialThread.start()
```

Listing 4.13: C++ Arduino Uno: Initializing communication.

The serial communication is executed in a function in the *GUI_communications.py* file. A code snippet of this function is shown in listing 4.14. Here the connection is initialised by selecting which port the RPi has the Arduino connected, and other parameters as baudrate and number of stopbits are set to correspond with the settings set in the Arduino in listing 4.8.

```
def serialCom():
    ardSer = serial.Serial('/dev/ttyACM0', 9600, timeout=1,
    parity=serial.PARITY_NONE, bytesize=serial.EIGHTBITS, stopbits=serial.
   STOPBITS_ONE)
```

Listing 4.14: C++ Arduino Uno: Initializing communication.

The main logic for the serial communication is located within a while loop that continuously iterates. The logic within this loop is separated as two *if statements*. The first one is shown in the beginning of listing 4.15, this checks if any new commands from the GUI has been received or internal logic in the RPi has been updated. If new actions are to be executed, a JSON structure will be formed and serialized, and lastly sent to the Arduino Uno.

The other if statement in listing 4.15 checks if the serial input buffer reserved for Arduino communication has received any bytes, if new bytes are found, the data is unserialized and saved to global variables, that can be easily accessed for the functionality that relies on those data values.

```
while 1:
    if config.newArduinoCommands:
        ArdDataOut = {}
        ArdDataOut["light"] = config.light
        ArdDataOut["runZone"] = config.runZone
        ArdDataOut["locked"] = config.interlockedZones
        ArdDataOut = json.dumps(ArdDataOut)
        ardSer.write(ArdDataOut.encode())
```

```
    if ardSer.in_waiting > 0:
        ArdDataIn = json.loads(ardSer.readline())
        config.temp = ArdDataIn["Temp"]
        config.depth = ArdDataIn["Depth"]
        config.leak = ArdDataIn["Leak"]
```

Listing 4.15: C++ Arduino Uno: Initializing communication.

4.5.3 Scanning imaging sonar - Raspberry Pi

The communication between the RPi and the sonar was implemented with serial connection over USB. To share information the data was sent using the Ping protocol [36], which is a purpose built protocol developed by the manufacturer of the sonar, BlueRobotics [17]. For our purposes the in depth working principle of the protocol is not crucial to understand, as the functionality is implemented in classes imported from the Ping360 libraries. In listing 4.16 the initialization of the connection is established by creating an object *p* that is used to command and read values from the sonar. Then a serial communication is initialized by using a class method with device path and baud-rate set by the command line interface using the Python *argparse* module.

```
p = Ping360()
p.connect_serial(args.device, args.baudrate)
```

Listing 4.16: C++ Arduino Uno: Initializing communication.

To read sensor values and send actuator commands, class methods on the previously initialized object is called. An example of this is shown in listing 4.17, here an excerpt of the main *while* logic is shown with a command that sends the new angle the sonar should rotate to.

```
while 1:
    p.transmitAngle(config.angle)
```

Listing 4.17: C++ Arduino Uno: Initializing communication.

This class method is located within the *Ping360* class, which is a child class of the *PingDevice* class. Listing 4.18 shows the class method called in the previous listing, 4.17. In this method a new function *control_transducer* is called, with the updated angle command and additional parameters that are required to control the sonar.

```
def transmitAngle(self, angle):
    self.control_transducer(
        0,
        self._gain_setting,
        angle,
        self._transmit_duration,
        self._sample_period,
        self._transmit_frequency,
        self._number_of_samples,
        1,
        0
    )
    return self.wait_message([definitions.PING360_DEVICE_DATA, definitions
    .COMMON_NACK], 0.5)
```

Listing 4.18: C++ Arduino Uno: Initializing communication.

The class method *control_transducer* is shown in listing 4.19. Here a new object *m* is initialized from class PingMessage. Furthermore parameters are set to the new object, and finally the object is serialized and sent over USB connection.

```
def control_transducer(self, mode, gain_setting, angle, transmit_duration,
     sample_period, transmit_frequency, number_of_samples, transmit,
    reserved):
        m = pingmessage.PingMessage(definitions.PING360_TRANSDUCER)
        m.mode = mode
        m.gain_setting = gain_setting
        m.angle = angle
        m.transmit_duration = transmit_duration
        m.sample_period = sample_period
        m.transmit_frequency = transmit_frequency
        m.number_of_samples = number_of_samples
        m.transmit = transmit
        m.reserved = reserved
        m.pack_msg_data()
        self.write(m.msg_data)
```

Listing 4.19: C++ Arduino Uno: Initializing communication.

4.5.4 Conductivity sensor - Raspberry Pi

The communication between RPi and the Aanderaa sensor is implemented as serial communication. Like the Arduino Uno and Sonar, the Aandera sensor program executes in a separate thread using multithreading, as initialized in listing 4.13. At the start of an iteration in the while loop, a command is sent to the Aanderaa that tells the sensor to perform a new sample, see listing 4.20. Furthermore commands for each parameters are sent, and then the returning information is stored in global variables. The program does this for salinity, speed of sound in the water, water density and conductivity.

```
def serialCom():
    condSer = serial.Serial('/dev/ttyUSB1', 9600)

    while 1:
        condSer.write("do_sample\n".encode())   # Commands conductivity
    sensor to conduct sample of parameters

        config.salinity = getAanderaaData(condSer, "get_salinity\n")
        soundSpeedReading = getAanderaaData(condSer, "get_soundspeed\n")
        config.density = getAanderaaData(condSer, "get_density\n")
        config.conductivity = getAanderaaData(condSer, "get_conductivity\n
    ")
```

Listing 4.20: C++ Arduino Uno: Initializing communication.

To send and receive the requested data, a function *getAanderaaData* is called, see listing 4.21. This function starts of by reading the input buffer, in a way that clears the buffer from unwanted characters that the sensor periodically sends out. Next, the command is sent out, and the returning data is decoded and saved in global variables. The input data goes through several checks for unwanted characters before finally saving the data.

```
def getAanderaaData(condSer, request_str):
    condIn = b''
    condIn = condSer.readline()
    condSer.write(request_str.encode())
    condIn = condSer.readline().decode()
    data = condIn.split('\t')
```

```
    data = data[-1]
    data = data.replace('\r\n', '')
    return float(data)
```

Listing 4.21: C++ Arduino Uno: Initializing communication.

4.5.5 Raspberry Pi - Personal Computer (GUI)

As seen in the communication overview in Figure 4.3, the communication between the RPi and the GUI application consists of two protocols. The camera data is sent from the RPi to the GUI using UDP transmission. And all other data, including the commands from the GUI, and other sensor data, is communicated with a TCP connection. Both of these protocols runs in separate threads in the RPi and on the GUI side. This was implemented by creating unique functions for each of the protocols, and calling them using the *multithreading* Python library. If for example the camera functionality fails, the system can still take commands and read crucial sensor values such as the leak sensor. The below sections explains how the UDP and TCP connections where programmed for the RPi and GUI.

UDP

The UDP connection between the RPi and GUI was set up with the UDP server in the GUI, and the respective client socket in the RPi. This was done purposely, as by design the client socket should be able to disconnect and reconnect without any user interaction. As for the picture taking functionality, the camera feed will pause to be able to adjust quality of frames captured, and automatically resume connection when picture is taken. In addition, if for some unexpected reason the camera feed stops, it will be able to reconnect again automatically given the problem fixes itself.

The initial setup for the UDP server is shown in listing 4.22. In the code snippet only an excerpt of the main logic of the GUI application is shown, here an instance of threading is started on the function *UDPCom*. This thread is started without any exit functionality, as the thread is designed to run continuously during the entire program execution.

```
if __name__=="__main__":
    cam_communication = threading.Thread(target=UDPCom)
    cam_communication.start()
```

Listing 4.22: Python in GUI application: Initializing communication with UDP protocol for camera feed in a seperate thread.

A part of the function *UDPCom* is shown in listing 4.23. Here the initial preparations for the connection is performed. First the size of each data packet, datagram, is defined to be the maximum allowable size, 65536 bytes. Further, a function to empty the input buffer is implemented.

```
def UDPCom():
    MAX_DGRAM = 2**16

    def dump_buffer(s):
        while True:
            seg, addr = s.recvfrom(MAX_DGRAM)
            print(seg[0])
            if struct.unpack('B', seg[0:1])[0] == 1:
                print("finish emptying buffer")
            break
```

Listing 4.23: Python in GUI application: Start of function that is executed in separate thread

Still within the UDP communication function, *UDPCom*, the UDP client is initialized and the static IP address for the users computer is bound with a free port, example configuration is shown in listing 4.24.

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('169.254.226.73', 20001))
```

Listing 4.24: Python in GUI application: Setup of connection with UDP client within *UDPCom* function.

Next in the UDP logic, a continuous running while loop is entered, see listing 4.25. This loop checks for the next UDP datagrams. For every datagram it receives it checks if the information should be added to a larger data variable, *dat*, that contains one image and if the datagram is the final information needed for one image. If sufficient data for one image has been received, the image is decoded and stored to an image variable. Later this image variable is displayed on the GUI for the user to see.

```
while 1:
   seg, _ = s.recvfrom(MAX_DGRAM)
    if struct.unpack("B", seg[0:1])[0] > 1:
        dat += seg[1:]
    else:
        dat += seg[1:]
        img = cv2.imdecode(np.frombuffer(dat, dtype=np.uint8), 1)
```

Listing 4.25: Python in GUI application: Functionality that unpacks datagrams into an image datatype.

For the RPi side of the UDP connection, an UDP client has to be initialized. The UDP communication is created within its own separate thread, in a similar way the UDP server was initialized in the GUI. The initialization for the UDP client is seen in listing 4.26.

```
if __name__ == "__main__":
    UDPThread = threading.Thread(target=UDP)
    UDPThread.start()
```

Listing 4.26: Excerpt of main functionality in Python *main.py* script. Initialises the UDP client in a separate thread.

In listing 4.27 a code snippet of the function that handles UDP for the RPi is shown. The first lines connects to the server in the GUI. Then the camera data is continuously sent over UDP using the class *FrameSegment*. If the operator of the GUI selects to take a picture, the UDP connection will be closed during this process, and restarted when the picture is taken and saved locally on the RPi.

```
def UDP():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    port = 20001
    fs = FrameSegment(s, port)

    while not config.takeHighResPhoto:
```

```
        fs.udp_frame(frame)
```

Listing 4.27: Connects to the GUI using UDP and initializes object that holds image frame.

As mentioned previously, a class *FrameSegment*, has a key role in sending data over UDP. A snippet of the most relevant functionality of this class is shown in listing 4.28. The sending of picture frames is handled by class method *udp_frame*, which takes the image taken by the OpenCV functionality, and divides it into optimal sized UDP datagrams.

```
class FrameSegment(object):
    def udp_frame(self, img):
        compress_img = cv2.imencode(".jpg", img)[1]
        dat = compress_img.tostring()
        size = len(dat)
        num_of_segments = math.ceil(size/(self.MAX_IMAGE_DGRAM))
        array_pos_start = 0

        while num_of_segments:
            array_pos_end = min(size, array_pos_start + self.
   MAX_IMAGE_DGRAM)
            self.s.sendto(
                        struct.pack("B", num_of_segments) +
                        dat[array_pos_start:array_pos_end],
                        (self.addr, self.port)
                        )
            array_pos_start = array_pos_end
            num_of_segments -= 1
```

Listing 4.28: Python in RPi: Excerpt of class with a method that divides up image frames to UDP datagrams and sends to GUI.

TCP

For all other communication between the RPi and the GUI, except for the camera feed, a TCP connection is used. The TCP connection is used in a way that only client or server sends data if needed, this was done to save bandwidth, as the camera data needs a large part of communication bandwidth. As sensor data is continuously needed in the GUI, the RPi sends data with a fixed interval. This data includes temperature, depth, sonar and other miscellaneous information. Data from the GUI to the RPI only consists of user commands selected in the GUI.

The TCP server was decided to be implemented in the RPi, as the RPi Python script should be running continuously during ROV operation. The TCP connection was created within a separate thread, which was initialized from the *main.py* scipt, as seen in listing 4.29.

```
if __name__=="__main__":
    other_communication = threading.Thread(target=TCPCom)
    other_communication.start()
```

Listing 4.29: Python in GUI application: TCP connection is initialized in a separate thread.

In listing 4.30 a snippet of the *TCPCom* function is shown. Here the initial commands for starting the TCP client is performed. The IP address and reserved port of the RPi TCP server is saved to local variables, and called in connection initializion functions from the *socket* library.

```
def TCPCom():
    SERVER = "169.254.226.72"
    PORT = 1422
    HEADERSIZE = 10

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((SERVER, PORT))
```

Listing 4.30: Python in GUI application: Excerpt of function that handles TCP communication with RPi. Shows initial set-up of connection as client side.

Next within the *TCPCom* function a continuously running while loop is entered, see listing 4.31. For the first incoming data after the *full_msg* variable has been reset to no data, the message length is found and stored to variable *msglen*. This variable is used to check if the partitioned data received is the final data to assemble one input message. Otherwise it can be extrapolated that additional TCP packets are needed to be added to the message.

```
while 1:
    msg = s.recv(8192)
    if new_msg:
        msglen = int(msg[:HEADERSIZE])
        new_msg = False

    full_msg += msg
```

Listing 4.31: Python in GUI application: Continuously checks for TCP data sent from RPI.

Next, within function *TCPCom*, if statements are used to check if assembled data is a finished message, see listing 4.32. If it is, the data is deserialized and saved to variable *RaspDataIn*. If at this point, new commands have been selected on the GUI, a global boolean flag has been activated, called *config.newCommands*, the TCP will respond with a message to the RPi. The sending functionality consists of simply saving global variables to a dictionary, and serializing this with the library *pickles*.

```
if len(full_msg)-HEADERSIZE == msglen:
    RaspDataIn = pickle.loads(full_msg[HEADERSIZE:])

if config.newCommands:
    print("[ATTENTION] New commands sent to Raspberry")
    config.newCommands = False

    RaspDataOut = {
        "light": config.light,
        ...
        "takeVideo": config.takeVideo
    }
```

Listing 4.32: Python in GUI application: Logic that checks if message has been parsed to appropriate size

On the RPi side of the TCP connection, a TCP server has to be initialized. This is done within the main while loop of the *main.py* program, as seen in listing 4.33. If the GUI client has not connected, the code will try to initialize a new thread with the logic that handles incoming TCP data. This thread will always be started as the GUI has no option to connect before the RPi program has started.

```
while 1:
    if config.address == "":
        TCPThread = threading.Thread(target=TCPIn)
        TCPThread.start()

    TCPOut(s, HOST, PORT, HEADERSIZE)
```

Listing 4.33: Excerpt of main while loop in *main.py*

In listing 4.34 the function that handles TCP communication to the RPi is shown. First the size of the header of the TCP is chosen, corresponding with selected in the GUI application. Next, the function continues to a series of while loops that checks for incoming data. If data is registered, the message is deserialized and reassembled to a dictionary. Finally, global variables are updated with the received information.

```
def TCPIn():
    HEADERSIZE = 10

    while 1:
        receiving = True
        full_msg = b''
        new_msg = True
        incoming_message = config.clientsocket.recv(8192)

        while receiving:
            if new_msg:
                msglen = int(incoming_message[:HEADERSIZE])
                new_msg = False

            full_msg += incoming_message

            if len(full_msg)-HEADERSIZE == msglen:
                GuiDataIn = pickle.loads(full_msg[HEADERSIZE:])
```

```
                config.light = GuiDataIn["light"]
                ...
                config.takeVideo = GuiDataIn["takeVideo"]

                config.newArduinoCommands = True

                receiving = False
                new_msg = True
                full_msg = b''
```

Listing 4.34: TCP functionality that handles TCP data sent from the GUI.

To send data to the GUI over TCP, an additional function *TCPOut* is needed, see listing 4.35. This function is called every program iteration, at the point new values from the sensors have been registered. If no TCP client is registered, the RPi tries to connect, otherwise data is formatted into a dictionary, serialized and sent.

```
def TCPOut(s, HOST, PORT, HEADERSIZE):
    communicating = True
    startReceive = True

    while communicating:
        receiving = True

        if not config.address:
            config.clientsocket, config.address = s.accept()
            print(f"Connection from {config.address} has been established.
   ")

        GuiDataOut = {
            "temp": config.temp,
            ...
            "density": config.density

        }

        msg = pickle.dumps(GuiDataOut)
```

```
msg = bytes(f"{len(msg):<{HEADERSIZE}}", 'utf-8') + msg
config.clientsocket.send(msg)

communicating = False
```

Listing 4.35: Python in RPi: TCP functionality that handles data so be sent to GUI.

4.6 Design and modelling

This section goes over how the group came up with the design of the ROV and the methods used to build the prototype.

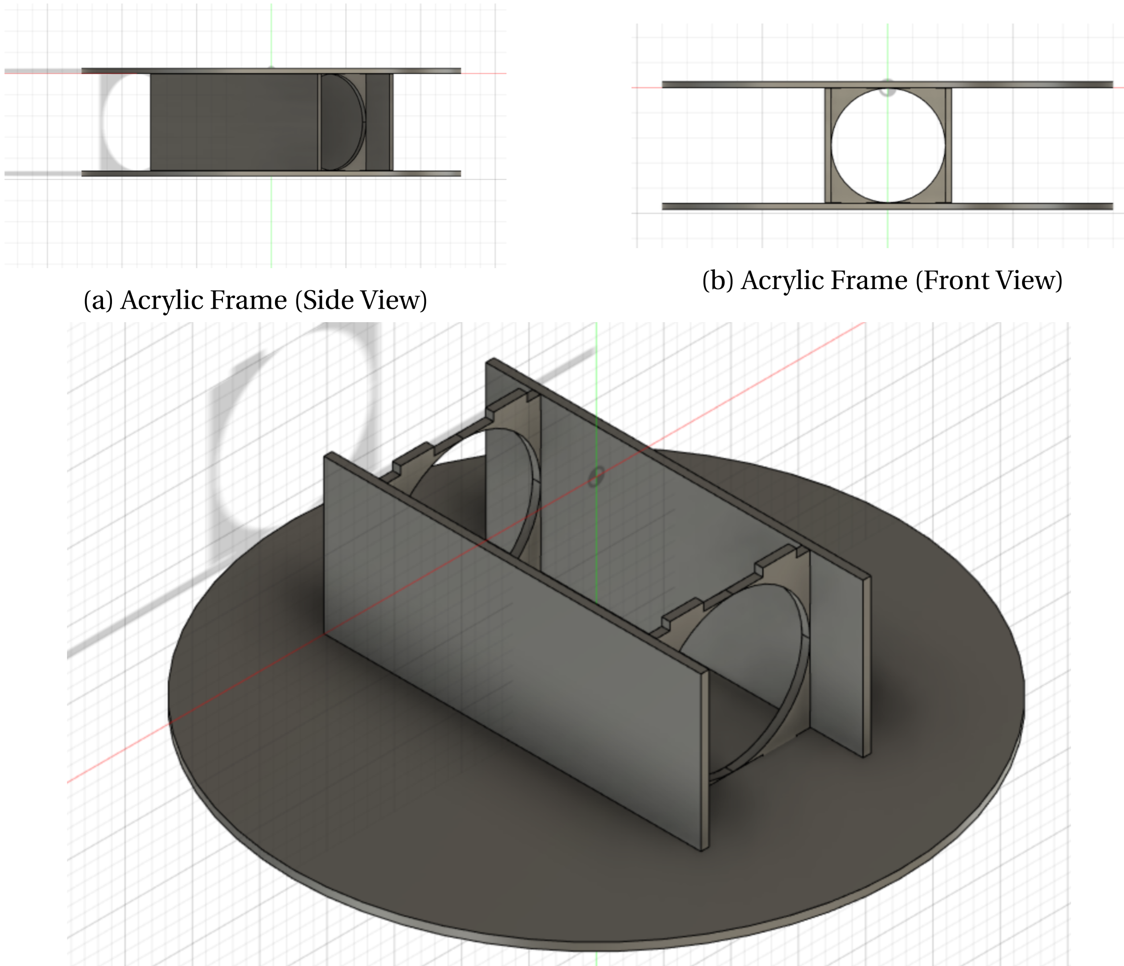
4.6.1 Concept

For the design it was decided early on that it should be a simple one due to the group having no previous design experience and this being an automation thesis. The group had a few ideas in mind when starting the project, but after selecting and ordering sensors, thrusters and the waterproof enclosure a specific one was decided.

4.6.2 Design and Manufacturing of ROV body

For the final design a circle shape was chosen due to it allowing easy mounting of thrusters. For the thruster positioning we decided to mount them at 120 degree intervals, this allows the ROV to move in all directions. This thruster placement was taken from an earlier ROV made by another group. The shape also gives a lot of surface area for mounting sensors, lights and other components. The circles are made out of acrylic, the material was chosen due to its light weight and high strength. Working with acrylic was made simple due to NTNUs laser cutter. The laser cutter allowed the group to make designs in Fusion 360 and then cut them out with high precision and speed, making prototyping new mounts and parts a lot easier.

To hold the two circles together three aluminum extrusions were used, they are fastened with nuts and bolts. These extrusions are also used as mounting points for the thrusters. To hold the



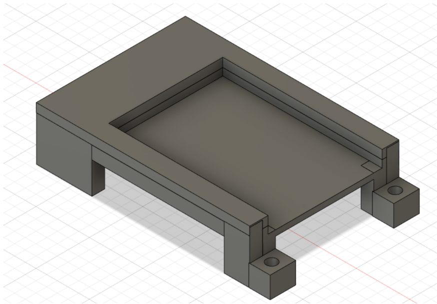
(a) Acrylic Frame (Side View)

(b) Acrylic Frame (Front View)

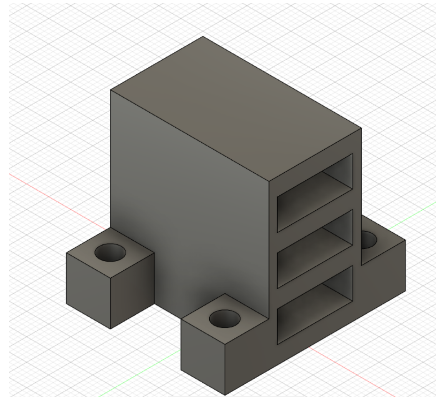
(c) Acrylic Frame (No top plate)

Figure 4.4: Fusion 360 models of ROV body

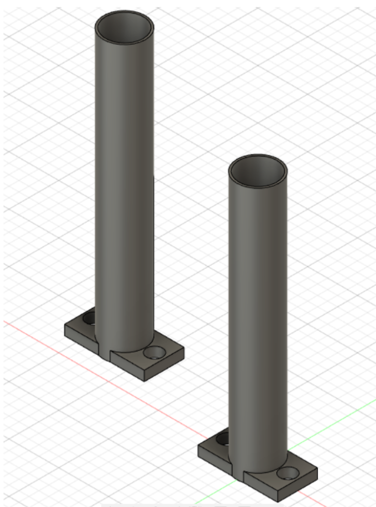
BlueRobotics 4 inch waterproof enclosure in place, a bracket made out of acrylic was designed in Fusion 360, see Figure 4.4. The bracket was cut out with the laser cutter and then glued in place with epoxy. The reason for not permanently mounting the enclosure in place was to make maintenance easier and allowing for reuse of the enclosure, seeing as the current design was made to be a prototype this was deemed optimal.



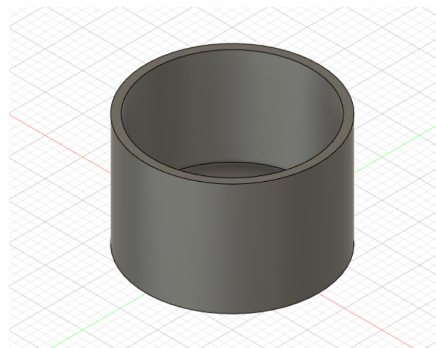
(a) Mount for Arduino and RaspberryPi



(b) Mount for ESC speed controllers



(c) Battery holder

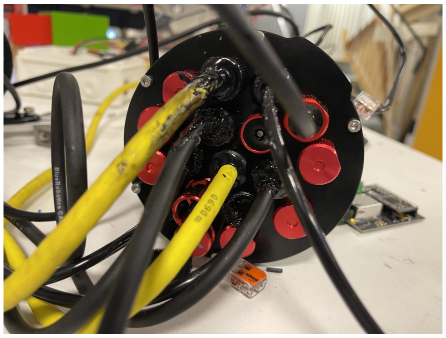


(d) Battery holder end cap

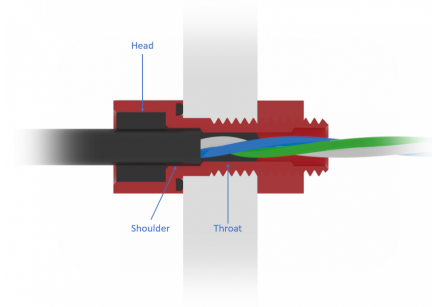
Figure 4.5: Fusion 360 Models of internal mounts

4.6.3 Making of Internal Mounts

Due to the small amount space in the enclosure and the quantity of components, custom hardware mounts were needed to be able to fit everything inside. It is also important to keep EMI emitting components away from sensitive components, as the RPi. To make these, Fusion 360 was used for designing, and then they were produced with the schools Ultimaker2+ 3D printers, see Figure 4.5.



(a) Photo of cable entry points



(b) Figure showing how the cable penetrators work, from [8]

4.6.4 External box

When working on putting everything together it quickly became apparent that space inside the enclosure would be a issue. It was quickly decided that adding an external box to house components that produce electrical noise and heat would be a good solution. The box is IP66 rated and is filled with clear casting epoxy so that it does not buckle under the pressure at the depths the ROV will operate in.

4.6.5 Waterproofing

The BlueRobotics enclosure is sealed by a set of double o-rings on both of the end caps. For the cable entry points BlueRobotics cable penetrators are used as seen in Figure 4.6a. These are threaded and screwed in place. They use a combination of an o-ring and epoxy to seal the cable entry point, see Figure 4.6b. A few of the cables came sealed from the factory. For the other external sensors and the thrusters we had to seal them ourselves with the use of an epoxy made by 3M called "Scotch-Weld Urethane Adhesive 620NS". This epoxy was recommended by the JMRobotics, which is the official retailer of BlueRobotics products in Norway. For the external box, the epoxy seals all the components in a way that even if water entered the box, the components would be safe.

4.7 Electrical

To power the ROV, 48VDC is sent through three pairs of wires that go down to the ROV. A DC to DC converter steps down the voltage to 12VDC. This converter is connected to a 5V Linear Voltage Regulator and another DC to DC converter that outputs 16.5V. The Voltage regulator is required to power the RPi. The 16.5V DC to DC converter is connected in parallel to a battery made up of 4 3,7V Lithium ion batteries. This setup is in place to stop voltage drops from turning off the RPi. There is also a 12V Voltage regulator which is needed to power the Aanderaa conductivity sensor and Arduino. For the voltage regulators capacitors are used to ensure stability, see Appendix D. The 5V regulator is connected to 12V DC to DC converter to limit the amount of heat the regulator produces.

The 5V and 12V supplies both have one fuse each and on the 16,5V supply there are three fuses. One for the thrusters, one for the lights and one for the sonar and Fathom-X Tether. There is also a fuse between the battery and the 16.5V DC-DC converter. It is in place to ensure safety of the batteries and the DC to DC converter itself.

4.7.1 External box

The components located inside the external box are the voltage regulators and the DC to DC converters. These components produce a significant amount of heat as well as electrical noise which could cause problems if they were located inside the main enclosure.

4.7.2 Wiring

As mentioned space was a issue so most of the cables are soldered or clamped to conserve space, these connections were covered with heat shrink tubing. For the poles of the batteries/power supply WAGO terminal blocks are used so its possible to disconnect the power without cutting any wires. This also allows the user to charge the battery.

Chapter 5

Result

The final results of the implementation of our proposed solutions described in methodology are presented in the chapter. The first chapters describes the results for the software and communication solutions, and finally more physical results as the electrical and structure are described.

5.1 Software solutions

In the following sub-chapters the results from the software solutions are described. The results are describing the most relevant results for the separate solutions, and finally how the solutions worked together.

5.1.1 Graphical User Interface

The final GUI satisfied most of our requirements. We wanted a simplistic and clean overview of the most relevant systems. The video stream and sonar plot occupied most of the space, as these are types of information that is advantageous to have in larger formats. The resulting space was used for sensory data that was numerical, as the salinity, depth and other values. The control mechanisms were placed close to the video plot, to make movements as intuitive as possible.

5.1.2 Software performance

The Arduino Uno executes code very quickly in relation to the Python program running in the RPi. It was therefore decided to only execute the Arduino serial communication sending function every 30th iteration of the Arduino while loop. As the measured sensor values have a slow rate of change, this was more than sufficient in terms of performance. During older versions of the program, the Arduino continuously sent data every iteration, and this used a lot of the processing power within the RPi, which reduces other functionality, as the collision avoidance system.

The Python RPi program consisted of multiple files. The distribution of the logic into multiple files was advantageous as finished functionality could be abstracted away and simply reused for other functionality by calling functions or object methods. Using multithreading for the RPi program was essential to achieve the desired functionality and required response times. As some communication configurations only sent data when certain conditions are met, and other continuously sends data every iteration of a program, an RPi system without multithreading would greatly delay response times.

5.1.3 Communication results

The system contains of multiple communication configurations. All were tested separately initially to be able to localize and find errors and bugs, and fix them early in the integration process. Later when integrating all the systems together, this was relatively problem free, as the different communication configurations had been hidden using programming abstraction.

The simplest communication configuration between devices, as the I2C and serial communications worked very reliably when connection was established. As communication libraries are already very refined, and relatively flexible with integration. However, the serial communication between the RPi and Arduino would sometimes crash upon initialization. When this happened, it was solved by restarting the RPi programs, and it would always work on the second attempt.

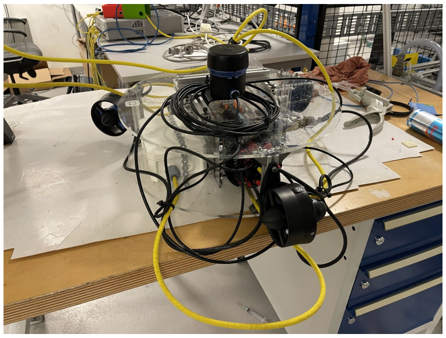
For the ethernet based communication protocols, TCP and UDP, the integration was more demanding. Initially, we planned to only use TCP for all of the communication between the GUI and RPi. However, it was concluded fairly early on that we would struggle with bandwidth, as especially the camera feed is very demanding. Still, by using UDP the communication was struggling to send the picture frames as quickly as we had intended, during the final sea tests we had between 2-4 fps on the GUI.

5.2 Electrical

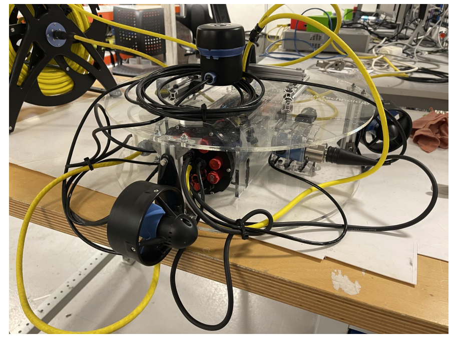
Most of the electrical system worked well during testing. All the of the components were powered and could perform their function as intended. It should be mentioned that the thrusters maximum output is limited artificially in order to keep the power draw within a reasonable level. The output is still enough to allow the ROV to do all of its movements and functions. The value of the implemented battery system is questionable, where it at times seems to have a positive effect, but often not. The current battery solution is described more in detail under the test chapters.

5.3 Physical

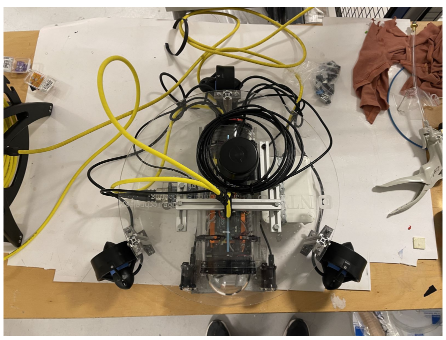
The final version of the ROV and the internals can be seen in Figure 5.1 and 5.2. We were mostly satisfied with how the ROV performed. It worked well in the sea and it held up well during testing with it only experiencing a few problems. The design is also very modular allowing for quick changes when needed.



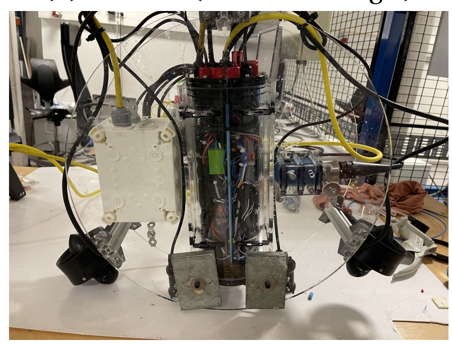
(a) ROV rear



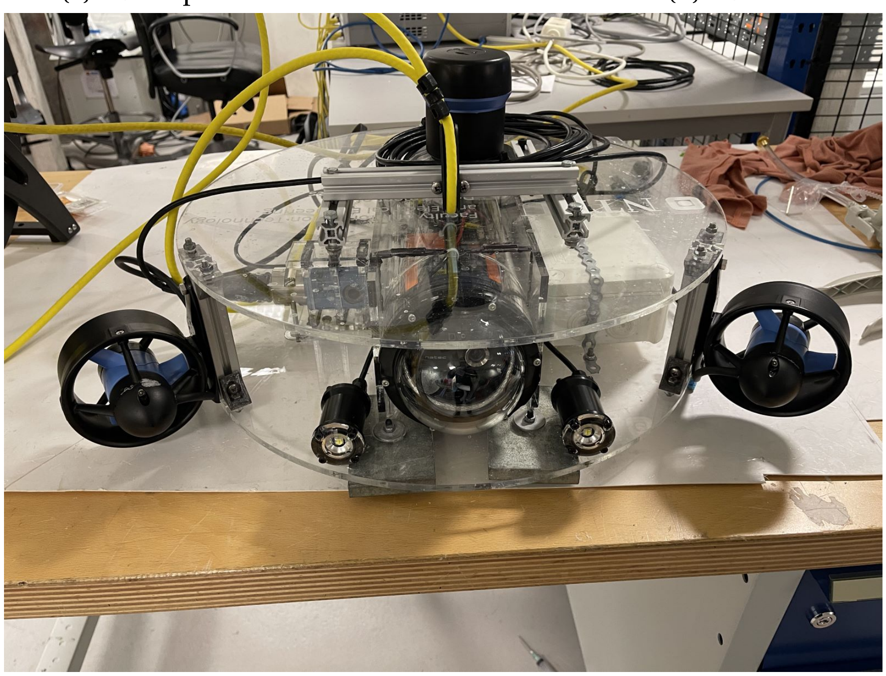
(b) ROV rear(alternative angle)



(c) ROV top

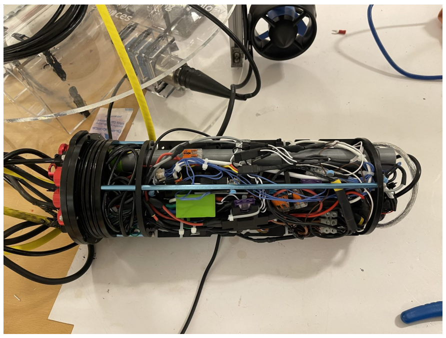


(d) ROV bottom

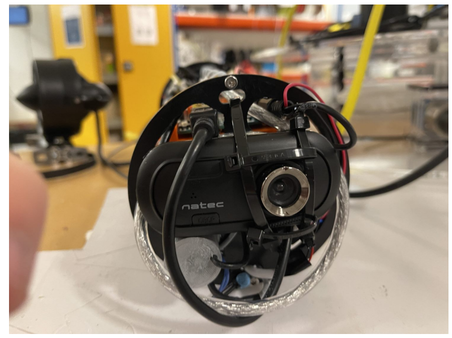


(e) ROV front

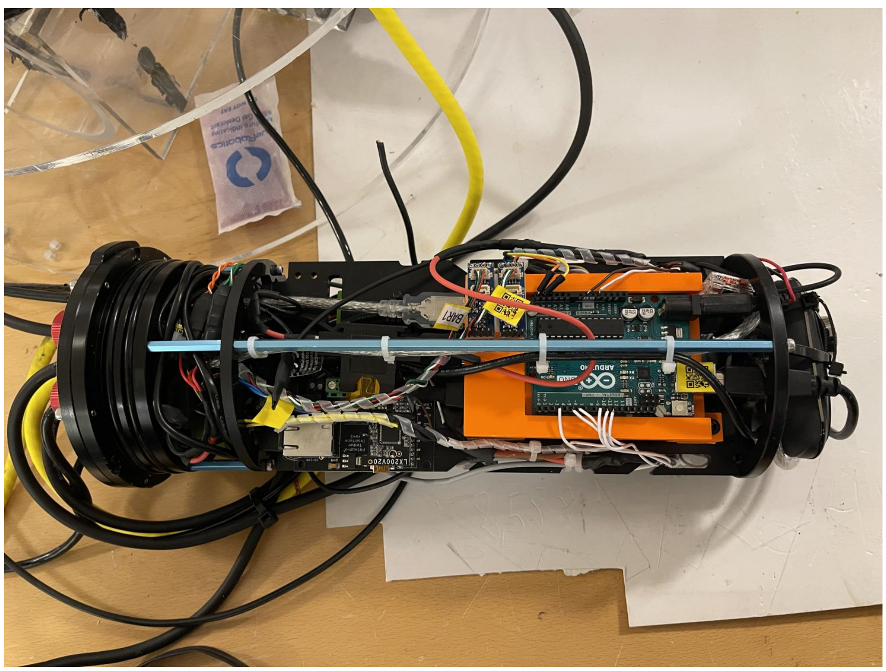
Figure 5.1: Photos of the ROV



(a) Internals of the ROV bottom



(b) Internals of the ROV front



(c) Internals of the ROV top

Figure 5.2: Photos of the internals of the ROV

5.4 Test 1

Test 1 is defined as the first test of the complete system, however multiple subtests between the different systems had been completed prior to this test. Test 1 was completed in the lab in a controlled environment, and not underwater at any point, see Figure 5.3. There was a multitude of uncertainties that had to be checked before filling the box containing the Linear Voltage Regulators and DC to DC converters with epoxy, see Figure 5.4. During the test it was discovered that some of the cables had bad connections and needed to be redone properly, especially the supply for the Arduino would sometimes fall out. This was done quickly so the rest of the testing could go ahead with small delays. When testing the system it seemed like the local battery inside the enclosure did not work as required. When using the ROV at high power, for example with lights at high power, and then starting the motors, the voltage over the RPi would drop significantly, and therefore reboot.

The job of the local batteries is to deliver power when a peak current is needed to run starting of the motors. As if there is high current, the voltage drop in the tether would be significant, and therefore the voltage within the ROV would drop below required levels. When the drop occurs, the batteries should take over supplying. But we suspect that the batteries never starts supplying, as the supply voltage after a multitude of voltage regulator is kept too high at the charging circuit of the batteries. However, no conclusion regarding the battery situation was reached. We increased the constraints of usable light power to a level that even when ran at maximum, and starting the thrusters, the current would not be high enough to induce a voltage drop high enough to disconnect the RPi.

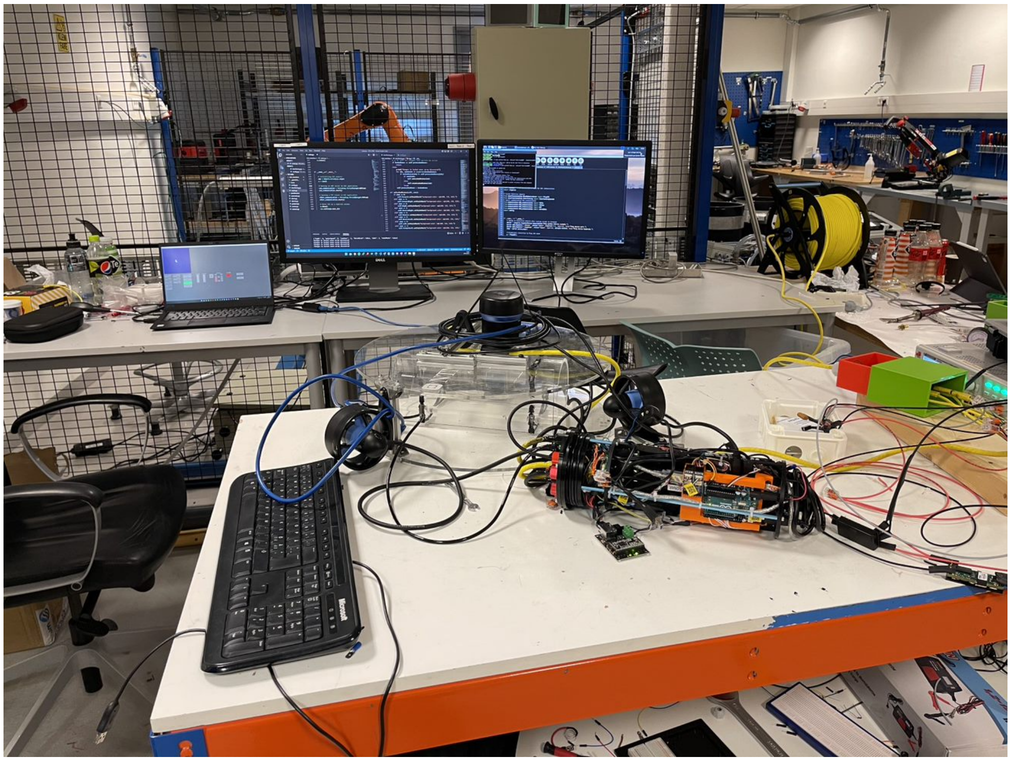
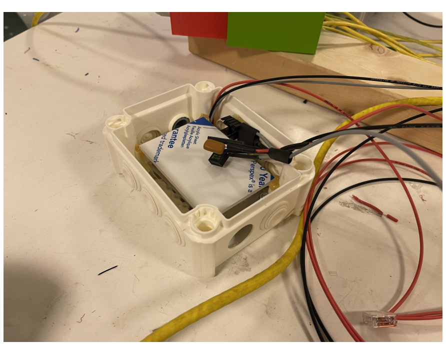
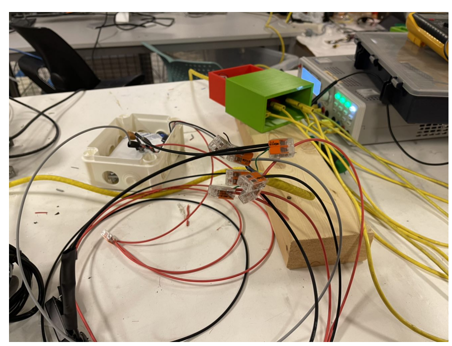


Figure 5.3: Setup for test 1.

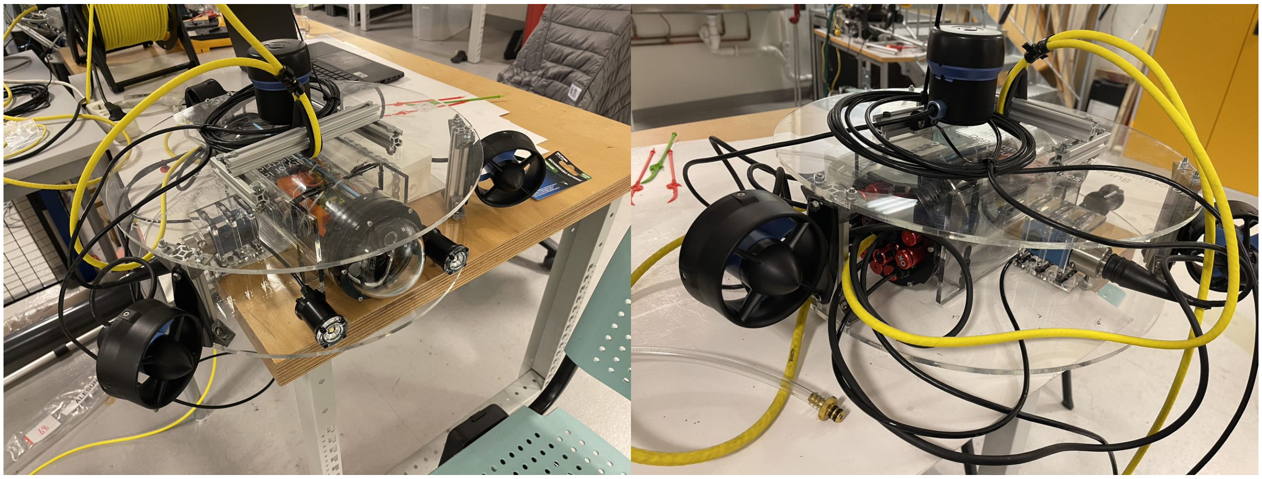


(a)



(b)

Figure 5.4: Photos of the external box during test 1



(a) (b)

Figure 5.5: Photos of ROV before test 2

5.5 Test 2

Test 2 was the first test of the full system in water. The test was done in a water tank located at NTNU Aalesund. The water tank has windows and its shallow enough to allow retrieval without any problems. This was the ideal location for the first water test. During this test as with test 1 a power supply was used to power the ROV. For this test the external box was filled with epoxy in order to waterproof it, see Figure 5.6. The Omron Sentech camera was also swapped out due to a suspected driver issue. The new camera was a standard 1080p webcam that required no drivers, and were therefore easy to integrate into the system. See Figure 5.5 to see how the ROV looked during test 2.

Before submerging the ROV it was checked for leaks. This was done by using a vacuum hand pump which allows you to suck air out of the ROV, see Figure 5.7. If it holds this negative pressure it means the seal is good and that the ROV is ready to be submerged. When the ROV was first submerged everything looked good and all systems performed as intended. There was however a problem with the balance as the rear was heavier leading to the front tilting upwards. To fix this problem two pieces of metal were screwed in place at the front of the ROV. This fixed the problem and the ROV became stable, see Figure 5.8. Another problem was the position of the camera, this was fixed after the test as it did not interfere with testing of the system as a whole.

Figure 5.6: External box filled with epoxy.

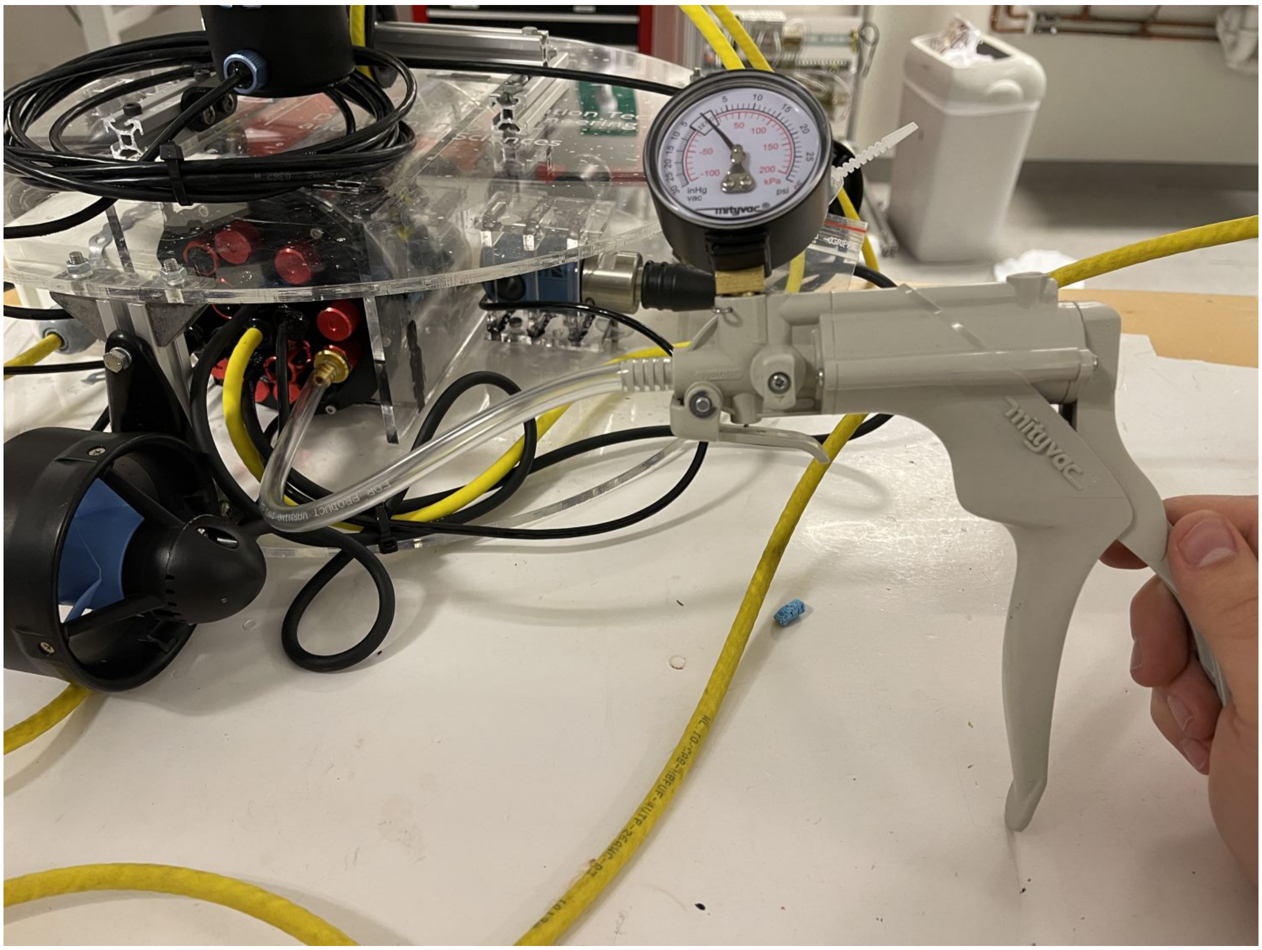


Figure 5.7: Checking the ROV for leaks.

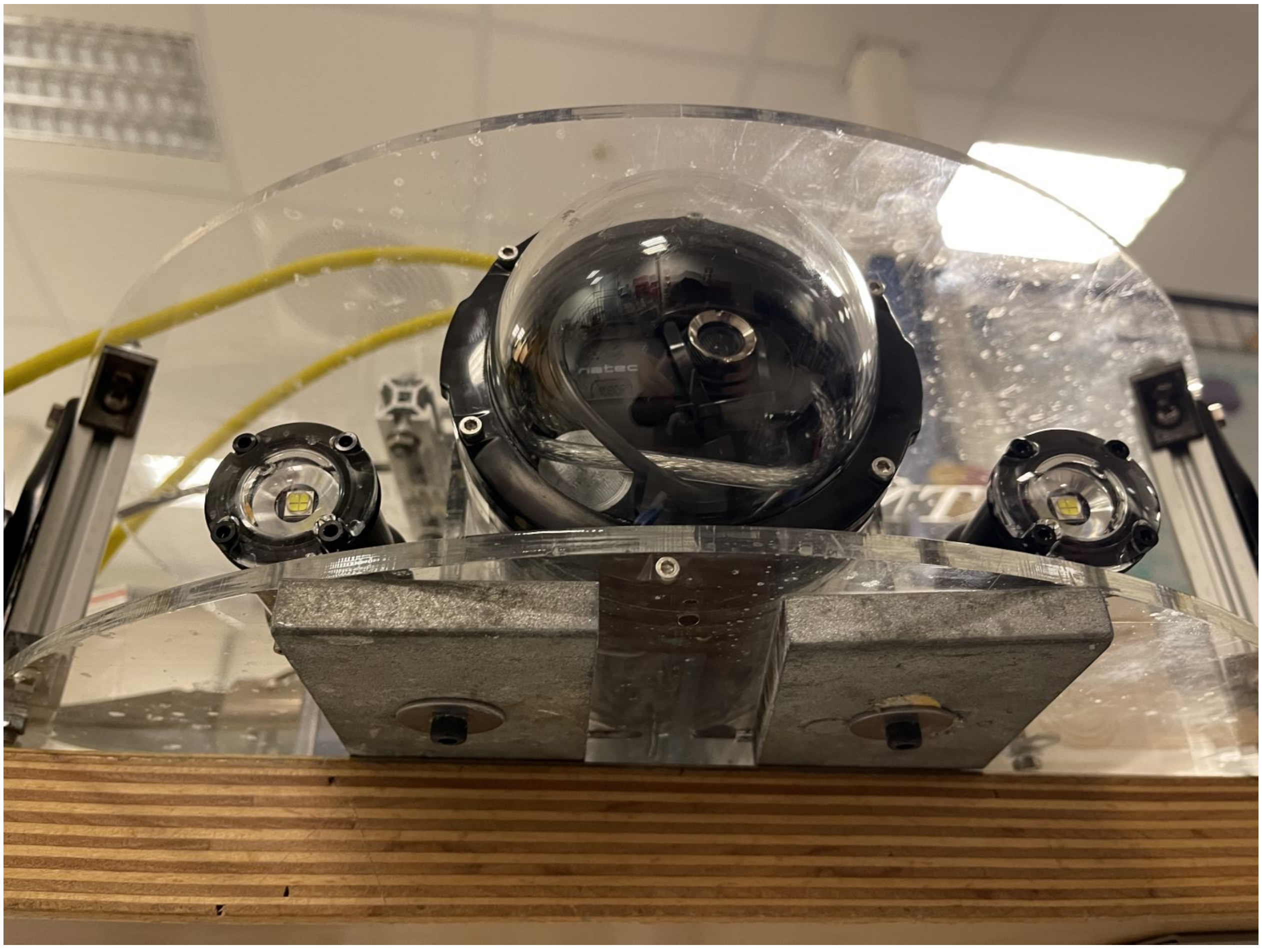


Figure 5.8: Ballast mounted to ROV

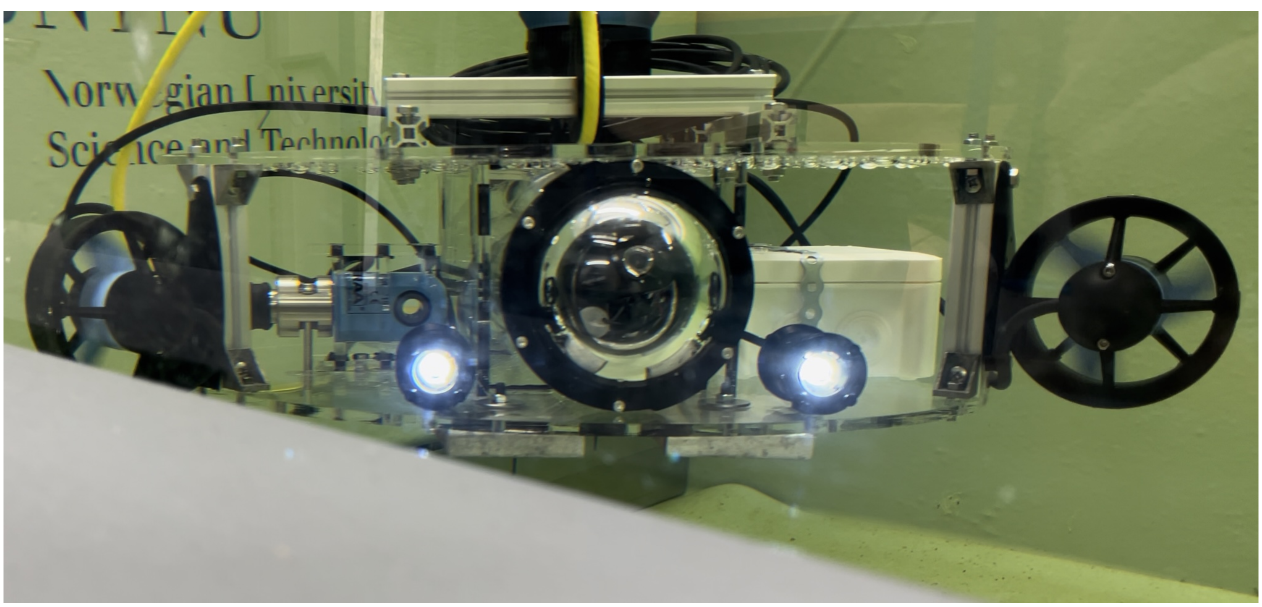


Figure 5.9: ROV submerged in water during testing

5.6 Test 3

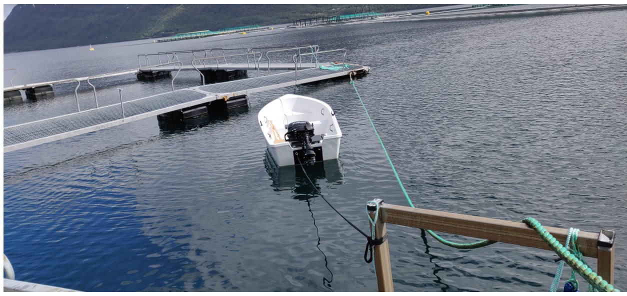
The ROV is primarily designed to be used to observe and inspect aquaculture conditions at sea, therefore it was essential to test the ROV in that environment. This testing allowed for testing of components that could not properly be tested in a simulation tank, like reading of salinity values, sonar data acquisition and depth and pressure testing of the overall structure.

This test took place in Storfjorden right outside of Glomset. At this location NTNU has a small boat with an outboard motor available to use, see Figure 5.10a. This allowed us to get to water where the depths reached up to about 100m, in addition to have underwater structures available to inspect. There was also several fish farming facilities, however we were not permitted to use the ROV in close proximity to those.

As we used a small Pioneer 13 boat, we did not have access to a high voltage output which would allow us to run power supplies to energize the ROV. Like we had done at previous testing in controlled environments at the modelation tank at NTNU. We therefore used four 12V - 75Ah batteries connected in series, to produce 48V, see Figure 5.10d. Whilst simultaneously taking out 12V from one of the batteries to power the communication card. By connecting an ethernet cable with RJ45 plugs to the communication interfacing box and to the topside computer, we were able to control the ROV from the GUI. See Figure 5.10b, 5.10c and 5.11 to see the equipment and the ROV from test 3.

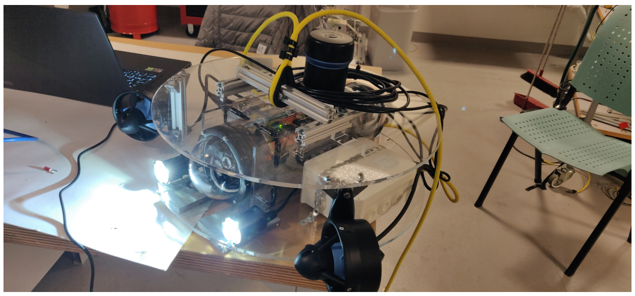
During the test several dives were completed starting at a couple of meters and then gradually increasing the depths as the test went on. During the tests everything worked as intended and all of the sensors were showing good and realistic data. The camera feed was stable, however as expected the FPS was a bit low to get a seamless user experience.

The testing ended during a gradual pressure increase during a dive at 70+ meters, where some of the components lost power. Many critical components lost power, including the communication board, which instantly made it possible to get any data or control the ROV further. We instantly turned off power, to prevent as much damage as possible if there was a leak. After



(a) The boat used for the test.

(b) The equipment used for test 3.



(c) ROV ready for test 3.

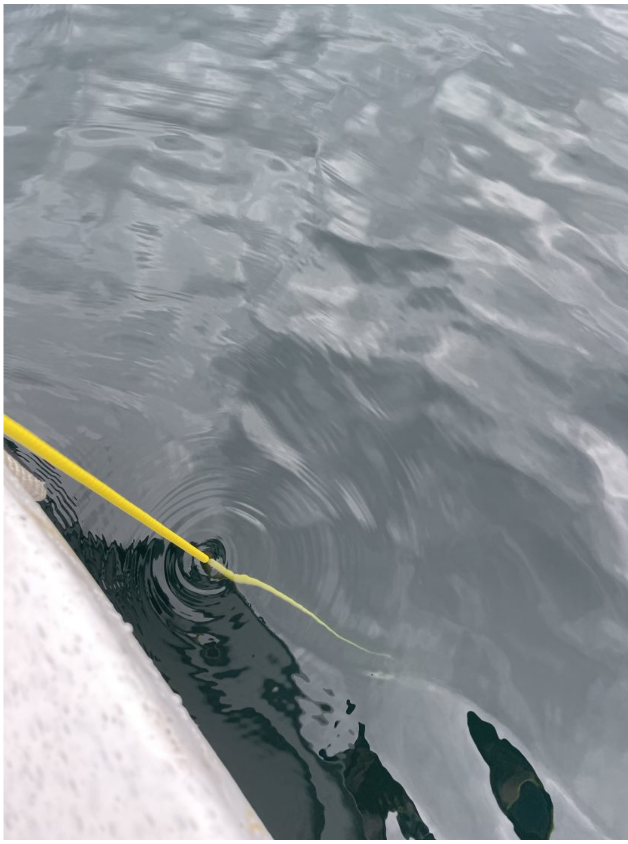
(d) Battery setup used for test 3.

Figure 5.10

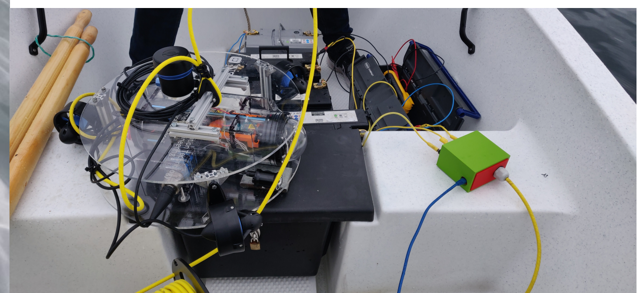
hoisting the ROV up to the boat again, we began troubleshooting the ROV. We could not regain control of the ROV at this point and decided to end test 3.

Further troubleshooting took place back at the lab at NTNU. During the troubleshooting all the components inside the enclosure were checked and none of them were broken. The problem was traced to the external box and specifically to the 16.5V converter. But, since all the components inside were encased in epoxy it was not feasible to find and fix the exact problem, and recast the external box with epoxy. The problem seems to be with either the converter itself or the cables going from the converter to the ROV. It seems that the pressure at the 70+ meters may have caused this, but without digging the components out of the epoxy its hard to conclude exactly.

When testing in water and operating the ROV using the GUI, we got good experience of how the user experience was, see Figure 5.12 for snapshot of testing. The numerical sensor values updates quickly enough for most purposes and is placed in a intuitive location. But, most noticeably, the camera feed has a static delay at around 1 second, in addition to only operating at a



(a) Tether cable in the sea



(b) ROV equipment rigged up and ready for test

Figure 5.11: Photos from Test 3

few fps. This combined with strong underwater currents, the control of the ROV could be challenging at times. However, after operating the ROV for extended amount of times, experience of how the ROV reacts to commands is gained, and controlling is easier.

Figure 5.12: Screenshot of the GUI during the test 3.

Chapter 6

Discussion

6.1 Technical results

This chapter will go over the technical results and discuss how the solutions performed, and what could have been done better.

6.1.1 Design

While working with the project it became clear that there was a lot of things that could have been done different when designing the ROV. The biggest thing that could have been done different is designing it with an additional external compartment in mind. For the current solution it was added midway due to space issues. It also helped with reducing electrical noise and heat inside the enclosure. If the batteries and motor controllers were also moved to a external box it would further reduce the heat and electrical noise. During sea testing an issue occurred with something inside the external box. However, due to to it being filled with epoxy, troubleshooting and fixing this problem would require us to remove the epoxy, which is quite a challenge. To prevent such an issue the external box could be replaced by a enclosure from BlueRobotics. This would allow easier access to the components inside, therefore making maintenance a lot easier. It would also make upgrading or changing out parts easier.

The overall size of the ROV could also have been reduced, as not all of the available surface area was used. But having this additional space is not necessarily bad, as it allows the implementation of additional sensors with relative ease and only minor modifications. When it comes to assembly the main thing that should have been done different is gluing the acrylic with acrylic glue. Instead of the current solution, which is using epoxy, as the acrylic glue would have ensured an even stronger hold. But based on the results from testing the epoxy worked fine and did not give any signs of loosening.

With the current design the rear aluminium extrusion has to be removed in order to access the BlueRobotics enclosure. To remove it four screws and bolts have to be taken out. This is fairly easy but somewhat time consuming. A solution that could simplify this could be to implement a quick disconnect system. This would be especially helpful when it comes to servicing the ROV.

Another area that can be improved is the length of the cables. Currently the cables are very long to make service easier, which was helpful during testing. However, the length also causes the cables to take up more space and is harder to manage. While the space is hard to do something with unless you want to compromise on the serviceability, the cable management could be improved with the use of cable clips that can be opened and closed. It should also be mentioned that the conductivity sensor's cable was kept extra long, due to it being expensive and the plugs were precast from factory.

6.1.2 Electronics

The original plan for supplying power was to exclusively use a DC-DC converter that could supply enough power to the system. However, after discussions with supervisors it was mentioned that the thrusters have a large current spike when starting and that could result in the RPi to reboot. When several thrusters are started they will attempt to pull high amounts of current for a short time. This will result in a high voltage drop over the tether cables, which in supplies less voltage over internal components in the ROV. Sensitive equipment, as the RPi, can not handle low voltages and will therefore shut down. Because of this it was decided that we should add batteries that could take over when such a large current surge occurs.

This decision came quite late and therefore the implementation could have been done better. From testing it seems to be working, but its unclear whether or not the batteries are actually taking over properly. Due to lack of time for testing we could not reach a conclusion on this. If we had planned on using batteries from the start, a battery management system could have been used. This would manage the batteries by keeping them in a safe operating area and preventing them from over-current, over-voltage, under-voltage and over-temperature. Currently there is no way to monitor the status of the batteries without opening the ROV, therefore this is a feature that should be added. This would give the operator better insight into the status of the ROV. A battery management system could also potentially eliminate the need for the DC to DC converters in the current setup. This is only possible if the management system allows high voltage inputs.

The voltage regulators in the current system should also be looked at. A voltage regulator lowers the voltage by converting the excess input voltage to heat. In the current system the 12V regulator works fine and doesn't produce much heat as the step down in voltage is low. The 5V regulator however produces a lot of heat and requires a big heat sink to stop it from overheating. A possible solution to this is to use a 5V DC to DC converter instead of a voltage regulator. The converter would take up more space but since it wont need a heat sink it should take up approximately the same amount of space.

The safety solutions that were implemented worked as intended. It is hard to say whether or not the leak sensors would detect a leak fast enough to allow us to save any components. This is because during testing we did not experience any leaks. The fuses worked well and none of the components inside the enclosure were destroyed. Currently blade style fuses are used, the same type of fuses used in cars. They worked great but could be replaced with smaller style fuses.

The wiring works as intended but as this is a prototype the cable management could have been done a lot better. By cutting the cables exactly to length and planning the cable routing better the space taken by cables could be reduced by a lot. Doing this would also make servicing the ROV easier. Another thing that should be added is a magnetometer this would allow us to integrate a compass in the GUI and therefore helping the to stay orientated in deep water.

6.1.3 Software

Below the implemented software solutions for the ROV system is discussed. Additionally, the feasibility of any potential improvements of the systems are considered.

Collision avoidance

The collision avoidance system was implemented to assist the operator with maneuvering in tight areas without crashing and damaging the ROV. We have determined that it was a good decision to implement the logic as an object, as difficult logic was abstracted into class methods. When using the functionality in the Python main script implementation was intuitive, and troubleshooting was simple. When testing the collision avoidance under a controlled environment, as the tank at the school, the functionality worked as intended. If an object was detected within a zone, movement towards that zone was prohibited. And this could be reset either automatically when the zone was cleared in the next scan cycle, or manually from user input in the GUI.

When testing the ROV under more demanding conditions, as the sea test, the collision avoidance system was harder to use properly. This comes from delay in the camera feed, coupled with low frame-rate, made it hard to maneuver properly. Additionally, underwater currents induced rotations of the ROV, which meant that interlocked zones was quickly not relevant. As the interlocked zone would quickly rotate in a way that the zone on the GUI did not correspond to the actual underwater obstruction. And an *open* zone, actually should instead have been interlocked. Another challenge with the collision avoidance system, was the rotation speed of the scanning. The ROV was operating quicker than the sonar was able to get updated information. We partly solved this by increasing the step size for each scan, but this would negatively affect the resolution and the quality of the sonar scan.

Communication

The serial communication was implemented in a fairly standard way, that worked in a stable way during the testing when connection had been established between the different devices. However as previously mentioned, the connection between the RPi and Arduino Uno would sometimes fail during the initialization of the connection. And an error indicating that the buffer array had overflown was given in the Python console. To fix this issue, a function that checks for incoming data, and clears the input buffer accordingly if the RPi is not yet ready to receive data, should be implemented. Alternatively, the order of initialization of the different communication between devices should be changed in a way that the Python serial connection happens at the same time as the Arduino is trying to initialize the connection.

The I2C and other serial connections worked as intended, and is implemented in a way that facilitates for future additions of sensors. Especially for more Aanderaa type sensors, that measures parameters such as oxygen and turbidity could easily be added to the Python program. But a USB hub, would have to be installed as all the USB ports on the RPi is currently in use.

The communication between RPi and the topside computer (GUI) is working as intended. To separate critical data, commands and camera feed to TCP and UDP, respectively, was a good decision. Three way handshake functionality guarantees with high certainty that no data packets are lost. And the UDP connection is optimal to save bandwidth when sending video frames, lost frames are no point in re-sending anyways, as those are in the past and irrelevant for the user.

Camera and light control

We had originally planned to use a Sentech machine vision camera suited for low visibility conditions. We performed multiple isolated tests with this camera, and a constant problem that occurred was that the frame size was very large, and not suitable for our tether communication. We had to compromise by reducing the quality and compressing the frames before sending them. This resulted with the quality of the images shown on the GUI was not as great as originally planned. By reducing the quality enough to get a usable frame rate, the camera quality was comparably the same as a normal 1080p web-camera. However, it could still be beneficial to use this camera, as when using the picture taking functionality, the Python scripts re-adjusts to not compress the frame, and a high quality image is saved to the RPi.

As previously mentioned in results, we had problems with the driver for the Sentech camera during the later stages of the project. It was therefore decided to switch to a 1080p USB camera, that required no drivers. This camera is not ideal for low light conditions, and we experienced that it was difficult to see contrasts when the ROV was tested at high depths. Even when the lights were set to maximum, underwater particles reflected the light and prohibited the camera to see at distance. A potential solution for this could be to place the subsea lights further apart and angle the lights in a way that both lights are focused on a single point, directly in front of the camera. For more advanced solutions, a control mechanism for the lights could be implemented, granting the user functionality to adjust the angle with the GUI.

Databases and historical information

Several functions were implemented to save relevant data for offline analysis after testing was completed. In the GUI a function was added allowing the user to take photos and videos of the camera feed during operation of the ROV. This data was saved locally on the RPi, and had to be transferred to the user computer after finished testing using either *SCP* from the terminal, or other programs as *WinSCP*. This could be improved by instead sending the photos using the same UDP communication, and instead saving the photos on the computer used to run the GUI. The video functionality could be done in a similar way, in addition to displaying the video frames on the GUI, the frames could be written to a video file, in a similar way this is done in the RPi. By implementing this logic, the user experience of accessing relevant data would be easier and consist of fewer steps.

The ROV does not have a function for saving sensor values currently. This could have been added by simply writing the received values in the GUI TCP connection to a csv file. By using the Python library *csv*, this could have been integrated to the system with a few lines. In addition to writing the value, information about which time that sample was taken should be written simultaneously for easier inspection later.

Graphical User Interface

The GUI was implemented with the functionality to operate the ROV under basic conditions. Using PyQt as the GUI software made it easy to design and use the functions we needed for the project, as the RPi on the other end of the communication is also designed in Python. The GUI had a very simplistic approach, containing all of the information and buttons needed for the project.

However, if as previously discussed, sensor values was stored in a database updated during operation. Historical plots could be visualized, constantly updating when new sensor values are read. These sensor plots could be visualized on multiple pages in the GUI, giving the operator an option to toggle what values are shown. In addition to this, logic could be implemented that gives further information or alerts about if values has a high rate of change, which could be important for values as salinity. As sometimes it could be difficult to notice areas in the water with different values.

6.2 Project accomplishments

This section goes over what the group learned and the unforeseen problems that were faced.

6.2.1 Distribution of work

The group had different preferences and experience when it came to working with software and hardware before the project started. One group member preferred working with hardware while the other preferred software. Due to this distributing the work was made easier. The work was split so that one group member worked on hardware while the other worked on software. There was of course some overlap and both group members learned a lot from each other while also gaining more knowledge on their assigned task. The group overall were satisfied with how the work distribution worked out.

6.2.2 Unforeseen consequences

During the project there were several unforeseen consequences, the one that caused the most problems was the amount time it took to order new parts and get them delivered. Another unforeseen issue was the implementation of the battery system. There was a lot of work on trying to get it work and while during testing all systems looked good the group is still unsure on whether or not it actually worked.

Chapter 7

Conclusions

The purpose for our project was to create a new prototype of a ROV for aquaculture inspection. ROVs for this purpose have been developed by other students at NTNU prior to our project, and through these developments, new desired functionality has been defined. Therefore, we got tasked with creating a ROV that was more lightweight, easier to work with and rated for deeper depths. Additionally, the ROV had to be fitted with sensors that measures aquaculture relevant parameters. With the functionality that we added, future groups can easily expand on our prototype and create the other systems that is required for the Aquaculture inspection platform, as the winch and platform itself.

From our proposed and defined goals set in the preliminary report, and from our results and discussion we can conclude that we created solutions that answered the thesis's main focus areas. The ROV can be easily controlled by other groups by using the same source code, and following the instructions defined in the user manual.

However, one of the planned improvements was not completed as originally planned. High quality live video streaming was not implemented as described in the pre-project report. As we had substantial problems with the driver for the camera, and we did not find time to solve the issues. Additionally, if the drivers are fixed, the communication would still struggle with sending high quality frames at a quick enough frame-rate for the user experience to be seamless.

Further work on the ROV and surrounding systems should be feasible. All parts, from the physical to software, was designed with usability and options to expand in mind. By addressing the challenges and limitations of our solution, the ROV could be implemented with the complete system and perform the required actions with high performance.

All things considered, the group believes the final ROV was a solid product, that addressed most of the system requirements. The project has given the group experience in terms of planning, cooperating and working through challenges. The ROV constantly evolved as new challenges appeared, however, through working systematically and with a solution-oriented focus, these challenges were solved and valuable experience and abilities were gained.

Appendices

A Preproject report

B Progress reports

C Gantt diagram

D Electrical drawings

E User Manual

F Arduino code

G Raspberry Pi code

H GUI code

I Meeting invitations

J Minutes of meeting

Bibliography

[1] Aanderaa. Conductivity sensor 5819. URL https://www.aanderaa.com/media/pdfs/d425_conductivity_sensor_5819.pdf.

[2] Arduino. Arduino uno rev3, . URL https://store.arduino.cc/products/arduino-uno-rev3?selectedStore=eu.

[3] Arduino. Wire library, . URL https://www.arduino.cc/en/reference/wire.

[4] Arduino. Serial.begin(), . URL https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/.

[5] Autodesk. Fusion 360.

[6] Biltema. Rechargeable icr18650 battery, 2950 mah.

[7] BlueRobotics. Bar30 high-resolution 300m depth/pressure sensor, . URL https://bluerobotics.com/store/sensors-sonars-cameras/sensors/bar30-sensor-r1/.

[8] BlueRobotics. Potted cable penetrator, . URL https://bluerobotics.com/store/cables-connectors/penetrators/penetrator-vp/.

[9] BlueRobotics. Basic esc, . URL https://bluerobotics.com/store/thrusters/speed-controllers/besc30-r3/.

[10] BlueRobotics. Fathom-x tether interface board, .

[11] BlueRobotics. Watertight enclosure for rov/auv 4inch series, . URL https://bluerobotics.com/store/watertight-enclosures/4-series/wte4-asm-r1/.

[12] BlueRobotics. Sos leak sensor, . URL https://bluerobotics.com/store/sensors-sonars-cameras/leak-sensor/sos-leak-sensor/.

[13] BlueRobotics. Lumen subsea light, . URL https://bluerobotics.com/store/thrusters/lights/lumen-r2-rp/.

[14] BlueRobotics. Ping 360 scanning imaging sonar, . URL https://bluerobotics.com/store/sensors-sonars-cameras/sonar/ping360-sonar-r1-rp/.

[15] BlueRobotics. T200 thruster, . URL https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/.

[16] BlueRobotics. Celsius fast-response, ±0.1°c temperature sensor (i2c), . URL https://bluerobotics.com/store/sensors-sonars-cameras/sensors/celsius-sensor-r1/.

[17] BlueRobotics. Bluerobotics, . URL https://bluerobotics.com/.

[18] BlueRobotics. Ping viewer documentation, . URL https://docs.bluerobotics.com/ping-viewer/#installing-and-running-the-application.

[19] Anthony A. DiBiase. Electromagnetic interference sources and their most significant effects. URL https://interferencetechnology.com/electromagnetic-interference-sources-and-their-most-significant-effects/.

[20] doc.qt.io. Qt designer manual. URL https://doc.qt.io/qt-5/qtdesigner-manual.html.

[21] ELFA. Rsdw60f-15 - dc/dc-omformer 9 ... 36v 15v 4a 60w, mean well. URL https://www.elfadistrelec.no/no/dc-dc-omformer-36v-15v-4a-60w-mean-well-rsdw60f-15/p/30230089?trackQuery=cat-DNAV_PL_10010202&pos=3&origPos=465&origPageSize=50&filterapplied=filter_Utgangsspenning+1~~V%3d15%26filter_Utgangsstr%c3%b8m+1~~A%3d4&track=true.

[22] Pamela Fox. Transmission control protocol (tcp). URL https://www.khanacademy.org/computing/computers-and-internet/

xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp#:~:text=The%20Transmission%20Control%20Protocol%20(TCP,duplicate%20packets%2C%20and%20corrupted%20packets.

[23] GeeksforGeeks. I2c communication protocol. URL https://www.geeksforgeeks.org/i2c-communication-protocol/#:~:text=I2C%20stands%20for%20Inter%2DIntegrated,protocol%20for%20short%2Ddistance%20communication.

[24] HILGRAEVE. Hyperterminal trial.

[25] Rustom Jehangir. Bluerobotics ms5837 library, . URL https://github.com/bluerobotics/BlueRobotics_MS5837_Library.

[26] Rustom Jehangir. Bluerobotics tsys01 temperature sensor library, . URL https://github.com/bluerobotics/BlueRobotics_TSYS01_Library.

[27] JetBrains. Clion, . URL https://www.jetbrains.com/clion/promo/?source=google&medium=cpc&campaign=11959979214&gclid=Cj0KCQiApL2QBhC8ARIsAGMm-KHSiCNnF4caJnbNrukvx3cgGSq7WLpLVcG7AMjo3X0yRbdtQuLvZWYaAiItwcB.

[28] JetBrains. Pycharm, . URL https://www.jetbrains.com/pycharm/.

[29] Fendadis John. Advantages and disadvantages of the tcp/ip model. URL https://www.tutorialspoint.com/Advantages-and-Disadvantages-of-the-TCP-IP-Model.

[30] Jennifer Kennedy. Salinity: Definition and importance to marine life. URL https://www.thoughtco.com/salinity-definition-2291679#:~:text=Salinity%20can%20affect%20the%20density,regulate%20its%20intake%20of%20saltwater.

[31] LawtonChuck Moozakis Linda Rosencrance, George. User datagram protocol (udp). URL https://www.techtarget.com/searchnetworking/definition/UDP-User-Datagram-Protocol#:~:text=User%20Datagram%20Protocol%20(UDP)%20is,provided%20by%20the%20receiving%20party.

[32] EMC Bayswater Pty Ltd. A guide to electromagnetic compatibility (emc) testing methods. URL https://www.emcbayswater.com.au/blog/emc-testing/commercial-emc-testing/guide-electromagnetic-compatibility-emc-testing-methods/.

[33] Anas Mazouni. ping360$_{s}onar.URL$.

Mouser. Uwe-12/6-q48nb-c. URL https://no.mouser.com/ProductDetail/Murata-Power-Solutions/UWE-12-6-Q48NB-C?qs=ElE%2F8fqQeYgYGfPEPkQx2A%3D%3D.

NickNothom, jaxxzer, patrickelectric, Williangalvani, ES-Alexander, and rjehangir. ping-python, . URL https://github.com/bluerobotics/ping-python.

NickNothom, jaxxzer, patrickelectric, Williangalvani, ES-Alexander, and rjehangir. ping-protocol, . URL https://github.com/bluerobotics/ping-protocol.

PCSCHEMATIC. Pcschematic. URL https://www.pcschematic.com/en/.

Raspberry Pi. Raspberry pi 4 tech specs. URL https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

Omron Sentech. Stc-mca503usb. URL https://automation.omron.com/en/mx/products/family/STUSB3/STC-MCA503USB.

Sonar-info. Target strength. URL http://www.sonar-info.info/p278/TS.pdf.

techopedia. Communication protocol. URL https://www.techopedia.com/definition/25705/communication-protocol.

Ultimaker. Ultimaker cura. URL https://ultimaker.com/software/ultimaker-cura.

Wikipedia. Buoyancy, . URL https://en.wikipedia.org/wiki/Buoyancy.

Wikipedia. Display resolution, . URL https://en.wikipedia.org/wiki/Display_resolution.

Wikipedia. Electromagnetic compatibility, . URL https://en.wikipedia.org/wiki/Electromagnetic_compatibility.

Wikipedia. Electromagnetic interference, . URL https://en.wikipedia.org/wiki/Electromagnetic_interference.

Wikipedia. Frame rate, . URL https://en.wikipedia.org/wiki/Frame_rate.

Wikipedia. Gantt chart, .

Wikipedia. Machine vision, . URL https://en.wikipedia.org/wiki/Machine_vision.

Wikipedia. Raspberry pi os, .

Wikipedia. Communication protocol, . URL https://en.wikipedia.org/wiki/Communication_protocol#:~:text=A%20communication%20protocol%20is%20a,and%20possible%20error%20recovery%20methods.

Wikipedia. Osi model, . URL https://en.wikipedia.org/wiki/OSI_model.

Wikipedia. List of network protocols (osi model), . URL https://en.wikipedia.org/wiki/List_of_network_protocols_(OSI_model).

Wikipedia. Target strength, . URL https://en.wikipedia.org/wiki/Target_strength#:~:text=The%20target%20strength%20or%20acoustic,as%20a%20number%20of%20decibels.&text=Target%20strength%20(TS)%20is%20equal,cross%20section%20is%204%CF%80%CF%83bs.

Wikipedia. Sonar, . URL https://en.wikipedia.org/wiki/Sonar#Passive_sonar.

Wikipedia. Conductivity (electrolytic), . URL https://en.wikipedia.org/wiki/Conductivity_(electrolytic).

Wikipedia. Salinity, . URL https://en.wikipedia.org/wiki/Salinity).

Wikiwand. Arduino. URL https://www.wikiwand.com/en/Arduino.

RF Wireless World. Sonar vs radar | difference between sonar and radar. URL https://www.rfwireless-world.com/Terminology/SONAR-vs-RADAR.html#:~:text=SONAR%20stands%20for%20SOund%20Navigation,above%20the%20land%20or%20sea.

Appendix A

Preproject report

FORPROSJEKT - RAPPORT
FOR BACHELOROPPGAVE

TITTEL:
Forprosjekt rapport for ROV- Remotely Operated Underwater Vehicle

KANDIDATNUMMER(E):				
Tony Paulsen **Petter Henriksen**				
DATO: **19.01.2022**	EMNEKODE: **IELEA2920**	EMNE: **Bacheloroppgave**		DOKUMENT TILGANG: - Åpen
STUDIUM: **ELEKTROINGENIØR-AUTOMATISERING OG ROBOTIKK**			ANT SIDER/VEDLEGG: 10/4	BIBL. NR: - Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
NTNU i Ålesund v/Lars Christian Gansel, Ottar L. Osen

OPPGAVE/SAMMENDRAG:

NTNU i Ålesund ønsker å videreutvikle en USV plattform, et ubemannet overflatefartøy som skal bære en ROV og vinsj for å kunne observere undervanns akvakulturer. ROVen skal ha kamera, sensorer for måling av diverse verdier og thrustere for bevegelse. Produksjon av en prototype av denne ROVen er gitt som bachelor oppgave til studenter som studerer Automatisering og Robotikk. Denne forprosjektrapporten er en prosjektbeskrivelse av denne bacheloroppgaven.

Bacheloroppgaven skal utrede et konsept for en slik ROV med fokus på integrering av kamera, sensorer og en brukervennlig GUI. Viktige designkriterier vil være vekt, kostnad og brukbarhet av bilde og sensorverdier for forskning på akvakulturer. Oppgaven skal også utføres på en måte som gjør det lett for fremtidige grupper å videreutvikle ROVen videre.

Prototypen skal testes i vann på NTNU test lokaler og ute på lokasjon for å kunne demonstrere at design og implementasjon av nye funksjoner fungerer som angitt.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved NTNU i Ålesund.

Postadresse	Besøksadresse	Telefon	Telefax	Bankkonto
Høgskolen i Ålesund	Larsgårdsvegen 2	70 16 12 00	70 16 13 00	7694 05 00636
N-6025 Ålesund	**Internett**	**Epostadresse**		**Foretaksregisteret**
Norway	www.hials.no	postmottak@hials.no		NO 971 572 140

INNHOLD

1 INNLEDNING

I Norge og spesielt på Vestlandet er fiske oppdrett en viktig del av økonomien, men oppdretts bedrifter har møtt på problemer i form av bakterier og lus. For å løse dette problemet trenger marine biologer redskap for å kunne observere fisken og miljøet rundt dem. For å hjelpe med dette skal vi videreutvikle en ROV prototype. ROVen skal ha et kamera og diverse sensorer for å kunne samle data, den skal også ha thrustere for bevegelse slik at man kan flytte kameraet der man ønsker. ROV skal i tillegg ha en GUI som gjør den brukervennlig, og viser relevante data på en oversiktlig måte. ROVen er en del av et større system som i består av en vinsj og en flytende plattform.

2 BEGREPER

- ROV (Remotely Operated Underwater Vehicle), betegnelse for fjernstyrt undervanns kjøretøy.
- GUI (Graphical User Interface), grafisk brukergrensesnitt, ett brukergrensesnitt for dataprogram som lar brukeren benytte utstyr som tastatur og datamus for å lese data og sende kommandoer.

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)
517297 – Tony Paulsen – Prosjektleder 510317 – Petter Henriksen – Sekretær

Tabell: Studentnummer(e) for alle i gruppen som leverer oppgaven for bedømmelse i faget IELEA2920

3.1.1 Oppgaver for prosjektgruppen – organisering

Alle gruppemedlemmene har samme ansvar for gjennomføring av prosjektet, både dokumentasjon og arbeid på prosjektet. Gruppemedlemmene skal til alle tider holde hverandre oppdatert om fremdrift og eventuelle avvik.

3.1.2 Oppgaver for prosjektleder

- Oppdatere Gantt-diagram
- Møteinnkalling med agenda
- Lede møter med styringsgruppen

3.1.3 Oppgaver for sekretær

- Reservasjon av eventuelt møterom for styringsgruppe-møter
- Skrive og distribuere møtereferat
- Skrive framdriftsrapport

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Styringsgruppen består av Ottar Osen og Lars Christian Gansel fra NTNU i Ålesund.

4 AVTALER

4.1 Avtale med oppdragsgiver

Oppgaven går ut på å videreutvikle og forbedre en ROV, arbeidet bygger på tidligere studentprosjekter som har blitt utført ved NTNU i Ålesund. Tidligere oppgaver rundt ROVen har hatt tverrfaglige grupper, der arbeidsoppgaver som produktdesign og kontrollsystemer ble utført av egnede personer. Gruppen vår består av kun automatiseringselever, og derfor blir design/ valg av materiell og lignende forenklet slik at gruppen har mulighet til å fokusere på mer relevante arbeidsoppgaver. Fra møte med styringsgruppen ble det avtalt at arbeidsoppgavene var veldig åpne, og at vi får stor valgfrihet til å velge ønskelige forbedringer på ROVen.

4.2 Arbeidssted og ressurser

Prosjektet skal utføres ved NTNU i Ålesund. Her jobber begge veilederne fra styringsgruppen, som er gunstig for tilgang til hjelp på kort varsel. Veilederne har uttrykt mulighet for testing ved lokasjon om dette er ønskelig. Møte med veilederne skal ta stede enten digitalt eller på avtalt lokasjon annenhver tirsdag.

4.3 Gruppenormer – samarbeidsregler – holdninger

Gruppen har avtalt å innføre en kjernetid mellom 09:00 til 14:00 hver ukedag. Dette er for å sikre god fremgang på prosjektet og å være tilgjengelige for hverandre ved spørsmål og diskusjoner. Likevel settes det som mål at gruppemedlemmene skal jobbe minimalt 37.5 timer hver uke, ekskludert pauser. I tillegg er det utarbeidet en samarbeidsavtale for å sette hverandre ansvarlige for gode holdninger innen gruppen, se vedlagt.

Gruppemedlemmene skal behandle hverandres meninger og synspunkt med respekt. Alle gruppemedlemmene skal arbeide nøyaktig, ærlig og være punktlig iht. avtaler og møter.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Prosjektet tar basis i tidligere prototyper av ROVer utarbeidet av studenter ved NTNU. Ettersom gruppemedlemmene ikke har mye erfaring innen design av fartøy, avgrenses oppgaven til å lage en veldig simpel prototype for ett ROV fartøy, som ønskelige sensorer kan monteres på og videre brukes/ testes.

Etter møte med styringsgruppen fikk vi informert om mulige utvidelser av ROV som var ønskelig. Fra møtet og etter intern diskusjon innen gruppen bestemte vi oss for å dele opp prosjektet i tre hovedtemaer, kamera, sonar og kombinasjonssensor.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Avsnittene under beskriver i mer detalj kravene og målene som settes for hver enkelt av de tre hovedtemaene som prosjektet består av. Først forklares kravene til oppgradering av kamera, deretter beskrives kravene for sonar og kombinasjonssensor. I tillegg til sonar sensor og kombinasjonssensor montering og avlesning av data, -beskrives det at ytterligere logikk som skal implementeres. Men en presis formulering av løsning er ikke spesifisert, dette er fordi det kreves mye forsking og utreding som skal utføres etter forprosjektrapporten for å finne realistiske og gode logikk implementasjoner.

5.2.1 Forbedre kamera

Fra tidligere iterasjoner av prosjektet er det nå montert et vanlig Webkamera. Vi ønsker å dimensjonere, montere og integrere ett kamera som er bedre egnet for dårlige lysforhold og som har mulighet for å skille mellom kontraster effektivt. En del av prosjektet blir å finne et slikt kamera som oppfyller kravene. Målet er også å finne et kamera som er bedre eller like bra som GoPro kameraene som blir brukt på andre ROVer som allerede er i bruk av biologi avdelingen. Men GoPro kameraene har egne program som er spesial-tilpasset for dem. Ved valg av nytt kamera som er bedre egnet for egenskapene som vi behøver, vil sending av informasjon til GUI en utfordring.

5.2.2 Sonar

Montere og integrere en sonar sensor på undersiden av ROV. Første steg blir å avlese dataen fra sonaren på en måte som er intuitivt på GUI, som i hovedsak vil gi lokasjon på fisk. Videre skal det utvikles kollisjons-beskyttelse for å sikre ROV mot skade under operasjon. Der vi ønsker å gi alarm på GUI, og utvikler logikk som aktiverer variabler som kan brukes til forrigling under kjøring av thrustere i retningen det oppdages objekt(er).

Ett alternativ for sonar er Ping360 fra BlueRobotics.

5.2.3 Kombinasjonssensor

Montere og integrere en kombinasjonssensor som måler flere viktige parameter som er essensielle for akvakultur. Målevariabler som saltholdighet, pH, vann-hardhet og vann-konduktivitet er relevante for ROVen. Andre variabler som oksygen og temperatur er allerede utredet og integrert på en god måte, og behøves ikke endres. Dataen fra denne sensoren skal behandles og det skal være mulighet for å vise infoen i GUIen. Videre skal det utvikles logikk som består av *moving-average* avlesninger av kritiske verdier for akvakultur, dersom endringene er store over kort tid, eller det oppstår verdier som er benevnet som kritiske, skal dette vises med alarmer og visuelle hjelpemiddel på GUI.

Ett alternativ for en kombinasjonssensor er Model 5819 fra Aanderaa.

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Gruppearbeidet vil utføres opp mot mål som er satt i Gantt diagrammet. Der aktivitetene som er oppført først skal prioriteres høyest. Gruppemedlemmet som er ansvarlig for en spesifikk aktivitet har fullt ansvar for å fullføre denne aktiviteten innen tiden, ellers gi beskjed om fristen ikke kan holdes.

Ettersom gruppen består av to medlemmer vil begge medlemmer jobbe med alle arbeidsoppgavene, der det byttes på hvilket medlem som har ansvar. Det er antatt at noen aktiviteter vil gå over fristen, eller aktiveten mangler informasjon eller komponenter, i slike situasjoner skal arbeidet justeres dynamisk slik at det blir alltid jobbet med relevante arbeidsoppgaver.

5.4 Informasjonsinnsamling – utført og planlagt

I løpet av forprosjekt rapport arbeidet har gruppemedlemmene studert relevant litteratur for prosjektet. Medlemmene har lest tidligere rapporter og dokumentasjon utarbeidd fra både studenter og faglærere ved NTNU, i tillegg har medlemmene studert informasjon fra tilsvarende internasjonale prosjekter.

Informasjonsinnsamlingen er svært viktig for å bygge ett godt kunnskaps fundament slik at beslutninger som valg av arbeidsoppgaver blir gjort slik at ROVen blir forbedret, samtidig som at løsningen som blir startet er overkommelig for to studenter som en bachelor oppgave.

Denne informasjonsinnsamlingen skal utføres kontinuerlig gjennom prosjektets gang. Medlemmene skal begynne fra starten av med å tilføre viktige og relevant informasjon under teoretisk bakgrunn på rapporten.

5.5 Vurdering – analyse av risiko

Ettersom gruppen består av to medlemmer vil begge medlemmer jobbe med alle arbeidsoppgavene, der det byttes på hvilket medlem som har ansvar. Det er antatt at noen aktiviteter vil gå over fristen, eller aktiveten mangler informasjon eller komponenter, i slike situasjoner skal arbeidet justeres dynamisk slik at det blir alltid jobbet med relevant arbeidsoppgaver.

Vi har utviklet en risikomatrise som viser risikoen for ulike aspekt av prosjektet, der en høy risiko betyr at det er stor sannsynlighet for at situasjonen som oppstår skaper høy tidsforsinkelse i prosjektet. Risikomatrisen skal utvikles videre gjennom prosjektet når nye situasjoner må tas i vurdering.

5.6 Hovedaktiviteter i videre arbeid

A. Hovedaktivitet: Utredning av ROV konsept
B. Hovedaktivitet: Utstyrsanskaffelse
C. Hovedaktivitet: Bygge prototype
D. Hovedaktivitet: Programvare utvikling
E. Hovedaktivitet: Integrasjon av alle del-systemer
F. Hovedaktivitet: Testing av protype
G. Hovedaktivitet: Fullføre rapport og endelig innlevering

En mer detaljert versjon av dette som inneholder tidsrammer og ansvars person vil være i Gantt-diagrammet

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Hovedplanen for prosjektet blir satt i form av Gantt-diagrammet. I dette diagrammet settes start -og stopp dato for hver enkelt aktivitet. Diagrammet viser også hvilket medlem som står med ansvar for hver enkelt aktivitet. Gantt-diagrammet skal utredes med en hierarkisk struktur, der store aktiviteter skal deles ned i mindre og mindre aktiviteter. Dette resulterer i god kontroll over prosjektet, i tillegg til at aktivitetene ikke virker uoverkommelige.

5.7.2 Styringshjelpemidler

For rapportskriving i LaTeX benytter vi Overleaf for å kunne enkelt arbeide samtidig, samtidig ha tilgang til hjelpemiddel som versjonskontroll og kommentering av tekst.

Til Gantt diagram brukes nettsiden *teamgantt.com*.

5.7.3 Utviklingshjelpemidler

For utvikling og simulering av data fra sonar sensor vil gruppen ha behov for Matlab.
For programmering av sensorer som er tilkoblet Arduino benyttes Arduino IDE og jetbrains Clion.
For kjøring av GUI benyttes NetBeans IDE.

5.7.4 Intern kontroll – evaluering

Prosjektleder har ansvar for utvikling og oppdatering av Gantt skjema minst en gang i uken.

Sekretær har ansvar for skriving av framdriftsrapporter.

Samtaler mellom gruppemedlemmene skal utføres daglig.

5.8 Beslutninger – beslutningsprosess

Konkrete rammevilkår for prosjektet bestemmes i møte mellom prosjektgruppen og styringsguppen.

Skal signifikante endringer utføres fra ønske av prosjekt -eller styringsguppen skal dette bli tatt opp i ett formelt møte.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

- Gantt-diagram
- Møteinnkallinger
- Møtereferater
- Framdriftsrapporter (m/ referat av møter)
- Underveispresentasjon
- Videopresentasjon av ferdig prosjekt
- Risikoanalyse

Alle dokumentert blir lastet opp i teams slik at de er sikkerhetskopiert

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

- Oppstarts møte 13.01.2022 klokken 13:00 med prosjekt gruppe, Ottar L. Osen og Lars Christian Gansel
- Planlagt møte annenhver tirsdag fra og med 01.02.22. Disse møtene gjør at prosjekt gruppen kan informere styringsgruppen om framdriften i prosjektet og be om veiledning fra styringsgruppen
- Sekretær sender møtereferat på mail så for som mulig til alle deltagere etter møte

7.1.2 Prosjektmøter

Prosjekt gruppen består bare av 2 personer og gruppen vil jobbe tett sammen gjennom det meste av prosjektet derfor er ikke det nødvendig med slike møter.

Dersom det oppstår problem i samarbeidet, kan det bli aktuelt med slike møter

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)

Før møter vil det bli laget en framdriftsrapport som viser arbeidet som skulle vært utført og det som faktisk ble gjort. Rapporten vil inneholde eventuelle endringer og avvik fra plan. I tillegg vil også arbeidet som skal jobbes med til neste møte bli presentert. Denne rapporten vil bli sendt til styringsgruppen dagen før møtet. Oppdatert Gantt-diagram vil bli sendt sammen med møte innkalling. Møtereferat blir sendt ut til alle deltakere etter hvert møte.

7.2.2 Møtereferater

Referatene vil inneholde:

- Navn på deltakere
- Framgang fra forrige møte
- Eventuelle avvik og endringer i prosjektet
- Mål til neste møte

8 PLANLAGT AVVIKSBEHANDLING

Dersom det oppstår avvik, vil gruppen stille disse spørsmålene i en intern diskusjon.

- Kan avviket fikses med mer ressurser?
- Kan avviket unngås?
- Kan avviket løses med ekstern hjelp?
- Kan arbeidsoppgaven bli byttet ut slik at man unngår avviket?

Eventuelle endringer som gruppen kommer fram til vil bli sendt til styringsgruppen slik at det kan komme eventuelle innspill før en endelig avgjørelse

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

- Utstyr / programvare eller andre spesielle ressurser som en vanligvis ikke har tilgang til og som er nødvendig for å gjennomføre prosjektet
- Eventuelt spesialutstyr / programvare som det søkes om innkjøp av- begrunnes
  (Vanligvis vil det være oppdragsgivers ansvar å stille slikt utstyr og programvare til disposisjon for prosjektgruppen)

Gruppen må finne og kjøpe inn alle deler som trengs for ROVen dette inkluderer sensorer, sonar og kamera. Utstyret trenger en trygg oppbevarings lokasjon under prosjektet. Det trengs også en plass for testing av ROVen, der vi behøver en vanntank for tidlige tester, og reise til en lokasjon for full skala test. Utstyr som blir essensielt som vi ikke har tilgjengelig og som vi antar må bestilles er

- 360 Sonar
- Aanderaa Sensor
- Nytt kamera

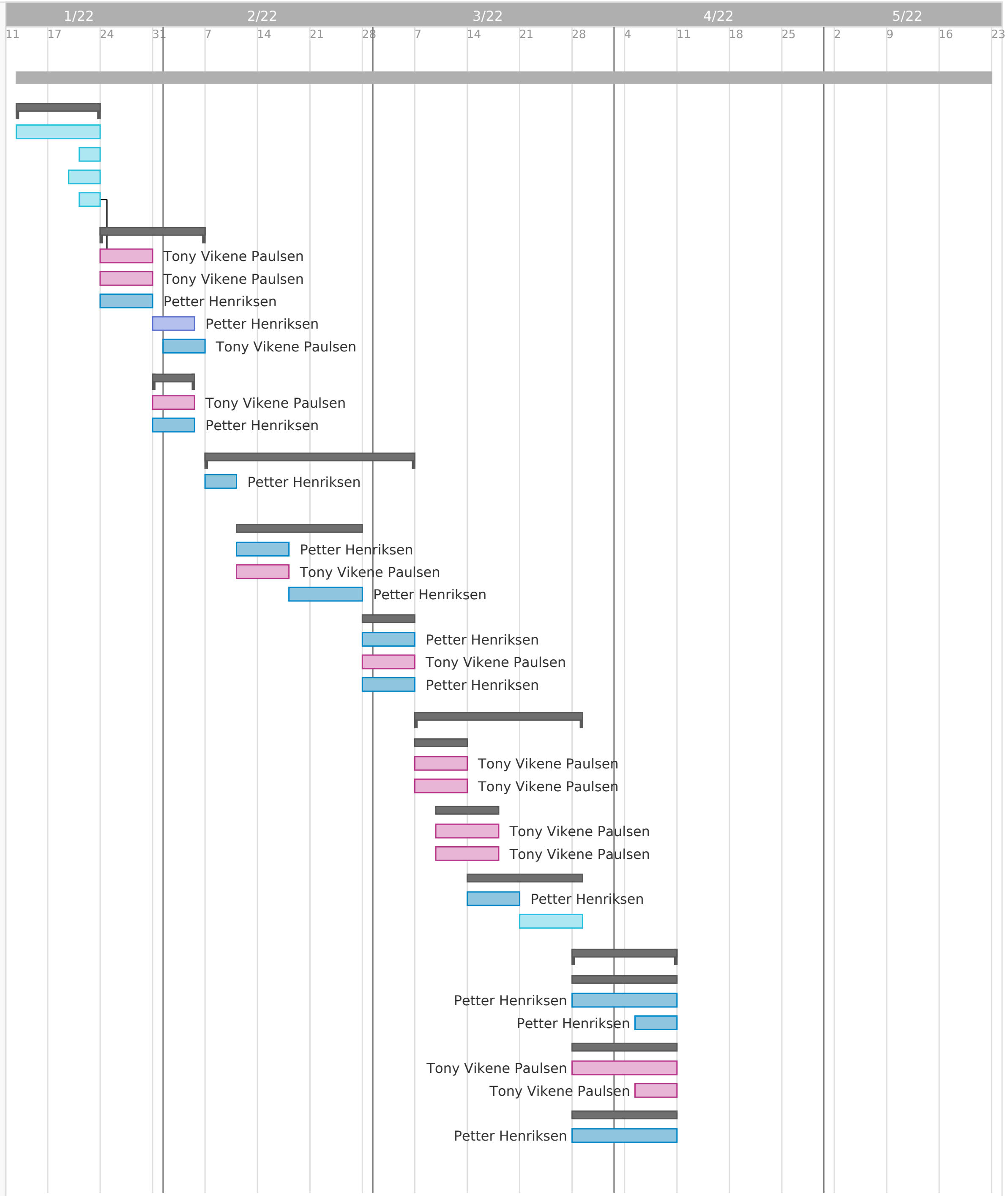
VEDLEGG

Vedlegg 1	Gantt-diagram
Vedlegg 2	Risikomatrise
Vedlegg 3	Samarbeidsavtale
Vedlegg 4	Møtereferat, første møte med styringsgruppe 13.01.22

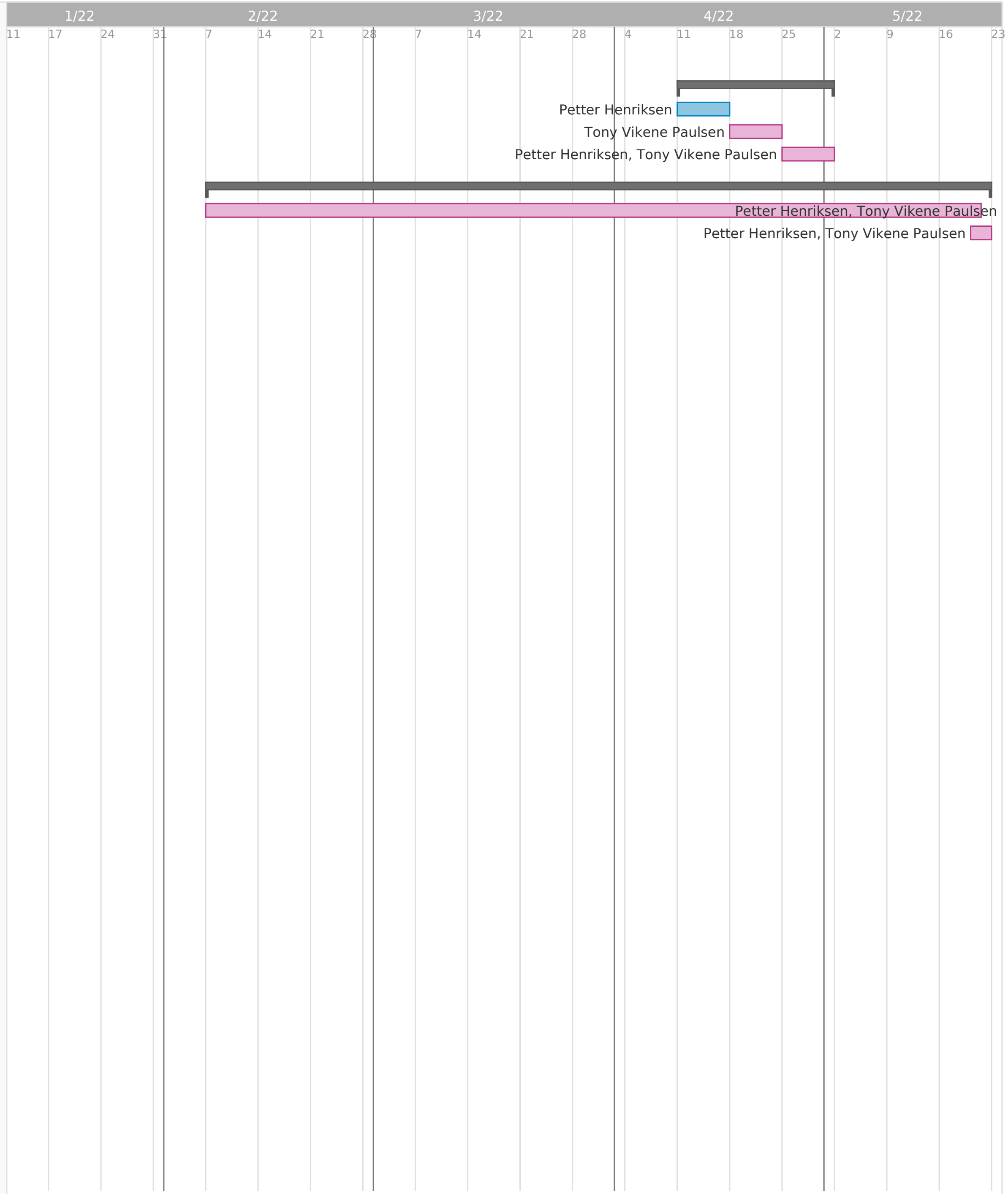
Vedlegg 1

Gantt-diagram

	start	end	0h	0%
IELEA2920_ROV				
Forprosjektrapport	**01/12/22**	**01/21/22**	**0h**	**0%**
Definere problemstillinger	01/12	01/21	0	0%
Lage risikomatrise	01/20	01/21	0	0%
Lage Gantt-diagram	01/19	01/21	0	0%
Kombinere alle dokument	01/20	01/21	0	0%
Utredning av ROV konsept	**01/24/22**	**02/04/22**	**0h**	**0%**
Bestemme sonar type	01/24	01/28	0	0%
Bestemme komb. sensor type	01/24	01/28	0	0%
Bestemme kamera	01/24	01/28	0	0%
Bestemme utforming av prototype sk...	01/31	02/03	0	0%
Vurdere plassering av sensorer	02/01	02/04	0	0%
Utstyrsanskaffelse	**01/31/22**	**02/03/22**	**0h**	**0%**
Bestille sensorer	01/31	02/03	0	0%
Bestille annet utstyr	01/31	02/03	0	0%
Bygge prototype	**02/07/22**	**03/04/22**	**0h**	**0%**
Demontere gammel ROV	02/07	02/09	0	0%
Montere gammelt utstyr	-	-	0	0%
Tilpasse prototype for sensorer	**02/10/22**	**02/25/22**	**0h**	**0%**
Lage montasje for sonar	02/10	02/16	0	0%
Lage montasje for komb. sensor	02/10	02/16	0	0%
Lage montasje for kamera m/ vann...	02/17	02/25	0	0%
Montere nye sensorer	**02/28/22**	**03/04/22**	**0h**	**0%**
Montere sonar	02/28	03/04	0	0%
Montere komb. sensor	02/28	03/04	0	0%
Montere kamera	02/28	03/04	0	0%
Programvare utvikling	**03/07/22**	**03/28/22**	**0h**	**0%**
Kombinasjons sensor	**03/07/22**	**03/11/22**	**0h**	**0%**
Program for avlesning data	03/07	03/11	0	0%
Program til system	03/07	03/11	0	0%
Sonar	**03/09/22**	**03/16/22**	**0h**	**0%**
Program for avlesning data	03/09	03/16	0	0%
Program til system	03/09	03/16	0	0%
Kamera	**03/14/22**	**03/28/22**	**0h**	**0%**
Vise kamera i GUI	03/14	03/18	0	0%
Tilpasse kode for system	03/21	03/28	0	0%
Integrasjon av alle del-systemer	**03/28/22**	**04/08/22**	**0h**	**0%**
Integrasjon av komb. sensor	**03/28/22**	**04/08/22**	**0h**	**0%**
Vise sensor verdier	03/28	04/08	0	0%
Gi beskjed om ending av snittverdi...	04/05	04/08	0	0%
Integrasjon av sonar data	**03/28/22**	**04/08/22**	**0h**	**0%**
Vise data som bilde	03/28	04/08	0	0%
Gi beskjed om kollisjonsfare	04/05	04/08	0	0%
Integrasjon av kamera	**03/28/22**	**04/08/22**	**0h**	**0%**
Vise kamera i GUI	03/28	04/08	0	0%



Testing av prototype	**04/11/22**	**04/29/22**	**0h**	**0%**
Testing av alle sensorer på GUI	04/11	04/15	0	0%
Testing av implementert logikk	04/18	04/22	0	0%
Full skala test på lokasjon	04/25	04/29	0	0%
Fullføre rapport	**02/07/22**	**05/20/22**	**0h**	**0%**
Fullføre rapport i LaTeX	02/07	05/19	0	0%
Kombinere LaTeX m/ alle vedlegg	05/19	05/20	0	0%



Vedlegg 2

Risikomatrise

Innvirkning

Sannsynlighet

	Neglisjerbart	Liten	Moderat	Betydelig	Alvorlig
Svært sannysnlig					
Sannsynlig			1	4	
Mulig		3			5,7
Usanssynlig			2	6	
Svært usannsynlig					8

Sannsynlighet beskriver hvor ofte en slik feil typisk vil oppstå.
Innvirkning beskriver hvor mye ekstra tid som vil medføre ved en feil.
Fargen gir ett overslag over hvor utfordrende denne feil typen er.

Risiko-tabell

Nr.	Identifiserte risikoer	Dato for risiko oppdagelse
1	Sykdom i gruppen (COVID-19)	17.01.2022
2	Nedstenging av arbeidslokale	17.01.2022
3	Problemer med sensor kalibrering	17.01.2022
4	Mangel på tid pga lang leverings tid på deler	19.01.2022
5	Feil med vanntetting av elektronikk	19.01.2022
6	Mangel på monterings-verktøy	19.01.2022
7	Mangel på nødvendige deler som feks kamera sensorer	19.01.2022
8	Ødeleggelse av sensorer (feil polaritet)	23.01.2022
9		
10		
11		
12		

Vedlegg 3

Samarbeidsavtale

Samarbeidsavtale

Leveranse

1. Alle møter til avtalt tid. Om du er forsinket, gi beskjed så raskt som mulig. Viss samme person er forsinket to eller flere ganger, skal dette noterast og gis grunn for i rapport.
2. Begge deler ansvaret likt for at utviklingsprosessen og rapporten er av tilfreds kvalitet for en høy karakter.

Tilfredshet

3. Vi ønsker at det er givende og gøy å jobbe med prosjektet. Det skal være en god atmosfære og om en av oss mener den andre er urettferdig eller negativ, skal dette bli tatt opp og diskutert.
4. Viss en av oss ikke har en god dag, eller er i dårlig humør, ønsker vi å ta det opp slik at samarbeidet blir tilpasset.
5. Vi tar opp og setter innleveringsfrister til hverandre underveis, slik at arbeidet ikke hoper seg opp mot slutten. Det er viktig at vi overholder disse fristene så godt som mulig.

Læring

6. Begge skal være åpne mot hverandre med å gi regelmessig konstruktiv kritikk for å gi oss størst mulig sjanse for ett vellykket prosjekt. Mottakeren av kritikken skal ta rådene seriøst og ikke ta dette personlig.
7. Vi skal utfordre oss selv med å ta deloppgaver som vi ikke har mestret enda, for å lære mer underveis.
8. Om noen er usikker eller sitter fast, skal det være enkelt å kontakte medarbeider for bistand.

Tony Paulsen

Petter Henriksen

Vedlegg 4

Møtereferat 13.01.22

Møte referat

Møte mellom prosjekt -og styringsgruppen, bachelor ROV		
Varighet: 60 min	Dato: 13.01.22	Start tidspunkt: 13:00

Møte lokasjon:	Zoom
Møte innkalt av:	Tony Paulsen
Møte type:	Fremdriftsmøte med styringsgruppe
Møtet styrt av:	Tony Paulsen
Sekretær:	Petter Henriksen
Tids ansvarlig:	Petter Henriksen
Deltakere:	Tony Paulsen, Petter Henriksen, Ottar L. Osen, Lars Christian Gansel

Agenda nummer	Agenda	Diskutert
1	Lars presenterer hva han ønsker seg	Ble informert at vi kunne tolke oppgaven veldig åpent. Interesse for sonar og ulike kombinasjonssensorer. Interesse for ett nytt kamera som er bedre egnet for lokasjon av fisk i vanskelige lysforhold. Viste eksempel på sonar og kombinasjonssensor.
2	Ottar sier litt om hva vi ønsker å prioritere	Ga tips om hvordan vi kan utvikle prototypen for ROV på en måte som er tilstrekkelig for vår testing. Informerte prosjektgruppe om at det må undersøkes hvilke sensorer som skal bestilles raskt som mulig.
3	Studentene reflekterer over informasjon	Uttrykte at vi ville fokusere på programmering og kryss implementasjon av sensorene på en måte som gir en tydelig forbedring av tidligere ROVer.

Appendix B

Progress reports

IELEA2920 Hovedprosjekt	Project AIP-ROV	Number of meeting this period 1). 0 planned	Firma - Oppdragsgiver NTNU i Aalesund	Side 1 av 2
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 50	Prosjektgruppe (navn) ROV	Dato 08.03.22

Main goal/purpose for this periods work

- Test all sensors (temperature, depth, conductance, sonar etc)
- Test all actuators (lights, thrusters)
- Make ROV-platform

Planned activities this period

- Write test scripts in Arduino IDE for I2C sensors, integrating all I2C sensors on same bus
- Setting up Raspberry PI with Python scripts for testing serial communication sensor
- Make mounts for sensors
- Mount Camera
- Mount Sensors to ROV

Actually conducted activites this period

- Finished testing all I2C sensors that will be connected to Arduino, with single scripts reading all devices on I2C bus
- Tested all actuators with test scripts in Arduino
- Set up Raspberry PI, but did not compete testing script for communicating over serial bus with conductance sensor
- Made ROV-platform and mounted lights(2), thrusters(3), Aanderaa conductivity sensor and Ping 360 Sonar

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

- Completed most of the testing tasks
- Serial communication with conductance sensor and sonar was not completed as planned, this has resulted from lack of time used on task. It is not a result of a single troubleshooting problem. Reduced time spent on project was because of sickness during some days, so time was directed towards other subjects.
- Camera wasn't mounted due to it not having arrived yet.

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

- Serial communication has to be fixed early during next two week period. It should not be a difficult task as there are many example scripts and well documented sources for this.

Main experience from this period

- The testing of sensors and actuators was relatively easy, as the progress was steady and there were only small fixable problems that was easy to solve. Raspberry PI was very simple to set up initially. However due to not enough spent on task not everything planned was completed.
- Learning how to use Fusion 360 to make 3D models and use the schools 1200W laser cutter to make parts was relatively easy to get the hang of and it.

Main purpose/focus next period

- Creating programs that are adjusted and unique for the system functions that we want to implement for the ROV.
- Finding solution and completing communication from the ROV to surface through thether cable.

- Mount and use camera

Planned activities next period

- Complete the now postponed activities that is creating testing scripts in Python for serial communication with sonar and conductance sensors.
- Create system programs for motor controls, lights, temperature, sonar, conductance.
- Test communication and see if the use of an Arduino can be bypassed

Other

Wish/need for counceling

- Discuss order of cable(s) that are watertight for the conductance, turbidity and oxygen sensors.

Approval/signature group leader	Signature other group participants
Tony Paulsen Tony Paulsen	Petter Henriksen Petter Henriksen

IELEA2920 Hovedprosjekt	Project AIP-ROV	Number of meeting this period 1). 0 planned	Firma - Oppdragsgiver NTNU i Aalesund	Side 1 av 2
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 70	Prosjektgruppe (navn)	Dato 22.03.22

Main goal/purpose for this periods work

- Creating programs that are adjusted and unique for the system functions that we want to implement for the ROV.
- Finding solutions and completing communication from the ROV to the surface through thether cable.

Planned activities this period

- Complete the now postponed activities that is creating scripts in Python for serial communication with sonar and conductance sensors.
- Create system programs for motor controls, lights, temperature, sonar, conductance.
- Test communication and see if the use of an Arduino can be bypassed.

Actually conducted activites this period

- Completed most of the sonar program for system, implemented functions that can change the scanning range (different modes for collision avoidance and general inspection). Collision avoidance algorithm needs resetting of interlocked zones logic to be completed.
- Sonar communication has not been initialized, and program has not been started.
- The removal of Arduino idea was scraped, as it was seen as a benefit to have it there and communication seems to work great using serial communication

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

- Sonar program progress was steady throughout the period, final completion of the program has not been completed due to testing various solutions for resetting of interlocked zones.
- Conductance sensor serial communication was worked on, but after researching connection diagrams, it was discovered that we did not have the appropriate testing cable for RS232 serial communication.
- Camera has not progressed much due to delivery issues, we have acquired a temporary substitute but we decided to focus on the problems that were guaranteed to be on the final product.

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

- Sonar program has a tendency to increase in complexity due to when researching libraries and functionality, new ideas of how we want the sonar to perform is gained. Important to keep in mind that we need all aspects of project to work, and from that point maybe add additional functionality.
- When we get required parts for conductance sensor, we have to focus on getting that part of the project finished. New cable was ordered. Plugs for the sensor has been found and is ready to be ordered.
- When looking at previous solutions we found that one group had used a product from BlueRobotics that allowed them to use 3 out 4 pairs on the tether for power and only 1 for communication while still maintaining usable data speeds. Whether or not this will be used needs to be discussed with project supervisors.

Main experience from this period

- Sonar functionality and improvement can always be improved, important to get basic

IELEA2920 Hovedprosjekt	Project AIP-ROV	Number of meeting this period 1). 0 planned	Firma - Oppdragsgiver NTNU i Aalesund	Side 2 av 2
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 70	Prosjektgruppe (navn)	Dato 22.03.22

functionality working, and focusing on fully completing all aspects of project first.
- Should have checked that we had all needed parts for initializing communicating with all sensors earlier in the project, to avoid having to wait for parts.

Main purpose/focus next period

- Complete communication from Raspberry Pi and conductance sensor.
- Complete conductance sensor programs and functionality in Raspberry Pi.
- Research and implement format for sending data over serial.
- Research and implement format for sending data from Raspberry Pi and surface GUI.
- Try to get camera up in Python

Planned activities next period

- When conductance sensor final parts arrive, finish communication and programs in Raspberry PI that handles the data.
- Find format of transmitting data between components that use serial (Raspberry PI and Arduino UNO), and format for transmitting data from Raspberry PI and surface GUI. Should be formatted in a way that is easy to expand and logical.
- Try to get substitute camera to work
- Decide and order parts for power delivery (step down converter)

Other

Wish/need for counceling

- Power delivery and comunication
- Camera solution

Approval/signature group leader	Signature other group participants
Tony Paulsen Tony Paulsen	Petter Henriksen Petter Henriksen

IELEA2920 Bachelor Project	Project A Flexible and Common Control Architecture for Rolls-Royce Marine Cranes and Robotic Arms	Number of meeting this period 1). 0 planned	Firma - Oppdragsgiver NTNU i Ålesund	Side 1 av 2
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 70	Prosjektgruppe (navn) AIP-ROV	Dato 07.04.22

Main goal/purpose for this periods work

- Complete communication from Raspberry Pi and conductance sensor.
- Complete conductance sensor programs and functionality in Raspberry Pi.
- Research and implement format for sending data over serial.
- Research and implement format for sending data from Raspberry Pi and surface GUI.
- Try to get camera up in Python

Planned activities this period

- When conductance sensor final parts arrive, finish communication and programs in Raspberry PI that handles the data.
- Find format of transmitting data between components that use serial (Raspberry PI and Arduino UNO), and format for transmitting data from Raspberry PI and surface GUI. should be formatted in a way that is easy to expand and logical.
- Try to get substitute camera to work
- Decide and order parts for power delivery (step down converter)

Actually conducted activites this period

- Finished researching and completed serial communication using UART for transmission between Arduino Uno and Raspberry Pi.
- Ordered communication solution for tether communication (Raspberry Pi to PC GUI). Decided communication protocol and borrowed similar components from a similar project to test our solution. Found easy and efficient libraries to use for serializing data and initializing communication.
- Added extra functionality for the sonar collision avoidance programs, added extra control options (inspection-mode and collision-avoidance-mode).
- Got Camera to work with OpenCV in Python
- Started work on GUI using Python and the library PyQt5
- Ordered parts for power delivery.

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

- Conductance sensor plug arrived last day of this working period, work on this part of the project has not been performed. However some research in data-sheets to prepare for easy integration.

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

- A substantial part of the work planned for this period revolved around the conductance sensor integration, as this was not possible, those work hours were redirected towards working on thether communication.

Main experience from this period

- Learned about advantages and disadvantages with threading and multiprocessing in Python.
- Increased knowledge about advantages and disadvantages with TCP and UDP transmission protocols.

- Increased knowledge about serialization of data techniques (JSON, Pickle) in Python. And how to properly set up communication between different programming languages efficiently.
- Learned how to make a GUI in Python and learned about object orientated programming

Main purpose/focus next period

- Finishing the GUI so that it is ready for implementation of communication
- Conductance sensor integration
- Make the ROV ready for testing in water.

Planned activities next period

- Finish GUI functions and visuals
- Integrate conductance sensor with the system.
- Finish making mounts for hardware inside the ROV.
- Determine if power delivery solution is sufficient.
- Wire up all components inside the ROV and start full scale-testing.
- Finishing most of theoretical basis/ methods and results in report.
-

Other

Wish/need for counceling

- Increasing communication bandwidth measures recommendations (multi- threading and processing).
- Discussing current achieved framerate of camera.

Approval/signature group leader	Signature other group participants
Tony Paulsen Tony Paulsen	Petter Henriksen Petter Henriksen

IELEA2920 Bachelor Project	Project AIP-ROV	Number of meeting this period 1). 0 planned	Firma - Oppdragsgiver NTNU i Ålesind	Side 1 av 2
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 70	Projectgroup (name) Tony Paulsen Petter Henriksen	Dato 22.04.22

Main goal/purpose for this periods work - Finishing the GUI so that it is ready for implementation of communication - Conductance sensor integration - Make the ROV ready for testing in water.
Planned activities this period - Finish GUI functions and visuals - Integrate conductance sensor with the system - Finish making mounts for hardware inside the ROV - Determine if power delivery solution is sufficient - Wire up all components inside the ROV and start full scale-testing - Finishing most of theoretical basis/ methods and results in report
Actually conducted activites this period - GUI was completed (sonar plots properly and all relevant data is displayed) - Conductance sensor has been integrated with RPi using serial communication and reads and samples data as desired - The mounts for components are done - Most of the components are wired up but not all
Description of/ justification for potential deviation between planned and real activities – Waiting for components has prevented the wiring of all components from being completed but most of it is done – Testing of power delivery has also not been done due to waiting for components
Description of/ justification for changes that is desired in the projects content or in the further plan of action – or progress report
Main experience from this period - Learned more about GUI implementation using PyQt5 - Learned how to integrate conductance sensor, writing commands and reading parameters - Learned about implementing multiple communication protocols simultaneously (UDP and TCP) for different types of data - The enclosure is tight so working with that requires a lot of space and cable -management
Main purpose/focus next period - Testing of ROV - Work on report - Tune system
Planned activities next period - Finish the unfinished task from this period - Complete multiple tests - Find weaknesses from tests and improve solutions - Finish all main parts of report
Other

IELEA2920 Bachelor Project Progress report	Project AIP-ROV	Number of meeting this period 1). 0 planned	Firma - Oppdragsgiver NTNU i Ålesind	Side 2 av 2
	Period/week(s) 2	Number of hours this period. (from log) Approx. 70	Projectgroup (name) Tony Paulsen Petter Henriksen	Dato 22.04.22

Wish/need for counceling

- Placement of Voltage Regulators and DC/DC Converter
- Setup of the power system
- Supplying voltage during tests

Approval/signature group leader	Signature other group participants
Tony Paulsen Tony Paulsen	Petter Henriksen Petter Henriksen

ID301702 Hovedprosjekt	Project A Flexible and Common Control Architecture for Rolls-Royce Marine Cranes and Robotic Arms	Number of meeting this period 1). 0 planned	Firma - Oppdragsgiver NTNU Aalesund	Side 1 av 2
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 70	Prosjektgruppe (navn)	Dato 06.05.22

Main goal/purpose for this periods work

- Testing of ROV
- Work on report
- Tune system

Planned activities this period

- Finish the unfinished tasks from this period
- Complete multiple tests
- Find weaknesses from tests and improve solutions
- Finish all main parts of report

Actually conducted activites this period

- Finished most of the methodology in the report
- Only integration tests over water was completed
- Some changes in software has been completed to improve full system functionality

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

- During the integration test the group found some issues and bugs that had to be sorted out so the focus went towards fixing these issues but that meant that we couldn't do as many test as hoped. One of the biggest issues had to do with the battery solution which did not work as intended and a lot of changes had to be made. More testing is required to verify if the fixes worked.

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

Main experience from this period

- Testing took more time than expected and the results can cause delays

Main purpose/focus next period

- Get a water test completed in the coming weekend
- Finishing report

Planned activities next period

- Finish test to get good data to describe in report
- Creating a *directions of use* for starting and using the ROV
- Focusing on discussion in report

Other

Wish/need for counceling

- General progress
- Priorities in final weeks
- Battery system fixes

Approval/signature group leader	Signature other group participants
Tony Paulsen	Petter Henriksen

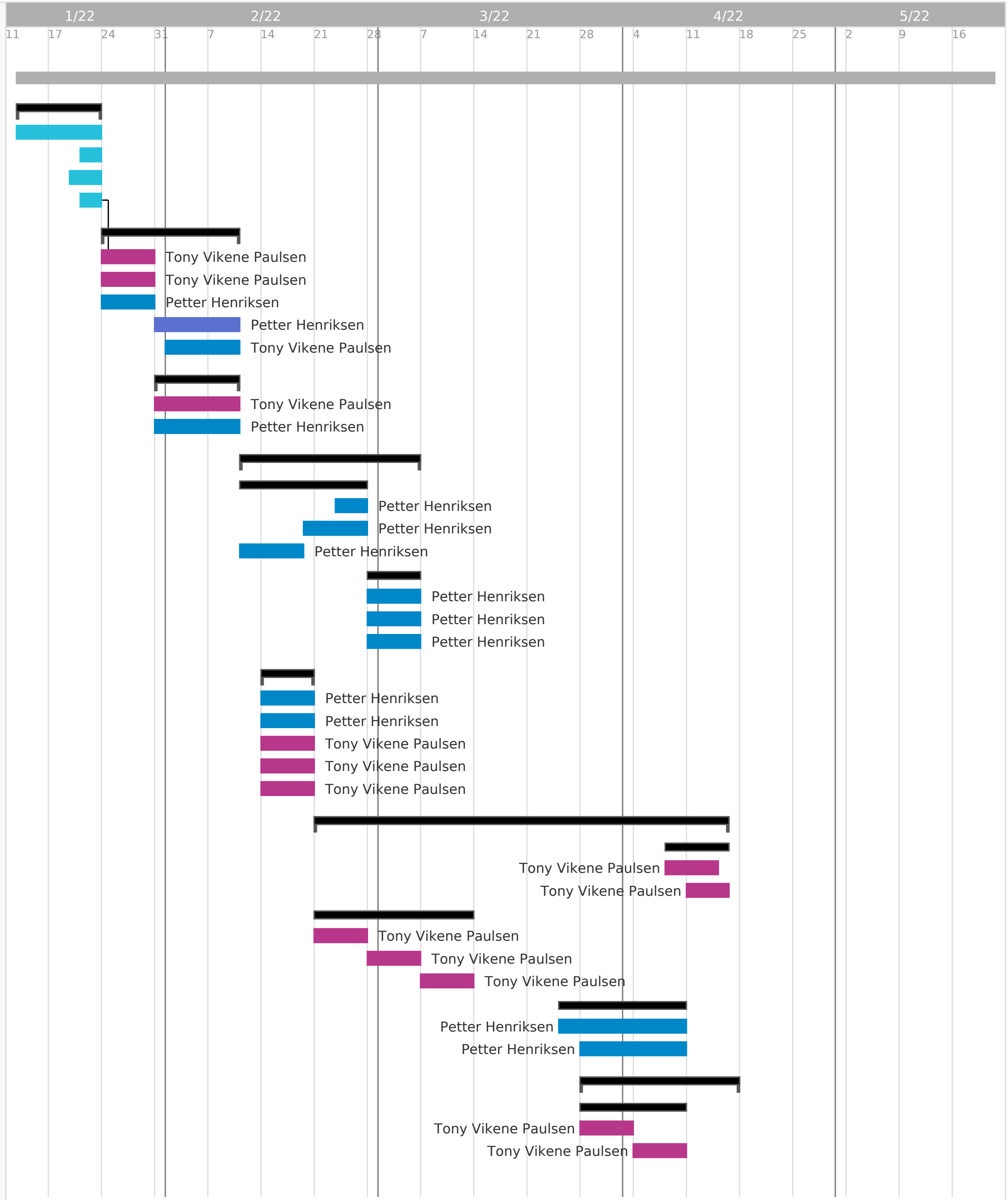
ID301702 Hovedprosjekt	Project A Flexible and Common Control Architecture for Rolls-Royce Marine Cranes and Robotic Arms	Number of meeting this period 1). 0 planned	Firma - Oppdragsgiver NTNU Aalesund	Side 2 av 2
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 70	Prosjektgruppe (navn)	Dato 06.05.22

Tony Paulsen	Petter Henriksen

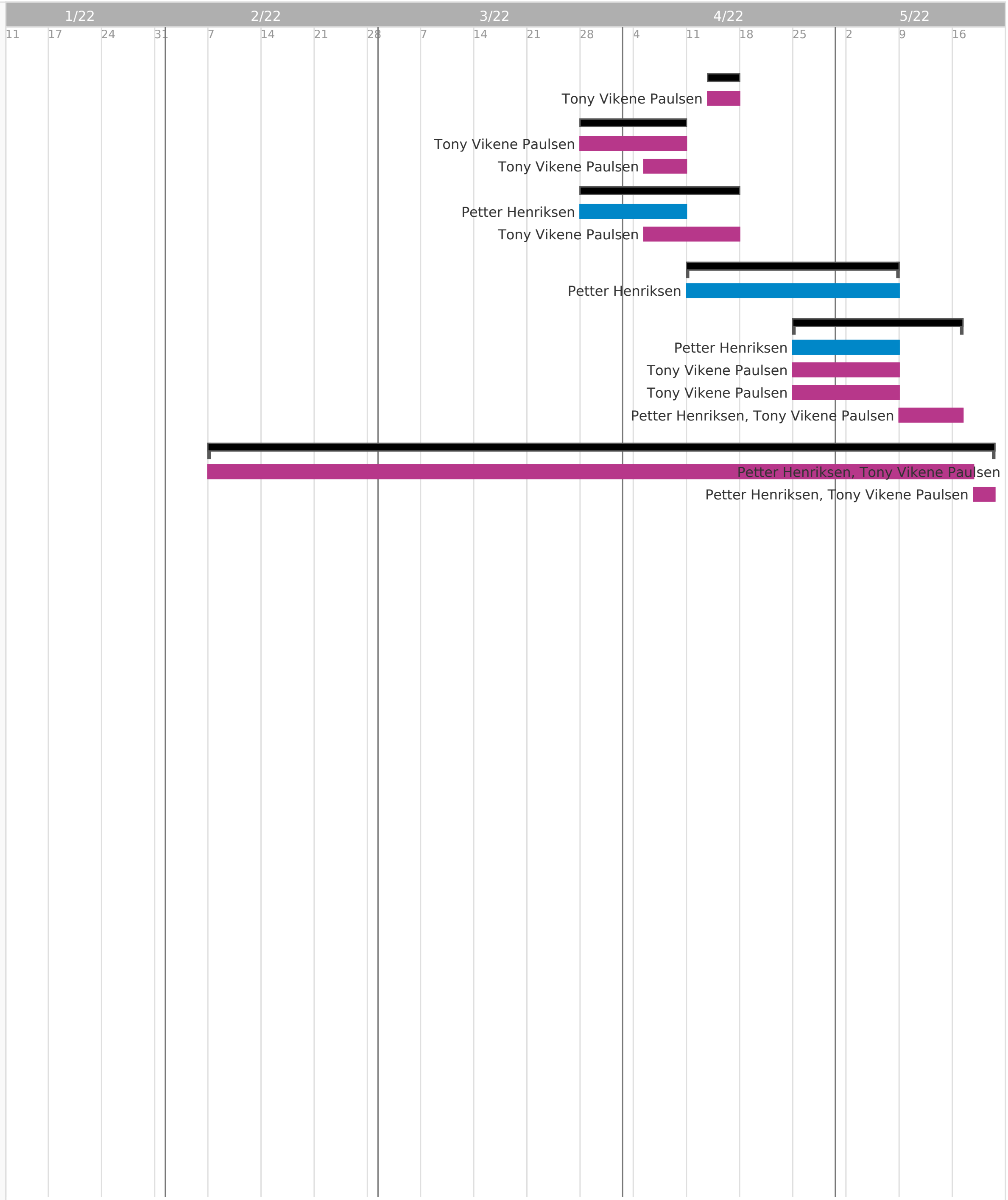
Appendix C

Gantt diagram

IELEA2920_ROV	start	end	0h	100%
Forprosjektrapport	**01/12/22**	**01/21/22**	**0h**	**100%**
Definere problemstillinger	01/12	01/21	0	100%
Lage risikomatrise	01/20	01/21	0	100%
Lage Gantt-diagram	01/19	01/21	0	100%
Kombinere alle dokument	01/20	01/21	0	100%
Utredning av ROV konsept	**01/24/22**	**02/09/22**	**0h**	**100%**
Bestemme sonar type	01/24	01/28	0	100%
Bestemme komb. sensor type	01/24	01/28	0	100%
Bestemme kamera	01/24	01/28	0	100%
Bestemme utforming av prototype sk...	01/31	02/09	0	100%
Vurdere plassering av sensorer	02/01	02/09	0	100%
Utstyrsanskaffelse	**01/31/22**	**02/09/22**	**0h**	**100%**
Bestille sensorer	01/31	02/09	0	100%
Bestille annet utstyr	01/31	02/09	0	100%
Bygge prototype	**02/10/22**	**03/04/22**	**0h**	**100%**
Tilpasse prototype for sensorer	**02/10/22**	**02/25/22**	**0h**	**100%**
Lage montasje for sonar	02/23	02/25	0	100%
Lage montasje for komb. sensor	02/18	02/25	0	100%
Lage montasje for kamera m/ vann...	02/10	02/17	0	100%
Montere nye sensorer	**02/28/22**	**03/04/22**	**0h**	**100%**
Montere sonar	02/28	03/04	0	100%
Montere komb. sensor	02/28	03/04	0	100%
Montere kamera	02/28	03/04	0	100%
Funksjonstest utstyr	**02/14/22**	**02/18/22**	**0h**	**100%**
Teste thrustere	02/14	02/18	0	100%
Teste lys	02/14	02/18	0	100%
Teste temperatur sensor	02/14	02/18	0	100%
Teste trykk sensor	02/14	02/18	0	100%
Teste fuktighets sensor	02/14	02/18	0	100%
Programvare utvikling	**02/21/22**	**04/14/22**	**0h**	**100%**
Kombinasjons sensor	**04/07/22**	**04/14/22**	**0h**	**100%**
Program for avlesning data	04/07	04/13	0	100%
Program til system	04/11	04/14	0	100%
Sonar	**02/21/22**	**03/11/22**	**0h**	**100%**
Program direkte til PC	02/21	02/25	0	100%
Program for avlesning data	02/28	03/04	0	100%
Program til system	03/07	03/11	0	100%
Kamera	**03/24/22**	**04/08/22**	**0h**	**100%**
Åpne Kamera i Python	03/24	04/08	0	100%
Åpne Kamera på Raspberry Pi	03/28	04/08	0	100%
Integrasjon av alle del-systemer	**03/28/22**	**04/15/22**	**0h**	**100%**
Kommunikasjon	**03/28/22**	**04/08/22**	**0h**	**100%**
Seriell (arduino-raspberry)	03/28	04/01	0	100%
TCP (raspberry-GUI)	04/04	04/08	0	100%



Integrasjon av komb. sensor	**04/13/22**	**04/15/22**	**0h**	**100%**
Vise sensor verdier	04/13	04/15	0	100%
Integrasjon av sonar data	**03/28/22**	**04/08/22**	**0h**	**100%**
Vise data som bilde	03/28	04/08	0	100%
Gi beskjed om kollisjonsfare	04/05	04/08	0	100%
Integrasjon av kamera	**03/28/22**	**04/15/22**	**0h**	**100%**
Vise kamera i GUI	03/28	04/08	0	100%
Vise bilde fra Raspberry	04/05	04/15	0	100%
Bygge prototype	**04/11/22**	**05/06/22**	**0h**	**100%**
Gjøre ROV klar for testing	04/11	05/06	0	100%
Testing av prototype	**04/25/22**	**05/16/22**	**0h**	**100%**
Testing av alle sensorer på GUI	04/25	05/06	0	100%
Testing av implementert logikk	04/25	05/06	0	100%
Tank test ved skulen	04/25	05/06	0	100%
Full skala test på lokasjon	05/09	05/16	0	100%
Fullføre rapport	**02/07/22**	**05/19/22**	**0h**	**100%**
Fullføre rapport i LaTeX	02/07	05/17	0	100%
Kombinere LaTeX m/ alle vedlegg	05/18	05/19	0	100%



Appendix D

Electrical drawings

Skoleversion

Electrical schematic - ROV 2022 - Bachelor

PCSCHEMATIC Automation

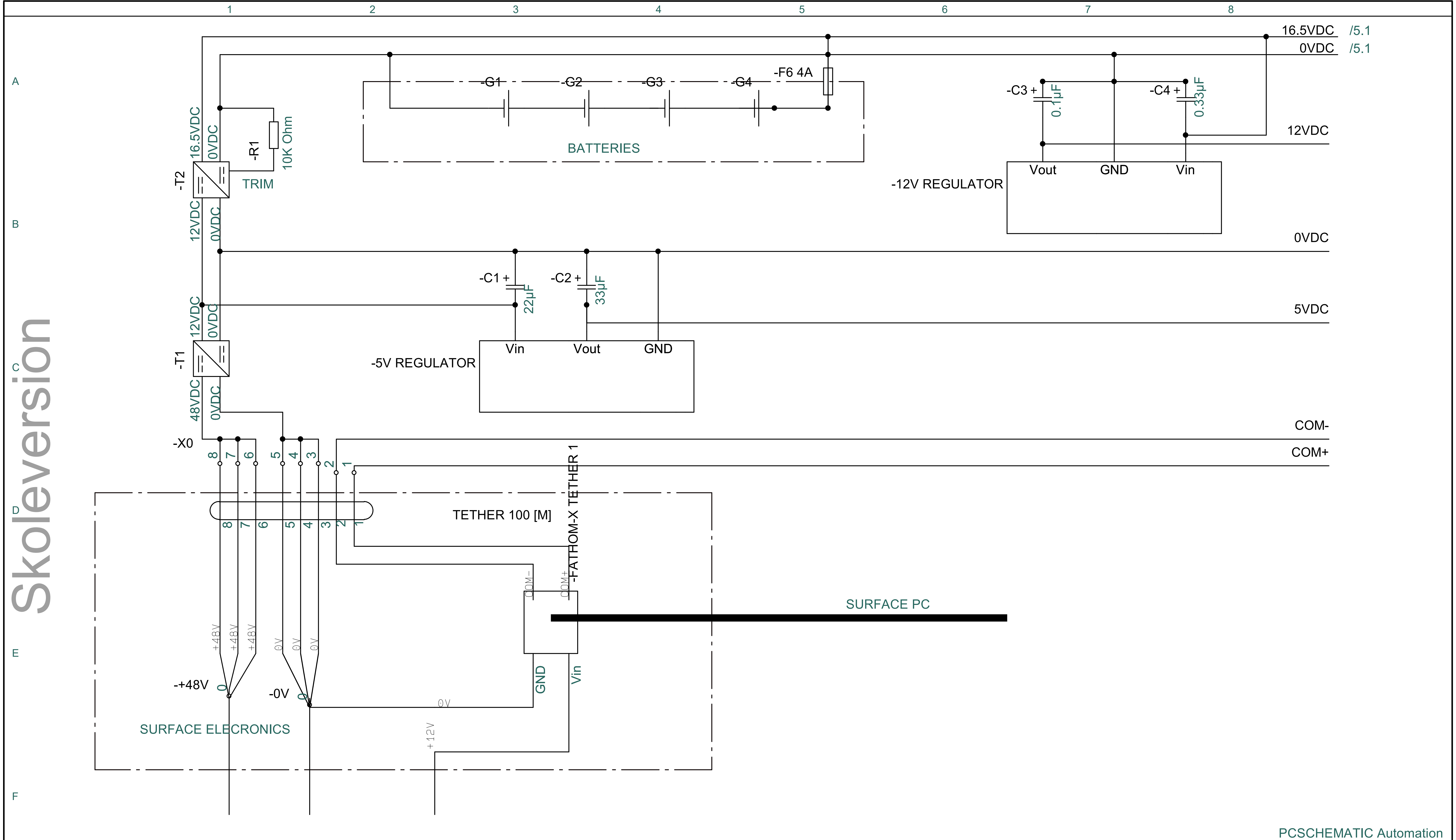
NTNU	**Project title:**	**Electrical schematic - ROV 2022 - Bachelor**	**Project no.:**		**Project rev.:**		**Page**	**1**
	Customer:	NTNU Ålesund	DCC:				Scale:	1:1
	Page title:	Front page	Dwg. no.:		Page rev.:		Previous page:	
	File name:	Prosjekt (1)	Eng. (proj/page):	Tony Paulsen	Last print:	19.05.2022	Next page:	2
	Page ref.:		Appr. (date/init):		Last edit:	19.05.2022	Total no. of pages:	9

Electrical schematic - ROV 2022 - Bachelor

Skoleversion

Skoleversion

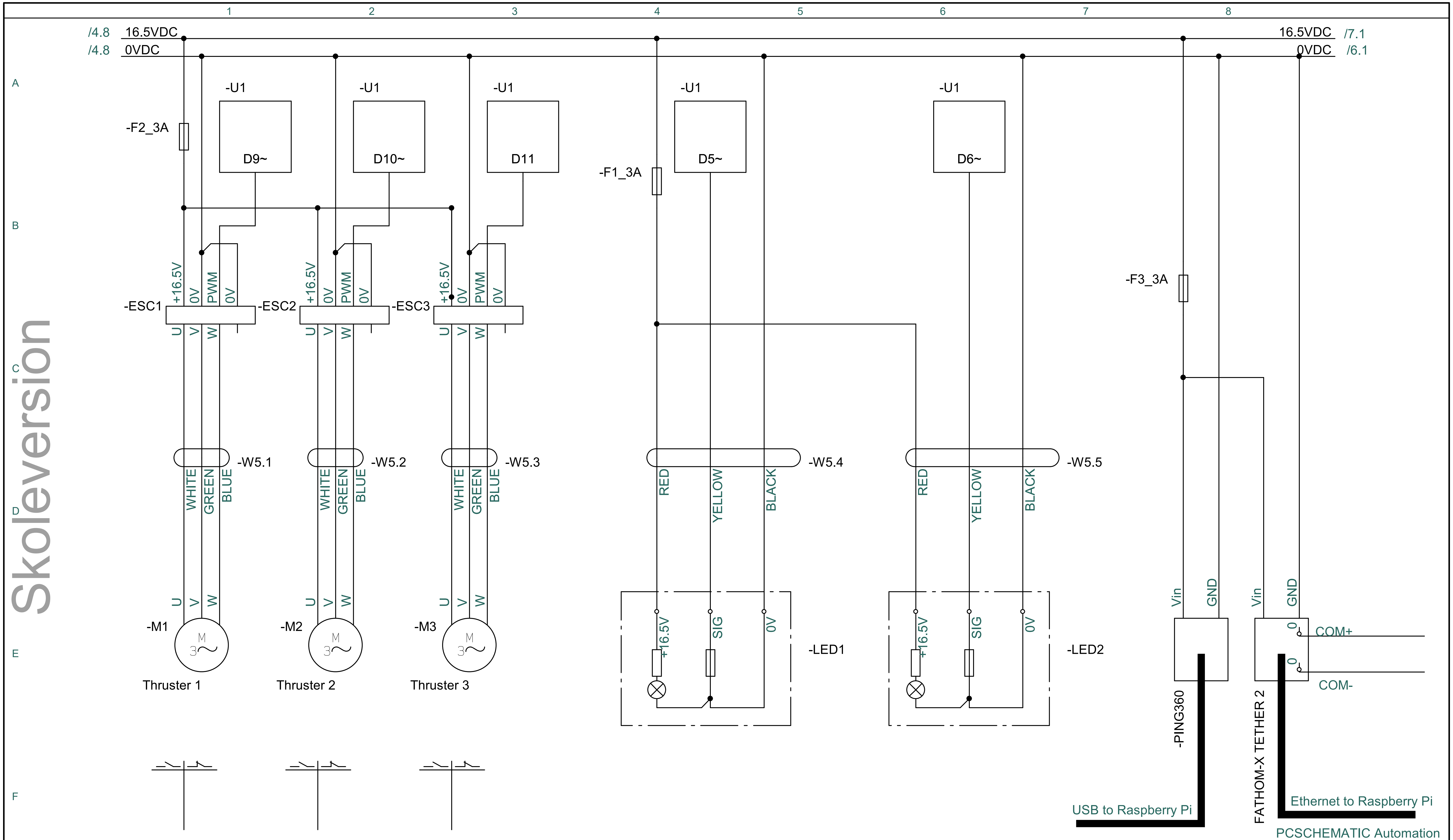
Diagrams



PCSCHEMATIC Automation

NTNU

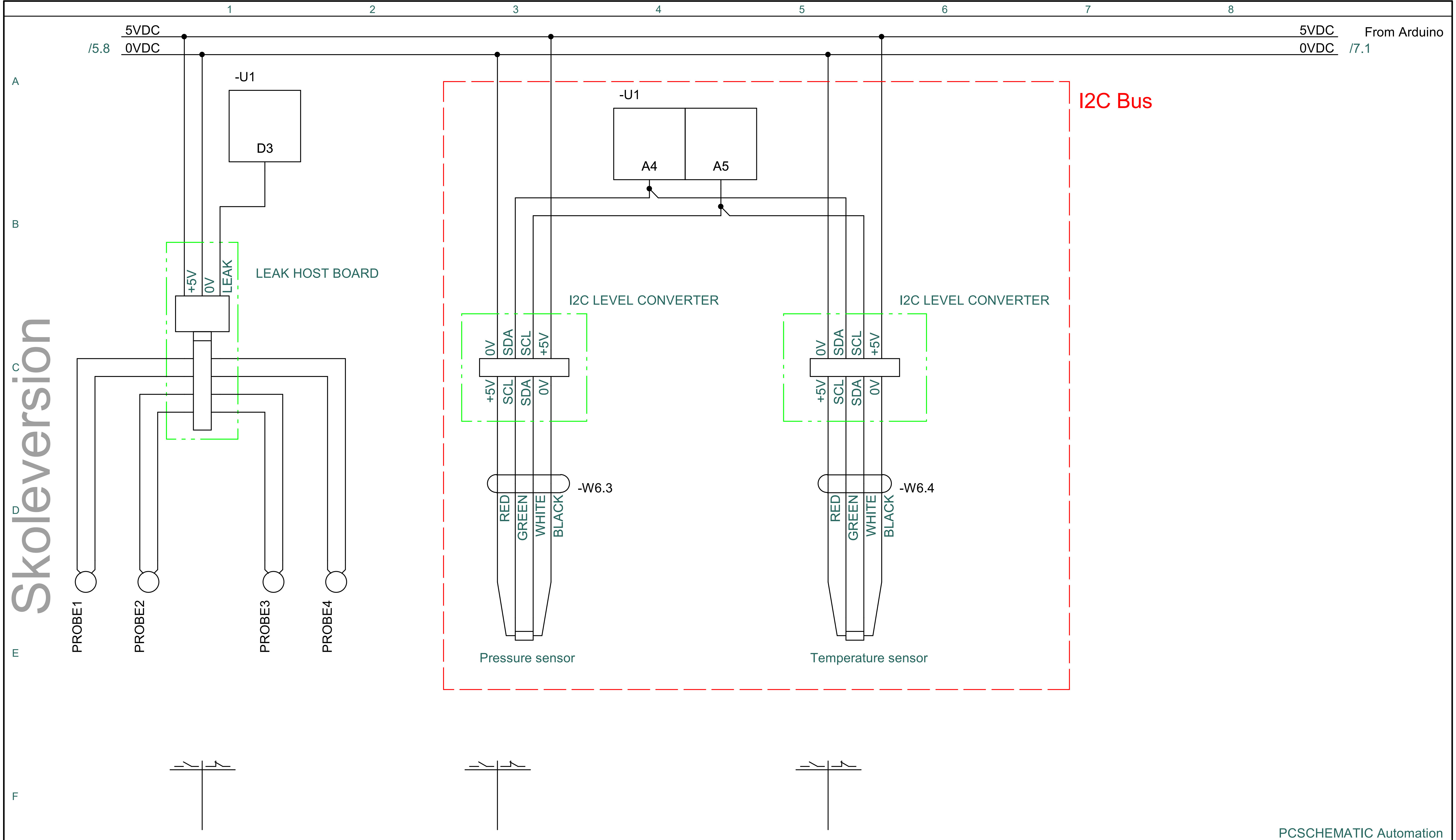
Project title:	**Electrical schematic - ROV 2022 - Bachelor**	**Project no.:**		**Project rev.:**		**Page**	4
Customer:	NTNU Ålesund	DCC:				Scale:	1:1
Page title:	Diagram	Dwg. no.:		Page rev.:		Previous page:	3
File name:	Prosjekt (1)	Eng. (proj/page):	Tony Paulsen	Last print:	19.05.2022	Next page:	5
Page ref.:		Appr. (date/init):		Last edit:	19.05.2022	Total no. of pages:	9



PCSCHEMATIC Automation

NTNU

Project title:	**Electrical schematic - ROV 2022 - Bachelor**	**Project no.:**		**Project rev.:**		**Page**	5
Customer:	NTNU Ålesund	DCC:				Scale:	1:1
Page title:	Diagram	Dwg. no.:		Page rev.:		Previous page:	4
File name:	Prosjekt (1)	Eng. (proj/page):	Tony Paulsen	Last print:	19.05.2022	Next page:	6
Page ref.:		Appr. (date/init):		Last edit:	19.05.2022	Total no. of pages:	9



PCSCHEMATIC Automation

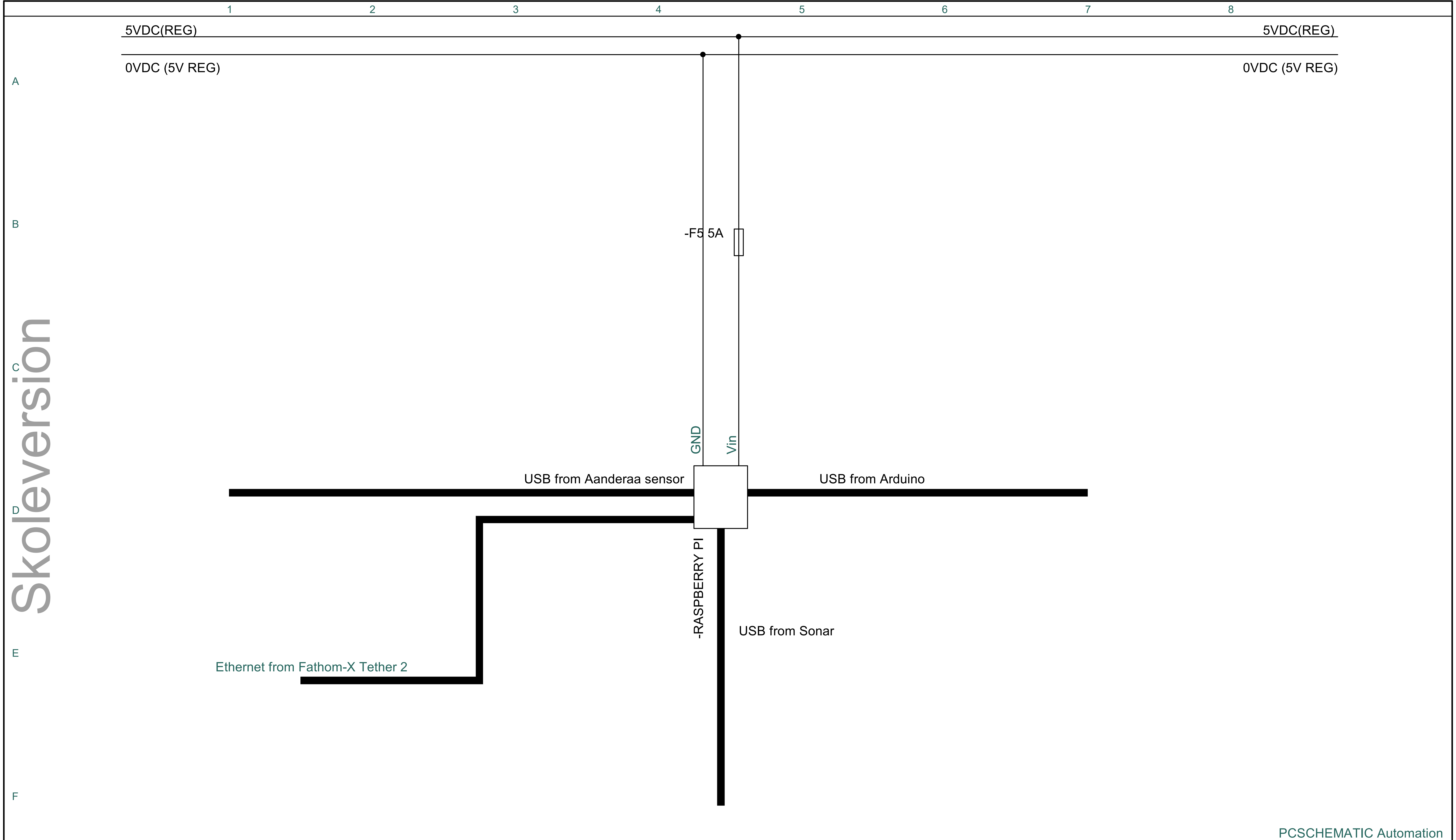
NTNU

Project title:	**Electrical schematic - ROV 2022 - Bachelor**	**Project no.:**		**Project rev.:**		**Page**	**6**
Customer:	NTNU Ålesund	DCC:				Scale:	1:1
Page title:	Diagram	Dwg. no.:		Page rev.:		Previous page:	5
File name:	Prosjekt (1)	Eng. (proj/page):	Tony Paulsen	Last print:	19.05.2022	Next page:	7
Page ref.:		Appr. (date/init):		Last edit:	18.05.2022	Total no. of pages:	9

PCSCHEMATIC Automation

NTNU

Project title:	**Electrical schematic - ROV 2022 - Bachelor**	**Project no.:**		**Project rev.:**		**Page**	**7**
Customer:	NTNU Ålesund	DCC:				Scale:	1:1
Page title:	Control circuit diagram	Dwg. no.:		Page rev.:		Previous page:	6
File name:	Prosjekt (1)	Eng. (proj/page):	Tony Paulsen	Last print:	19.05.2022	Next page:	8
Page ref.:		Appr. (date/init):		Last edit:	19.05.2022	Total no. of pages:	9



PCSCHEMATIC Automation

NTNU

Project title:	**Electrical schematic - ROV 2022 - Bachelor**	**Project no.:**		**Project rev.:**		**Page**	**8**
Customer:	NTNU Ålesund	DCC:				Scale:	1:1
Page title:	Control circuit diagram	Dwg. no.:		Page rev.:		Previous page:	7
File name:	Prosjekt (1)	Eng. (proj/page):	Tony Paulsen	Last print:	19.05.2022	Next page:	
Page ref.:		Appr. (date/init):		Last edit:	19.05.2022	Total no. of pages:	9

Appendix E

User Manual

AIP-ROP User Manual

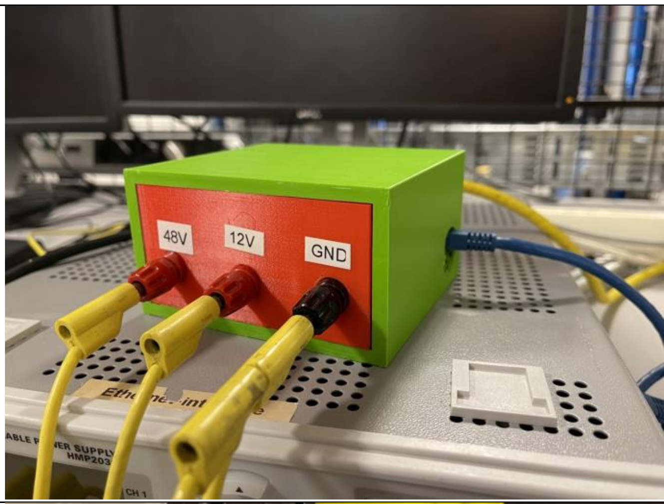
Describes the preparations and procedures needed to use the ROV under defined environment conditions. Additionally, shows how to connect and start the programs to be able to control the ROV using the GUI.

<table>
<tr><td>

Step 1:

Connect 48V, 12V and 0V to terminals as shown in picture. Both positive supplies has to have common ground, otherwise communication might not work as intended.
Connect an ethernet cable with RJ45 plug into box with the other end to the computer running the GUI.

</td><td>



</td></tr>
<tr><td>

Step 2:

Connect the tether cable coming from the communication interface box shown in previous step to the tether spool to complete the circuit to the ROV.
If power is activated for interface box the ROV should be energized. Arduino Uno lights should blink, and electronic speed controllers should make initialization sounds.

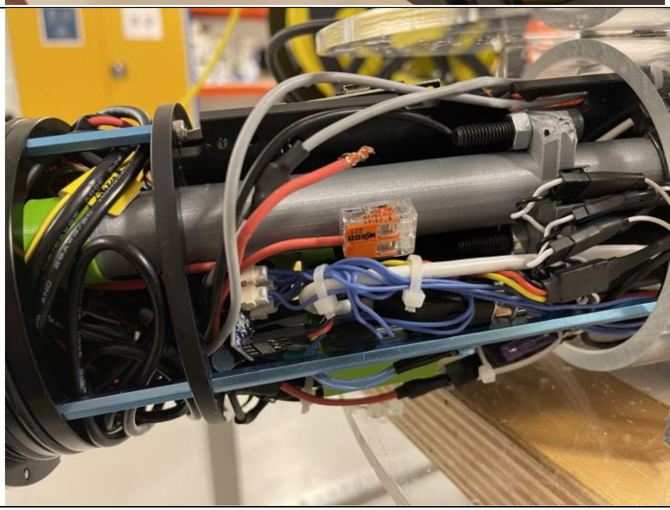
</td><td>

</td></tr>
<tr><td>

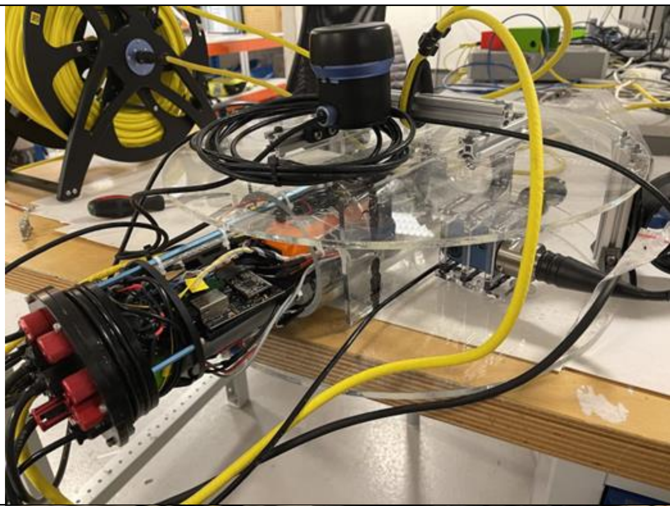
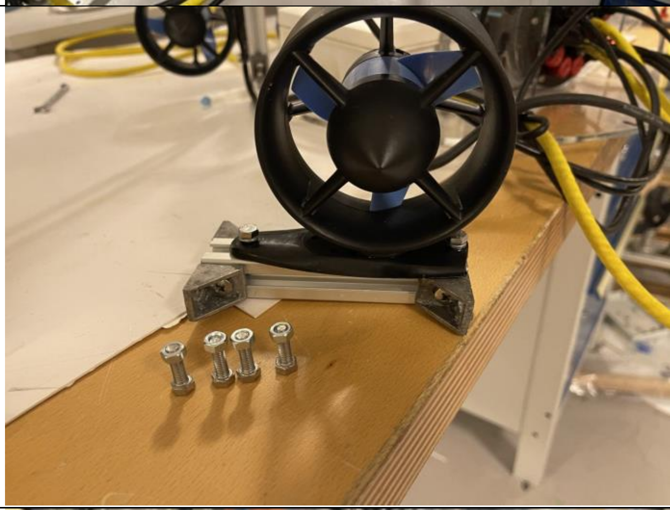
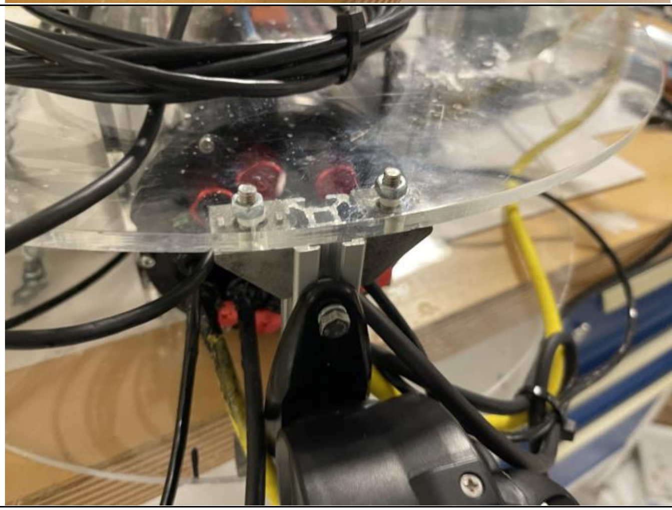
Step 3:

Inside the ROV enclosure, the local battery pack must be connected to the circuit. This is done by connecting the red wire to the Wago quick connector.

</td><td>

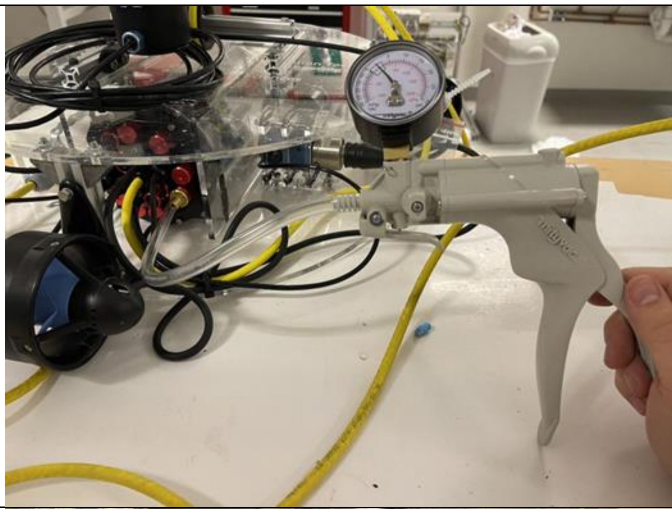


</td></tr>
</table>

Step 4: Slide the electronics tray into the acrylic cylinder. Be careful for wires that may get stuck and unplugged. The tray is slid all the way in to seal properly.	
Step 5: Mount the aft thruster to the acrylic ROV foundation using four M5 bolts as shown in figure.	
Checkpoint After the above instructions the ROV should be mounted as seen in the figure. It is important that the thruster is mounted in the orientation as shown in the figure.	

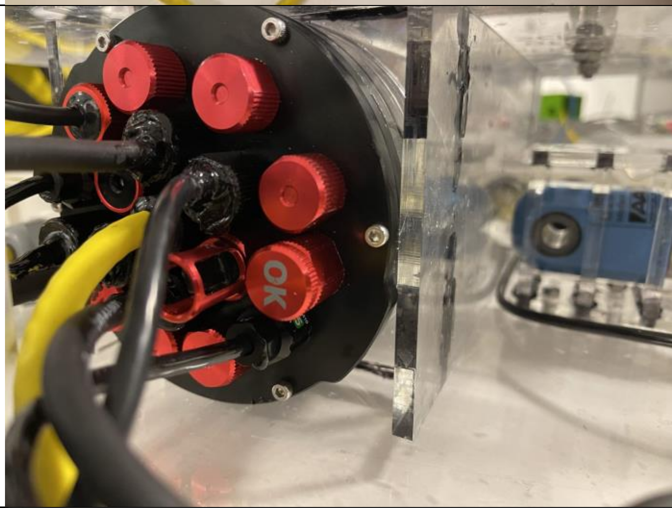
Step 6:

Attach the vacuum pump hose to the vent plug at the backplate of the ROV.
Pump until the pressure on the manometer shows 15 inHg.
Wait for 15 minutes, and check if the pressure is still above 14.5 inHg.
If it is not return to step 4 and check for broken seals or other leak indicators.
If pressure is above the limit defined, the ROV is ready to be used in water.



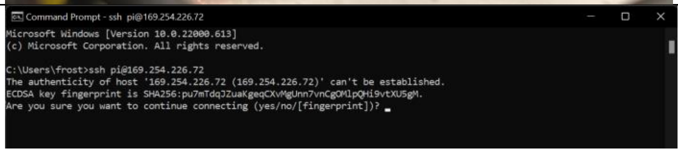
Step 8:

Remove the hose plug from the vacuum pump, and screw in the plug indicated with <OK>.
It is extra important that this plug is properly greased and tightened, as it was not tested in the previous vacuum test.



Step 9:

Next open a terminal on the computer that will run the GUI.
Set the ethernet adapter on the computer to: 169.254.226.73
Connect to the ROV using SSH and command:
>>ssh pi@169.254.226.72

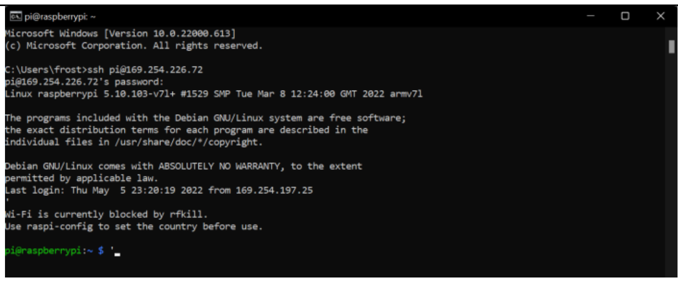


Step 10:

If tether and power is properly connected, the terminal will prompt for the RPi password.
Enter password:
>>rov2022

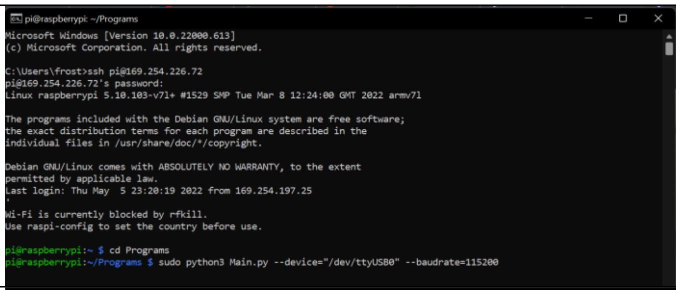
Step 11:

You know have access to the RPi computer. Check for previously saved photos and videos and transfer them to external device. Media is located at relative path from login within
pi/Programs/Photos
pi/Programs/Videos



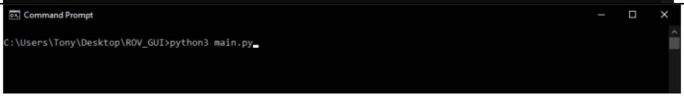
Step 12:

For running the GUI, navigate into Programs folder.
To run the Python ROV program use command as seen in figure.



Step 13:

When the RPi program has been started, the terminal will display a message indicating that the operator has to launch GUI script.
Navigate to where the Python GUI scripts are located, and run the files with command as shown in figure.



Step 14:

You have now launched all systems for the ROV and should see the sensor values update on the GUI regularly.
Control of the motors are done with the arrows, and brightness of lights can be adjusted with slider.

Appendix F

Arduino code

```
/***********************************************************************
 * PROGRAM THAT CONTROLS AN ARDUINO FOR A REMOTE OPERATING VEHICLE (ROV).
 * ROV IS DESIGNED FOR AQUACULTURE INSPECTION. THE PROGRAM CONTROLS
 * MOVEMENT WITH THREE THRUSTERS AND VISION WITH TWO SUBSEA LIGHTS.
 * ADDITIONALLY THE PROGRAM READS TEMPERATURE AND PRESSURE FROM THE
 * ENVIRONMENT THROUGH I2C COMMUNICATION. THE ARDUINO COMMUNICATES
 * WITH RASPBERRY PI THROUGH SERIAL COMMUNICATION.
 **********************************************************************/

#include <Arduino.h>
#include <Servo.h>
#include <Wire.h>
#include "MS5837.h"
#include "TSYS01.h"
#include <ArduinoJson.h>
#include "Communication.h"
#include "Functions.h"


// Initialize I2C OBJECTS
MS5837 pressSensor;
TSYS01 tempSensor;

// Initialize actuator objects for motors and lights
Servo motor_1;
Servo motor_2;
Servo motor_3;
Servo starboard_light;
Servo port_light;

// Initializes run commands from the globals
int runZone = -1;   // Initial run state is set to offf
bool z1lock;
bool z2lock;
bool z3lock;
bool z4lock;
bool z5lock;
bool z6lock;
bool z7lock;
bool z8lock;

// Input and output pins are selected based on PWM capabilities
byte pinM1 = 9;
byte pinM2 = 10;
byte pinM3 = 11;
byte l1 = 5;
byte l2 = 6;

// Local variables used for logic and communication
int val;
int leakPin = 3;   // Leak Signal Pin //pin must be 3 not 1 or 2
int leak = 0;      // 0 = Dry , 1 = Leak
float temp;
float depth;
bool leakStatus;
int missedPackets;
int i;

// Json document used to hold and format data to be sent to RPi
```

```
StaticJsonDocument<48> outDoc;

void setup() {
  // Initialize serial communication with RPi
  Serial.begin(9600);

  // Initialize I2C bus sensors
  Wire.begin();
  pressSensor.init();
  tempSensor.init();

  // Declaring properties to pressure sensor object
  pressSensor.setModel(MS5837::MS5837_30BA);
  pressSensor.setFluidDensity(997);

  // Declaring PWM pins for motors and lights
  port_light.attach(l1);
  starboard_light.attach(l2);
  motor_1.attach(pinM1);
  motor_2.attach(pinM2);
  motor_3.attach(pinM3);

  // Declaring leak sensor as input
  pinMode(leakPin, INPUT);

  // Motors need to receive zero thrust signal for 7
  // seconds to properly initialize
  fullStop();
  setLights(0); // Lights are set to off for inital operation
  delay(7000);
}

void loop() {
  // Read I2C sensor values
  tempSensor.read();
  pressSensor.read();

  // Gets the required sensor values and place in variables
  temp = tempSensor.temperature();
  depth = pressSensor.depth();
  leakStatus = digitalRead(leakPin);

  // Every 30 program iteration the Arduino sends data to Raspberry
  if (i > 30) {
    sendToRaspberry(temp, depth, leakStatus);
    i = 0;
  }
  i++;

  // If data is found within serial buffer, handle data
  if (Serial.available()) {
    receiveFromRaspberry();
    missedPackets = 0;
  } else {  // Otherwise connection is broken and motors are stopped
    missedPackets++;
    if (missedPackets > 3) {
      fullStop();
    }
  }

```

```
  // Control motor speed and directions based on commanded zone
  // and information from collision avoidance system
  setMotorSpeeds(runZone, z1lock, z2lock, z3lock,
  z4lock, z5lock, z6lock, z7lock, z8lock);
}
```

```
/***********************************************************************
 * SUBPROGRAM OF THE ARDUINO UNO CODE FOR CONTROLLING ROV. FILE CONTAINS
 * THE FUNCTIONS NEEDED TO CONTROL THE MOTORS AND LIGHTS.
 **********************************************************************/
#include <Arduino.h>
#include "Functions.h"

/**
 * Function that controls the direction the ROV is moving in. Takes in
 * a zone that operator wants the ROV to be moving towards, and given that
 * zone is not prohibited from the interlocking system, the function sends out
 * the appropiate PWM signals to the motor drivers. In addition to linear
 * movements, the ROV can rotate both clockwise -and counterclockwise, these
 * directions are never prohibited by the interlocing system.
 * @param zone    Requsted propulsion in this zone direction
 * @param z1lock  Propulsion in zone 1 direction prohibited variable
 * @param z2lock  Propulsion in zone 2 direction prohibited variable
 * @param z3lock  Propulsion in zone 3 direction prohibited variable
 * @param z4lock  Propulsion in zone 4 direction prohibited variable
 * @param z5lock  Propulsion in zone 5 direction prohibited variable
 * @param z6lock  Propulsion in zone 6 direction prohibited variable
 * @param z7lock  Propulsion in zone 7 direction prohibited variable
 * @param z8lock  Propulsion in zone 8 direction prohibited variable
 */

void setMotorSpeeds(int zone, bool z1lock, bool z2lock, bool z3lock,
bool z4lock, bool z5lock, bool z6lock, bool z7lock, bool z8lock) {
  switch (zone) {
  case 0: // Forward
    if (z1lock) {
      fullStop();
    } else {
      controlMovement(1550, 1550, 1500);
    }
    break;
  case 1: // Forward-right
    if (z2lock) {
      fullStop();
    } else {
      controlMovement(1550, 1500, 1550);
    }
    break;
  case 2: // Right
    if (z3lock) {
      fullStop();
    } else {
      controlMovement(1530, 1470, 1550);
    }
    break;
  case 3: // Reverse-right
    if (z4lock) {
      fullStop();
    } else {
      controlMovement(1500, 1450, 1550);
    }
    break;
  case 4: // Reverse
    if (z5lock) {
      fullStop();
```

```cpp
    } else {
      controlMovement(1450, 1450, 1500);
    }
    break;
  case 5: // Reverse-left
    if (z6lock) {
      fullStop();
    } else {
      controlMovement(1450, 1500, 1450);
    }
    break;
  case 6: // Left
    if (z7lock) {
      fullStop();
    } else {
      controlMovement(1470, 1530, 1450);
    }
    break;
  case 7: // Forward-left
    if (z8lock) {
      fullStop();
    } else {
      controlMovement(1500, 1550, 1450);
    }
    break;
  case 8: // Counterclock-wise
    controlMovement(1470, 1530, 1530);
    break;
  case 9:  // Clock-wise
    controlMovement(1530, 1470, 1470);
    break;
  case -1:  // Stand still
    fullStop();
    break;
  }
}

/**
 * Sets subsea lights power output, both lights have the exact same value. Takes
 * in an integer that ranges from 0-255 and translates that to 0-100% light power.
 * Sets light with PWM setting.
 * @param pwr      Integer that ranges from 0-255 for 0-100% power
 */
void setLights(int pwr) {
  int val;
  val = map(pwr, 0, 255, 1100, 1500); // 1900 draws 2.5A current
  port_light.writeMicroseconds(val);
  starboard_light.writeMicroseconds(val);
}

/**
 * Function that sets all motors to neutral output. Zero propulsion in any
 * direction for all three motors.
 */
void fullStop() {
  motor_1.write(1500);
  motor_2.write(1500);
  motor_3.write(1500);
}

```

```
/**
 * Function that sets all the motor power and direction outputs. Takes in three
 * arguments with PWM settings that controls each respective motor.
 * @param m1pwr      PWM setting for motor 1
 * @param m2pwr      PWM setting for motor 2
 * @param m3pwr      PWM setting for motor 3
 */
void controlMovement(int m1pwr, int m2pwr, int m3pwr) {
  motor_1.writeMicroseconds(m1pwr);
  motor_2.writeMicroseconds(m2pwr);
  motor_3.writeMicroseconds(m3pwr);
}

/**
 * Function that rounds an input float number to a new float number with reduced
 * number of decimals.
 * @param value      input argument for number to reduce decimals
 * @param prec       number of decimals to be returned
 * @return           returns a float number rounded as specificed
 */
float roundNum(float value, unsigned char prec) {
  float pow_10 = pow(10.0f, (float)prec);
  return round(value * pow_10) / pow_10;
}
```

```
/*******************************************************************
 * SUBPROGRAM OF THE ARDUINO UNO CODE FOR COMMUNICATION BETWEEN THE
 * ARDUINO AND THE RASPBERRY PI
 *******************************************************************/
#include <Arduino.h>
#include <ArduinoJson.h>
#include "Communication.h"
#include "Functions.h"  // Remove after testing

/**
 * Function that handles communication Raspberry Pi. Takes in three arguments
 * with predefined types, and structures the data to JSON format. The JSON strings
 * are serialized and sent to Raspberry Pi.
 * @param arg1  Environment temperature value
 * @param arg2  Environment pressure value
 * @param arg3  Value that indicates if there is leak inside ROV
 */
void sendToRaspberry(float arg1, float arg2, bool arg3) {
  outDoc["Temp"] = roundNum(arg1, 1);
  outDoc["Depth"] = arg2;
  outDoc["Leak"] = arg3;

  // Format the data to serial
  serializeJson(outDoc, Serial);

  // Sending to Raspberry Pi
  Serial.println();
}

/**
 * Function that handles serial data communicated from the Raspberry Pi. First
 * eight function specific bool variables are declared. Those values stores
 * information about which control zones are interlocked. The data is stored to
 * a string variable which is in turn deserialized and loaded into a JSON
 * document. The JSON document data is accessed using keys and stores the values in
 * variables that are then used in function calls for movement and light control.
 */
void receiveFromRaspberry() {
  bool z1lock; bool z2lock; bool z3lock; bool z4lock;
  bool z5lock; bool z6lock; bool z7lock; bool z8lock;

  String  payload;
  payload = Serial.readStringUntil( '\n' );
  StaticJsonDocument<512> doc;
  deserializeJson(doc, payload);

  // If new values where communicated from Raspberry, update outputs
  setLights(doc["light"]);

  runZone = doc["runZone"];
  z1lock = doc["locked"][0];
  z2lock = doc["locked"][1];
  z3lock = doc["locked"][2];
  z4lock = doc["locked"][3];
  z5lock = doc["locked"][4];
  z6lock = doc["locked"][5];
  z7lock = doc["locked"][6];
  z8lock = doc["locked"][7];

```

```
}
```

```
#define Functions.h
#include "Servo.h"

// Declaring global servo objects
extern Servo motor_1;
extern Servo motor_2;
extern Servo motor_3;
extern Servo starboard_light;
extern Servo port_light;

// Declaring global variables
extern int runZone;
extern bool z1lock;
extern bool z2lock;
extern bool z3lock;
extern bool z4lock;
extern bool z5lock;
extern bool z6lock;
extern bool z7lock;
extern bool z8lock;

// Functions declaration
void setMotorSpeeds(int zone, bool z1lock, bool z2lock, bool z3lock,
bool z4lock, bool z5lock, bool z6lock, bool z7lock, bool z8lock);
void setLights(int pwr);
void fullStop();
void controlMovement(int m1pwr, int m2pwr, int m3pwr);
float roundNum(float value, unsigned char prec);
```

```
#define Communication.h
#include "ArduinoJson.h"

// Defining global JSON documents
extern StaticJsonDocument<512> inDoc;
extern StaticJsonDocument<48> outDoc;

// Functions declaration
void sendToRaspberry(float arg1, float arg2, bool arg3);
void receiveFromRaspberry();
```

Appendix G

Raspberry Pi code

```python
#!/usr/bin/env python3
"""
RASPBERRY PI MAIN PROGRAM FOR ROV OPERATION. PROGRAM USES MULTIPLE
PYTHON SUB SCRIPTS FOR COMMUNICATION, COLLISION AVOIDANCE AND GENERAL
SENSOR READINGS.
"""

import numpy as np
import threading
from math import *
import socket
import argparse

# Custom libraries imports for specific functionality
from sonarFunctionality.BlueRoboticsSonar import Ping360
from sonarFunctionality.Interlocking import InterlockingSystem
from COM.communication import TCPIn
from COM.communication import TCPOut
from COM.communication import UDP
from COM.communication import serialCom
import config


if __name__ == "__main__":
    # Terminal connection alternatives for sonar connection
    parser = argparse.ArgumentParser(description="Ping python library example.")
    parser.add_argument('--device', action="store", required=True, type=str, help="Ping device port.")
    parser.add_argument('--baudrate', action="store", type=int, default=2000000, help="Ping device baudrate.")
    args = parser.parse_args()

    # Establishes connection to Ping 360 sonar
    p = Ping360()
    p.connect_serial(args.device, args.baudrate)

    # Defining sonar parameters
    print("Initialized: %s" % p.initialize())
    p.set_transmit_frequency(1000)
    p.set_sample_period(50)
    p.set_number_of_samples(1200)
    p.set_range(50)

    # Initial zone control command to stand-still thrusters
    prevMode = -1

    # Initalize interlocking system for motor driving zones
    ils = InterlockingSystem()

    # Variables for internal logic
    objectData = []
    operatorForceReset = False

    # TCP communication variables
    HOST = "169.254.226.72"  # The IP address of the RASPBERRY Pi assigns to this communication
    PORT = 1422  # Port to listen on (non-privileged ports are > 1023)
    HEADERSIZE = 10

```

```python
    # Establishes a reliable data delivery TCP connection
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((HOST, PORT))    # Binds <eth0> port to requested IP and Port
    s.listen(2)             # Specifies number of unaccepted connection before refusing new

    # Initialize serial communication with seperate thread
    SerialThread = threading.Thread(target=serialCom)
    SerialThread.start()

    # Initialize UDP communication with seperate thread
    UDPThread = threading.Thread(target=UDP)
    UDPThread.start()

    # Continuously running while loop handling communication and various commands
    while 1:

        # Sets new angle for sonar to scan
        p.transmitAngle(config.angle)

        # Reads sonar echo strengths into array for one angle
        data = bytearray(getattr(p,'_data'))

        # Empties sonar data from previous iteration from array
        config.data_lst = []

        # Stores echo strengths in global variable
        for k in data :
            config.data_lst.append(k)

        # If no TCP connection already established, attempt to establish
        if config.address == "":
            print("[ATTENTION] Start computer script to initialize TCP communication with ROV!")
            TCPOut(s, HOST, PORT, HEADERSIZE)
            TCPThread = threading.Thread(target=TCPIn)
            TCPThread.start()

        # Updating system variables for communication with GUI
        config.step = p.get_step()
        config.interlockedZones = ils.lockedZones

        # Sends sensor and system data to GUI
        TCPOut(s, HOST, PORT, HEADERSIZE)

        # If new command for mode control is received, perform changes
        if config.mode != prevMode:
            print("A new mode has been activated")
            p.changeOperatingMode(config.mode)
            prevMode = config.mode

        # Checks for operator induced forced reset of interlocked zones
        if config.forceReset:
            print("Operator is forcing reset of all interlocked zones")
            ils.resetAllZones()

        # If object is found, interlock the current zone
        if ils.findObject(config.data_lst):
            ils.setInterlockZone(ils.findZone(config.angle), config.angle)

```

```python
        # Incrementing for next sonar-scan
        config.angle = (config.angle + p.get_step()) % 400

        # If the current angle is equal to any of the angles that were used to lock a zone
        if ils.checkIfResetPermitted(config.angle):
            # Reset that zone as the sonar has scanned that zone again, and no object is detect
            ils.resetInterlockZone(ils.findZone(config.angle))

```

```python
"""
RASPBERRY PI SUB PROGRAM CONTAINING GLOBAL VARIABLES USED BETWEEN
THE OTHER PYTHON SCRIPTS.
"""

# Global variables being received from GUI
light= 0
motorSpeed = 0
runZone = -1
mode = 1
forceReset = False
takeHighResPhoto = False
takeVideo = False

# Global variables being sent to GUI
temp = 0
depth = 0
leak = False
angle = 0
data_lst = []
step = 0
interlockedZones = [False] * 8
salinity = 0
conductivity = 150
density = 1000

# General functionality not used by communication
address = ""
clientsocket = ""
newArduinoCommands = False
```

```python
"""
RASPBERRY PI SUB PROGRAM CONTAINING A PING 360 CLASS USED
FOR CONTROLLING AND READING FROM THE SCANNING IMAGING SONAR

MOST OF THIS CLASS, AND ALL OF THE <brping> MODULES IMPORTED
BELOW, ARE DEVELOPED BY THE MANUFACTORER OF THE SCANNING IMAGING
SONAR, BLUEROBOTICS.
"""

from brping import definitions
from brping import PingDevice
from brping import pingmessage

class Ping360(PingDevice):
    def initialize(self):
        if not PingDevice.initialize(self):
            return False
        if (self.readDeviceInformation() is None):
            return False
        self._speed_of_sound = 1500
        self._step = 2
        return True

    ##
    # @brief Get a device_data message from the device\n
    # Message description:\n
    # This message is used to communicate the current sonar state. If the data field is populated, the other fields indicate the sonar state when the data was captured. The time taken before the response to the command is sent depends on the difference between the last angle scanned and the new angle in the parameters as well as the number of samples and sample interval (range). To allow for the worst case reponse time the command timeout should be set to 4000 msec.
    #
    # @return None if there is no reply from the device, otherwise a dictionary with the following keys:\n
    # mode: Operating mode (1 for Ping360)\n
    # gain_setting: Analog gain setting (0 = low, 1 = normal, 2 = high)\n
    # angle: Units: gradian Head angle\n
    # transmit_duration: Units: microsecond Acoustic transmission duration (1~1000 microseconds)\n
    # sample_period: Time interval between individual signal intensity samples in 25nsec increments (80 to 40000 == 2 microseconds to 1000 microseconds)\n
    # transmit_frequency: Units: kHz Acoustic operating frequency. Frequency range is 500kHz to 1000kHz, however it is only practical to use say 650kHz to 850kHz due to the narrow bandwidth of the acoustic receiver.\n
    # number_of_samples: Number of samples per reflected signal\n
    # data: 8 bit binary data array representing sonar echo strength\n
    def get_device_data(self):
        if self.request(definitions.PING360_DEVICE_DATA, 4) is None:
            print("empty request")
            return None
        data = ({
            "mode": self._mode,  # Operating mode (1 for Ping360)
            "gain_setting": self._gain_setting,  # Analog gain setting (0 = low, 1 = normal, 2 = high)
            "angle": self._angle,  # Units: gradian Head angle
            "transmit_duration": self._transmit_duration,  # Units: microsecond Acoustic transmission duration (1~1000 microseconds)
```

```
            "sample_period": self._sample_period,  # Time interval between individual signal intensity samples in 25nsec increments (80 to 40000 == 2 microseconds to 1000 microseconds)
            "transmit_frequency": self._transmit_frequency,  # Units: kHz Acoustic operating frequency. Frequency range is 500kHz to 1000kHz, however it is only practical to use say 650kHz to 850kHz due to the narrow bandwidth of the acoustic receiver.
            "number_of_samples": self._number_of_samples,  # Number of samples per reflected signal
            "data": self._data,  # 8 bit binary data array representing sonar echo strength
        })
        return data

    ##
    # @brief Send a device_id message to the device\n
    # Message description:\n
    # Change the device id\n
    # Send the message to write the device parameters, then read the values back from the device\n
    #
    # @param id - Device ID (1-254). 0 and 255 are reserved.
    # @param reserved - reserved
    #
    # @return If verify is False, True on successful communication with the device. If verify is False, True if the new device parameters are verified to have been written correctly. False otherwise (failure to read values back or on verification failure)
    def device_id(self, id, reserved, verify=True):
        m = pingmessage.PingMessage(definitions.PING360_DEVICE_ID)
        m.id = id
        m.reserved = reserved
        m.pack_msg_data()
        self.write(m.msg_data)
        if self.request(definitions.PING360_DEVICE_ID) is None:
            return False
        # Read back the data and check that changes have been applied
        if (verify
                and (self._id != id or self._reserved != reserved)):
            return False
        return True  # success        m.id = id
        m.reserved = reserved
        m.pack_msg_data()
        self.write(m.msg_data)

    def control_reset(self, bootloader, reserved):
        m = pingmessage.PingMessage(definitions.PING360_RESET)
        m.bootloader = bootloader
        m.reserved = reserved
        m.pack_msg_data()
        self.write(m.msg_data)

    def control_transducer(self, mode, gain_setting, angle, transmit_duration, sample_period, transmit_frequency, number_of_samples, transmit, reserved):
        m = pingmessage.PingMessage(definitions.PING360_TRANSDUCER)
        m.mode = mode
        m.gain_setting = gain_setting
        m.angle = angle
        m.transmit_duration = transmit_duration
        m.sample_period = sample_period
```

```
        m.transmit_frequency = transmit_frequency
        m.number_of_samples = number_of_samples
        m.transmit = transmit
        m.reserved = reserved
        m.pack_msg_data()
        self.write(m.msg_data)


    def set_mode(self, mode):
        self.control_transducer(
            mode,
            self._gain_setting,
            self._angle,
            self._transmit_duration,
            self._sample_period,
            self._transmit_frequency,
            self._number_of_samples,
            0,
            0
        )
        return self.wait_message([definitions.PING360_DEVICE_DATA,
definitions.COMMON_NACK], 4.0)

    def set_gain_setting(self, gain_setting):
        self.control_transducer(
            self._mode,
            gain_setting,
            self._angle,
            self._transmit_duration,
            self._sample_period,
            self._transmit_frequency,
            self._number_of_samples,
            0,
            0
        )
        return self.wait_message([definitions.PING360_DEVICE_DATA,
definitions.COMMON_NACK], 4.0)

    def set_angle(self, angle):
        self.control_transducer(
            self._mode,
            self._gain_setting,
            angle,
            self._transmit_duration,
            self._sample_period,
            self._transmit_frequency,
            self._number_of_samples,
            0,
            0
        )
        return self.wait_message([definitions.PING360_DEVICE_DATA,
definitions.COMMON_NACK], 4.0)

    def set_transmit_duration(self, transmit_duration):
        self.control_transducer(
            self._mode,
            self._gain_setting,
            self._angle,
            transmit_duration,
            self._sample_period,
```

```python
            self._transmit_frequency,
            self._number_of_samples,
            0,
            0
        )
        return self.wait_message([definitions.PING360_DEVICE_DATA, definitions.COMMON_NACK], 4.0)

    def set_sample_period(self, sample_period):
        self.control_transducer(
            self._mode,
            self._gain_setting,
            self._angle,
            self._transmit_duration,
            sample_period,
            self._transmit_frequency,
            self._number_of_samples,
            0,
            0
        )
        return self.wait_message([definitions.PING360_DEVICE_DATA, definitions.COMMON_NACK], 4.0)

    def set_transmit_frequency(self, transmit_frequency):
        self.control_transducer(
            self._mode,
            self._gain_setting,
            self._angle,
            self._transmit_duration,
            self._sample_period,
            transmit_frequency,
            self._number_of_samples,
            0,
            0
        )
        return self.wait_message([definitions.PING360_DEVICE_DATA, definitions.COMMON_NACK], 4.0)

    def set_number_of_samples(self, number_of_samples):
        self.control_transducer(
            self._mode,
            self._gain_setting,
            self._angle,
            self._transmit_duration,
            self._sample_period,
            self._transmit_frequency,
            number_of_samples,
            0,
            0
        )
        return self.wait_message([definitions.PING360_DEVICE_DATA, definitions.COMMON_NACK], 4.0)


    def readDeviceInformation(self):
        return self.request(definitions.PING360_DEVICE_DATA)

    def transmitAngle(self, angle):
        self.control_transducer(
            0, # reserved
```

```
            self._gain_setting,
            angle,
            self._transmit_duration,
            self._sample_period,
            self._transmit_frequency,
            self._number_of_samples,
            1,
            0
        )
        return self.wait_message([definitions.PING360_DEVICE_DATA, definitions.COMMON_NACK], 0.5)

    def transmit(self):
        return self.transmitAngle(self._angle)


    """
    Below functions are created specifically for the ROV project, functions over are developed
    by the manufactor of the sonar, BlueRovotics.
    """

    # Function that returns the currently set speed of sound
    def get_speed_of_sound(self):
        return self._speed_of_sound

    # Function that changes the speed of sound
    def set_speed_of_sound(self, newSpeed):
        if (newSpeed == self.get_speed_of_sound()):
            print("Requested speed of sound is already set")
            return
        else:
            self._speed_of_sound = newSpeed

    # Function that returns the sample period set for the sonar
    def samplePeriod(self):
        # Multiply with samplePeriodTickDuration which is 25 nanoseconds
        return self._sample_period * 25E-9

    # Returns the currently set scanning range
    def get_range(self):
        return self.samplePeriod() * self._number_of_samples * self.get_speed_of_sound() / 2

    # Sets new sonar scan range
    def set_range(self, newRange):
        # Checks if new argument is different from set range
        if (newRange == self.get_range()):
            return
        else:
            # Calculate the new sample period to achieve requested distance
            self._sample_period = int(newRange/(self._number_of_samples*25E-9*750))

    # Sets new step size, that indicates how many angles the sonar jumps for every iteration
    def set_step(self, newStep):
        self._step = newStep

    # Returns the current step
    def get_step(self):
```

```python
        return self._step

    # Function that changes settings needed for the sonar to scan in a new mode
    def changeOperatingMode(self, newMode):
        # Checks if input is different from last iteration
        if newMode == 0:
            self.set_range(20) # Short range collision avoidance mode
            self.set_step(4)
            self.set_gain_setting(0)
        elif newMode == 1:
            self.set_range(50) # Medium range collision avoidance mode
            self.set_step(2)
            self.set_gain_setting(1)
        elif newMode == 2:
            self.set_range(2) # Aquaculture inspection mode
            self.set_step(10)
            self.set_gain_setting(0)
        elif newMode == 3:
            self.set_range(4) # Aquaculture inspection mode
            self.set_step(8)
            self.set_gain_setting(0)
        else:
            print("Did not recognize mode command")
            print("Corrupt or invalid data given")
            return
```

```python
"""
RASPBERRY PI SUB PROGRAM CONTAINING THE LOGIC TO HANDLE
THE COMMUNICATION BETWEEN ALL DEVICES CONNECTED TO THE RPI.
INCLUDES SERIAL COMMUNICATION, TCP AND UDP.
"""

import socket
import pickle
import numpy as np
import config
import threading
import struct
import math
from imutils.video import VideoStream
import serial
import json
import time


class FrameSegment(object):

    # Initialization of functionality that handles dividing picture frames to correctly sized UDP datagrams
    def __init__(self, sock, port, addr="169.254.226.73"):
        self.s = sock
        self.port = port
        self.addr = addr
        self.MAX_DGRAM = 2**16
        self.MAX_IMAGE_DGRAM = self.MAX_DGRAM - 64

    # Function that takes in a frame, compresses it, divides it into UDP datagrams and
    # sends it over UDP to the GUI
    def udp_frame(self, img):
        # Compress image to .jpg format
        compress_img = cv2.imencode(".jpg", img)[1]
        dat = compress_img.tostring()
        size = len(dat)
        # Finds number of datagrams needed to be sent for this frame
        num_of_segments = math.ceil(size/(self.MAX_IMAGE_DGRAM))
        array_pos_start = 0

        # Sends out all the datagrams needed for the frame
        while num_of_segments:
            array_pos_end = min(size, array_pos_start + self.MAX_IMAGE_DGRAM)
            self.s.sendto(
                struct.pack("B", num_of_segments) +
                dat[array_pos_start:array_pos_end],
                (self.addr, self.port)
                )
            array_pos_start = array_pos_end
            num_of_segments -= 1




# Function that handles the TCP data coming from the GUI
def TCPIn():

```

```python
    HEADERSIZE = 10

    # Constantly checking for new messages
    while 1:
        receiving = True
        full_msg = b''
        new_msg = True
        incoming_message = config.clientsocket.recv(8192)

        while receiving:

            if new_msg:
                msglen = int(incoming_message[:HEADERSIZE])
                new_msg = False

            full_msg += incoming_message

            # If full message received, update global variables
            if len(full_msg)-HEADERSIZE == msglen:
                GuiDataIn = pickle.loads(full_msg[HEADERSIZE:])
                print("[ATTENTION] New data has been applied to global variables from GUI commands")
                config.light = GuiDataIn["light"]
                config.motorSpeed = GuiDataIn["motorSpeed"]
                config.runZone = GuiDataIn["runZone"]
                config.forceReset = GuiDataIn["forceReset"]
                config.mode = GuiDataIn["mode"]
                config.takeHighResPhoto = GuiDataIn["takePhoto"]
                config.takeVideo = GuiDataIn["takeVideo"]
                config.newArduinoCommands = True

                # Resetting variables for next iteration
                receiving = False
                new_msg = True
                full_msg = b''


# Function that handles the TCP data to be sent to the GUI
def TCPOut(s, HOST, PORT, HEADERSIZE):
    communicating = True
    startReceive = True

    while communicating:
        receiving = True

        # If no connection is established, try to find one
        if not config.address:
            config.clientsocket, config.address = s.accept()
            print(f"Connection from {config.address} has been established.")

        # Finalizing dicitionary with all values to be sent to GUI
        GuiDataOut = {
            "image": "",
            "temp": config.temp,
            "depth": config.depth,
            "leak": config.leak,
            "angle": config.angle,
            "step": config.step,
            "lockedZones": config.interlockedZones,
            "dataArray": config.data_lst,
```

```python
            "salinity": config.salinity,
            "conductivity": config.conductivity,
            "density": config.density

        }

        # Serializing the dicitionary and sending
        msg = pickle.dumps(GuiDataOut)
        msg = bytes(f"{len(msg):<{HEADERSIZE}}", 'utf-8') + msg
        config.clientsocket.send(msg)

        communicating = False


# Function that handles the UDP communication with GUI
def UDP():
    # Establish connection with server
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    port = 20001

    # Declare object for handling image frames
    fs = FrameSegment(s, port)

    # Variables for naming photo and video files
    photoNum = 0
    videoNum = 0

    # Opens camera port and defines video format
    cap = VideoStream(src=0).start()
    size = (640, 480)

    while 1:
        # If commanded from GUI, take photo and save to determined path
        if config.takeHighResPhoto:

            photoNum += 1
            photo = cap.read()

            status = cv2.imwrite(f'/home/pi/Programs/Photos/photo_{photoNum}.png', photo)
            print(f'Image written to file system status: {status}')
            time.sleep(1)    # Sleeps for 1 second before resuming UDP video stream
            config.takeHighResPhoto = False

        # If no commands to take picture, resume video stream over UDP to GUI
        while not config.takeHighResPhoto:
            frame = cap.read()

            # If user commands to save video, store video to file
            if config.takeVideo:
                if not vidConfigured:
                    videoNum += 1
                    result = cv2.VideoWriter(f'/home/pi/Programs/Videos/Video{videoNum}.avi',
                    cv2.VideoWriter_fourcc(*'mp4v'), 12, size)
                    vidConfigured = True

                result.write(frame) # Writing to disk as a video
            else:
                vidConfigured = False
```

```python

            fs.udp_frame(frame) # Sending to GUI using UDP communication

    cap.release()
    cv2.destroyAllWindows()
    s.close()


# Function that handles serial communication with Arduino Uno and combination sensor
def serialCom():

    # Initialize serial communication with Arduino UNO
    ardSer = serial.Serial('/dev/ttyACM0', 9600, timeout=1,
    parity=serial.PARITY_NONE, bytesize=serial.EIGHTBITS, stopbits=serial.STOPBITS_ONE)
    print(f'Arduino serial communication status: {ardSer.isOpen()}')

    # Initialize serial communication with conductivity/combination sensor
    condSer = serial.Serial('/dev/ttyUSB1', 9600)
    print(f'Conductivity sensor communication status: {condSer.isOpen()}')

    time.sleep(2)

    # Continuously send and receive over serial connection
    while 1:

        # Commands conductivity sensor to conduct sample of values
        condSer.write("do_sample\n".encode())

        # Commands for values, and reads response to global variables
        config.salinity = getAanderaaData(condSer, "get_salinity\n")
        soundSpeedReading = getAanderaaData(condSer, "get_soundspeed\n")
        config.density = getAanderaaData(condSer, "get_density\n")
        config.conductivity = getAanderaaData(condSer, "get_conductivity\n")

        # If new commands has been updated for Arduino Uno, structure data from global variable, serialize and send
        if config.newArduinoCommands:
            ArdDataOut = {}
            ArdDataOut["light"] = config.light
            ArdDataOut["runZone"] = config.runZone
            ArdDataOut["locked"] = config.interlockedZones
            ArdDataOut = json.dumps(ArdDataOut)
            ardSer.write(ArdDataOut.encode())
            config.newArduinoCommands = False

        # If the serial input buffer reserved for Arduino traffic has data, unserialize and store in global variables
        if ardSer.in_waiting > 0:
            ArdDataIn = json.loads(ardSer.readline())   # Deserializes input from Arduino
            config.temp = ArdDataIn["Temp"]
            config.depth = ArdDataIn["Depth"]
            config.leak = ArdDataIn["Leak"]

        # Have to read and purge input buffer for combination sensor, as old data can ruin future readings
        condIn = condSer.readline()


```

```python
# Function that compresses frame read from camera
def commpressImage(img, k):
    width = int((img.shape[1])/k)
    height = int((img.shape[0])/k)
    return cv2.resize(img, (width, height), interpolation=cv2.INTER_AREA)

# Function that sends commands and reads response. Reads value
# based on given parameter
def getAanderaaData(condSer, request_str):
    condIn = b''
    condIn = condSer.readline()  # Have to read the buffer to stop future splitting issues
    condSer.write(request_str.encode())
    condIn = condSer.readline().decode()
    data = condIn.split('\t')
    data = data[-1]
    data = data.replace('\r\n', '')
    return float(data)
```

```python
"""
RASPBERRY PI SUB PROGRAM CONTAINING THE INTERLOCKING LOGIC
NEEDED FOR ASSISTING THE OPERATOR OF THE ROV TO CONTROL THE
ROV IN TIGHT SPACES.
"""

class InterlockingSystem:
    def __init__(self):
        print("Interlocking system initalized")
        self.lockedZones = [False] * 8
        self.zoneLockedAngles = [None] * 8

    # Resets all interloked zones
    def resetAllZones(self):
        print("Operator-forced reset of all interlocked zones")
        self.lockedZones = [False] * 8

    # By taking in the currently scanned angle, finds which zone the angle
    # is a part of, and return this zone
    def findZone (self, angle):
        if 175 < angle <= 225:
            return 0
        elif 125 < angle <= 175:
            return 1
        elif 75 < angle <= 125:
            return 2
        elif 25 < angle <= 75:
            return 3
        elif 325 < angle <= 375:
            return 5
        elif 275 < angle <= 325:
            return 6
        elif 225 < angle <= 275:
            return 7
        else:
            return 4

    # Function taking in the echo strengths from sonar, and finds if these
    # values within close proximity is above a certain treshold, and determines
    # if an object was located
    def findObject (self, dataPoints):

        # Constants for object detection, can be adjusted for range and sensitivity
        numOfValues = 100
        thresholdObjDetect = 40

        # Slicing list to appropiate values, changing this means changing what range are scanned for objects
        objectData = dataPoints[numOfValues:2*numOfValues]
        avrObj = sum(objectData)/numOfValues
        # print("Average echo strength (0-255): ", avrObj)

        # If average of datapoints is above threshold return true (object is detected)
        if avrObj > thresholdObjDetect:
            return True
        else:
            return False

```

```python
    # Takes in zone and angle and interlocks that zone for no movement in that direction
    # Additionally the angle which the object was located at is saved, for later resetting
    # of zone during normal operation
    def setInterlockZone (self, zone, angle):
        if zone == 0:
            self.lockedZones[0] = True
            self.zoneLockedAngles[0] = angle
        elif zone == 1:
            self.lockedZones[1] = True
            self.zoneLockedAngles[1] = angle
        elif zone == 2:
            self.lockedZones[2] = True
            self.zoneLockedAngles[2] = angle
        elif zone == 3:
            self.lockedZones[3] = True
            self.zoneLockedAngles[3] = angle
        elif zone == 4:
            self.lockedZones[4] = True
            self.zoneLockedAngles[4] = angle
        elif zone == 5:
            self.lockedZones[5] = True
            self.zoneLockedAngles[5] = angle
        elif zone == 6:
            self.lockedZones[6] = True
            self.zoneLockedAngles[6] = angle
        elif zone == 7:
            self.lockedZones[7] = True
            self.zoneLockedAngles[7] = angle
        else:
            print("Invalid set zone given")

    # Checks if the scanned angle has been interlocked last revolution
    # and if no object was found this reviolution, reset the zone
    def checkIfResetPermitted (self, angle):
        for i in range(len(self.zoneLockedAngles)):
            if (self.zoneLockedAngles[i] == angle):
                self.resetInterlockZone(self.findZone(angle))

    # Takes in an angle, and resets the zone containing that angle
    def resetInterlockZone (self, zone):
        if zone == 0:
            self.lockedZones[0] = False
        elif zone == 1:
            self.lockedZones[1] = False
        elif zone == 2:
            self.lockedZones[2] = False
        elif zone == 3:
            self.lockedZones[3] = False
        elif zone == 4:
            self.lockedZones[4] = False
        elif zone == 5:
            self.lockedZones[5] = False
        elif zone == 6:
            self.lockedZones[6] = False
        elif zone == 7:
            self.lockedZones[7] = False
        else:
            print("Invalid reset zone given")
```

Appendix H

GUI code

```python

"""
MAIN PROGRAM FOR GUI APPLICATION. HANDLES THE COMMUNICATION WITH THE
RASPBERRY PI INSIDE THE ROV ENCLOSURE. THIS PROGRAM CONTAINS A TCP
CLIENT AND A UDP SERVER. PROGRAM IS DEPENDANT ON CONFIG.PY FOR GLOBAL
VARIABLES, AND INTERFACING.PY FOR GUI APPLICATION FUNCTIONALITY.
"""

from xmlrpc.client import Server
import socket
import pickle
import cv2
import time
import numpy as np
from PyQt5.QtWidgets import QWidget, QApplication, QLabel, QVBoxLayout
import sys
import cv2
from math import *
import threading
import config
from interfacing import VideoThread, App
import struct

# Sonar constants needed for plotting visualization
MAX_RANGE = 80*200*1450/2
LENGTH = 640
CENTER = (LENGTH/2,LENGTH/2)
image = np.zeros((LENGTH, LENGTH, 1), np.uint8)

"""
Function that creates a circular plot of the sonar readings. Takes
in the angle scanned, step size for next angle and a list of echo
strengths.
"""
def plotSonarInput(angle, step, data_lst):

    linear_factor = len(data_lst)/CENTER[0]
    for i in range(int(CENTER[0])):
        if(i < CENTER[0]*MAX_RANGE/MAX_RANGE):
            try:
                pointColor = data_lst[int(i*linear_factor-1)]
            except IndexError:
                pointColor = 0
        else:
            pointColor = 0
        for k in np.linspace(0,step,8*step):
            image[int(CENTER[0]+i*cos(2*pi*(angle+k)/400)),
int(CENTER[1]+i*sin(2*pi*(angle+k)/400)), 0] = pointColor

    # Updates GUI animation with new sonar plot
    a.update_sonar(cv2.applyColorMap(image, cv2.COLORMAP_JET))

"""
Function that handles the TCP Communication between the GUI and the
Raspberry Pi.
"""
def TCPCom():
    # Define connection parameters of RPi server
    SERVER = "169.254.226.72"
```

```python
    PORT = 1422
    HEADERSIZE = 10

    # Using socket library, initialize a communication object
    # and initialize TCP connection.
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((SERVER, PORT))

    print(f"[NEW CONNECTION] With {SERVER} established")


    full_msg = b''
    new_msg = True
    while 1:
        # Receives TCP packets and stores in variable
        msg = s.recv(8192) # 8192

        # If the start of a data message is sent, find length of expected information
        if new_msg:
            msglen = int(msg[:HEADERSIZE])
            new_msg = False

        # Sum all packets to a full message
        full_msg += msg

        # If length of message is as previously decided, unpack data and call functions
        if len(full_msg)-HEADERSIZE == msglen:

            RaspDataIn = pickle.loads(full_msg[HEADERSIZE:])

            a.setDisplayValues(RaspDataIn["temp"], RaspDataIn["depth"], RaspDataIn["leak"],
            RaspDataIn["lockedZones"], RaspDataIn["salinity"], RaspDataIn["conductivity"],
            RaspDataIn["density"])
            plotSonarInput(RaspDataIn["angle"], RaspDataIn["step"], RaspDataIn["dataArray"])

            # Resets for next message
            new_msg = True
            full_msg = b""

        # If TCP packet was not succesfully unpacked last message
        # reset the input buffer
        if len(full_msg) > 2500:
            new_msg = True
            full_msg = b""


        # If user has performed actions on GUI, sends commands over TCP to RPi
        if config.newCommands:
            print("[ATTENTION] New commands sent to Raspberry")
            config.newCommands = False

            RaspDataOut = {
                "light": config.light,
                "motorSpeed": config.motorSpeed,
                "runZone": config.runZone,
```

```
                "forceReset": config.forceReset,
                "mode": config.mode,
                "takePhoto": config.takePhoto,
                "takeVideo": config.takeVideo
            }

            RaspDataOut = pickle.dumps(RaspDataOut)
            RaspDataOut = bytes(f'{len(RaspDataOut):<{HEADERSIZE}}', 'utf-8') + RaspDataOut
            s.send(RaspDataOut)

"""
Function that handles the UDP communication between the GUI and the
Raspberry Pi.
"""
def UDPCom():
    # Datagram set to maximum allowable size
    MAX_DGRAM = 2**16
    dat = b''

    """
    Function that dumps the UDP buffer.
    """
    def dump_buffer(s):
        while True:
            seg, addr = s.recvfrom(MAX_DGRAM)
            print(seg[0])
            if struct.unpack('B', seg[0:1])[0] == 1:
                print("UDP input buffer emptied")
            break

    # Using socket library, initialize a communication object
    # and initialize UDP connection.
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind(('169.254.226.73', 20001))

    # Dumps buffer, so that completely new data will be read at this point
    dump_buffer(s)

    # Continuously checking for UDP datagrams and unpacks if an entire message
    # has been received, else summing up datagrams until this is true
    while 1:
        seg, _ = s.recvfrom(MAX_DGRAM)
        if struct.unpack("B", seg[0:1])[0] > 1:
            dat += seg[1:]
        else:
            dat += seg[1:]
            img = cv2.imdecode(np.frombuffer(dat, dtype=np.uint8), 1)
            try:
                config.rovCamera = img
            except:
                print("error (-215:Assertion failed)")
            if cv2.waitKey(20) & 0xFF == ord('q'):
                break
            dat = b''

    cv2.destroyAllWindows()
    s.close()
```

```python
if __name__=="__main__":

    # Initializing GUI objects
    app = QApplication(sys.argv)
    a = App()

    # Opening an UDP server in GUI application
    cam_communication = threading.Thread(target=UDPCom)
    cam_communication.start()

    # Opening a TCP client in GUI application
    other_communication = threading.Thread(target=TCPCom)
    other_communication.start()

    # Opens GUI as a seperate window
    a.show()
    sys.exit(app.exec_())
```

```python
"""
PYTHON SCRIPT FOR GUI APPLICATION FOR ROV. CONTAINS THE GLOBAL VARIABLES
THAT ARE SHARED BETWEEN MAIN.PY AND INTERFACING.PY, RESPECTIVELY COMMUNICATION
AND MAIN GUI SCRIPTS.
"""
import numpy as np

# Holds the frame communicated from Raspberry Pi
rovCamera = np.array([])

# Commands from GUI to raspberry with initial values
light = 0
motorSpeed = 0
runZone = -1
forceReset = False
mode = 1
takePhoto = False
takeVideo = False

# Sensor information from Raspberry with inital values
temp = 0
pressure = 0
leak = 0
angle = 0
interlockedZones = [True] * 8

# Boolean flag that indicates if new actions have been
# performed on GUI
newCommands = False
```

```python
"""
PYTHON SCRIPT FOR GUI APPLICATION FOR ROV. CONTAINS AUTOGENERATED
GUI OBJECTS FROM QT DESIGNER AND FUNCTIONS NEEDED FOR GUI AND
COMMUNICATION THE WORK TOGETHER.
"""
from PyQt5 import QtGui
from PyQt5 import QtWidgets
from PyQt5 import QtCore
from PyQt5.QtGui import QFont
from PyQt5.QtWidgets import QWidget, QApplication, QLabel, QVBoxLayout
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import pyqtSignal, pyqtSlot, Qt, QThread,QTime,QTimer
import cv2
import numpy as np
import config


class VideoThread(QThread):
    change_pixmap_signal = pyqtSignal(np.ndarray)

    def __init__(self):
        super().__init__()
        self._run_flag = True

    def run(self):
        while self._run_flag:
            try:
                if (len(config.rovCamera) > 10):
                    self.change_pixmap_signal.emit(config.rovCamera)
                    config.rovCamera = np.array([])
            except:
                continue

    def stop(self):
        self._run_flag = False
        self.wait()

class App(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("ROV-AIP USER INTERFACE")
        self.setGeometry(0,0,2560,1440)
        self.setStyleSheet("background-color: rgb(93, 93, 93);")
        self.g = 9.81 # [m/s^2]
        self.density = 1000 # [kg/m^3]
        self.prevLockedZones = [True] * 8

        self.image_label = QLabel(self)
        self.image_label.setGeometry(690, 0, 1221, 671)
        self.image_label.setFrameShape(QtWidgets.QFrame.Panel)

        self.sonar=QLabel(self)
        self.sonar.setText("Sonar_Placeholder")
        self.sonar.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.sonar.setGeometry(15, 15, 640, 640)
        self.sonar.setFrameShape(QtWidgets.QFrame.Panel)

        self.Forward = QtWidgets.QToolButton(self)
        self.Forward.setGeometry(QtCore.QRect(1280, 770, 51, 41))
```

```
        self.Forward.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Forward.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_forward.png'))
        self.Forward.setIconSize(QtCore.QSize(32, 32))
        self.Forward.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.Forward.setArrowType(QtCore.Qt.NoArrow)
        self.Forward.setObjectName("Forward")
        self.Forward.pressed.connect(lambda: self.activateZone(0))
        self.Forward.released.connect(self.release)

        self.Reverse = QtWidgets.QToolButton(self)
        self.Reverse.setGeometry(QtCore.QRect(1280, 870, 51, 41))
        self.Reverse.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Reverse.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_reverse.png'))
        self.Reverse.setIconSize(QtCore.QSize(32, 32))
        self.Reverse.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.Reverse.setArrowType(QtCore.Qt.NoArrow)
        self.Reverse.setObjectName("Reverse")
        self.Reverse.pressed.connect(lambda: self.activateZone(4))
        self.Reverse.released.connect(self.release)

        self.Left = QtWidgets.QToolButton(self)
        self.Left.setGeometry(QtCore.QRect(1210, 820, 51, 41))
        self.Left.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Left.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_left.png'))
        self.Left.setIconSize(QtCore.QSize(32, 32))
        self.Left.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.Left.setArrowType(QtCore.Qt.NoArrow)
        self.Left.setObjectName("Left")
        self.Left.pressed.connect(lambda: self.activateZone(6))
        self.Left.released.connect(self.release)

        self.Right = QtWidgets.QToolButton(self)
        self.Right.setGeometry(QtCore.QRect(1350, 820, 51, 41))
        self.Right.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Right.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_right.png'))
        self.Right.setIconSize(QtCore.QSize(32, 32))
        self.Right.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.Right.setArrowType(QtCore.Qt.NoArrow)
        self.Right.setObjectName("Right")
        self.Right.pressed.connect(lambda: self.activateZone(2))
        self.Right.released.connect(self.release)

        self.ForwardRight = QtWidgets.QToolButton(self)
        self.ForwardRight.setGeometry(QtCore.QRect(1350, 770, 51, 41))
        self.ForwardRight.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ForwardRight.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_forward_right.png'))
        self.ForwardRight.setIconSize(QtCore.QSize(32, 32))
        self.ForwardRight.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.ForwardRight.setObjectName("ForwardRight")
        self.ForwardRight.pressed.connect(lambda: self.activateZone(1))
        self.ForwardRight.released.connect(self.release)

        self.ForwardLeft = QtWidgets.QToolButton(self)
        self.ForwardLeft.setGeometry(QtCore.QRect(1210, 770, 51, 41))
        self.ForwardLeft.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ForwardLeft.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_forward_left.png'))
        self.ForwardLeft.setIconSize(QtCore.QSize(32, 32))
```

```
        self.ForwardLeft.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.ForwardLeft.setObjectName("ForwardLeft")
        self.ForwardLeft.pressed.connect(lambda: self.activateZone(7))
        self.ForwardLeft.released.connect(self.release)

        self.ReverseLeft = QtWidgets.QToolButton(self)
        self.ReverseLeft.setGeometry(QtCore.QRect(1210, 870, 51, 41))
        self.ReverseLeft.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ReverseLeft.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_reverse_left.png'))
        self.ReverseLeft.setIconSize(QtCore.QSize(32, 32))
        self.ReverseLeft.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.ReverseLeft.setObjectName("ReverseLeft")
        self.ReverseLeft.pressed.connect(lambda: self.activateZone(5))
        self.ReverseLeft.released.connect(self.release)

        self.ReverseRight = QtWidgets.QToolButton(self)
        self.ReverseRight.setGeometry(QtCore.QRect(1350, 870, 51, 41))
        self.ReverseRight.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ReverseRight.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_reverse_right.png'))
        self.ReverseRight.setIconSize(QtCore.QSize(32, 32))
        self.ReverseRight.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.ReverseRight.setObjectName("ReverseRight")
        self.ReverseRight.pressed.connect(lambda: self.activateZone(3))
        self.ReverseRight.released.connect(self.release)

        self.CounterClockwise = QtWidgets.QToolButton(self)
        self.CounterClockwise.setGeometry(QtCore.QRect(1210, 710, 51, 41))
        self.CounterClockwise.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.CounterClockwise.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_rotate_left.png'))
        self.CounterClockwise.setIconSize(QtCore.QSize(32, 32))
        self.CounterClockwise.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.CounterClockwise.setObjectName("CounterClockwise")
        self.CounterClockwise.pressed.connect(lambda: self.activateZone(8))
        self.CounterClockwise.released.connect(self.release)

        self.Clockwise = QtWidgets.QToolButton(self)
        self.Clockwise.setGeometry(QtCore.QRect(1350, 710, 51, 41))
        self.Clockwise.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Clockwise.setIcon(QtGui.QIcon('GUI_interface\Icons\icon_rotate_right.png'))
        self.Clockwise.setIconSize(QtCore.QSize(32, 32))
        self.Clockwise.setPopupMode(QtWidgets.QToolButton.InstantPopup)
        self.Clockwise.setObjectName("Clockwise")
        self.Clockwise.pressed.connect(lambda: self.activateZone(9))
        self.Clockwise.released.connect(self.release)

        self.Temp = QtWidgets.QLCDNumber(self)
        self.Temp.setGeometry(QtCore.QRect(30, 710, 141, 31))
        self.Temp.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Temp.setFrameShadow(QtWidgets.QFrame.Plain)
        self.Temp.setSmallDecimalPoint(True)
        self.Temp.setDigitCount(8)
        self.Temp.setSegmentStyle(QtWidgets.QLCDNumber.Flat)
        self.Temp.setProperty("value", 0.0)
        self.Temp.setObjectName("Temp")
```

```python
        self.Temp.display(999)

        self.Depth = QtWidgets.QLCDNumber(self)
        self.Depth.setGeometry(QtCore.QRect(30, 790, 141, 31))
        self.Depth.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Depth.setFrameShadow(QtWidgets.QFrame.Plain)
        self.Depth.setSmallDecimalPoint(True)
        self.Depth.setDigitCount(8)
        self.Depth.setSegmentStyle(QtWidgets.QLCDNumber.Flat)
        self.Depth.setObjectName("Depth")
        self.Depth.display(999)

        self.Salinity = QtWidgets.QLCDNumber(self)
        self.Salinity.setGeometry(QtCore.QRect(30, 870, 141, 31))
        self.Salinity.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Salinity.setFrameShadow(QtWidgets.QFrame.Plain)
        self.Salinity.setSmallDecimalPoint(True)
        self.Salinity.setDigitCount(8)
        self.Salinity.setSegmentStyle(QtWidgets.QLCDNumber.Flat)
        self.Salinity.setObjectName("Salinity")
        self.Salinity.display(999)

        self.Conductivity = QtWidgets.QLCDNumber(self)
        self.Conductivity.setGeometry(QtCore.QRect(30, 950, 141, 31))
        self.Conductivity.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Conductivity.setFrameShadow(QtWidgets.QFrame.Plain)
        self.Conductivity.setSmallDecimalPoint(True)
        self.Conductivity.setDigitCount(8)
        self.Conductivity.setSegmentStyle(QtWidgets.QLCDNumber.Flat)
        self.Conductivity.setObjectName("Conductivity")
        self.Conductivity.display(999)

        self.Density = QtWidgets.QLCDNumber(self)
        self.Density.setGeometry(QtCore.QRect(30, 1030, 141, 31))
        self.Density.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Density.setFrameShadow(QtWidgets.QFrame.Plain)
        self.Density.setSmallDecimalPoint(True)
        self.Density.setDigitCount(8)
        self.Density.setSegmentStyle(QtWidgets.QLCDNumber.Flat)
        self.Density.setObjectName("Density")
        self.Density.display(999)

        self.Light_Value_Slider = QtWidgets.QSlider(self)
        self.Light_Value_Slider.setGeometry(QtCore.QRect(920, 710, 61, 201))
        self.Light_Value_Slider.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Light_Value_Slider.setOrientation(QtCore.Qt.Vertical)
        self.Light_Value_Slider.setTickPosition(QtWidgets.QSlider.TicksBelow)
        self.Light_Value_Slider.setObjectName("Light_Value_Slider")
        self.Light_Value_Slider.setMaximum(255)
        self.Light_Value_Slider.setMinimum(0)
        self.Light_Value_Slider.valueChanged.connect(self.userInteractLights)

        self.ResetButton = QtWidgets.QPushButton(self)
        self.ResetButton.setGeometry(QtCore.QRect(1080, 710, 100, 100))
        self.ResetButton.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ResetButton.setObjectName("ResetButton")
        self.ResetButton.setText("Reset\ninterlocked\nzones")
        self.ResetButton.pressed.connect(self.setReset)
        self.ResetButton.released.connect(self.releaseReset)
```

```python

        self.TakePhoto = QtWidgets.QPushButton(self)
        self.TakePhoto.setGeometry(QtCore.QRect(1740, 710, 171, 61))
        self.TakePhoto.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.TakePhoto.setObjectName("TakePhoto")
        self.TakePhoto.setText("Take HD photo")
        self.TakePhoto.pressed.connect(self.takePhoto)
        self.TakePhoto.released.connect(self.streamVideo)

        self.StartVideo = QtWidgets.QPushButton("toggle", self)
        self.StartVideo.setGeometry(QtCore.QRect(1740, 810, 171, 61))
        self.StartVideo.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.StartVideo.setObjectName("StartVideo")
        self.StartVideo.setText("Start video capture")
        self.StartVideo.setCheckable(True)
        self.StartVideo.clicked.connect(self.takeVideo)

        self.ScanMode20m = QtWidgets.QPushButton(self)
        self.ScanMode20m.setGeometry(QtCore.QRect(450, 710, 200, 71))
        self.ScanMode20m.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ScanMode20m.setObjectName("ScanMode20m")
        self.ScanMode20m.setText("Scanning [20m]")
        self.ScanMode20m.clicked.connect(lambda: self.userInteractModeSonar(0))

        self.ScanMode50m = QtWidgets.QPushButton(self)
        self.ScanMode50m.setGeometry(QtCore.QRect(450, 790, 200, 71))
        self.ScanMode50m.setStyleSheet("background-color: rgb(50, 205, 50);")
        self.ScanMode50m.setObjectName("ScanMode50m")
        self.ScanMode50m.setText("Scanning [50m]")
        self.ScanMode50m.clicked.connect(lambda: self.userInteractModeSonar(1))

        self.CollisionAvoid2m = QtWidgets.QPushButton(self)
        self.CollisionAvoid2m.setGeometry(QtCore.QRect(450, 870, 200, 71))
        self.CollisionAvoid2m.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.CollisionAvoid2m.setObjectName("CollisionAvoid2m")
        self.CollisionAvoid2m.setText("Collision Prevention [2m]")
        self.CollisionAvoid2m.clicked.connect(lambda: self.userInteractModeSonar(2))

        self.CollisionAvoid4m = QtWidgets.QPushButton(self)
        self.CollisionAvoid4m.setGeometry(QtCore.QRect(450, 950, 200, 71))
        self.CollisionAvoid4m.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.CollisionAvoid4m.setObjectName("CollisionAvoid4m")
        self.CollisionAvoid4m.setText("Collision Prevention [4m]")
        self.CollisionAvoid4m.clicked.connect(lambda: self.userInteractModeSonar(3))

        self.lblTemperature = QtWidgets.QLabel(self)
        self.lblTemperature.setGeometry(QtCore.QRect(30, 680, 141, 21))
        self.lblTemperature.setStyleSheet("\n""background-color: rgb(134, 134, 
134);")
        self.lblTemperature.setFrameShape(QtWidgets.QFrame.Panel)
        self.lblTemperature.setObjectName("lblTemperature")
        self.lblTemperature.setText("Temperature [°C]:")

        self.lblDepth = QtWidgets.QLabel(self)
        self.lblDepth.setGeometry(QtCore.QRect(30, 760, 141, 21))
        self.lblDepth.setStyleSheet("\n""background-color: rgb(134, 134, 134);")
        self.lblDepth.setFrameShape(QtWidgets.QFrame.Panel)
        self.lblDepth.setObjectName("lblDepth")
        self.lblDepth.setText("Depth [m]:")

```

```
        self.lblSalinity = QtWidgets.QLabel(self)
        self.lblSalinity.setGeometry(QtCore.QRect(30, 840, 141, 21))
        self.lblSalinity.setStyleSheet("\n""background-color: rgb(134, 134, 134);")
        self.lblSalinity.setFrameShape(QtWidgets.QFrame.Panel)
        self.lblSalinity.setObjectName("lblSalinity")
        self.lblSalinity.setText("Salinity [g/kg] PSU:")

        self.lblConductivity = QtWidgets.QLabel(self)
        self.lblConductivity.setGeometry(QtCore.QRect(30, 920, 141, 21))
        self.lblConductivity.setStyleSheet("\n""background-color: rgb(134, 134,
134);")
        self.lblConductivity.setFrameShape(QtWidgets.QFrame.Panel)
        self.lblConductivity.setObjectName("lblConductivity")
        self.lblConductivity.setText("Conductivity [mS/cm]:")

        self.lblDensity = QtWidgets.QLabel(self)
        self.lblDensity.setGeometry(QtCore.QRect(30, 1000, 141, 21))
        self.lblDensity.setStyleSheet("\n""background-color: rgb(134, 134, 134);")
        self.lblDensity.setFrameShape(QtWidgets.QFrame.Panel)
        self.lblDensity.setObjectName("lblDensity")
        self.lblDensity.setText("Water density [kg/m^3]:")

        self.lblLightIntensity = QtWidgets.QLabel(self)
        self.lblLightIntensity.setGeometry(QtCore.QRect(900, 680, 101, 21))
        self.lblLightIntensity.setStyleSheet("background-color: rgb(134, 134,
134);")
        self.lblLightIntensity.setFrameShape(QtWidgets.QFrame.Box)
        self.lblLightIntensity.setFrameShadow(QtWidgets.QFrame.Plain)
        self.lblLightIntensity.setObjectName("lblLightIntensity")
        self.lblLightIntensity.setText("Light intensity")

        self.lblControls = QtWidgets.QLabel(self)
        self.lblControls.setGeometry(QtCore.QRect(1255, 675, 100, 30))
        self.lblControls.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.lblControls.setFrameShape(QtWidgets.QFrame.Box)
        self.lblControls.setObjectName("lblControls")
        self.lblControls.setFont(QFont('Arial', 14))
        self.lblControls.setText("Controls")

        self.lblAlarms = QtWidgets.QLabel(self)
        self.lblAlarms.setGeometry(QtCore.QRect(1500, 675, 100, 30))
        self.lblAlarms.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.lblAlarms.setFrameShape(QtWidgets.QFrame.Box)
        self.lblAlarms.setObjectName("lblAlarms")
        self.lblAlarms.setFont(QFont('Arial', 14))
        self.lblAlarms.setText("Alarms")

        self.lblLeaks = QtWidgets.QLabel(self)
        self.lblLeaks.setGeometry(QtCore.QRect(1500, 730, 100, 60))
        self.lblLeaks.setStyleSheet("background-color: rgb(60, 179, 113);")
        self.lblLeaks.setObjectName("lblLeaks")
        self.lblLeaks.setText("   No detected\n       leaks")

        self.SonarMode = QtWidgets.QLabel(self)
        self.SonarMode.setGeometry(QtCore.QRect(440, 680, 211, 21))
        self.SonarMode.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.SonarMode.setFrameShape(QtWidgets.QFrame.Box)
        self.SonarMode.setObjectName("SonarMode")
        self.SonarMode.setText("Current Sonar Mode: ")

```

```python

        # Create new thread that handles video stream
        self.thread = VideoThread()
        # Connect thread to function that updates image
        self.thread.change_pixmap_signal.connect(self.update_image)
        # Start thread
        self.thread.start()


    def closeEvent(self, event):
        self.thread.stop()
        event.accept()

    def update_image(self, cv_img):
        qt_img = self.convert_video_qt(cv_img)
        self.image_label.setPixmap(qt_img)

    def update_sonar(self, cv_img):
        qt_img = self.convert_sonar_qt(cv_img)
        self.sonar.setPixmap(qt_img)

    def convert_video_qt(self, cv_img):
        rgb_image = cv2.cvtColor(cv_img, cv2.COLOR_BGR2RGB)
        h, w, ch = rgb_image.shape
        bytes_per_line = ch * w
        convert_to_Qt_format = QtGui.QImage(rgb_image.data, w, h, bytes_per_line, QtGui.QImage.Format_RGB888)
        p = convert_to_Qt_format.scaled(1920,960, Qt.KeepAspectRatio)
        return QPixmap.fromImage(p)

    def convert_sonar_qt(self, cv_img):
        rgb_image = cv2.cvtColor(cv_img, cv2.COLOR_BGR2RGB)
        h, w, ch = rgb_image.shape
        bytes_per_line = ch * w
        convert_to_Qt_format = QtGui.QImage(rgb_image.data, w, h, bytes_per_line, QtGui.QImage.Format_RGB888)
        return QPixmap.fromImage(convert_to_Qt_format)

    def setDisplayValues(self, temp, depth, leak, lockedZones,
    salinity, conductivity, density):
        self.Temp.display(temp)
        self.Depth.display(depth)
        self.Salinity.display(salinity)
        self.Conductivity.display(conductivity)
        self.displayLeakStatus(leak)
        self.density = density
        self.Density.display(self.density)
        self.configureZonesDisplay(lockedZones)

    def displayLeakStatus(self, leak):
        if leak:
            self.lblLeaks.setStyleSheet("background-color: rgb(255, 0, 0);")
            self.lblLeaks.setText("    DETECTED\n       LEAKS")

    def configureZonesDisplay(self, lockedZones):
        if lockedZones == self.prevLockedZones:
            return
        else:
            print("Changes in locked zones array discovered")
            for idx, zoneState in enumerate(lockedZones):
```

```
                if lockedZones[idx] != self.prevLockedZones[idx]:
                    if zoneState:
                        self.setLockedStatus(idx)
                    else:
                        self.resetLockedStatus(idx)
                    break
            self.prevLockedZones = lockedZones

    def setLockedStatus(self, zone):
        if zone == 0:
            self.Forward.setStyleSheet("background-color: rgb(200, 134, 134);")
        elif zone == 1:
            self.ForwardRight.setStyleSheet("background-color: rgb(200, 134, 134);")
        elif zone == 2:
            self.Right.setStyleSheet("background-color: rgb(200, 134, 134);")
        elif zone == 3:
            self.ReverseRight.setStyleSheet("background-color: rgb(200, 134, 134);")
        elif zone == 4:
            self.Reverse.setStyleSheet("background-color: rgb(200, 134, 134);")
        elif zone == 5:
            self.ReverseLeft.setStyleSheet("background-color: rgb(200, 134, 134);")
        elif zone == 6:
            self.Left.setStyleSheet("background-color: rgb(200, 134, 134);")
        elif zone == 7:
            self.ForwardLeft.setStyleSheet("background-color: rgb(200, 134, 134);")

    def resetLockedStatus(self, zone):
        if zone == 0:
            self.Forward.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif zone == 1:
            self.ForwardRight.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif zone == 2:
            self.Right.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif zone == 3:
            self.ReverseRight.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif zone == 4:
            self.Reverse.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif zone == 5:
            self.ReverseLeft.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif zone == 6:
            self.Left.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif zone == 7:
            self.ForwardLeft.setStyleSheet("background-color: rgb(134, 134, 134);")

    def resetAllLockedStatus(self):
        self.prevLockedZones = [False] * 8
        self.Forward.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ForwardRight.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Right.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ReverseRight.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Reverse.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ReverseLeft.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.Left.setStyleSheet("background-color: rgb(134, 134, 134);")
        self.ForwardLeft.setStyleSheet("background-color: rgb(134, 134, 134);")

    def release(self):
        config.runZone = -1
        config.newCommands = True

    def activateZone(self, zone):
```

```
        config.runZone = zone
        config.newCommands = True

    def userInteractModeSonar(self, mode):
        display_string = ""
        if mode == 0:
            display_string = "Scanning [20m]"
            self.ScanMode50m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.ScanMode20m.setStyleSheet("background-color: rgb(50, 205, 50);")
            self.CollisionAvoid2m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.CollisionAvoid4m.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif mode == 1:
            display_string = "Scanning [50m]"
            self.ScanMode50m.setStyleSheet("background-color: rgb(50, 205, 50);")
            self.ScanMode20m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.CollisionAvoid2m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.CollisionAvoid4m.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif mode == 2:
            display_string = "Collision [2m]"
            self.ScanMode50m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.ScanMode20m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.CollisionAvoid2m.setStyleSheet("background-color: rgb(50, 205, 50);")
            self.CollisionAvoid4m.setStyleSheet("background-color: rgb(134, 134, 134);")
        elif mode == 3:
            display_string = "Collision [4m]"
            self.ScanMode50m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.ScanMode20m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.CollisionAvoid2m.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.CollisionAvoid4m.setStyleSheet("background-color: rgb(50, 205, 50);")

        self.SonarMode.setText(f"Sonar Mode: {display_string}")
        config.mode = mode
        config.newCommands = True

    def userInteractSpeeds(self):
        config.motorSpeed = self.Motor_Speed_Slider.value()
        config.newCommands = True

    def userInteractLights(self):
        config.light = self.Light_Value_Slider.value()
        config.newCommands = True

    def setReset(self):
        self.resetAllLockedStatus()
        config.forceReset = True
        config.newCommands = True

    def releaseReset(self):
        config.forceReset = False
        config.newCommands = True

    def takePhoto(self):
```

```
        self.TakePhoto.setStyleSheet("background-color: rgb(50, 205, 50);")
        config.takePhoto = True
        config.newCommands = True

    def takeVideo(self):
        if config.takeVideo == False:
            self.StartVideo.setStyleSheet("background-color: rgb(50, 205, 50);")
            self.StartVideo.setText("Capturing video...")
            config.takeVideo = True
        else:
            self.StartVideo.setStyleSheet("background-color: rgb(134, 134, 134);")
            self.StartVideo.setText("Start video capture")
            config.takeVideo = False

        config.newCommands = True

    def streamVideo(self):
        self.TakePhoto.setStyleSheet("background-color: rgb(134, 134, 134);")
        config.takePhoto = False
        config.newCommands = True
```

Appendix I

Meeting invitations

Agenda

Periodic meeting between project and control group

Date:	25.01.2022
Time:	09:15 – 09:45
Place:	Teams
Participants:	Tony Paulsen Petter Henriksen Ottar L. Osen Lars Christian Gansel

Item 1	Is the equipment list as listed in the pre-project report approved?
Item 2	Discuss project-groups proposed solution as defined in pre-project report. - Anything missing? - Part-tasks listed that is not relevant? - Any extra functionality the control group wants?

Agenda

Periodic meeting between project and control group

Date:	08.02.2022
Time:	09:15 – 10:00
Place:	Teams
Participants:	Tony Paulsen Petter Henriksen Ottar L. Osen Lars Christian Gansel

Item 1	Access to software from ROV 2017
Item 2	Discuss solution on communication/ power supply in tether cable
Item 3	Proposed camera solution discussion
Item 4	Conductivity sensor solution discusion

Agenda

Periodic meeting between project and control group

Date:	23.02.2022
Time:	11:00 – 11:30
Place:	Teams
Participants:	Tony Paulsen Petter Henriksen Ottar L. Osen Lars Christian Gansel

Item 1	General progress report discussion
Item 2	Cables for Aanderaa sensor(s)
Item 3	Camera order update
Item 4	Lazercutting / fusion360 tips/course discussion

Agenda

Periodic meeting between project and control group

Date:	08.03.2022
Time:	09:15 – 10:00
Place:	Zoom
Participants:	Tony Paulsen Petter Henriksen Ottar L. Osen Lars Christian Gansel

Item 1	General progress
Item 2	Discuss ordering waterproof cable(s) for conductance, turbidity, and oxygen sensors.
Item 3	Communication from ROV to surface GUI in tether

Agenda

Periodic meeting between project and control group

Date: 22.03.2022

Time: 15:25-16:00

Place: Teams

Participants: Tony Paulsen
Petter Henriksen
Ottar L. Osen
Lars Christian Gansel

Item 1	General progress, discuss progress report
Item 2	Camera - Currently using school camera for testing
Item 3	Conductance sensor progress
Item 4	Tether communication and power supply - Using BlueRobotics fathom tether X - Power converter DC-DC, 48V-12V

Agenda

Periodic meeting between project and control group

Date:	07.04.2022
Time:	13:00 – 13:30
Place:	Teams
Participants:	Tony Paulsen Petter Henriksen Ottar L. Osen Lars Christian Gansel

Item 1	General progress discussion
Item 2	Important contents for report
Item 3	Framerate camera
Item 4	Increasing program execution measures

Agenda

Periodic meeting between project and control group

Date:	22.04.2022
Time:	13:00 – 13:30
Place:	Teams
Participants:	Tony Paulsen Petter Henriksen Ottar L. Osen Lars Christian Gansel

Item 1	General progress update
Item 2	Electrical connections - Converter solution - Placement of Regulators and Converter - Setup of the power delivery

Agenda

Periodic meeting between project and control group

Date:	06.05.2022
Time:	12:00 – 12:30
Place:	Tunglab
Participants:	Tony Paulsen Petter Henriksen Ottar L. Osen Lars Christian Gansel

Item 1	General progress update
Item 2	Priorites forward
Item 3	Battery solution limitations

Appendix J

Minutes of meeting

Minutes of meeting

Meeting between Project group and Supervisors bachelor ROV		
Length: 30min	Date: 25.01.2022	Start time: 09:15

Meeting location:	Zoom
Meeting called by:	Tony Paulsen
Meeting type:	Progress report with Supervisors
Meeting led by:	Tony Paulsen
Secretary:	Petter Henriksen
Time responsibility:	Petter Henriksen
Participants:	Tony Paulsen, Petter Henriksen, Ottar L. Osen, Lars Christian Gansel

Agenda number	Agenda	Discussed
1	Discussion around materials list from pre-project report	Aanderaa sensor, camera og combination sensor was discussed. It was also mentioned that the ROV is going to be built with all new parts. The group also need research and select an umbilical cable. Special parts need to be ordered as fast as possible so that it doesn’t cause delays.
2	Discuss the proposed solutions that were defined in the pre-project report	The examples around implementation of logic were approved. They were considered as relevant functions for the system. For now, the group is planning to use the sonar for collision avoidance. The alarm for big changes in measured values can be done with python. If alternative solutions are found, they will be considered and compared to the current solutions.

Minutes of meeting

Meeting between Project group and Supervisors bachelor ROV		
Length: 35min	Date: 08.02.2022	Start time: 09:15

Meeting location:	Teams
Meeting called by:	Tony Paulsen
Meeting type:	Progress report with Supervisors
Meeting led by:	Tony Paulsen
Secretary:	Petter Henriksen
Time responsibility:	Petter Henriksen
Participants:	Tony Paulsen, Petter Henriksen, Ottar L. Osen, Lars Christian Gansel

Agenda number	Agenda	Discussed
1	Access to programs from earlier projects	The group asked if earlier work on the previous ROV was available specifically code for the propulsion system. Ottar is going to check but the project finished a long time ago so its highly likely it won't be found. From the reports we have a lot of info.
2	Discuss communication/power delivery	It was decided that the main plan for the tether is to use 2 pairs for communication and 2 pairs for power delivery. A few other solutions were mentioned like sending the communication on top of the power. The group is also going to look at previous work on the Towed ROV who have had to overcome similar problems when it comes to communication and power delivery.
3	Camera solution	It was decided that the camera from FLIR ticked all the boxes. The lenses were also agreed

		on. Something the group must take into consideration is reflection inside the acrylic dome.
4	Conductivity sensor	The sensor from Aanderaa was deemed too expensive so the group will borrow one instead. If it's necessary ordering a new one can be considered. The supervisors also wanted the group to make swapping sensors easy so that the ROV is more modular.

Minutes of meeting

Meeting between Project group and Supervisors bachelor ROV		
Length: 30min	Date: 23.02.2022	Start time: 11:00

Meeting location:	Teams
Meeting called by:	Tony Paulsen
Meeting type:	Progress report with Supervisors
Meeting led by:	Tony Paulsen
Secretary:	Petter Henriksen
Time responsibility:	Petter Henriksen
Participants:	Tony Paulsen, Petter Henriksen, Ottar L. Osen

Agenda number	Agenda	Discussed
1	General Progress	The group were told to make progress reports and to update the Gantt diagram more frequently.
2	Aanderaa cable	There was some discussion regarding cables from Aanderaa since they are quite expensive. No decision was made but it was concluded that the work could continue and that a decision could wait until the next meeting.
3	Camera update	Waiting on order confirmation from supplier.
4	Laser cutting	Ottar gave tips on how to make and design mounts and gave suggestions on gluing the acrylic together.

Minutes of meeting

Meeting between Project group and Supervisors bachelor ROV		
Length: 45min	Date: 08.03.2022	Start time: 09:15

Meeting location:	Zoom
Meeting called by:	Tony Paulsen
Meeting type:	Progress report with Supervisors
Meeting led by:	Tony Paulsen
Secretary:	Petter Henriksen
Time responsibility:	Petter Henriksen
Participants:	Tony Paulsen, Petter Henriksen, Ottar L. Osen, Lars Christian Gansel

Agenda number	Agenda	Discussed
1	General progress	The group informed the supervisors on the progress made during the previous period.
2	Discuss ordering waterproof cable(s) for conductance, turbidity and oxygen sensors.	The group were told to get more information from Aanderaa before a decision is made.
3	Communication from ROV to surface GUI in tether	Tips for establishing communication between GUI and ROV were given. The group was also told to look at other similar projects for more solutions.

Minutes of meeting

Meeting between Project group and Supervisors bachelor ROV		
Length: 52min	Date: 22.03.22	Start time: 15:25

Meeting location:	Teams
Meeting called by:	Tony Paulsen
Meeting type:	Progress report with Supervisors
Meeting led by:	Tony Paulsen
Secretary:	Petter Henriksen
Time responsibility:	Petter Henriksen
Participants:	Tony Paulsen, Petter Henriksen, Ottar L. Osen, Lars Christian Gansel

Agenda number	Agenda	Discussed
1	Camera situation	Due to long shipping time the group has gotten to borrow an unused camera from another project. It was decided that due to the time constraint that this borrowed camera will be used instead so that the group will be able to finish the project within the given time frame.
2	Power and communication	Based on recommendations from Ottar the group checked out the report from “Slepe ROV” and found out how they were able to power it and have communication. This was done by using a product from BlueRobotics called “Fathom-X Tether”. This product allows use of a single pair in the CAT5 cable for data transfers up to 80mb/s. This frees up the 3 other pairs to be used for power delivery. It was decided that this would be used on this project.

Minutes of meeting

Meeting between Project group and Supervisors bachelor ROV		
Length: 35min	Date: 07.04.22	Start time: 15:00

Meeting location:	Teams
Meeting called by:	Tony Paulsen
Meeting type:	Progress report with Supervisors
Meeting led by:	Tony Paulsen
Secretary:	Petter Henriksen
Time responsibility:	Petter Henriksen
Participants:	Tony Paulsen, Petter Henriksen, Ottar L. Osen, Lars Christian Gansel

Agenda number	Agenda	Discussed
1	General Progress	The group gave a quick summary of the progress on the project and what had been done this period. Areas where the group could test the ROV in the ocean were given. Two areas were mentioned either take a boat to the fish farms or the floating jetty at Sunnmøre Museum.
2	Framerate camera	The camera feed from the raspberry was shown to supervisors and a discussion regarding how the group could improve the framerate and resolution. A couple of different methods were mentioned like sending the feed as a video with Keyframes(eframes) instead of sending every frame as a photo. Another method that was mentioned was sending the camera data separately from everything else and with a different protocol and speeds. Something else that was mentioned regarding the camera was implementing features allowing the user to select the fps and resolution. Another feature was a record button and a picture button that would take a full resolution picture and save it when clicked.

3	Program execution measures	The group asked about tips regarding how to make the program run better and what data should be prioritised. A few things were mentioned like splitting all the processes into individual threads and to then setup these with different levels of importance. Another thing that was mentioned was to not send commands all the time and to setup one server with multiple clients

Minutes of meeting

Meeting between Project group and Supervisors bachelor ROV		
Length: 45min	Date: 22.04.22	Start time: 13:00

Meeting location:	L044
Meeting called by:	Tony Paulsen
Meeting type:	Progress report with Supervisors
Meeting led by:	Tony Paulsen
Secretary:	Petter Henriksen
Time responsibility:	Petter Henriksen
Participants:	Tony Paulsen, Petter Henriksen, Ottar L. Osen, Lars Christian Gansel

Agenda number	Agenda	Discussed
1	General Progress	The group gave a quick summary of the progress on the project and what had been done this period. The group showed off how the ROV looked, and the supervisors suggested changing the position of the lights to improve the performance of the lights. The supervisors also mentioned that adding a compass would be beneficial.
2	Electrical connections	The planned circuit diagram for the power delivery system was shown to the advisors and feedback was given. This system includes voltage regulators and the DC-to-DC converter. It was also decided that these components will be in their own box that will be filled with epoxy.

Minutes of meeting

Meeting between Project group and Supervisors bachelor ROV		
Length: 30min	Date: 06.05.22	Start time: 12:00

Meeting location:	L044
Meeting called by:	Tony Paulsen
Meeting type:	Progress report with Supervisors
Meeting led by:	Tony Paulsen
Secretary:	Petter Henriksen
Time responsibility:	Petter Henriksen
Participants:	Tony Paulsen, Petter Henriksen, Ottar L. Osen, Lars Christian Gansel

Agenda number	Agenda	Discussed
1	General Progress	The group gave a quick summary of the progress on the project and what had been done this period. The group showed of how the the current state of the ROV and updated the supervisors on the results of the first test of the whole system.
2	Priorities going forward	The groups plan for the final weeks of the project were discussed. This plan includes getting at least one test in the water tank with as many systems as possible in working order.

Kunnskap for en bedre verden