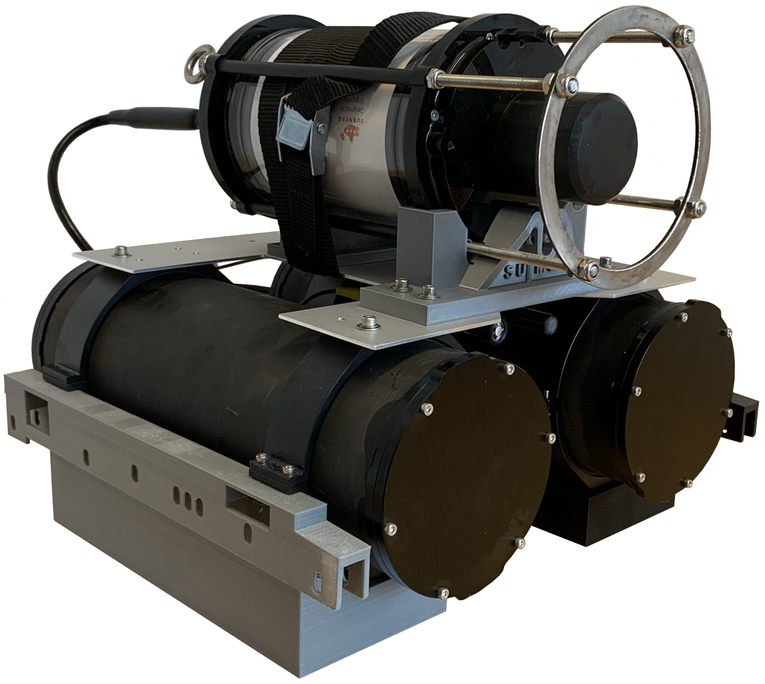
Sivert Berg Knudsen
Peder Brandstorp Sanden

A Standalone Underwater Monitoring Station (SUMS) for marine wildlife monitoring

Bachelor's thesis in Electrical engineering
Supervisor: Behdad Aminian
May 2023

Bachelor's thesis



Sivert Berg Knudsen
Peder Brandstorp Sanden

A Standalone Underwater Monitoring Station (SUMS) for marine wildlife monitoring

Bachelor's thesis in Electrical engineering
Supervisor: Behdad Aminian
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Thesis Title: A Standalone Underwater Monitoring Station (SUMS) for marine wildlife monitoring			
Authors: Sivert Berg Knudsen Peder Brandstorp Sanden		**Project Number:**	E2309
		Due Date:	30.05.2023
		Grade:	[x] Open [ ] Closed
Study Program:	Bachelor in Electrical Engeneering		
Field of Study:	Autonomation and Robotics		
Internal Supervisor: **Department:**	Behdad Aminian Department of Engeneering Cybernetics		
Client: **Contact Person:**	Department of Engeneering Cybernetics Damiano Varagnolo, damiano.varagnolo@ntnu.no, +47 481 28 922		

Abstract:

Knowledge of the ocean remains limited due to challenges such as inaccessibility and technological limitations. In this context, the standalone underwater monitoring station (SUMS) emerges as a crucial tool for oceanographic research. Designed to measure and communicate vital parameters in the ocean, the SUMS rig overcomes logistical challenges and provides unprecedented insights into the marine environment. Its versatility allows integration with Remotely Operated Vehicles (ROVs) for exploration in inaccessible areas. Advanced acoustic modem technology enables the formation of an interconnected network of observation posts, facilitating data collection and analysis to enhance our understanding of the ocean's dynamic nature.

Sammendrag:

Kunnskapen om havet forblir begrenset på grunn av utfordringer som utilgjengelighet og teknologiske begrensninger. I denne sammenhengen blir den frittstående undervannsovervåkingsstasjonen (SUMS) et viktig verktøy for havforskning. Designet for å måle og kommunisere vitale parametere i havet, overvinner SUMS-riggen logistikkmessige utfordringer og gir enestående innsikt i det marine miljøet. Dens fleksibilitet tillater integrasjon med fjernstyrte undervannsfartøy (ROVer) for utforskning av utilgjengelige områder. Avansert akustisk modemteknologi muliggjør dannelse av et sammenkoblet nettverk av observasjonsposter, som letter datainnsamling og analyse for å forbedre vår forståelse av havets dynamiske natur.

Keywords:	**Stikkord:**
Underwater technology, Acoustic comunication, ROS2, Data collection, 3D modeling and printing	Undervannsteknologi, Akustisk kommunikasjon, ROS2, Data innhenting, 3D modelering og printing

Acknowledgements

We extend our heartfelt gratitude to our supervisor, Behdad Aminian for his invaluable guidance, support, and expertise. His insightful feedback and continuous encouragement have been crucial in shaping our thesis and enriching our learning experience.

Furthermore, we would like to express our sincere appreciation to Damiano Varagnolo for providing us with the exhilarating opportunity to work on this unique and exceptional project. His collaboration and valuable inputs have contributed significantly to the success of our research.

We thank the electrical and mechanical workshops at the Department of Engineering Cybernetics. Also, Pedro De La Torre for guidance and great recommendations. Their contribution has greatly facilitated all practical aspects of our project.

We would also like to thank Emil Wrengle and Simon Andreas Hagen Hoff, two doctoral fellows for their valuable insights and technical assistance.

Lastly, we would like to express our deepest appreciation to our family and friends for their unwavering support and encouragement throughout our academic journey. Their love and patience have been invaluable in enabling us to pursue our goals and successfully complete Bachelors degree in electrical engineering.

Norwegian University of Science and Technology
Trondheim, May 2023

[illegible] [illegible]

Sivert Berg Knudsen Peder Brandstorp Sanden

Abstract

Our understanding of the Earth's atmosphere has made significant strides, while our knowledge of the ocean remains limited due to challenges such as inaccessibility and technological limitations. In this context, the standalone underwater monitoring station (SUMS) emerges as a crucial tool for oceanographic research. Designed to measure and communicate vital parameters in the ocean, the SUMS rig overcomes logistical challenges and provides unprecedented insights into the marine environment. Its versatility allows integration with Remotely Operated Vehicles (ROVs) for exploration in inaccessible areas. Advanced acoustic modem technology enables the formation of an interconnected network of observation posts, facilitating data collection and analysis to enhance our understanding of the ocean's dynamic nature.

The development of the SUMS rig has reached significant milestones, demonstrating successful sensor readings and data processing capabilities. However, further examination is required to ascertain the functionality and accuracy of the sensors, particularly in the case of inconsistencies observed with Atlas Scientific sensors. Hardware improvements and software optimization have contributed to a reliable and efficient final product.

The project's mass production aspects have yielded notable outcomes, with two fully operational rigs ready for deployment. Detailed documentation ensures reproducibility and scalability of the design. Overall, the progress made showcases the team's dedication to developing a reliable and efficient underwater monitoring system, laying a strong foundation for future refinement and expansion in the field of underwater technology.

Sammendrag

Vår forståelse av jordens atmosfære har utviklet seg betydelig, mens vår kunnskap om havet forblir begrenset på grunn av utfordringer knyttet til utilgjengelighet og teknologiske begrensninger. I denne sammenhengen framstår den frittstående undervannsovervåkingsstasjonen (SUMS) som et viktig verktøy for havforskning. Utviklet for å måle og kommunisere vitale parametere i havet, overvinner SUMS-riggen logistiske utfordringer og gir enestående innsikt i det marine miljøet. Dens allsidighet tillater integrering med fjernstyrte undervannsfarkoster (ROV-er) for utforskning av utilgjengelige områder. Avansert akustisk modemteknologi gjør det mulig å danne et nettverk av sammenkoblede observasjonsposter, som letter datainnsamling og analyse for å forbedre vår forståelse av havets dynamiske natur.

Utviklingen av SUMS-riggen har nådd betydelige milepæler, med vellykkede sensoravlesninger og datahåndteringsevner. Imidlertid kreves ytterligere undersøkelser for å fastslå funksjonaliteten og nøyaktigheten til sensorene, spesielt med hensyn til inkonsekvenser observert med Atlas Scientific-sensorene. Maskinvareforbedringer og optimalisering av programvaren har bidratt til et pålitelig og effektivt sluttprodukt.

Prosjektets aspekter knyttet til masseproduksjon har gitt betydelige resultater, med to fullt operative rigger klare for utplassering på havets dyp. Detaljert dokumentasjon sikrer reproduksjonsevne og skalerbarhet av designet. Samlet sett viser fremgangen som er oppnådd, teamets dedikasjon til å utvikle et pålitelig og effektivt undervannsovervåkingssystem, og legger et solid grunnlag for fremtidige forbedringer og utvidelser innen feltet undervannsteknologi.

Table of Contents

List of Figures

List of Tables

List of Listings

Acronyms

ACK Acknowledged.

ADC Analog Digital Converter.

AUR-Lab Applied Underwater Robotics Lab.

CAD Computer-aided Design.

DO Dissolved Oxygen.

GPIO General Purpose Input/Output.

IC Integrated Circuit.

IES Internal Electronic System.

IP Internet Protocol.

I^2C Inter-Integrated Circuit.

LAN Local Area Network.

MAC Media Access Control.

NACK Not Acknowledged.

NIC Network Interface Controller.

NTNU Norwegian University of Science and Technology.

OS Operating System.

PSM Power Sense Module.

PSU Power Supply Unit.

ROS Robot Operating System.

ROV Remotely operated vehicle.

RPi Raspberry pi.

RTC Real Time Clock.

SMBus System Management Bus.

SSH Secure Shell.

SUMS Standalone Underwater Monitoring Station.

UART Universal Asynchronous Receiver-Transmitter.

WiFi Wireless Fidelity.

Chapter 1

Introduction

Our understanding of the Earth's atmosphere has flourished, thanks to factors such as accessibility, research focus, and direct observation. In contrast, our understanding of the ocean remains constrained, facing challenges arising from inaccessibility, technological limitations, and the intricate nature of its depth and complexity. Covering approximately 71% of the Earth's surface, the ocean poses formidable logistical challenges for exploration and data gathering. Conducting oceanographic research demands specialized equipment, dedicated vessels, and costly expeditions. The vast depths of the ocean, coupled with its dynamic and diverse conditions such as temperature, salinity, currents, marine life, and chemical composition, further add complexity to comprehensive study endeavors.

Welcome to the realm of underwater technology, a domain where the Standalone Underwater Monitoring Station (SUMS) stands as a testament to our ongoing pursuit of knowledge and innovation. This rig is designed to measure and communicate vital parameters such as temperature, salinity, pressure, and dissolved oxygen in the ocean. As a standalone monitoring station, it can collect valuable data over extended periods, providing unprecedented insights into the marine environment without the need for an indispensable ecosystem of technological accessories. In addition to its stationary functionality, the SUMS rig can be seamlessly integrated with Remotely Operated Vehicles (ROVs), allowing for mobility and exploration of otherwise inaccessible areas. This versatility enhances the ability to uncover new discoveries and expand our understanding of the ocean's mysteries.

The SUMS rigs employ advanced acoustic modem technology, enabling them to communicate and form a network of interconnected observation posts. This network facilitates data collection and analysis, enhancing our knowledge of the ocean's dynamic nature. These technological advancements pave the way for groundbreaking research, sustainable ocean management, and the preservation of marine ecosystems. The SUMS rig empowers us to delve into uncharted waters, shedding light on the secrets hidden beneath the ocean's surface.

1.1 Thesis Assignment

This project is a combination of very diverse and fascinating challenges. Therefore, it provides an exceptional opportunity for participants to not only enhance their knowledge but also expand the scope of their own experiences, preparing for their next career stage in industry or academia. Since this project is the finalizing phase of previous ones, it provides a unique opportunity to explore different areas and also the chance to launch the final product in the field. The mentioned project is a combination of the following tasks:

1. Finalizing the sensing rig: During previous projects, a sensing rig for sensing the environmental data, such as dissolved oxygen, temperature, etc. has been designed and developed. Also, the designed rig can publish the collected data on ROS topics. In this project, the mentioned sensing rig should be verified and enhanced (by adding power consumption data, hydrophones and integrating stereo camera videos). Finally, as part of this project, students should finalize such a sensing rig and prepare it for field usage.

2. Preparing the communication Rig: The collected data from the sensing rig should then be communicable to other stations through a long-range acoustic modem. In this part, the published data from the sensing rig should be read from ROS topics and transmitted. This part provides a unique opportunity for experiencing acoustic communication.

3. Wrapping up and packing the final product: During the previous development, using 3D printing, many parts have been designed and printed. In this project, the required modifications should be applied, and the remaining parts should be designed so that both sensing and combination rigs should be nicely packed together for real experiments.

4. Mass production: We hope to be able to run a final field demo experiment where we monitor for some hours the movement of fish outside of Munkholmen. We hope that at least three rigs will be operational, even if one will already

suffice to draw the attention of the various stakeholders interested in wildlife monitoring at the Scandinavian level.

1.1.1 Narrowing of Project

The narrowing of the project that was done in the preliminary report is as follows:

First and foremost, a solution needs to be found to transmit and potentially receive sensor data using Subnero's acoustic modem. This can be achieved by establishing a dedicated node in ROS that subscribes to relevant topics and forwards the data from these topics through the JANUS protocol via the modem.

The modem needs to be mounted on the existing rig. In this regard, the project group needs to design an enclosure to securely attach the modem in a simple yet robust manner. It is proposed to 3D-print a bracket along with a strap to hold the modem in place without subjecting it to unnecessary strain on its housing. The oxygen and salinity sensors also need to be secured to the rig, as currently they are suspended freely along their data cables.

The aforementioned tasks constitute the core of the project and should be given the highest priority during project planning and execution. All other aspects are part of the problem statement but neither have high priority nor are necessary for the product to function as described by the client at the end of the bachelor thesis. The other issues to be addressed are as follows.

1.1.2 Challenges

The group encountered several challenges. The biggest challenge was the loss of the third member. The third member decided for personal reasons not to complete the thesis. The decision came early in the project and so the remaining group had time to reorganize the scope of the project. The group also did not have access to the modem used for acoustic communication in the period from late April to the middle of May. This hindered the development of the communication structure as testing without the modem was difficult.

Furthermore, the group was tasked with 3D printing brackets for the rig, the challenge faced was to print the brackets used to connect the rig to a ROV. The only printer big enough for these parts was placed behind locked doors which the group members only had access in work hours five days a week. In addition to this, there were several problems with the printers.

1.2 Achievements

The group managed to design and manufacture robust brackets for mounting the modem to the rig. A working program for sensing underwater environments, and communicating the sensor values wirelessly underwater was completed.

Achievements regarding learning have been learning: ROS2, 3D modeling, and 3D printing.

1.3 How the Report is Structured

The report consists of seven chapters, with each chapter covering separate themes such as theory, methodology, results, discussion, and conclusion. The method chapter is divided into two different chapters containing two different themes. The chapters and their theme are as follows:

Chapter 1 is the introduction and so it introduces the thesis.

Chapter 2 is about the background of the project and contains some information about how knowledge was gathered.

Chapter 3 focuses on the theory of the project. In this chapter theories and components associated with the thesis and the sensing rig will be covered.

Chapter 4 describes the method used to configure and develop the environment and software of the sensing rig. This section delves into the methods used to build a ROS2 program that is executed on a headless Raspberry pi.

Chapter 5 contains all the physical elements of the sensing rig. It covers the methods used to create solutions for mounting parts, creating cables for underwater usage, waterproofing, calibrating, and usage of the sensors equipped on the sensing rig.

Chapter 6 showcases the results of all hardware created in this project, furthermore, it presents the results of tests done to the sensors and some flowcharts for easy code comprehension.

Chapter 7 is the discussion where results and other aspects regarding the finished product and obstacles met in the project are discussed.

Second to last is the conclusion of the thesis.

The last chapter is the Appendix.

The report will mention three different color rigs: red, blue, and black. These refer to the three rigs. The color indicates the Internal Electronic System (IES) belonging to the rig.

Chapter 2

Background

2.1 Previous Work

The project is a further development of a bachelor project completed in 2022 [25]. The previous bachelor project was also a further development of a project, making this the third development of the sensing rig. The sensing rig is intended as both a stand alone sensing rig and to be mounted on top of a BlueRobotics BlueROV2 [5].

The group had access to the work of the previous group, this included a code repository containing basic code for using the sensors [29], multiple 3D files, and their report which included many manuals and descriptions for ease of use.

2.2 Gathering knowledge / Prerequisites

The project required new knowledge about ROS2, Linux, 3D modeling and waterproofing electrical components. The group had an overview of ROS2 through previous studies but no experience using it.

2.2.1 Learning Linux

The Raspberry pi (RPi) is a small computer and thus it needs an Operating System (OS) to run. For this project the OS chosen to run the Raspberry pi was Ubuntu server [32]. Ubuntu server was chosen due to it being one of the most popular Linux distributions. This makes it easier to consult different forums on the web to gather knowledge.

2.2.2 Learning ROS2

From the previous group, the code was written in ROS2, naturally, the group then decided to continue on their structure. To gather knowledge about ROS and how to use the framework it provides to make a functional program the internet was used extensively.

Several tutorials provided by the ROS2 Humble home site were used to get the essence of the core concepts of ROS. After this, a YouTube tutorial covering everything from installing ROS to creating a custom program with nodes developed in Python was used as a springboard into development.

During development, several forums were used extensively to find functionality not covered by the beginner tutorials.

This beginner tutorial is perfect for starting to code python nodes in ROS2 Humble, it goes through the steps from installing ROS2 Humble to controlling an autonomous turtle built upon the standard turtle simulator [41].

2.2.3 Learning Fusion360

Initially the plan was to use SketchUp to design all parts needed for this project. SketchUp is another CAD software mainly used by architects. This was used by the previous group to design all the 3D-modeled parts. This group was also familiar with SketchUp. Hence, it was natural to continue using this software. It was later discovered that SketchUp now is a paid software, therefore the group decided to use Fusion 360 by Autodesk which is free for students at NTNU. Fusion 360 is a great application for this type of task, however, none of the group members had any experience with Fusion 360.

Here are some pointers on how to start learning Fusion 360 quickly. Learning Fusion 360 to design simple rigid parts is easy. A good start is to find a good tutorial online and follow it to the letter. This way one can learn good habits right away. Learning what the functionality is capable of and what it shouldn't be used for. When the basic concepts are understood, try designing the same part, this time by heart.

This beginner tutorial consisting of three episodes introduces the viewer to start a new project in Fusion 360 and how to make your first component [10].

Chapter 3

Theory

3.1 Raspberry Pi 4 Model B

Raspberry pi (RPi) is a small and powerful single-board computer, it is designed for a range of applications including robotics and small projects. The Raspberry pi 4 model B has a quad-core ARM-based processor which runs at 1.8GHz. The model used in this project comes with 8GM LPDDR4-3200 SDRAM. Furthermore, the RPi is equipped with Ethernet access, Bluetooth 5.0, and both 2.4 GHz and 5.0 GHz WiFi. The RPi also has a 40-pin GPIO header which is integral for this project.

It is the GPIO functionality of the RPi that makes it viable for this project. The RPi communicates with the sensors with I^2C, it then processes the data before it stores it and sends it off to the modem for further transmission.

The RPi has no integrated storage so the whole OS is stored on a Micro SD card which has to be flashed.

3.1.1 System Management Bus

System Management Bus (SMBus) is a communication protocol that is used for interconnecting low-speed devices, such as sensors and peripheral devices, to single-board computers like the RPi. SMBus is a subset of the I^2C interface that is explained in section 3.9.1 and is needed to to utilize I^2C communication with python.

3.2 Inter-intergrated Circuit

Inter-Integrated Circuit (I^2C) is a simple and versatile communication protocol between one or multiple masters and one or multiple slaves. It uses two cables to transfer information: SDA (Serial Data) is the line where data is transmitted and SCL (Serial Clock) is the line which carries the clock signal. The protocol is bidirectional meaning that the master can send data to the slave and vice versa. It is both synchronous and serial, this means that data is transferred bit by bit along the SDA line while the master control the shared clock signal in the SCL line. The clock is used to synchronize the sampling of bits. All messages are built to the same structure, they always start with a start condition to signal that a new message is starting. After the start condition there is a frame containing the binary address the data is sent to followed by a read/write bit. After this there is an Acknowledged (ACK)/Not Acknowledged (NACK) bit, after this the actual data is sent in 8-bit intervals separated by ACK/NACK bits, and the message ends with a stop condition [24].

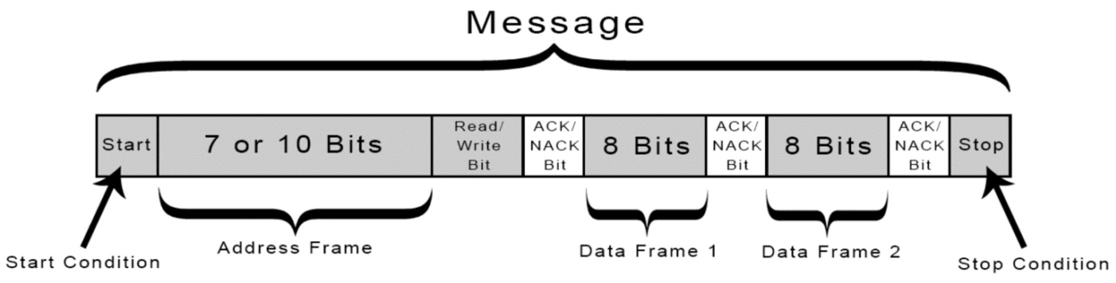


Figure 3.1: I^2C message structure.

3.3 Ubuntu server

Ubuntu server is a Linux distribution composed mostly of free and open-source software. It utilizes the Debian package management system, which enables easy installation, removal, and updating of software packages. It follows a modular design approach, allowing administrators to select and install only the necessary components for their specific requirements, resulting in a lightweight and optimized system which is important when running on a small computer such as the Raspberry pi. There is no bloatware or other unnecessary software preinstalled.

Ubuntu Server does not come with a graphical overlay or desktop environment installed by default. It is designed to be a command-line-based operating system optimized for server deployments. The absence of a graphical interface helps to reduce resource usage and improve performance which is crucial when using small computers such as Raspberry pi. Additionally, Python3 comes default in Ubuntu Server, providing a powerful programming language for scripting and automation tasks [8].

3.4 Computer Networks

Computer networks can be defined as the collection of interconnected computers and other devices that are linked together to facilitate communication and data sharing. At its core, computer networks consist of nodes and links. Nodes refer to the equipment used for data communication, such as modems, computers, or Ethernet switches, while links represent the physical mediums like wires, cables, or the open space in wireless networks. The functioning of computer networks can in its shortest form be described as the utilization of protocols or rules that facilitate the transmission and reception of data through the links. A protocol comprises a collection of rules and standards that govern the transmission of data across a network.

3.4.1 Internet Protocol

The Internet Protocol (IP) is a communication protocol for relaying datagrams across network boundaries. Its routing function enables internetworking and essentially establishes the Internet. An IP address can be broken down into two parts: the network portion and the host portion.

The network portion identifies the network to which the device belongs. The network portion is determined by the subnet mask applied to the IP address. The subnet mask helps define what part of the IP address is reserved for the network portion. The subnet mask is mainly expressed in two different forms *Dotted Decimal Notation* and *Classless Inter-Domain Routing (CIDR)* Notation. *Dotted Decimal Notation* is when the subnet mask is expressed as 4 bytes separated by dots (255.255.255.0). *CIDR* represents the subnet mask as an IP address followed by a forward slash (/) and the number of network bits (129.241.170.69/24) where both examples have the same subnet mask size. If a node wants to send data to another node on a different network the data is first sent to the default gateway. The default gateway is usually a router that handles communication between different networks.

The host portion identifies the specific device within a network. It is defined by the portion of the IP address not reserved by the subnet mask.

The division of the IP facilitates routing and delivering of data packets to the appropriate destination within a network. The network portion helps determine the network to which the IP address belongs, while the host portion identifies the specific device within that network. If the network portion of the receiver is not the same as for the sender the data packets are sent to the default gateway[28].

3.4.2 Secure Shell

Secure Shell (SSH) is a secure network protocol for remote access and management of systems. It utilizes encryption to provide a secure environment where remote systems can be accessed for executing of commands and transferring of files. SSH is widely used and has in this project been essential for communicating with the RPi [45].

3.5 Python

Python is a popular and powerful programming language known for its simplicity, readability, and extensive library ecosystem. Its clean and straightforward syntax makes it easy to learn and understand. With a rich collection of libraries and frameworks, Python caters to diverse domains like data analysis, rapid prototyping projects, and more. Its strong community support provides resources and assistance to developers, while its cross-platform compatibility ensures applications can run seamlessly on different systems.

In addition to its general strengths, Python's support for object-oriented programming is particularly advantageous for building more complex programs. Object-oriented programming allows developers to create reusable and organized code through the use of classes and objects. By encapsulating related data and functionality within classes, the code becomes modular and easier to maintain. This architectural approach is well-suited for complex applications where different components and sensor types need to be managed and controlled efficiently [39].

3.5.1 Classes

A class is a blueprint that defines the properties (attributes) and behaviors (methods) that objects of that class possess. Object-oriented programming promotes the concepts of encapsulation, inheritance, and polymorphism:

- Encapsulation involves bundling data and methods together within a class, allowing for data abstraction and controlling access to the internal state of objects. This ensures that data is protected and can only be accessed and modified through defined methods.
- Inheritance enables the creation of new classes (derived classes) based on existing classes (base classes). The derived classes inherit the attributes and behaviors of the base class, which promotes code reuse and enables the creation of specialized classes.
- Polymorphism allows objects of different classes to be treated as objects of a common superclass. This allows for the creation of more generic code that can operate on objects of different types, providing flexibility and modularity.

Comprehending the principles of object-oriented programming, and the concepts explained above becomes essential when exploring the Robot Operating System (ROS) [34].

3.6 Robot Operating System

The Robot Operating System (ROS) is an open-source and flexible framework designed for building robotic systems. It provides a collection of software libraries, tools, and capabilities that aid in the development of robot applications. ROS offers a distributed architecture that allows modular and independent components, known as nodes, to communicate with each other through messages, services, and other mechanisms.

ROS supports several programming languages but the biggest and most widely adopted language is Python. It offers excellent integration with ROS and provides a high-level interface for developing ROS nodes, handling communication, and accessing the wide range of ROS functionalities.

The structure is based on a distributed and modular architecture that allows for the development of complex systems. The key components of the ROS structure are as follows:

- Nodes are the fundamental building blocks of ROS. They are independent units of execution that perform specific tasks. Nodes can communicate with each other by publishing and subscribing to messages over topics or by providing and using services.
- Messages define the data structure and content that nodes exchange over topics. They can represent sensor data, commands, status information, or any other relevant information for the system.
- Topics facilitate communication and data exchange between nodes in a publish-subscribe manner. Nodes can publish messages on a topic, and other nodes can subscribe to those topics to receive the messages. Topics enable asynchronous and decoupled communication, allowing nodes to operate independently and at their own pace.
- Services provide a synchronous and request-response style of communication between nodes. A node can offer a service, specifying a request message and a response message. Other nodes can call that service to send a request message and receive a response in return.
- Packages are a way to organize and share ROS code. A package contains libraries, executables, configuration files, and other resources related to a specific functionality or component. Packages provide modularity, code reusability, and ease of distribution, allowing developers to share their work and leverage existing ROS packages.
- The ROS Master is responsible for facilitating the discovery and registration of nodes, topics, and services. It helps nodes find each other and establish communication channels. The ROS Master maintains a centralized registry of information about the available nodes and their communication endpoints.

By leveraging this structure, ROS enables the development of flexible and scalable systems. It promotes code reuse, modular design, and interoperability among different components, making it easier to build complex and collaborative applications [40].

3.7 3D modeling and printing

Designing prototypes for projects like this one is easy with the help of Computer-aided Design, commonly referred to as CAD. It is used to create and design virtual 2D and 3D models of objects, buildings, machines, and various products. It allows engineers to create precise and accurate digital designs, visualizing the product before it is physically manufactured. CAD software is essential in modern prototyping and manufacturing as it helps to streamline the design and development process, reducing costs and time to market. It can be used to create digital models of products or parts, which can then be simulated, tested, and refined before any physical prototyping takes place.

CAD software plays a vital role in 3D printing as it is used to create a 3D digital model of an object, which can then be printed layer by layer using a 3D printer. The process includes the following steps:

1. Using CAD software, create a digital 3D model of the object you want to print. This model can be created from scratch or imported from existing designs.
2. Once the 3D model is complete, export it in a file format compatible with the 3D printer you will be using. Common file formats for 3D printing include STL, OBJ, and 3MF.
3. In preparation for printing, the 3D model needs to be sliced into thin layers. This is typically done using slicing software that generates a G-code file that the 3D printer uses to print the object layer-by-layer.
4. Load the G-code file onto the 3D printer and initiate the printing process. The printer will begin printing the object layer-by-layer, following the instructions in the G-code file.
5. Once the object is printed, post-processing may be required, such as removing support structures, sanding or polishing the object, or painting it.

CAD software is essential in the 3D printing process, as it allows designers and engineers to create precise and accurate 3D models that can be printed quickly and efficiently. With CAD software, it is possible to make modifications to the digital model, improving the accuracy and functionality of the final printed object.

3.7.1 Fusion 360

The CAD-software used in this project is called Fusion 360. Fusion 360 is a cloud-based 3D modeling software developed by Autodesk, which offers a powerful and versatile set of features for creating and testing designs. In recent years, Fusion 360 has gained popularity among engineers and product designers due to its unique combination of parametric modeling, cloud-based collaboration, and use in 3D-printing simple prototypes.

One of the key features of Fusion 360 is its ability to support both parametric and direct modeling. Parametric modeling allows designers to create a model by defining parameters and constraints, which can then be adjusted to modify the design. This allows for greater design flexibility and enables designers to create complex models quickly and easily. Additionally, the parametric modeling feature enables designers to modify their designs at a later stage, making it easier to make modifications to the design without the need to start from scratch.

Another important feature of Fusion 360 is its cloud-based collaboration tools. As a cloud-based software, Fusion 360 allows designers to work on the same project simultaneously, share designs with others, and comment on designs in real time. This makes it particularly useful for distributed teams or those working remotely. Furthermore, the software's collaboration tools enable designers to work more efficiently and effectively, ensuring that designs are completed on time and to a high standard.

Fusion 360's compatibility with 3D printing technology also makes it a valuable tool for prototyping simple designs. The software allows designers to create accurate 3D models that can be exported in a file format compatible with 3D printers. The ability to create and print prototypes quickly and easily can be particularly valuable in product design, where testing and prototyping are critical to the design process.

3.8 Sensors

This section explores the sensors used by the sensing rig. These sensors are vital for acquiring accurate data from the underwater environment, supporting scientific research, environmental monitoring, and underwater exploration. The sensors enable measurement of parameters such as temperature, pressure, salinity, dissolved oxygen, and the power level of the rig itself. By understanding these sensors, their capabilities, and their limitations.

3.8.1 Pressure Sensor

The pressure sensor is made by BlueRobotics and is called "Bar30 High-Resolution 300m Depth/Pressure Sensor" [4]. The sensor itself is the Measurement Specialties MS5837-30BA, which can measure up to 30 bar (300m depth) and communicates over I2C. BlueRobotics have sealed the sensor from water and made it ready to install in a watertight enclosure. The sensor has a relative accuracy of ±200 mBar (204 cm in fresh water) and a resolution of 0.2 mBar (2 mm in fresh water). It requires a supply voltage 2.5 - 5.5 V. BlueRobotics also specifies that **the sensor must be completely dried once per day or the pressure and temperature readings will drift.**

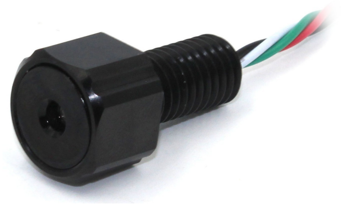


Figure 3.2: BlueRobotics pressure sensor.

3.8.2 Power Sense Module

The power sense module is made by BlueRobotics, it provides analog current and voltage values from the battery. It uses a hall effect current sensor for excellent accuracy at low current draw, the voltage sensing method is not listed. It has a maximum input voltage of 25.2V. It is originally made for usage with a flight controller but the sensor data can be extracted with an ADC. The PSM comes with a 6p JST-GH connector that is used to output the sensor data [36].

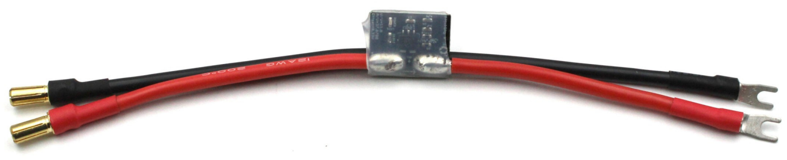


Figure 3.3: BlueRobotics Power Sense Module.

3.8.3 Dissolved Oxygen Sensor

The dissolved oxygen sensor is developed by Atlas Scientific. The sensor comes in a kit that consists of 1 EZO™ Dissolved Oxygen Circuit, 1 Lab Grade Dissolved Oxygen Probe, 1 30ml bottle of Electrolyte Solution, 3x 20ml Zero Dissolved Oxygen calibration solution pouches, 1 Electrically isolated EZO™ Carrier Board, and 1 syringe.

The galvanic dissolved oxygen probe consists of a PTFE membrane, an anode bathed in an electrolyte and a cathode. The working principle is that oxygen molecules defuse through the probes membrane at a constant rate. The membrane is there to slow down the reaction to make sure it does not happen too quickly. When the oxygen molecules have crossed the membrane they meet the cathode and are reduced. This reduction produces a small voltage which can be measured, this voltage is from 0 mV to 60 mV. If there are no oxygen molecules present, the probe will output 0 mV. As the oxygen increases so does the voltage output from the probe until it has reached the maximum voltage of 60 mV. All probes from Atlas Scientific output a different voltage in the presence of oxygen, even as they output 0 mV in the absence of oxygen. A drawback of using a galvanic probe is that it consumes a very small amount of the oxygen it reads. Hence, to take accurate readings a small amount of water movement is necessary. (Approximately 60ml/min).

The sensor has a range 0 - 199 mg/L, accuracy of $\pm$ 0.05 mg/L, response time of $\approx$ 0.3 mg/L/per sec, and is rated for a maximum depth of 352 m (3 447 kPa) [14].

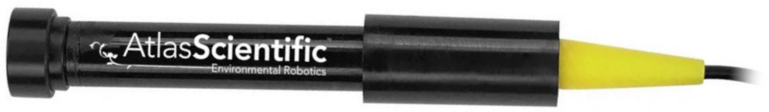


Figure 3.4: Dissolved oxygen probe.

3.8.4 Conductivity Sensor

The salinity sensor also referred to as the conductivity sensor is also developed by Atals Scientific. The sensor comes in a kit that consists of 1 EZO™ Conductivity Circuit, 1 Conductivity Probe, 1 Electrically Isolated EZO™ Carrier Board and 2 125ml calibration solutions consisting of one 12 880μS and one 80 000μS.

The operating principle is that the probe measures the electrical conductivity of a solution. This is done by applying an AC voltage between two electrodes positioned opposite from each other. The AC voltage causes cations to move to the negatively charged electrode, while the anions move to the positively charged electrode. A higher level of free electrolytes in the liquid translates to higher electrical conductivity. A crucial part of getting accurate readings is to ensure that there are no air bubbles caught between the two graphite plates.

The sensor has a range of 5 - 200 000 $\mu S/cm$, accuracy of $\pm 2\%$, response time of 90% in 1s, and is rated for a maximum depth of 352 m (3 447 kPa) [12].

Figure 3.5: Conductivity probe.

3.8.5 Temperature Sensor

The temperature sensor is made by BlueRobotics and called "Celsius Fast-Response, ±0.1°C Temperature Sensor (I2C)" [9]. The sensor itself is Measurement Specialties TSYS01, which is accurate to ±0.1°C and communicates over I2C. TSYS01 has a fast time response and BlueRobotics designed the whole package to maintain that speed to allow accurate temperature profile measurements even while descending/ascending quickly. The sensor has a response time constant of $\tau = 1$ second (with 0.5 m/s flow) and $\tau = 2$ seconds (in still water). Furthermore, the sensor is sealed from the water, protected by an aluminum cage, and ready for installation in a watertight enclosure.

The sensor requires a supply voltage of 3.3 - 5.5 V, it is rated for depths from 0 - 975 meters and operating temperatures from -40 to +125°C [9].

Figure 3.6: BlueRobotics temperature sensor.

3.9 Interface electronics

Several hardware components were employed to ensure that all sensors receive power, can transfer data, and function as intended. These components will be discussed in this subsection.

3.9.1 Inter-Intergrated Circuit Bus Splitter

To facilitate information flow between the sensors and the RPi, the I^2C protocol was used. Since there are 5 sensors and the RPi only have one connection for I^2C, a I^2C bus splitter was needed. The splitter allows connection to multiple I^2C devices by sharing the same bus. It consists of four header pin holes, four DF13 connectors, and four JST-GH connectors, making it compatible with multiple devices with different headers.

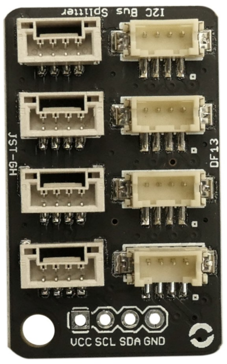


Figure 3.7: I^2C bus splitter.

3.9.2 Power supply

In this project two different power supplies have been utilized, the reason for this is that the power supply that was utilized in previous iterations of the project turned out to be unreliable and at risk of destroying the RPi.

3.9.2.1 Traco Power DC/DC converter [New]

A DC/DC voltage converter is needed to convert the voltage between the battery and the RPi. The battery outputs a voltage of 22.2V, this is too much for the RPi which requires 5V to run. The DC/DC converter used in this project is the THN 15-2411WIR. It is a robust and reliable 15 W DC/DC converter with plenty of features. Some features include increased resistance against electromagnetic interference, shock/vibration, and thermal shock, and most importantly for this project is that it has short circuit protection to prevent it from frying the RPi.

The DC/DC converter accepts an input voltage from 9V - 36V DC and outputs a maximum current of 3A and is rated with an efficiency of 90% [49].

Figure 3.8: DC/DC converter.

3.9.2.2 Blue Robotics power supply [Old]

The power supply utilized in previous iterations of this project is made by Blue Robotics. It is a DC/DC converter and accepts a supply voltage from 7V - 26V and outputs a 5V DC voltage with a maximum 6A current. It was decided to not use this power supply due to the risk of frying the RPi. As both this group and the previous group had done [2].

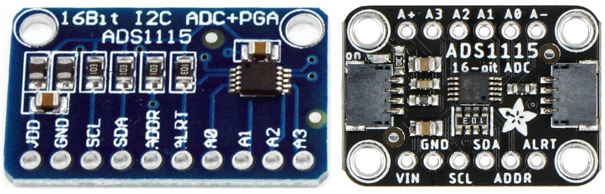
Figure 3.9: Old power supply.

3.9.3 ADC

To be able to read the values from the PSM a Analog Digital Converter (ADC) was needed. An ADC is a device or component that takes an analog signal as input and converts it into a digital value represented in binary form. The ADC measures the strength or magnitude of the analog signal relative to a reference voltage and transforms it into discrete digital levels or bits. This process allows the analog signal to be processed and manipulated digitally by a computer or digital system. For this project, the ADS1115 produced by Adafruit was chosen. It should be noted that since the project was started Adafruit has updated the design of this ADC, this will be discussed in more detail in Section 7.1.1 *Issues regarding the main plate.*

The ADC is 16-bit and can communicate through I^2C, it requires an input voltage of 2 - 5V. It has 4 different single-ended input channels or if desired it can be configured to have two differential channels [26]. The new version also supports the use of *Stemma QT*-connectors for both power and communication using the I^2C-bus protocol.

$$value = \frac{reading * 2^{15}}{3.3V} \tag{3.1}$$



(a) Old design. **(b)** New design.

Figure 3.10: Adafruit ADS1115.

3.9.4 Ethernet switch

For further development and robustness of the system a Ethernet switch was implemented. The switch that was chosen is developed by BotBlox and is called GigaBlox - Small GigaBit Switch. It is a tiny and compact Ethernet switch with five 10/100/1000M ports. It runs on voltage from 6 to 60V with a tolerance up to 65V. The switch uses a non-blocking fabric, this means that it is capable of transmitting 1 Gbps simultaneously over all five ports. What makes the switch so small is that all Ethernet connectors are of the type 8p PicoBlade which is much smaller than the standard RJ-45. This also means that all standard Ethernet cables that is to be connected to the switch needs to be spliced [18].

Figure 3.11: GigaBlox Ethernet switch.

3.9.5 Barrier strip

Two barrier strip was installed to organize the main power to the sensors and modem. Each barrier strip consists of 6 connectors that are all joined together. There is one strip for the high voltage and one for the low making it easy to connect all components that need power. The barrier stips used in this project are made by Molex and have a voltage rating of 300V and 15A. It is also rated for wires between 22 and 14 AWG or cables with a diameter between 0.644 and 1.628mm [1]. In This project, a jumper is installed to ensure that all terminals are shorted.

3.10 Battery

To supply the sensing rig and the modem with power a battery was needed. The battery chosen in this project was a 12000mAh Lipo battery produced by GensTattu. The battery consists of 6 cells that are connected in series which gives it an average voltage of 22.2V, though the voltage varies between 19.8V at 0% charge and 25.2V at 100% charge. The minimum cell voltage is described to be 3.3V in the data sheet for this specific battery [48]. The discharge rate is 15C and it is rated for a max burst discharge rate of 30C. The battery comes with one AS150 on the positive cable and an XT150 connector on the negative cable, though they are going to be replaced.

The sensing rig is not dependent on any specific brand of battery though there are some parameters that have to be changed in the code if a battery that is not 6S1P is chosen[47].

Figure 3.12: Battery.

3.11 Understanding Dissolved Oxygen in Water

The dissolved oxygen concentration in water is mainly affected by three factors. These factors are temperature, salinity, and pressure.

Temperature
Temperature is one of the biggest, if not the most, common factors that directly affect DO in water. Generally, cold water can hold more dissolved oxygen than warm water. This is due to the physical properties of water and the effects of temperature on gas solubility.

The solubility of gases in liquids increases as the entropy of the system decreases. When water is cold, its molecules are closer together, and there is a more ordered arrangement of water molecules. This leads to a decrease in the entropy of the water molecules. As a result, the decrease in entropy favors the dissolution of gas molecules, such as oxygen. Conversely, warm water has increased thermal energy which causes the water molecules to move more rapidly, leading to higher entropy. The higher entropy makes it more difficult for gas molecules, like oxygen, to dissolve in the water.

Salinity
Salinity, which refers to the concentration of dissolved salts in water, also affects the solubility of gases, including oxygen. In general, increasing salinity with sodium chloride tends to decrease the solubility of gases in water and vice versa. This is because the presence of dissolved salts disrupts the molecular interactions between water molecules, making it more difficult for gas molecules to dissolve. Generally the relationship between dissolved oxygen and salinity is exponential and at the same pressure and temperature, seawater holds about 20% less dissolved oxygen than freshwater.

Pressure
Pressure also plays a role in the solubility of gases, including oxygen. Generally, increasing pressure enhances the solubility of gases, while decreasing pressure reduces their solubility.

There are also other factors that affect the dissolved oxygen concentration in the real world. Some of these are microbial decomposition, lack of atmospheric contact for diffusion, and the absence of photosynthesis [13].

3.12 Waterproofing

For waterproofing the electronics, a solution provided by Blue Robotics was used. The solution is to encapsulate all electronics in a housing consisting of three main parts, the three different parts come from the modular capabilities of Blue Robotics. The biggest part is an aluminum cylinder, in this project, the cylinder is 4" and it houses all electronics that are not already waterproof. The flange and the cap combined make the lid that encloses the housing and makes it waterproof. One rig consists of two assembled cylinders, one for holding the RPi and the other for the battery.

3.13 Subnero Acoustic Modem

The acoustic modem used in this project is made by Subnero. The specific modem used is the 3. generation research edition modem (Model number: WNC-M25MRS3). This modem is now discontinued and removed from their website. The modem is designed to bridge the gap between development and high-end commercial deployments. It works by using sound waves that travel through water to transmit information. This is done with a transducer that converts electrical signals to acoustic signals. The transducer is an underwater speaker and microphone, and it can also convert acoustic signals to electrical.

The modem has a nominal operating range of 1 km, and it has an operating temperature from 0 to 40°C. Its operating depth is 100 m and its power consumption varies from 4 to 25 W depending on if it is receiving or transmitting. When transmitting the modem can reach speeds up to 15 kbps.

3.13.1 UnetStack

UnetStack is an open-source software stack designed for underwater communication networks. It provides a framework for developing and implementing communication protocols and applications for underwater networks. UnetStack is a collection of technologies to extend communication networks underwater and it does this by providing a comprehensive set of tools, libraries, and protocols for several different programming languages.

Chapter 4

Environment and Software

This chapter explores the essential steps and considerations involved in setting up and configuring the software environment. This proceeding sections delves into both the configuration of a software environment and how the program meant to run in said environment is structured and built. Additionally, tutorials and information about how to run and maintain the program will be given.

4.1 Flashing Operating System to SD-Card

In order to use the RPi, an OS had to be flashed. As discussed in section 3.3, Ubuntu server was chosen to be the OS. The task of flashing Ubuntu server on the RPi is rather simple. The ensuing tutorial will shed light on the procedure of flashing an OS onto an SD card for RPi from an Ubuntu computer. Nonetheless, the fundamental steps of this method may also be applicable to computers with other OSs.

The first step is to insert the SD card that is going to be used for the RPi in the computer. Then the Raspberry Pi Imager can be downloaded, in Ubuntu this can be done by opening the terminal and pasting the command shown in listing 4.1. Once the Raspberry Pi Imager is downloaded it can be opened and the leftmost button labeled *CHOOSE OS* can be selected. This will open a new window where *Other general-purpose OS* can be selected followed by *Ubuntu* and then for this tutorial *Ubuntu Server 22.04.2 LTS (64-bit)* is selected. After the Operating System has been selected the rightmost button labeled *CHOOSE STORAGE* can be selected, then select the SD-card the OS is to be installed on (do not press *WRITE* yet).

```
$ sudo snap install rpi-imager
```

Listing 4.1: Command for installing RPi Imager.

Once the SD card has been chosen, the *Settings* button, which is distinguishable by its depiction of a gear icon, can be clicked. In this window the *hostname* of the RPi can be selected, a username and password can be defined and Secure Shell (SSH) should be enabled. To ease further configuration the *Set locale settings* button can be selected to configure the time zone and keyboard layout to the appropriate region. After all the settings are set they can be saved and written to the SD-Card by pressing *SAVE* and *WRITE* [22].

4.1.1 First Boot Configuration

It is advisable to connect the RPi to a monitor and keyboard during the first boot to facilitate any necessary configuration adjustments. This will ensure that the RPi later can be used seamlessly without a monitor and keyboard, enabling a headless setup.

During the first boot a tool called cloud-init is doing configuration, wait for it to finish before trying to log in. It usually takes less than 2 minutes. When it finishes some lines will be outputted to the terminal.

4.1.2 Setting Static IP

To use the RPi fully headlessly it is essential to configure a static IP address. This is necessary to allow the user to know which IP address to connect to via SSH. The easiest way to do this is to manually set the IP address. Setting the IP address manually can be easily accomplished, and this tutorial is designed to be universally applicable to all Linux systems.

The first step is to boot the system, login and connect the RPi to Ethernet. After this the command shown in listing 4.2 can be used to find the subnet mask and Ethernet interface of the device. The Ethernet interface is recognized by starting with **e** (for the RPi it will be eth0). If the device is connected to Ethernet it will also show the current IP and subnet mask like shown in line 5, listing 4.2. If the device is not connected to Ethernet it will not show a IP address or a subnet mask.

To configure a static IP address, the initial step is to locate the relevant *.yaml*

```
$ ip a
...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state
 ↪  UP group default qlen 1000
    link/ether e4:5f:01:27:63:11 brd ff:ff:ff:ff:ff:ff
    inet 129.241.170.225/24 metric 100 brd 129.241.170.255 scope
     ↪  global dynamic eth0
       valid_lft 10489sec preferred_lft 10489sec
    inet6 fe80::e65f:1ff:fe27:6311/64 scope link
       valid_lft forever preferred_lft forever
...
```

Listing 4.2: Network Characteristics.

configuration file. This can be done by using the commands shown in listing 4.3. It should be noted that the filename for the configuration file may vary depending on the particular system in use. Nevertheless, in the folder, there is only one file, which is the correct configuration file.

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

Listing 4.3: Navigating to Netplan.

In order to implement the required modifications, the configuration file should be adjusted to resemble the code presented in listing 4.4. It is worth noting that the assigned IP address and subnet mask will depend on the specific network to which the user intends to connect [33].

4.2 Updating System

It is a good idea to update the system especially right after a new OS is installed. Also many tutorials and dependencies require the system to be up to date before installation. To update the system run the command shown in listing 4.5. The update command is used to resynchronize the package index files from their sources. The upgrade command is used to install available upgrades of all packages currently installed on the system from the sources [19].

Many installations such as ROS2 Humble require this command to be run before

```yaml
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    eth0:           # Depends on what is found when running `ip a`
      addresses:
        - 192.168.42.70/24 # [IP]/[subnet mask]
      routes:
        - to: default
          via: 192.168.42.1 # Default gateway
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
# This part is optional and contains information about connection to
↪  WiFi
  wifis:
    wlan0:
      dhcp4: true
      optional: true
      access-point:
        '<WiFi-name>':
          password: '<Password>'
```

Listing 4.4: 50-cloud-init.yaml.

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

Listing 4.5: Updating system.

the actual installation can start. If it is the first time that the system is upgraded it may take some time as there are many packages that need upgrading.

4.3 Program-Dependent Packages and Libraries

In order to execute the program on the Raspberry pi, it is necessary to install certain external libraries. This section will provide an overview of all the non-standard libraries that need to be installed in order to ensure proper program functionality.

- To interact with the EZO-circuit used by the Atlas Scientific sensors it is essential that *smbus* and *i2c-tools* have to be installed. The *smbus* package is responsible for communication between python3 code and the I^{2}C devices. The *i2c-tools* package provides a collection of command-line tools for working with devices connected to I^{2}C. Both packages can be downloaded with the commands below.

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

The following packages are installed with the Python package manager (PIP). If PIP is not installed it can be done with the following command.

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

- *unetpy* is the Python package responsible for interacting with modems running UnetStack. This package needs to be installed in order to get any communication with the modem.

```
$ pip install unetpy
```

- *Adafruit_ GPIO* is the package responsible for communicating with the ADC. The ADC uses a specific library that relies on this package provided by their manufacturer instead of the standard *smbus*.

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

4.4 Downloading ROS

To run the program for the Internal Electronic System (IES) ROS2 Humble have to be installed on the RPi, this can be done by following the tutorial provided by Open Robotics [51]. Adhere to the Debian installation guide and install the ROS-Base version. ROS-Base is a bare bones version of ROS that excludes graphical features that are useless on a headless computer.

To simplify usage, it is also recommended to execute the commands provided in listing 4.6 upon completion of the installation process. The command in line 1 will eliminate the need to source the environment for every new terminal window that is opened. This is done by adding *source /opt/ros/humble/setup.bash* to the *bashrc* file that defines environment variables, aliases, and shell functions that customize the behavior of the Bash shell (terminal). The command in line 4 will make auto-completion of ROS2 commands possible, but before this command can be initiated, make sure that *colcon common extensions* is downloaded by running the command in line 3. Auto-complete will help save time and lower the risk of errors caused by typos. Finally to get the updates to the current terminal *bashrc* needs to be sourced as done in line 5.

```
1  $ echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
2
3  $ sudo apt install python3-colcon-common-extensions
4  $ echo "source
    ↪  /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash" >>
    ↪  ~/.bashrc
5  $ source ~/.bashrc
```

Listing 4.6: Sourcing ROS2 Dependencies.

4.4.1 Downgrading Setuptools

To be able to build the ROS2 environment the package *setuptools* may need to be downgraded. As of 30.05.2023 setuptools have to be downgraded from version 59.6.0 to 58.2.0. The downgrade can be completed by following the commands shown in listing 4.7. In order to run this command ensure that pip is installed as described in Section 4.3 *Program-Dependent Packages and Libraries*.

```
# Check what version of setuptools that is currently installed
$ pip3 list | grep setuptools
setuptools                             59.6.0

# Downgrade to version 58.2.0
$ pip3 install setuptools==58.2.0
```

Listing 4.7: Downgrading Setuptools.

4.5 Running the Program

This section will explain the process for setting up the program on the RPi. Features like automatic startup when the RPi is connected to power and correct timekeeping will also be explained and guides for setting it up is provided.

4.5.1 Maintaining Correct Time

Due to Raspberry pi not having an on-board Real Time Clock (RTC) it does not have a built-in mechanism for maintaining accurate time, and it relies on external sources such as the Internet to set the time during boot-up. This results in time discrepancies over periods where the RPi is booted without an Internet connection. In order for SUMS to keep correct time it either has to be connected to the Internet or the time will have to be set manually before the program is started. To manually set the time use the command shown in listing 4.8

```
$ sudo date --set="YYYY-MM-DD HH:MM:SS"
```

Listing 4.8: Command for Manually Setting Time on RPi.

4.5.2 Downloading Program to Raspberry Pi

The easiest way to download the program is to clone the repository from GitHub. This method requires that the device intended to run the program must be connected to the Internet. The instructions for downloading through GitHub can be found in the README.md file located in the GitHub repository[42]. In situations where the RPi does not have Internet access, it may be more convenient to download the repository to a computer, before manually transferring it to the appropriate folder in the RPi-image on the SD card. This can be accomplished by inserting the SD card in the computer containing the program. Two new drives should appear, open the drive labeled *writable*. This drive contains all data that can be modified by the user and the Operating System. The recommended approach is to place the file in the */home/<user>/* directory.

4.5.3 Enable Automatic Launch on Reboot

To start the ROS2 program automatically when booting up the RPi, one can use systemd, a popular init system in Ubuntu and other Linux distributions, which handles boot configuration. Here's a step-by-step guide to setting it up:

1. Create a systemd service unit file by running the command:

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

2. Edit the service unit file entering the lines in Listing 4.9. Remember to 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 4.9: ros2_SUMS_startup.service.

3. Save and close the file by pressing Ctrl + X followed by Y then Enter.

4. Enable and start the service using the following commands:

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

Now, whenever the RPi boots up, the ROS2 program specified in the service will be automatically launched. The service will first, from the working directory, source the *setup.bash*. This ensures that the shell is able to find the project's launch file and everything else connected to the project. Next, it launches the program by running every node in the SUMS workspace.

To verify the running status of each node following the service startup and RPi reboot, execute the following command to display all active ROS2 nodes:

```
$ ros2 node list
```

To see the output of the nodes one must use the status command as shown in line 4 in Listing 4.10. This command will display the status of the service as well as the last couple of output lines from the shell running the nodes. If it is desirable to kill all the nodes, the easiest way is by stopping the service. This is done by using the command on line 1 in Listing 4.10. To disable the service and reboot without automatic launch, enter the commands in line 2 and 3 in Listing 4.10.

```
1  $ sudo systemctl stop ros2_SUMS_startup.service
2  $ sudo systemctl disable ros2_SUMS_startup.service
3  $ sudo reboot
4  $ systemctl status ros2_SUMS_startup.service
```

Listing 4.10: Useful systemd commands.

4.6 Handling Sensor Data

This section will discuss the changes, additions, and the flow of the ROS2 program that runs all the sensors and logic of the rig. As mentioned earlier this project is a continuation of a previous bachelor thesis, the group's focus has been on adding capabilities and streamlining the existing program.

4.6.1 Program Structure

The program was designed in compliance with the ROS2 Humble architecture and implemented using the Python programming language. It can be broken down into three main parts as visualized by figure 4.1, the figure can be seen in full scale in Appendix A.

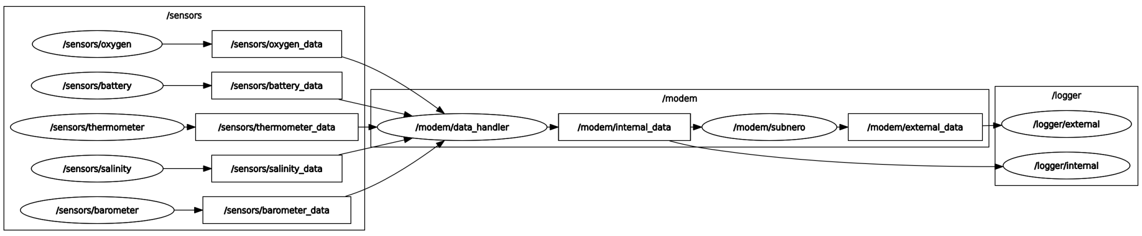


Figure 4.1: ROS2 rqt_graph.

The first and biggest part is the sensor part, which contains all the sensors that gather data about the environment around the rig. It also includes a sensor that measures voltage from the battery. All sensors are implemented in different packages as separate nodes. To keep the data flow separated and easy to maintain, each node publishes on a distinct topic.

The modem part is made up of two nodes implemented in one package. The first node subscribes to all topics from the sensors. When the modem node has recognized that all topics have been updated it formats all data for transmission through the modem and publishes this data to the topic (internal data). The second node is responsible for communicating with the modem, it subscribes to the *internal_ data* topic and whenever this topic is updated the node tells the modem to transmit the data. While the modem is not sending data it is listening for data sent by other agents, if such data is received the node will publish the received data to a topic reserved for data received by the modem (*external_ data*).

The last part is the logger, it consists of one package that contains two nodes. The first node is the internal logger that subscribes to the internal data topic. The node then saves the data from the topic into a file. The second node is quite similar only in that it saves the data from the external data topic in a different file.

4.6.1.1 Sensor Nodes

There are in total five different sensors, four sensors that are sensing external data and one sensing internal data. Despite the sensors being physically different, the code for running them are very similar, not counting the power sensor. To keep the workspace organized all sensors were implemented in their own package. This makes it easy to identify what files and libraries that are relevant for what node/package. Furthermore, the main function and the class definition where all logic is placed were also divided into two files. All sensors rely on a library that is placed in each package.

4.6.1.2 Libraries

A library is essential for the collection of sensor values, it provides a structured and organized way to interface with the sensors, read their values, and process the data. All sensor nodes utilize different libraries to successfully sens the environment. All sensors except the sensors produced by Atlas Scientific utilize a stock library created by the manufacturer. The libraries for the Atlas Scientific sensors are a combination of the generic library Atlas Scientific provide [3] and the library for the temperature sensor [50]. The libraries referred to are the *catlas01.py* and the *doatlas.py*, which respectively belong to the conductivity and dissolved oxygen sensors.

Due to the two Atlas Scientific libraries being a hybrid of two libraries, they were messy. An effort was made to organize and document the library for easier usage. An *init()* function that checks the connection with the sensor was also implemented to facilitate equal code through all sensors and make debugging easier.

4.6.2 Node Main Functions

All nodes require to be initialized, as mentioned earlier this is done in its own file. The code shown in listing 4.11 will give a general overview of how the main files are structured.

```python
import rclpy
from <Package Name> import <Node File> as node

def main(args=None):
    rclpy.init(args=args)

    # Construct the node
    <Variable> = node.<Class name>()

    # Start the node's event loop
    rclpy.spin(<Variable>)

    # Clean up when script is stopped
    <Variable>.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Listing 4.11: Node Main Function.

First *rclpy* is imported, this is the ROS Client Library and it provides tools for initializing and interact with the ROS2 system. Then the file where the node is defined is imported, for the temperature sensor the *<Package Name>* would be *sensor_ thermometer* and the *<Node File>* would be *thermometer_ data_ publisher_ node*. After everything is imported the *main(args=None)* function is defined, it is defined so that if no arguments are given it defaults to none. Subsequently *rclpy.init(args=args)* is called to initialize the ROS 2 communication infrastructure and sets up the node to communicate with other nodes in the ROS 2 network. The argument *args=args* ensures that the arguments passed in the main function are also passed in the *init()* function. After the node is initialized it can be constructed and given a variable name by running *<Variable> = node.<Class name>()* where *<Variable>* can be any unused variable to make the code clear to understand. To make the node run more than one time and make callback functionalities usable the node needs to be spun with *rclpy.spin(<Variable>)*. When the program eventually have to be stopped *rclpy.shutdown()* shuts down the node, everything inside the node and all ROS2 communications. Lastly the if statement ensures that the main function can only be run from a terminal, not if it is imported in another file. This is standard practice when making modules in pyhton.

4.6.3 Sensor Interfaces

All nodes are utilizing custom message types to communicate, these messages can be found in the folder *src/sensor_ interfaces/msg/*. The messages are made up by some variables and type definition of said variables. Listing 4.12 illustrates what the custom message contains.

```
float64 battery_percent
float64 battery_voltage
float64 battery_current
string local_time
```

Listing 4.12: Battery message.

4.6.4 Environment Sensors

Since all sensors are generally structured in the same manner, a general explanation for the temperature sensor will be given.

4.6.4.1 Code Walkthrough

```python
from rclpy.node import Node

from sensor_thermometer import tsys01
from sensor_interfaces.msg import Thermometer
import time
```

Listing 4.13: Temperature Sensor Imports.

The code showcased in listing 4.13 imports all necessary libraries. The *rclpy.node* library contains a set of tools for creating and managing a ROS node. The *tsys01* is the library for the temperature sensor, this library will vary depending on what sensor code is inspected. After this the custom message type *Thermometer* is imported, this message enables easy and clear data transfer within ROS. Lastly *time* is imported to fetch the local system time.

```python
class ThermometerDataPublisher(Node):

    def __init__(self):
        super().__init__('ThermometerDataPublisher')
        self.publisher_ = self.create_publisher(Thermometer, 'thermometer_data', 10)
        self.sample_time  = self.declare_parameter('sample_time', 2.0).value
        self.timer = self.create_timer(self.sample_time, self.thermometer_read_and_publish)

        self.sensor = tsys01.TSYS01()
        if not self.sensor.init():
            self.get_logger().error("Sensor could not be initialized")
            exit(1)
```

Listing 4.14: Temperature Sensor Constructor.

Listing 4.14 displays the class *ThermometerDataPublisher()* is defined, it inherits from *Node* as defined earlier. After this, the class constructor *__init__()* is defined and the first operation is to run *super().__init__()*. This will initialize the parent class constructor and ensure that the necessary components and services are properly initialized. Subsequently a publisher is defined to publish messages of the type *Thermometer* to the topic *thermometer_data* with a queue size of 10. Thereafter the parameter *sample_time* is declared and given the default value of

2. After the sample time is set, a timer can be created. The timer will run the *thermometer_read_and_publish* function every *sample_time* second.

The code from line 17 in listing 4.14 initializes the sensor, first an instance of *TSYS01* from the module *tsys01* is defined as *sensor*. After this the member function *init()* of *TSYS01* is run, note that *TSYS01.init()* $\neq$ *TSYS01.__init__()*. *TSYS01.init()* is a function that initializes and calibrates the sensor and is required to be called before using any other methods [50]. The *init()* function returns *True* if the sensor was successfully initialized and *False* otherwise. It is called within an if statement that logs an error message if the sensor could not be initialized.

```python
    def thermometer_read_and_publish(self):
        msg = Thermometer()

        # Getting the local time
        current_time = time.localtime()
        msg.local_time =  time.strftime("%H:%M:%S",current_time)

        # Reading sensor and populating message
        if self.sensor.read():
            msg.temperature_celsius     = self.sensor.temperature()
            msg.temperature_farenheit   = self.sensor.temperature(tsys01.UNITS_Farenheit)
        else:
            self.get_logger().error("Sensor read failed!")
            exit(1)

        # Publishing to /thermometer_data
        self.publisher_.publish(msg)
        self.get_logger().info('\ttime: %s  T: %0.2f C' % (msg.local_time, msg.temperature_celsius))
```

Listing 4.15: Temperature Sensor Publisher.

The code showcased in listing 4.15 showcases the publisher function that is called every *sample_time* seconds. It starts by defining the message type as that of *Thermometer* afterwards it uses the *time* library to get the local time and turn it to string format before putting it in the message. The time for all sensors is collected in HH:MM:SS format. Subsequently, the actual sensing is executed by the *read()* function. The function returns *True* if the sensor was successfully read and *False* otherwise. It is called from an if statement and when the sensor is successfully read both SI and Imperial units are put in the message. In the case that the sensor read should fail, an error message is sent to the log stream and the node is shut down with *exit(1)*. Lastly, the message is published and a log message containing the data collected is printed to the terminal.

4.6.4.2 Removing of MAC Addresses

In the code developed for the previous project, all sensor publishers are implemented to get the Media Access Control (MAC) address of the system and add it to the message containing the appropriate sensor readings. MAC addresses are unique identifiers assigned to a Network Interface Controller (NIC). It is used as an address in communication within a network segment. The only component that has an assigned MAC address in this project excluding the modem is the RPi, this means that the MAC address of the RPi is gathered and published for all sensor readings. The group concluded that this was redundant as if needed the MAC address could be sampled when the communication via Ethernet was needed.

4.7 Power Sensor

The power sensor is more complex than the environment sensors, the sensor returns analog values that need to be digitized before they can be handled by the RPi. This is done by an ADC that communicates with the RPi. The structure of the sensor code is mostly similar to the environmental sensors, and thus only the code exclusive to the power sensor will be explained.

Listing 4.16 shows the import of the Battery message type that is used for publishing. After this, the class *BatteryDataPublisher* is defined inheriting from Node as described in Section 4.6.4.1 *Code Walkthrough.* Some constants that are specific to the ADC and the PSM are defined. In lines 17 and 18 there are also two constants defining what voltage the battery outputs when it is fully charged and empty.

After all constants are defined the class constructor can be defined. Listing 4.17 shows the constructor code without the code that is identical to the sensors already explained in Section 4.6.4.1 *Code Walkthrough.* The ADC is initialized and two constants are calculated based on the hardware-specific constants.

Listing 4.18 shows the *battery_ read_ and_ publish* function that is run every *sample_ time* seconds. This function contains the collecting of data from the ADC, what is not shown in the listing is that a message is defined to be of the type *Battery*, and the local time is collected. The ADC is read, and the measured values are

```
from sensor_interfaces.msg import Battery
...
class BatteryDataPublisher(Node):
    # HARDWARE CONSTANTS
    A0 = 0                          # Channel 0 on ADC (Voltage)
    A1 = 1                          # Channel 1 on ADC (Current)
    GAIN = 1                        # 4.096V reference point
    REFERENCE = 4.096               # Volt
    MAX_VALUE = 2**15               # 16 Bits (signed)
    VOLTAGE_OFFSET = 0.33           # Volt
    CURRENT_SENSE = 37.8788         # Ampere / Volt
    VOLTAGE_SENSE = 11              # Volt / Volt
    MIN_BATTERY_VOLATAGE = 19.8 # Volt
    MAX_BATTERY_VOLTAGE = 25.5  # Volt
    CONSTANT_OFFSET = 0.6           # Volt
```

Listing 4.16: Power Sensor Member Variables.

```
    def __init__(self):
...
        self.sensor = ads1x15.ADS1115()

        # Calculate the voltage and current constants
        self.voltage_constant = (self.REFERENCE/self.MAX_VALUE) *
        ↪   self.VOLTAGE_SENSE
        self.current_constant = (self.REFERENCE/self.MAX_VALUE -
        ↪   self.VOLTAGE_OFFSET) * self.CURRENT_SENSE
```

Listing 4.17: Power Sensor Constructor.

multiplied with the constants defined in the constructor before they are put in the message and published and written to the terminal with *get_logger()*. A constant *CONSTANT_OFFSET* is added to the voltage reading. This is done to read the correct voltage as the power sensor tends to read a voltage with a constant offset compared to the actual value. This was found using a digital PSU displaying the output voltage entering the PSM.

To calculate the battery capacity in percentage a linear approximation was found between the minimum and maximum battery voltage. This was calculated using the member variables to ensure high flexibility in the code if a different battery with different characteristics were to be used.

```
    def battery_read_and_publish(self):
...
        # Reads voltage and current from ADC
        voltage_value = self.sensor.read_adc(self.A0,
        ↪   gain=self.GAIN)
        current_value = self.sensor.read_adc(self.A1,
        ↪   gain=self.GAIN)

        # Calculates all values
        V = voltage_value * self.voltage_constant +
        ↪   self.CONSTANT_OFFSET
        I = current_value * self.current_constant # Does not work
        percent = 100 / (self.MAX_BATTERY_VOLTAGE -
        ↪   self.MIN_BATTERY_VOLATAGE) * V - 100 /
        ↪   (self.MAX_BATTERY_VOLTAGE - self.MIN_BATTERY_VOLATAGE) *
        ↪   self.MIN_BATTERY_VOLATAGE
```

Listing 4.18: Power Sensor Publisher.

4.8 Modem Nodes

To facilitate and make wireless underwater communication possible. The data have to be packaged and sent from the RPi to the modem. In this project this action is performed by two nodes in the *modem_ communication* package.

4.8.1 Data Handler

The data handler node is responsible for subscribing to all sensor topics. It registers when all sensors have published a new value to their topic. It then re-formats the collected topic data to a string that can be sent to the modem. The logic for registering when all sensors have been published is made to be robust and work even if some sensors should fail, albeit running slower than with all sensors working. After the data is reformatted it is published to the *internal_ data* topic for further use by the *internal_ logger* and the *modem_ data_ sender*.

Listing 4.19 shows the imports needed for the *data handler node*. As illustrated by the listing, all message types and the standard Python library *time* are imported. *Node* from *rclpy* is also imported as always in a ROS node.

After all imports the class *ModemDataHamdler* is defined with member variables as shown in listing 4.20. The first variable that is defined is *times_ checked*, this

```python
from rclpy.node import Node
from sensor_interfaces.msg import Barometer, Battery, Modem, Oxygen,
    Salinity, Thermometer
import time
```

Listing 4.19: Data Handler imports.

variable is responsible for remembering how many sensor values have been updated. The next variables are the memory variables for all sensor data. They are defined as Python dictionaries that are updated every time a new message is published on their topic. They include the time the sensor data was collected and the measured values.

```python
class ModemDataHandler(Node):
    times_checked = 0
    barometer_data = {
        'time': '00:00:00',
        'depth': 0.0,
        'pressure': 0.0,
        'pressurePSI': 0.0}
    battery_data = {
        'time': '00:00:00',
        'voltage': 0.0,
        'current': 0.0,
        'percent': 0.0}
    oxygen_data = {
        'time': '00:00:00',
        'oxygen': 0.0}
    salinity_data = {
        'time': '00:00:00',
        'salinity': 0.0}
    temperature_data = {
        'time': '00:00:00',
        'temperature': 0.0,
        'temperatureF': 0.0}
```

Listing 4.20: Data Handler Member Variables.

Listing 4.21 shows the constructor for *ModemDataHandler*. The first thing that is done is to create a publisher and collect parameters. The publisher is set to publish on the topic *internal_ data*, this topic contains the formatted data that is later transmitted by the modem and logged by the internal logger. The two parameters that are collected are *n_ sensors* and *precision*. The variable *n_ sensors* defines how many expected working sensors there are. The variable *precision* governs how many decimals the data transmitted data should contain. The parameter is an integer and defines how many decimal points the value should include.

After the parameters are defined and collected, all subscriptions have to be defined. In this walkthrough, only the temperature subscription will be explained, but the other subscriptions are defined the same way only with arguments/parameters relevant to what topic they are subscribing to. The subscription function *temperature_callback* is created. It will be invoked whenever data is updated on the */sensors/thermometer_data* topic.

```python
    def __init__(self):
        super().__init__('ModemDataHandler')

        self.internal_modem_publisher_ = self.create_publisher(Modem, 'internal_data', 10)
        self.n_sensors  = self.declare_parameter('sensor_count', 5).value
        self.precision  = self.declare_parameter('transfer_precision', 2).value
...
        self.temperature_subscription = self.create_subscription(
            Thermometer,
            '/sensors/thermometer_data',
            self.temperature_callback,
            10)
```

Listing 4.21: Data Handler Constructor.

Listing 4.22 shows the workings of the temperature subscriber. The callback function is executed whenever a new message is received on the subscribed topic. It then saves all data from the topic to the member variables defined in listing 4.20. Finally, the member function *publish_data()* is called.

```python
    def temperature_callback(self, msg:Thermometer):
        self.temperature_data['temperature'] = msg.temperature_celsius
        self.temperature_data['time'] = msg.local_time

        self.publish_data()
```

Listing 4.22: temperature_callback().

Listing 4.23 shows the workings of the function *publish_data()*. This function is responsible for reformatting the data and publishing it. It starts with incrementing and checking how many times the member variables have been updated. If all sensors are working correctly the if statement will be true every *sample_ time* seconds. If there is something wrong with one or more sensors the if statement will be true only when the working sensors are updated *n_ sensors* times. This is implemented to keep the code working even if some sensors fail but it comes at the cost of not transferring data as often as intended.

To keep track of how often data is transferred the local time of the system is collected and put in the *data* variable. *data* is a string that contains all data that is transferred via the modem. It is made to be modular in the sense that the precision of the data that is transferred can be selected at launch.

After the *data* variable is created and filled it is published through the publisher created in the constructor. The variable*data* is also written to the console and *times_ checked* is set to 0 so the process can be repeated.

```
    def publish_data(self):
        self.times_checked += 1
        if self.times_checked >= self.n_sensors:
            current_time = time.localtime()
            local_time =  time.strftime("%H:%M:%S",current_time)

            data = (
        f"{local_time},"
        f"{self.barometer_data['pressure']:.{self.precision}f},"
        f"{self.battery_data['voltage']:.{self.precision}f},"
        f"{self.battery_data['current']:.{self.precision}f},"
        f"{self.oxygen_data['oxygen']:.{self.precision}f},"
        f"{self.salinity_data['salinity']:.{self.precision}f},"
        f"{self.temperature_data['temperature']:.{self.precision}f}"
            )

            modem_msg = Modem()
            modem_msg.internal_data = data

            self.get_logger().info('Data Published to Topic')
            self.internal_modem_publisher_.publish(modem_msg)
            # Reset counter
            self.times_checked = 0
```

Listing 4.23: publish_ data().

4.8.2 Data Communicator

The data communicator is the node responsible for communicating with the modem. It utilizes the *unetpy* library to communicate with the modem. The node works by subscribing to the *inernal_data* topic, when the topic is updated the callback function is called and the data from the topic is sent to the modem. The data is sent through *Unetsocket*, which is a Python module of *unetpy*. After the data is sent, the modem will be set in receive mode for a random amount of time. When the modem is in receive mode it listens for incoming data. If data is received the data and where it came from is decoded and published to the *external_data* topic.

Listing 4.24 displays the unique imports for this node and the class definition of the node. It also shows the variable *start_time* which is set to 0.

```python
from unetpy import UnetSocket
import time
import random

class ModemCommunicator(Node):
    start_time = 0.0
```

Listing 4.24: Data communicator import and class definition.

Listing 4.25 shows the constructor of the *ModemCommunicator* class, the constructor mostly consists of parameter collection for modem-specific variables. It also creates the publisher *external_modem_publisher_* that is set to publish on the *external_data* topic. The parameters that are collected are as follows: *sample_time* and *transfer_delay*. They are used to time when, and for how long the modem should listen for incoming transmissions. *modem_IP* and *modem_PORT* are used in the *UnetSocket* object to declare where to find the modem. Lastly *lower_bound* and *upper_bound* are used to set the modem in receive mode for a random time within the bounds.

After this an instance of *UnetSocket* is created and assigned to the *sock* variable. The two arguments that are given to *UnetSocket* are the IP address to the modem and the port number. For the physical modems, this is always 1100, but will vary when using the UnetStack simulator. Lastly, the bounds that determine how long the modem should be in receive mode are printed to the terminal.

Listing 4.26 shows the *modem_callback()* function. This function is called whenever there is an update to the topic *internal_data*, it starts with setting *start_time* to the system time. Then the *sock.cancel()* function is called, this function is used to cancel

```
    def __init__(self):
        super().__init__('ModemCommunicator')

        self.external_modem_publisher_ =
        ↪  self.create_publisher(Modem, 'external_data', 10)
        self.sample_time =
        ↪  self.declare_parameter('sample_time',2.0).value
        self.transfer_delay =
        ↪  self.declare_parameter('transfer_delay',6.0).value
        self.modem_IP =
        ↪  self.declare_parameter('modem_IP','0.0.0.0').value
        self.modem_PORT =
        ↪  self.declare_parameter('modem_port',1100).value
        self.lower_bound =
        ↪  self.declare_parameter('lower_bound',3000).value
        self.upper_bound =
        ↪  self.declare_parameter('upper_bound',9000).value

        self.sock  = UnetSocket(self.MODEM_IP, self.MODEM_PORT)

        self.get_logger().info('Bounds for this runtime:\
            \nLower Bound: %i \nUpper Bound: %i' % \
            (self.lower_bound, self.upper_bound))
```

Listing 4.25: Data communicator constructor.

any ongoing *sock.receive()*. This is needed because when *receive()* is called it blocks the potential sending of data and can lead to deadlocks. After this, the message from the topic is extracted and sent to the modem. The sending is encapsulated in a try-except block. This tries to send the data with the *sock.send()* function, if successful the code in the else statement will run and the data will be sent acoustically with the modem. If the sending fails, an error will be printed in the terminal and the program will continue as normal.

Next the *modem_ callback()* will set the modem to listen while the time between the start of the function plus the transfer delay is less than the sample time. This is done to have another measure to avoid the *sock.receive()* function from blocking future sending of data. The actual listening happens in the *modem_ listen()* function.

Listing 4.27 show how the *modem_ listen()* function works. First, a timer for the *sock.receive()* function is set. This timer will make the modem listen for a random interval between the parameters defined in the constructor. *random.randrange()* returns a random number within the range given by the upper and lower bounds with

```
38      def modem_callback(self, msg:Modem):
39          self.start_time = time.time()
40          self.sock.cancel()
41          data = msg.internal_data
42
43          try:
44              self.sock.send(data, 0)
45          except:
46              self.get_logger().error('COULD NOT SEND DATA TO MODEM')
47          else:
48              self.get_logger().info('Data Sent to Modem \n%s' % data)
49
50          # Receiving
51          full_time = self.start_time + self.transfer_delay
52          while (time.time() - full_time) < self.sample_time:
53              self.modem_listen()
```

Listing 4.26: modem_callback().

intervals of 300. The modem is then set to receive mode and when a transmission is received the modem will pack it into an object that is then passed in the variable *rx*. In the case that no transmission is received the *rx* will be *None*. When a transmission is received the if statement in line 59 will be true and the unpacking of the received data is started. The data sent is then put in a string with the ID of the sender and published on the *external_ data* topic and lastly printed to the terminal.

```
55      def modem_listen(self):
56          self.sock.setTimeout(random.randrange(self.lower_bound,
         ↪   self.upper_bound, 300))
57          rx = self.sock.receive()
58
59          if rx is not None:
60              data = str(rx.from_) + ',' + bytearray(rx.data).decode()
61
62              msg = Modem()
63              msg.external_data = data
64              self.external_modem_publisher_.publish(msg)
65              self.get_logger().info('Recieved data:\n%s' % data)
```

Listing 4.27: modem_listener().

4.8.3 Connecting to the Modem

Since the modem is only connected to the RPi through an Ethernet switch it is important to make sure that all devices are connected to the Local Area Network (LAN). The easiest way to do this is to set static IP's on all connected devices. When setting static IP it is important to ensure that the subnet mask of the IP matches with the one on the modem. Also, ensure that none of the connected devices have identical addresses.

Browser Interface
The modem also offers a browser interface, this interface can be used to gain more control over the modem. Through the browser interface parameters like the transmission effect and sleep schedules can be modified, The UnetStack handbook offers a full overview of what the browser interface offers [52].

4.9 Logger Nodes

To ease data collection and storage two logger nodes were created. The two nodes that were created are the *internal logger* and the *external logger*. The internal logger logs the data that is sent to the modem (*internal_ data*) while the external logger logs all data that is received from the modem (*external_ data*). Each logger saves all data from the topic to a file that is created upon startup, the files can be found in the *log_ data* folder. They are saved as *.csv* files.

4.9.1 External Logger

The external logger node displayed in listing 4.28 is responsible for logging all data that the modem receives. It works by writing all updates to the subscribed topic to a *csv* file. When the program is started the file name and folder is generated, the folder is generated by the system's local date while the filename includes the time the program was started, also fetched from the system. The file is put inside another folder making the full path from the workspace directory: *log_ data/DD_ MM_ YYYY/external/external_ log_ HH_ MM_ SS*. The generation of timestamps is done by utilizing the *datetime* library.

After the file name and location is defined ,the file is opened to write the column names. This will increase user experience if the data is to be inspected. The imports

```python
class ExternalLoggerNode(Node):
    def __init__(self):
        super().__init__('external_logger')
        self.header  = self.declare_parameter('log_header', '').value

        current_date = datetime.datetime.now().strftime("%d_%m_%Y")
        current_time = datetime.datetime.now().strftime("%H_%M_%S")

        directory = 'log_data/'+ current_date + '/external/'
        self.file = directory + 'external_log_' + current_time + '.csv'

        if not os.path.exists(directory):
            os.makedirs(directory)

        with open(self.file, 'w', newline='') as csv_file:
            writer = csv.writer(csv_file, delimiter=';')
            writer.writerow([self.header])
```

Listing 4.28: External Logger constructor.

of this listing are left out as it is mostly the same as all other nodes, the two exceptions are the standard library modules *csv* and *os*.

Listing 4.29 shows the *external_logger_callback* function, this function is called every time the topic is updated. It writes the data from the message in a new row in the log file. The letter *'a'* in the *open()* function determines that what is done in the function is to be appended.

```python
    def external_logger_callback(self, msg:Modem):
        data = msg.external_data
        with open(self.file, 'a', newline='') as csv_file:
                writer = csv.writer(csv_file, delimiter=';')
                writer.writerow([data])
```

Listing 4.29: external_logger_callback.

4.9.2 Internal Logger

The internal logger is almost identical to the external logger, the only thing differentiating it other than the node name, topic, file name, and location is that the internal logger does not include *Modem_ID*. This column is meant to differentiate what agent the data comes from and since the internal logger only logs data from itself it is not needed.

4.10 Launch File

To increase user-friendliness the launch file was updated and expanded. The launch file launches all necessary nodes for the rig to work properly. It was expanded to include parameters that easily can be modified by a user without having to dive into nodes in order to change parameters. Upon launch all parameters are given to the nodes depending on them, after the parameters are passed, the node is started. For a full overview of what nodes take what parameters see Table 4.1.

Node	Parameters
All sensors	Sample time
Loggers	Log header
Modem data handler	Number of working sensors Value precision
Modem data communicator	Sample time Transfer delay Modem IP Modem port Lower bound Upper Bound

Table 4.1: Nodes and parameters.

The only logic in the launch file is the generating of the bounds used in the modem communicator node. Listing 4.30 highlights how the bounds that are given as parameters to the modem data communicator are created. They are randomly generated for every launch between the bounds declared in lines 19 and 20. This is done to minimize the risk of having two rigs that run the same program running into deadlocks.

```python
ms = 1000
step = 200
lower_bounds = [2 * ms, 4 * ms]
upper_bounds = [4 * ms, 6 * ms]

random_bounds = [
    random.randrange(lower_bounds[0], upper_bounds[0], step),
    random.randrange(lower_bounds[1], upper_bounds[1], step)
]
```

Listing 4.30: Bounds generation in launch.py.

Chapter 5

Integration and Optimization of Mechatronic Aspects of the Sensing Rig

This chapter delves into the integration of mechanical solutions for designing, manufacturing, and testing structural and electrical rig parts. In this chapter, a comprehensive explanation will be provided of the hardware modifications, including the creation and decisions behind specific adapters and cables, elucidating their construction and purpose.

Furthermore, a thorough exploration of all sensors requiring calibration, accompanied by detailed explanations. Additionally, comprehensive testing methods for assessing sensor functionality will be presented. The chapter will dive into waterproofing factors, such as vacuum testing, to ensure the system's reliability. Finally, a comprehensive bill of materials for the rig will be provided, outlining all necessary components.

5.1 Electrical Design

The electrical design of the Standalone Underwater Monitoring Station (SUMS) plays a crucial role in ensuring reliable data acquisition and communication. This design includes the integration of various components such as sensors, analog-digital converters (ADCs), power supply units (PSUs), and cabling. The project has made some modifications to the electrical design since the previous project group in 2022.

Several hardware modifications have been carried out on the sensing Internal Electronic System (IES), accompanied by the development of custom adapters required for interconnecting various components. Additionally, specialized underwater cables were researched and produced to facilitate the connection of the acoustic modem. Two distinct methods were employed, each of which will be comprehensively described.

Furthermore, this section provides a comprehensive overview of the system integration and highlights the specific alterations implemented during the SUMS project phase. Video demonstrations showcasing the complete system setup have been produced [30].

5.1.1 Hardware Modifications

5.1.1.1 Bypassing the Isolation on the new PSU

The new PSU, Traco THN 15-1211WIR dc-dc converter, was modified in the making of the IES. The reason for the change was due to the fact that the PSU has an isolated output, resulting in a floating output voltage. However, ground is also floating, which is undesirable for the system. A common ground is necessary for the entire rig, and this is achieved by shorting the PSU's input ground point with the output ground point. This results in a common ground at the battery's negative terminal and the RPi's ground.

5.1.1.2 Modifications to the PSM

The PSM reads the battery status and sends the measured values as analog signals to the ADC on the IES. The ADC's analog input requires voltages between 2 and 5.5 Volts. However, the PSM outputs a voltage that is approximately equal to the battery voltage. The reason for this is unknown, but the solution is known. Out of the box from BlueRobotics, the PSM's ground is not connected to the Integrated Circuit (IC). The solution is to modify the PSM by connecting the ground to the IC. **(Forgetting to modify the PSM will result in a broken ADC)**

BlueRobotics' technical description of the output voltage on pins 3 and 4 (current and voltage sensor) is 3.3 Volts. This is only true if the ground is soldered to the PSM. To do this, first remove the plastic covering the PSM's IC. Then, cut the ground cable in two and strip the wire. Finally, solder the wire onto the back of the

IC, as shown in Figure 5.1. Remember to use either electrical tape or heat-shrink tubing to cover the PSM's electronics. Do not cover the JST-GH connector.

The soldering should be done with plenty of solder and a soldering iron that can deliver over 100W. Using a tiny soldering tip and low power can result in the cable not getting hot enough, which can cause the solder not to enter the strands of the cable which will result in poor solder.

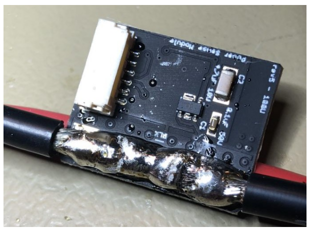


Figure 5.1: Ground soldered to the PSM.

5.1.1.3 Micro Switch RJ45 Female Adapter

In order to establish a connection to the IES with a computer through the small switch, an adapter cable is required. This adapter cable features an 8-pin PicoBlade connector on one end, which is plugged into one of the switch's connection ports, and an RJ45 female connector on the other end, which can be connected to a computer or a network using a standard Ethernet cable.

Wire slot	Wire color
1	Orange
2	Red
3	Yellow
4	Brown
5	Black
6	Green
7	Purple
8	Blue

Table 5.1: Wire table for Actassi S1 and PicoBlade adapter.

The cables that come with the switch are pre-fitted with PicoBlade to RJ-45 male adapters. These adapters need to be removed and replaced with Actassi S1 connectors, which are RJ-45 Cat6 UTP-compatible plugs manufactured by Schneider. The following procedure describes how to make the necessary modifications:

1. Open the connector by pressing the white button on the opposite side of the hinges.
2. Thread all cables through the cable hole.
3. Place the correct color wire into the corresponding wire slot, as indicated in Table 5.1.
4. Cut off any excess wire protruding beyond the wire slot using wire cutters.
5. Close the connector.
6. Test the cable using a multimeter, verifying that each wire is properly connected and has good conductivity.

To simplify the testing procedure, it is recommended to strip a small length of wire from the RJ-45 male adapter that was previously removed, and insert it into the connector to be tested. This allows for a color-by-color testing of the wires, reducing the likelihood of errors during the testing process.

5.1.1.4 Micro Switch Molex SL Adapter

The acoustic modem's communication wires have been modified to have Molex SL connectors to fit through the cylinder cap holes. To be able to establish a connection to the RPi with the modem, it must be connected to the switch. This section will describe the making of the adapter needed for this purpose.

To create a PicoBlade to Molex SL adapter, an adapter is needed from the Gigablox switch. This adapter will be modified to become the described adapter. Here is a description of the modifications that are made:

1. Start by cutting off the RJ45 plug as close to the plug as possible.
2. Remove the following cables:
   - Black
   - Brown
   - Purple
   - Blue
3. Crimp a Molex SL female connector onto each wire.
4. Ensure that all the cables are properly twisted (the twisting should be in the same direction as they were originally twisted).
5. Place a dual plug over the crimps.
6. Optional: Apply heat shrink tubing.

It is important to note that Molex SL connectors can be plugged together in either orientation. Therefore, it is crucial to verify the correct alignment of the connectors and match the appropriate cables accordingly. Refer to Table 5.3 for guidance on matching conductors.

5.1.2 Waterproofing Modem Cables

In order to connect the acoustic modem to the sensor cylinder, a cable is required that can both supply power and enable communication. However, the cables that come with the modem are excessively long and bulky. To address this issue, a shorter cable needs to be constructed with a SubConn male plug on one end and a WetLink penetrator on the other end to connect to the cylinder. The modem then needs to be connected to the power supply terminals in the cylinder and to the switch to enable communication with the Raspberry Pi.

Two methods were employed to create cables for the modem throughout the project. The first method is not intended for use in future projects as it is overly complex and expensive. This section will elaborate on the development of both techniques and outline the requirements to reproduce the two cables that were produced during the project.

5.1.2.1 Pigtail Splicing Method

In order to create a waterproof cable with an MCIL8M pigtail from MacArtney, several steps need to be taken to ensure a high-quality product suitable for use. The first step is to strip the end to a suitable length. It is recommended to leave some extra cable length for a loop between the modem and the cylinder. It is crucial to take the time and be meticulous when stripping the cable, as damaging the pigtail beyond the conductor's insulation will render it unusable.

To strip the pigtail, it is recommended to use a knife blade from a utility knife and attach it to a table vise. Refer to Figure 5.2 for the desired setup, ensuring that the thickness of the outer insulation and the amount of blade protruding from the vise is matched, perhaps even slightly less than anticipated to reduce the risk of damaging the pigtail. The cable should then be cut around the insulation, being mindful to cut perpendicular to the cable. This will result in a straight cut around the entire cable, rather than an angled cut, which can lead to poor sealing when attaching the WetLink penetrator. **Keep in mind that when the blade in mounted to the table vise it is a hazardous zone. Take precautions and analyze the risks.** After the cable has been stripped, trim away conductors 7 and 8, which correspond to the white/black and red/black wires, respectively. These will not be used.

The subsequent step involves fitting the red locking mechanism (MCDLS-F), followed by attaching the WetLink 9.5mm high compression penetrator to the cable.

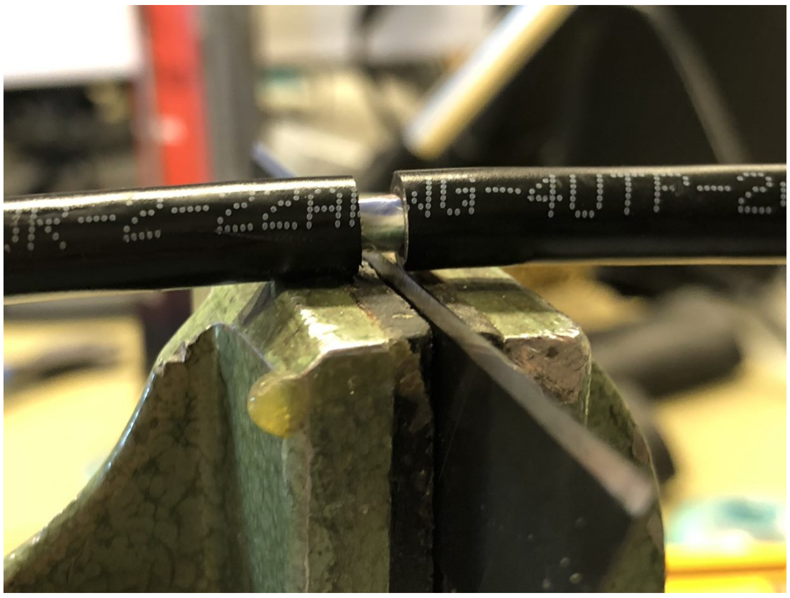
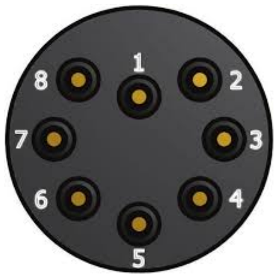


Figure 5.2: Cable stripping with a blade mounted to a table vise.

This is done by sliding the plug onto the cable, then followed by the seal, and finally the bulkhead. These components are then fastened together as tightly as possible. Although specific tools are available for this task, all that is required are two appropriately sized wrenches to securely fasten the plug and bulkhead. A more comprehensive guide on how to perform this assembly can be found on the Blue Robotics' website [7]. Since the penetrators bulkhead body diameter is M14, and the cylinder cap holes are M10, it is necessary to expand one of the holes on the cap. This was done by the mechanical workshop at ITK. It is important to apply grease to the O-ring that accompanies the bulkhead before mounting it on the cylinder.



MCIL8M		PicoBlade connector	Barrier strip
1	Black	Orange	
2	White	Red	
3	Red	Yellow	
4	Green		Negative (÷)
5	Orange		Positive (+)
6	Blue	Green	

Figure 5.3 & Table 5.2: Overview of matching conductors.

In order to complete the cable assembly, Molex SL male connectors must be attached to each wire that will be connected to the switch. Dual-plugs are used for each pair. The first step is to attach the crimp to the wires, which requires a special tool to crimp the contacts in place over the wire. The innermost clamp on the connector should clamp onto the conductor material, while the outermost clamp should clamp onto the insulation. The purpose of this is to hold the connector securely so that it cannot be pulled off the cable. When crimping the connectors with the tool, use the 1 mm notch. One should go two notches higher when crimping the insulation than when crimping conductor material.

After all the pins have been attached, a small heat shrink tube can be applied, and the pairs should be twisted together. Then the dual plug can be attached by pushing both pins into the plastic plug until they come to a stop. It is important to note the orientation of the plug in relation to the wire colors. The same procedure is applied to the PicoBlade connector using female connectors. See Table 5.3 to match the correct colors between the modem cable and the PicoBlade connector. Lastly, crimp some cable sleeves on the power conductors (pin 4 and 5).

The cable should then be tested. First, use a multimeter to ensure continuity in all wires. Then to verify the waterproofness of the cable, it must be installed to a cylinder cap. The cylinder must then be vacuum tested using the method described in Section 5.6.1 *Vacuum Testing Procedure.*

5.1.2.2 Plug Splicing Method

This method will not be used in the future. See Section 5.1.2.1 *Pigtail Splicing Method* for the new cable.

Prior to the cable assembly process, all necessary components must be obtained in order to produce a finished product. A cable with two power conductors, including two twisted pairs, is required. The cable used in this project had two conductors intended for power supply, which were slightly thicker than the conductors of the twisted pairs. Two of these pairs were trimmed away because the cable consisted of 4 pairs even though only 2 pairs were needed. Furthermore, a SubConn MCOM8M plug, a locking mechanism (MCDLS-F), and a rubber cap (OMBMC) are necessary. The rubber cap can and should be replaced with a shrinkable tube with adhesive, which will be explained in greater detail later. Finally, for the cylinder end of the cable, a WetLink penetrator that fits the cable being used is required. This depends on the cross-section [6]. For this project, the WetLink 7.5mm low-compression was the correct penetrator for the chosen cable. A means of connecting the power conductors to the barrier strip in the IES is necessary. Sleeves were used in this task, but cable shoes may also be used. Additionally, communication with the switch must be prepared for. The switch uses a PicoBlade connector that, unfortunately, is too large to pass through the cover of the cylinder. It is possible, but requires a lot of bending of the cable, which can damage it over time. Furthermore, connecting and disconnecting the PicoBlade connector from the switch is discouraged as it can also damage equipment, and in the worst-case scenario, the switch. As a result, Molex SL connectors are used, which are easy to terminate and connect and disconnect.

Cutting and Insulation the Cable

First, cut the cable to a suitable length. It is recommended to have a length long enough to reach from the cylinder to the modem, plus some extra slack. Additionally, consider some extra cable inside the cylinder (approximately 15 cm). After cutting the cable, strip the insulation from each cable end. A detailed description of how to strip these cables is provided in Section 5.1.2.1 *Pigtail Splicing Method*. The end that will be spliced with the SubConn plug should be stripped to slightly over half the length of the rubber cap, while the other end should be stripped approximately 15 cm.

Solder the Cable to the Plug

Begin by soldering the cable to the SubConn plug. Ensure that each conductor is soldered to an equal length so that the conductors are approximately parallel when all of them are soldered together. Use heat shrink tubing on each conductor to prevent short circuits. See Figure 5.4 for reference. The heat shrink tubing should not be too tight or too loose.

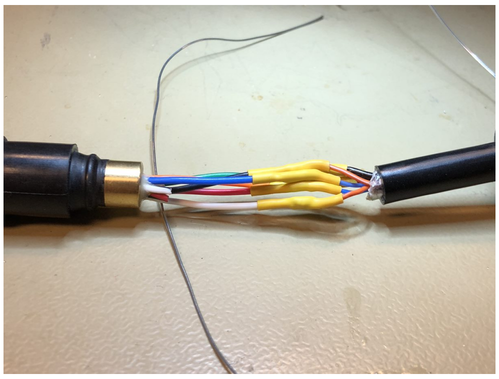


Figure 5.4: MCOM8M spliced to the cable.

If the heat shrink tubing is too loose, it may not shrink enough during heating. As a consequence, a small void may be created where the epoxy mixture, which will be cast around the conductors later, cannot reach. This could introduce small air pockets that will compress under high pressure and potentially draw in water in a worst-case scenario.

If the heat shrink tubing is too tight, it will cause problems during the soldering process. When soldering the conductors, one must slide the heat shrink tubing onto the cable before soldering the conductors together. If the conductors are not soldered together within a short timeframe, the tubing will shrink slightly because of the heat from the solder. As a result, the tubing will not be able to slide over the joint and will become unusable.

When soldering the cables together, it is possible to freely choose which conductors to use. Each cable may have a different color coding, making it impossible to provide a specific guide on which color should be soldered together with the conductors on the SubConn plug. It is not important to note which color is soldered because it can easily be found using a multimeter after the soldering process. This step is necessary regardless as continuity in each conductor needs to be tested. The only important consideration is to solder the green and orange conductors (pin 4 and 5) on the MCOM8M together with the power conductors of the cable being used, and to solder the black and white conductors (pin 1 and 2) together with a twisted pair on the cable, and to solder the red and blue conductors (pin 3 and 6) with another twisted pair.

Preparing for Epoxy Cast

Furthermore, a decision must be made regarding whether to use the rubber cap that belongs to the MCOM8M plug solution or to use heat shrink tubing with adhesive. This issue is described and discussed in detail in Section 7.2.2 *Comparing Heat Shrink Against Rubber Cap with Plug Splicing Method.*

Once all the conductors are soldered together and heat shrink tubing is shrunk over each conductor, use some clean alcohol to remove any grease. This step is important to ensure proper adhesion of the epoxy mixture. This must be done to the end of the cable jacket as well, but before that, use some sandpaper to create a texture on the cable jacket. This will help the epoxy stick to the cable jacket. This is only necessary if the rubber cap is used.

Proceed by sequentially attaching the rubber cap, or alternatively, the adhesive-lined heat shrink tubing measuring 12 cm in length. Finally, slide the locking mechanism (MCDLS-F) in place. If heat shrink tubing is used, shrink 1 cm of the end facing the MCOM8M plug using a heating gun. This prevents epoxy from leaking out when casting. The cables are now ready to be mounted in a vertical position with the plug facing downwards.

[21]

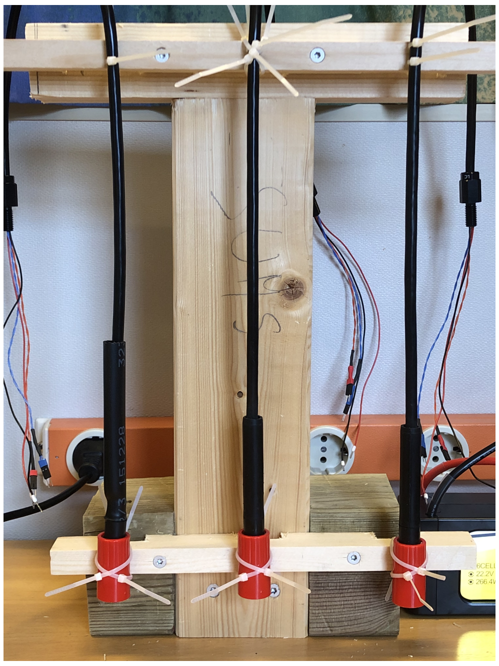


Figure 5.5: Modem cable plug casting setup.

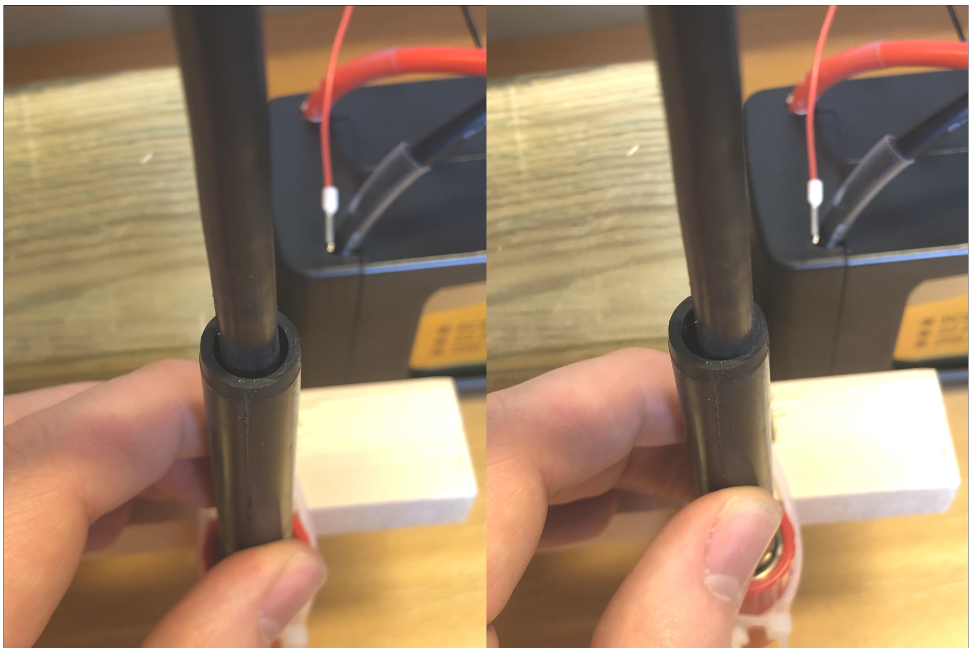


Figure 5.6: Rubber cap positioning.

One of the risks associated with casting using MCOM8M is the limited overlapping contact area between the cable jacket and the provided rubber cap that comes with the plug. Additionally, there is a risk of the jacket and rubber cap being too close to each other during casting due to the nearly identical diameter of the cable jacket and the inner diameter of the rubber cap. Therefore, precise casting is essential. The rubber cap and cable must be aligned parallel to each other, with the cable centered within the rubber cap. Refer to the left picture in Figure 5.6. The right picture is how it shouldn't be after casting.

There are several measures that should be taken to ensure optimal casting. Here is a brief list of measures:

- Ensure maximum overlap between the cable jacket and the rubber cap. In other words, the solder joint should be as short as possible.
- The cable should lie parallel in the center of the rubber cap after casting.
- Securely fasten the cable to prevent bending and movement during curing.
- Roughen the portion of the cable jacket that will be in contact with the epoxy mixture using sandpaper to create some texture.
- Thoroughly clean the cable jacket, plug, and conductors with alcohol or another solution to remove grease.

The setup of the casting process is illustrated in Figure 5.5. It was made by the group to secure a controlled and stable curing process.

Casting Epoxy Mixture

Once the cable is securely tensioned with the plug facing downward and the rubber cap ready for filling, the preparation of the epoxy mixture can commence. Scotch-Cast 2131 epoxy was employed for the casting. It is important to note that the epoxy has a limited working time before it starts to solidify, so it is imperative to have all the necessary preparations completed beforehand. Although the package indicates a solidification time of 13 minutes at room temperature, it has been observed through experience that the epoxy thickens significantly after approximately 5 minutes of exposure, impeding its flow into the rubber cap.

To proceed with the casting, the epoxy must first be mixed with the hardener. This is accomplished by firmly squeezing the bag containing the epoxy and hardener together for approximately 30 seconds. Once the contents are thoroughly mixed, a corner of the bag can be cut to facilitate easy dispensing of the epoxy into the rubber cap. It is crucial to completely fill the cap while ensuring that no contact occurs between the cap and the cable.

During the execution of this procedure, it was discovered that there is insufficient time to fill three caps within the 5-minute timeframe; only two caps can be effectively filled. Beyond this point, the epoxy mixture becomes excessively thick, making it unmanageable. After completing the filling process, it is necessary to allow the epoxy to cure fully before conducting any further testing on the cable. The epoxy takes approximately 36 hours to fully cure.

If a shrinking tube was used to cast, the end must be shrunk to complete the SubConn end of the cable. This is done by using a heating gun. It is very important to be careful when shrinking. Not all cables can withstand the heat needed to effectively heat up the shrinking tube. Be patient and start by using the lowest heat setting on the heating gun. Start by heating the hollow part near the epoxy and work it towards the cable as it shrinks. Excessive heat on the cable can introduce deformation which in the worst case scenario will break the cable jacket making it unusable. See Figure 6.8 for reference of a deformed cable jacket.

Installing WetLink Penetrator and Crimp Conductors

The final step to complete the modem cable is to install the WetLink penetrator, crimp on Molex SL male connectors, and put some cable sleeves on the power conductors. The 7.5mm low-compression WetLink penetrators were used with the provided cable for this project. The process of installing WetLink penetrators and Molex SL connectors is explained in Section 5.1.2.1 *Pigtail Splicing Method.*

The cable is now finished and ready for a final test using a multimeter to confirm that all conductors have conductivity and are isolated from each other. Also, a vacuum test must be done with the cable installed to the rig to ensure it is indeed waterproof. Performing a vacuum test is explained in Section 5.6 *Vacuum Testing.*

5.1.3 Wiring

There have been some minor changes to the cabling in SUMS since the previous project group in 2022, as described in Section 3.8 *Wiring* in the 2022 thesis [25]. This project has been focusing on integrating the acoustic modem, which is the main difference since 2022. Other changes include the Gigablox switch being powered through a barrier strip, and the RPi receiving power from a new PSU via USB-C (At present, this applies to a single PSU, but it is intended to become the standard configuration). As part of these changes, a new cabling diagram for SUMS has been created. Refer to Figure 6.16.

The procedure for connecting everything on the rig is explained in a video demonstrating the assembly of the IES [30]. It is important to note that when connecting the cables to the I2C splitter, **extreme caution must be exercised**, especially with the DF13 connectors. These connectors are fragile and can break if the cable is pulled straight out of its contact point. Use a small terminal screwdriver to gently pry the plug out, as demonstrated in the video. The same applies to the PicoBlade connectors of the Gigablox switch.

5.2 Designing Mounting Parts

Solutions for securing multiple components of equipment to the rig were developed throughout the course of the project. This involved attaching the acoustic modem and sensors produced by Atlas Scientific. The solution had to be secure and hold the equipment in place without compromising its integrity, and it had to be easy to remove when necessary.

A strap was considered for attaching the modem, along with some 3D-modeled brackets designed using Fusion 360. After the design, all necessary parts were ordered and/or produced before being assembled onto the rig.

As part of the work on the rig, a solution for mounting the oxygen and salinity sensors had to be designed and assembled. These sensors were loosely hanging from the cable that exits the sensor cylinder. The cable and sensor had to be securely attached to the rig in a way that was easy to remove and install. Two clips were designed and 3D-printed for the sensors to clip into, eliminating the need for tools to attach the sensors. The cable between the cylinder and sensor was secured to the rig using zip ties to prevent it from getting tangled underwater.

5.2.1 Preparation Work

The first task in designing was the bracket for the modem. This was of the highest priority. However, in order to experiment with various solutions it was desired to have a virtual model in CAD of the rig before designing parts. All components from Blue Robotics were directly obtained from their website. Atlas Scientific provided a CAD file for their oxygen sensor, which was imported into Fusion 360. Finally, a request was sent to Subnero for a model of the M25MRS3 modem. Later, it was discovered that the received 3D model did not completely correspond to the physical model. This will be explained later. The final models required to assemble a complete virtual model of the rig were the self-produced parts. These included the left and right ROV mounting brackets, which were available as .stl files, and the aluminum mounting plate, which was created from physical measurements to then be re-designed in Fusion 360. It is worth mentioning that all .stl files that are to be imported into a CAD software must be converted from a mesh to a solid, as the .stl format only represents a shell composed of many small triangles. It is not possible to work with this in CAD, so later on, it will be explained how to convert a mesh to a solid in Fusion 360.

5.2.1.1 Setting up a Workspace to Work With Fusion 360

The first thing one should do before starting working on a project is to make a structured and secure workspace where all files needed for the project can be found. This workspace should be backed up somehow, either it is the cloud or a scheduled external-drive backup. The workspace for modeling in this project was divided into 3 different folders on Microsoft Teams. Each folder is dedicated to a single file format. Since many models will have several versions, it doesn't take long before one folder containing all different models, and its versions in different file formats, become a great mess. To avoid this, it was made a folder called *Models*. This folder is containing:

- 3D-models(stl-files)
- 3D-prints(gcode-files)
- CAD(STEP-files)

It is easy to understand what is inside these folders, and it makes it easier to find the right files in a later stage of the project.

In Fusion 360 it is possible to create a project it is possible to invite coworkers. All the files are backed up on Autodesk's cloud system automatically. Follow **this** guide on how to create a project in Fusion 360 [17].

5.2.1.2 Using Joint Functionality in Fusion 360

The joint functionality in Fusion 360 is a powerful tool and can be used in many different applications. All parts designed in this project are rigid parts with no degrees of freedom, therefore rigid joint is the only setting that is necessary to use for this application. This is selected in the second tab of the tool window.

To join two components with the *Joint*-tool one simply press *j* on the keyboard and left-click the two points on the components that are supposed to join. The second object pressed is the object that will stay stationary after the operation. The first part will move to the second component. It is now acting as one rigid part. The most intuitive way to use it is by joining two screw holes together. Rotating after the joining has taken place might be necessary. In this project, this tool was used frequently to assemble all the different parts making a virtual rig in Fusion 360.

5.2.1.3 Converting Meshes to a Solid in Fusion 360

The right and left ROV mounting brackets were obtained from prior work as .stl files. This file format is dimensionless and not solid when imported into CAD programs. To make it a solid component, the *Convert mesh* functionality in Fusion 360 can be used. Prior to use, mesh groups of the model must be created. If conversion problems arise, the mesh *repair*-tool must be utilized to close any holes in the model. These holes may arise from mesh grouping and cause deformation of the model during conversion. Then use the *Convert mesh*-tool on the mesh to make it a solid. The *prismatic* conversion method is selected to avoid the creation of numerous triangles (which is what .stl files or meshes are composed of). Once the conversion is complete, unnecessary lines crossing surfaces can be deleted from the solid component.

5.2.1.4 Modifications to the Modem Model

During the bracket design process, it was discovered that there was an inconsistency between the Subnero model of the modem and the physical modem used in the project. The model's original and modified variants are shown in Figure 5.7.

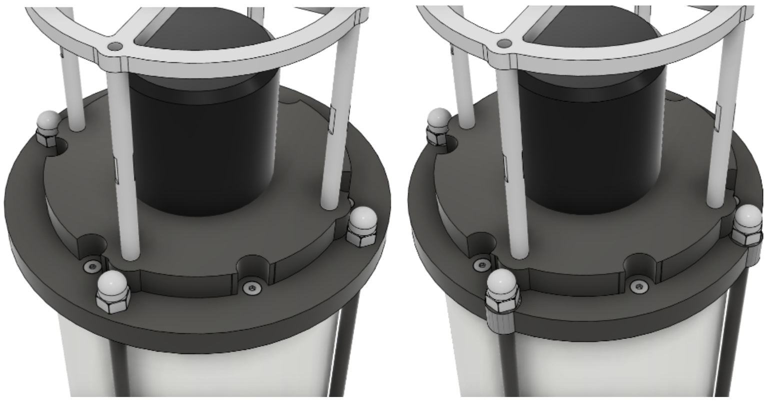


Figure 5.7: Modifications to acoustic modem model.

The left model is the original and right is the modified one. The issue with the original was the plates holding the cylinder together with threaded rods. The outer diameter of the plate was circular with a greater diameter than the physical model. As one can see on Figure 5.7, the right model's plate have dents around the part where the threaded rod enter the plate. There was the same issue on the plate in the other end of the modem, though this is not shown in the figure.

The modification was done by creating a sketch on the plane of the plate (remember to find the plate within the models objects and select it before editing). Then a smaller circle was created within the plate's circle corresponding the correct diameter measured of the physical modem. Then one would have a circle within the outer edge of the plate. It is then desired to remove the overlap from the inner circle of the plate. This can be done by extruding the area between the plate's edge and the circle that just were made. Using the extrude tool in Fusion 360, select *to object* and select the other side of the plate. Then select *cut* at the bottom of the tool window. The diameter of the plate is now correct, but it is necessary to go back to the sketch and add four additional circles where the threaded rods enters the plate. The diameter of these *semicircles* are difficult to measure precisely, thus they were done by eye. The circles are placed in the center of where the holes to the rods are with a diameter found to be about 12 mm. Do this to all four holes on the plate and start extruding the circles in the same way done when removing the overlapping circle as mentioned before. Only this time instead of *cut*, use *join*. This was done to both sides of the modem. The model was now the same as the physical modem.

Assemble a Digital Clone

The preparation work done was gathering all 3D models of existing components that were a part of the rig. The next step was to import these parts into Fusion 360 one by one into the project folder. Then all imported models were converted from a .stl file into a solid model. The project folder is then full of parts used in the project. Now it is time to assemble the parts.

First, make a blank Fusion 360 file to begin the assembly of all the single parts. In there, one can simply drag and drop all parts from the left side data panel into the workspace. It is a good practice to place the first component in the origin in a way it would make sense. The way it was done in this project was by importing the aluminum mounting plate first and aligning a corner of the plate with the origin. Either use the *Move*-tool in Fusion 360 to place the object in the origin, or use the *Align*-tool. It is recommended to use the *Align*-tool because it simply is the easiest and fastest way to move an object or component to a point in space in Fusion 360. Simply click the point to align on the aluminum mounting plate, in this case, a corner, and then click the origin. Now it's time to import all the remaining parts one by one. For each imported part it is smart to check if *Capture Design History* is enabled by right-clicking the component. Then use the *Joint* functionality to assemble all the parts in a rigid structure as described in 5.2.1.2 *Using joint functionality in Fusion 360.*

5.2.2 Modem Mounting Brackets

It was intended for the modem to be fastened on the center of the aluminum mounting plate, mounted in parallel to the cylinders, and secured with straps around the plate. In order to ensure that the modem remained stable and securely fastened, a solution was proposed whereby two brackets would be created at each end of the rig. These brackets would serve as docking points for the modem before being secured with straps around the aluminum mounting plate. This idea was devised as a means of preventing the modem from sliding around and becoming dislodged during use. By incorporating the brackets as part of the overall rig design, the modem could be more effectively anchored in place and made less susceptible to any external forces that might cause it to shift or move. These brackets are to be screwed onto the aluminum mounting plate and be a part of the rig itself. To simplify the process, the screw holes were placed to overlap with the threads of the clamps, thus avoiding the need to create additional holes in the aluminum mounting plate and to use nuts to attach the brackets.

The function of the brackets is to keep the modem stationary when it is strapped in place. Additionally, there will be extra security on the front bracket so that if the strap were to fail, the modem would still remain attached to the bracket.

5.2.2.1 Designing the Front Bracket

An idea for designing the brackets was to create a square block in CAD and then import the modem's model into the same file. The modem could then be placed inside this block, and its model subtracted from the block, leaving a mold of the modem. This mold had to be further refined and adjusted to ensure that the modem could be placed in the bracket. See figure 5.8 below to see snapshots of the process used to achieve the desired result.

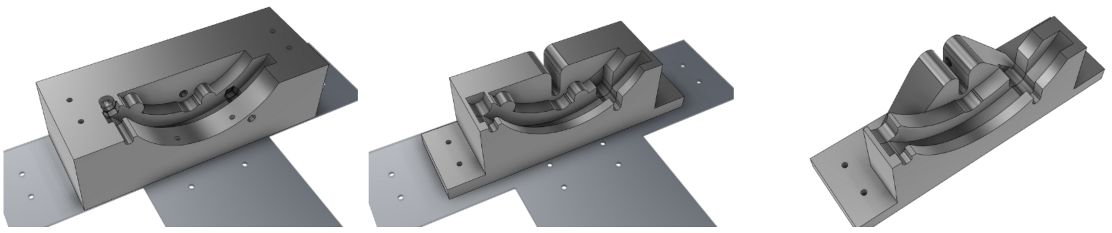


Figure 5.8: Front bracket modem mold process.

To do this, the *combine*-tool was used. Before using the tool, make a block and place the part of the modem that is desired to be in contact with the bracket inside each other. To use the *combine*-tool, first select the block, then select the modem after. To ensure all object of the modem is selected, isolate the modem using the left overview object list, then toggle only the modem's component to be visible. After that select the whole modem by drag-selecting a big square covering the whole modem. When this is done, select the option to cut in the tool menu and then execute the operation. The *combine*-tool will now remove all parts where the modem intersects with the block. The result should look something like the left snapshot in Figure 5.8. It might be necessary to delete or toggle away some objects that are unwanted in the bracket. The remaining job is to make closed parts of the bracket accessible to be docked by the modem. For instance, the struts of the modem need access to its location on the bracket.

The front bracket is intended to ensure the orientation of the acoustic modem. It is not desirable for the modem to rotate, shift, or change position in any way when it is strapped to the rig. To prevent any movement, the bracket was designed with the ability to lock the modem in place with a zip tie or metal wire. See Figure 6.2.

5.2.2.2 Designing the Rear Bracket

The same method used to create the bracket in front was employed to manufacture the first draft of the bracket at the back of the rig. However, the major difference is that the latter bracket has no means of attaching itself to the modem. It was supposed to be placed at the rear end of the aluminum mounting plate to facilitate the use of the clamp's threads which are placed underneath the plate. See Figure 5.9a.

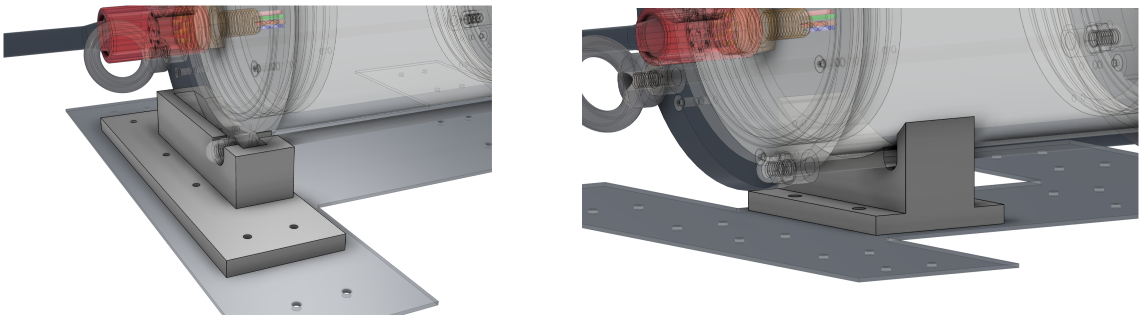

(a) First draft of the rear bracket. **(b)** Second draft of the rear bracket.

Figure 5.9: Rear brackets for the Subnero modem.

After printing the first draft, an issue was discovered with the method used to create the brackets. Although the modem model in CAD is perfect, the real-world implementation is not. The front and back caps are not directly above each other, and therefore, the modem's struts are not perpendicular to the caps. Consequently, when the brackets are docked in place in front and at the back, the base of the brackets is not level with each other. There is a slight rotation about the modem's axis. To correct this, either the front or back bracket must be adjusted in CAD to bring them into alignment. The rear bracket requires less work as it has fewer details to consider.

The second draft was formed using the same method once again. A block was placed on the aluminum mounting plate. This time it was placed on the middle cylinder part of the modem. The modem was subtracted from the block and some small adjustments were made. The section where the threaded rod is located was expanded to create some room for maneuver as the modem's struts were not perpendicular to the caps. Furthermore, screw holes were added to allow the bracket to be secured to the aluminum mounting plate. The holes do not line up with the clamp's threads, which means it will need some nuts to be secured properly.

5.2.3 Designing the Sensor Holder

When designing the sensor holder simplicity was key. There should be no need for tools to mount the sensor. The solution was to utilize the rod-like design of the sensors to make a clip for the sensor to fit into. The diameter of the sensors are 12.0mm and the sensors holder is 12.2mm. The sensor holder was made with the same diameter as the sensor. However, this made it hard to clip the sensor in place. Because PLA is a hard compound, some sensor holders broke when the sensor was clipped in place.

To mount the sensor holder to the aluminum mounting plate a hole with the dimension of a M3 screw was extruded. It will be mounted to the aluminum plate under the modem, between the waterproof cylinders. The placement of the sensor holders was chosen to shield the sensors from environmental factors to protect them.

When printing the sensor holder it is important to follow these instructions to ensure the best result. The holder should be placed on the side in the slicer, meaning the sensor would be pointing upwards if it was clipped in place. Also, 0.2mm should be set as the resolution because a finer resolution will make it less sturdy and resistant to bending. There should also be no need to enable support to print this object.

5.3 Main Plate Modifications

The new design of the ADC had different dimensions, which necessitated a redesign of the main plate for attachment. Previously, the plate was positioned halfway under the I^{2}C-splitter, but it is now placed slightly further out to secure access to all the pin-outs and future use of the Stemma QT connectors. This redesign was carried out in SketchUp Pro after encountering complications with importing the model of the main plate in Fusion 360.

Due to the non-planar underside of the ADC caused by the pinout pins, it cannot be placed directly on the main plate. To address this issue, four 3mm high raised holes were created. Since the screw holes on the ADC are not large enough for M3 screws, the method used to create screw mounts for the Atlas sensors could not be replicated. M2 screws must be used to attach the ADC, so it was decided to have a diameter of 1.8mm for the screw holes on the main plate. This results in a 0.2mm difference between the screw's outer diameter and the hole into which the screw is threaded. The distances between the screw holes were found to be 12,7 mm and 20,32 mm using the dimensions from Adafruit's web page on the ADC [26]. The threads for the Atlas carrier boards were also aligned. The height was differentiated by 1 mm, but is now at the same level, 4mm over the surface of the main plate.

5.4 Calibration of Atlas Scientific's Sensors

Before the sensors from Atlas Scientific could be cut and potted to fit the waterproof casing, they had to be calibrated and tested to verify their functionality. These sensors are the dissolved oxygen sensors and the conductivity sensors. There were two sensors that had already been calibrated by the previous project group. The dissolved oxygen sensor was recalibrated. After the calibration was complete a test was conducted with all the sensors to verify functionality.

5.4.1 Changing Communication Mode

Before calibration and testing were performed the sensors were set to be in I^{2}C-mode. This is done because it is the communication protocol that is used in this project, it is also easier to set up communication with I^{2}C than it is with Universal Asynchronous Receiver-Transmitter (UART). If the color on the EZO-circuit's led is blue, it is already in I^{2}C-mode, and there will be no need to continue reading this section. If however, the color is green, it means the sensor is using the UART protocol. It is still possible to calibrate the sensors using this protocol, but it is recommended to change it to I^{2}C-mode right away to make sure it is not forgotten.

To change the mode to I^{2}C, connect the EZO-circuit board to a breadboard, making sure to place it in such a way that all pins on the EZO-circuit do not short. Then the SDC/TX port can be connected to the PGND port. After this, the VCC and GND can be connected like normal (the easiest way is to use 3.3 V and GND from the RPi). After power and ground are connected the EZO-circuit should light up and change color to blue and the cables can be disconnected. The EZO-circuit is now successfully changed to I^{2}C-mode and can be used as normal. For the wiring diagram, see Figure 5.10.

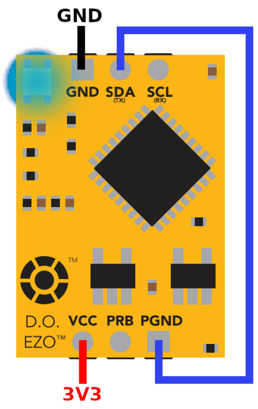


Figure 5.10: Wiring diagram for changing communication.

5.4.2 Calibration Dependencies

Before the calibration can be started ensure that the system is up to date as described in Section 4.2 *Updating System*, also ensure that both *smbus* and *i2c-tools* are installed as described in Section 4.3 textitProgram-Dependent Packages and Libraries.

When the system is up to date and the libraries are installed the code for interacting with the EZO-circuit can to be downloaded. Atlas Scientific provides this code and it can be found on their GitHub [3]. The repository can be cloned from GitHub to the root folder on the RPi with the following commands. Keep in mind that this only works if the RPi is connected to the Internet.

```
$ cd ~/
$ git clone
 ↪  https://github.com/AtlasScientific/Raspberry-Pi-sample-code.git
```

5.4.3 Calibration of the Dissolved Oxygen Sensor

Before calibrating the dissolved oxygen sensor, it is necessary to assess whether the electrolyte solution in the sensor needs to be replaced and if the PTFE membrane on the probe should be changed. These two components should be replaced annually to ensure optimal functionality and prevent drift in the readings [37]. To replace the electrolyte solution, one can follow the instructions provided by Atlas Scientific

[44]. To replace the electrolyte, the membrane should be unscrewed and removed. Then, the probe should be turned upside down, shake it, and gently tap it against a piece of paper to remove any residual solution. Next, the provided syringe should be used to fill the probe with fresh electrolyte solution until it overflows.

Once ready to calibrate the sensor, it is important to ensure that the preceding steps have been followed. This includes setting the sensor to I^2C mode and all dependencies and files have been downloaded as explained in Section 5.4.2 *Calibration Dependencies*. The calibration can then be performed by running the *i2c.py* script from the GitHub repository.

The sensor should be dry and exposed to atmospheric pressure, approximately 101 kPa. The temperature should be maintained at 20 °C. Since the calibration process initially takes place in air, there is no salinity factor to consider unless the test is conducted near seawater. It is worth mentioning, relative air humidity also plays a big part in the values that the DO sensor reads, and therefore also the calibration. This is not mentioned in the datasheet for the sensor, but is mentioned in a blog post written by Atlas Scientific [23].

At this point, the *Poll* command can be executed in the script, and the sensor should start providing readings at approximately one-and-a-half-second intervals. Once the values have stabilized and reached a decimal point precision of tenths, the calibration process can be continued. Terminate the sensor measurement by pressing *Ctrl+C* on the keyboard. Enter the command *cal* in the script and wait for confirmation from the program. Proceed by executing the *Poll* command and ensure that the sensor now reads values around 9.09 mg/L. The high point calibration has now been completed.

The next step is to calibrate the low point by submerging the sensor in the Zero Dissolved Oxygen calibration solution. Open the package and place the sensor inside. Gently stir and agitate the sensor to remove any air bubbles. Once again, wait for the readings to stabilize. This process may take longer than stated in the Atlas Scientific guide, which suggests a range of 30 to 90 seconds. When four consecutive sensor readings are consistent, assuming the sensor is sampling at 1.5-second intervals, the measurement can be interrupted, and the command *cal, 0* can be entered. The calibration process is then considered complete.

5.4.4 Calibration of the Conductivity Sensor

To calibrate the conductivity sensor, one must execute the sample code, *i2c.py.* It is also essential to ensure that the steps outlined in Section 5.4.2 *Calibration Dependencies*, have been followed before initiating the calibration process. Once the *i2c.py* script is executed, sensor values can be read using the *poll* command. The sensor is likely to provide values close to zero in a dry state. When the values stabilize, the sensor reading can be terminated with *Ctrl+C*, followed by the command *cal, dry.*

Two salt solutions accompanying the sensor kit will be used for calibration: a low conductivity solution with 12880 micro Siemens per centimeter and a high conductivity solution with 80000 micro Siemens per centimeter. It is important to note that the bottle has a table printed on the label indicating the conductivity at different temperatures. Familiarize yourself with this table before proceeding with the calibration. Pour a small amount of each solution into separate glasses.

Read sensor values and place the sensor in the low conductivity solution first. Wait until the values stabilize, then enter the command *cal,low,12880* into the program. Rinse the sensor with freshwater and ensure it is dry. Once the sensor is clean and dry, read the values and place it in the high-conductivity solution, waiting for the sensor values to stabilize. It is worth noting that this process takes a long time and the readings may never stabilize completely. As there are no specific instructions on the duration, it is necessary to reach a satisfactory point. To calibrate the high point, enter the command *cal,high,80000.* The sensor should now be calibrated and ready for use. Once the conductivity sensor is calibrated, there should be no need for re-calibration [38].

5.5 Functionality Tests of Sensors

In order to say anything about how the sensors behave they had to be tested against known values. This section covers a series of tests that were performed.

5.5.1 Functionality Tests of Dissolved Oxygen Sensors

In section 6.4.1 *The functionality test of the sensors* from the bachelor thesis in 2022 [25] it was claimed that the dissolved oxygen sensor did not adhere to the laws of physics. The basis for this claim was that there was a test done on the dissolved oxygen sensor where it was put in both cold and warm water. As explained in

Section 3.11 *Understanding Dissolved Oxygen in Water*, cold water can hold more dissolved oxygen than hot water. The test from the 2022 thesis found that the dissolved oxygen sensor measured more dissolved oxygen in hot water than in cold water. This is the opposite of the expected results and to tackle this problem two tests were devised:

Verification of Thesis 2022 Dissolved Oxygen Test
A test to verify the findings from the 2022 thesis was created. This test included two tea cups, one filled with $\approx 20°C$ tap water. The other cup was filled with warm water $\approx 45°C$ water that was created from mixing hot and cold tap water. The probe would then be put in the cold water cup first, so in the warm water cup, and finally back to the cold water cup. The temperatures were found using a TP101 thermometer.

Test of Dissolved Oxygen Sensor With Salt Water
A functionality test without the temperature variable was created to see if the functionality of the sensor could be tested in another way. The solution was to test two bowls with differing salt concentrations.

The preparation for the test was done by filling 0.5 l water in two bowls, the water was collected from the tap and was measured to be 21.0°C. The temperature was measured using a TP101 thermometer. One bowl contains fresh water and the other one contains a saline solution. The bowl containing the freshwater was left as is, while the bowl containing the brine solution had to be made.

The salt water solution was made by mixing salt and water. The mixture was made up of 178 grams of common salt (NaCl) and 0.5l tap water, this yields a concentration of 26.25%. The solution was then mixed until no more salt could be dissolved and left for ≈ 24 hours.

The test was also run with a pump in the bowl, the utilized pump constituted a Submersible DC motor [27] which the students had at their disposal, obtained from prior projects.

Functionality Tests of all Dissolved Oxygen Sensors
Once it was established that the dissolved oxygen sensor works as expected with salt water a functionality test for all three sensors was devised. The objective of this test was to confirm the proper functioning of all the sensors and their ability to accurately measure dissolved oxygen levels.

The test procedure is the same as the test described in the previous paragraph but with different variables. In this test, both bowls were filled with 0.6 l water. The salt water solution was made with 15 grams of salt. The bowls were then left for $\approx$ 24 hours before the test was conducted. In addition, no pump was used in this test. All three sensors were placed in the same bowl simultaneously to ensure that they would measure the same environment.

Since the bowls were left for $\approx$ 24 hours some water had evaporated before the start of the test. The original salt concentration was 2.44%, but if we estimate that 0.1 l water evaporated we can calculate the new concentration to be 2.91%.

5.5.2 Functionality Tests of Conductivity Sensors

The test for the conductivity sensors was executed simultaneously with the functionality test of all dissolved oxygen sensors. They were testing the same medium and put in each of the solutions at the same time as the dissolved oxygen probes.

5.5.3 Functionality Tests of Pressure Sensors

A test was conducted to verify the functionality of the two pressure sensors that were yet to be deployed. The objective of the test was to confirm the proper functioning of the sensors and their ability to accurately measure pressure levels.

The test procedure involved placing the rig in an elevator and taking the elevator from the fifth floor down to the ground floor, and then up again to the fifth floor in a building at Gløshaugen. During the test, the pressure was logged using the logger node. Each sensor was tested separately with a sample time of 1 second.

By riding down and up in the elevator, the pressure on the sensors changed in accordance with the variation in height. This allowed for the observation and recording of pressure fluctuations at different floors. The data from the sensors were analyzed to assess whether accurate and consistent readings were provided.

5.5.4 Functionality Tests of Temperature Sensors

There were two temperature sensors that had not yet been utilized or tested. These sensors were intended to be integrated into the final two rigs. Tests were conducted to assess their functionality. The tests were performed on one sensor at a time with a small interval between them. This was caused by the process of changing the sensor and restarting the program.

Transient Test

The first test conducted was a transient test to observe the behavior of the sensors when exposed to a sudden temperature change. Two glass bowls containing water at different temperatures were used for this purpose. One bowl had a temperature of 23 °C, while the other bowl had a temperature of 33 °C. These temperatures were measured using a TP101 thermometer. After the water temperatures were recorded, the test was initiated as quickly as possible to ensure comparable results between the sensors. The temperature of the warm water decreases relatively rapidly.

Initially, temperature sensor 2 was placed in the cold water bath for 30 seconds and then swiftly moved to the warm water bath for another 30 seconds. The same procedure was repeated with temperature sensor 1 afterward.

The data was acquired by running the program for the entire rig using the launch file. The logger node would then log all sensor values and store them. The values were extracted, plotted in graphs, and analyzed.

Precision Test

The next test aimed to assess the precision of the two sensors and verify their accuracy in measuring temperature. The test was conducted in a water bath that had reached room temperature. The temperature of the water was measured as a reference using the TP101 thermometer. The water temperature was measured both before and after each test, including during the test of temperature sensor 2.

Initially, sensor 1 was placed in the water bath for a few minutes prior to the test. At that time, the water was measured to be 23.5 °C. The test was initiated by running the launch file and logging the results. The sensor remained submerged for 5 minutes before the test concluded. Subsequently, temperature sensor 1 was replaced with sensor 2. Sensor 2 was immersed in the water for a couple of minutes before the test commenced, which was 8 minutes after the completion of the previous test. The temperature of the water was then measured to be 23.3 °C. Sensor 2 was tested for a total duration of 3 minutes. In the middle of the test, the TP101 thermometer was placed in the water to measure the water temperature, which was found to be 22.8 °C at the end of the test.

5.5.5 Functionality Tests of PSM

To verify the functionality of the Power Sense Module (PSM) and the custom-developed code responsible for reading values from the ADC, several tests were conducted to ensure accurate and consistent measurements. In the early stages of the project, it was observed that the PSM successfully transmitted correct voltage measurements but failed to transmit current measurements. The cause of this remains uncertain. Results from these initial tests were not logged, thus lacking any recorded data.

A subsequent test was performed later in the project phase, specifically using the red IES. The objective of this test was to compare the values obtained from the ADC with the actual applied voltage. The IES was connected to a digital PSU capable of displaying the voltage and current flowing through its output. The IES was connected to the positive (+) and negative (-) terminals of the PSU through the battery cables, simulating a similar cable resistance as if the battery were connected.

Initially, a voltage equivalent to the maximum battery voltage (25.5V) was applied by adjusting the knob on the PSU. The voltage was then gradually decreased by 25% of the battery voltage range at one-minute intervals until it reached the minimum battery voltage (19.8V). Subsequently, the data log was retrieved from the SD card and analyzed. This process was repeated multiple times to assess the repeatability of the measurements.

5.6 Vacuum Testing

It is essential to pressure-test all rigs before submersion in water to detect any potential leaks. This section will describe how to perform such a test, how to troubleshoot leaks, which rigs were tested, and their configuration while the tests were conducted.

5.6.1 Vacuum Testing Procedure

Before conducting the pressure test, it is important to ensure that all seals are properly secured. Each seal should have an O-ring with grease applied, and the lids of the watertight cylinders should be tightly fitted and greased prior to installation. To eliminate any faults in the vacuum pump, seal the nozzles with a rubber stopper and pump it to 15 inHg for 30 seconds to a minute. If the needle stays stationary the vacuum pump is ready to be used.

When performing a pressure test on the rig, it is critical that both cylinders are tested simultaneously, as the battery cables between the cylinders will leak air if one end is not pressurized properly. To perform the test, a vacuum pump with a splitter should be used. Each nozzle from the splitter should be connected to its own cylinder (battery cylinder and sensor cylinder). The connection is made to the pressure relief valve on the cylinders. The blue plug on the cylinder should be unscrewed and the nozzle of the pump inserted, ensuring that grease is applied to the nozzle beforehand.

The test can then be performed by pumping out the air by pressing the hand pump repeatedly until the needle of the manometer has reached 15 inHg. This corresponds to approximately 0.5 bar. The purpose is not to test the rig's ability to withstand high pressure, but to detect any possible leaks. When the manometer has reached 15 inHg, carefully lay down the hand pump and wait. If the needle has moved within two minutes, it can be confirmed that the cylinders are leaking, and troubleshooting can begin immediately. If the pressure holds, the test can continue. According to the supplier of the waterproof cylinders, the test should last between 10 to 15 minutes. It can be tested longer than this to be on the safe side.

5.6.2 Configurations of the Performed Vacuum Tests

Several vacuum tests were conducted during the project period. In total, two rigs and two acoustic modems were tested. The tests were performed according to the procedure described in Section 5.6.1 *Vacuum Testing Procedure*. A reliable method to differentiate between the rigs has not been established yet. Therefore, for clarity, the rig originally created as part of the previous project phase in 2022 will be referred to as the *blue rig* as it contains the blue IES. The second rig developed by this project group will be referred to as the *red rig* because the red IES belongs to the rig.

5.6.2.1 Configuration of the Blue Rig during Vacuum Test

Multiple tests were conducted on this rig with various configurations. Initially, the rig was tested without the modem cable attached, followed by tests with the cable connected. During the resulting test with the modem cable, the following configuration was utilized.

All sensors were connected to the sensor cylinder during the test. This included pressure, temperature, conductivity, and dissolved oxygen sensors. Additionally, the modem cable, battery cables, a blank bulkhead, and two pressure relief valves were connected. One valve was of the old type (red), while the other was of the new type (blue). The test was conducted using the new valve. The battery cylinder had the battery cables connected, along with three blank bulkheads.

5.6.2.2 Configuration of the Red Rig during Vacuum Test

The configuration of the red rig was similar to that of the blue rig. The battery cylinder had an identical configuration, while the sensor cylinder had two fewer sensors. It was connected to a pressure sensor, a temperature sensor, the modem cable, battery cables, a pressure valve (new type), and four blank bulkheads.

5.6.3 Vacuum Testing the Acoustic Modem

The two Subnero modems were tested before an intended field test. The Subnero modems are equipped with the old pressure valve mounted on their rear side. To perform a pressure test on the modems, the same procedure as testing the cylinders can be followed. It involves applying a pressure differential of 15 inHg and allowing it to remain for 10-15 minutes.

5.6.4 Troubleshooting Leaks

If a pressure drop is observed, the following steps can be troubleshooted:

- Check all O-rings and make sure they are not dusty or have any dirt attached to them.
- Check that all O-rings have grease on them.
- Check that all penetrators on the lid are sufficiently tight.
- Check that the lid sits securely and has no gap.
- Both cylinders are being tested simultaneously.
- Use the elimination method by replacing one bulkhead fitting at a time with a blank to identify potential leakage in either the sensor or cable.

5.7 Bill of Materials

To facilitate the creation of two new rigs, a request was made for a bill of materials encompassing all the components of the Standalone Underwater Monitoring Station. Identifying all the components proved to be a time-consuming task, as the system consists of over 30 different parts. In order to provide the most comprehensive overview for the continuation of the project, an Excel document has been created listing all the parts involved. Each component is assigned a unique number, category, comment, and links to the datasheet and preferred store for procurement. Additionally, a sheet for spare and maintenance parts, as well as a table listing all the screws used, has been included in the document.

Chapter 6

Results

The Results chapter showcases visual representations, such as images, highlighting the structural parts of the project. It provides a comprehensive understanding of the modified and self-developed hardware components, as well as the cables/adapters made as part of the project. Furthermore, it presents a general overview of the system's architecture and design.

In addition, the section presents the outcomes of the sensor testing, supported by corresponding graphs. These results demonstrate the performance and characteristics of the employed sensors.

6.1 Structural Parts

6.1.1 Subnero Mounting Brackets

In order to securely mount the Subnero modem onto the Standalone Underwater Monitoring Station (SUMS), two brackets were designed and fabricated. The brackets, namely the Front Bracket and Rear Bracket, were created using Fusion 360 software and 3D printed using Prusa mk3 printers with PLA.

The front bracket, shown in Figure 6.1, is designed to securely hold the upper part of the modem near the transducer. It provides a secure attachment point that can lock the transducer's protective cage in place using cable ties or steel wire. This is demonstrated in Figure 6.2.

The Rear Bracket, shown in Figure 6.3, wraps around the plastic cylinder of the modem, providing additional support and stability.

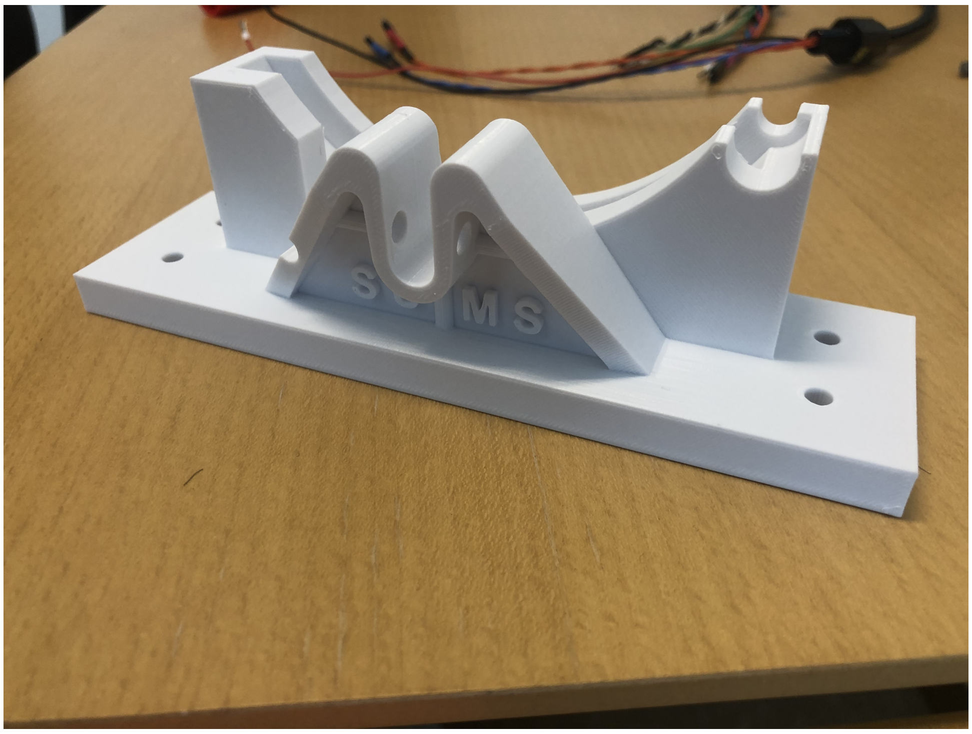


Figure 6.1: Final result of the Subnero front bracket.

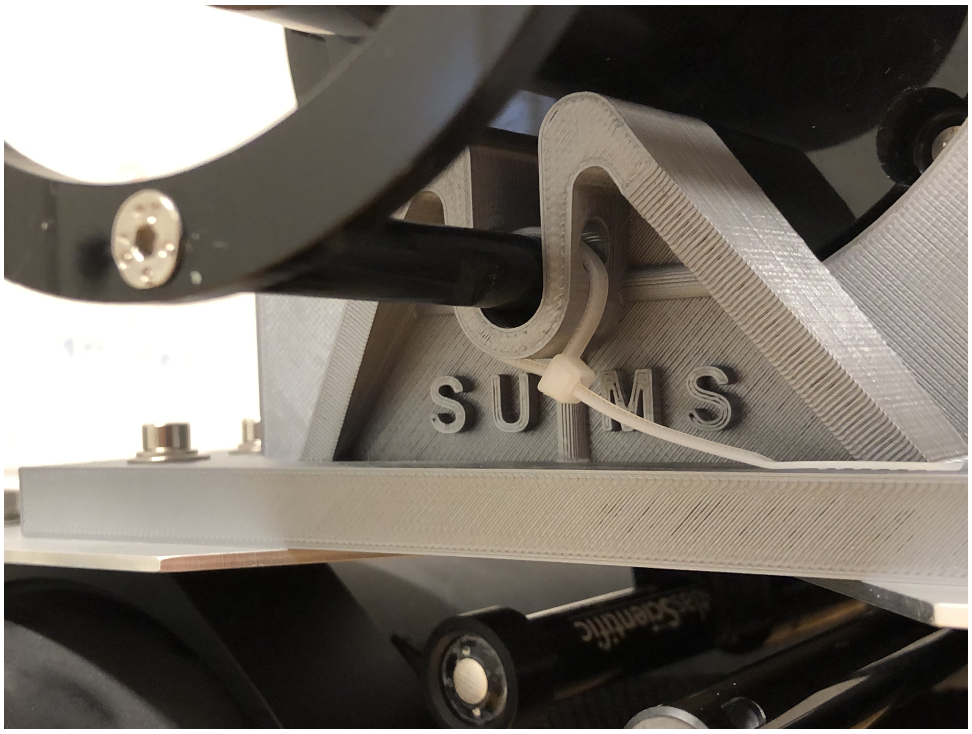


Figure 6.2: Front bracket locking mechanism.

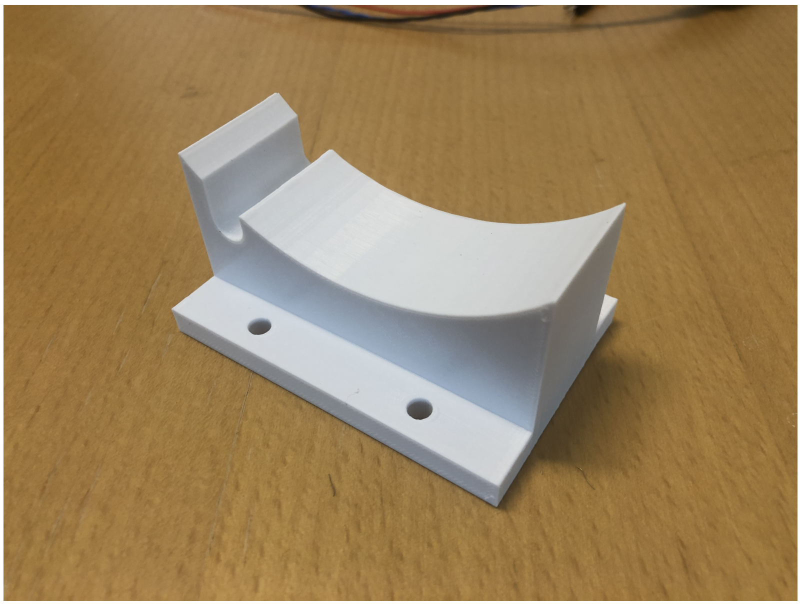


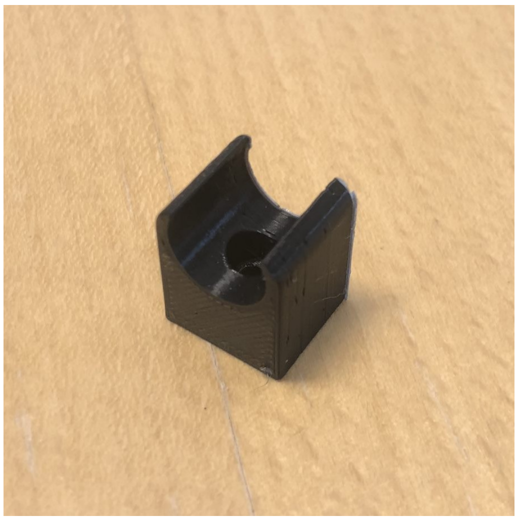
Figure 6.3: Final result of the Subnero rear bracket.

The design and fabrication of these brackets ensure the Subnero modem is held securely in place to the sensing rig. These brackets were specifically tailored for the project and play a crucial role in maintaining the structural integrity of the system.

6.1.2 Atlas Scientific Sensor Holder

In order to securely mount the Atlas Scientific sensors onto the sensing rig, custom sensor holders were designed and manufactured. These holders are simple clips that allow the sensors to be easily attached by pushing them into place. Similar to the previously mentioned brackets, the sensor holders were also 3D printed using a Prusa mk3 printer and PLA filament.

Figure 6.4a showcases the result of the sensor holder design. While Figure 6.4b showcases the positioning of the sensor with the sensors attached to the sensor holder.


(a) 3D-printed sensor holder.

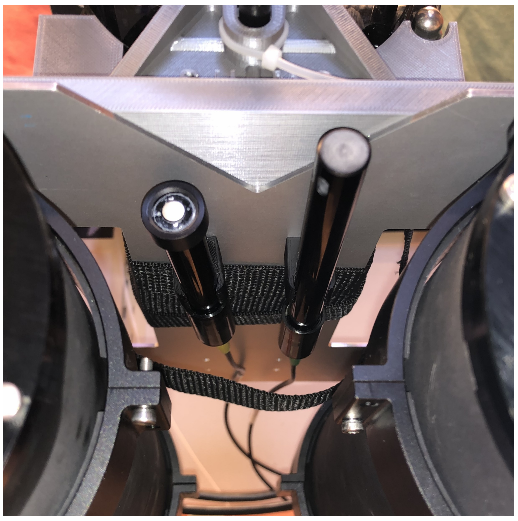

(b) Atlas Scientific sensors clipped in place.

Figure 6.4: Final result of the Atlas sensor holder.

6.1.3 Aluminium Mounting Plate

The aluminium mounting plate serves as the structural backbone, holding all the components of the rig together. A detailed digital model of the plate was created using Fusion 360 software, which served as the basis for manufacturing the physical plates. The fabrication process involved utilizing a CNC machine at the ITK Mechanical Workshop.

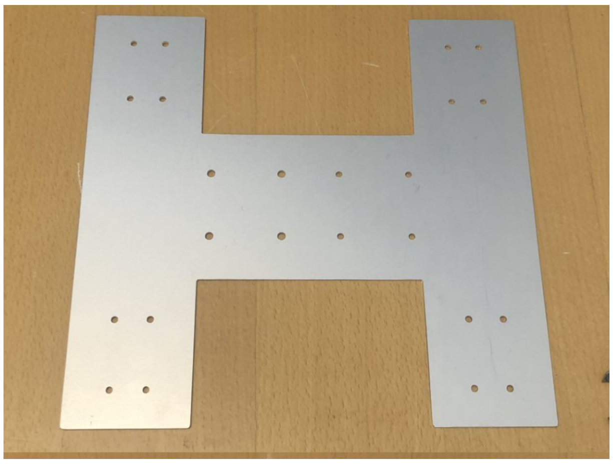


Figure 6.5: Final result of the aluminum mounting plate.

The mounting plate is designed as a foundation upon which the various components are securely mounted. Refer to Figure 6.5 for a visual representation of the aluminum mounting plate. Notably, the rear portion of the rig corresponds to the left side of the figure, while the front portion corresponds to the right side. This can be observed by the presence of larger holes at the rear, specifically intended for the attachment of the rear bracket.

There were produced two aluminum mounting plates in total to complete a total of three rigs.

6.2 Hardware and Cables

6.2.1 Modem Cables

A total of four modem cables have been manufactured, where two of them were pressure tested and three verified functional in conjunction with the modem and sensor rig. One cable was damaged during the epoxy casting process and rendered unusable.

6.2.1.1 MCOM8M Plug Cable

Three cables of MCOM8M plug type were manufactured, with two of them being cast using the original rubber boot, while one was cast using heat shrink tubing. All cables share the same type of cable and bulkhead.

The production of cables using this method has resulted in two completed and functional cables. They have undergone and passed the vacuum testing at 15 inHg while being connected to the rigs during the test. It is important to note that they have not been tested under high pressure conditions. See Figure 6.6 for the result of a cable of this type with the original rubber cap casting method.

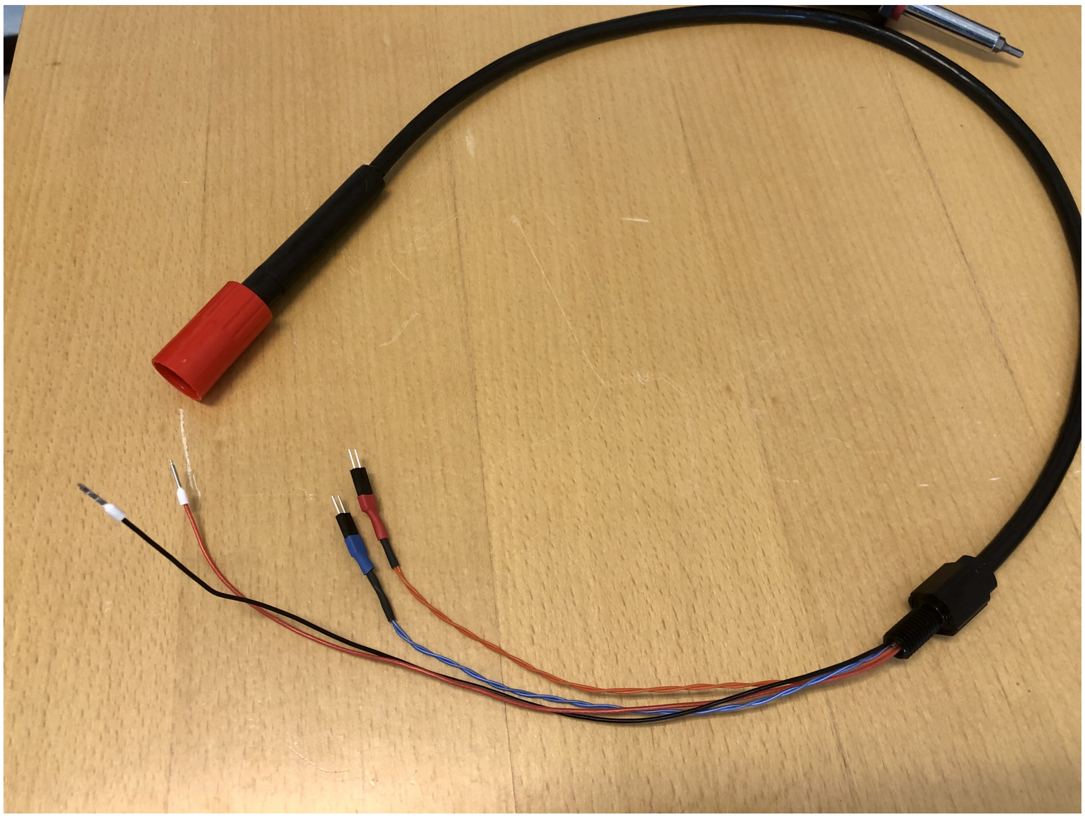


Figure 6.6: Modem cable with MCOM8M and original rubber cap.

The MCOM8M plug that was cast with a heat-shrinking tube is presented in Figure 6.7. While shrinking the end of the heat shrink the cable jacket got exposed to a higher temperature than it could withstand which resulted in a deformation. Refer to Figure 6.8.

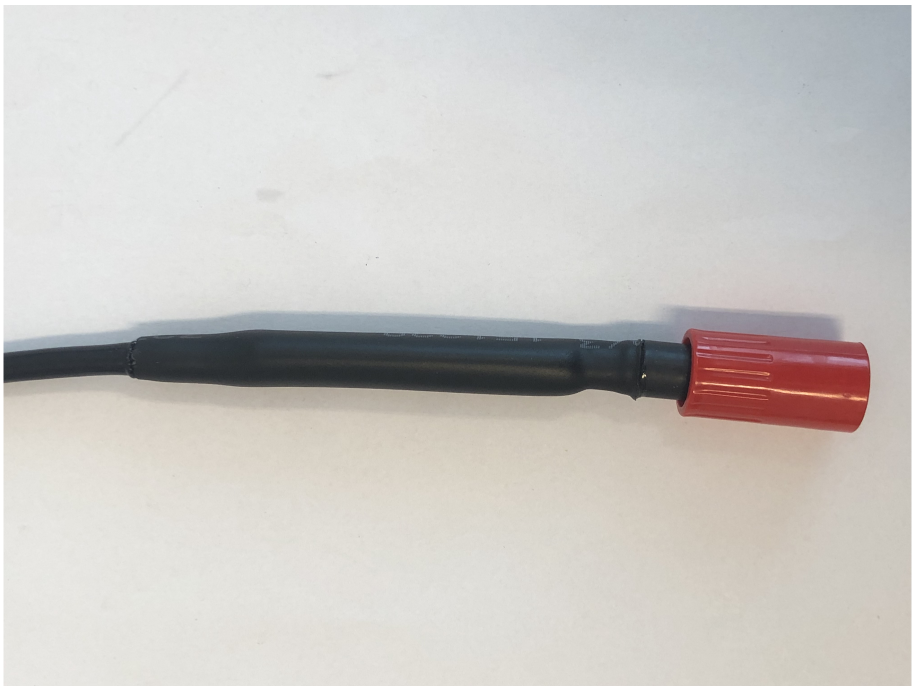


Figure 6.7: Modem cable with MCOM8M with heat shrink.

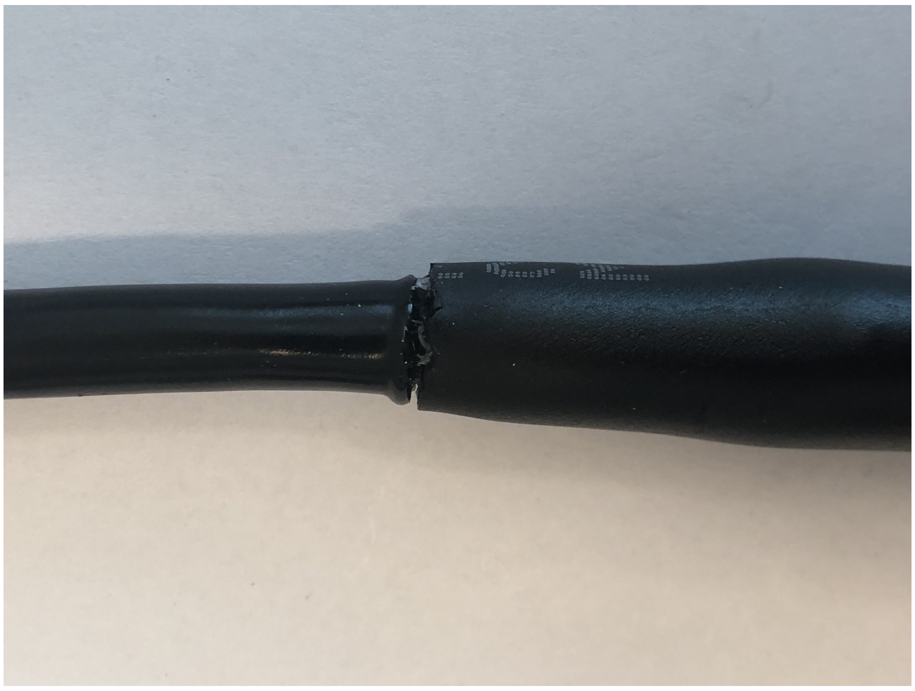


Figure 6.8: Deformation on cable after heat shrink.

6.2.1.2 MCIL8M Pigtail Cable

A cable with an MCIL8M pigtail was produced. This cable is depicted in Figure 6.9. It features a different WetLink penetrator compared to the other cables fabricated in the project, as the pigtail cable it comes with has a larger diameter than the cable used with MCOM8M. According to the cable's datasheet, the pigtail cable has a diameter of 9.2mm. The WetLink 9.5mm high-compression was utilized in conjunction with the MCIL8M pigtail cable.

The cable underwent functional testing using a multimeter to verify conductivity and insulation between conductors. It has also been validated with the modem, although it has not been subjected to a vacuum test.

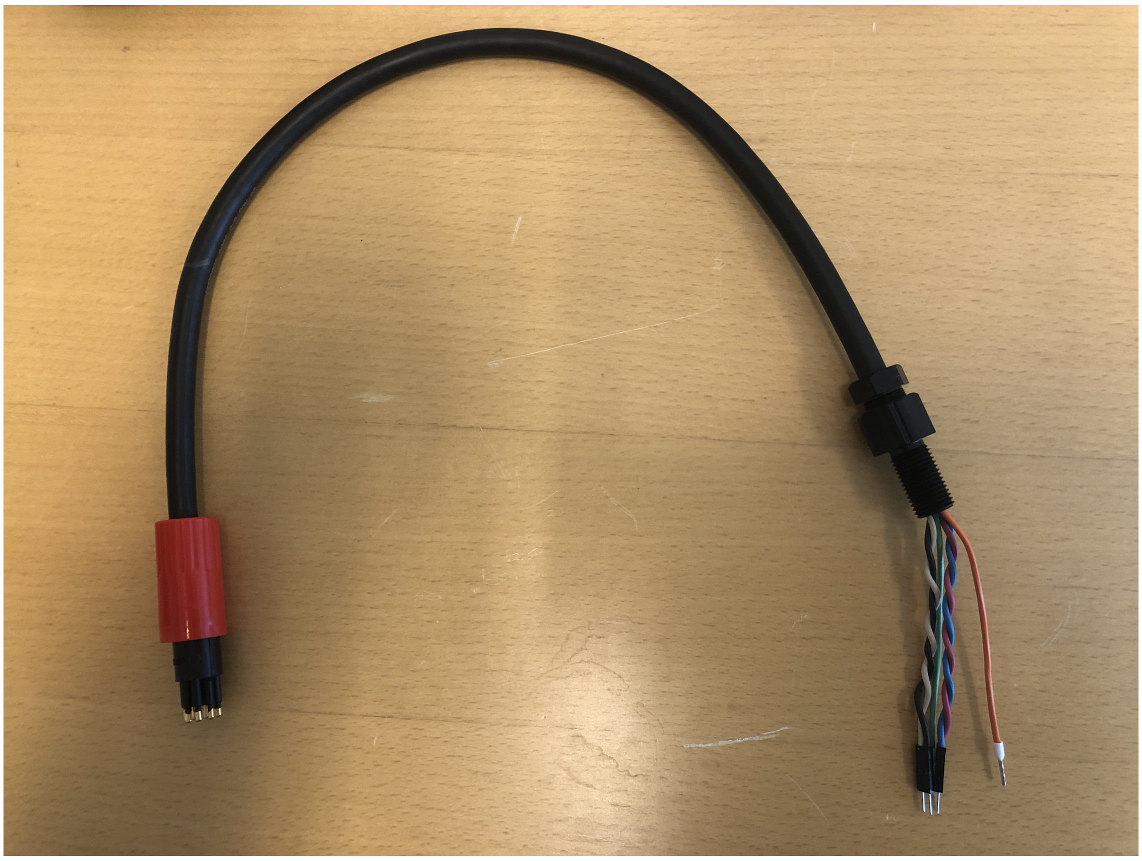


Figure 6.9: Modem cable made with MCIL8M.

6.2.2 Switch Adapters

6.2.2.1 PicoBlade to RJ45 Adapter

A adapter with 8p PicoBlade in one end and a RJ45 female connector in the other end. The Rj45 connector is made by Schneider and the component is the Actassi S-One C6 Connector RJ45. To communicate with the sensing rig while the modem is connected, this adapter is needed to connect a computer directly to the switch and use SSH to communicate. The adapter is shown in Figure 6.10. The cable consists of four twisted pairs, enabling its use for 1000BASE-T applications.

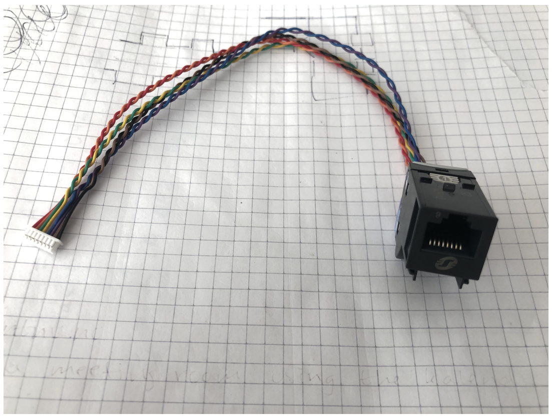


Figure 6.10: PicoBlade 8p to RJ45 female adapter.

6.2.2.2 PicoBlade to Molex SL adapter.

This adapter is made for connecting the acoustic modem to the switch. It is made with two dual molex SL female connectors. The cable consists of two twisted pairs, enabling its use for both 100BASE-T and 100BASE-TX applications. The adapter is presented in Figure 6.11.

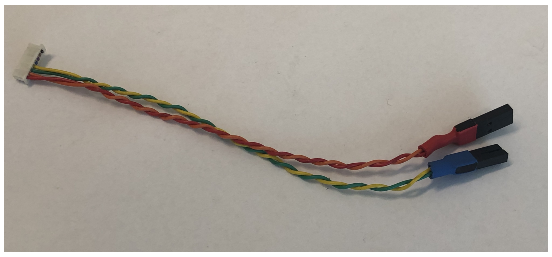


Figure 6.11: PicoBlade 8p to Molex SL female adapter.

6.2.3 Power Supply

The final result of the Power Supply Unit (PSU) is a modified Traco THN 15-2411WIR. Two PSUs were utilized and had different cable connectors. The first one was made with the cables from the broken PSU from Blue Robotics, and the second one was made with cables and a USB-C power connector which was soldered to the cable. See Figure 6.13. The negative terminals on the PSU were shorted using wiring and isolated with heat shrinking tube. See Figure 6.12. The positive terminals were kept isolated and cables were soldered to them. The input cable is red and black corresponding +Vin and -Vin. The input cable had the same color coding as the input.

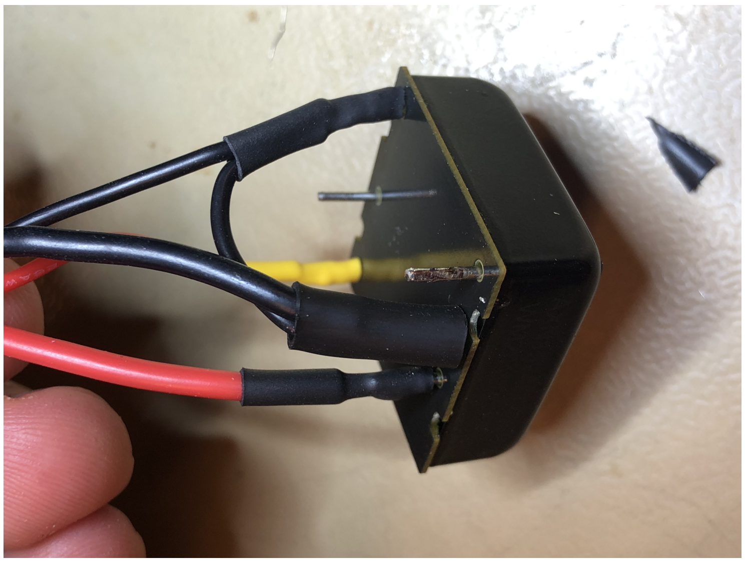
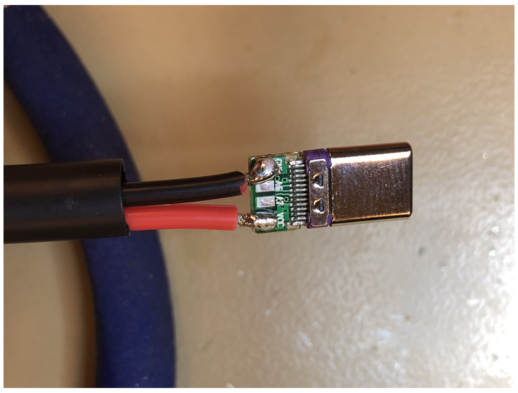
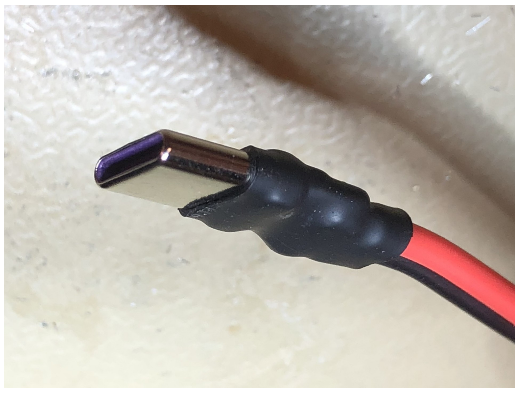


Figure 6.12: Shortage between -Vin and -Vout on Traco THN 15-2411WIR.



(a) Soldering point on the USB-C.



(b) Final result of PSU output.

Figure 6.13: USB-C on the PSU's output cables.

6.2.4 Power Sense Module

Power Sense Module (PSM) underwent some modification during the project involving soldering the ground cable to the IC and crimping a dual Molex SL female connector to the data cable to be able to connect it to the ADC. The result of this is shown in Figure 6.14 and Figure 6.15.

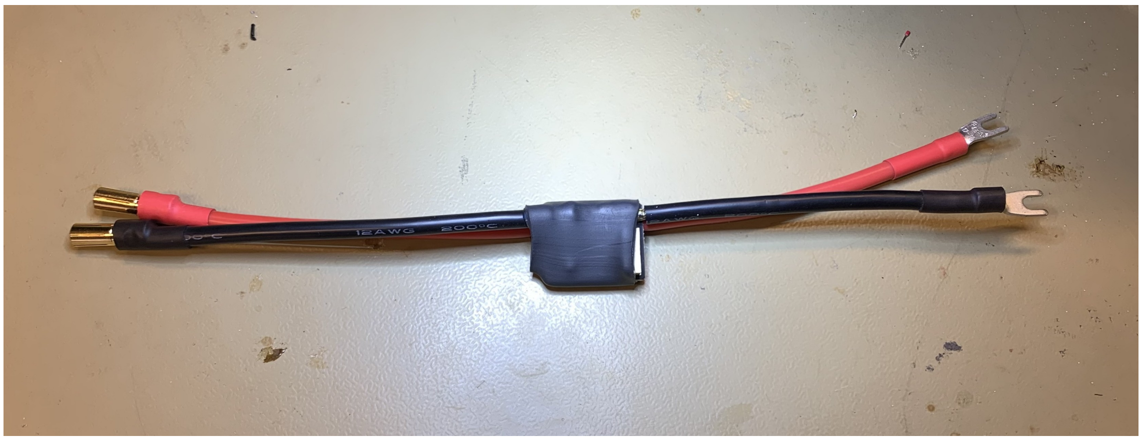


Figure 6.14: PSM with the soldered ground and new heat shrink.

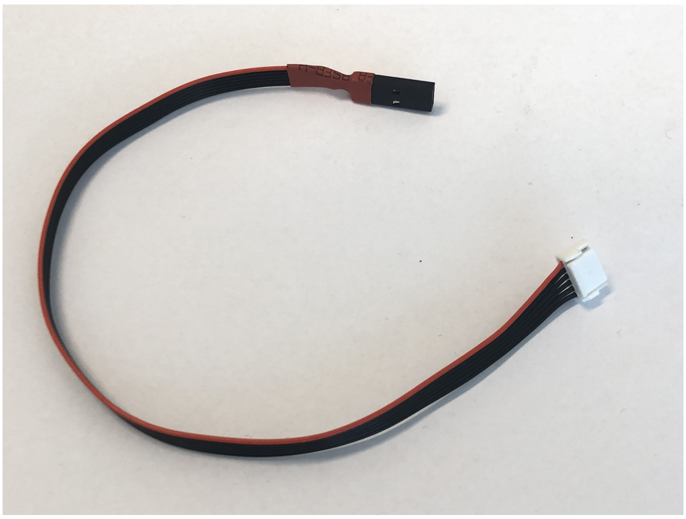


Figure 6.15: PSM data cable with 6p JST-GH to dual Molex SL.

6.2.5 System Wiring

This is the final wiring diagram for the whole system is shown in figure 6.16. Note that connections like the bullet connectors on the battery cables are excluded from this diagram. Also, note that not all black cables are GND. The color coding in the diagram is described in the figure itself.

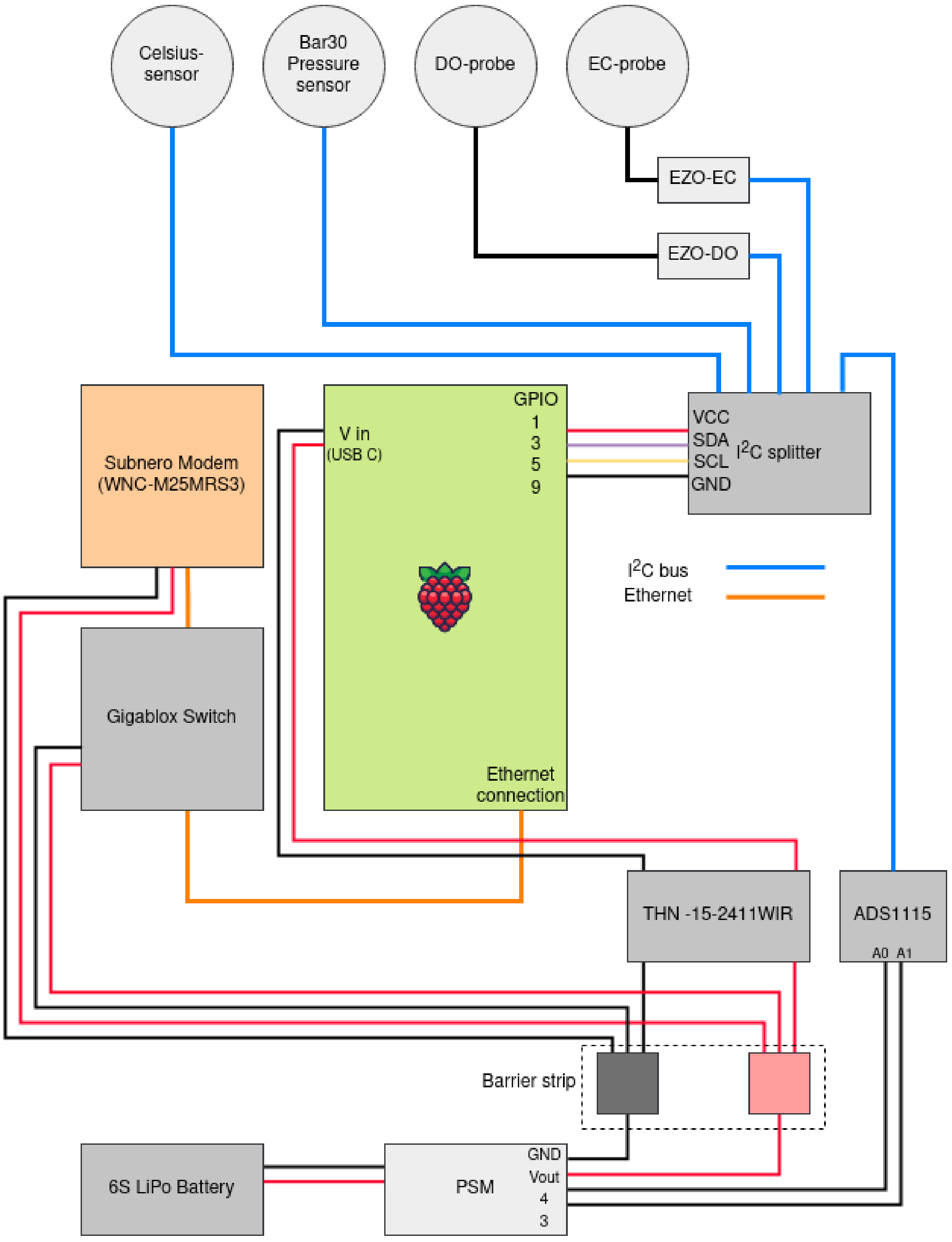


Figure 6.16: SUMS wiring diagram.

6.3 Rig's Performance Under Vacuum Pressure

The blue rig underwent a vacuum pump test to check its ability to maintain pressure. The test was conducted under a relative pressure of 15 inHg for just under a day. The needle on the pressure gauge did not move from its position since the pressure was pumped out. This suggests that the rig is capable of being submerged in water without leaking. The cylinders that were tested were of the older generation from Blue Robotics watertight enclosure tube series. It was first tested without the manufactured modem cable installed.

Later on, the rig was tested again with the modem cable attached to it. The rig underwent a vacuum test of 15 inHg for 15 minutes with no leaks, suggesting it is ready for use in water with the configuration given in 5.6.2.1.

The red rig was assembled to perform a field test. The new generation of the watertight enclosure tube series was used to manufacture the rig. Both rigs were tested under the same conditions and had the same results, making both rigs ready to be submersed. The red rig was configured as described in Section 5.6.2.2.

6.4 Sensor Results

6.4.1 Dissolved Oxygen 2022 Verification Test

Figure 6.17 shows the results from the test described in section 5.5.1. Samples 1 - 7 (orange) is the probe exposed to air, samples 7 - 100 (blue) is the probe in the cup with 20 °C water, samples 100 - 200 (green) is the probe in the cup with the 45 °C water, lastly samples 200 - 300 (blue) is the probe back in the cup with 20 °C water but now warmed up from sitting in the cup with warmer water. The transients show the probes' reaction to different temperatures.

The sample time was $T = 0.6$ seconds, the test lasted for ≈ 3 minutes.

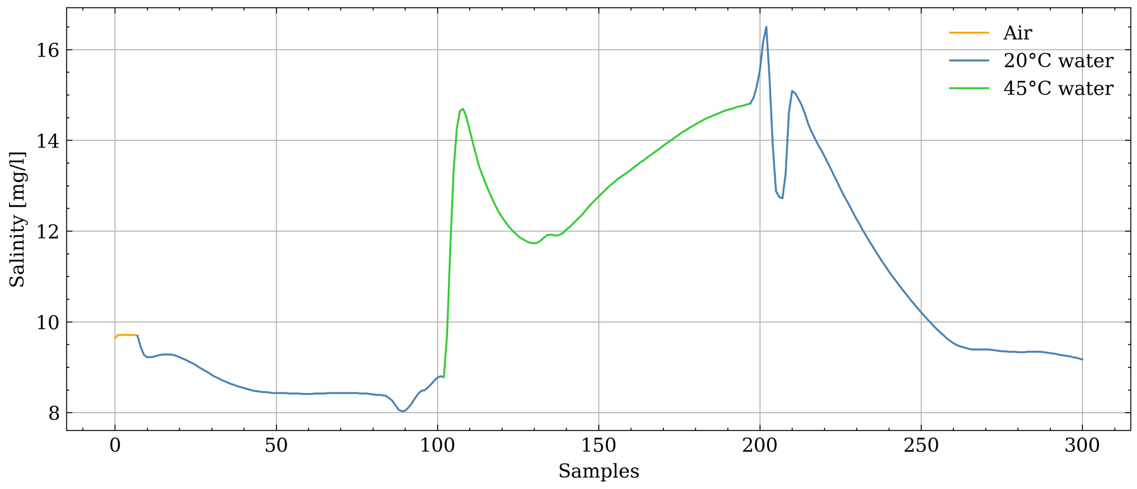


Figure 6.17: Test to verify the findings from 2022 thesis.

6.4.2 Dissolved Oxygen Saline Solution Functionality Test

Figure 6.18 shows the results from the test described in section 5.5.1. The test lasted for 7 minutes, and a sample time of $T = 0.6$ seconds was used. As the bowls were left overnight, some of the water evaporated when the test was conducted, in addition, more salt than what the water could dissolve was used, therefore some salt remained undissolved at the bottom of the bowl.

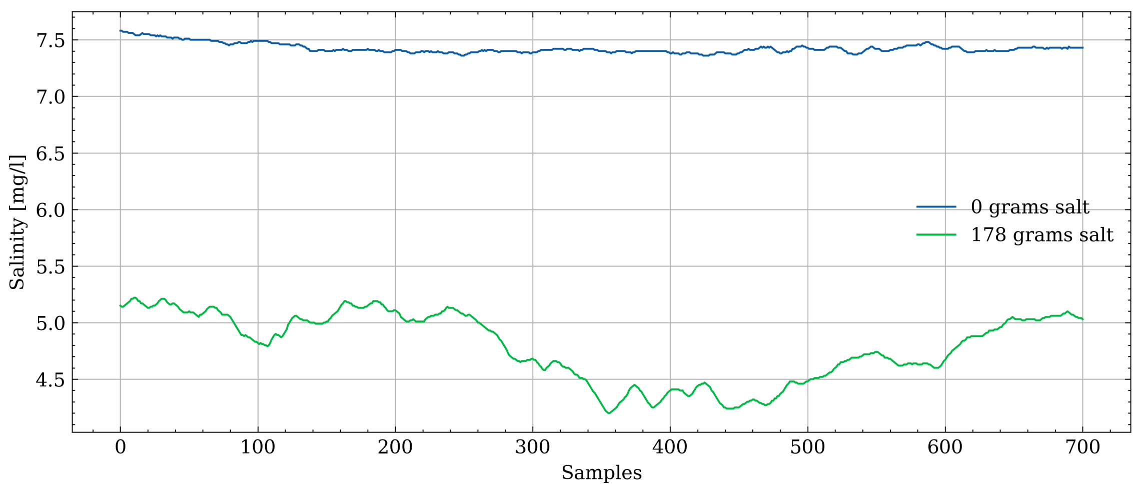


Figure 6.18: Test to verify the dissolved oxygen sensor.

6.4.3 Data from Dissolved Oxygen Sensors

A series of tests were conducted in controlled environments. All three dissolved oxygen sensors were initially calibrated following the method described in Section 5.4.3. After the calibration, the tests were performed as described in 5.5.1, and the results are presented below. The first test was done in freshwater and all the sensors were placed in the water at the same time for a duration of 10 minutes. The temperature was measured before the tests and both fresh- and saltwater was found to be 20.6°C. The sensors had been dried and exposed to air before the test. The result is shown in Figure 6.19.

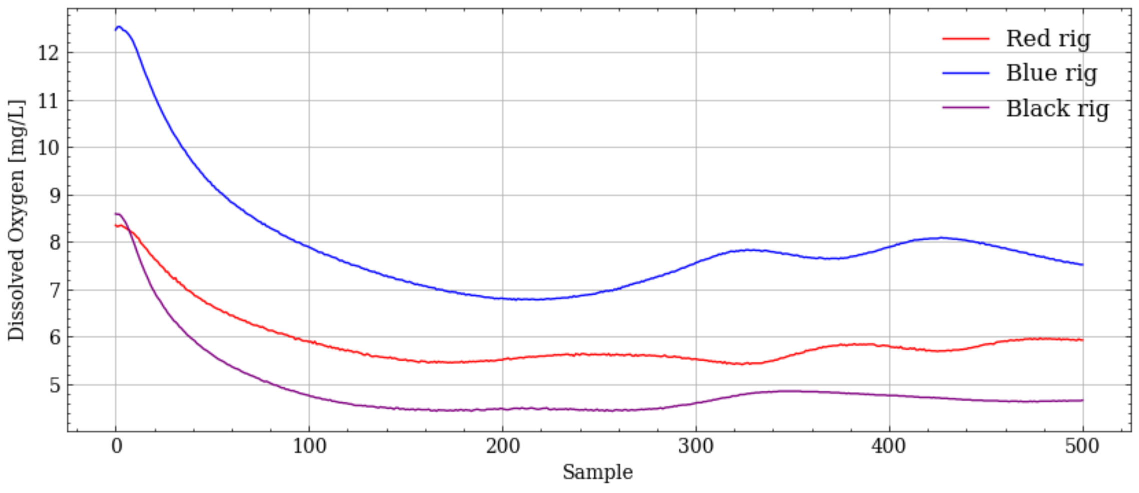


Figure 6.19: Dissolved oxygen sensors in freshwater.

The next test was done in saltwater with the same test period. The sensors were dried and exposed to air for ≈ 5 minutes before the test was conducted. The saltwater was made from 0.6l of water and 15 grams of salt(NaCl) containing iodine(0.5mg/100g). The solution rested for more than 24 hours before the test was conducted. The result is shown in Figure 6.20.

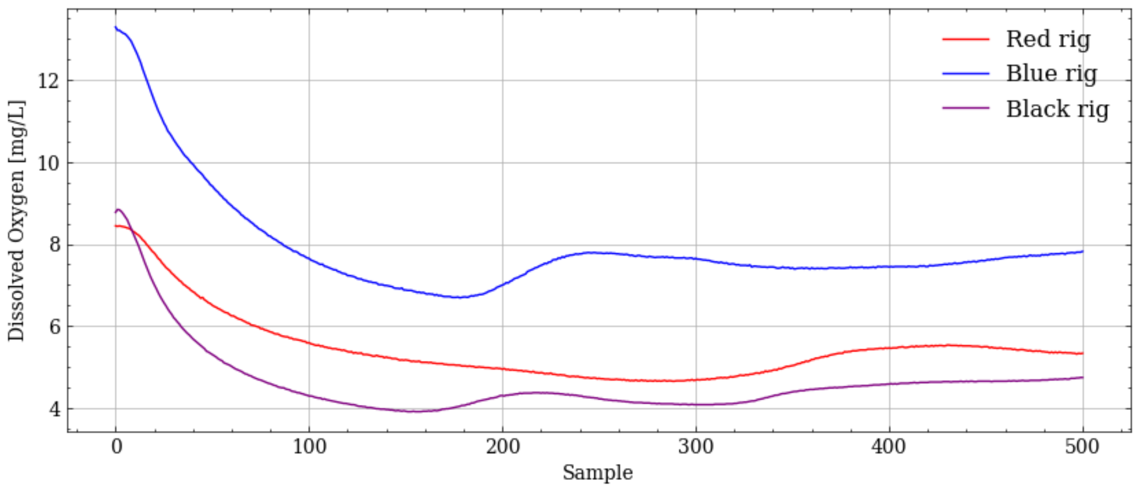


Figure 6.20: Dissolved oxygen sensors in saltwater.

All of the values from the DO sensors were plotted side-by-side to compare their value in salt versus fresh water. This is illustrated in Figure 6.21

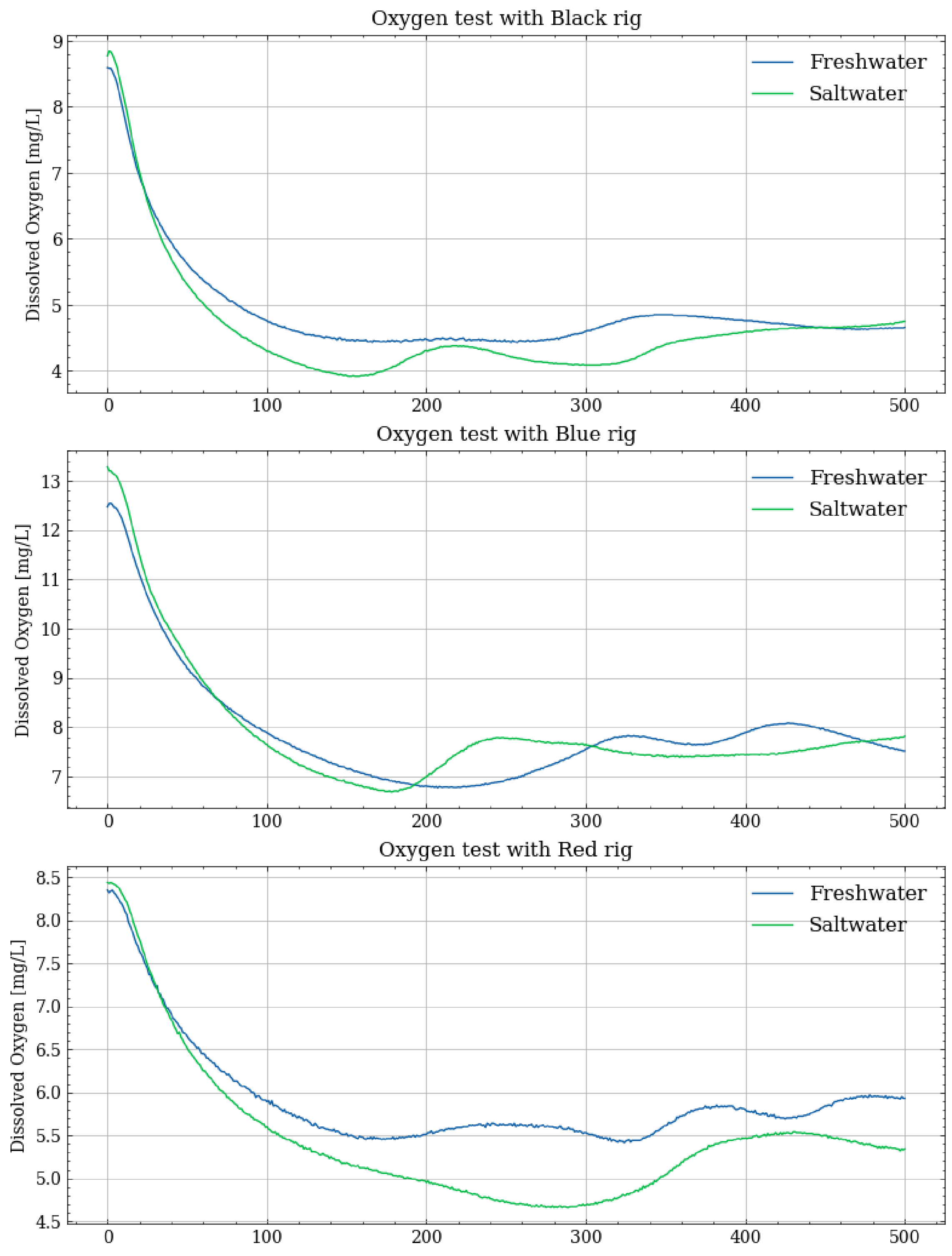


Figure 6.21: Comparison of oxygen levels in freshwater VS saltwater, T=1.2s.

6.4.4 Data From Conductivity Sensors

The conductivity sensors were tested alongside the dissolved oxygen sensors. All sensors were subjected to the same conditions under each of the two tests. The result from the conductivity sensors in freshwater is shown in Figure 6.22

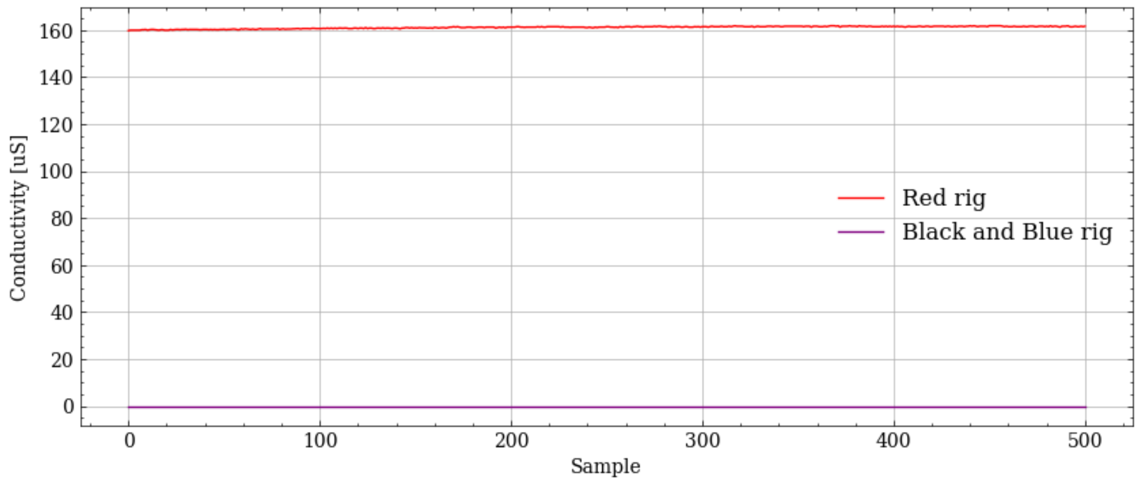


Figure 6.22: Conductivity sensors in freshwater.

The result from the test in salt water is shown in Figure 6.23. The saltwater solution is the same as described in Section 6.4.3 *Data From Dissolved Oxygen Sensors*. All of the values from the conductivity sensors were plotted side-by-side to compare their values in saltwater versus freshwater. This is illustrated in Figure 6.24.

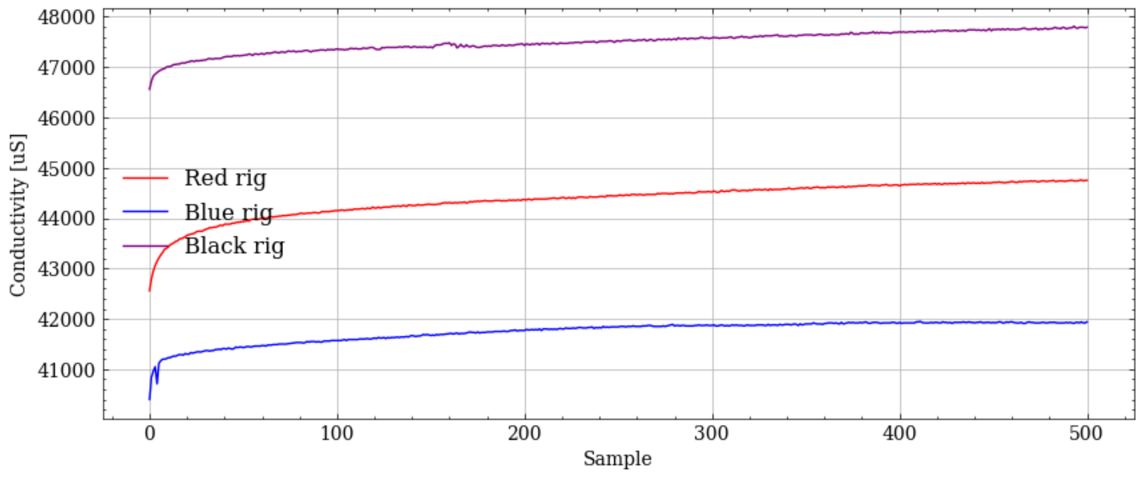


Figure 6.23: Conductivity sensors in saltwater.

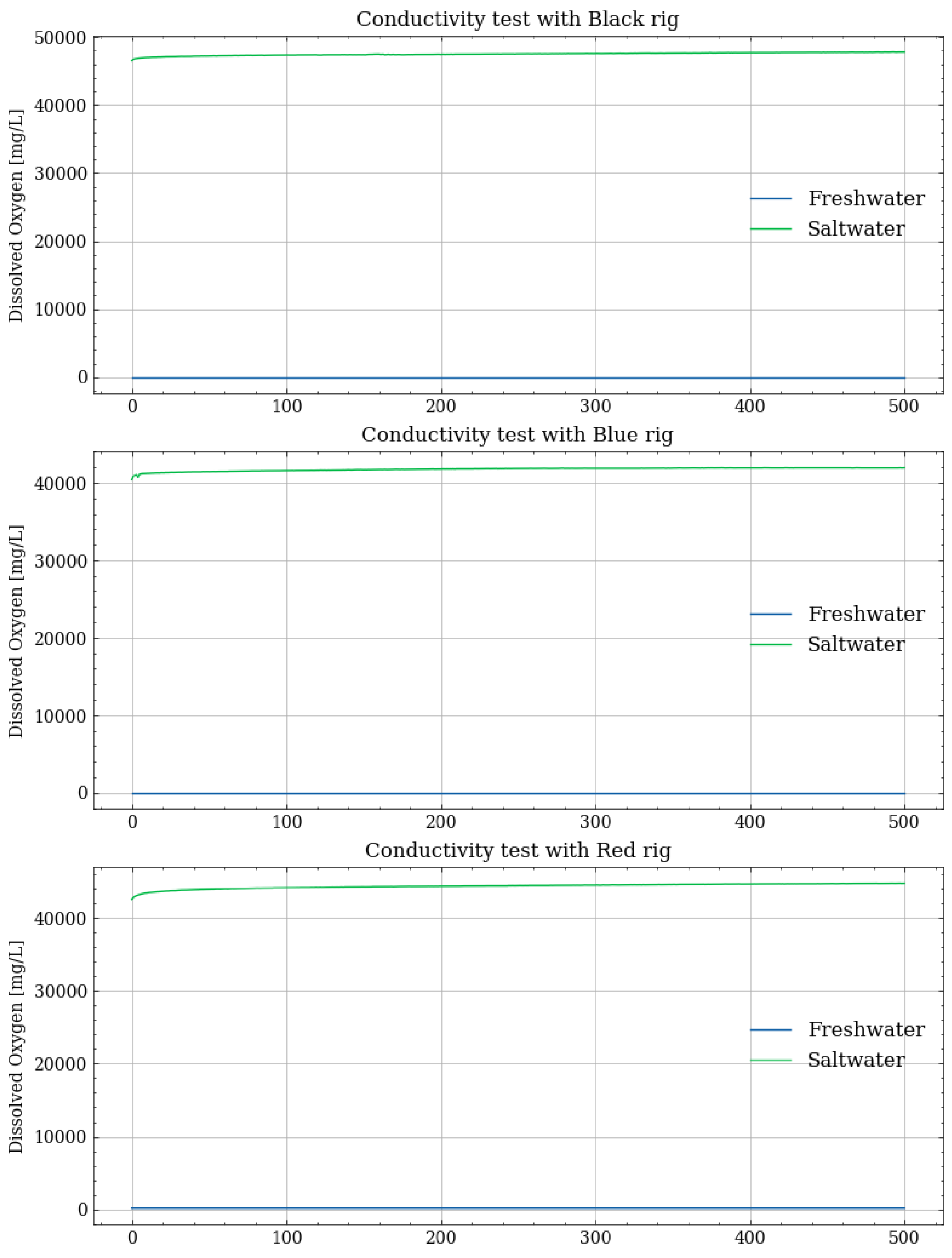


Figure 6.24: Comparison of conductivity in freshwater VS saltwater, T=1.2s.

6.4.5 Data From Pressure Sensors

The data in this section presents the values of the two pressure sensors that were implemented in the two new rigs. They were tested in accordance with the method described in Section 5.5.3 *Functionality Tests of Pressure Sensors*. The data recor-

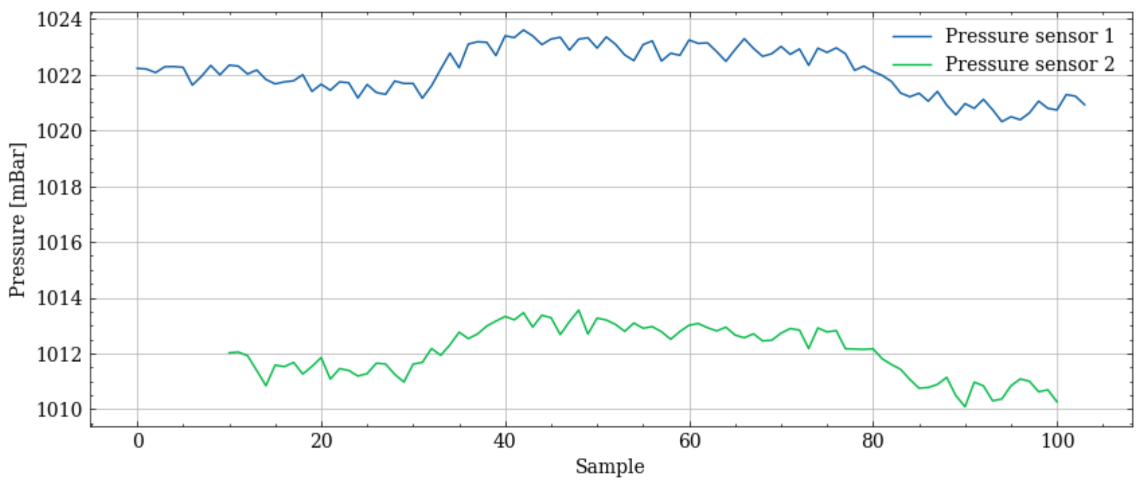


Figure 6.25: Pressure sensor values from taking the elevator down and up again.

ded is illustrated in Figure 6.25. The sample time was set to 1 second and the height difference in meters is unknown, but the elevator traveled 6 floors down, then up again.

6.4.6 Data From Temperature Sensors

The performance of the temperature sensors was evaluated through two tests: the transient test and the precision test described in Section 5.5.4. The results of the transient test are depicted in Figure 6.26, which illustrates the temperature readings recorded by the two sensors. This graph provides insights into the sensors' response to dynamic temperature changes. Additionally, Figure 6.27 showcases the temperature readings obtained during the precision test, offering a visualization of the sensors' accuracy and stability under near static temperature conditions.

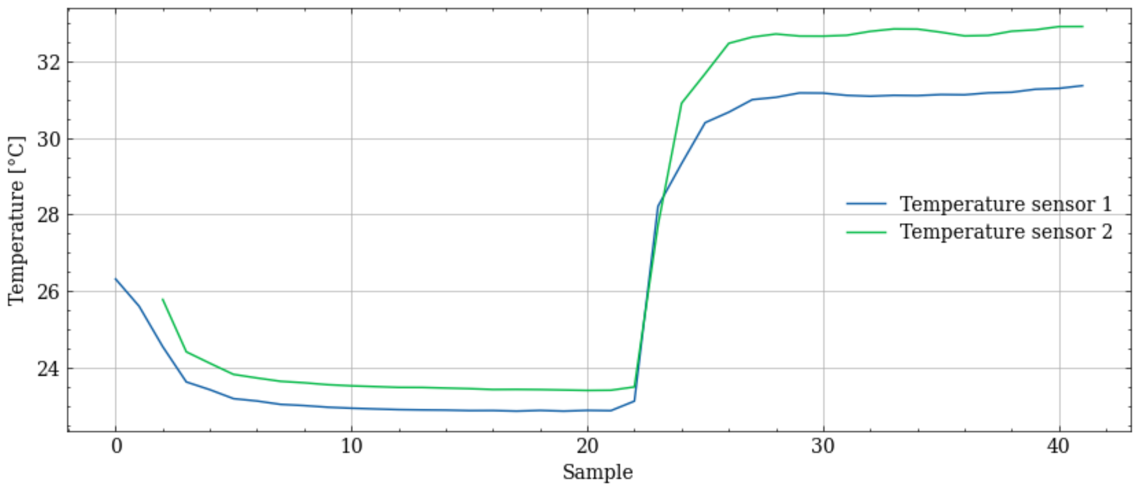


Figure 6.26: Temperature sensor values in water from 23 to 33 °C.

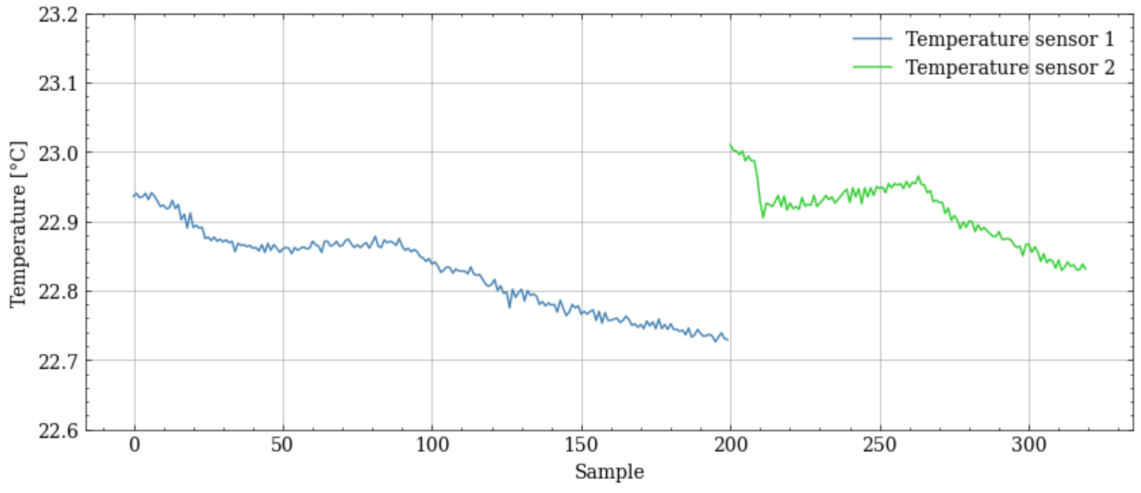


Figure 6.27: Temperature sensor values in water with 23 °C.

6.4.7 Data From Power Sense Module

The data retrieved from testing of the power sense module is presented in Figure 6.28. It compares measurements from the PSM in three different tests against the reference voltage outputted by the PSU.

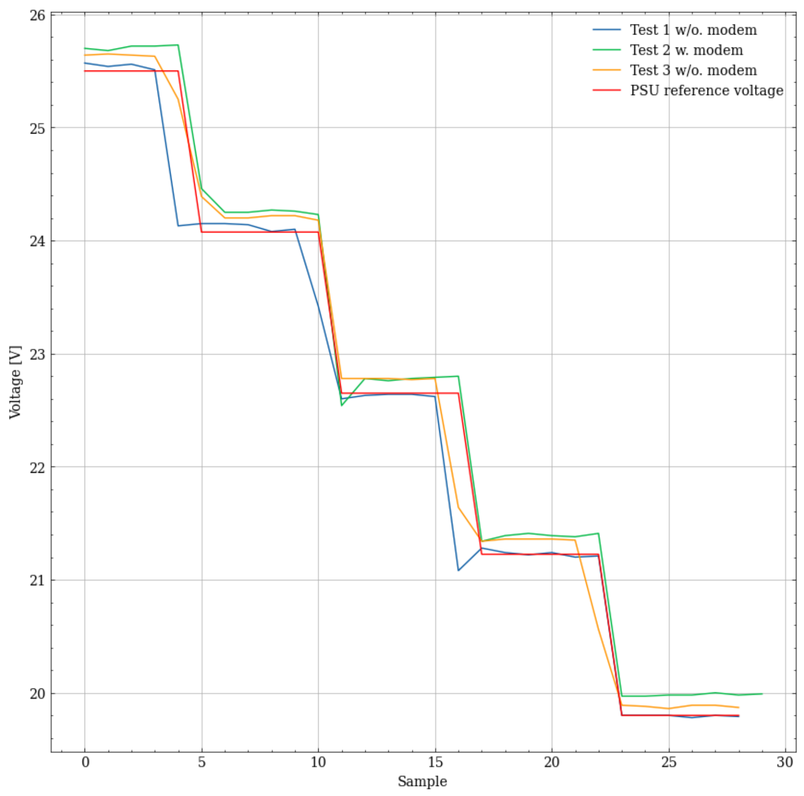


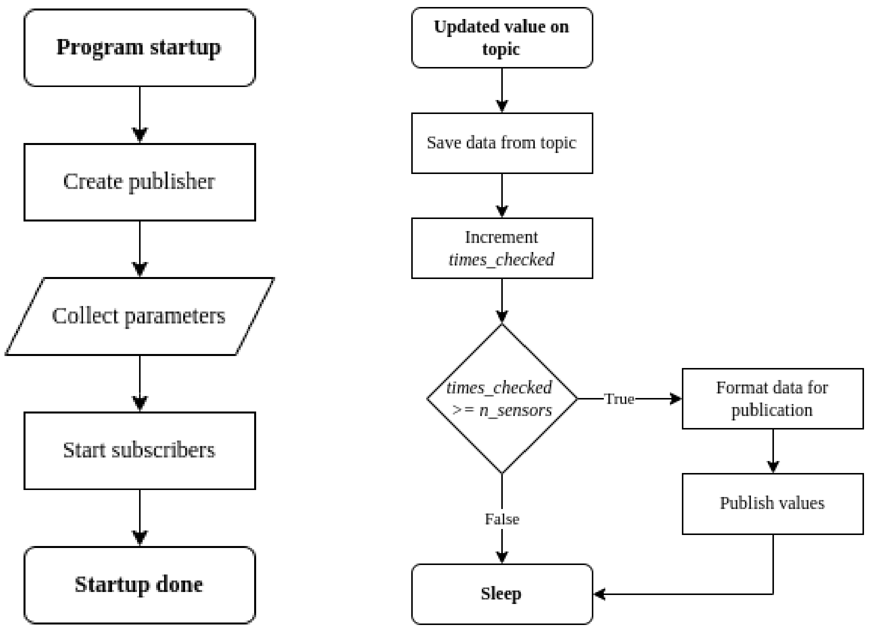
Figure 6.28: PSM voltage readings on Red IES.

6.5 Program Flow and Structure

This section will present flowcharts for the modem nodes for easier code comprehension.

6.5.1 Data Handler

Figure 6.29 shows the structure of the modem data handler node, subfigure 6.29a displays what happens when the node is started. Subfigure 6.29b displays the program flow every time a new value is posted on a sensor topic.

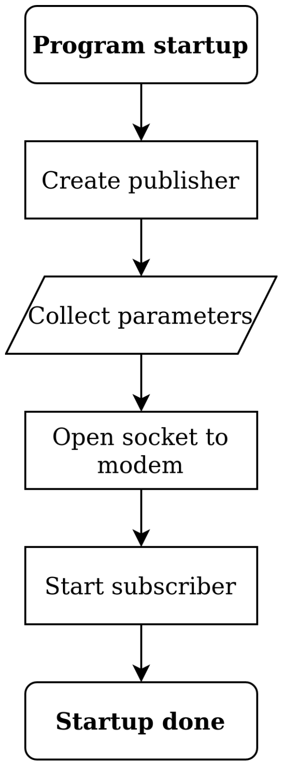


(a) Setup. **(b)** Main.

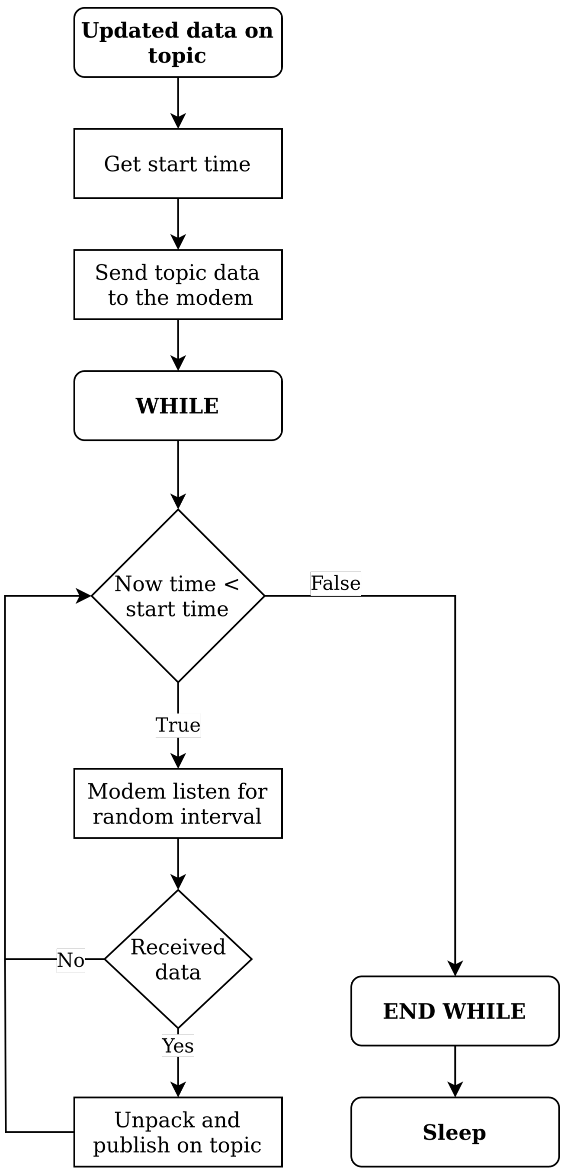
Figure 6.29: Modem data handler program flow.

6.5.2 Data Communicator

Figure 6.30 shows the structure of the modem data communicator node, subfigure 6.30a displays what happens when the node is started. Subfigure 6.30b displays the program flow every time the modem data handler node publishes.



(a) Setup.



(b) Main.

Figure 6.30: Modem data communicator program flow.

6.6 Battery Percentage

There were made illustrations of the relationship between battery voltage and battery percentage. Figure 6.31 shows the linear approximation of the relationship compared to the characteristics of a 6S lipo battery's actual capacity relationship.

Using the given formula:

$$\frac{a \cdot e^{x}}{b \cdot sin(x) + b \cdot cos(x) + d \cdot x^{8}} \tag{6.1}$$

Where $a = 2.32 \cdot 10^{-6}$, $b = 576.39$, $c = -22.52$ and $d = 1.20 \cdot 10^{-8}$, a nonlinear

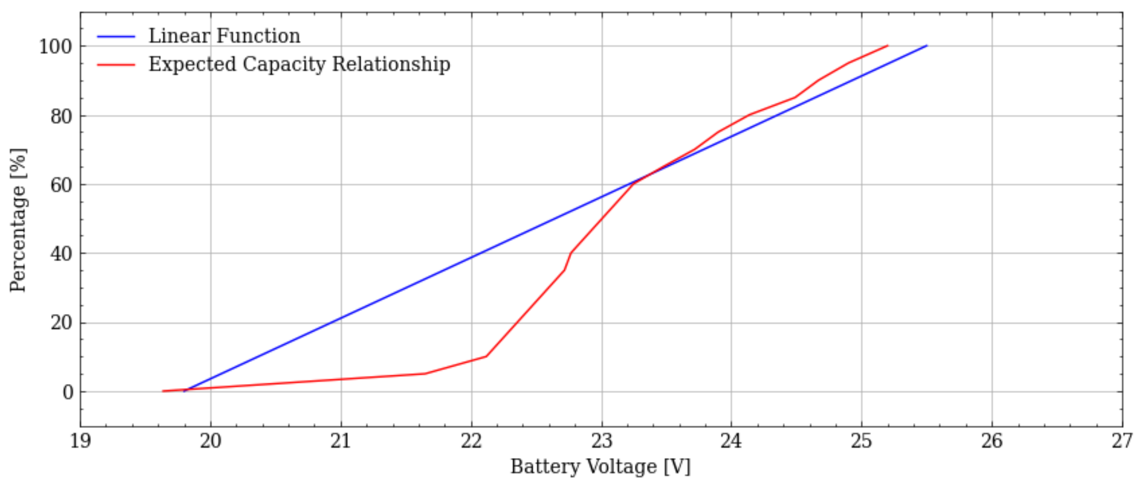


Figure 6.31: Linear approximation of battery capacity in percentage.

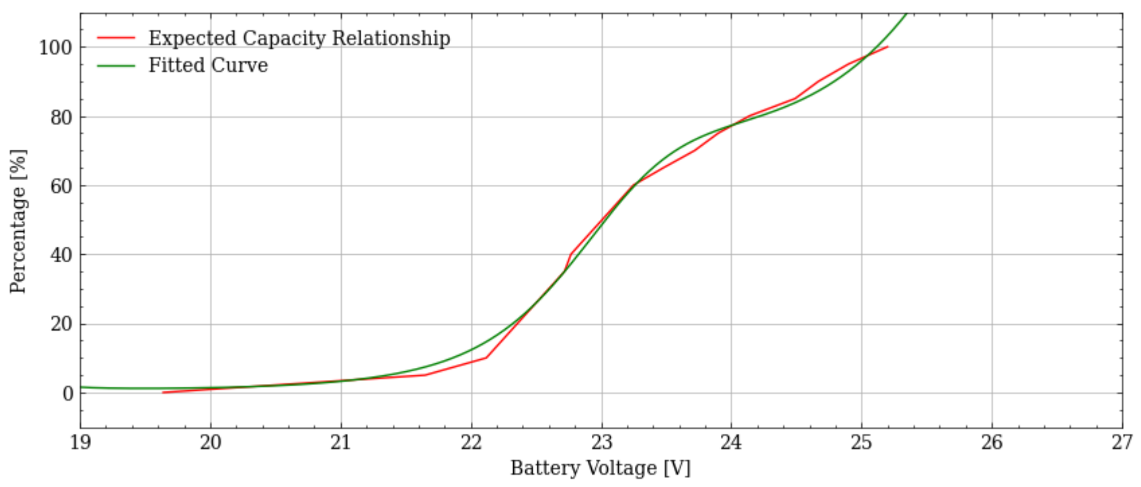


Figure 6.32: Nonlinear approximation of battery capacity in percentage.

regression matching the characteristics of a 6S lipo battery was found. Refer to Figure 6.32.

6.7 The Standalone Underwater Monitoring Station

The Standalone Underwater Monitoring Station consists of three Internal Electronic System (IES), each distinguished by its own color: blue, red, and black. These systems are depicted in Figure 6.33. The blue internal electronic system was developed as part of the previous project group in 2022, while the other two systems were created during the course of this project. However, the Atlas Scientific EZO circuit boards on the black internal electronic system are yet to be connected along with the PSM.

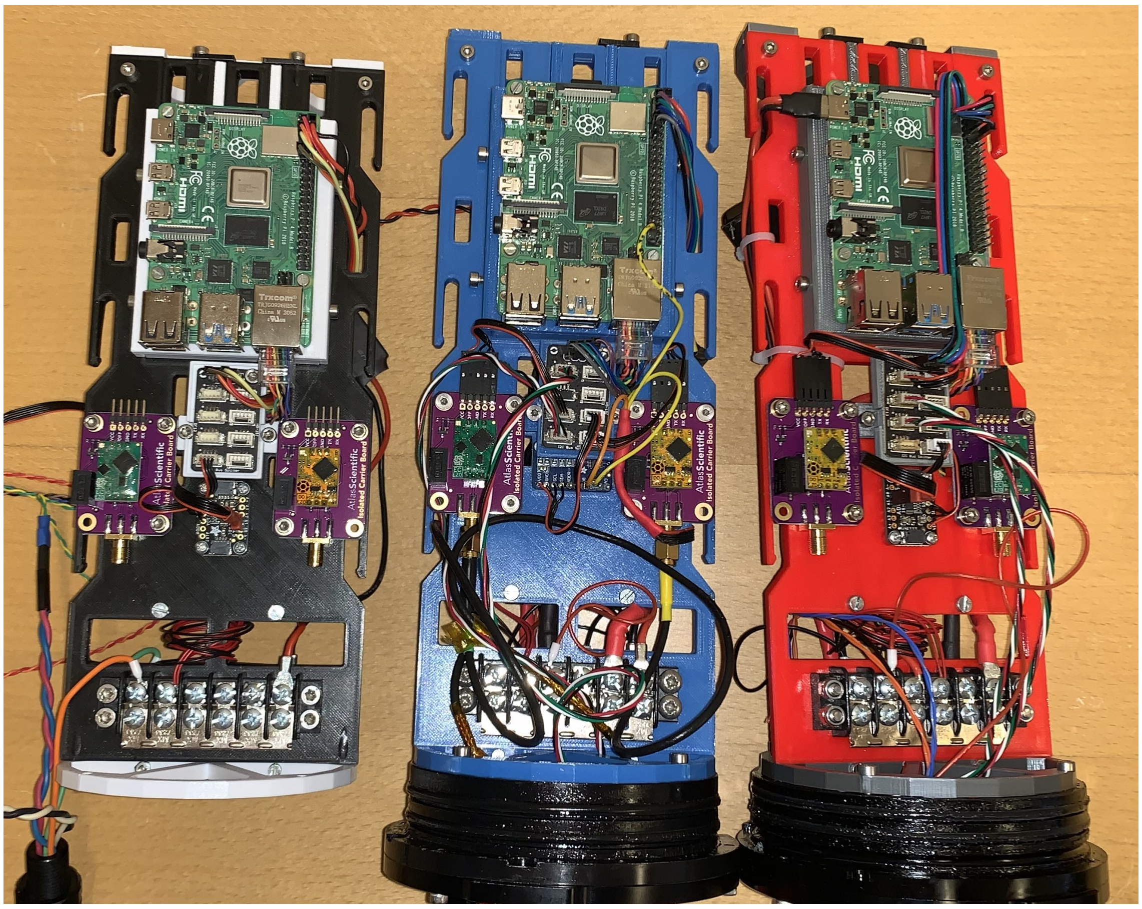


Figure 6.33: All three assembled internal electronic systems.

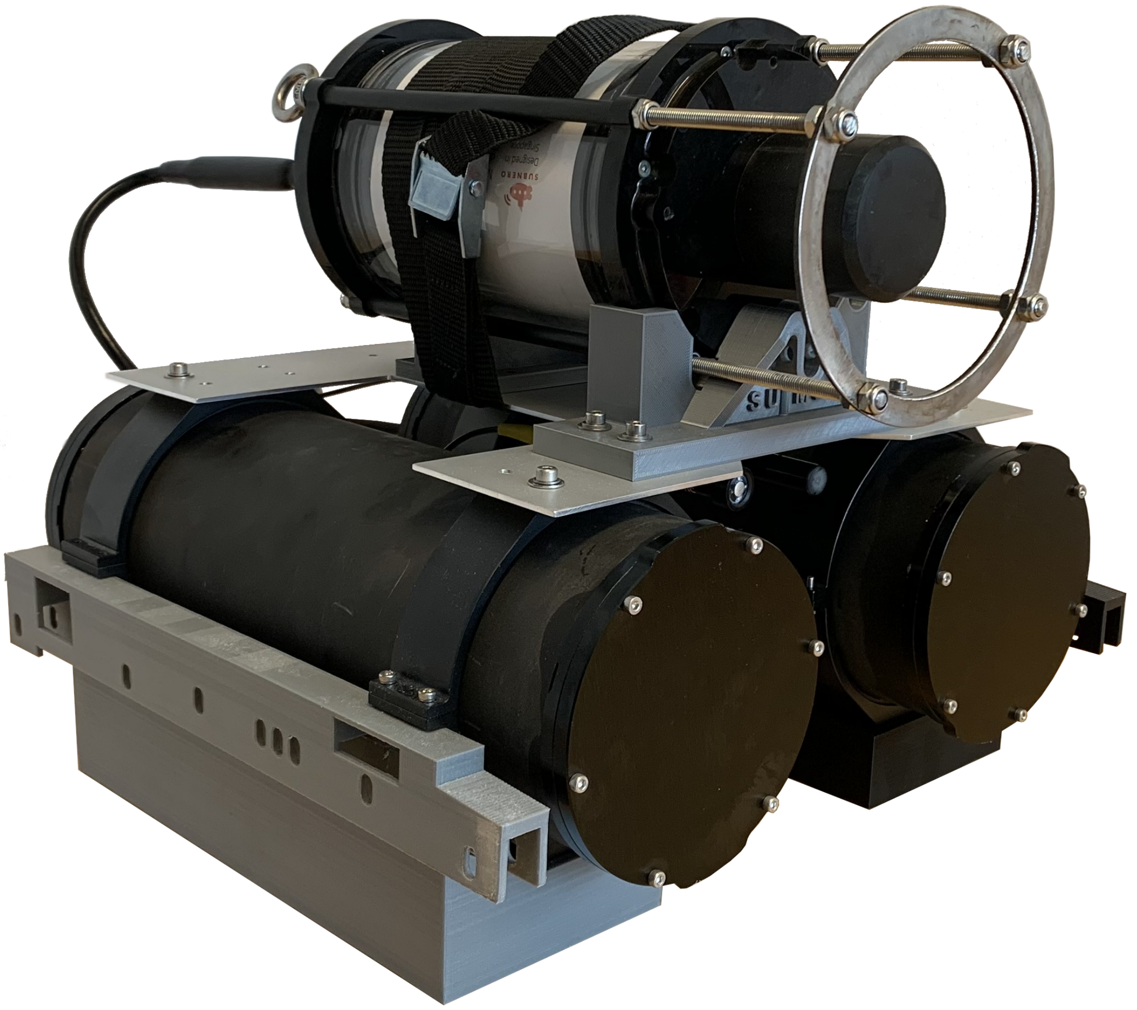


Figure 6.34: Standalone Underwater Monitoring Station fully assembled.

Chapter 7

Discussion

7.1 Structural Design

7.1.1 Issues regarding the main plate

There were several issues regarding the main plate. These issues will be explained in this section and solutions will be discussed. The main issue was found early in the project while the group was making a component list of all the parts needed for SUMS. The problem was that the ADC used in the project was discontinued. The ADC was still for sale, but it was a new design with different dimensions. This meant that it was necessary to redesign the main plate, where the ADC is mounted, and therefore import a model of the main plate into Fusion 360 to be able to redesign. As mentioned earlier, the group did not have access to SketchUp Pro which is where the main plate was designed.

There were found two possible solutions to this. The first solution was to modify the main plate using SketchUp Pro Trial version. The second solution was to import a .stl file of the main plate into Fusion 360 and convert the mesh into a solid. This solution was the best solution, however, not the easiest solution. Since all other parts regarding SUMS construction are imported into Fusion 360's project folder, it would make sense to have such an essential component there as well. There were several attempts at this but unfortunately, all of them failed. Fusion 360's mesh converting algorithm did not work properly. The reason why the conversion failed is still unknown. The conversion did come through, but the result was not a solid object. The main plate had missing surfaces which resulted in a shell structure, rather than a solid object of the main plate.

To be sure it wasn't a one-time problem, numerous attempts were made to execute the conversion process. The process included all the steps described in 5.2.1.3 *Converting meshes to a solid in Fusion 360*, and did not change the result of the main plate's shell-like structure of a failure. No more time was wasted on this problem, and the first solution to redesign the main plate in the SketchUp Pro trial version was the last resort.

There was one last way around the problem regarding the discontinued ADC. This is not a solution to the problem, but more of a workaround. A Norwegian vendor was still selling the old version of the ADC. An email was sent to order a couple of these, but it was later discovered that the order never was placed. The reason for this was that the email got lost in the crowd by the NTNU ordering system, and therefore was never sent out to the vendor. Unfortunately, the vendor had changed their product from the old ADC design to the new design by the time this was discovered.

Additionally, it was identified that the corners and a few edges had warped after printing the two additional main plates required to have three full rigs. This is common if the following factors are at risk according to Ultimaker[21]:

- Uneven environmental temperature
- No brim or raft around the print
- Wrong bed temperature for the material
- Greasy or dusty bed
- Bad fan settings while printing
- Bed is not leveled correctly

Some of these factors may have led to the warping of the main plates. It is unlikely that the room temperature was uneven given the printers are in a closed room with little traffic, and the printer is featured with a front door. The print did have a brim, however, it did not have a raft. This was added in the slicer after the failed print. Cooling is the main cause of warping according to Ultimaker [21]. It is advised to lower the fan speed or to have it off for the first set of layers. Normal fan speed was set to initiate at layer 6 instead of 4. Conclusively, using a raft instead of a brim, and changing the fan settings did indeed help and the printed main plates were less warped than the two first attempts.

7.1.2 Modem Bracket Design and Functionality

Brackets for the modem have proven to work well, provided that the modem is securely attached to the rig when strapped in place. There is minimal swaying and movement. Once the modem is positioned in the brackets and securely strapped, it has no possibility of rotating, shifting, or lifting. However, it does have a tendency to tilt slightly before being fastened. This is due to a small modeling error in the process described in 5.2.1.4 *Modifications to the Model of the Modem.* It was mentioned that determining the diameter of a geometric detail on the physical modem was challenging. This diameter was measured visually using an eye and a ruler, but an exact measurement was not taken. This has resulted in the *semicircle* being slightly smaller than that of the physical modem, causing a tight fit for the modem in these specific areas. Rectifying this issue is a straightforward task by accessing Fusion 360 and increasing the diameter, followed by reprinting the brackets. Despite the snug fit, the modem sits securely, and the brackets function as intended.

A viable solution has been found for designing brackets for the rig. There already exists a digital clone of the entire rig. In the event that a different modem is to be used in the future, a model of that modem can be imported, and the same process used to create these brackets can be applied. The details of this process are explained in 5.2.2.1 *Designing the Front Bracket.*

7.1.3 Evaluation of the Sensor Holder

The sensor holder was designed alongside the modem brackets to secure the Atlas Scientific sensors in place, as they were previously hanging loosely from the cable of the sensor cylinder. The results of the sensor holders can be seen in Figure 6.1.2. They have proven to be effective in maintaining the sensor's position, while also being easy to manufacture and assemble. The requirement for easy installation and removal has been successfully met. The design of the sensor holder was inspired by the Castor EC clip [15].

However, it should be noted that the clips have not been tested extensively in saltwater conditions over a prolonged period. Since reliable sources on the long-term durability of PLA material in saltwater were not found, it remains uncertain whether it is a viable long-term solution. Further investigation into the material's compatibility with prolonged saltwater exposure is recommended.

7.1.4 Modifications of the Aluminium Mounting Plate

The aluminum mounting plate underwent several minor modifications compared to the design used by the previous project group. Two new rigs needed to be constructed, requiring two additional plates for the task. Since no drawings or models of the plate were available, it was necessary to create them from scratch. Measurements were taken, and the plate was sketched accordingly. During the measurement process, it was observed that the plate lacked symmetry. Consequently, the decision was made to rectify this issue when creating a 3D model of the plate using Fusion 360.

The resulting plate is depicted in Figure 6.5. It can be observed that the plate now features multiple holes to accommodate various components, such as the modem brackets. Additionally, there are holes to facilitate the attachment of the sensor holder in two different positions: either further forward or further back. One notable deviation from the previous design is the absence of a cutout at the front of the plate. The group opted not to implement this feature after learning that the indentation was specifically intended to create additional space for EvoLogics' acoustic modems. As these modems were not utilized in this bachelor project, it was deemed unnecessary to retain this particular detail. All 3D models will be made accessible to relevant individuals who may require them, enabling easy access to the aluminum mounting plate design for any necessary modifications.

7.2 Hardware and Cables

7.2.1 Lubing of SubConn Connector

In all cases, SubConn connectors should be lubricated during connection. This requirement is specified in MacArtney's instructions for handling SubConn connectors [46]. It states that only Molykote 44 Medium grease should be used. The amount of grease to be applied before mating depends on whether the connection will be submerged or not. When mating contacts in dry environments, a minimum of 1/10 of the depth of the female contact should be applied with Molykote 44 Medium grease. On the other hand, when mating contacts that will operate underwater, approximately 1/3 of the depth should be filled with grease. After applying the grease, the plugs should be connected once, disconnected, and then checked to ensure that the grease has been evenly distributed before reconnecting them.

The lubrication process for SubConn connectors was not practiced by the group since this information came to light later in the project phase. Individuals experienced with SubConn connectors were contacted, the verdict was that grease had not been used by them either, yet the waterproofing remained effective for several years.

Given the critical nature of the underwater rig and the potential risks associated with disregarding manufacturer recommendations, it is advisable to consider implementing the prescribed lubrication process for SubConn connectors to ensure the highest level of reliability and longevity for your project. Further investigation and evaluation are recommended to determine the most appropriate course of action.

7.2.2 Comparing Heat Shrink Against Rubber Cap with Plug Splicing Method

Two methods were employed for encapsulating the modem cables with MCOM8M plugs. This was due to the group having soldered the cable and plug with slightly excessive conductor length. As a result, the overlap between the rubber cap (OMBMC) and the cable was too short to ensure a secure encapsulation. It is important to note that the overlap between the cable jacket and the mold is what actually ensures the water-tight seal. To address this issue, the group consulted the Applied Underwater Robotics Lab (AUR-Lab), which has expertise in the field and practical experience with such procedures. It was revealed that heat shrink tubing with adhesive lining was commonly used instead of the rubber cap due to its greater flexibility. The heat shrink tubing allows for longer splices as desired lengths can be cut, and it can be shrunk at both ends of the encapsulation, reducing the risk of cable wear during bending.

However, using heat shrink tubing carries the risk of cable deformation as seen in Figure 6.8. Or in the worst case, damage the cable jacket too much rendering it unusable. It is therefore crucial to either use cables that are heat-resistant, such as rubber cables, or apply lower heat settings when shrinking the tubing using a heat gun.

7.2.3 Optimal Waterproof Cable Solution: Utilizing Pre-Made Pigtail Connectors

To achieve optimal results with a waterproof cable for use with the modem, it is recommended to use a pre-made pigtail connector from MacArtney. A pigtail connector is a cable with a pre-attached plug on one end and an unterminated cable ready to be spliced on the other end. These types of cables are easier to splice than MCOM8M connectors, which require greater precision due to the shorter conductor length to work with. Another advantage is that there is no need to splice anything at the unterminated end because the pigtail comes in lengths of 0.6m, which is long enough for the intended use of the cable. Therefore, all that is required is to attach a bulkhead fitting, such as the WetLink 9.5mm penetrators, expand one of the holes on the sensor cylinders end cap to fit M14, and prepare for switch connection and power to the modem.

Epoxy sealing or soldering the cable is not necessary, resulting in both time and cost savings. In addition, the MCIL8M pigtail is currently priced at about half the cost of the MCOM8M connector. The outcome of this method has proven to be easy and functional. The result of the MCIL8M modem cable is shown in 6.9. It is worth mentioning that a vacuum test has not been performed on the cable, however, the only possible error could come from the WetLink bulkhead installed on the other end of the pigtail. This is also a possible source of leakage in the other method using the MCOM8M plug as it also requires installing a WetLink penetrator.

7.2.4 Evaluation of the system's Power Supply Unit

During testing of the IES, the battery was connected to power up the system. However, when the battery was disconnected and reconnected, a minor electrical issue occurred within the PSU, leading to an unexpected event. This event resulted in the PSU experiencing a component failure, causing the transmission of an abnormal current or voltage to the RPi. The damaged component on the PSU is shown in Figure 7.1. As a consequence, the RPi was rendered non-functional, with only a constant red light illuminating upon attempting to power it on. The absence of a green light, which indicates normal operation, signified an error.

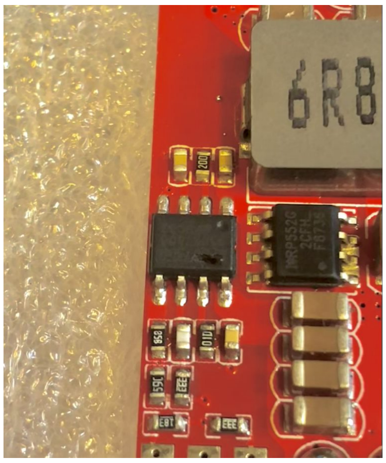


Figure 7.1: A hole in a component on the broken PSU.

This mirrored the behavior observed in the initial RPi received at the beginning of the project, which was also damaged, raising concerns about the PSU's reliability. Consultation with individuals involved in the previous project phase (bachelor 2022) confirmed that a PSU failure had occurred at that time as well.

Considering the recurrent performance issues with the Blue Robotics PSU, it has been decided to remove it from the SUMS system to avoid further unnecessary delays and unexpected issues. It has been replaced with TRACO's THN 15-2411WIR, which is also a DC/DC converter providing a 5 V output and accepting an input range of 9 - 36 V. While TRACO's PSU delivers a maximum of 3 A output compared to BlueRobotics' 6 A, it is considered a more reliable and robust alternative. Considering the current configuration of SUMS, which only requires a power supply to the RPi, it is unlikely to draw more than 3 A.

7.2.5 Extracting Data from a Closed Cylinder

As of the current state, there is no possibility to monitor the program running on the underwater rig while it is submerged. The only way to access the data is by connecting a PC to a modem and listening for the received values. During field tests, controlling test values without opening the cylinder is not feasible, necessitating a new vacuum test before submerging the sensing rig again. This is not an optimal solution for monitoring or debugging purposes, and therefore, several alternative solutions will be discussed. Three potential solutions have been identified.

SubConn Adapter
It is possible to access data from the cylinder without making any modifications to the sensor cylinders as they are currently assembled. This can be achieved by creating an adapter that connects to the modem cable attached to the sensor cylinder. This adapter would enable access to the switch and communication with the RPi using SSH. To accomplish this, a custom adapter can be constructed using MacArtney's SubConn MCIL8F pigtail terminated with an RJ45 connector. This would provide a 100BASE-T or 100BASE-TX connection since the modem cable consists of only two twisted pairs.

The advantage of this solution is its cost-effectiveness. Only a pigtail and an RJ45 connector would be required, without any modifications to the sensing rig. However, this solution does not support high-speed connections. Connecting over long distances is not possible due to the relatively short pigtail length. Additionally, accessing the RPi requires disconnecting the modem from the rig.

SubConn Bulkhead on Sensor Cylinder

To access the program at longer distances without disconnecting the modem, underwater cables provided by Subnero can be utilized. These cables are available in lengths up to 30m and provide both power and Ethernet connectivity, with Ethernet being the relevant interface in this context. To employ these cables, an 8-pin bulkhead female connector would need to be installed on the sensor cylinder, establishing a connection to the switch. With this solution, a 1000BASE-T connection can be achieved, and connection to the RPi or modem is made through the switch.

The advantage of this solution is that it enables full access to the sensing rig without disconnecting the modem, and monitoring the program is possible at distances up to 30m. However, it is expensive and requires modifications to the sensing rig.

Cobalt Bulkhead on Sensor Cylinder

The final solution follows a similar principle to the previously mentioned one. However, it utilizes Cobalt series connectors instead of SubConn. It would be necessary to install a Cobalt 8-pin bulkhead connector on the sensor cylinder, which can be internally connected to the switch or a Fathom X [11][16]. Instead of using Subnero cables, a Fathom tether of varying lengths can be employed, and combined with the Fathom X, supporting distances up to 300m.

The advantage of this solution is its high transfer speed, support for longer cables, and wide range of cable length options. With this solution, it is also possible to monitor the program without any compromises, as opposed to the adapter solution. However, it is more expensive. The Cobalt connector solution appears to be the most suitable choice, offering superior performance and greater flexibility.

7.3 Rig's Performance Under Vacuum Pressure and Submersion

7.3.1 Waterproofing and Pressure Holding Ability of the Enclosed Cylinders

The aim is for the rig to be able to operate between 50 and 200 meters below the water's surface. To withstand pressure down to 200 meters, it must be able to withstand 19.6 bar. The hand pump measures inches of mercury (inHg). The test is performed at 15 inHg, which corresponds to approximately 0.5 bar. The test is not done at 19.6 bar because it is not possible without submerging the cylinders in water. The atmospheric pressure limits testing a pressure difference of about 1 bar, even if we can achieve a complete vacuum in the cylinders. Moreover, it is not necessary to test for such high pressure since the equipment already used for the construction of the rig is classified to withstand pressure above what is desired to expose it to. Therefore, it is only of interest if the cylinders can hold the pressure.

In section 5.2 *Testing the waterproofing of the cylinder* from the bachelor thesis in 2022 [25], a pressure loss of 2 inHg was observed after 30 seconds. It became clear to us that the power cables from the battery cylinder to the sensor cylinder were not tight inside. The inner part of the cable is not sealed, allowing for the passage of air or water through the cable's insulation from one side to the other. In other words, both ends of the cables must be connected to a cylinder to avoid leakage, and both cylinders are therefore tested simultaneously.

The red hand pump used tends to lose pressure if placed too hard on the table after pumping. It may help to place it gently on the table after the desired pressure is achieved. The pressure test was successful and held the cylinder's pressure for just under a day before the test was concluded. All tests conducted after this have been successful, with the exception of one Subnero modem.

7.3.2 Full Scale Test

A full scale test to test how two rigs would behave in a natural environment. The test was planned to be executed in Korsvika where they would be placed on ≈ 1 m depth. The Two rigs would be secured from land with a rope connected both to the rig and the modem. They would then be set to sample sensor data from the environment and exchange this data with each other through the modem. The rigs would then be left there for around half an hour with a sampling time of 10 seconds.

This test would be a final test to assess how all parts of the rig worked together. As all values measured before this had been in controlled environments and all communication between rigs had been done in air over very short distances. It would have provided a good overview of what the rig is capable of.

During the preparation of the test, it was discovered that one of the two modems the group had access to was not watertight. The modem had previously encountered issues regarding its lack of waterproofing. However, it underwent a repair process and was officially declared waterproof one month prior to the scheduled testing. In addition to this, they had been laying in a box at NTNU for most of this time. They were meant to be used for research in Stockholm but due to miscommunication they never left Norway and had been laying in the box for most of the period. The group was not informed of the modem's presence so they were retrieved when the researcher that was going to use them came back from Stockholm. The vacuum test of the modems was therefore seen as a formality as they were assumed to be waterproof. Unfortunately, it turned out that one of the modems was not waterproof, leading to the cancellation of the test.

The modem was first tested with a relative pressure of 15 inHg, after 7 minutes the pressure had decreased to 10 inHg. It was first assumed that there was something wrong with the vacuum pump. The vacuum pump was then tested without problems as described in Section 5.6.1. The modem was then brought to a chief engineer who had been the person dealing with the previous leak. An attempt to repair the modem was done by opening it, inspecting the pressure valve and the seal between the lid and the cylinder. Additional grease was applied before the modem was put back together and a new test was conducted. This time the test was conducted with a relative pressure of 20 inHg. This was done to try to identify where the modem took in air. After a quick inspection, no visible areas were found where air could be entering the modem. However, it was observed that the pressure had dropped by 2 inHg after one and a half minutes, the modem was deemed not waterproof and the test was canceled.

7.4 Evaluation of Sensor Data

This section will discuss the sensors used and the results found in Section 6.4.

7.4.1 Atlas Sensor Calibration

All Atlas Scientific sensors except for one conductivity sensor calibrated last year were calibrated by the group. In addition, the electrolyte in all three dissolved oxygen sensors was changed before they were tested. The calibration was done following the instructions given by Atlas Scientific. The group members have been in contact with the Atlas Scientific support team to ask for help but the support team only referred to the instructions in the data sheet.

Dissolved Oxygen Sensor
In conjunction with the tests verifying that the dissolved oxygen sensor worked a report was made and sent to the Atlas Scientific support team. They said that the sensors looked fine and that the sensors should not be used in development. In their experience, the sensors worked best if they were calibrated and then touched as little as possible after that. This is the exact opposite of what the group have done while testing them, this may have affected the sensors and may be a reason why all dissolved oxygen sensor values look different.

The dissolved oxygen sensor used in this project works best with a flow of 60 ml/min. Putting this sensor on a rig that is attached to a ROV will most likely result in a flow much higher than 60 ml/min. The cheapest solution to this would be to make a casing around the probe that would let in a small amount of water to force the flow around the sensor to be low. Another possibly more expensive solution would be to buy an optical dissolved oxygen sensor. Optical dissolved oxygen sensors are not flow dependent and so the problem would be eliminated.

7.4.2 Verification of the Dissolved Oxygen Sensors

The results from the reconstructed test of the 2022 thesis described in Section 6.4.1 give the same result as what was found in the 2022 thesis. This was expected and the reason the test in Section 3.8.3 was created. The results from this test matched better with the hypothesis that fresh water holds more dissolved oxygen than salt water can. This test was somewhat extreme as the salt concentration was 26.25% and normal sea water has a salt concentration of 3.5%.

The unstable readings are most likely due to the pump creating a high flow, as mentioned in Section 3.8.3. As mentioned the dissolved oxygen sensor operates best with a flow of 60 ml/min and the pump connected to a 9V battery most likely created a much higher flow than this. The data may have been corrupted because the flow in the test environment was too high. Even if the flow was excessively high, the sensor would still adhere to the principle of more saline water containing less dissolved oxygen, implying that the sensor respects the theory of dissolved oxygen in brine solutions.

Another mail was sent to the Atlas Scientific team to inquire about how the sensor is affected by temperature and various other functionalities of the sensor. This mail was however never responded to by their support team.

7.4.3 Pressure Sensor

7.4.3.1 Pressure Sensor Discrepancy

The pressure sensors on the red and black rigs were tested to verify their functionality. Upon analyzing the test results, it was observed that the sensors exhibited different measurement values under identical conditions. There was a 20 mBar difference between the two sensors. The specification of the pressure sensor indicates a relative accuracy of $\pm$ 200 mBar. This means that the observed error was 5% of the absolute relative accuracy between the two sensors [4]. The values discussed are illustrated in Figure 6.25.

Both sensors recorded an approximately 2 mBar pressure difference after the ascent, indicating a constant measurement error between the sensors. Further testing is required to determine whether this is solely an offset or if there is a sensitivity drift. In the case of an offset, the measurement error can be corrected by initializing the sensor to measure 1013 mBar at program startup, assuming most tests are conducted at sea level (0 meters above sea level), as atmospheric pressure at sea level is a known value.

There are two issues with this solution. Firstly, it limits testing to seawater. Secondly, high-pressure and low-pressure conditions can affect the initialization calibration. On the other hand, precise pressure values are not essential for depth measurement; relative pressure differences are typically more relevant.

If sensitivity drift is present, further discussion and exploration of possible solutions or alternative sensors should be considered. If not, the issue can be disregarded, and the sensor can continue to be used.

7.4.3.2 Avoiding Sensor Drift on the Pressure Sensor

The chosen pressure sensor has specific usage considerations to avoid sensitivity drift. The instructions provided by Blue Robotics state that the sensor must be dried for a minimum of 2 hours per day [4]. Consequently, if prolonged measurements spanning more than one day are desired, an alternative sensor should be utilized.

7.4.4 Temperature Sensor

Tests were conducted on the two temperature sensors intended for use with the remaining rigs to be assembled. In this section, the results of these tests and the integrity of the testing process will be discussed.

The transient test aimed to observe the behavior of the sensors when subjected to a significant temperature change. It was observed that the measured values were not identical, with a difference of 0.6 °C in the cold bath and 1.5 °C in the hot bath. This discrepancy may indicate a sensitivity drift in the sensors. The accuracy of the sensors is specified as ± 0.1 °C [9].

As explained in Section 5.5.4, each sensor was tested individually. There is a possibility that the temperature of the water baths changed between the tests, even though the tests were conducted consecutively. Figure 6.26 shows that temperature sensor 2 measured higher values than sensor 1. Since sensor 2 was tested before sensor 1, it is possible that this temporal difference in testing contributed to the observed discrepancy. It is therefore recommended to conduct a similar test where both sensors are tested simultaneously in the same water bath, in order to potentially eliminate this measurement error.

The second test conducted was the precision test, which aimed to determine the accuracy of the sensors. The water temperature had stabilized at room temperature, measuring 23 °C. Prior to initiating the test, the water temperature was measured as 23.5 °C. After the test, the temperature had dropped to 22.8 °C. This temperature variation could be attributed to energy transfer between the measurement instrument, the sensor, and the water, or vice versa.

In Figure 6.27, the sensors are plotted consecutively rather than overlapping. This is done because the tests were performed consecutively in this order. In theory, there should have been a seamless transition between sensor 1 and 2, but this was not observed. There is a clear temperature jump between the measurements, which, as discussed above, may be due to energy transfer.

The temperature measurement instrument used for control measurements of the water temperature reported a temperature of 23.3 °C after temperature sensor 1 was removed from the water bath. Sensor 1 measured a temperature of 22.7 °C. Eight minutes later, temperature sensor 2 was placed in the water bath, measuring 23.0 °C. It is difficult to definitively determine whether the water temperature dropped by 0.3 °C between the tests. It is not impossible considering the gap between the two measurements, during which the time between the tests and the energy from sensor 2 could have affected the water.

For further testing, it is again recommended to conduct the tests on both sensors simultaneously to exclude factors related to energy transfer. The tests should also be performed in an insulated container to ensure a stable water temperature. Additionally, conducting the tests over an extended period with an independent high-accuracy measurement instrument is advised.

7.4.5 PSM

The PSM has been a problematic sensor through both iterations of this project. As mentioned in section 5.1.1.2 the PSM will output 22 V if not modified. The technical details listed on their website say that it should output a voltage of maximum 3.3 V which is incorrect in the group's experience. The PSM is also made for working with the Pixhawk Flight controller [35] but is supposed to work with a ADC as done in this project.

7.4.5.1 Current Sensing

Several attempts to sense current with the sensor were attempted but to no avail. The problem is that the current reading is near constant even as the voltage is changing. A different PSM [20] was tested to see if the problem was specific to the BlueRobotics PSM. Both PSMs gave the same results. The focus was then to see if the problem could lay within the ADC. The current signal was connected to the pin that originally decoded the voltage signal but the current still showed near-constant.

Due to voltage readings being the most interesting value the current sensing was "abandoned". All codes for measuring, calculating, and posting current values are still left in the ROS node. The current readings are not part of the data sent via the modem as it seemed excessive to send useless data.

7.4.5.2 Voltage Sensing

To verify the functionality of the hardware and software, a test was conducted using the Power Sense Module on the red rig. During the initial measurement, a constant offset of 0.2 V was observed, which was corrected in the code by adding 0.2 V to the measurements. This adjustment proved effective in the subsequent test, as evidenced by the measurements labeled as "Test 1 w/o. modem" in Figure 6.28. The measurements closely followed the reference values with millivolt precision.

A subsequent test was conducted with the modem connected to examine any potential influence on the results. As shown in the graph, Test 2 exhibited a new constant offset of 0.2 V above the reference. The modem was then disconnected, and the test was repeated. Test 3 yielded results similar to Test 2, albeit slightly lower on average.

The obtained results were unexpected and raised concerns within the group. Despite properly calibrating the PSM in Test 1, it reverted to incorrect measurements even though the code remained unchanged and the applied voltage was consistent. The role of the modem in this discrepancy remains uncertain since the measurements remained inaccurate even after disconnecting the modem. Further experimentation and additional tests are required to elucidate the factors causing variations in the measurements.

7.5 Software

This section discusses the thought process that was used while building the software for the program. Explanations as to why implementations are done the way they are will be given. Furthermore flaws and improvements will be discussed.

7.5.1 Measurement Units

In previous iteration of this project the temperature sensor and pressure sensor both included code for gathering and publishing measurements in both imperial and SI units. The measurement is only collected one time in SI units but a function in the sensor library is used to convert the values and both values are published to the topic. The *modem_ data_ handler* node contains functionalities for transmitting the Imperial units but these values are not used or passed any further. They can easily be transmitted with minor changes to the modem data handler node.

7.5.2 Low Power Sleep Function

There was a wish to implement functionality to shut down the program and set the modem to sleep when power levels were critically low. To avoid damaging the battery the data sheet specifies to not let the voltage of a single cell be lower than 3.3 V [48] since the battery used in this project is a 6S battery the battery voltage should not be less than 19.8 V. Without any functionalities for setting the modem or program in sleep mode there is a risk of draining the battery. It is therefore advised to monitor the data sent from the modem to keep an eye on the power levels. The modem is the most power-hungry component and therefore an attempt to put it to sleep was made. A sample code is provided in the SUMS-Tools repository explained in Section 7.5.9 *SUMS-Tools* have a feature for setting a sleep schedule for the modem. The problem with this is that the sleep schedule is remembered between boots and can only be turned off manually via the Subnero browser interface.

The sleep function was tried implemented at a later stage of the project, therefore with limited time and limited access to the modem due to it being used for research in Stockholm, the whole concept was abandoned. For further implementation, a solution like a relay controlled by the RPi to switch the modem on and of could be an easy solution to circumvent the sleep schedule.

7.5.3 Timekeeping

As the Raspberry pi does not have any method for keeping time between boots all timestamps collected from the system's local time will most likely be incorrect. This is especially important to remember when looking at the timestamps for transmitted and received data. As the logger files and their location also relies on the system local time they may also show a different date and time.

By itself, incorrect timekeeping is not a big problem as if the user knows when the program was started a simple offset can be utilized to get globally correct time. Locating log files is also easy as the clock starts on every boot and thus to find the latest log the user simply has to locate the newest log file in the newest folder. When introducing other agents in the underwater network and sharing data with each other the problem becomes more difficult as the timestamps may vary vastly between agents and pinpointing when the measurements were collected will be more difficult.

The easiest solution for keeping relatively accurate time between power losses apart from setting the time manually would be to include a Real Time Clock (RTC). A RTC is a device that would be connected to the RPi through I^2C. The RTC is powered by a battery and thus it is able to keep time even when the RPi is powered off. The RTC would be able to keep relatively accurate time with RTCs like the DS3231 boasting an accuracy of $\pm$ 2 minutes over a year.

7.5.4 Sensors

All sensors include code for detecting if the sensor cannot be initialized. Something to watch out for with this is that it will not work if the RPi cannot communicate with the sensor. In a scenario where the I^2C connection is bad or broken, the program will not be able to instantiate the sensor object from the library. The result is that the program will crash before the if statement can be evaluated and no error will be logged. An error message will however be printed to the terminal and it will say that the crash happened somewhere in the library. This error cannot be fixed as it would require changing the library of all sensors and would be too big of a task for the scope of this project.

PSM

In order to retrieve battery measurements from the analog signal of the Power Sense Module, code had to be implemented to fetch this signal from the Analog Digital Converter. For this purpose, Adafruit's own library designed specifically for this Analog Digital Converter was utilized. However, the code provided by the previous project group did not function correctly, despite using the correct library. The error was traced to the creation of the sensor object using the wrong class. The correct class name is *ADS1115()*.

A method was introduced to convert the battery voltage into battery percentage. This method is presented in Section 4.7 and employs a linear approximation of the relationship between battery voltage and battery capacity in percentage. This linear approximation is illustrated in Figure 6.31. In the figure, the linear function is compared with data that provides a better approximation of the voltage-to-percentage relationship. The voltage of a LiPo battery exhibits nonlinearity with respect to capacity as it approaches full charge or nears depletion. Therefore, an alternative solution was sought, although it has not yet been implemented in the program. It involves a nonlinear approximation of the data found on Ampow [31], as shown in Figure 6.32. The function was determined using the *curve_fit* function from the *scipy* library, which takes the data and an optional function as input. The function yielding the best result is represented by Equation 6.1.

7.5.5 Modem Data Handler

The modem data handler is the node responsible for determining when all sensor values have been updated. The implementation standing ensures that all sensor values are updated before the data is formatted and published for logging and transmitting. It was also made to work even if a sensor stops working, albeit sending less frequently. As the code stands now the variable *n_ sensors* in the launch file needs to be accurate to ensure that sending of data happens every *sample_ time* seconds.

Although the method used now is not perfect it was seen as more favorable than other methods. Some of the other methods considered were to put the publish function in one sensor callback function. The problem with this is that if that particular sensor stops working, both transmitting and logging functions would also stop working as no new messages would be posted on the *internal_ data* topic.

A version where data would be formatted and published every *sample_ time* seconds was also considered. The problem with a solution like this would be that the delay from sensing a value to publishing said value on the topic would be unknown and long. In an implementation like this, there would be a risk of publishing on the topic right before new sensor data is collected. Making the receiving agent wait for *sample_ time* seconds longer than the current solution.

A functionality that could improve the existing architecture could be to register what sensor is not working and account for it by adjusting *n_ sensors* to match the current number of working sensors.

Simplifying Data Selection for Transmission

Another problem with the implemented solution is that it is difficult to change what data that is going to be sent. The implementation of the *log_ header* parameter in the launch file was an attempt to simplify this process. This header parameter only applies to the logger nodes and thus it has to be synchronized with what is actually posted on the topic. Python dictionaries were also used to increase the ease of modifying what values to send, though this still has to be edited from the actual node.

A possible solution could be to have a parameter pass an array that specifies what sensor values are desired to send. This was not implemented as all valuable data that is sensed is already sent and sending other data from a possible new sensor would require modifications anyways. This solution was also thought of late in the project and thus there was no time to implement it.

7.5.6 Modem Data Communicator

Due to the modem being unavailable for much of the last part of the project the development of the sending and receiving structure was hindered. The group had access to a modem simulator developed by UnetStack, but this simulator does not behave exactly like the modem and therefore is not suitable for developing algorithms for sending and receiving. The first version of the sending and receiving structure was developed the night before the modems left for Stockholm. Further development was then done with the simulator, the problem with this was that the simulator did not behave exactly like the modem and things that worked on the modem did not work on the simulator. As a result of this, the modem data communicator node was finalized once the modem "returned" from Stockholm.

There is a small computer inside the modem that handles the bulk of the sending and receiving logic, it is this computer that enables the use of *unetpy* and easy communication with the modem. This computer also handles cases where the modem is asked to send data faster than what it is physically capable of. In a case like this, the modem has a scheduler that keeps track of what is sent and what needs to be sent. While this scheduler is occupied with sending data the task of setting the modem in receive mode is less prioritized and vice versa. An effort was made to avoid stacking the scheduler with tasks as this would lead to sending and receiving of data being unpredictable. The solution to this was to compensate for the time it took the modem to send data. One problem with this however is that the time it takes the modem to send data varies based on the amount of data that is sent. Due to lack of time, this was not explored further. The sending and receiving works but there may be improvements to be found in tweaking the *transfer_ delay* variable found in the launch file. The bounds for the receive timer were chosen arbitrarily and optimizing with regards to these variables may be worthwhile.

Testing the Communication Algorithm
Several tests of the communication was done in air but due to the full scale test discussed in Section 7.3.2 *Full Scale Test* falling through no test was performed in

water. The group have consulted with specialists in the field and as the sending works in air and the fact that it is developed as a library of UnetStack it should also work in water. These tests was done in a more informal manner than other tests presented in the *Results* section and so they and their results are presented and discussed here.

The tests performed was done with the modems transducer 3cm apart and both rigs were set to sample every 30 seconds. This test showed a transmission loss of 18.3% and 19.6%. This is quite high but considering that there were only 2 nodes in the network. The rigs are set to broadcast to all agents and one could argue that the real world loss in a setting where there are more than two agents the transmission loss would be lower. In the case where no transmission can be lost a implementation where the same data can be transmitted multiple times can be implemented. This would increase the odds of the other agent listening when the data is transmitted and so lower the transmission loss. Another way is to tweak parameters from the launch file as mentioned above.

A second test was done to simulate how the rig would behave in a setting where there is four or more agents. This was done by setting one of the agents sample time to be 7 seconds while the other agent sampled every 30 seconds. This test confirmed that the algorithm for listening for more than message per sample was possible. This feature was not tested before access to the modem was lost, so the feature was tested extensively with the simulator. The problem with this was that the feature did not work in the simulator. The problem was that when the *modem_listen()* was inside a while loop the simulator would not send data. For using the simulator the fix was to change the *wile loop* to a *if statement*. This test was conducted after the modems was inaccessible and so there was not much time for improvements.

7.5.7 Launch File

The launch file is a crucial part of the program, it has also increased the usability of the program by gathering all parameters to every node in one place. When paired with the launch command in ROS2 it runs all nodes instead of having to run each node separately and pass the correct parameters. It is also a core part of running the program upon startup of the RPi as described in Section 4.5.3. It is important that the end user understands what this file contains and the importance of setting the correct parameters. If for example the IP address of the modem is not set in the launch file the whole communication will fail as the modem data communicator node cannot establish a connection to the modem. Other parameters as sample time and how many sensors the program is utilizing are also set here.

7.5.8 Loggers

The two loggers log data from both ends of the communication. And thereby no log of the sensor topics is created. One could argue that the sensors are logged when their value is repackaged by the modem data handler node. This still excludes the specific time each sensor value was collected. ROS2 includes functionalities for logging messages with the *rclpy.node.get_ logger()*. This capability was discovered by the group very late in the project and as the most valuable data was already logged it was not implemented. All data printed to the terminal in this program is done with *rclpy.node.get_ logger()* and thus finding the command to log this to a file will eliminate this problem.

External Logger Node
The external logger as it stands now logs all received data to the same file. This means that to analyze data from an agent from the external log file the data have to be filtered. It is generally easier to use the internal logs if the sensor data is to be analyzed.

When deciding what type of file to write the logged data to, the choice naturally fell on *CSV* file. This is the file format that the members of the group were most familiar with and it seemed easiest for data analysis with both Python pandas and Microsoft Excel.

7.5.9 SUMS-Tools

In addition to the main program a repository with tools was created. This repository consists of some files that may help for future development. It also includes a detailed *README.md* file explaining all code that is included. In addition to the files that can be used for further development, there is a simulator for running the SUMS program on computers without I^2C capabilities. A simulator developed by UnetStack is also linked. This UnetStack simulator is capable of simulating modem communication when the modems are unavailable. The repository can be found on GitHub[43].

7.6 Details for Further Work

For future work, it is recommended to explore solutions to ensure accurate timekeeping during power losses, thereby eliminating uncertainties associated with program-generated timestamps. Implementing an automatic shutdown or sleep mechanism for both the program and the modem when the battery voltage drops too low would make the sensing rig fully standalone, providing assurance that the battery will not be completely depleted or damaged. Additionally, enabling data extraction from the Raspberry pi without the need to open the watertight enclosure would enhance usability for the user. Creating a comprehensive Table of Screws that covers not only the Internal Electronic System would be beneficial. Investigating nonlinear approaches for estimating battery capacity could also be explored. Lastly, fine-tuning the communication algorithm may reduce data loss during acoustic transmission.

Conclusion

The further development of the Standalone Underwater Monitoring Station has reached significant milestones, bringing this project to a stage where it can be deployed and utilized. The rig has successfully read sensor values with the exception of current sensing from the Power Sense Module. Through the utilization of ROS topics and data handling, the collected data is further processed, logged, and transmitted to the Subnero acoustic modem node. Establishing communication between multiple modems has been accomplished. The rig effectively processes data and transmits it via the Subnero modem, while also demonstrating the capability to receive messages from external modems in air.

Through extensive testing of each sensor, it has been concluded that further examination is required to ascertain their functionality accurately. Notably, the sensors from Atlas Scientific have shown considerable inconsistencies even under identical conditions, which raises concerns. On the other hand, the sensors from Blue Robotics appear to perform as expected, although improvements in testing methodologies are necessary to minimize measurement errors and determine precision and repeatability. These findings underscore the importance of continued evaluation and refinement of the testing procedures to ensure reliable and precise data acquisition from the sensors.

The design of the rig prioritizes functionality, reliability, and simplicity. Its adherence to waterproof standards through vacuum testing procedures, and custom-made waterproof cables contribute to a reliable and efficient final product. Hardware improvements, including the integration of a new ADC, PSU, practical adapters, cables, and an acoustic modem, enhance the rig's performance. Additionally, to optimize the effectiveness of the software aspects of the rig, the task to minimize stress on the Raspberry Pi was facilitated by the reorganization of the entire system environment. By utilizing Ubuntu Server, optimal performance and stability are secured for future development.

The project's mass production aspects have yielded notable outcomes. While two rigs are ready for submersion, minor component additions are needed to finalize the last one. Two fully operational rigs, excluding the potting of the Atlas Scientific sensors, have been successfully produced. A comprehensive bill of materials, along with all necessary 3D models and a digital clone of the rig, ensure the reproducibility of the design. Detailed instructions on the current configuration of the rigs and an assembly guide further facilitate the utilization and scalability of the project.

In summary, the progress made in the development and finalization of the underwater rig showcases the team's dedication and commitment to achieving a reliable and efficient monitoring system. The accomplishments thus far provide a strong foundation for utilization, as well as opportunities for continued refinement and expansion in the field of underwater technology.

The rig has been finalized in the sense that: - it reads all sensor values except current sensing values. - it publishes the data on ROS topics - Two rigs are ready for submersion with the last one lacking some minor components

The communication between several modems: - processes the data and sends it via the Subnero modem - the rig is able to listen and receive messages sent from an external modem. - Have not been tested underwater

The rig has been designed in a way that the final product: - is easy to assemble - has a neat design - is tested to be waterproof - have been further improved hardware-wise (facilitated new adc, new psu, and new waterproof cables)

The mass production aspects of the project have given: - two fully operational rigs (disregarding the atlas scientific sensor) - bill of materials for entire rig - further instruction on rigs current configuration - 3d models and digital clone of the rig - assembly guide

Bibliography

[1] *38770-0106 Molex | Mouser*. en-no. URL: https://no.mouser.com/ProductDetail/538-38770-0106 (visited on 4th May 2023).

[2] *5V 6A Power Supply*. en-US. URL: https://bluerobotics.com/store/comm-control-power/control/bec-5v6a-r1/ (visited on 2nd May 2023).

[3] *AtlasScientific/Raspberry-Pi-sample-code*. URL: https://github.com/AtlasScientific/Raspberry-Pi-sample-code/tree/master (visited on 9th May 2023).

[4] *Bar30 High-Resolution 300m Depth/Pressure Sensor*. en-US. URL: https://bluerobotics.com/store/sensors-sonars-cameras/sensors/bar30-sensor-r1/ (visited on 19th Apr. 2023).

[5] BlueRobotics. *BlueROV2*. URL: https://bluerobotics.com/store/rov/bluerov2/ (visited on 18th Apr. 2023).

[6] BlueRobotics. *How to Choose a WetLink Penetrator*. Nov. 2022. URL: https://bluerobotics.com/learn/how-to-choose-a-wetlink-penetrator/ (visited on 9th June 2023).

[7] BlueRobotics. *WetLink Penetrator Assembly Guide*. Apr. 2023. URL: https://bluerobotics.com/learn/wetlink-penetrator-installation-guide/#introduction (visited on 9th June 2023).

[8] Christian Cawley. *Ubuntu Desktop vs. Ubuntu Server: What's the Difference?* en. Section: Linux. Dec. 2019. URL: https://www.makeuseof.com/tag/difference-ubuntu-desktop-ubuntu-server/ (visited on 18th Apr. 2023).

[9] *Celsius Fast-Response, ±0.1°C Temperature Sensor (I2C)*. en-US. URL: https://bluerobotics.com/store/sensors-sonars-cameras/sensors/celsius-sensor-r1/ (visited on 19th Apr. 2023).

[10] Lars Christensen. *Fusion 360 for Absolute Beginners*. Dec. 2016. URL: https://www.youtube.com/playlist?list=PL40d7srwyc_Ow4aaOGXlP2idPGwD7ruKg (visited on 1st Mar. 2023).

[11] *Cobalt Series Underwater Connectors and Cables.* Rev 2.6. Blue Trail Engineering. Mar. 2023. URL: https://e4c7c5dc-3be9-4c5a-b172-8072b57438af.usrfiles.com/ugd/e4c7c5_be15154513ab488fb0f04c2260100f55.pdf (visited on 27th May 2023).

[12] *Conductivity K 1.0 Kit.* en-US. URL: https://atlas-scientific.com/kits/conductivity-k-1-0-kit/ (visited on 25th Apr. 2023).

[13] *Dissolved Oxygen.* en-US. URL: https://www.fondriest.com/environmental-measurements/parameters/water-quality/dissolved-oxygen/ (visited on 21st May 2023).

[14] *Dissolved Oxygen Kit.* en-US. URL: https://atlas-scientific.com/kits/dissolved-oxygen-kit/ (visited on 25th Apr. 2023).

[15] *EC piping clips.* en. URL: https://castoras.no/produkt/?lang=en&id=47 (visited on 27th May 2023).

[16] *Fathom-X Tether Interface Board.* en. URL: https://bluerobotics.com/store/comm-control-power/tether-interface/fathom-x-tether-interface-board-set-copy/ (visited on 27th May 2023).

[17] *Get Started in Fusion 360 - Create your first project.* en. URL: https://help.autodesk.com/view/fusion360/ENU/?guid=GUID-BB9917D0-2FC7-4750-BBA6-2A9564B08FE0 (visited on 5th May 2023).

[18] *GigaBlox – Small GigaBit Switch.* en. URL: https://botblox.io/products/gigablox-small-gigabit-switch (visited on 3rd May 2023).

[19] Vivek Gite. *What does sudo apt-get update command do on Ubuntu/Debian?* en-US. June 2007. URL: https://www.cyberciti.biz/faq/what-does-sudo-apt-get-update-command-do-on-ubuntu-debian/ (visited on 20th May 2023).

[20] *Holybro Pixhawk PM02 V3 12S Power Module.* en. URL: https://www.getfpv.com/holybro-pixhawk-pm02-v3-12s-power-module.html (visited on 24th May 2023).

[21] *How to fix warping.* en. Nov. 2022. URL: https://support.makerbot.com/s/article/1667337577679 (visited on 5th May 2023).

[22] *How to install Ubuntu Server on your Raspberry Pi.* en. URL: https://ubuntu.com/tutorials/how-to-install-ubuntu-on-your-raspberry-pi (visited on 5th May 2023).

[23] *How To Test Dissolved Oxygen In Water.* en-US. Sept. 2021. URL: https://atlas-scientific.com/blog/how-to-test-dissolved-oxygen-in-water/ (visited on 21st May 2023).

[24] *I^2C*. en. Page Version ID: 1153206437. May 2023. URL: https://en.wikipedia.org/w/index.php?title=I%5C%C2%5C%B2C&oldid=1153206437 (visited on 28th Apr. 2023).

[25] Julie Klingenberg Ida Frøhoel Jonas G Johnsen. 'Developing an underwater sensing rig'. Bachelor's thesis. Norwegian University of Science and Technology, 2022. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3004828 (visited on 18th Apr. 2023).

[26] Adafruit Industries. *ADS1115 16-Bit ADC - 4 Channel with Programmable Gain Amplifier*. en-US. URL: https://www.adafruit.com/product/1085 (visited on 2nd May 2023).

[27] Adafruit Industries. *Submersible 3V DC Water Pump with 1 Meter Wire - Horizontal Type*. en-US. URL: https://www.adafruit.com/product/4546 (visited on 21st May 2023).

[28] *IP adress*. en-US. URL: https://en.wikipedia.org/wiki/IP_address (visited on 18th Apr. 2023).

[29] JoJoHTM. *Sensors_drivers_BROV2*. URL: https://github.com/JoJoHTM/Sensors_drivers_BROV2 (visited on 18th Apr. 2023).

[30] Sivert B. Knudsen. *SUMS - Instruction videos - YouTube*. URL: https://www.youtube.com/playlist?list=PL3A_L7zbhrrv5CL7Yg0aM-x3OEBS8wmF4 (visited on 30th May 2023).

[31] *Lipo Voltage Chart: Show the Relationship of Voltage and Capacity*. Sept. 2018. URL: https://blog.ampow.com/lipo-voltage-chart/ (visited on 26th Mar. 2023).

[32] Canonical Ltd. *Ubuntu Server*. URL: https://ubuntu.com/download/server (visited on 18th Apr. 2023).

[33] *Network configuration*. en. URL: https://ubuntu.com/server/docs/network-configuration (visited on 6th May 2023).

[34] *Object-oriented programming*. en-US. URL: https://en.wikipedia.org/wiki/Object-oriented_programming (visited on 10th Apr. 2023).

[35] *Pixhawk Flight Controller*. en-US. URL: https://bluerobotics.com/store/comm-control-power/control/pixhawk-r1-rp/ (visited on 24th May 2023).

[36] *Power Sense Module*. en-US. URL: https://bluerobotics.com/store/comm-control-power/control/psm-asm-r2-rp/ (visited on 28th May 2023).

[37] Jordan Press. *EZO-DO Embedded Dissolved Oxygen Circuit*. V5.6. Atlas Scientific. Mar. 2023. URL: https://files.atlas-scientific.com/DO_EZO_Datasheet.pdf (visited on 21st May 2023).

[38] Jordan Press. *EZO-EC Embedded Conductivity Circuit*. V6.4. Atlas Scientific. May 2023. URL: https://files.atlas-scientific.com/EC_EZO_Datasheet.pdf (visited on 21st May 2023).

[39] *Python (programming language)*. en-US. URL: https://en.wikipedia.org/wiki/Python_(programming_language) (visited on 10th Apr. 2023).

[40] *Robot Operating System*. en. Page Version ID: 1155022397. May 2023. URL: https://en.wikipedia.org/w/index.php?title=Robot_Operating_System&oldid=1155022397 (visited on 21st May 2023).

[41] *ROS2 Tutorials - ROS2 Humble For Beginners - YouTube*. URL: https://www.youtube.com/playlist?list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy (visited on 30th May 2023).

[42] Peder Brandstorp Sanden. *Standalone Underwater Monitoring Station*. original-date: 2023-01-26T15:07:24Z. Mar. 2023. URL: https://github.com/Pederbs/SUMS (visited on 6th May 2023).

[43] Peder Brandstorp Sanden. *SUMS-Tools*. original-date: 2023-05-23T17:00:30Z. May 2023. URL: https://github.com/Pederbs/SUMS-Tools (visited on 23rd May 2023).

[44] Atlas Scientific. *How to tell if you Dissolved Oxygen probe needs maintenance*. Dec. 2018. URL: https://www.youtube.com/watch?v=L8fw78CsBE0 (visited on 21st May 2023).

[45] *Secure Shell*. en. Page Version ID: 1152080600. Apr. 2023. URL: https://en.wikipedia.org/w/index.php?title=Secure_Shell&oldid=1152080600 (visited on 21st May 2023).

[46] *SubConn handling instructions*. en. URL: https://www.macartney.com/what-we-offer/systems-and-products/connectors/subconn/subconn-technical-information/subconn-handling-instructions/ (visited on 27th May 2023).

[47] *Tattu Plus 12000mAh 22.2V 15C 6S1P Lipo Smart Battery Pack with AS150 + XT150 Plug (New Version)*. en. URL: https://genstattu.com/tattu-plus-15c-12000mah-6s1p-as150-xt150-plug-lipo-battery.html (visited on 3rd May 2023).

[48] *TATTU PLUS INTELLIGENT FLIGHT BATTERY*. en. TA-P2-15C-12000-6S1P-AS150. Tattu. URL: https://www.genstattu.com/content/Tattu-Plus.pdf (visited on 4th May 2023).

[49] *THN 15-2411WIR | Traco Power*. URL: https://www.tracopower.com/int/model/thn-15-2411wir (visited on 2nd May 2023).

[50] *tsys01-python*. original-date: 2017-01-21T07:05:48Z. Feb. 2022. URL: https://github.com/bluerobotics/tsys01-python (visited on 10th May 2023).

[51] *Ubuntu (Debian) — ROS 2 Documentation: Humble documentation.* URL: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html (visited on 6th May 2023).

[52] *Underwater Networks Handbook.* URL: https://unetstack.net/handbook/unet-handbook.html#_part_ii_setting_up_underwater_networks (visited on 28th May 2023).

Appendix

A Screenshot of RQT Graph

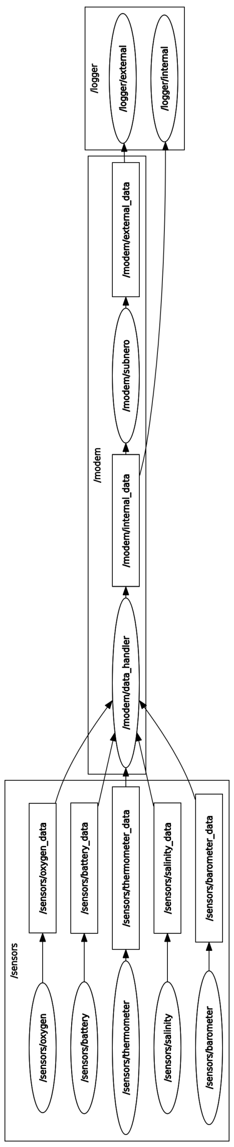
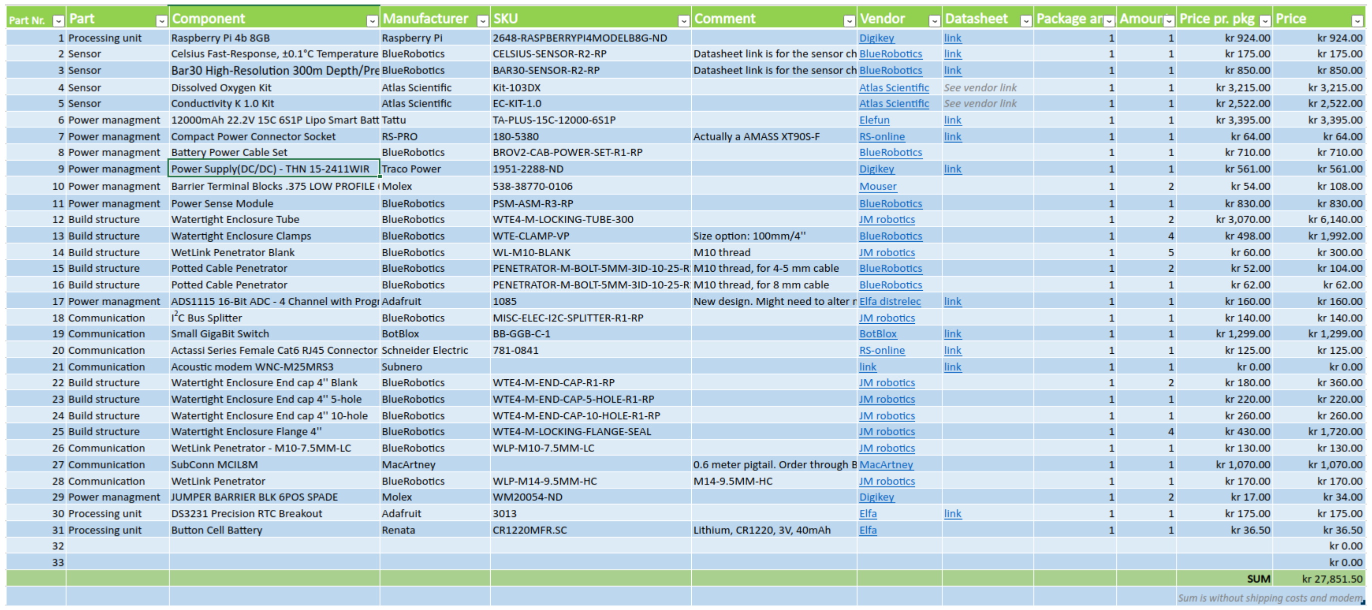


Figure 7.2: SCREENSHOT OF rq_graph.

B Bill of materials



Part Nr.	Part	Component	Manufacturer	SKU	Comment	Vendor	Datasheet	Package ar	Amour	Price pr. pkg	Price
1	Processing unit	Raspberry Pi 4b 8GB	Raspberry Pi	2648-RASPBERRYPI4MODELB8G-ND		Digikey	link	1	1	kr 924.00	kr 924.00
2	Sensor	Celsius Fast-Response, ±0.1°C Temperature	BlueRobotics	CELSIUS-SENSOR-R2-RP	Datasheet link is for the sensor ch	BlueRobotics	link	1	1	kr 175.00	kr 175.00
3	Sensor	Bar30 High-Resolution 300m Depth/Pre	BlueRobotics	BAR30-SENSOR-R2-RP	Datasheet link is for the sensor ch	BlueRobotics	link	1	1	kr 850.00	kr 850.00
4	Sensor	Dissolved Oxygen Kit	Atlas Scientific	Kit-103DX		Atlas Scientific	*See vendor link*	1	1	kr 3,215.00	kr 3,215.00
5	Sensor	Conductivity K 1.0 Kit	Atlas Scientific	EC-KIT-1.0		Atlas Scientific	*See vendor link*	1	1	kr 2,522.00	kr 2,522.00
6	Power managment	12000mAh 22.2V 15C 6S1P Lipo Smart Batt	Tattu	TA-PLUS-15C-12000-6S1P		Elefun	link	1	1	kr 3,395.00	kr 3,395.00
7	Power managment	Compact Power Connector Socket	RS-PRO	180-5380	Actually a AMASS XT90S-F	RS-online	link	1	1	kr 64.00	kr 64.00
8	Power managment	Battery Power Cable Set	BlueRobotics	BROV2-CAB-POWER-SET-R1-RP		BlueRobotics		1	1	kr 710.00	kr 710.00
9	Power managment	Power Supply(DC/DC) - THN 15-2411WIR	Traco Power	1951-2288-ND		Digikey	link	1	1	kr 561.00	kr 561.00
10	Power managment	Barrier Terminal Blocks .375 LOW PROFILE (	Molex	538-38770-0106		Mouser		1	2	kr 54.00	kr 108.00
11	Power managment	Power Sense Module	BlueRobotics	PSM-ASM-R3-RP		BlueRobotics		1	1	kr 830.00	kr 830.00
12	Build structure	Watertight Enclosure Tube	BlueRobotics	WTE4-M-LOCKING-TUBE-300		JM robotics		1	2	kr 3,070.00	kr 6,140.00
13	Build structure	Watertight Enclosure Clamps	BlueRobotics	WTE-CLAMP-VP	Size option: 100mm/4''	BlueRobotics		1	4	kr 498.00	kr 1,992.00
14	Build structure	WetLink Penetrator Blank	BlueRobotics	WL-M10-BLANK	M10 thread	JM robotics		1	5	kr 60.00	kr 300.00
15	Build structure	Potted Cable Penetrator	BlueRobotics	PENETRATOR-M-BOLT-5MM-3ID-10-25-R	M10 thread, for 4-5 mm cable	BlueRobotics		1	2	kr 52.00	kr 104.00
16	Build structure	Potted Cable Penetrator	BlueRobotics	PENETRATOR-M-BOLT-5MM-3ID-10-25-R	M10 thread, for 8 mm cable	BlueRobotics		1	1	kr 62.00	kr 62.00
17	Power managment	ADS1115 16-Bit ADC - 4 Channel with Progr	Adafruit	1085	New design. Might need to alter r	Elfa distrelec	link	1	1	kr 160.00	kr 160.00
18	Communication	I^{2}C Bus Splitter	BlueRobotics	MISC-ELEC-I2C-SPLITTER-R1-RP		JM robotics		1	1	kr 140.00	kr 140.00
19	Communication	Small GigaBit Switch	BotBlox	BB-GGB-C-1		BotBlox	link	1	1	kr 1,299.00	kr 1,299.00
20	Communication	Actassi Series Female Cat6 RJ45 Connector	Schneider Electric	781-0841		RS-online	link	1	1	kr 125.00	kr 125.00
21	Communication	Acoustic modem WNC-M25MRS3	Subnero			link	link	1	1	kr 0.00	kr 0.00
22	Build structure	Watertight Enclosure End cap 4'' Blank	BlueRobotics	WTE4-M-END-CAP-R1-RP		JM robotics		1	2	kr 180.00	kr 360.00
23	Build structure	Watertight Enclosure End cap 4'' 5-hole	BlueRobotics	WTE4-M-END-CAP-5-HOLE-R1-RP		JM robotics		1	1	kr 220.00	kr 220.00
24	Build structure	Watertight Enclosure End cap 4'' 10-hole	BlueRobotics	WTE4-M-END-CAP-10-HOLE-R1-RP		JM robotics		1	1	kr 260.00	kr 260.00
25	Build structure	Watertight Enclosure Flange 4''	BlueRobotics	WTE4-M-LOCKING-FLANGE-SEAL		JM robotics		1	4	kr 430.00	kr 1,720.00
26	Communication	WetLink Penetrator - M10-7.5MM-LC	BlueRobotics	WLP-M10-7.5MM-LC		JM robotics		1	1	kr 130.00	kr 130.00
27	Communication	SubConn MCIL8M	MacArtney		0.6 meter pigtail. Order through B	MacArtney		1	1	kr 1,070.00	kr 1,070.00
28	Communication	WetLink Penetrator	BlueRobotics	WLP-M14-9.5MM-HC	M14-9.5MM-HC	JM robotics		1	1	kr 170.00	kr 170.00
29	Power managment	JUMPER BARRIER BLK 6POS SPADE	Molex	WM20054-ND		Digikey		1	2	kr 17.00	kr 34.00
30	Processing unit	DS3231 Precision RTC Breakout	Adafruit	3013		Elfa	link	1	1	kr 175.00	kr 175.00
31	Processing unit	Button Cell Battery	Renata	CR1220MFR.SC	Lithium, CR1220, 3V, 40mAh	Elfa		1	1	kr 36.50	kr 36.50
32											kr 0.00
33											kr 0.00
										SUM	kr 27,851.50
										Sum is without shipping costs and modem	

Figure 7.3: Bill of materials.

A Standalone Underwater Monitoring Station

Sivert Berg Knudsen, Peder Brandstorp Sanden
Department of Engineering Cybernetics
Norwegian University of Science and Technology

Abstract

Our understanding of the Earth's atmosphere has made significant strides, while our knowledge of the ocean remains limited due to challenges such as inaccessibility and technological limitations. In this context, the standalone underwater monitoring station (SUMS) emerges as a crucial tool for oceanographic research. Designed to measure and communicate vital parameters in the ocean, the SUMS rig overcomes logistical challenges and provides unprecedented insights into the marine environment. Its versatility allows integration with Remotely Operated Vehicles (ROVs) for exploration in inaccessible areas. Advanced acoustic modem technology enables the formation of an interconnected network of observation posts, facilitating data collection and analysis to enhance our understanding of the ocean's dynamic nature.

The development of the SUMS rig has reached significant milestones, demonstrating successful sensor readings and data processing capabilities. However, further examination is required to ascertain the functionality and accuracy of the sensors. Hardware improvements and software optimization have contributed to a reliable and efficient final product ready to be submersed. [1]

The Sensing Rig

The rig contains five working sensors: temperature, pressure, dissolved oxygen, conductivity, and a power sense module to measure the voltage from its battery. SUMS aims to be a standalone monitoring system, meaning no dependencies such as external technologies should be needed to gain knowledge and collect useful data from the bottom of the sea.

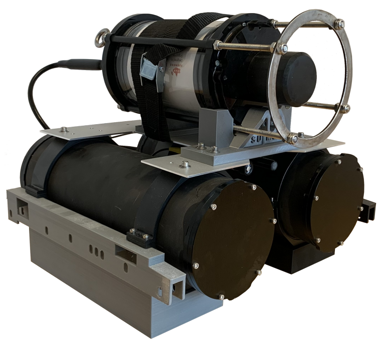


Fig1: Assembled rig.

With a high-capacity LiPo battery, the rig can monitor the environment for long periods. The acoustic modem mounted on top of the rig is responsible for communicating data to the surface. Combined it makes a completely wireless monitoring system ready to research areas that never before have been observed.

To ensure maximum range and comprehensive underwater exploration, the ability to communicate between the rigs and transfer data to each other has been implemented, theoretically ensuring an infinite covering range. This method is in its early stage and is one of the challenges regarding SUMS.

Using ROS2 to Implement Acoustic Communication

The program that makes up the sensing rig was made in ROS2 Humble. It consists of 9 nodes exchanging data and working together in order to sense the environment and communicate the collected values with custom messages posted to topics to the nodes responsible for communication with the acoustic modem. Furthermore, all data exchanged between agents in the underwater network are logged to the Raspberry Pi (RPi).

The modem communicator node is responsible for all communication with the acoustic modem. It sends data as soon as the topic is updated and from there it will listen for incoming transmissions. Incoming transmissions are logged for later use. Relevant parameters are specified from a launch file. The program is automatically launched as the rig is powered on, requiring no fiddling in the field.

3D Modeling and Production

The project aimed to integrate the acoustic modem into the existing rig construction. This involved securely mounting the modem to the rig using a reliable and straightforward approach. CAD modeling and production of rigid parts were chosen as the method to achieve this objective. A digital clone of the entire rig was created to ensure complete control over the various possibilities. Solutions were developed to securely attach multiple equipment components, including the acoustic modem and sensors, to the rig, maintaining equipment integrity while enabling easy removal when needed.

In order to connect the acoustic modem to the sensor cylinder, a cable is required that can both supply power and enable communication. However, the cables that come with the modem are excessively long and bulky. To address this issue, a solution for a custom waterproof cable needed to be found.

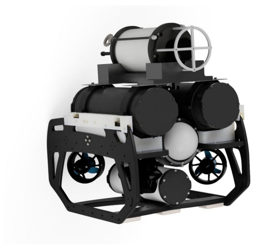


Fig2: SUMS 3D model attached to a ROV.

Internal Electronic System

The internal electronic system (IES) is comprised of sensors, an RPi, an Ethernet switch, and a DC/DC converter. The RPi is configured to be headless enabling fast computing and low sampling times.

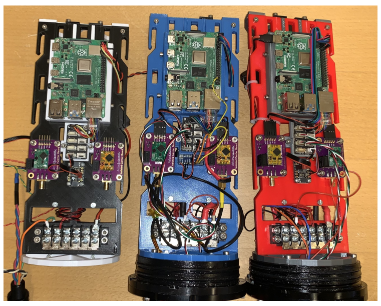


Fig3: Internal electronic system of all rigs.

Conclusions

The SUMS development has reached significant milestones, allowing field deployment. Sensor readings are successful, except for current sensing from the power sense module. Data is processed, logged, and transmitted via the acoustic modem. Ongoing evaluation is needed to ascertain sensor functionality accurately, particularly due to inconsistencies with the salinity and dissolved oxygen sensors. Pressure and temperature sensors perform as expected, but testing methodologies require improvement for better precision. The rig's design prioritizes functionality, reliability, and simplicity, adhering to waterproof standards. Hardware enhancements and software optimization boost performance and stability. Two rigs are ready for submersion, with minor component additions needed for the final one. Two fully operational rigs, excluding potting of two sensors, have been produced. Detailed instructions and an assembly guide ensure reproducibility and facilitate project scalability.

In summary, the progress made in the development and finalization of the underwater rig showcases a reliable and efficient monitoring system. The accomplishments thus far provide a strong foundation for utilization, as well as opportunities for continued refinement and expansion in the field of underwater technology.

References

[1] Sanden Peder Knudsen Sivert. "A Standalone Underwater Monitoring Station (SUMS) for marine wildlife monitoring". Bachelor thesis. Norwegian University of Science and technology, 2023.