NTNU

Kunnskap for en bedre verden

Department of Electronic Systems

Underwater Wildlife Monitoring Stations

Bachelor's Thesis

Authors:
Erjok Aguto
Torstein Kristiansen
Henrik Malkenes
Filip Ræder

May 21st, 2024

Thesis Title: Underwater Wildlife Monitoring Stations	
Authors: Erjok Aguto, Torstein Kristiansen, Henrik Malkenes, Filip Ræder	**Project Number:** E2414
	Due Date: 21.05.2023
Study Program: Bachelor in Electrical Engineering	**Grade:**
Field of Study: Automation and Robotics, Electronics and Sensor Systems	[X] Open [ ] Closed
Internal Supervisor: Terje Mathiesen	**Department:** Department of Electronic Systems
Client: Department of Engineering Cybernetics	**Contact Person:** Damiano Varagnolo damiano.varagnolo@ntnu.no +47 481 28 922

Abstract:

With our planet ever changing, preserving our ecosystems is a vital part of preserving our health, and ensuring the sustainability of nature. By being able to monitor the ocean and its wildlife, scientist get a clearer vision of how the ecosystems are changing, and how we can preserve them. The Underwater Wildlife Monitoring Stations provide a versatile way of acquiring and transmitting data that is vital to this cause. By further expanding our knowledge of the ocean and its ecosystems, preserving them becomes a challenge that we might just be able to overcome.

Sammendrag:

I en verden som stadig er i endring, er det avgjørende å bevare våre økosystemer for å opprettholde god helse og sikre naturens fremtid. Gjennom å overvåke havene og deres dyreliv får forskere verdifull innsikt i økosystemenes dynamikk og strategier for deres bevaring. Undervannsstasjoner for overvåking gir en effektiv metode for å samle og formidle viktig data til dette formålet. Ved å utdype vår forståelse av havet og dets økosystemer, vil vi være bedre rustet til å bevare økosystemene.

Keywords:	**Stikkord:**
Underwater technology, Acoustic communication, ROS2, Real-Time Data collection, 3D modeling, 3D Printing and Computer Vision	Undervannsteknologi, Akustisk kommunikasjon, ROS2, Sanntidsdatainnhenting, 3D modellering, 3D Printing og Robotsyn

Abstract

Our planet is ever changing, and human impact affects ecosystems all around the globe. To ensure the health of coastal communities, oceanic ecosystems and economies, we need to keep track of oceans- and coastal areas. By assessing sensor data readings, aswell as monitoring wildlife, surveying the oceans will help scientist get a better understanding of how the ecosystems are changing and how we can positively affect them. (NOS 2024)
The Underwater Wildlife Monitoring Stations, hereby referred to as 'Sensing rigs', are designed to monitor and transmit data and image wirelessly through mounted Acoustic Modems. With each rig having mounted modems, this enables establishing of a network of monitoring stations. Facilitating data for analysis helps gain insights into the development of ecosystems. The versatility of the sensing rigs also allows for integration with remotely operated vehicles, such as ROVs, USVs and AUVs.

This years project is a continuation of earlier bachelors, where the rigs "demonstrated successful sensor readings and data processing abilities". (Knudsen and Brandstorp 2023)
Further development from last years sensing rigs can be summarized to the following:

- Implementation of Camera module for wildlife monitoring
  - Expanded the existing enclosure to ensure proper use of camera
  - Implemented Computer Vision algorithm
- Mounted an RTC and subsequent module for Real-Time Data Collection
- Upgrading single-board computer and subsequent software implementation
- Optimizing the sesning rigs
- Added autoshutdown feature when battery is low

Throughout the project, the group made sure to document both progress and problems encountered. This documentation ensured a strong foundation for the thesis as a user guide for future work. A final test was carried out to demonstrate the system's progression as well as its durability under water.

Sammendrag

Planeten vår er i konstant endring, og menneskelig påvirkning har store konsekvenser for økosystemer over hele kloden. For å sikre bærekraftige kystsamfunn, bevare havets økosystemer og støtte bærekraftig økonomi er det avgjørende å kunne overvåke hav- og kystområder. Ved å analysere sensordata og bilder kan forskere få en bedre forståelse av hvordan økosystemene endres, og hva vi kan gjøre for å hjelpe dem. (NOS 2024) Underwater Wildlife Monitoring Station, som vi herfra vil refere til som 'Sensing rigs', er designet for å overvåke og sende sensordata og bilder trådløst via de Akustiske Modemene som er montert på riggen. Ved at hver rig har et slik modem, kan en danne et nettverk av slike overvåkningsstasjoner, som sørger for data som igjen kan analyseres slik at en får mer innsikt i utviklingen av økosystemer i havet. Allsidigheten til riggene gjør også at de kan integreres med andre proejskter, deriblant fjernstyrte båter og undervannsdroner.

Årets prosjekt er en videreføring av tidligere Bacheloroppgaver, og tar utgangspunkt i fjorårets oppgave der riggene hadde "vellykkede sensoravlesninger og datahåndteringsevner". (Knudsen and Brandstorp 2023) Gruppens bidrag til videre utvikling av sensorriggene kan oppsummeres til følgende:

- Implementering av kameramodul for å overvåke marint liv
  - Utvidet de eksisterende riggene for bruk av kamera
  - Implementere robotsynsalgoritme
- Monterte en RTC og RTC-modul i systemet for å sikre sanntidsdatainnhenting
- Oppgraderte til en kraftigere mikrochip computer og tilpasset nødvendig software
- Optimalisering av riggene
- Lagt til autoshutdown funksjon når batteri er lavt

Underveis i arbeidet av prosjektet sørget gruppen for å dokumentere alt av både framgang og problemer som ble støtt på. God dokumentasjon underveis i arbeidet sørget for et godt grunnlag når det kom til å skrive selve oppgaven, da det ble formidlet et ønske om at den skulle fungere som en brukergruide for fremtidig arbeid. Prosjektet ble avsluttet med en siste test, der robustheten til systemet i realistiske omgivenheter, samt årets progresjon ble demonstrert.

Acknowledgements

We would like to express our gratitude towards those who has provided support, guidance and assistance during this iteration of the project. Primarily, we thank our client and supervisor, Damiono Varagnolo for his guidance through our weekly meetings and contributions through email throughout this project. A huge thanks goes out to the remaining supervisory staff for their valuable support, feedback and contributions to this project. This thanks goes to Sivert Knudsen, Behdad Aminian, Ovidijus Cironka, Simon Hagen Hoff and Terje Mathiesen.

Finally, we want to direct our attention towards the department of Electronic Systems for providing resources and funding, and towards the mechanical workshop for their collaborative work in designing and manufacturing the camera housing. A special thanks to Ivan Ræder for helping with- and providing a 3D-printer we could use during the project.

This project would not have been possible without the invaluable contributions of these individuals and institutions. Thank you all.

Norwegian University of Science and Technology
Trondheim, May 2024

[illegible] [illegible] [illegible] [illegible]

Erjok Aguto **Torstein Kristiansen** **Filip Ræder** **Henrik Malkenes**

Table of Contents

List of Figures

1 Introduction

1.1 Background

Underwater ecosystems include a wide variety of habitats, from sand dunes to coral reefs. These environments play a crucial role in supporting global food security, regulating climate patterns, and sustaining livelihoods for millions of people worldwide. However, despite their ecological significance, underwater ecosystems face rising threats from human activities, such as over fishing, habitat destruction, pollution, and climate change.

Being able to properly monitor these areas serve multiple purposes. It enables scientists to gather data on key environmental indicators, allowing for a more comprehensive understanding of underwater ecosystems and their dynamics. By continuously monitoring these parameters, researchers can detect changes over time, identify trends, and assess the impacts of human activities and natural events on marine ecosystems.

Monitoring and understanding underwater ecosystems face challenges due to the nature of working in aquatic environments. Traditional monitoring methods, such as diver surveys and remote sensing, offer valuable insights but are often limited in regards to area and ability to provide real-time data. This highlights the need for innovative technologies that can overcome these limitations and provide detailed, continuous monitoring of underwater environments.

Underwater monitoring stations with embedded sensors represent a promising solution to this challenge. These stations are equipped with an array of sensors capable of measuring various environmental parameters in real-time. These parameters include but are not limited to: water temperature, salinity, dissolved oxygen concentrations and pressure.

Moreover, underwater monitoring stations can play a critical role in wildlife conservation. By tracking the movements and behaviors of marine life, these stations provide valuable insights into migration patterns, habitat preferences, breeding behaviors, and population dynamics. This data is crucial for developing effective conservation strategies, creating marine protected areas, and reducing the impacts of human-caused stressors on vulnerable species.

(NOS 2024)

1.2 Sustainable Development Goals

The United Nations SDGs provide the framework for addressing global challenges. By having these in mind when developing new technology and planning, we aim to improve the steadily declining trend of global environmental degradation and social inequality. Having healthy and balanced ecosystems in oceans and lakes correlate directly to several of these goals.

(a) Zero Hunger (b) Climate Change (c) Life Below Water

Figure 1: Sustainable Development Goals (*Sustainable Development Goals* 2024)

Goal 2: Zero Hunger
Goal 2 is about creating a world free of hunger by 2030. Underwater ecosystems contribute to fulfilling this goal by supporting fisheries as well as other marine resources and seafood. By ensuring the health and availability of nutritious seafood, we are one step closer to achieving food security.

Goal 13: Climate Action
Goal 13 is about taking action against climate change and its impacts. Being able to monitor underwater ecosystems is crucial for understanding the impacts of climate change. By tracking changes, one can contribute to climate mitigation and adaptation strategies.

Goal 14: Life below water
Goal 14 is about conserving and sustainably using the oceans, seas and marine resources. Underwater ecosystems are integral to conserving and sustainably using marine resources. By being able to monitor these ecosystems, one can directly affect the trajectory of marine life and resources. (*Sustainable Development Goals* 2024)

1.3 Overview

The sensing rig consists of two cylinders and a modem mounted on a metal plate. The basic principle of the system is that the underwater rig will collect sensor data and images. This will be transferred to the mounted acoustic modem via a SubConn cable. From here, the data is transmitted with acoustic signals to a modem on the surface where the data can be collected or further transmitted. The system is powered by a rechargeable battery. While the current project focuses on data and image collection as well as transmission, there are possibilities for future expansion, such as creating a network of rigs communicating with each other, to enhance monitoring and research capabilities in underwater environments.

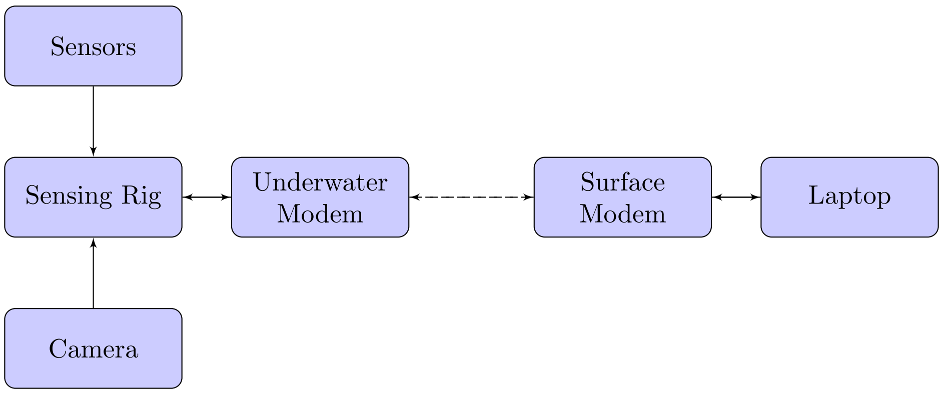


Figure 2: Flow chart of the basic system

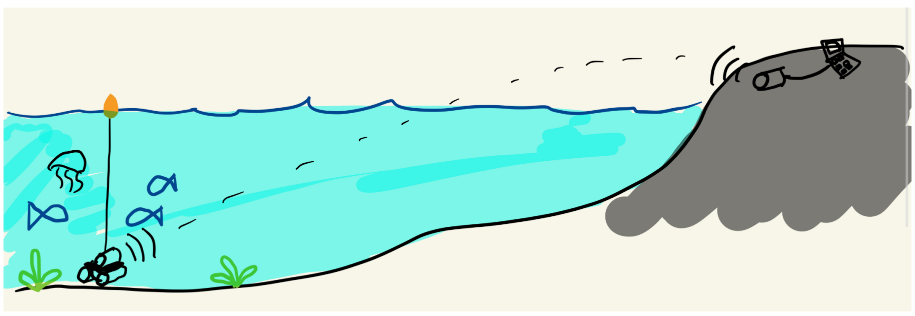


Figure 3: An illustration of how the system would be deployed

1.4 Prerequisites

The thesis is targeted at other electrical engineering students. The reader of the thesis is expected to have a basic understanding of electronics and circuits (i.e Ohm's Law). Without a basic understanding of electronics, diagrams and tables such as the circuit diagram and Pinout tables could be difficult to comprehend. In depth knowledge of specific fields is not expected of the reader, and non-trivial concepts that are relevant to the project will be explained in the theory section of each chapter. The software of the project is written mainly in Python, and the computers are running a headless Ubuntu-OS. All work is done through a terminal so basic commands for navigating workspaces should be familiar before reading the software part.

1.5 Technical Definitions

- **SDG** - Sustainable Development Goal
- **ROS** - Robot Operating Software
- **ROV** - Remotely Operated Vehicle
- **USV** - Unmanned Surface Vehicle
- **AUV** - Autonomous Underwater Vehicle
- **RTC** - Real Time Clock
- **SSH** - Secure Shell, is a protocol to wirelessly access the terminal of another computer with encrypted traffic
- **Yolov8** - You Only Look Once. A computer vision algorithm trained with AI and used for image recognition.
- **CAD** - Computer-Aided Design is using computer-based software to design and draw in both 2D and 3D
- **I2C** - Inter-Integrated Circuit, a serial communication protocol for data transfer through wires
- **Siemens** - Unit of electrical conductance.
- **LiPo** - Lithium Polymer. Describes the type of battery.
- **SBC** - Single-Board Computer
- **RAM** - Random-Access Memory
- **Rpi** - Raspberry Pi
- **CPU** - Central Processing Unit
- **GPU** - Graphics Processing Unit
- **NPU** - Neural Processing Unit
- **PA** - Project Administration. Folder for all files regarding Bachelor's project.
- **SoC** - System on Chip
- **NMCLI** - Network Manager Command-line Interface

1.6 Structure and Literature review

The client wanted the thesis to be structured like a user guide for future users. Therefore, instead of having main sections for theory, previous work, current state, and future work, we organized the thesis into chapters. Each chapter has its own subsections for these topics. There are 4 main chapters all with the same structure:

- **Main chapter**
  - Theory
  - Previous Work
  - Current State
  - Further Work

By writing each chapter independently, the reader will not need to read through all sections to gain an understanding of a specific topic. This approach makes the thesis more fluid and easier to navigate for the reader. At the end, the chapters will be tied together, and the conclusion will describe the entire system.

1.7 System and Objectives

The sensing rigs are composite devices capable of measuring and transferring underwater sensor data via an acoustic modem. The group took over the project at a stage where the sensing rigs could successfully read sensor values and establish communication with multiple modems.

The goal of this project is to further develop the sensing rig through implementation of an RTC module, a submarine camera with computer vision for recognizing marine wildlife and general upgrades where the group identifies opportunities for improvement. This requires a redesign of the enclosure and electronics to facilitate for the new devices.

2 Electronics

The electronics of the sensing rig play a fundamental role in the development of an underwater monitoring station. The primary objective in the development of hardware is to accommodate both a camera and computer vision algorithms. Achieving this will require a thorough selection of components. The ultimate goal is for the electronic components to collect various types of data, process it, and then transmit the data to the user via the acoustic modem.
The circuit diagram showcases relevant electrical connections, and can be freely edited for future additions using the links:

- *Circuit Diagram Raspberry Pi*

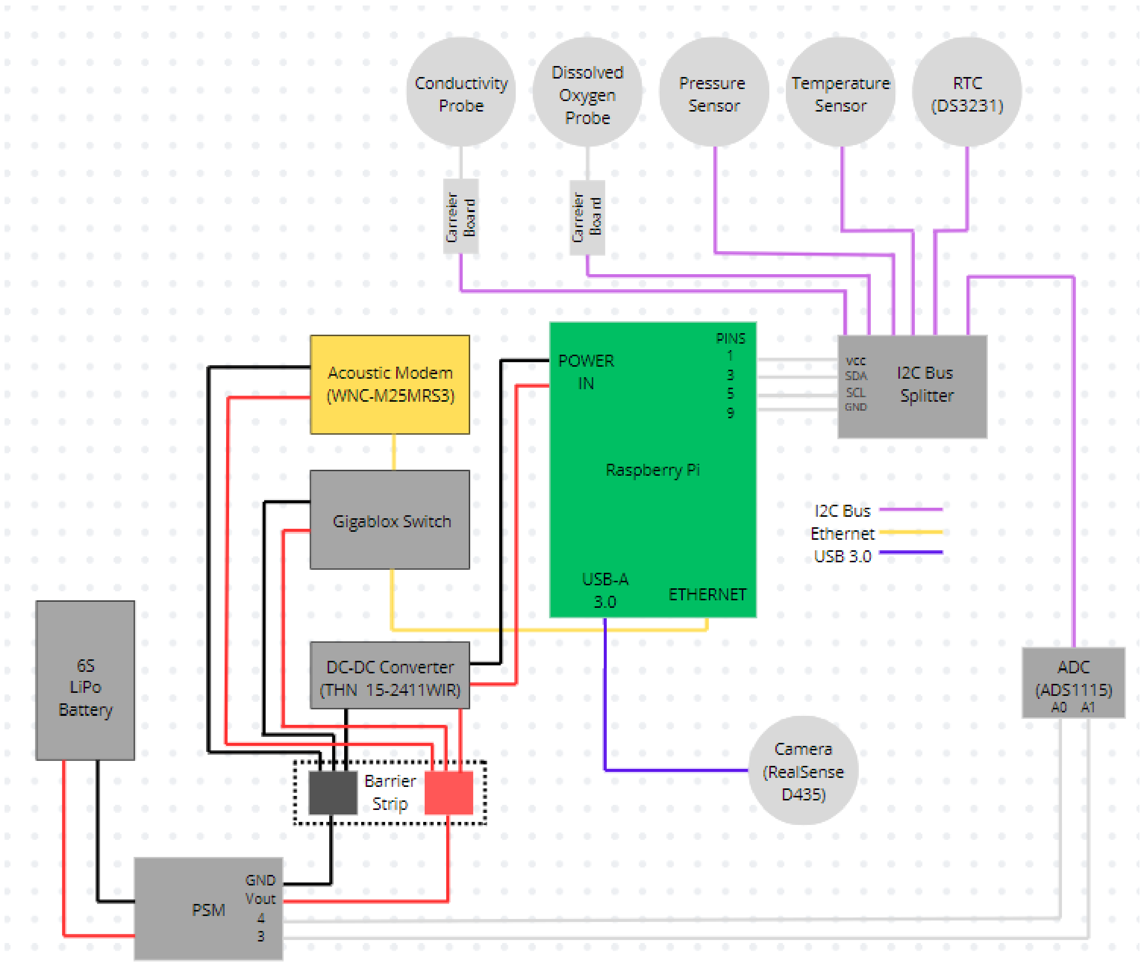


Figure 4: Circuit Diagram for the rigs using Raspberry Pi

- *Circuit Diagram Khadas VIM*

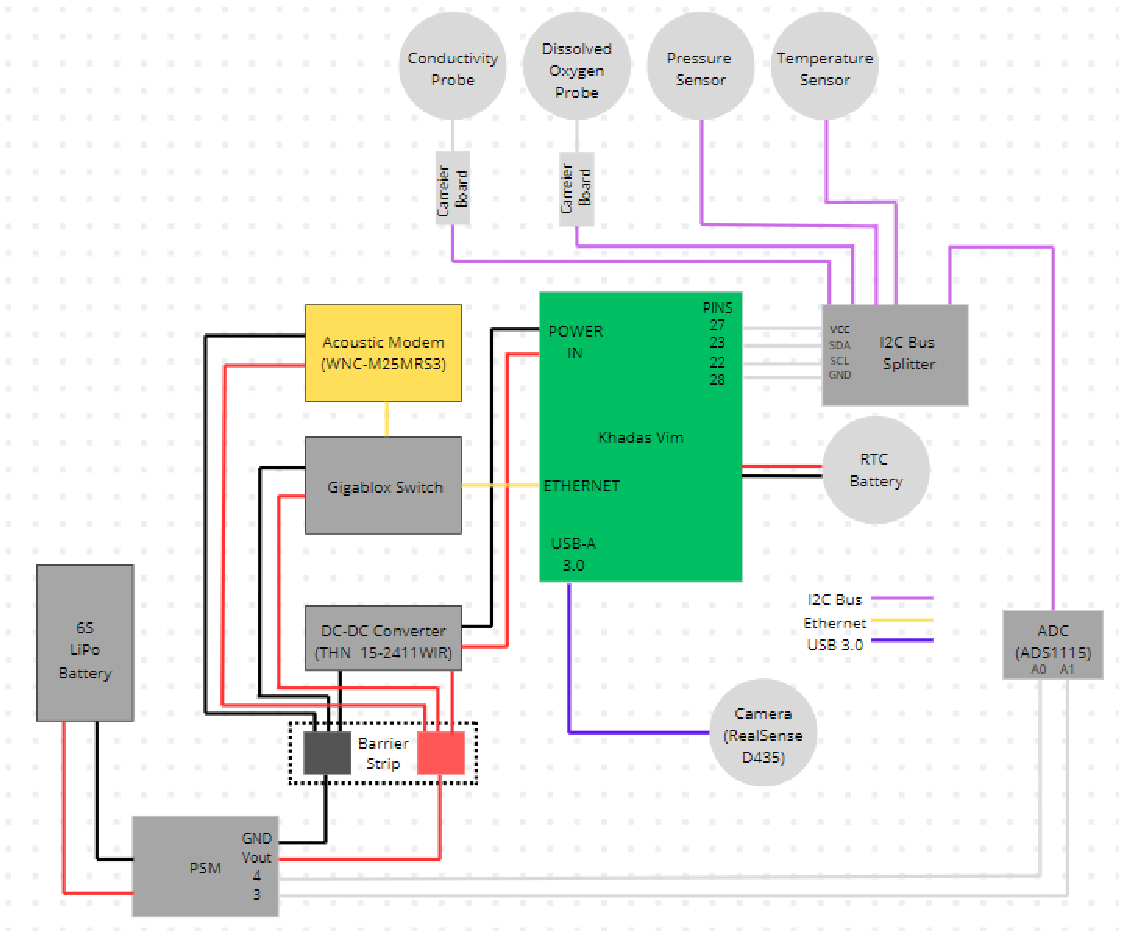


Figure 5: Circuit Diagram for the rigs using Khadas VIM

2.1 Theory

Figure 4 and figure 5 helps illustrate how the different sensors interact with each other. In this section, all the different sensors and components, as well as their use in the project will be briefly explained. The goal of this section is not just to provide raw facts on each component, but also provide a foundation for understanding why the components are implemented. In particular, the components that are new to the sensing rigs in this years project will be studied deeper, to make for a more comprehensive understanding of each decision made throughout the project.

2.1.1 Sensors

Pressure sensor

The sensor consists of a pressure-sensitive piezoelectric crystal, that deforms in response to changes in pressure. This deformation generates an electrical signal proportional to the pressure applied. The pressure sensor can measure pressure up to 30 bar (300m). The pressure sensor detects changes in pressure both as the rig descends deeper, but also when the oceans currents and tidal patterns change. Integration of the sensor gives the system deeper insights into the distribution of monitored wildlife, with respect to both depth, currents and tidal flow. (*Pressure Sensor* 2024)

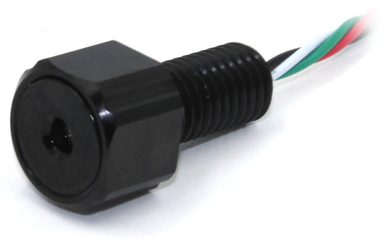


Figure 6: Pressure sensor (*Pressure Sensor* 2024)

Temperature sensor
The temperature sensor is used to make temperature profiles. As the temperature changes, the resistance of the thermistor changes accordingly, allowing the sensor to measure the temperature, and converting it to a 24 bit ADC value. The sensor is accurate to ± 0.1°C. These temperature profiles helps with understanding marine life habits as shifts in temperature occur. Adding the sensor to the rig improves the effectiveness of our wildlife monitoring station. (*Temperature Sensor* 2024

Figure 7: Temperature sensor (*Temperature Sensor* 2024

Conductivity sensor
The conductivity sensor indicates the waters ability to conduct electricity, by assessing factors such as salinity and other dissolved solids in the water. The sensor consists of two electrodes that are submerged in the water. When the current flows between these electrodes, it encounters resistance due to the ions present in the water, which affects the conductivity. The sensor measures the electrical impedance or resistance caused by these ions and converts it into a conductivity

measurement. Changes in conductivity levels indicate water quality and thus helps with evaluating habitats for marine life. The sensor has a conductivity range of [10-100,000] µS/cm (Microsiemens per centimetre). (*Conductivity Kit* 2024)

Figure 8: Conductivity probe (*Conductivity Kit* 2024)

Dissolved Oxygen sensor
This sensor detects the amount of oxygen dissolved in the water, vital for supporting aquatic life. As the rig explores diverse areas, the sensor measures variations in dissolved oxygen levels, offering insights into water quality and ecosystem health. Low levels of dissolved oxygen can signal poor water quality and stress on marine organisms. By monitoring dissolved oxygen levels, we can pinpoint areas with insufficient oxygen supply and implement necessary conservation measures. (*Dissolved Oxygen Kit* 2024)

Figure 9: Dissolved Oxygen Probe (*Dissolved Oxygen Kit* 2024)

Power Sense Module
The Power Sense Module provides analog readings of current and voltage from the battery. It has a maximum input voltage of 25,2 V and unlike other power sense modules, this PSM from BlueRobotics will not provide 5V power supply to the circuit. It has an significant accuracy at low current draw due to the use of a hall effect current sensor. The PSM is excellent for monitoring the battery condition. (*Power sense module* 2024)

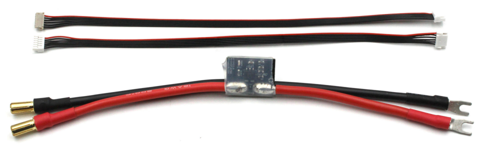


Figure 10: Power Sense Module (*Power sense module* 2024)

2.1.2 System Integration Components

I2C Bus Splitter
The I2C bus splitter makes it possible to connect multiple I2C devices by sharing the same bus. It features two different header pin configurations, including DF13- and JST-GH connectors for great compatibility with various devices. The I2C bus splitter establishes seamless communication pathways by serving as the intermediary between the different sensors and the SBC. (*I2C Bus Splitter* 2024)

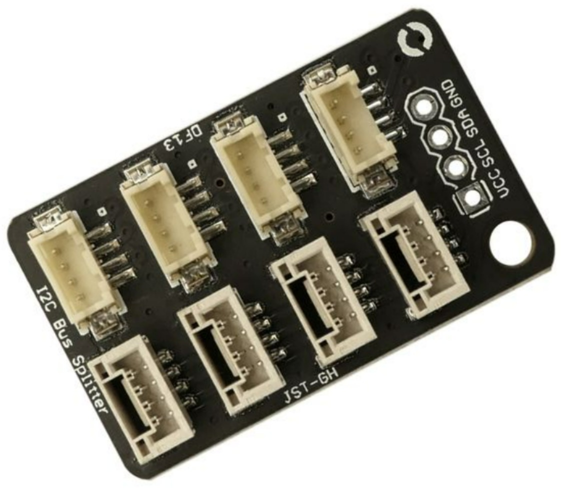


Figure 11: I2C splitter (*I2C Bus Splitter* 2024)

DC-DC Converter
The DC-DC converter works by transforming a high DC voltage down to a low, more managerial DC voltage. The model in use is the THN 15-2411WIR manufactured by Traco Power which is used to reduce the battery voltage of 25 V to a more suitable 5 volts DC, that powers both the SBC and sensors within the system. This converter can accommodate a wide range of input voltages,

spanning from 9 volts up to 36 volts DC. Despite this variability, it maintains an efficiency of 89 percent, ensuring minimal energy loss during the conversion process. Additionally, the converter is equipped with a build-in short circuit protection, safeguarding the system from potential damage caused by electrical faults. (Power 2022)

Figure 12: DC-DC Converter Power 2022)

Analog-to-Digital Converter
An analog-to-digital converter(ADC) enables digital circuits like SBCs and micro-controllers to communicate with the real world. The ADC functions by sampling an analogue voltage at one moment in time and then converting it to a digital code using binary bits to represent the voltage level. The resolution of the ADC is determined by the number of bits utilized to represent the voltage. (*Analogue to Digital Converter* 2024)

The model utilized is the ADS1115, manufactured by Adafruit. With a resolution of 16 bit and a sample rate of 860 samples per second, this device is great at converting the analog PSM readings into a digital format. Moreover, it is equipped with four single-ended input channels that can also function as two differential inputs if required, improving its versatility. Notable features includes low current consumption, a wide operating voltage range from 2 to 5,5 volts, and compatibility with the I2C interface, making it compatible with various systems. (Industries 2024)

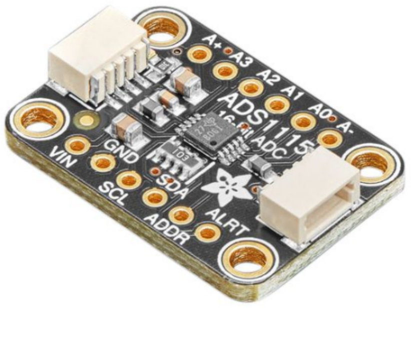


Figure 13: ADC (Industries 2024)

Ethernet Switch
An Ethernet switch operates as a central networking device that facilitates efficient data transfer among various devices within a local area network(LAN). A LAN is simply a group of interconnected devices within the same physical area. The switch operates by directing incoming data to its intended destination by utilizing the devices' media access control(MAC) addresses. By analyzing

these MAC addresses within each data packet, the switch effectively routes the data, reducing any network traffic and improving overall efficiency. (Danielson 2024)

The model in use is the Gigablox Switch, manufactured by BotBlox. This 5-port gigabit Ethernet switch supports data transfer speeds of up to 1 gigabit per second. Its ports are equipped with Molex picoblade connectors, ensuring a compact size. It can operate on a wide span of input voltages ranging from 5,1 to 60 volts making it extremely versatile and suitable for various environments. The device operates as an unmanaged switch, meaning it requires no manual configuration, simplifying its usage and making it beginner-friendly. With such high data transfer speeds across its 5 ports, the Gigablox switch efficiently routes data packets between the modem and the SBC, while also serving as a flexible and scalable connection device for future expansions. (BotBlox 2023)

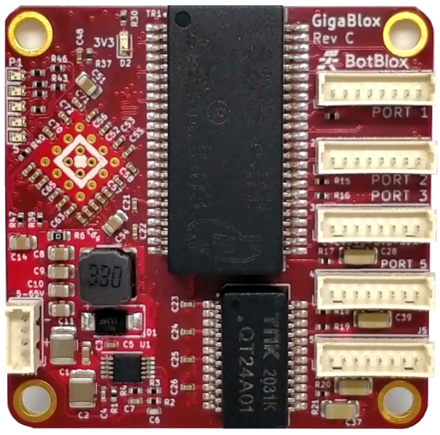


Figure 14: Ethernet Switch BotBlox 2023)

Barrier Strip
A barrier strip is an insulated strip intended for connecting multiple wires in one location. Featuring screw terminals divided by plastic barriers, the barrier strip facilitates safe and organized wire interconnection while offering convenience through its easily adjustable screw terminals. The model utilized is manufactures by Molex and features a high voltage and current rating of 300 volts and 15 amperes, ensuring robustness and durability. Additionally, equipped with 6 positions for wire connections, it is ideal for powering the interface circuit. (*Barrier Strip Terminal Blocks: A Guide to Electrical Connections* 2024)

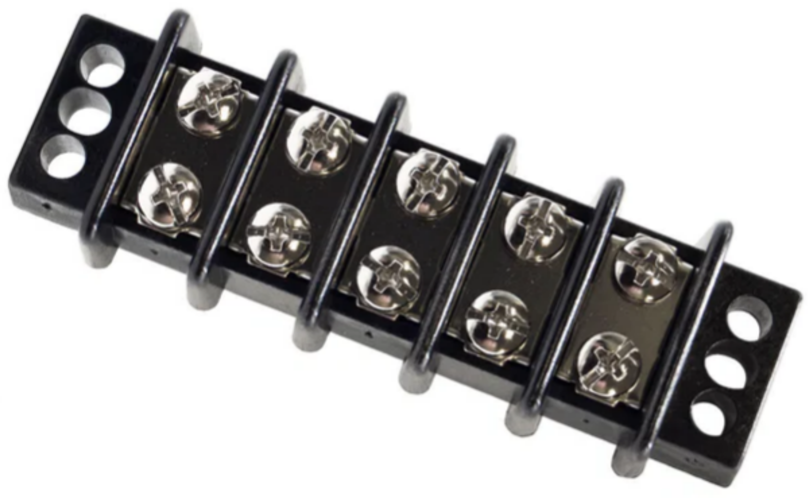


Figure 15: Barrier strip (*Mauser Electronics* 2024)

Battery
The battery in use is a six-cell LiPo battery manufactured by Tatto Plus. With its six cells, this battery outputs a voltage of 22,2 volts and has a total capacity of 12000 mAh. Operating within a wide temperature range from -10 to 40 degrees Celsius, it is suitable for various conditions, with recommended charging and discharging temperatures between 0 and 40 degrees Celsius. This smart battery features several safety and alert functions, including over and under voltage and temperature alert and alarms. Equipped with a Battery Management System (BMS), it autonomously manage itself, extending battery life and making it ideal for unmanned devices. These features add to the safety and reliability of the product, making it suitable for the sensing rig. (*TATTU PLUS INTELLIGENT FLIGHT BATTERY* 2024)

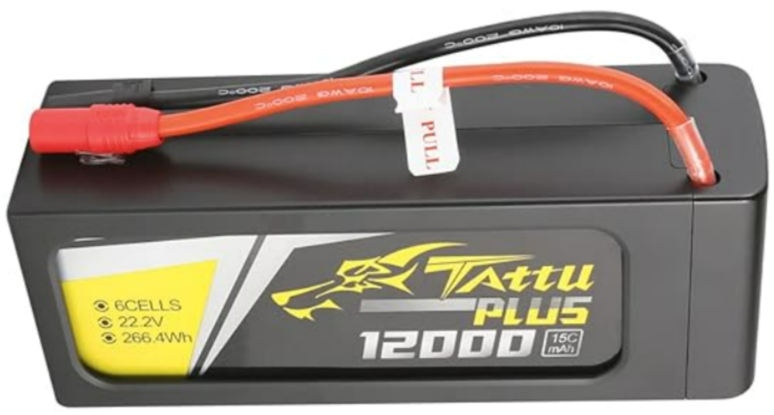


Figure 16: Battery (*Lipo Smart Battery Pack* 2024)

Acoustic Modem
An acoustic modem is a device utilized for wireless data transmission underwater. It operates by converting digital data into acoustic sound waves and transmitting it to a second acoustic modem which converts the sound waves back into digital data. While acoustic waves typically offer lower data transmission rates compared to electromagnetic waves, electromagnetic waves are outperformed in underwater environments due to acoustic waves' ability to establish communication over great distances, overcoming the challenges that affect electromagnetic signals.

Figure 17: EvoLogics' Acoustic Modem

2.1.3 Raspberry Pi 4 Model B

A Raspberry Pi is a type of single-board computer (SBC). Much like a regular PC, this device is equipped with a processor with an integrated graphics driver, and a random-access memory (RAM). The Raspberry Pi 4 model B comes equipped with multiple components and ports, making it a versatile computer with many uses. Some of these components includes Ethernet, ports for camera, display and audio, SD-card slot for data storage and operating system, USB connectors and GPIO header (*Raspberry Pi 4 Tech Specs* 2024). The processor used in this RPi is a Broadcom BCM2711 Quad Core Cortex-A72 (ARM v8) 64-bit SoC, which, like previously mentioned, features an integrated GPU (VideoCore VI). Because of the size and the possibilities of the RPi there are a lot of applications and areas it can be used, such as robotics, automation, learning programming and a lot more.

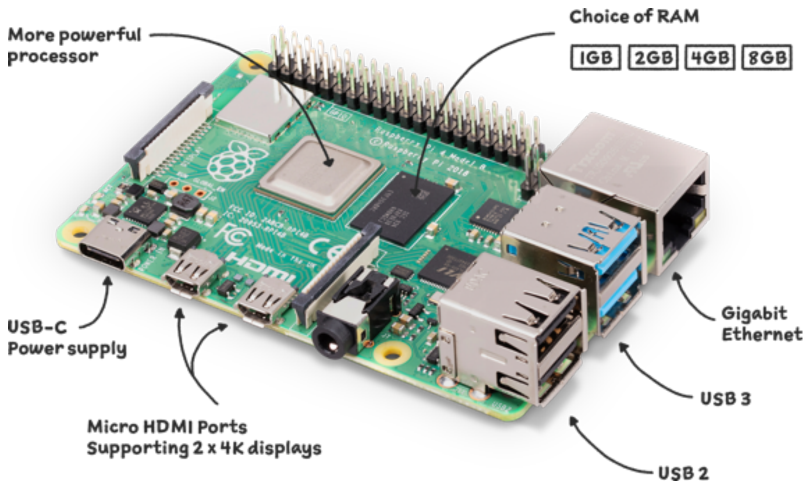


Figure 18: Raspberry Pi 4 model B (*Raspberry Pi 4* 2024)

2.1.4 Khadas VIM3

The Khadas VIM3 is also a type of single-board computer and shares a lot of similarities with the Raspberry Pi. It uses Amlogic A311D SoC: x4 Cortex A73 performance-cores as well as x2 Cortex A53 efficiency-cores (*VIM3* 2024), making it more powerful than the Raspberry Pi. It shares a lot of the same components types as the RPi, in terms of ports and features, as well as taking it even further, making a much more complex and powerful SBC. The Khadas VIM3 uses an m.2 slot for the SD-card, enabling high-performance data storing. Another important feature on the Khadas VIM3 is the implementation of an on-board neural processing unit, which greatly increases the performance of neural network applications, such as computer vision, machine learning and other uses of artificial intelligence. This device is well suited for AI applications, robotics, and crypto mining because of it's ability to run and solve algorithms very effectively, but it can also be used for the same things as a RPi, such as a mini-computer and hosting a server.

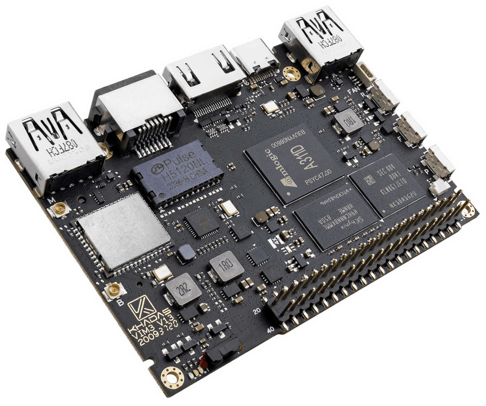


Figure 19: Khadas VIM3 (*Khadas VIM3* 2024)

2.1.5 Real Time Clock

A real time clock, or RTC, is a digital clock with a primary function to keep accurate track of time even when a power supply is turned off or a device is placed in low power mode. RTCs are used in a variety of applications where they play a critical role in keeping accurate track of the current time while also providing alarms, timers, and interrupt functions and helping to reduce power consumption. (Meaney 2024)
The **DS3231** is an extremely accurate I2C RTC, with an integrated temperature compensated crystal oscillator. The chip offers a battery input designated to keeping the time-keeping correct over long periods where the system is without power. The RTC also offers low power consumption, making it perfect for use-cases with remote and long-term sensing projects. The DS1307 was briefly considered due to its lower cost than the DS3231, but ultimately the pros of the DS3231 made it the clear choice for the long term vision of the project.

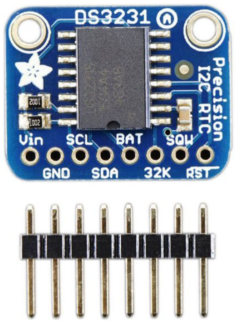


Figure 20: DS3231 Real Time Clock (*DS3231* 2015)

2.1.6 Camera - Intel RealSense D435

The Intel RealSense D435 is a depth-sensing camera designed for robotics and augmented reality. It utilizes stereo vision technology to capture depth information, enabling accurate depth perception and spatial understanding. The D435 features a pair of infrared cameras along with an RGB camera, allowing it to capture both color and depth data simultaneously. With its compact form factor and robust software support, the RealSense D435 is well-suited for integration into other project, such as the sensing rig. (*D435 Depth Camera* 2024)

Figure 21: Camera (*D435 Depth Camera* 2024
)

2.2 Previous Work

Before deciding the goals of the project, getting an overview of the previous work was essential. The group was introduced to three sensing rigs, supposedly identical, but with some slight differences. To differentiate between the three rigs, they are referred to according to the colour of the main bracket; red, blue or black. Along with the sensors and system components, all the rigs were integrated with Raspberry Pi 4 B's as SBCs.

Sensors
All three rigs were mounted with the listed sensors from the theory section. The group was also made aware that the Dissolved Oxygen and the Conductivity sensors were prone to issues. These issues came into play at a later stage.

System Integration Components
All rigs were mounted with the listed system integration components, but both the red and black rig were in need of some small modifications. The group was informed that the DC-DC converter that was mounted on the red rig had a risk of short-circuiting, and therefore needed to be replaced. In addition to this, two of the I2C Bus Splitters had a missing header pin, the DF13 connector.
The red and black rigs were also equipped with SubConn MCIL8M pigtails for connection with the acoustic modem used in the project. This modem, from SubNero, is different from the one

being used in this years project, which leads to an issue when it comes to connecting the rig to the modem. The issue in question is that the modem used in this years project, the EvoLogics modem, need SubConn MCIL8F pigtails to establish a connection.

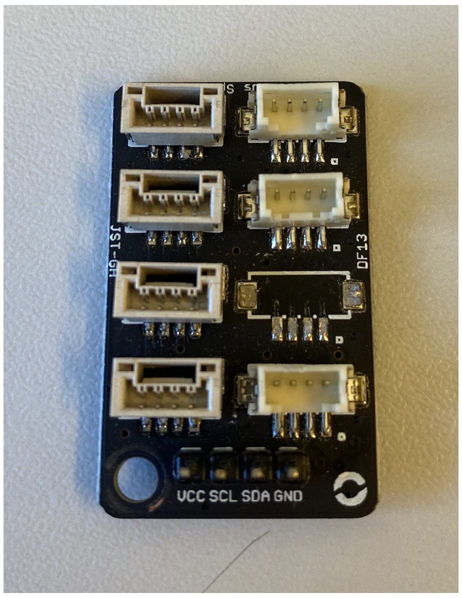
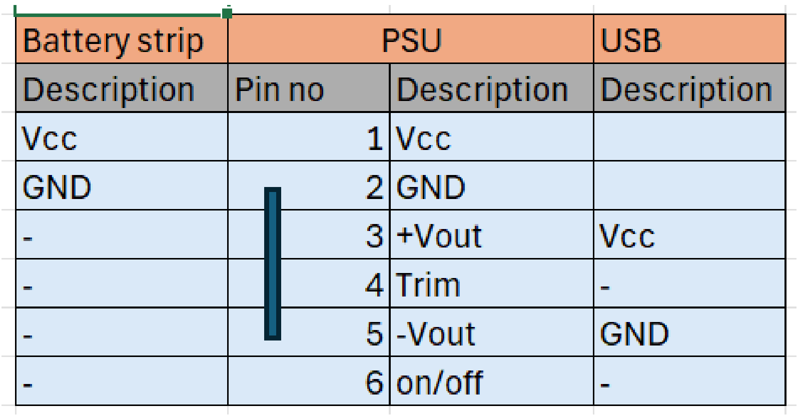


Figure 22: Missing header pin on I2C Splitter

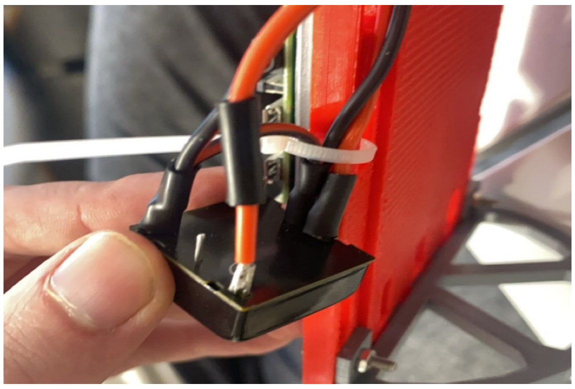
2.3 Current state

Replacing DC-DC Converter
Two of the three DC-DC converters had already been replaced with the Traco-THN converter. The reason for the change was an isolated output on the original converter, resulting in both a floating output voltage as well as a floating ground, which is undesirable for the system. By directly soldering a common ground on the new DC-DC converter from the barrier strip, this problem is avoided. The last converter was then replaced, with the following table and images illustrating how the soldering took place with a common ground. In the following datasheet for the converter, further information is listed as well as an illustration of the pin locations. Power 2022



Battery strip	PSU		USB
Description	Pin no	Description	Description
Vcc	1	Vcc	
GND	2	GND	
-	3	+Vout	Vcc
-	4	Trim	-
-	5	-Vout	GND
-	6	on/off	-

Figure 23: Pinout table for PSU, with common ground, NOTE ”Barrier strip*”


(a) PSU

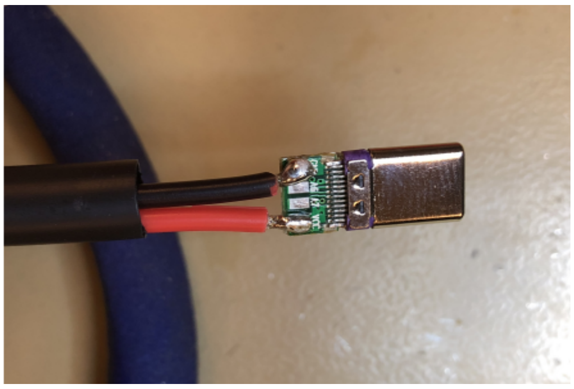
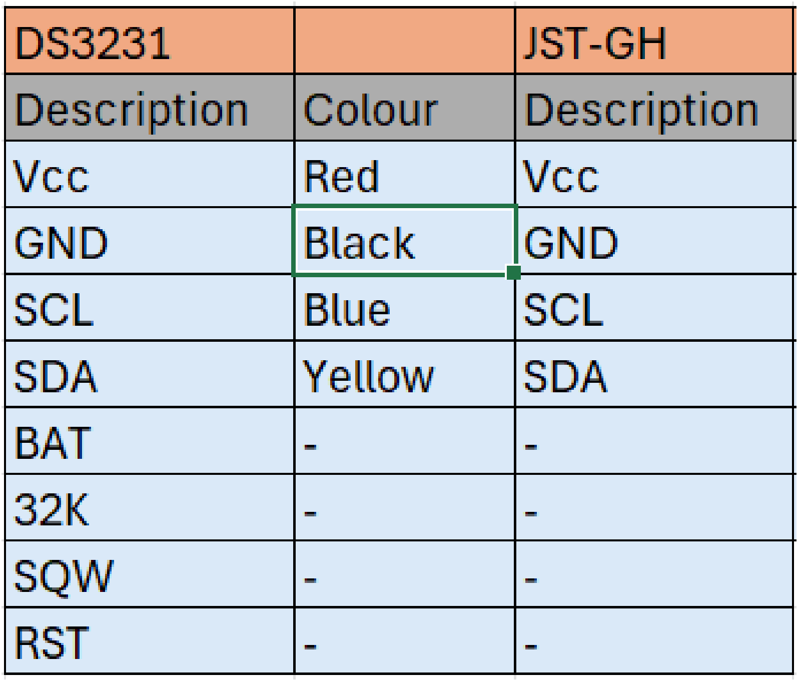

(b) USB-C

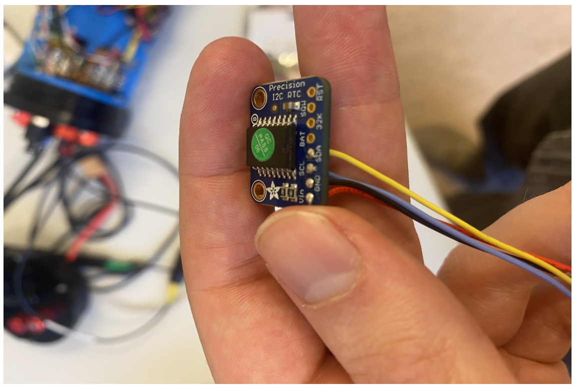
Figure 24: Soldering points

Adding RTC
The DS3231 uses I2C communication, which integrates seamlessly into the existing project. The RTC connects directly to the I2C-Bus splitter via either a JST-GH connector or a DF13 connector. In the project, a JST-GH connector was used, with the following pinout table illustrating the connections. The colour coding used in the table is trivial, but to maintain a sense of continuity as well as easier understanding, the exact same colour coding was used on all three RTCs.

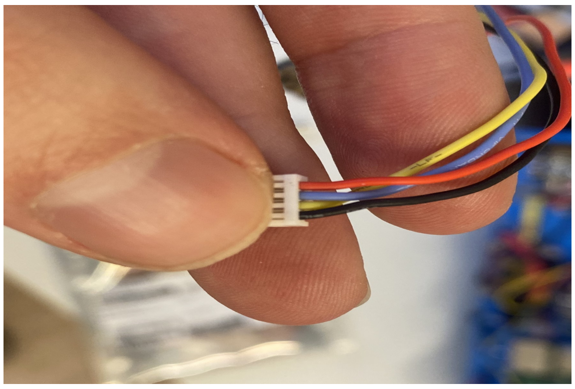


DS3231		JST-GH
Description	Colour	Description
Vcc	Red	Vcc
GND	Black	GND
SCL	Blue	SCL
SDA	Yellow	SDA
BAT	-	-
32K	-	-
SQW	-	-
RST	-	-

Figure 25: Pinout table for RTC



(a) RTC soldering points



(b) JST-GH connector

Connecting to the Acoustic Modem
Due to last years project being compatible with SubNero modems, and not EvoLogics modem, the cable had to be replaced. An order for new SubConn MCIL8F pigtails was pushed, but due to delivery time, a testing cable was also produced to be able to do a final test of the system.
The cable was made by soldering and testing the SubConn according to the pinout table in figure 28. Making the SubConn end of the cable will not need to be replicated when the pigtails are delivered, but the procedure is still briefly explained here. Properly installing the WetLink to the other end of the cable will need to be replicated when the ordered cables are delivered.

How-to make, along with important things to note when making the SubConn end:

1. Cut the wires quite short to ensure that all wires and cast will be covered by the tube that will be fitted around the plug and cable jacket.
2. Cut the tube so that it fits tight onto the cable jacket. It should be tight fitting and barely able to slide into place after soldering.
3. Solder the splices according to the pinout table. You will need to thread a heat shrinking tube on the wires BEFORE soldering, and then apply heat so that the heat shrinking tubes are snug.
4. Fill the tube with cast and leave it in an upright position to harden overnight.

Installing the WetLink Penetrator
When installing the WetLink penetrator, you will need to know which tools to use. There are different bulkhead sizes and plug types according to which cable you are using. Some often used cables are listed in BlueRobotics' web page for use with WetLink Penetrators. For example the testing cable that was made was directly from BlueRobotics, where the correct tools were already specified. When installing the WetLink on the SubConn MCIl8F pigtail, the correct tool will be a **9.5 Penetrator with a HC plug**. This specifies the diameter of the penetrator and how tight the plug will squeeze on the cable jacket. All info regarding choosing the correct WetLink tools as well as properly installing the penetrator correctly can be found here: *WetLink Penetrator Installation Guide* 2024

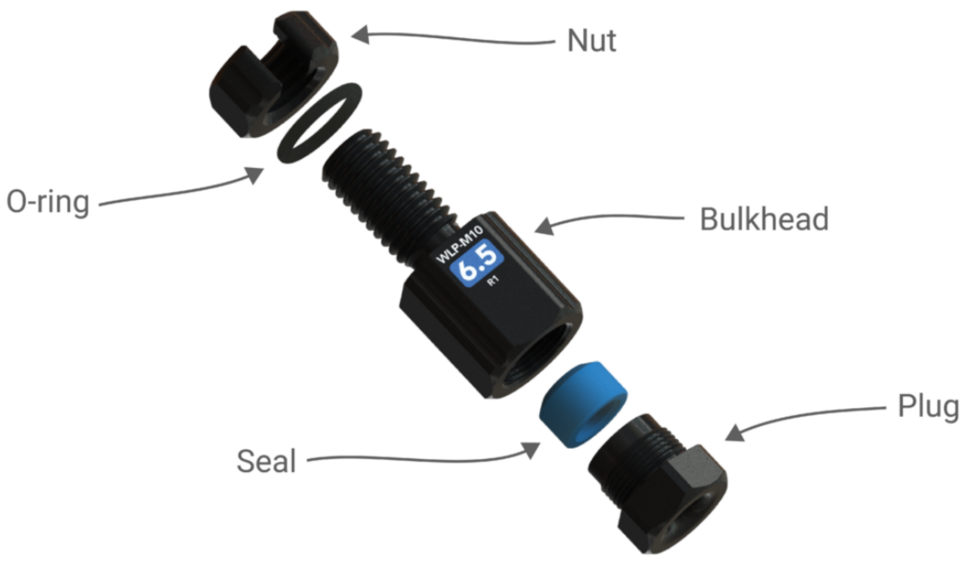
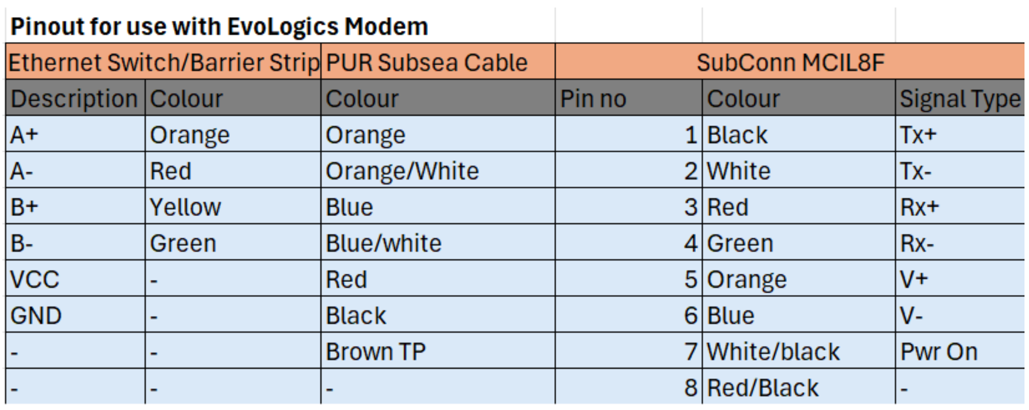


Figure 27: Illustration of the WetLink Penetrator with parts



Pinout for use with EvoLogics Modem

Ethernet Switch/Barrier Strip		PUR Subsea Cable	SubConn MCIL8F		
Description	Colour	Colour	Pin no	Colour	Signal Type
A+	Orange	Orange	1	Black	Tx+
A-	Red	Orange/White	2	White	Tx-
B+	Yellow	Blue	3	Red	Rx+
B-	Green	Blue/white	4	Green	Rx-
VCC	-	Red	5	Orange	V+
GND	-	Black	6	Blue	V-
-	-	Brown TP	7	White/black	Pwr On
-	-	-	8	Red/Black	-

Figure 28: Pinout table for use with Evologics modem

Ethernet Connection

To be able to connect to a sensing rig with any computer, a wired Ethernet connection was established. By installing a cobalt connector, which is connected to the Ethernet switch, on the lid of the sensing rig, a wired connection between the rig and any pc, laptop or Ethernet switch with an Ethernet port becomes possible. This solution makes extracting data in remote areas with any computer easy and accessible. A cable with a cobalt plug on one end and an RJ45 plug on the other end was made and tested according to the pinout table for Ethernet connection, fig 30.

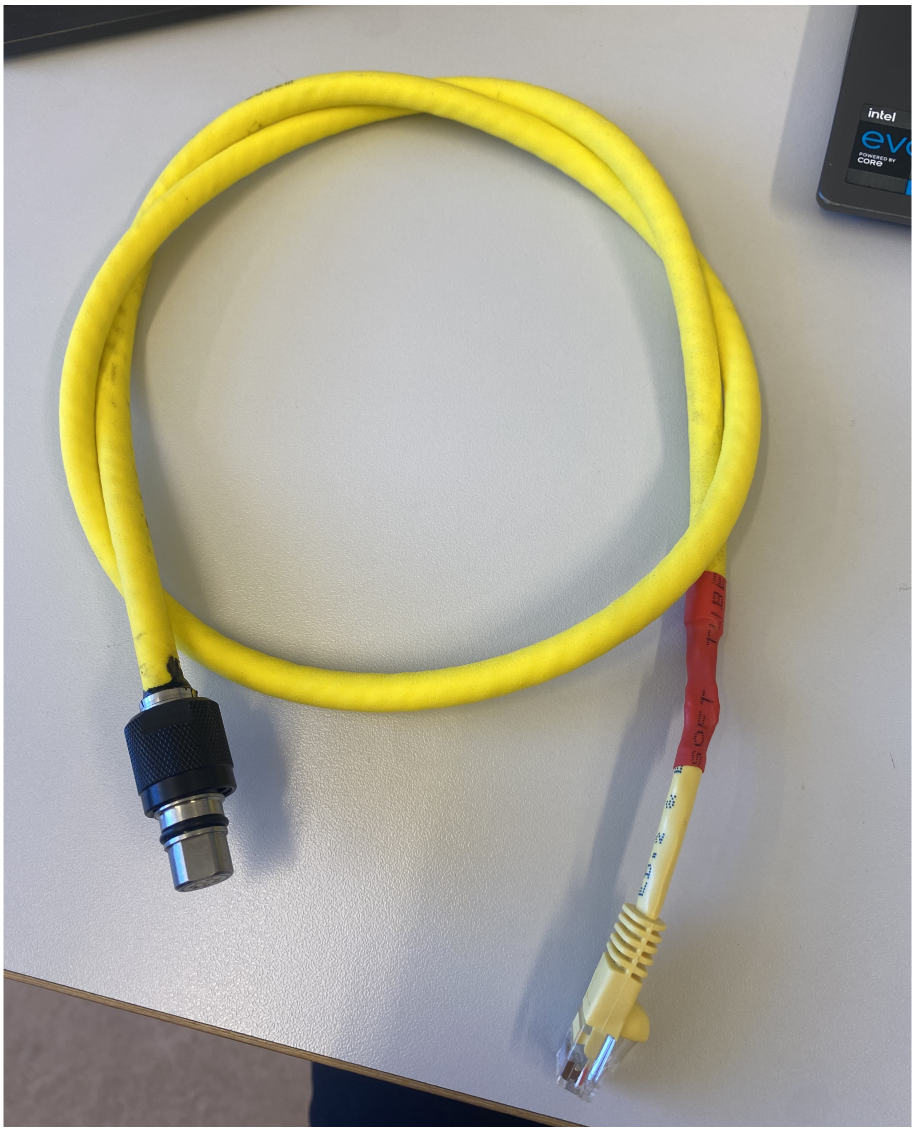
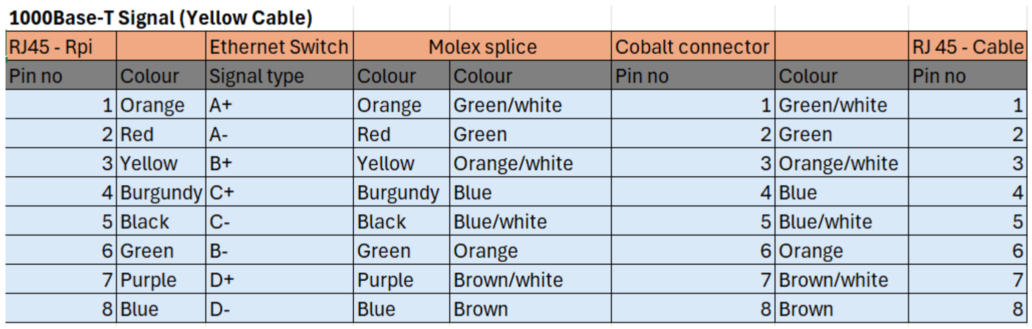


Figure 29: Cable for Ethernet Connection



1000Base-T Signal (Yellow Cable)

RJ45 - Rpi		Ethernet Switch	Molex splice		Cobalt connector		RJ 45 - Cable
Pin no	Colour	Signal type	Colour	Colour	Pin no	Colour	Pin no
1	Orange	A+	Orange	Green/white	1	Green/white	1
2	Red	A-	Red	Green	2	Green	2
3	Yellow	B+	Yellow	Orange/white	3	Orange/white	3
4	Burgundy	C+	Burgundy	Blue	4	Blue	4
5	Black	C-	Black	Blue/white	5	Blue/white	5
6	Green	B-	Green	Orange	6	Orange	6
7	Purple	D+	Purple	Brown/white	7	Brown/white	7
8	Blue	D-	Blue	Brown	8	Brown	8

Figure 30: Pinout table for Ethernet connection

Implementing VIM3
The implementation of VIM3 began by acquiring a micro SD card to expand the internal storage. With a capacity of 128 GB, this micro SD-card significantly increases the SBC's storage capacity, meeting future storage demands. The next step involved redesigning the SBC mount through CAD and 3D print it to accommodate the VIM3. Following the production of the bracket, the VIM3 was

connected to the I2C splitter and power source, completing the implementation process regarding hardware.

Replacing I2C Bus Splitter
Due to the fragility of the DF13 plugs on the I2C-Bus splitter, there was a need of replacing the splitter on the RED rig. Here, two of the plugs were removed, and such, there were not enough plugs for the connectors needed. This was replaced simply by soldering pins onto the board, which are then connected with Molex plugs.

2.4 Further work

This section covers suggestions for further improvement of the project along with useful things to note when working with the electronics.

Deficiencies
The group observed multiple deficiencies related to the Atlas sensors. Several Atlas sensors on the black and red rigs were defective, leaving only the salinity sensor on the red rig and both sensors on the blue rig operational. The defective sensors were undetectable by the SBC over I2C. For further work, consider replacing the Atlas sensors with another brand through a design review or ordering an excess of sensors to reduce the impact of such deficiencies. Note that an order was placed for a set of Atlas sensors to replace the faulty ones, but they have not been delivered at the time of this writing.

Another deficiency concerns the RTC battery for the VIM3, which is needed for accurate timekeeping without requiring an internet connection on boot-up. Similarly, an order for the RTC battery was placed, but it was never received by the group after its delivery to NTNU.

Sensor calibration
For future development, sensor calibration will be essential to verify the accuracy of the transmitted data. Although calibration was not a focus point during this iteration of the project, it is crucial for achieving the end goal of ensuring the data from the sensing rig is accurate and reliable.

Replace testing SubConn with SubConn MCIL8F pigtails
Using the guide provided by BlueRobotics (*WetLink Penetrator Installation Guide* 2024), install WetLink Penetrator 9.5 HC on the pigtail. Connect the wires to Molex plugs (Guide in PA-folder), using the pinout table for use with EvoLogics modem (fig 28).
NOTE: Colour coding may not be the same, but pin number will still match.

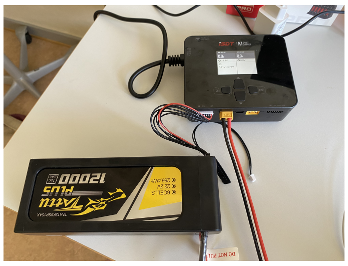
VIM replacement
For greater processing power and utilization of AI applications, it is recommended to equip all three rigs with the VIM3. If heavy processing is taking place, consider adding a heatsink or fan to the SBC to enhance the safety and lifespan of the sensing rig. For software integration of the VIM3, refer to the guide on 4.3.5

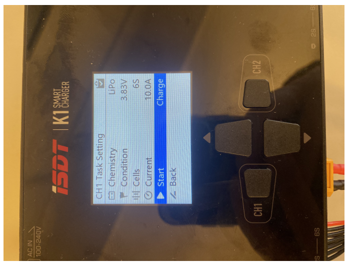
Wiring
For more versatility when working with the sensing rigs, using Molex plugs where possible is advantageous. In addition to this, having pinout tables for all connections where deemed useful could be a nice improvement to the system.
Furthermore, organising all wires in a more neat way on the main bracket will make the system easier to understand.

2.4.1 Charging the LiPo battery

The battery cells have a working voltage range of about 3,2-4,2V. If the cells reach lower voltages than 3,2V, they can be permanently damaged, and a higher voltage then 4,2 comes with a risk of the battery bursting into flames. Lithium batteries are recommended to be stored at about 40% capacity, with the LiPo cell storage charge being at 3,8V.
When charging the battery, all you need is the battery itself and the ISDT K1 Smart Charger. The procedure is very simple, but it is VERY important that you never leave the battery unattended while charging.

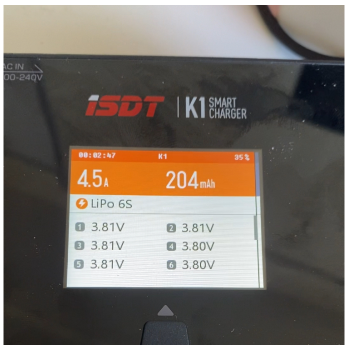
1. Connect the balance plug and power plug from the charger to the battery. There will be a loose plug coming from the charger, but disregard this.
2. Navigate to the configuration menu of the charge cycle by holding in the CH1 button. You want to set the battery type, target voltage, amount of cells and current.
3. Start charging. When the cells are balanced at the desired voltage, you can stop charging manually.


(a) Connection between Battery and charger



```
CH1 Task Setting
Chemistry      LiPo
Condition      3.83V
Cells          6S
Current        10.0A
Start          Charge
Back
```

(b) Configuration menu



```
00:02:47        K1         35%
4.5A            204mAh
LiPo 6S
1 3.81V         2 3.81V
3 3.81V         4 3.80V
5 3.81V         6 3.80V
```

(c) Battery Cell Voltage

Figure 31: Charging the battery

3 Enclosure

One of the main aspects in the development of the sensing rigs was the implementation of cameras. There is a lot of information that can be obtained and processed in the ocean, but there is a limited amount that can be obtained through sensors only. The goal of the cameras is to be able to take pictures when the sensing rigs detect that something is happening, and both process them for us to see and send them through a computer vision algorithm. There is a lot of applicabilities for a system like this, but for it to work the cameras need to be waterproofed and be able to see.

Another focus point was continuing the development of last years group on the already existing enclosure. This includes any changes to the inside of the rigs, either replacement of existing parts or implementation of new ones. This differs from the camera enclosures, as that was an extension of the outside of the enclosure itself. The inside of the rigs refers to any add-ons or changes to the main mounting plate.

3.1 Theory

3.1.1 Water and electricity

The importance of waterproofing electronic hardware comes from the term *conductivity*. Non-pure water, which is all natural sources of water, is conductive, meaning it can conduct or carry electrical current. Though the water itself is not conductive, the free ions in the water is. The two most common elements in seawater after hydrogen and oxygen, is sodium and chloride (NOAA 2024), which when combined makes salt (NaCl). Salts are ionic compounds composed of positively- and negatively charged ions (USGS 2024), which when added to water conducts electricity. This means that electronics covered in water will connect points together that are not supposed to and conduct electricity between them, which will cause a short-circuit and damage or destroy the hardware.

3.1.2 Proofing

There are two main needs of proofing for the sensing rigs, the first of which is waterproofing. Waterproofing is the process of prohibiting flow of liquids into an object or structure. There are many use-cases for waterproofing and many different methods in doing so. The sensing rigs have their needs for waterproofing and a specific way of doing it. It is important to use solid materials and to seal any openings. The most common way to waterproof openings is to use o-rings, which is a ring-shaped seal made of rubber, and a silicone-based gel. Since the o-ring is flexible, it will adapt to the tolerances of the object in order to keep it sealed (Sealution 2024). Additionally, the pressure applied by the water to the object will push the cover into the opening, making sure it's completely shut.

For any object that will be in deep water there is also the need of being pressure-resistant, which means being able to withstand the pressure applied by the water to prevent implosion. Implosion happens when the pressure applied to an object is greater than what the object is able to withstand, and thus bursting inwards, or 'exploding' inwards. To calculate the hydrostatic pressure on an underwater object, you use the equation $P = \rho \cdot g \cdot h$, where ρ is the density of the water, g is the gravitational force on the object and h is how far underneath the surface the object is. Then you have the amount of pressure the object will have to withstand, which you will need to use when designing it. It is important to know which area of use the object will have when determining the materials of it, what type of water it will be submerged in and how deep it will be.

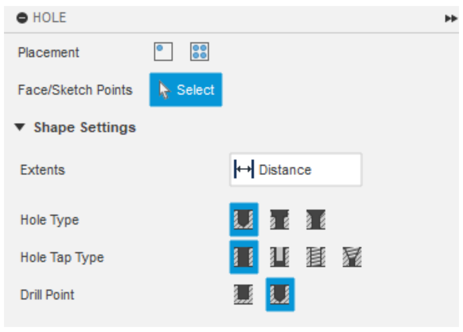
3.1.3 Computer-Aided Design

Computer-Aided Design (CAD) is using computer-based software to design and draw in both 2D and 3D. CAD programs and applications make it possible to turn blueprints and drawings into digital prototypes. It is a very important tool when it comes to 3D-modelling in the physical world, as it allows for a very hands-on design process where you have a lot of control when it comes to designing and making design changes. It can be described as a tool to improve the design- and manufacturing process for designers and engineers. Some of the benefits to using CAD includes lower production costs, higher quality design with documentation built into the file, makes collaboration easier and the software tools increases effectiveness with built-in features to help with design (Chai 2024).

3.1.4 Fusion 360

Fusion 360 is a three dimensional 'design and make'-software developed by Autodesk (*What is Autodesk Fusion?* 2024). Its main functionality is designing and creating 3D-models using computer aided design, or CAD. It is a very effective software for high-precision manipulation of 3D-models, which allows you to create very detailed models. It comes equipped with a lot of tools which comes in handy when designing parts. Some of those tools includes:

- **Create sketch:** This tool is what you use to create the shape of the model you are designing. Creating a sketch happens in two dimensions, allowing you to create a baseline for the model, which can be manipulated further later on.
- **Extrude:** In order to get a three dimensional model, you need to add the third dimension to the previously created sketch. *Extrude* allows you to do this by selecting the sketch and changing the value of the missing dimension. Further you can alter any object with *extrude* by selecting any side of the object and changing the value of the dimension not present on that two dimensional surface.
- **Revolve:** For creating symmetrical circular objects, *revolve* is an important tool. It allows you to take a two dimensional plane and wrap it around or revolve it around an axis.
- **Holes:** This tool is specifically used for creating round holes through or into a model. It is very useful for creating screw holes and gives you a lot of different options to choose from, like shown in figure 32.


HOLE

Placement

Face/Sketch Points: Select

Shape Settings

Extents: Distance

Hole Type

Hole Tap Type

Drill Point

Figure 32: Holes-tool options

- **Shell:** This tool is a bit situational, however it is very useful. It is used to remove part of the inside of a model, making it hollow with a specified thickness for the walls. This tool can be used in a lot of different ways, so it's just about being a bit creative with it.
- **Move/copy:** The *move/copy* tool is pretty straightforward. It is used to move or copy parts within the model, such as holes or standalone objects that make up the model. This gives you quite a lot of flexibility and makes it easy to make changes to the model as you go.

Learning how to use these tools is key to creating and editing parts for any project involving CAD.

3.2 Previous work

At the takeover from last years project, the state of the sensing rigs in terms of enclosure was three waterproof rigs, where two of them were equipped with mounts for a communication modem. The inside of the enclosures came equipped with a main plate which held all the different components used in the sensing rigs. These components were mounted to the plate with their own 3D-printed parts.

Two of the three enclosures were identical to each other, with the last one being slightly older and missing the brackets for the communication modem, as well as the SubConn cable. The casing itself is also a bit different in terms of waterproofing, with the feature of additional O-rings missing. Nonetheless all of the sensing rigs passed the vacuum test at the takeover, meaning they are all airtight and waterproof.

In terms of the 3D-models that has been used in the project, there were a lot of missing CAD files, meaning it would be hard to make any changes to the models if that's needed. The only CAD files that were accessible at the takeover, was the models created by last years group.

3.3 Current state

3.3.1 Camera - Intel RealSense d435

The cameras used in the sensing rigs, Intel RealSense d435, is connected to the computers via a USB-C to USB-A 3.0 cable. This cable came with the camera and is specifically designed for video data transmission. Therefore, it is not recommended to replace this cable with a third-party cable, as there is no guarantee that it will work well or at all. If the cable needs to be replaced due to a broken one or any other reason, it needs to have a high transmission speed. The only third-party brand that is confirmed to be compatible with the Intel RealSense d400-series is the Newnex USB-C 3.1 cables (*USB 3.1 Type-C Cables for Intel® RealSense™ Camera* 2024). The cameras are placed by themselves in the camera housing mounted by a couple of screws.

3.3.2 Enclosure

Camera housing
The biggest change to the sensing rigs' enclosure is the implementation of the camera housing. Since the Intel RealSense camera is not waterproof or made for underwater-use by itself, there was a case of coming up with a solution to accommodate for the underwater-usage. The first option that was being explored was having a standalone enclosure for the camera, which would be mounted on the sensing rigs. Difficulties with wiring led to this idea being scrapped, as the USB-C cables loses capabilities when being cut, leading to slower data transmission. Further brainstorming led to an idea consisting of expanding the sensing rigs to fit the camera inside with the electronics. This way, there was no need for waterproofing and cutting the cables, as well as being a more space-efficient solution. The sketch for the camera housing [figure 33] was based on similar projects, though it was created before the decision to combine the camera housing and the enclosure.

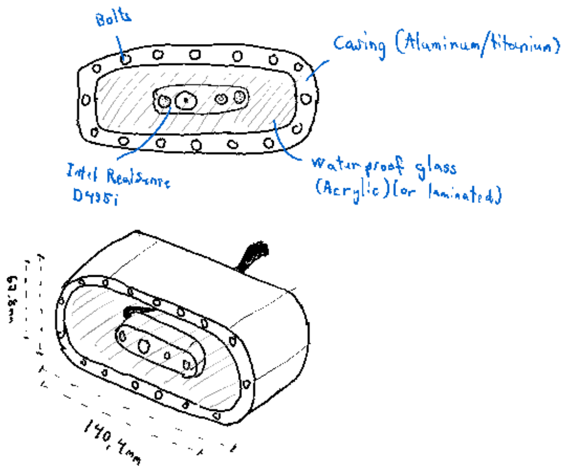

Bolts
Casing (Aluminum/titanium)
Intel RealSense D435i
Waterproof glass (Acrylic)(or laminated)
63,8mm
140,4mm

Figure 33: Early sketch of camera housing

The final design was made in collaboration with the mechanical workshop, as they had experience in similar projects as well. The sketch of the separate housing was redesigned to fit the camera size and measurements, and then attached to a panel fitting the BlueRobotics O-Ring Flanges. The camera housing is made to fit with different thicknesses of plexiglass, supporting up to 10mm thick glass. Currently it is equipped with 3mm, and is not advised going further than 50m below the water surface. As seen in figure 34, the final iteration includes a back plate that matches the BlueRobotics O-Ring Flanges, screw-holes for mounting the camera to the plate, holes for wiring between the camera housing and the rig and a replaceable plexiglass window for the camera.

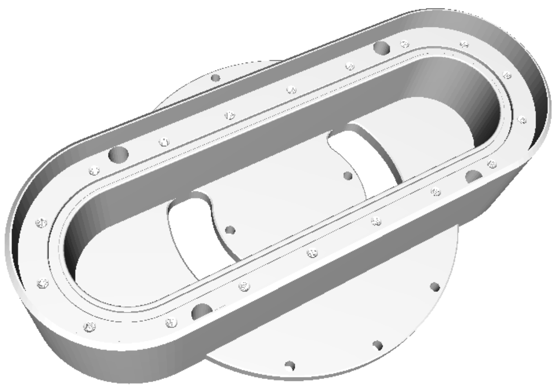


Figure 34: 3D-model of the camera housing

VIM mount

Another big change, which only included one of the rigs, was replacing the Raspberry Pi 4 with a Khadas VIM3. For this to be possible, there was some changes and adjustments needed for the mount and the parts inside the rig. The Khadas VIM3's mounting screw holes were different to the Raspberry Pi's, with the location and size of them differing. Thus the mount would need to be edited to fit the Khadas VIM3, and the track plate would need to fit the mount. For the rigs still using the Raspberry Pi, there was an implementation of a Real Time Clock, which also needed to be mounted. The mount for this device was created from the bottom according to the specifications of the RTC and fitting the mainplate. This particular mount was attached using plastic glue, as it is quite a lightweight part which will not need further adjustment. For one of the rigs, there was a problem with the splitter stand, as it was not possible to flip the I2C-splitter 180 degrees horizontally, which needed to be done to access the USB-A 3.0 port on the Raspberry Pi. This was changed in the new version of the splitter stand, which adds an identical screw hole and pit to the other side. Lastly, the mainplate and the already edited parts have been converted from an .stl-file to an editable CAD-file.

3.3.3 Convert .stl to .step

Like mentioned in section 3.2, most of the models used in this project were unavailable as CAD-files (.step), making it hard to make any changes to the parts. Luckily the 3D-models (.stl) were accessible, which means there was no need to design new parts as CAD-files. However, there is no good way to convert an .stl-file to a .step-file. This way of converting models is very flawed, and leaves the CAD-file in a non-ideal state, where the model is hollow and may be generated with a varying amount of holes. This can become an issue when 3D-printing the parts, as it may either print a bugged model or not print at all. Even then, this seems to be the only way to do it, without having to design a brand new model:

- **Step 1 -** The first thing you would want to do is importing the 3D-model to Fusion 360. It's important that you use the *insert* tool to get the right measurements.

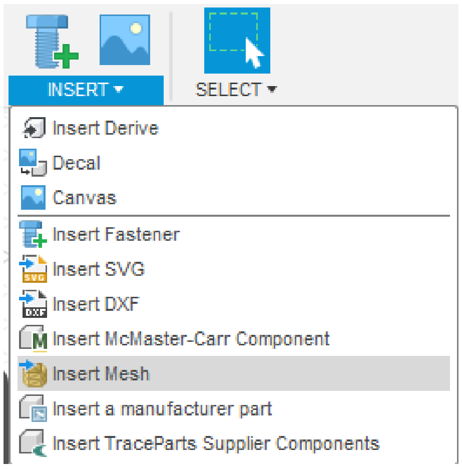

INSERT
SELECT
Insert Derive
Decal
Canvas
Insert Fastener
Insert SVG
Insert DXF
Insert McMaster-Carr Component
Insert Mesh
Insert a manufacturer part
Insert TraceParts Supplier Components

Figure 35: Insert mesh

  Then you just select *Insert Mesh* and upload the .stl-file that you want to convert.

- **Step 2 -** The next step you need to do is to generate face groups. The way you do this is clicking the *Prepare* tab and scrolling down to and selecting *Generate Face Groups*.

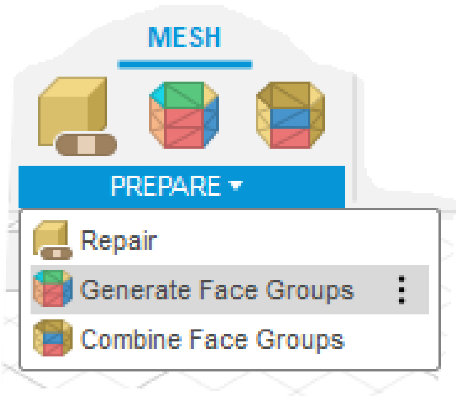
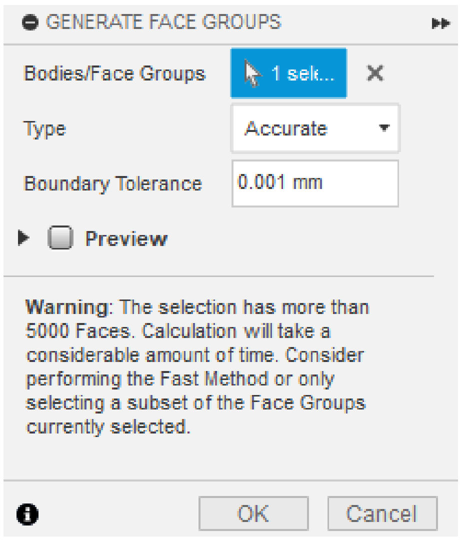


Figure 36: Prepare tab

  You will then get a window to pop up. In this window you need to select *Accurate* in the *Type* tab, then select the model you want to generate face groups for and click *OK*.



GENERATE FACE GROUPS

Bodies/Face Groups: 1 sele...

Type: Accurate

Boundary Tolerance: 0.001 mm

Preview

Warning: The selection has more than 5000 Faces. Calculation will take a considerable amount of time. Consider performing the Fast Method or only selecting a subset of the Face Groups currently selected.

Figure 37: Generate Face Groups

- **Step 3 -** Once you have generated the face groups, the next step would be to converting the mesh. You can do this by clicking the *Modify* tab and navigating down to the *Convert Mesh*-button.

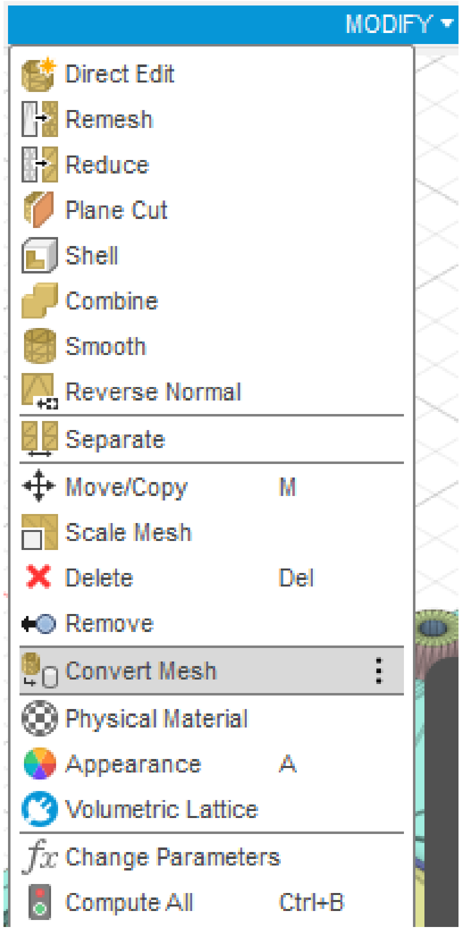
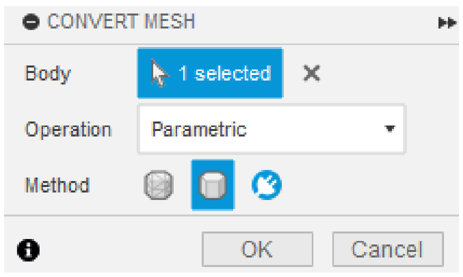


Figure 38: Modify tab

Once again a window will pop up. In this window you select the *Parametric* operation and the *Prismatic* method and click *OK*. Note that the *Prismatic* method is only in the licensed version of Fusion 360, so make sure you have this version. If you don't have access to this version, you will need to select the *Faceted* method, which will not make one solid model, but a model consisting of many different parts.


CONVERT MESH

Body: 1 selected

Operation: Parametric

Figure 39: Convert Mesh

Now you should have a solid model, but like mentioned earlier, there may be some holes in the model, which will need to be patched up.

- **Step 4 -** The way to do this is using the *Sketch* tool to create a sketch matching the shape of the hole, then using the *Extrude* tool to filling it.

3.4 Further work

This section covers some suggestions for further work in terms of the enclosure.

File conversion
In order to create a complete library of all the parts used and previously used, it is necessary to convert more files into CAD files (.step) to ensure they are easily editable in a CAD program. The guide in section 3.3.3 outlines the process for converting .stl files to .step files.

Replace the plexiglass
Additionally, to increase the maximum recommended depth beyond the current 50 meters, the plexiglass on the camera housing will need to be replaced with a thicker variant. The camera housing supports up to 10mm thick plexiglass, whereas the current plexiglass in use is 3mm thick. By using 10mm thick plexiglass, the sensing rigs will be able to withstand the pressure at depths greater than 50 meters.

Implementation of light source
The implementation of lighting for the camera is another consideration. The sensing rigs have not yet been tested in deep water, so there is no information on how well the camera performs at greater depths. After conducting tests in the sea, it may be beneficial to decide whether or not to add external lighting.

Buoyancy
Lastly, the buoyancy of the sensing rigs needs to be addressed. Currently, the rigs either float, end up upside down, or both. Therefore, adjustments are required to ensure that the sensing rigs sink and remain upright.

4 Software

Software development is a crucial aspect of the sensing rig's development. The main focus is on integrating and configuring both the camera and VIM3. This includes adapting GitHub code/files to accommodate image transfer and basic computer vision processing for both the Raspberry Pi and VIM3 and transmitting it through the acoustic modem. This section is formed as a detailed tutorial covering the complete development process of both the SBC's and camera software for a greater understanding.

4.1 Theory

The goal of this section is to provide a foundation of knowledge necessary for understanding and engaging in the software-related development conducted. Detailed reviews of Python and Ubuntu servers has been considered excessive, but are valuable reading material leading up to this report. These topics can be found in previous year's thesis (Knudsen and Brandstorp 2023). The topics covered in this section are directly relevant to the structure of software where development has taken place.

4.1.1 ROS2

ROS 2 is the updated iteration of ROS (Robot Operating System), which, despite its name, isn't a traditional operating system like Windows or macOS. Instead, it acts more like an extensive suite of libraries and software tools. It is designed to aid developers in building and operating complex robotic systems, which often consist of actuators, sensors, and control systems.

Robot systems quickly become intricate, and ROS 2 helps manage this complexity by providing robust tools for developing components independently and facilitating communication between them. This communication is achieved through a modular approach where different parts of a robot's system are handled by "nodes" which perform specific tasks.

When starting with ROS 2, the first step is setting up a workspace environment, which involves organizing a specific directory structure recognized by ROS 2. Users must also source setup files each time they initiate a new terminal session to correctly configure the environment.

In ROS 2, communication between these nodes can occur through four main mechanisms:

- **Topics:** Nodes publish data to topics that other nodes can subscribe to. This setup allows for data, such as continuous temperature readings, to be broadcasted and received, similarly to using hashtags on social media to follow specific content.
- **Services:** This method involves a request-and-response interaction similar to TCP/IP protocols. It's comparable to making a customer service call where a request is placed and a response is awaited.
- **Actions:** Actions blend the characteristics of topics and services. They facilitate ongoing communication for tasks that need continuous operation, such as moving a robot arm from point A to B. During such operations, the action server regularly updates the action client with feedback until the task is completed, while also allowing for the operation to be halted mid-process if necessary.
- **Parameters:** These serve as configurable values (comparable to global variables in programming) that can be adjusted across the ROS environment, typically manipulated via services to alter operational settings like the control signals to actuators.

This structured approach allows for the efficient operation and real-time adjustment of robotic systems, making ROS 2 a powerful framework for developing sophisticated robots. *ROS documentation* 2024

4.1.2 Git and GitHub

Git is a powerful software tool that enables developers to collaborate effectively on projects. It tracks all modifications made to the project files, clearly documenting what changes were made and by whom. One of its standout features is the ability to revert to earlier versions of the project, which enhances control and reduces the risk of errors disrupting the work.

Git operates through a structure called a repository, which logs every change made within it. This structure supports branching and merging, key features for collaborative work. Think of branching like a road that splits into two separate paths. Developers can diverge from the main project (the main path) and experiment on a branch without affecting the core project. If errors occur, it's easy to revert to the point of divergence and start again.

Meanwhile, other developers can create their own branches and make independent changes. When these changes prove successful and don't introduce any problems, branches can be merged. This merging process is similar to reuniting stray roads into a single road, after which new branches can again be created. This cycle allows multiple developers to work simultaneously on the same project without interfering with each other's work. Once significant changes are finalized on a branch, merging it back into the main branch makes those updates available to all team members, facilitating seamless collaboration.

GitHub complements Git by providing an online platform where projects can be stored and shared. It is an open-source hub that not only hosts projects but also facilitates access to others' work, enabling collaborative development across the globe. Developers can 'push' their local projects and updates to GitHub, allowing others to 'pull' these projects onto their own machines for further development.

4.1.3 Computer vision

YOLOv8

Computer vision is a way for computers to see the world through cameras, using AI or machine learning to identify the surroundings. The projects uses OpenCV which is a software tool that helps process the live video from the camera and processing the images that will be run through the YOLOv8 algorithm. The YOLOv8 algorithm is an easy to use but powerful computer vision algorithm that uses image recognition to identify specific objects in a camera frame. For example differentiating a fish from its surroundings.

YOLO stands for "You Only Look Once" and v8 simply means its version 8. Since the start of this project a new version has been launched, YOLOv9. This project will only focus on YOLOv8 as this is the algorithm used. To get started with YOLOv8 you need to:

1. **Install ultralytics.** You can follow their installation guide at *Ultralytics Installation* 2024.
2. **Find a dataset to train YOLOv8 algorithm on.** A good place to find datasets is either on *Open Images Dataset V7* 2024, or *Kaggle, Your Machine Learning and Data Science Community* 2024.
3. **Label images.** On Open images you can download prelabeled images. It is also possible to find prelabeled dataset on kaggle, but you need to search more. The images need a bounding box and a label to each image you want your algorithm to train on. This is because the YOLOv8 algorithm need to know what in the image it is supposed to locate.
4. **Train algorithm.** For a full guide on how to do this see Listing 6.
5. **Use the algorithm.** To see how the algorithm has been implemented in this course please see chapter 4.3.1

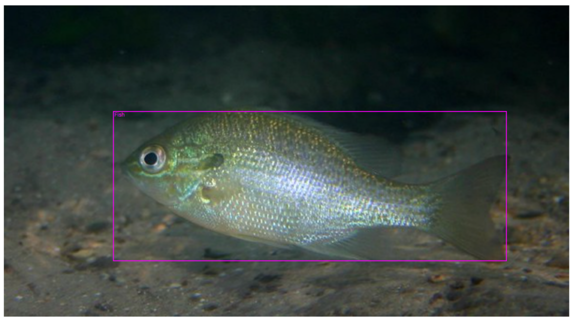
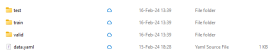


Figure 40: *Example of boundingboxes and label.* 2024

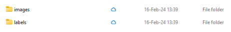
The bounding boxes for training comes in the form of labels in a .txt file. The format of this file is "class" "x_center (normalized)" "y_center (normalized)" "Width of the boanding box (normalized)" "Height of the boanding box (normalized)". An example of this is as follows "0 0.525258 0.540021 0.850797 0.461040". This means that it is class 0 wich might mean salmon, and class 1 might mean trout for example. X_center is 0.525258, Y_center is 0.540021, width is 0.850797 and height is 0.461040. Normalized means it is a ratio of the image. So if the height is the whole image it would be 1 and if it would have no height, it would be 0.

The file structure is important when training on the given data set. You need a folder containing 3 folders called: test, train and valid as well as a data.yaml file. The 3 folders should look the same inside. It should have one folder called images and one folder called labels.



test	16-Feb-24 13:39	File folder	
train	16-Feb-24 13:39	File folder	
valid	16-Feb-24 13:39	File folder	
data.yaml	15-Feb-24 18:28	Yaml Source File	1 KB

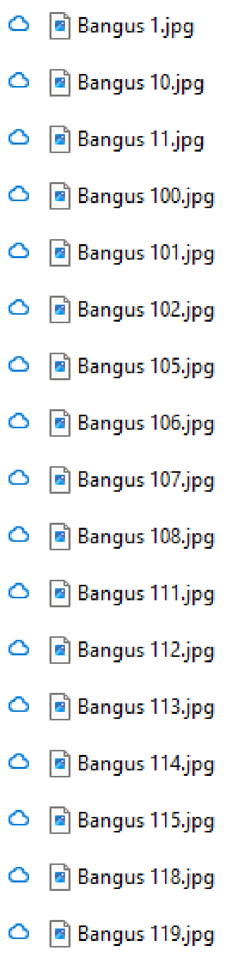
(a) Outer folder



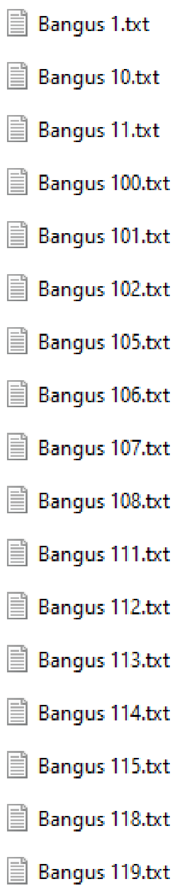
images	16-Feb-24 13:39	File folder
labels	16-Feb-24 13:39	File folder

(b) Inside the 3 folders

The images folder is where the images of your dataset is supposed to be, and the labels folder is the boanding boxes associated with each image. It is important that the labels and images have the same name.


Bangus 1.jpg
Bangus 10.jpg
Bangus 11.jpg
Bangus 100.jpg
Bangus 101.jpg
Bangus 102.jpg
Bangus 105.jpg
Bangus 106.jpg
Bangus 107.jpg
Bangus 108.jpg
Bangus 111.jpg
Bangus 112.jpg
Bangus 113.jpg
Bangus 114.jpg
Bangus 115.jpg
Bangus 118.jpg
Bangus 119.jpg

(a) Images


Bangus 1.txt
Bangus 10.txt
Bangus 11.txt
Bangus 100.txt
Bangus 101.txt
Bangus 102.txt
Bangus 105.txt
Bangus 106.txt
Bangus 107.txt
Bangus 108.txt
Bangus 111.txt
Bangus 112.txt
Bangus 113.txt
Bangus 114.txt
Bangus 115.txt
Bangus 118.txt
Bangus 119.txt

(b) Labels

The dataset should be divided roughly 70-80% training data set and 10-15% each on the validation and testing data set. The training set is what the algorithm will train on. While training it will use the validation set to check how well it is performing. And use the results for further training. The testing data set is used after the training is complete as a test of how well the algorithm is working.

The code for doing the actual training can be found at *Ultralytics Training* 2024. When finished training a model the result is saved as a .pt file which contains the ”weights” of the algorithm. This means the different parameters that is gathered through the training process. To use the model you can load the .pt file into a python script.

```python
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.yaml')  # build a new model from YAML
model = YOLO('yolov8n.pt')  # load a pretrained model (recommended for training)
model = YOLO('yolov8n.yaml').load('yolov8n.pt')  # build from YAML and transfer
    weights

# Train the model
results = model.train(data='coco8.yaml', epochs=100, imgsz=640)
```

Listing 1: Loading and training models

OpenCV

OpenCV (Open computer vision) is the worlds biggest computer vision library. It is open source and contains over 2500 algorithms aimed at real-time applications. It is meant for cross-platform use and offers support for C++, Python, Java on Linux, MacOS, Windows, iOS and Android. It is primarily used for video capture, image processing, and displaying or saving processed images. Four important functions used in the program is:

- **cv2.VideoCapture(0)**
  - **Purpose:** This function initializes the video capture object for reading video frames. The parameter 0 indicates that OpenCV should use the default webcam of the system.
- **cap.read()**
  - **Purpose:** Reads the next frame from the video capture device.
- **cv2.rectangle(frame, top_left, bottom_right, (0, 255, 0), thickness=2)**
  - **Purpose:**Draws a rectangle on an image. The parameters include the image/frame, the top-left and bottom-right coordinates of the rectangle, the color of the rectangle (green in this case, represented as (0, 255, 0) in BGR format), and the line thickness.
- **cv2.imwrite(filename, processed_frame)**
  - **Purpose:** Saves an image to a specified file.

OpenCV 2024

4.2 Previous work

After last years thesis there were three rigs running the same setup. It was using Ubuntu 22.04. As well as ROS2 humble. Last years GitHub Repository can be found here: *Last years Github Repository* 2023. The thesis explaining everything can be found here: *Knudsen and Brandstorp 2023*. The rig had a ROS2 node for each sensor: *Thermometer, barometer, oxygen and salinity.* As well as a node for measuring voltage from the battery, a node for logging all data, and a node for packaging and sending the sensor data. The whole setup was able to launch from a launch script that started all nodes simoultaniously.

4.3 Current state

The system is still running Ubuntu 22.04. It has a ROS2 layout with 11 nodes. These nodes include running sensors, running camera vision algorithm, sending information, and monitoring battery level as well as autoshutdown when battery is too low. New nodes from last year is the cv_algorithm, auto_shutdown and image_client. The whole layout can be found at this years GitHub Repository: *This years Github Repository (Raspberry Pi)* 2024, *This years Github Repository (VIM3)* 2024 .

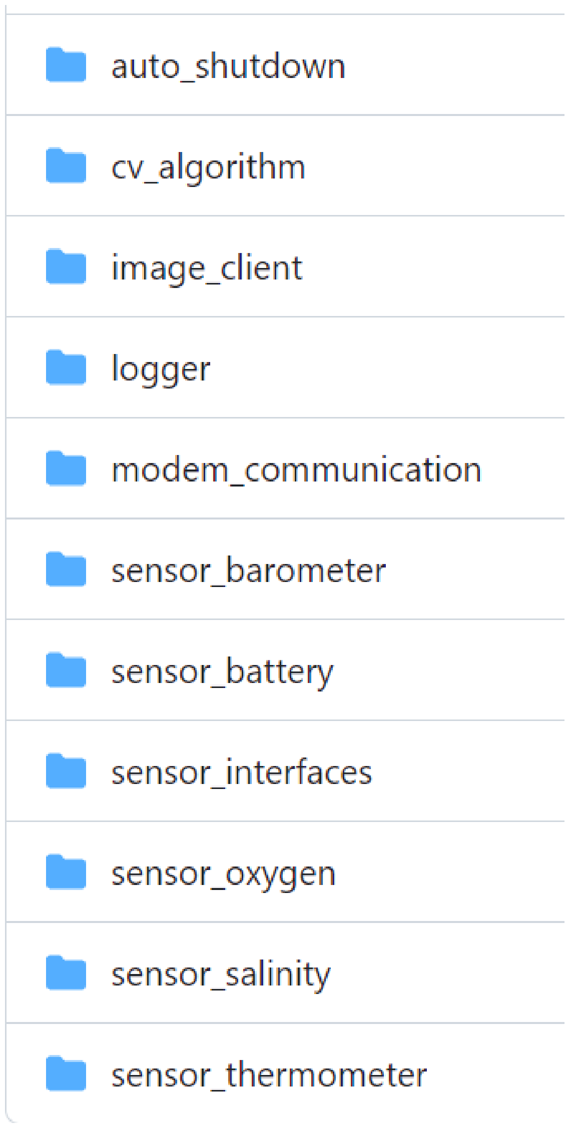

auto_shutdown
cv_algorithm
image_client
logger
modem_communication
sensor_barometer
sensor_battery
sensor_interfaces
sensor_oxygen
sensor_salinity
sensor_thermometer

Figure 43: ROS2 layout

For more in depth explanation of the work done from previous years, you should revert to the bachelor thesis: *Knudsen and Brandstorp 2023* as well as the GitHub repository from the same year: *Last years Github Repository* 2023. This chapter will focus on the new nodes from this year starting with the computer vision module called cv_algorithm. After that, the rest of the new ROS_2 nodes will follow, along with a brief explanation of how the RTC software was implemented. Ending with an explanation of how the new VIM3 was implemented. For a step-by-step tutorial of setting up a SSH client, check the testing tutorial in the PA-folder.

4.3.1 Computer vision module

The computer vision module consists of a python script running the YOLO algorithm and attatch boxes to the correct object. The script consists of 3 main functions:

1. get_squares

This function extracts the bounding box coordinates of detected objects from the results provided by the YOLO model.

- It initializes an empty list outlist.

- It iterates over each detected object's bounding box (results[0].boxes.xyxy), extracting the coordinates (x, y, x2, y2) of the top-left and bottom-right corners.
- These coordinates are stored in outlist as a list of lists, each list representing one detected object's bounding box.

```python
def get_squares(results):
    outlist = []
    print(len(results[0].boxes.xyxy))
    for i in range(len(results[0].boxes.xyxy)):
        x = results[0].boxes.xyxy[i][0].item()
        y = results[0].boxes.xyxy[i][1].item()
        x2 = results[0].boxes.xyxy[i][2].item()
        y2 = results[0].boxes.xyxy[i][3].item()
        outlist.append([x, y, x2, y2])
    return outlist
```

Listing 2: Getting bounding boxes from YOLO-algorithm

2. highlight_objects

This function takes the original video frame and the bounding box coordinates to visually highlight detected objects.

- It loops through each bounding box in squares.
- It uses cv2.rectangle to draw a green rectangle (RGB: (0, 255, 0)) around the detected objects, with a thickness of 2 pixels.

```python
def highlight_objects(frame, squares):
    for square in squares:
        top_left = (int(square[0]), int(square[1]))
        bottom_right = (int(square[2]), int(square[3]))
        cv2.rectangle(frame, top_left, bottom_right, (0, 255, 0), thickness=2)
    return frame
```

Listing 3: Visually highlight the boanding boxes on image

3. main

This is the main function where the script's primary operations are orchestrated:

- **Initialize YOLO Model:** Loads a pre-trained YOLO model from a specified path (model_path).
- **Webcam Initialization:** Attempts to capture video from the default webcam.
- **Real-time Processing Loop:**
  - Continuously reads frames from the webcam.
  - If a frame is successfully read, it predicts objects in the frame using the YOLO model with a confidence threshold of 0.35.
  - Extracts bounding box coordinates using get_squares and highlights them using highlight_objects.
  - Every 10 seconds, saves the processed frame to a specified directory (output_dir) and prints a message to confirm the saved file.
- **Cleanup:** Releases the webcam after the loop ends (either from an error in reading frames or manually stopping the loop).

```python
def main():
    # Output directory
    output_dir = r"C:\Users\Public\Computer_vision_test"

    # Initialize the YOLO model
    model_path = r"C:\Users\Public\Computer_vision_test\yolov8_weights.pt"
    model = YOLO(model_path)

    # Initialize webcam
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("Error: Could not open webcam")
        exit()

    start_time = time.time()
    save_interval = 10  # seconds
    frame_count = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Error: Can't receive frame (stream end?). Exiting ...")
            break

        # Perform detection (assuming direct frame processing is possible)
        results = model.predict(frame, conf=0.35)
        squares = get_squares(results)
        processed_frame = highlight_objects(frame.copy(), squares)

        current_time = time.time()
        if current_time - start_time >= save_interval:
            # Save processed frame with detections
            filename = f"{output_dir}/output_{frame_count}.jpg"
            cv2.imwrite(filename, processed_frame)
            print(f"Saved: {filename}")
            start_time = current_time
            frame_count += 1

    # When everything is done, release the capture
    cap.release()
```

Listing 4: Main loop for cv_algorithm

Training the Algorithm

The algorithm has been trained on facial recognition in order to use it effectively while implementing. For training the website kaggle.com has been used, since it offers more GPU power for training. The code that has been run in the kaggle notebook for training is:

```python
%pip install ultralytics
import ultralytics
ultralytics.checks()
```

Listing 5: Installing the ultralytics library

```
!yolo task=detect mode=train model='/kaggle/working/faces/face_detection/weights/
    best.pt' data='/kaggle/input/face-detection-henrik/Face_detection_B_and_W/data.
    yaml' epochs=50 batch=16 imgsz=640 project=faces name=face_detection
```

Listing 6: Code for training

Change the location of ”model” and ”data” to correct location.

Understanding the training algorithm

An epoch is a complete pass through the entire training dataset. Given that not all data can be passed through the network at once (especially with very large datasets), the data is divided into batches. The number of iterations per epoch is equal to the number of batches necessary to process the entire dataset. So, if you have 1600 training images, and a batch size of 16, it would take 100 iterations to complete one epoch. In general: The more epochs the better result, but the longer time as well. The batch size determines the number of training samples that are used to calculate the gradient and update the model’s weights during a single iteration of the training loop. There is a balance between too small and too large batch sizes. Very large batches might lead to poorer generalization to unseen data. This is a subject of ongoing research, and empirical results often guide the choice of an appropriate batch size.

4.3.2 Auto Shutdown Module

There has been implemented an auto shutdown node. This is done in order to not discharge the battery. There are two important functions for this program.

One called **__init__ (self):** This constructor initializes the node with the name 'BatteryMonitor' and sets up a subscription to the 'battery_data' topic that expects messages of type Battery. The subscription triggers the battery_callback method whenever a new message is published, with a Quality of Service (QoS) setting to buffer the last 10 messages. The reference to self.subscription after its assignment prevents linting tools from flagging it as an unused variable.

The other called **battery_callback(self, msg):** This method is invoked for each Battery message received. It logs the current battery percentage contained in the message. If the battery percentage falls below 20%, the method logs a warning about the low battery and executes a system shutdown command using subprocess.run(['sudo', 'shutdown', '-h', 'now']). This command ensures the system is shut down safely to avoid data loss or hardware damage due to an abrupt power outage.

```python
class BatteryMonitor(Node):
    def __init__(self):
        super().__init__('BatteryMonitor')
        self.subscription = self.create_subscription(
            Battery, 'battery_data', self.battery_callback, 10)
        self.subscription  # prevent unused variable warning

    def battery_callback(self, msg):
        self.get_logger().info(f'Received battery data: {msg.battery_percent}%')
        if msg.battery_percent < 20.0:
            self.get_logger().info('Battery low, shutting down...')
            # Command to shut down the Raspberry Pi safely
            subprocess.run(['sudo', 'shutdown', '-h', 'now'])
```

Listing 7: BatteryMonitor node class for auto shutdown

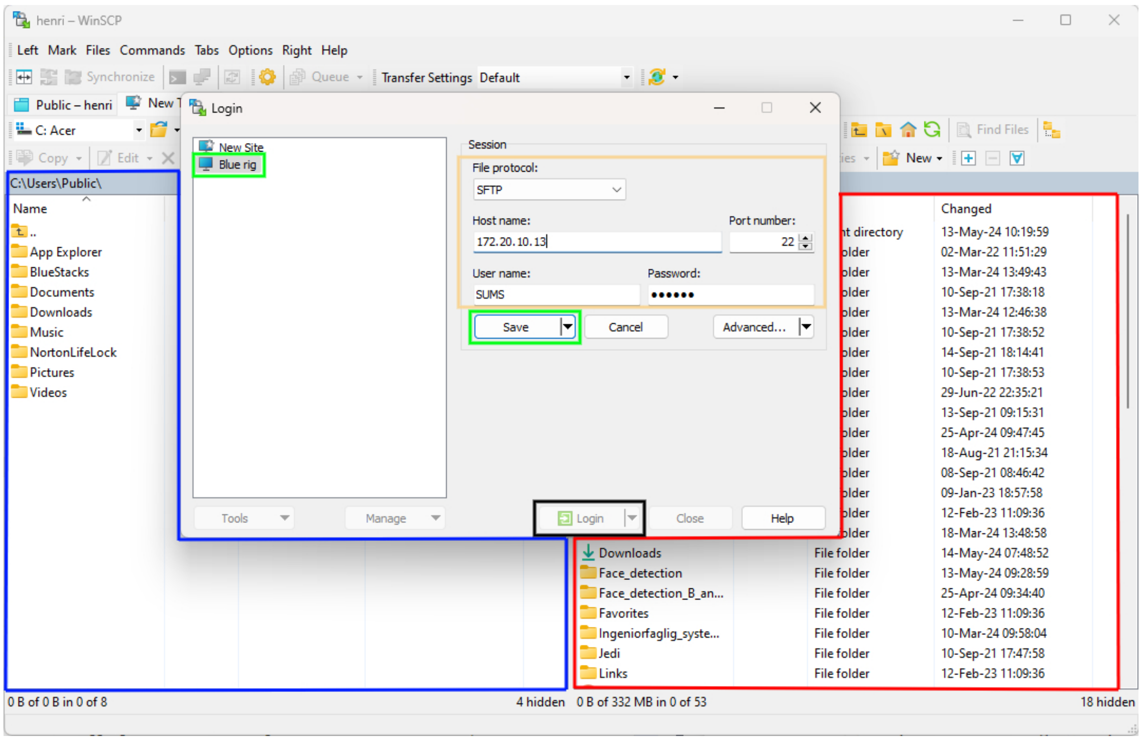
The auto shutdown needs administration rights to run and will therfore ask for password before shutting down. This does not work since the program is supposed to happen automatic. To fix this problem the shutdown call has been added to the sudoers exception list. This is done by opening the sudo visudo file and then entering shutdown as an exception:

```
$ sudo visudo
your_username ALL=(ALL) NOPASSWD: /sbin/shutdown
```

In order to make the auto shutdown happen administration rights is needed. This can be achieved with any sudo call before starting the node, for example sudo apt-get update. After you have entered the password once the auto shutdown script will run automatically without asking for password.

4.3.3 Implementation through WinSCP

WinSCP is a program that allows drag and drop of files between the rigs and your PC. It has been used as a very helpful tool when implementing new things on the SBC's. Especially the computer vision algorithm since the rigs are running a headless OS, meaning there is no desktop, only a terminal. Through WinSCP images has easily been transferred between the SBC and PC.



File protocol: SFTP
Host name: 172.20.10.13
Port number: 22
User name: SUMS
Password: ••••••

```
C:\Users\Public\
Name
..
App Explorer
BlueStacks
Documents
Downloads
Music
NortonLifeLock
Pictures
Videos
0 B of 0 B in 0 of 8                4 hidden
```

```
                          ...t directory   Changed
                          ...older         13-May-24 10:19:59
                          ...older         02-Mar-22 11:51:29
                          ...older         13-Mar-24 13:49:43
                          ...older         10-Sep-21 17:38:18
                          ...older         13-Mar-24 12:46:38
                          ...older         10-Sep-21 17:38:52
                          ...older         14-Sep-21 18:14:41
                          ...older         10-Sep-21 17:38:53
                          ...older         29-Jun-22 22:35:21
                          ...older         13-Sep-21 09:15:31
                          ...older         25-Apr-24 09:47:45
                          ...older         18-Aug-21 21:15:34
                          ...older         08-Sep-21 08:46:42
                          ...older         09-Jan-23 18:57:58
                          ...older         12-Feb-23 11:09:36
                          ...older         18-Mar-24 13:48:58
Downloads                 File folder      14-May-24 07:48:52
Face_detection            File folder      13-May-24 09:28:59
Face_detection_B_an...    File folder      25-Apr-24 09:34:40
Favorites                 File folder      12-Feb-23 11:09:36
Ingeniorfaglig_syste...   File folder      10-Mar-24 09:58:04
Jedi                      File folder      10-Sep-21 17:47:58
Links                     File folder      12-Feb-23 11:09:36
0 B of 332 MB in 0 of 53                   18 hidden
```

Figure 44: WinSCP

Figure 44 shows the outlay of WinSCP. The blue box will show the file system of your computer while the red box will show the file system of the rig you are connected to. In order to connect the rig it first has to be connected to internet, see 4.3.6. Then you fill out information shown inside the yellow box. File protocol should be SFTP. Host name is the IP address of the rig you are connected to. Port number should be 22. User name and password is the same as for the login of the rigs. So for the black and blue rig the username is: "SUMS" and password would be: "123456". For the red rig the user name is "khadas" and the password is "123456". You can save the information for each rig as shown in green. When ready you connect by pressing the login shown in black.

4.3.4 Running multiple ROS2_nodes

In order to run multiple ROS2_nodes simoultaniously there are several ways of doing it. Two different scripts have been made that runs multiple nodes. One runs all the nodes, and the other runs all the nodes except for the auto_shutdown, because this node has not been tested and calibrated with battery percentage. How these scripts can be run, how the launch files work and how to run different specific nodes together will now be explained:

In order to run all nodes simoultaniously you can use this command:

```
    $ cd bachelor-2024/
    $ source install/setup.bash
    $ ros2 launch launch/sensor.launch.py
```

Remember that in order for the auto_shutdown mode to run a sudo command followed by entering password needs to be ran before starting the nodes. This can be done for example with: sudo apt-get update

To run all nodes except for the auto_shutdown node. This command can be used:

```
    $ cd bachelor-2024/
    $ source install/setup.bash
    $ ros2 launch launch/sensor.launch.without.shutdown.py
```

To run a specific set of nodes you can use this command:

```
    $ cd bachelor-2024/
    $ source install/setup.bash
    $ ros2 run packagename node1 &
      ros2 run packagename node2 &
      wait
```

A detailed explanation of how the launch files are set up and how to include a new node into the launch file will follow. The launch file can be found at the github_repository under launch: *This years Github Repository (Raspberry Pi)* 2024, *This years Github Repository (VIM3)* 2024

Importing necessary modules for the ROS 2 launch file:

```
from launch import LaunchDescription
from launch_ros.actions import Node
import random
```

- `LaunchDescription` from `launch`: Used to create a description of the launch, essentially a script that ROS 2 will execute.
- `Node` from `launch_ros.actions`: Represents a ROS node, which is a process that performs computations.
- `random`: A standard Python library used here to generate random numbers.

Configuration Variables

```
sample_time = 10.0              # Sample time in seconds (float)
n_sensors = 5                   # How many sensors that are in use
transfer_delay = 6.0            # How long transferring data takes in seconds(float)
modem_IP = '192.168.42.195'     # IP for the modem
modem_port = 1100               # Port for the modem
precision = 2
```

These lines define various settings:

- `sample_time`: The interval at which sensors sample data.
- `n_sensors`: Total number of sensors involved in the system.
- `transfer_delay`: Time taken to transfer data, likely used in communication configurations.
- `modem_IP` and `modem_port`: Networking details for modem communication.
- `precision`: Related to data precision in logging or data handling.

Random Time Generation

```
# Random generator to avoid deadlocks
lower_bounds = [2, 4]   # Seconds
upper_bounds = [4, 6]   # Seconds
random_bounds = [
    random.randrange(lower_bounds[0]*1000, lower_bounds[1]*1000, 200),
    random.randrange(upper_bounds[0]*1000, upper_bounds[1]*1000, 200)
]
```

These lines implement a mechanism to generate random timing intervals (in milliseconds) to avoid potential deadlocks in operations:

- `lower_bounds` and `upper_bounds` define the minimum and maximum intervals.
- `random_bounds` generates random numbers within these specified ranges, converted to milliseconds.

Node Configuration and Implementation

```
def generate_launch_description():
    return LaunchDescription([
        Node(
            package='sensor_oxygen',
            namespace='sensors',
            executable='oxygen_publisher',
            name='oxygen',
            parameters=[{'sample_time': sample_time}]
        ),
        Node(
            package='sensor_thermometer',
            namespace='sensors',
            executable='thermometer_publisher',
            name='thermometer',
            parameters=[{'sample_time': sample_time}]
        ),
])
```

Each `Node` in the launch description represents a ROS node:

- `package`: The ROS package containing the node's executable.
- `namespace`: A ROS namespace under which the node operates, helps in organizing nodes.
- `executable`: The executable file within the package to run.
- `name`: A unique identifier for the node in the system.
- `parameters`: Configuration parameters the node needs, here defining the `sample_time`.

Adding a new node

To add a new node, such as a pressure sensor, append another `Node` instantiation to the list in the `generate_launch_description` function:

```
Node(
    package='sensor_pressure',
    namespace='sensors',
    executable='pressure_publisher',
    name='pressure',
    parameters=[{'sample_time': sample_time}]
),
```

More information is found at Open_Robotics 2024

4.3.5 Implementation of the VIM3

To set up the VIM3, several software-related configurations were performed to ensure optimal performance. Initially, the VIM3 came preinstalled with Ubuntu 22.04, eliminating the need to flash the Ubuntu OS onto a SD card. The fist step involved logging into the SBC using the default username and password: "khadas". Then, a system update was conducted using an Ethernet cable for internet connectivity. The following commands will install any new software versions and is recommended to be executed regularly for system maintenance:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

WiFi connection
To establish a WiFi connection, one can follow the guide provided by Khadas regarding the VIM3 (*WiFi* 2024). This step by step guide requires Network Manager preinstalled to access the Network Manager Command-line Interface (NCMLI) commands.

Begin by installing Network Manager if not already done:

```
$ sudo apt install -y network-manager
$ sudo systemctl start NetworkManager.service
```

Next, scan for available WiFi networks using nmcli:

```
$ nmcli d wifi list
```

Then, establish a WiFi connection using the following command, replacing "your_ssid" and "your_password" with your SSID and password:

```
$ sudo nmcli d wifi connect your_ssid password your_password
```

It is recommended to use mobile WiFi for simpler connection due to fewer security measures.

Finally, verify WiFi connection by using the following command:

```
$ ip a
```

This command will display the SBC's IP address if the WiFi connection is established. The command is equivalent to "IP address show". If connected, the IP address will show under "wlan0" next to "inet", otherwise it will display "ff:ff:ff:ff:ff:ff". This command is also useful for establishing a ssh connection.

In order to establish automatic WiFi connections on startup, an additional package is required. This package is called netplan.io and is a utility for easy configuration of networks on a linux-based system. First you need to download the package by typing the following command:

```
$ sudo apt install netplan.io
```

Listing 8: Installation of netplan.io

After the package is downloaded you will find a new directory, /etc/netplan, which you should navigate to in order to create the configuration file, using the following commands:

```
$ cd /etc/netplan
$ sudo nano 50-cloud-init.yaml
```

Listing 9: Navigating to and creating the network configuration

When the .yaml file has been created you need to follow the steps in section 4.3.6 in order to set up automatic WiFi connection on startup.

Downloading ROS 2 packages
The next step in this process involves downloading ROS 2 packages. The installation of ROS 2

was done by following the guide for installing the Debian packages for ROS 2 Humble Hawksbill provided by Open Robotics (*Ubuntu (Debian packages)* 2024). ROS-Base is the utilized version of ROS, which is the bare bones version containing only the necessary packages for a headless computer making it suitable for the VIM3.

The steps begin with setting up sources:

```
    $ sudo apt install software-properties-common
    $ sudo add-apt-repository universe
    $ sudo apt update && sudo apt install curl -y
    $ sudo curl -sSL https://raw.GitHubusercontent.com/ros/rosdistro/
       ↪ master/ros.key -o /usr/share/keyrings/ros-archive-keyring.
       ↪ gpg
```

Next, add the repository to the sources list:

```
    $ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share
       ↪ /keyrings/ros-archive-keyring.gpg] http://packages.ros.org/
       ↪ ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME)
       ↪ main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/
       ↪ null
```

After setting up the repositories, update and build the apt repository caches:

```
    $ sudo apt update
    $ sudo apt upgrade
```

Then, download the ROS - Base and development tools:

```
    $ sudo apt install ros-humble-ros-base
    $ sudo apt install ros-dev-tools
```

Finally, source the setup script by replacing ".bash" with your shell if you're not using bash.

```
    $ source /opt/ros/humble/setup.bash
```

Downloading essential packages and libraries
To run the program on the VIM3, additional package downloads is required. These external libraries are essential for ensuring optimal performance. For using the Atlas Scientific EZO sensors, the SBC needs to install both python-smbus and I2C-tools. The smbus package facilitates communication between the python3 code and the I2C devices, while the I2C-tools package provides command-line tools for I2C related operations.

To download these packages, run the following commands:

```
    $ sudo apt-get install python3-smbus
    $ sudo apt-get install i2c-tools
```

The remaining essential packages are downloaded using the Python packet manager (PIP):

```
    $ sudo apt install python3-pip
```

Next, install Unetpy, a Python package that provides interaction with modems running UnetStack:

```
    $ pip install unetpy
```

Finally, install Adafruit_GPIO, a package that establishes communication with the ADC by utilizing a specific library formed by their manufacturer:

```
    $ pip install Adafruit-GPIO
```

I2C configurations
To be able to use the I2C-pins on the VIM3, I2C needs to be enabled. This can be done by editing the device tree overlays' configuration file and adding an I2C node to fdt_overlays.

To edit the configuration file, use the following command:

```
$ nano /boot/dtb/amlogic/kVIM3.dtb.overlay.env
```

Then, add the I2C node to the fdt_overlays:

```
$ fdt_overlays=i2c3
```

Finally, save and exit the file and reboot the SBC for the changes to take effect.

Fetching the GitHub files
To use the program on the VIM3, adjustments to the GitHub files were necessary. Unlike the Raspberry Pi, which uses the I2C bus 1, the VIM3 uses I2C bus 3 for I2C communication. Since the program was designed for the Raspberry Pi, adjustments were made in the program to change all references from I2C bus 1 to 3, thus making it compatible with the VIM3. These following instructions for downloading and running the GitHub files can also be found in the README.md file in the GitHub repository.

Begin by fetching the files from the GitHub repository:

```
$ git clone https://GitHub.com/ErjokAguto/SUMS
```

Next, build all the packages with init.py:

```
$ python3 init.py
```

Then, source the setup files:

```
$ . install/setup.bash
```

Finally, launch all the nodes:

```
$ ros2 launch launch/sensor.launch.py
```

Automatic launch
To automatically start the ROS 2 program when booting up the VIM3, utilize the system and service manager (systemd). The systemd service will launch at startup and run every node in the SUMS workspace, making the sensing rig collect data the moment it is powered. This following step-by-step guide will set up automatic launch:

Create a systemd service unit file:

```
$ sudo nano /etc/systemd/system/ros2_SUMS_startup.service
```

Edit the service unit file and insert the following lines. Make sure you insert the correct username in the file

```
[Unit]
Description=ROS2 SUMS automatic launch service.
After=network.target

[Service]
ExecStart=/bin/bash -c "source /home/<user>/SUMS/install/setup.bash &&
    ↪ ros2 launch launch/sensor.launch.py"
WorkingDirectory=/home/<user>/SUMS
StandardOutput=syslog
StandardError=syslog
```

```
SyslogIdentifier=ros2_SUMS_startup
Restart=always
User=<user>

[Install]
WantedBy=multi-user.target
```
Listing 10: Editing the service unit file

Save and close the file before enabling and starting the service:

```
    $ sudo systemctl enable ros2_SUMS_startup.service
    $ sudo systemctl start ros2_SUMS_startup.service
```

To verify the status of each running node after the service startup, use the following commands to display all active ROS2 nodes:

```
    $ ros2 node list
```

Tuning the RTC
To ensure correct timekeepig, the built-in RTC in the VIM3 required configuring. The RTC needed to be tuned into the correct timezone.

To change the systems timezone to Europe/Oslo, use the following command:

```
    $ sudo timedatectl set-timezone Europe/Oslo
```

To verify the changes, use the following command:

```
    $ timedatectl
```

4.3.6 Network configuration

It is very important for the Raspberry Pi and Khadas VIM to have a network connection during development. In order to download, install, update and upgrade the computers, it needs to be online so it can grab the necessary packs and files. With Ubuntu, setting up and changing the network configuration is quite easy. The configuration file is a .yaml-file, which you will have to navigate to. Then you will have to open the 50-cloud-init.yaml-file in the editor, like shown in listing 11.

```
$ cd /etc/netplan
$ ls
50-cloud-init.yaml
$ sudo nano 50-cloud-init.yaml
```
Listing 11: Navigating to network configuration

The code should look something like in listing 12. In the file there are two sections; one for an ethernet connection and one for WiFi. To establish an ethernet connection, it is pretty straight forward, as both the RPi and Khadas VIM has an ethernet port. If you are looking to establish a WiFi connection you will have to be changed in order to fit the desired network, meaning you will have to change the "network name" and "password" to the network you want to establish a connection with. If you want the rigs to work on multiple networks, you will have to add multiple "network name" and "password" sections. If this is done, the computer will prioritize trying to connect with the first network, and if it doesn't work, it will try the next in line.

```
network:
    renderer: networkd
    ethernets:
        eth0:
```

```
#           Static IP-address for ethernet
            addresses:
                - 192.168.0.69/24 # wanted static IP-address
#           Host IP-address for ethernet
            routes:
                - to: default
                  via: 192.168.0.1
            nameservers:
                addresses: [8.8.8.8, 1.1.1.1]
    version: 2
    wifis:
        wlan0:
            dhcp4: true
            optional: true
            access-points:
                "<WIFI-NAME>":
                    password: "<WIFI-PASSWORD>"
#           Static IP-address for WiFi
            addresses:
                -192.168.121.50/24 # wanted static IP-address
            nameservers:
                addresses: [4.2.2.2, 8.8.8.8]
#           Host IP-address for WiFi
            routes:
                - to: default
                  via: 192.168.121.99
```
Listing 12: Inside the network configuration file

It will also be beneficial to set a static IP-address for the computers. Whenever you establish an internet connection, either via ethernet or WiFi, an IP-address will be given to the device. However, this IP-address might change every time you connect to the internet, which isn't ideal for the development and use of the rigs. This means you would have to log on to the computer directly and grab the IP-address if you would want to access it. For a sensor rig that is meant to be sealed shut, it makes it quite difficult to work with it. Setting a static IP-address will prioritize giving a specific IP-address to the device, which eliminates the problem of not knowing the IP-address after connecting to a network. To set a static IP-address, you will have to access the same file as previously, 50-cloud-init.yaml. When you are there, there are sections under both ethernets and WiFis called *addresses*, *nameservers* and *routes*. The *nameservers* addresses should be as shown in listing 12. For the next part you will have to find the routers IP-address. This can easily be done by navigating to home on the computer, and writing the command for grabbing the routers IP-address, like shown in listing 13. Note that in order to do this, you will have to be connected to the network.

```
$ ip r
```
Listing 13: Finding host device IP-address

Navigate back to the 50-cloud-init.yaml-file, then change the IP-address under *addresses*, to the desired address. This address have to share the first three numbers with the routers IP-address, for example if the routers IP-address is 192.168.90.1, you will need to set a static IP-address as 192.168.90.x/24, where x can be any number from 1 to 254, but can't be the same as the routers IP-address, and 24 is the subnet mask (255.255.255.0), which may also vary based on your provider.

If a mobile hotspot is being used, the host IP-address may change from time to time, so setting a static IP-address might be moot. It works and is proof of concept, though from a development point of view, it would be beneficial to be connected to a router when working on the rigs.

4.3.7 Configuring RTC on Raspberry Pi

Configuring the RTC on Raspberry Pi proved more challenging than first anticipated. The following tutorial from the raspberry pi forums ended up providing a solution to the issues: **RTC-tutorial**.

Prerequisites

1. Working internet connection
2. An up to date Ubuntu Server

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Configuring the RTC

- Check the RTC drivers are installed. This should run without an error.

```
$ sudo modprobe rtc-ds1307
```

- Create the new RTC interface

```
$ sudo bash
$ echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
$ exit
```

- Set the correct time to RTC

```
$ sudo hwclock -r
$ date
$ sudo hwclock -w
```

- Load RTC drivers at boot up, navigate to the following file and add the following line.

```
$ cd /etc/modules-load.d
$ sudo nano modules.conf
```

```
rtc-1307
```

Listing 14: Inside the module.conf file

- Create interface and sync the time at boot. Create rtc file and add the following lines:

```
$ cd /etc/
$ sudo nano rtc
```

```
#!/bin/bash
#this is the script to be executed at boot to create the rtc interface
echo 'ds1307 0x68' | sudo tee /sys/class/i2c-adapter/i2c-1/new_device
sudo hwclock -s
```

Listing 15: Inside the rtc file

- Make the script executable by setting correct file permissions

```
$ sudo chmod 755 rtc
$ sudo chmod +x rtc
```

- Create and enable rtc.service file in system folder

```
$ cd /etc/systemd/system/
$ sudo nano rtc.service
```

```
#this calls the rtc script at boot to create the rtc interface new_device
[Unit]
Description=RTC Clock
Before=cloud-init-local.service
Requires=systemd-modules-load.service
After=systemd-modules-load.service
[Type]
Type=oneshot
[Service]
ExecStart=/etc/rtc
[Install]
WantedBy=multi-user.target
```

Listing 16: Inside the rtc.service file

```
$ sudo systemctl start rtc.service
$ sudo systemctl enable rtc.service
```

If this outputs something like the following, you have succesfully configured the RTC.

```
"Created symlink /etc/systemd/system/multi-user.target.wants/rtc.service ->
    /etc/systemd/system/rtc.service."
```

Final test

- Reboot and check system time and RTC time.

```
$ reboot
$ date
$ sudo hwclock -r
```

4.4 Further work

Computer vision
The implemented computer vision algorithm is as of now in a state of proof of concept. This means it needs a lot of work in the future in order to make it appropriately working for some specific wildlife monitoring task. The ROS_2 module is working, and the camera recognizes objects in images that are stored locally every 10 seconds. If there has been some recognition, and if requested, information from the camera is sent through the modem. The algorithm is trained on facial recognition in order to better test the data pipeline during integration and development. A new training of the **object recognition** algorithm is therefore needed for each wildlife species that one wishes to be monitored. There is also the need for developing a **trigger** for the camera to start, in order to save battery and not have the camera running at all times. Lastly there needs to be discussed **what to do with the data.** For example, should it be saved, or only sent through the modem once there is a pull request for data.

VIM 3
To enhance the performance of the VIM3, it's essential to effectively utilize the additional processing power. Featuring an unused NPU chip, this becomes a huge area of improvement for future developers given the many applications such a chip can have. Furthermore, considering the available processing power and a 128 GB SD card, adjustments to the GitHub files/code are necessary to optimize the performance of the sensing rig.

Others
Other things that need work is: Testing and calibrating the battery_node and the auto_shutdown node. The auto_shutdown node works, but it has only been tested when the Raspberry Pi is connected through USB and the battery_percentage is off. This needs to be tested with the battery that is going to be used and confirm that the battery_percentage is showing correct battery_level.

Cleaning up all code. The code from this years thesis has been added on top of last years thesis and are not written in the same style and may cause confusion for later development. The main things that needs to be fixed are a set way of implementing nodes. In the setup.py file there are no standard way of naming nodes. From last years thesis the nodes contain two python scripts. One containing the actual code and one main file that is running the code from the other script. In this years thesis all code and execution are written in the same script.

5 Tests

One crucial step in this process involves conducting various tests to verify and document the performance of the sensing rigs. The main motivation for these tests is to establish proof of concept, particularly due to the demanding integration of the camera, which introduces both image processing and transmission over the acoustic modem. Other key motivations includes validation of performance under various conditions and identification of issues and areas of improvement.

NOTE: A full tutorial on how to perform tests can be found in the PA-folder. The tutorial will be referred to in this section.

5.1 Preliminary testing

In the early stages of the project, the main focus was getting familiarized with the system. To gain an understanding of how the system worked, as well as how to navigate the workspace in the Ubuntu OS. Before testing, ensuring a sealed rig, and starting sensor readings were both necessary (Section 1.2 and 1.3 of the Testing Tutorial). At this early stage, the group did not have the experience to connect via SSH, and a camera module was not yet implemented.

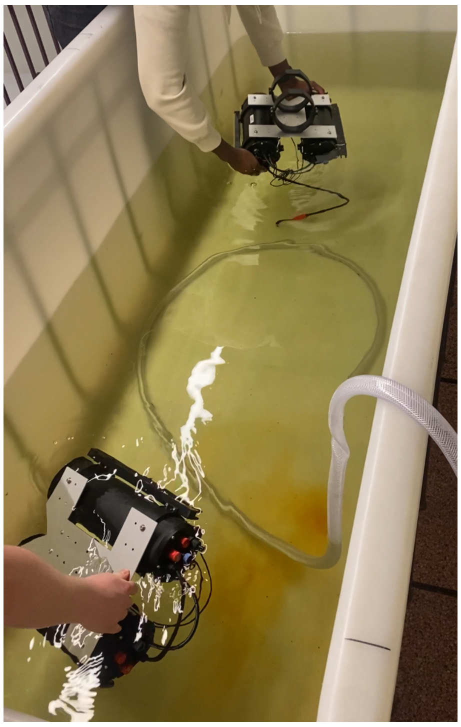
Once the sensor were running and the rig was sealed, the test could take place. In communication with Stefano Bertelli, the pool in Forsøkshallen (D0050) was filled with water, for testing to take place on the 7th of Feb. The quality of sensor readings were not of major concern for the group for this test, but rather just identifying that there were in fact readings and how to locate them. What was already known from previous projects was that the current readings were off, and displayed only 0.00 mA.

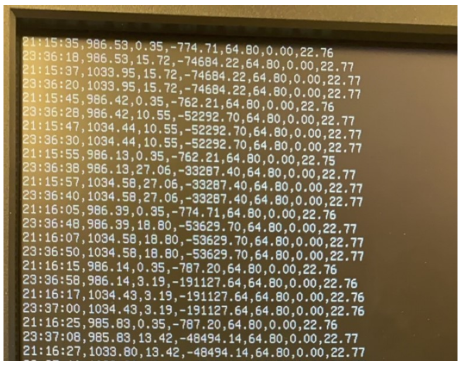
5.2 Final tests

To finalise the project, a field test was planned to evaluate the performance of the rigs in real-world conditions. For a field test to be conducted, cooperation with another group that was responsible for communication between modems, was necessary. However, despite the groups best efforts, and with the aid of supervisor Damiano, an integration of projects with the necessary group did not materialize.
Due to this, the final test was conducted inside, at the university, with the purpose of highlighting the progress made during this years project. The test was conducted on Friday 10Th of May in D0050 (Forsøkshallen).

5.2.1 Objectives

The objective of the final test is to display improvements made to the rigs.


(a) The rigs being submerged in the pool for preliminary testing



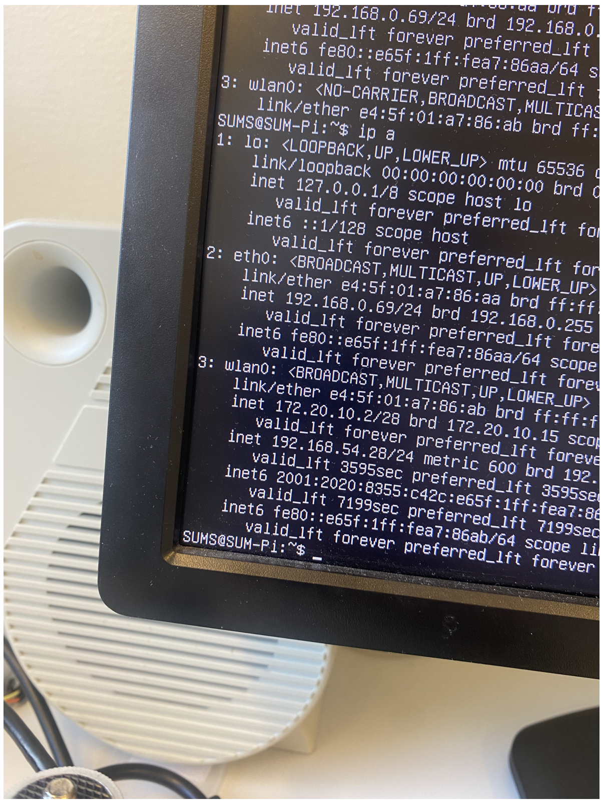
```
21:15:35,986.53,0.35,-774.71,64.80,0.00,22.76
23:36:18,986.53,15.72,-74684.22,64.80,0.00,22.77
21:15:37,1033.95,15.72,-74684.22,64.80,0.00,22.77
23:36:20,1033.95,15.72,-74684.22,64.80,0.00,22.77
21:15:45,986.42,0.35,-762.21,64.80,0.00,22.76
23:36:28,986.42,10.55,-52292.70,64.80,0.00,22.77
21:15:47,1034.44,10.55,-52292.70,64.80,0.00,22.77
23:36:30,1034.44,10.55,-52292.70,64.80,0.00,22.77
21:15:55,986.13,0.35,-762.21,64.80,0.00,22.75
23:36:38,986.13,27.06,-33287.40,64.80,0.00,22.77
21:15:57,1034.58,27.06,-33287.40,64.80,0.00,22.77
23:36:40,1034.58,27.06,-33287.40,64.80,0.00,22.77
21:16:05,986.39,0.35,-774.71,64.80,0.00,22.76
23:36:48,986.39,18.80,-53629.70,64.80,0.00,22.77
21:16:07,1034.58,18.80,-53629.70,64.80,0.00,22.77
23:36:50,1034.58,18.80,-53629.70,64.80,0.00,22.77
21:16:15,986.14,0.35,-787.20,64.80,0.00,22.76
23:36:58,986.14,3.19,-191127.64,64.80,0.00,22.76
21:16:17,1034.43,3.19,-191127.64,64.80,0.00,22.76
23:37:00,1034.43,3.19,-191127.64,64.80,0.00,22.76
21:16:25,985.83,0.35,-787.20,64.80,0.00,22.76
23:37:08,985.83,13.42,-48494.14,64.80,0.00,22.77
21:16:27,1033.80,13.42,-48494.14,64.80,0.00,22.77
```

(b) Results after preliminary testing

- **Collect real-time sensor reading** - The success-criteria will be determined by proper time stamping and sensor readings when gathering the data.
- **Capture underwater photos at set intervals** - The success-criteria will be determined by distinguishable pictures taken in the pool.
- **Use Computer Vision to recognize human faces** - The success-criteria will be determined by green boxes outlining faces when reviewing the images after testing.
- **Evaluate waterproofing and durability of rigs** - The success-criteria will be determined by any leaks or damage to any part of the rigs.
- **Operate the rigs via SSH** - The use of SSH during final tests will verify the functionality.

5.2.2 Execution

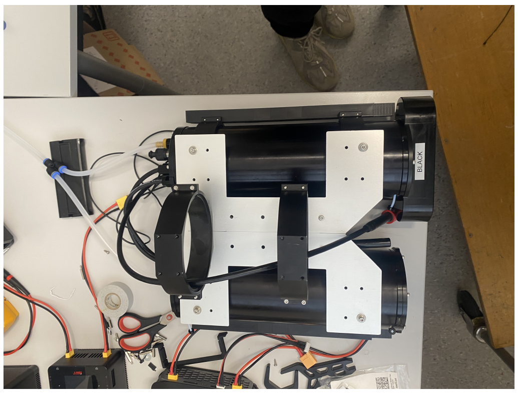
1. **Connecting the rigs to mobile hotspot**
   To be able to perform the tests, SSH connection needed to be established. Section 1.1 of the testing tutorial covers how to set up the network configuration of the rigs. The computer was connected to the same hotspot, and the rigs were all accessible via laptop.



```
    inet 192.168.0.69/24 brd 192.168.0.
       valid_lft forever preferred_lft
    inet6 fe80::e65f:1ff:fea7:86aa/64 s
       valid_lft forever preferred_lft
3: wlan0: <NO-CARRIER,BROADCAST,MULTICAS
    link/ether e4:5f:01:a7:86:ab brd ff:
SUMS@SUM-Pi:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536
    link/loopback 00:00:00:00:00:00 brd
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft fo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft fo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP>
    link/ether e4:5f:01:a7:86:aa brd ff:ff
    inet 192.168.0.69/24 brd 192.168.0.255
       valid_lft forever preferred_lft fore
    inet6 fe80::e65f:1ff:fea7:86aa/64 scope
       valid_lft forever preferred_lft forev
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP>
    link/ether e4:5f:01:a7:86:ab brd ff:ff:f
    inet 172.20.10.2/28 brd 172.20.10.15 scop
       valid_lft forever preferred_lft foreve
    inet 192.168.54.28/24 metric 600 brd 192.
       valid_lft 3595sec preferred_lft 3595se
    inet6 2001:2020:8355:c42c:e65f:1ff:fea7:8
       valid_lft 7199sec preferred_lft 7199sec
    inet6 fe80::e65f:1ff:fea7:86ab/64 scope li
       valid_lft forever preferred_lft forever
SUMS@SUM-Pi:~$ _
```

Figure 46: IP-address for blue-rig final test

2. **Preparing rigs for submersion into water**
   Before the rigs could be submerged into the testing pool, a vacuum test was performed. The vacuum test is covered in detail in section 1.3 of the testing tutorial. Additionally, the acoustic modems were connected to the rigs for a more realistic approach to the final test. The modems were not of any use in the final test, but rather to resemble the operational conditions of the rigs.



(a) Vacuum test performed on black rig

(b) All rigs ready for submersion

3. **Run ROS2 nodes**
   When performing the test, the sensor readings and camera needed to begin before submerging the rigs into water, as the hotspot connection is lost when submerged. Due to this, setting up sensor interfaces and running ROS2 nodes needed to be done just before the test occurred. This was done by:

   - Navigate to the bachelor-2024 workspace

```
$ cd bachelor-2024/
```

- Setting up the sensor interfaces:

```
$ source install/setup.bash
```

- Run the ROS2 nodes

```
$ ros2 launch launch/sensor.launch.without.shutdown.py
```

Note that the launch command is without shutdown module. This is because the battery percentage module is not fine tuned, and can cause the system to shut down without there actually being any risk of damaging the battery. By running the nodes the following way, you will include the auto-shutdown module.

```
$ ros2 launch launch/sensor.launch.py
```

4. **Perform tests**
To validate the objectives for the final tests, different aspects of the rigs had to be tested.

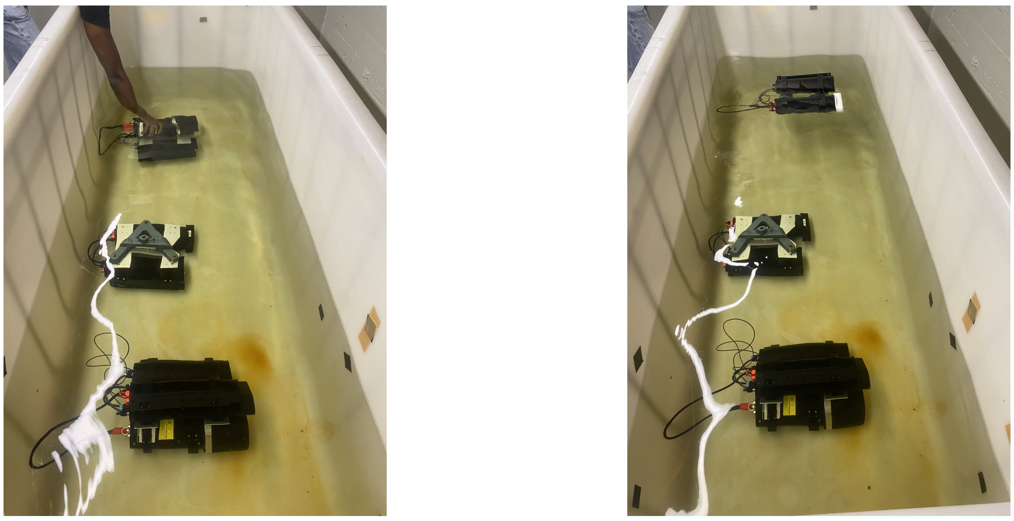
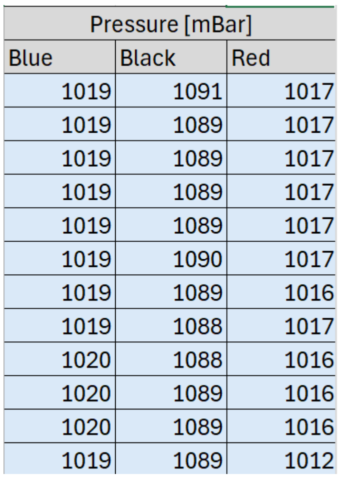


Figure 48: Submerging the rigs into the pool for final tests

- All rigs were submerged underwater to evaluate waterproofing and durability of the rigs.
- The black and blue rigs both took pictures underwater to assess the capability and quality of underwater photos. A Coke bottle was placed in front of the camera lens at different distances, to assess the quality both close range and further away. In addition to this, a hand-watch was placed in front of the camera to see how detailed the pictures were. Being able to read the time would deem the result good.
- The red rig was used to evaluate the computer vision algorithm to recognize human faces. This test was not performed underwater. The group members that conducted the final test were the subjects for this test.

5.2.3 Results

Real-time sensor readings
The implementation of the DS3231 RTC worked as intended, with both rigs using this achieving real-time collection. The VIM, which has an integrated RTC, did not manage proper time stamping. The RTC batteries that were ordered, did not arrive in time for testing, which means that this part of the test went as expected. The following test results are from two minutes of testing, with intervals of ten seconds. It is important to once again note that the quality of data was not a focus area for this years project, but rather just collecting any data using the sensors.



Pressure [mBar]		
Blue	Black	Red
1019	1091	1017
1019	1089	1017
1019	1089	1017
1019	1089	1017
1019	1089	1017
1019	1090	1017
1019	1089	1016
1019	1088	1017
1020	1088	1016
1020	1089	1016
1020	1089	1016
1019	1089	1012

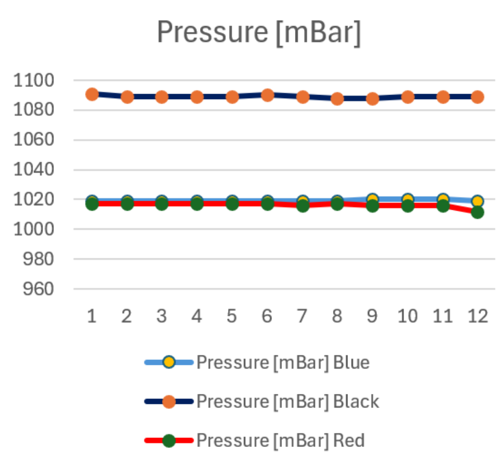
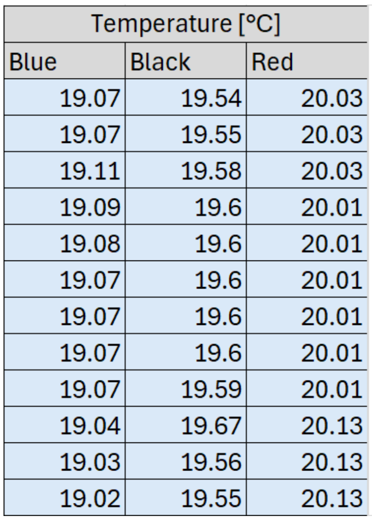


Figure 49: Pressure test results

The expected average atmospheric pressure is 1013 mBar. The red rig was tested on land and showed a pressure reading between 1012-1017. According to expected result, this is showing correct pressure readings.
The blue and the black rig was tested submerged in the testing pool. By using the formula: $P = \rho \cdot g \cdot h$ The result should be $1013mBar + 49mBar = 1062mBar$. From the results it is clear that the blue rig is measuring too low pressure and the black rig is measuring too high pressure. This indicates that the calibration of the pressure sensor needs to be addressed.



Temperature [°C]		
Blue	Black	Red
19.07	19.54	20.03
19.07	19.55	20.03
19.11	19.58	20.03
19.09	19.6	20.01
19.08	19.6	20.01
19.07	19.6	20.01
19.07	19.6	20.01
19.07	19.6	20.01
19.07	19.59	20.01
19.04	19.67	20.13
19.03	19.56	20.13
19.02	19.55	20.13

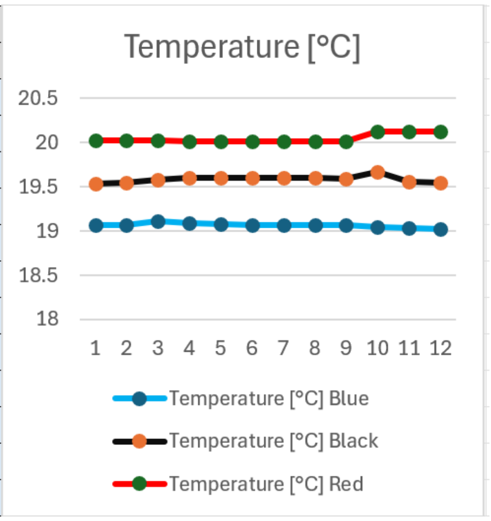
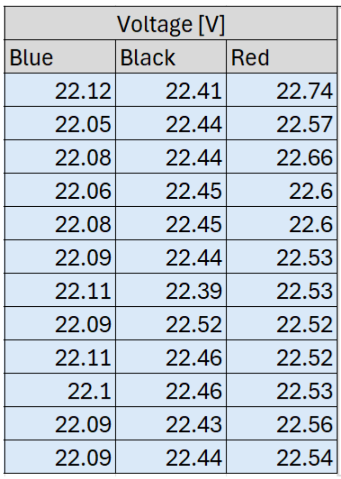


Figure 50: Temperature test results

The sensor readings from the temperature sensor show that the red rig measures the highest temperatures, which makes sense as this was done in room temperature, while the water in the testing pool was slightly cooler. One can also notice that there is a difference between the black and blue rig, which suggest that these sensors require some form of calibration, as the environments were practically identical when testing.



Voltage [V]		
Blue	Black	Red
22.12	22.41	22.74
22.05	22.44	22.57
22.08	22.44	22.66
22.06	22.45	22.6
22.08	22.45	22.6
22.09	22.44	22.53
22.11	22.39	22.53
22.09	22.52	22.52
22.11	22.46	22.52
22.1	22.46	22.53
22.09	22.43	22.56
22.09	22.44	22.54

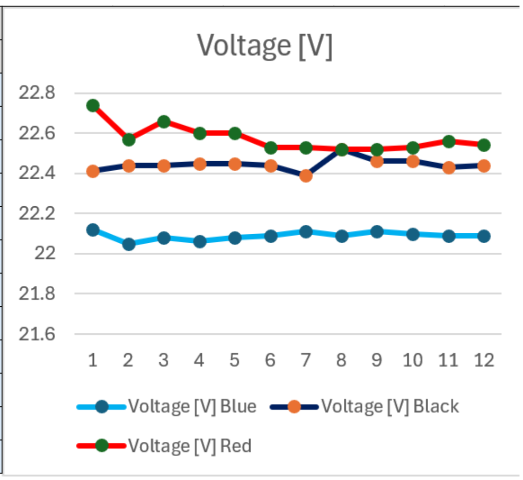
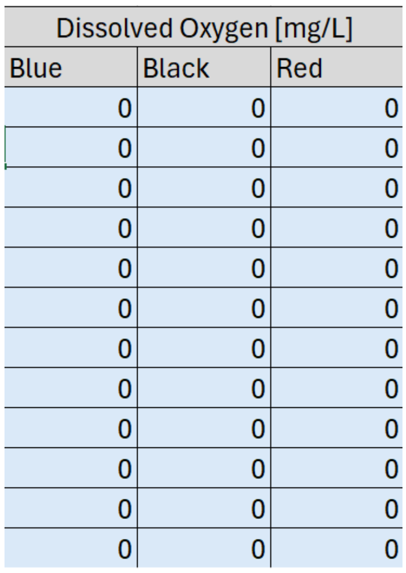


Figure 51: Voltage readings

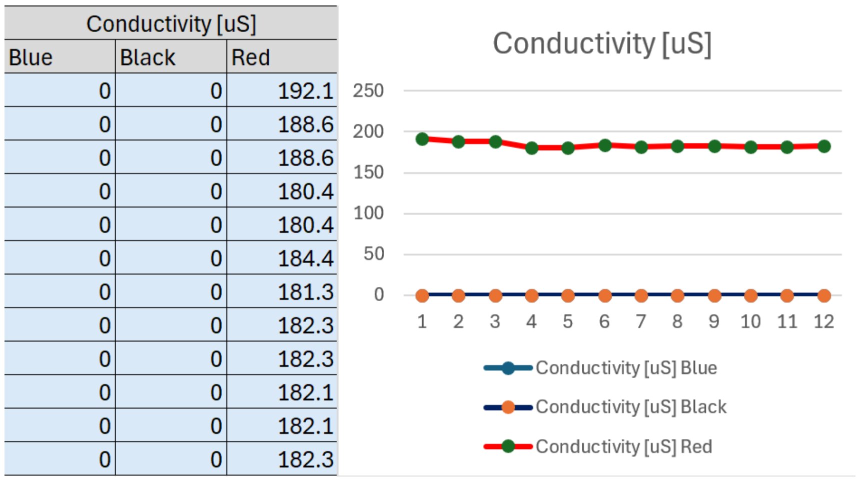
The voltage readings show some slight differences, most notably in the red rig. The black and blue rig demonstrate quite stable voltage readings, while the red rig displays some variation in voltage, particularly in the initial readings. These fluctuations could be random, but they can also be caused by a slight difference in the test conditions. As the red rig uses the VIM3, in comparison to the other two rigs, the red rigs power consumption and regulation could result in less stable voltage readings. The results may also be due to the testing environment. As the red rig was tested in an open cylinder above water, while the other two were tested in a vacuum sealed cylinder underwater.



Dissolved Oxygen [mg/L]		
Blue	Black	Red
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Figure 52: Oxygen test results

The dissolved oxygen results were not as expected. Both the black and red rigs were not expected to provide any results, as the Atlas sensors on these rigs were facing issues, and I2C connection could not be established. The blue rig however, managed to establish I2C connection to all sensors, and should therefore provide some sensor readings. This issue is worth pursuing for further work.



Conductivity [uS]		
Blue	Black	Red
0	0	192.1
0	0	188.6
0	0	188.6
0	0	180.4
0	0	180.4
0	0	184.4
0	0	181.3
0	0	182.3
0	0	182.3
0	0	182.1
0	0	182.1
0	0	182.3

Figure 53: Conductivity test results

Similarly to the dissolved oxygen test, the results from the salinity sensor, measuring conductivity, was not as expected. The black rig had a malfunctioning salinity sensor, so these readings were expected, but once again, the blue rig not having any measurements warrants concern. Both the red and blue rig established I2C connection to the salinity sensor, however only the red rig actually collected data from the sensor.

Capturing underwater photos
In our underwater sensing rig test, we evaluated the capabilities of the system by capturing three distinct images. The first two images featured a Coke bottle, one taken from a close distance and the other from a further distance, to assess the rig's ability to maintain clarity and detail at varying ranges. The third image captured a hand watch, chosen specifically to evaluate the rig's precision in capturing fine details and intricate features underwater. These images provided a comprehensive assessment of both the resolution and the range of the sensing rig, allowing us to determine its effectiveness in capturing clear and detailed underwater photographs.

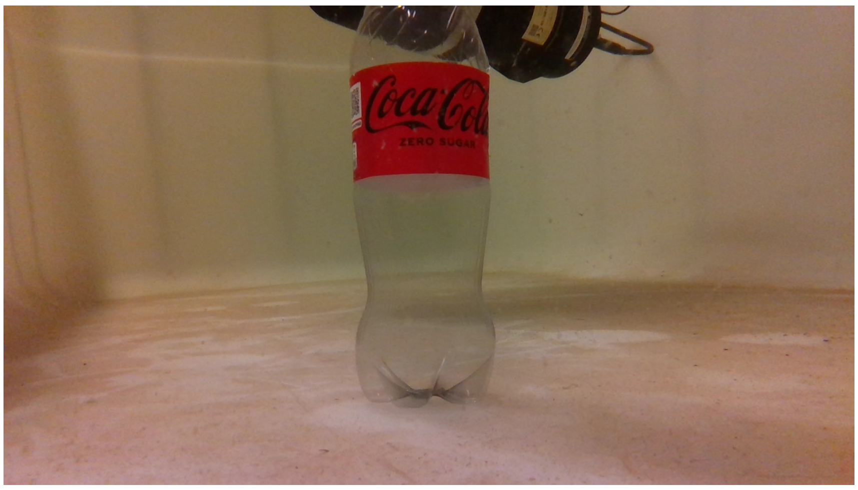


Figure 54: Close up picture

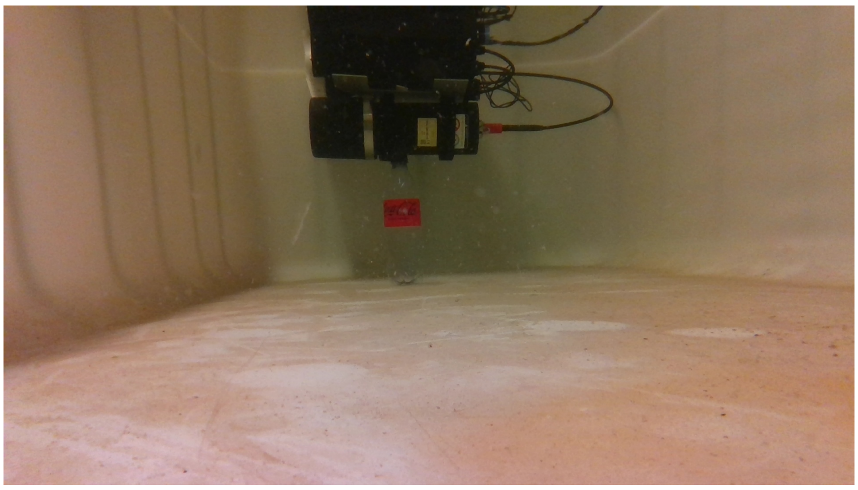


Figure 55: Distance picture

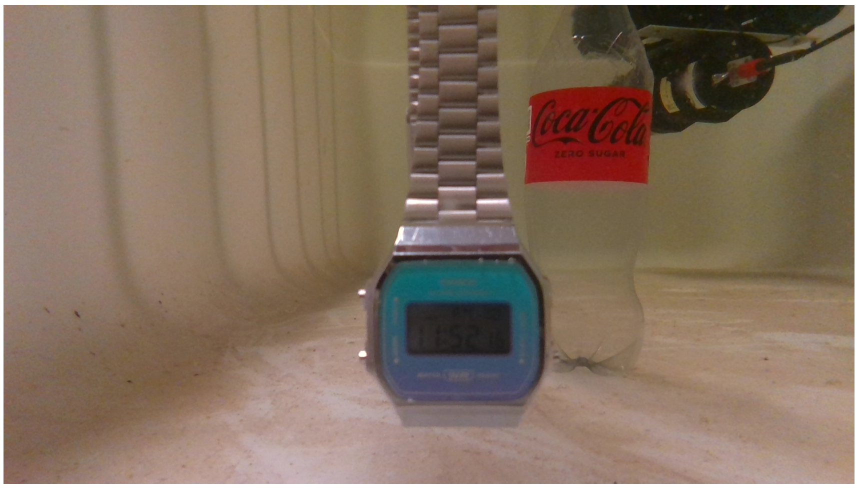


Figure 56: Picture to assess detail

Computer vision to recognize faces

To test the computer vision capabilities of our sensing rig, we used an algorithm initially trained on human faces, with plans to retrain it for detecting fish. The test involved capturing images above water, where the algorithm was evaluated on its ability to detect objects and highlight them with a green box. The algorithm successfully identified and marked these faces in the test images, demonstrating its fundamental object detection capabilities. This above-water test provided a preliminary validation of the detection framework, showcasing its potential and setting the groundwork for future training with fish-specific datasets to enhance accuracy and applicability for underwater detection.

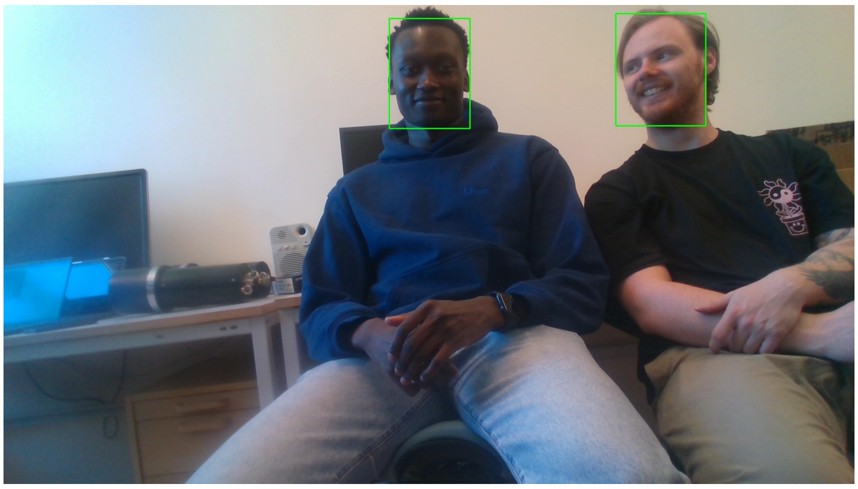


Figure 57: Computer vision algorithm recognizing faces

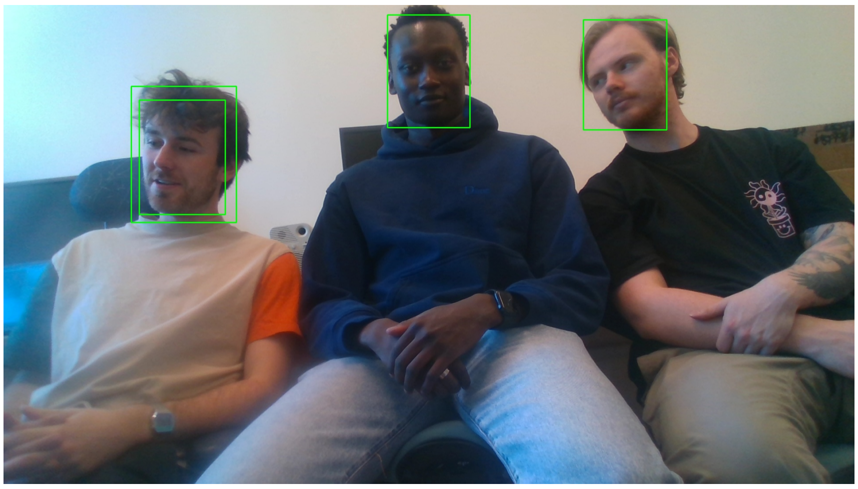


Figure 58: Computer vision algorithm recognizing more faces

6 Discussion

This chapter discusses various aspects of the project, including identified opportunities, challenges faced and suggestions for further work. By analyzing these areas, we aim to provide a greater understanding of the project's progress and potential improvements for future iterations.

6.1 Opportunities identified

Throughout the project, several opportunities to implement ideas that were not initially planned, but proved beneficial where encountered. This section aims to analyze these ideas, focusing on why such implementations were performed and their convenience for the project.

6.1.1 Electronics

In terms of electronics, the introduction to the VIM3 was a significant enhancement for improving the processing capabilities of the sensing rig. The decision to switch from the Raspberry Pi to the VIM3 on the red rig was made after one incident where the Raspberry Pi stopped working and required replacement. After a design review, it was concluded that the VIM3 offered superior performance and accessibility compared to the Raspberry Pi.

Another identified opportunity was the installation of cobalt connectors on all rigs, which facilitated for data extraction from the sensing rigs without the need for opening them. Additionally, the sorting and soldering of Ethernet cables was an identified opportunity which improved the overall manageability of the cables. These adjustments were made to enhance overall reliability and functionality of the electronic system in the sensing rig

6.1.2 Enclosure

The implementation of the camera required waterproofing in some way or another. There was a number of different ways being discussed based on the goal of the project, the specifications and limitations of the rigs. The first decision was figuring out what camera to use. There were generally two possible directions being discussed; one being a camera developed for underwater-use, where the camera is encased in steel as a method for waterproofing. In this case there wouldn't be the need of making a camera house. The other option was using a standalone camera and developing a waterproof camera house which the camera will sit in. The front would be a window made of acrylic or a similar material, such as plexiglasss, allowing the camera to see outside the enclosure.

For the underwater camera, the cons outweighed the pros. As these cameras are generally very expensive and give very little flexibility. The only pro was eliminating the need for creating a camera housing. Going for the standalone camera gives a lot more flexibility as you can choose the desired camera based on specifications and use-cases, as well as being easier to use and make changes to. It's not necessarily a cheaper option, it just depends on the camera you choose.

After choosing the standalone camera, there was still a decision in how to implement it into the sensing rigs, in terms of waterproofing. The first idea that was being explored was having an extra enclosure just for the camera. The early sketches of the enclosure were based on that idea, but this would have created a couple of challenges. First and foremost, there would have been the need to mount this extra enclosure on the sensing rigs. This is an easy task in theory, but there might not have been good way of doing it. Since the sensing rigs already consists of two tubes and a big communication-modem, a problem might occur when being mounted onto the ROVs. The second problem would be the wiring; USB-c cables generally have a high data transmission speed, but they are also quite intricate, meaning cutting the cables could easily lead to the cables losing a lot of its properties. Another idea was having a rounded glass dome as the camera enclosure, which would be attached to the rigs, like the BlueRobotics dome. The problem with this would have been fitting the camera inside and mounting it, as well as the camera would not have 180 degrees vision anyways. Based on this, it was concluded it was better to create a custom housing to fit the size and specifications of the chosen camera.

There was also the need for changing some of the parts used inside the sensing rigs. However, before this could be done, access to the editable CAD-files was needed, which was not available. Therefore there was a case of converting the .stl-files that was available, into .step-files that could be worked on. A good way to convert the files is yet to be found, but there has been created a guide in section 3.3.3 on how to do it in a working way. This method imports the .stl-files as a mesh, which will then be further converted into a solid CAD-model. Like mentioned in the guide, there might be created some holes in the process, which will need to be patched before 3D-printing. After the needed models had been converted, they could be edited and changed accordingly. The Raspberry Pi mount was changed to fit the Khadas Vim3 instead, the track plate was changed to fit the new Khadas Vim mount, the I2C-splitter stand was changed to allow the board to sit the other way around, there was created a mount for the I2C and there was made a CAD-file for the main plate.

6.1.3 Software

A new node has been implemented in the ROS workspace for auto-shutdown. This has been done in order to not discharge the battery and lose data or damage parts of the rig. The node starts an auto-shutdown when the battery goes below 20%. Further testing and calibration is though needed in order to guarantee success.

The plan for computer vision was to implement recognition of marine wildlife, especially fish. Because we had no access to good training data for fish, and not enough time to do training and testing for good results. The computer vision has been implemented with facial recognition. This has made it much easier to implement and to deal with arising problems.

6.2 Challenges

Throughout the project, several challenges were encountered, impacting the progress and functionality of the sensing rigs. This section highlights the main challenges faced and the issues encountered along the way.

6.2.1 Electronics

The project faced several challenges related to electronics. One of the main issues was the defective Atlas sensors on the black and red sensing rigs. The functioning Atlas sensors were the conductivity

sensor on the red rig and both sensors on the blue rig. The faulty senors could not be detected over I2C. New Atlas sensors were ordered, but could not be delivered in time for final tests. Due to this, the group were unable to solve this challenge.
The RTC battery for the VIM3 was ordered, but ultimately was not located after delivery.

While testing the black rig with the faulty Atlas sensors, the Raspberry Pi became faulty itself. With the RPi being stuck in the boot-up phase. This incident posed the best opportunity to conduct a design review to replace the Raspberry Pi with the VIM3, improving the rig's performance.

Another challenge arose when replacing the acoustic modem from SubNero to EvoLogics. This required the group to replace the SubConn cable due to the difference in cable connectivity on the modems. To solve this, the group made testing cables, while also ordering pigtails for future use, as it is more robust.

6.2.2 Enclosure

The first challenge related to the enclosure was making changes to the existing parts. Like mentioned in section 6.1.2, the problem started with the absence of CAD-files, which made it impossible to make changes to the parts. Luckily, there was no need to completely recreate the parts. In order to get an editable file you need to convert the .stl-file into a .step-file. A method for this was developed and there was made a guide for this in section 3.3.3.

Like mentioned in section 6.1.2, the original plan for the camera enclosure was to have a standalone housing mounted to the rigs, with wiring between them for communication between the camera and the computer. Because of the intricacy of the USB-c cables this would have likely resulted in reduced data transmission speed and poor picture quality. There may be workarounds to this, but the camera housing extension of the original rigs ended up being a better solution as it saves space and unnecessary waterproofing and wiring.

After the camera enclosure had been produced by the mechanical workshop, the vacuum testing showed promising results, as the enclosure, while mounted on the sensing rigs, was completely airtight. After setting everything up with the camera and wiring, another test was conducted, and the enclosure was no longer airtight. A lot of time was spent making adjustments and vacuum testing trying to figure out what the problem was. It is hard to say what exactly the problem was, as multiple things were done before the last test, which ended with good results again. The things that were done was cleaning inside the camera housing and the plexiglass, applying more silicone gel, changing screws and applying silicone gel to the vacuum pumps. The changes to the screws was made because the former screws would not tighten and instead spin around loosely. The new screws were longer and went deeper into the holes. It is likely that the screw threads were broken from tightening screws to hard, which happened after opening and closing the enclosure many times. The longer screws would tighten, since they went further down into where the threads were not broken. It also seemed like the vacuum pumps were not sitting right in the openings in the rigs, which is why there was applied silicone gel to the o-rings on the pumps. After this they looked more attached to the openings. There is a chance that all of these reasons contributed to the rigs not being airtight, so it is important to remember this in the future to spare unnecessary time spent trying to figure out what is causing the problem.

Currently the camera housing is equipped with 3mm plexiglass, which according to the simulations done by the mechanical workshop will be safe from implosion as far as 50m underneath the surface. The camera housing is created with support of plexiglass up to a thickness of 10mm. Replacing the glass with thicker glass will result in an increased max depth for the sensing rigs. New simulations will have to be done if these changes are made.

6.2.3 Software

Throughout the project there has been a lot of delays because of waiting for others. Because of this the communication has not been fully tested. The ROS2 code "should" be working, and it runs smoothly, but we had problems with getting a hold of the communications group, so the communication has not been actually tested with the modem they are supposed to work with. We have also tried to get hold of training data for recognition of fish. This data came too late and not prelabeled, resulting in it not being implemented. Therefore the computer vision is still in a prototype state with facial recognition instead of recognition of marine wildlife.

6.3 Further work

6.3.1 Electronics

In section 2.4, there was made some suggestions for further work. Some of these are quite important for the continuation of the project, while others are suggestions for quality-of-life changes.

- **New sensors:**
  There may be need for new sensors, specifically the Atlas Scientific sensors since they seem to be unstable. If it is decided to order new sensors, a design review should be put together with an overview of the different products and their specifications. This should be used as help in choosing the best sensors for the Sensing Rigs.
- **Replacing Raspberry Pi's:**
  It has already been stated that the Khadas VIM3 is a more powerful SBC than the Raspberry Pi 4. In order to fully utilize the AI applications with the camera module, it will be beneficial to replace the two remaining Raspberry Pi's with Khadas VIM3. This will also provide the Sensing Rigs with the same SBC's, which makes it easier to develop the rigs further.
- **Calibration:**
  The sensors will need recalibration, as some of the sensor data either isn't quite right or is unstable. Recalibrating the sensors ensures that the right values are being recorded and evaluates the sensors. If any sensor needs to be changed, the new sensors will also need calibration.
- **Replacing SubConn cables:**
  As mentioned in section 2.4, there is the need of replacing the testing SubConn with SubConn MCIL8F pigtails, as the testing SubConn only acts as a temporary feature. The procedure is described in the section mentioned above.
- **Wiring:**
  Sorting wires inside the Sensing Rigs could be beneficial for further development as it will provide more overview, and decrease risk of short-circuiting and ruining parts. Additionally, using Molex plugs where possible is advantageous as they provide flexibility and secure connections. Lastly, it would be useful to have pinout tables for all connections where it is needed, for overview purposes.

6.3.2 Enclosure

In section 3.4, there was made some suggestions on further work. Even though these are suggestions, it is likely that they will have to be done at some point.

- **File conversion:** The file conversion is pretty straight forward, but can be quite tedious. There are quite a few files that need converting in order to complete the library of parts. This is not an urgent task, so it should not overshadow the main tasks in the project or more important matters, but can be done either while waiting for orders or during some downtime. The guide in section 3.3.3 could end up making a broken model, with missing surfaces. **Step 4** in the guide focuses on patching up the model if this happens, but it is quite a flawed process. If a better way to patch the model is found, the guide should be updated accordingly.

- **Replacing the plexiglasss:** This is another non-urgent task, but this is quite a quick fix and doesn't acquire much work. The mechanical workshop did not have any 10mm plexiglasss, so this will have to be ordered. When the glass arrives, talk to the mechanical workshop and they will aid in cutting out the plexiglasss to fit the camera housing. **DO NOT** conduct any tests deeper than 50m until the 3mm plexiglasss is replaced as the pressure may break it and fill the rigs with water. After replacing the plexiglasss, the mechanical workshop can run a simulation which will set a new max recommended depth.
- **Implementation of light for the camera:** Sadly, there was not conducted any deep water tests during this project, so the implementation of an external light is still a question-mark. If deep water tests are conducted by the next group, it is important to assess the pictures taken by the sensing rigs, and decide whether or not an external light is needed.
- **Buoyancy:** Currently, the communication modem and the tube including the battery are the two heaviest objects on the sensing rigs, which means the rigs will turn in the water and end up upside down. It is currently not known how the communication modem will be affected by this, but the camera will end up in an 135°off-angle, which is not ideal. To fix this the center of gravity of the sensing rigs needs to be shifted from the communication modem to between the tubes of the sensing rigs and maybe even further down. Since the sensing rigs can be mounted on the BlueROV, the solution may need to be detachable so it does not interfere with the mounting. For further details, it would be beneficial to contact the team working on the BlueROV.

6.3.3 Software

For future development the computer vision needs further development. Recommendations for each of these topic will be listed down below:

- **Optimizing battery life**
  These stations are typically deployed in remote and challenging environments, where accessibility for maintenance and battery replacement is limited. Ensuring efficient battery usage directly impacts the longevity, reliability, and overall effectiveness of the monitoring systems.
- **New algorithm:**
  Work needs to be done in figuring out what classes of wildlife needs to be recognized. For example: 1. Fish, 2. Crabs, 3. Lobsters etc. Or differentiating between different types of fish. Then a data collection needs to be done in order to make a dataset to train on. Here, a biologist or similar should be contacted to help with figuring out distinguishing marks of different types of wildlife. The two options for collecting data set is either collecting it by yourself, or getting it from someone who already has it.
  - **Collecting by yourself** will give the best results, as you can use the same camera, collect videos from the correct depth, and the images will therefore be most similar to real world conditions. This will take a lot of time as you need to collect enough data as well as labelling all the data. There is also no guarantee you will get a big enough dataset needed to get a good algorithm.
  - **Getting dataset from other** will save alot of time and most likely give a much bigger data sample. The downside is that the dataset might not resemble real world application for your use. If this approach is used, we recommend contacting SINTEF and asking for prelabeled data for training. They have a big dataset of salmon from a salmon pen.
- **Creating a trigger for video capture:**
  There needs further development in creating a trigger mechanism for the computer vision algorithm. The algorithm is very process demanding, and the camera don't need to run all the time. It can save a lot of battery only having the camera and the computer vision algorithm running when needed. In order to do this we recommend creating another ROS_2 node with a motion detector. The camera can be used as the motion detector. This node should publish to a topic, and the computer vision module can subscribe to this topic and start running when needed.

- **Post processing of images:**
  There needs further work in figuring out what to do with the camera vision module. At the moment it saves an image every 10 seconds with a bounding box around each object in the image. Saving this many images takes alot of storage space, and is only done to show a proof of concept. The low bandwith of the modem also creates challenges. There needs to be performed more tests with the modem to figure out best way to send information, and if the images needs to be resized etc. Here is alot of options for further work. The computer vision algorithm can for example be used only for counting different types of wildlife at all times, and create a log. It can be used to send images in a set interval etc.

7 Conclusion

The development of the standalone underwater monitoring station has progressed significantly, approaching closely with the overarching goal of providing valuable insight into aquatic environments. This iteration of the project has seen notable enhancements across all sections, bringing the rigs significantly closer to fulfilling their ultimate purpose.

Starting with the electronics, the integration of the Intel RealSense camera stands out as a significant enhancement, endowing all three rigs with vision capabilities and enhancing their observational capabilities. Furthermore, the implementation of the VIM3 into the red rig has provided greater processing power and has enabled the utilization of AI applications for future use through its NPU chip. Additional enhancement include adding a RTC to the Raspberry Pi's for correct timekeeping. These enhancements directly increases the rigs capabilities while simultaneously introduce a whole new field of future development in AI applications.

In terms of enclosure design, the introduction of the water-resistant camera housing is the most noteworthy enhancement. It effectively protects the camera from water intrusion to a maximum depth of 50 meters without compromising field of view and minimizing distortion caused by the glass. Moreover, the conversion of key components from .stl files to CAD models has contributed to enhanced versatility and customization options for future development on the rigs.

Regarding software development, the implementation of computer vision algorithm has laid a solid foundation for future work. The utilization of the YOLO algorithm, initially trained on facial recognition, presents the opportunity for effective classification of underwater wildlife. Additionally, the integration of the VIM3 ensured minimal differences between the three rigs regarding software, setting the stage for future development with computer vision algorithms and increased processing power. Further enhancements, such as the configuration of RTC on the Raspberry Pi's and the implementation of an auto shutdown module, contribute to the overall robustness of the sensing rigs software.

Through extensive testing, the group aimed to validate and document the performance of the sensing rigs. Initially, the plan to perform a final field test in real-world conditions did not materialize due to cooperation issues with the communication group resulting in an unfinished communication module. However, the group successfully managed to conduct the final tests, validating the progress made while documenting challenges and deficiencies. These deficiencies vary across the three rigs. The most significant challenges relates to the defective Atlas sensors on the red and black rigs, difficulties in submerging the rigs due to their floating nature in water, and an unfinished communication module.

In conclusion, this iteration of the project has significantly contributed to the overarching goal by enhancing all sections on the sensing rigs. By structuring the thesis into several sections focusing on areas where development has taken place, it provides greater understanding and continuity while serving as a solid foundation for future work. Furthermore, the abundance of step-by-step guides for implementation amplifies this, while also ensuring scalability and reproducibility. While challenges remain, the group has made significant progress during this iteration, setting the stage for huge advancements in underwater monitoring technology.

Bibliography

Analogue to Digital Converter (2024). Eleectronics Tutorials. URL: https://www.electronics-tutorials.ws/combination/analogue-to-digital-converter.html (visited on 16th Apr. 2024).

Barrier Strip Terminal Blocks: A Guide to Electrical Connections (2024). North Park Group. URL: https://northparkgroup.com/barrier-strip-terminal-blocks/ (visited on 17th Apr. 2024).

BotBlox (2023). *GigaBlox*. URL: https://botblox.io/content/GigaBlox%20DataSheet%20C_E%20%28Oct%202023%29.pdf (visited on 17th Apr. 2024).

Chai, Wesley (2024). *CAD (Computer-Aided Design)*. URL: https://www.techtarget.com/whatis/definition/CAD-computer-aided-design (visited on 19th Mar. 2024).

Conductivity Kit (2024). Atlas Scientific Environmental Robotics. URL: https://atlas-scientific.com/kits/conductivity-k-1-0-kit/ (visited on 5th Apr. 2024).

D435 Depth Camera (2024). Intel RealSense. URL: https://www.intelrealsense.com/depth-camera-d435/ (visited on 20th May 2024).

Danielson, Shawnee (2024). *What Is The Ethernet?* URL: https://robots.net/tech/what-is-the-ethernet/ (visited on 17th Apr. 2024).

Dissolved Oxygen Kit (2024). Atlas Scientific Environmental Robotics. URL: https://atlas-scientific.com/kits/dissolved-oxygen-kit/ (visited on 5th Apr. 2024).

DS3231 (2015). Maxim Integrated. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/DS3231.pdf (visited on 8th Apr. 2024).

Example of boundingboxes and label. (2024). Google. URL: https://storage.googleapis.com/openimages/web/visualizer/index.html?type=detection&set=train&c=%2Fm%2F0ch_cf&id=6b548fb27158b129 (visited on 2024).

I2C Bus Splitter (2024). JM Robotics. URL: https://www.jmrobotics.no/p/i2c-bus-splitter (visited on 8th Apr. 2024).

Industries, Adafruit (2024). *Adafruit 4-Channel ADC Breakouts*. URL: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-4-channel-adc-breakouts.pdf (visited on 16th Apr. 2024).

Kaggle, Your Machine Learning and Data Science Community (2024). Kaggle. URL: https://www.kaggle.com/ (visited on 2024).

Khadas VIM3 (2024). Armbian. URL: https://www.armbian.com/khadas-vim3/ (visited on 15th Apr. 2024).

Knudsen, Sivert Berg and Peder Brandstorp (2023). *A Standalone Underwater Monitoring Station for Marine Wildlife Monitoring*. Bachelor's Thesis.

Last years Github Repository (2023). URL: https://github.com/Pederbs/SUMS (visited on 2023).

Lipo Smart Battery Pack (2024). Tattu Plus. URL: https://genstattu.com/tattu-plus-15c-12000mah-6s1p-as150-xt150-plug-lipo-battery.html (visited on 19th Apr. 2024).

Mauser Electronics (2024). Molex/Mouser. URL: https://no.mouser.com/ProductDetail/538-38770-0106 (visited on 17th Apr. 2024).

Meaney, David (2024). *What is a Real Time Clock*. ECS Inc International. URL: https://ecsxtal.com/what-is-a-real-time-clock-rtc/ (visited on 8th Apr. 2024).

NOAA, National Oceanic 'and' Atmospheric Administration (2024). *Sea Water*. URL: https://www.noaa.gov/jetstream/ocean/sea-water (visited on 19th Mar. 2024).

NOS, National Ocean Service (2024). *Monitoring Oceans and Coasts*. URL: https://oceanservice.noaa.gov/observations/monitoring/ (visited on 11th Mar. 2024).

Open Images Dataset V7 (2024). Google. URL: https://storage.googleapis.com/openimages/web/index.html (visited on 2024).

Open_Robotics (2024). *How to create launch file for ROS2 nodes*. URL: https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html (visited on 24th May 2024).

OpenCV (2024). Open Source Vision Foundation. URL: https://opencv.org/ (visited on 30th Apr. 2024).

Power, Traco (2022). *THN 15WIR Datasheet*. URL: https://docs.rs-online.com/56c5/0900766b816e9099.pdf (visited on 8th Apr. 2024).

Power sense module (2024). Blue Robotics. URL: https://bluerobotics.com/store/comm-control-power/control/psm-asm-r2-rp/ (visited on 10th Apr. 2024).

Pressure Sensor (2024). Blue Robotics. URL: https://bluerobotics.com/store/sensors-cameras/sensors/bar30-sensor-r1/ (visited on 5th Apr. 2024).

Raspberry Pi 4 (2024). Raspberry Pi Foundation. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (visited on 15th Apr. 2024).

Raspberry Pi 4 Tech Specs (2024). Raspberry Pi Foundation. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (visited on 15th Apr. 2024).

ROS documentation (2024). Open Robotics. URL: https://www.ros.org/ (visited on 2024).

Sealution (2024). *What is an O ring: all you need to know*. URL: https://sealution.eu/knowledgebase/o-ring/ (visited on 19th Mar. 2024).

Sustainable Development Goals (2024). United Nations. URL: https://www.un.org/sustainabledevelopment/sustainable-development-goals/ (visited on 6th May 2024).

TATTU PLUS INTELLIGENT FLIGHT BATTERY (2024). Tattu Plus. URL: https://www.genstattu.com/content/Tattu-Plus.pdf (visited on 19th Apr. 2024).

Temperature Sensor (2024). Blue Robotics. URL: https://bluerobotics.com/store/sensors-sonars-cameras/sensors/celsius-sensor-r1/ (visited on 5th Apr. 2024).

This years Github Repository (Raspberry Pi) (2024). URL: https://github.com/herkis/bachelor-2024 (visited on 2024).

This years Github Repository (VIM3) (2024). URL: https://github.com/ErjokAguto/SUMS/tree/main (visited on 2024).

Ubuntu (Debian packages) (2024). Open Robotics. URL: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html (visited on 26th Apr. 2024).

Ultralytics Installation (2024). Ultralytics Inc. URL: https://docs.ultralytics.com/quickstart/ (visited on 2024).

Ultralytics Training (2024). Ultralytics Inc. URL: https://docs.ultralytics.com/modes/train/#tensorboard (visited on 2024).

USB 3.1 Type-C Cables for Intel® RealSense™ Camera (2024). Newnex Technology Corp. URL: https://newnex.com/realsense-3d-camera-connectivity.php (visited on 9th May 2024).

USGS, U.S. Geological Survey's Water Science School (2024). *Conductivity (Electrical Conductance) and Water*. URL: https://www.usgs.gov/special-topics/water-science-school/science/conductivity-electrical-conductance-and-water (visited on 19th Mar. 2024).

VIM3 (2024). Khadas Technology Co. URL: https://www.khadas.com/vim3 (visited on 15th Apr. 2024).

WetLink Penetrator Installation Guide (2024). BlueRobotics. URL: https://bluerobotics.com/learn/wetlink-penetrator-installation-guide/ (visited on 1st May 2024).

What is Autodesk Fusion? (2024). Autodesk. URL: https://www.autodesk.com/solutions/what-is-fusion-360 (visited on 20th May 2024).

WiFi (2024). Khadas Technology Co. URL: https://khadas.github.io/linux/vim3/Wifi.html (visited on 26th Apr. 2024).