NTNU
Kunnskap for en bedre verden

INSTITUTT FOR IKT OG REALFAG

AIS2900 - BACHELOROPPGAVE INGENIØRFAG

Next Generation Training Environment

Name:
Torstein Skare, 10006
Robert Alan Moltu, 10023

Dato: May 20, 2024

Preface

This is a project report made by two automation students from the Norwegian University of Science and Technology for Kongsberg Maritime located in Ålesund. The aim of this project is to enhance the existing training environment by developing a new system that utilizes visualizations to intuitively display data from control systems. This new system builds upon the foundations established in a previous communications program developed during a pre-project, aiming to further improve the communication and visualization capabilities that the prototype project had.

Acknowledgements

First and foremost, we would like to extend our gratitude to our project supervisors, Håkon Lunheim from Kongsberg Maritime (KM), Robin T. Bye, and Ottar Osen from Norwegian University of Science and Technology (NTNU). Giving us their guidance and support during our meetings throughout this semester. Their expertise, insights, and feedback have been important in the progress and development of our project.

We would also like to express our heartfelt thanks to our friends and families for their support, encouragement and motivation throughout this project.

A special thank you goes to Kongsberg Maritime for providing us with this thesis opportunity. We are grateful to Geir Olav Otterlei for initiating this project and to Per Magne Dalseth, whose inspiration and innovative ideas have been crucial. Their enthusiasm and support have allowed us to develop and explore our own ideas, making this project exceptionally rewarding and enjoyable. We are also grateful to Arnstein Erdal for his assistance with helping with parts of the filming and for loaning us the necessary camera equipment needed for the video that was made.

Finally, we appreciate the openness and willingness of Kongsberg Maritime Ålesund to give us the chance to build on our own ideas and visions. This opportunity has made the project especially unique and fun to undertake.

Thank you all for your contributions and support.

Abstract

The purpose of this project is to enhance the learning environment of Kongsberg Maritime (KM) systems. We propose achieving this goal through the integration of a SCADA-like interface and a visualization to view 3D models in augmented reality (AR). Building upon the ideas and prototypes developed during the summer and fall of 2023, our project group aims to create an innovative and practical product suitable for training, service, and sales purposes. This report documents the development of a new communication script and the visualization techniques used. The report details the problem-solving approaches adopted by the group and the challenges that the team encountered. Our project seeks to enrich the product environment and provide users with a more immersive experience, ultimately aiming to create a solution that can enhance the training, sales and effectiveness for the company.

Sammendrag

Formålet med dette prosjektet er å forbedre læringsmiljøet for Kongsberg Maritime (KM) systemer. Vi foreslår å oppnå dette målet gjennom integrering av et SCADA-lignende grensesnitt og en visualisering for å se 3D-modeller i ”augmented reality” (AR). Med utgangspunkt i ideene og prototypene utviklet i løpet av sommeren og høsten 2023, har vår prosjektgruppe som mål å skape et innovativt og praktisk produkt som egner seg for opplæring, service og salgsformål. Denne rapporten dokumenterer utviklingen av et nytt kommunikasjons skript og visualiseringsteknikkene som ble brukt. Rapporten beskriver problemløsningsmetodene gruppen har tatt i bruk og utfordringene gruppen møtte. Prosjektet vårt søker å berike produktmiljøet og gi brukerne en mer helhetlig opplevelse, og tar sikte på å skape en løsning som kan forbedre opplæringen, salget og effektiviteten for selskapet.

Table of Contents

Terminology

PLC Programmable Logic Controller

KM Kongsberg Maritime

CAF Common Application Firmware: A KM-developed application to monitor and tune parameters on a Marine Controller

Marine Controller A KM-developed controller that manages and controls different aspects of a vessel's operation

SWBD Software Block Diagram

GUI Graphical User Interface

SCADA Supervisory Control and Data Acquisition

WS Websocket

I/O Input/Output

API Application Programming Interface

UML Unified Modeling Language

RPM Rotations Per Minute

TT Tunnel-Thruster

TCNS Swing-up Azimuthing Thruster [9]

SDK Software Development Kit

1 Introduction

1.1 Background

Having a proper training environment is important for the company's reputation, economy, and environmental impact. Properly trained employees not only enhance operational efficiency but also play a crucial role in minimizing environmental impact through optimized fuel consumption and ensuring company reputation. Investing in proper and improved training for service engineers can yield numerous short-term benefits such as immediate skill enhancement and improved customer satisfaction, while also paving the way for long-term advantages like sustainable competitive advantage, higher retention rates, and enhanced reputation and customer loyalty. Additionally, aligning with Kongsberg Maritime's statement of 'Technologies for Sustainable Oceans'[10] ensures that the company's solutions not only drive profitability but also will contribute positively to Kongsberg Maritime's vision and values.

1.2 Current Training Environment

The current training environment relies on PowerPoint presentations and several lengthy paper documents, detailing the configuration of a fully equipped system. These paper documents include software block diagrams illustrating the signal flow between different software functions.

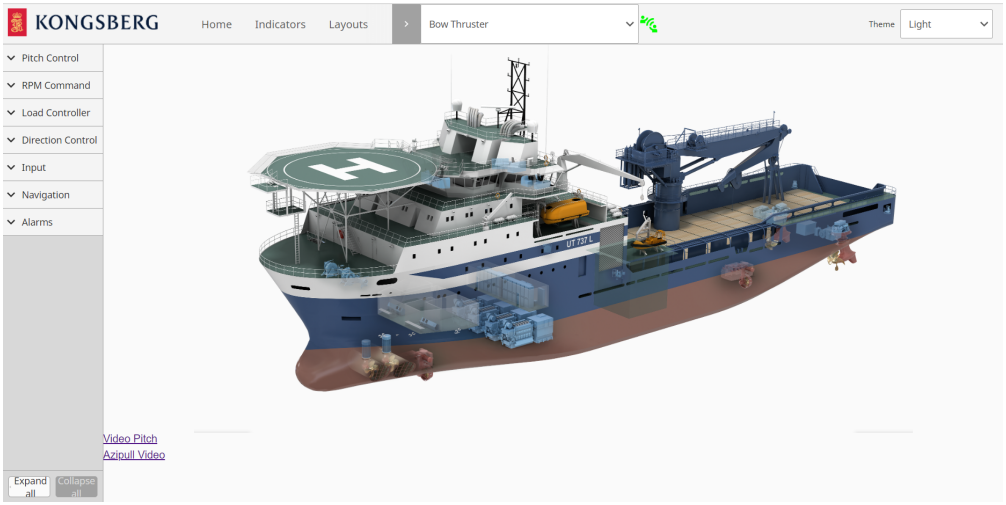
Moreover, the training setup comprises pelicases that houses a comprehensive thruster control system, excluding bulky hardware such as valves and an actual thruster, look at figure 1. Instead, the thruster is simulated using a stepper motor and an encoder, mimicking its behavior in response to the current setpoint. Currently, both in training and actual commissioning, CAF is used to access and change values and parameters in the control system. While this method works reasonably well, finding the right values in CAF can be a lengthy and complicated process, which may be challenging for course participants to navigate.

Figure 1: Thruster control system in a pelicase

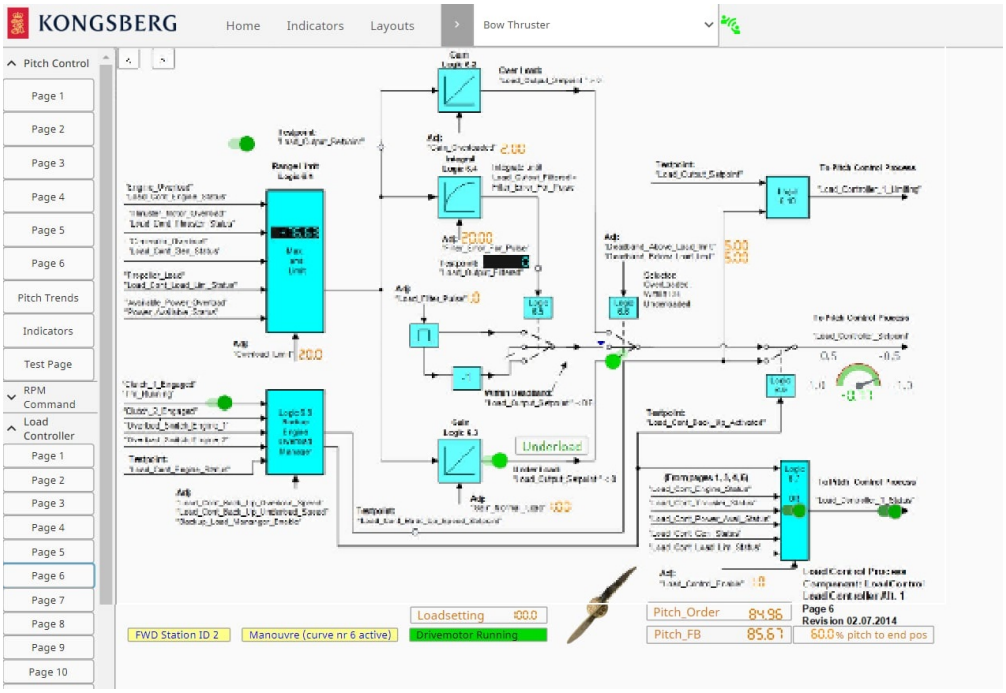
1.3 Project Baseline

During the summer and autumn of 2023, the project group developed a prototype that gathered data from the pelicase and displayed it on the Ignition platform, look at figure 2. To achieve this, a Python script was written to send requests to the server on the marine controller, which in turn responded with messages. This script was asynchronous and unfortunately unstable and prone to crashing frequently. Moreover, the time delay in response varied significantly, ranging from 2 seconds to 30-40 seconds when the program was initiated. Often, it was necessary to start the program multiple times to ensure it ran without crashing.

On the Ignition GUI designer, various topics and logics were seamlessly integrated with their corresponding software blocks. Interactive pop-ups were implemented to provide detailed software information upon double-clicking, while strategically placed displays provided real-time visualization of input and output values and other critical parameters.



(a) Homepage of First Prototype using Ignition.



(b) Load Control Page Live Interactive Software Block Diagram, *Blurred due to confidentiality.*

Figure 2: Prototype screenshots from summer and autumn 2023, showcasing the Ignition platform.

1.4 Project Objectives and Innovation

The project group aims to improve the training environment for new employees at Kongsberg Maritime. Our strategy focuses on streamlining the number of information sources and enhancing visualization. By integrating traditional systems with new technology, we aim to create an interactive and immersive learning experience.

Currently, it is challenging to effectively demonstrate how thrusters operate in unison. The project group and Per Magne Dalseth, one of the training instructors in Ålesund, believed that a 3D visualization of a vessel, along with the orientation and thrust direction of its thrusters, could provide trainees with a clearer overview of the system before delving into specific details about the software and hardware. Considering the growing interest in augmented reality (AR), we decided to explore the implementation of visualizing in AR.

The objectives for this project are:

- Improve stability and reliability of the communication program: Ensuring that the training system is dependable and reduces downtime caused by software crashes.
- Adapt the program to accommodate various CAF versions: Providing flexibility and compatibility with different versions of the software used in training.
- Collect and display data from multiple thrusters simultaneously: Enhancing the training by providing real-time data from multiple sources, giving a more comprehensive understanding of the system.
- Write data back to CAF using the Ignition GUI: Allowing trainees to interact with the system more effectively, simulating real-world scenarios.
- Visualize thruster movements in AR: Providing an innovative and engaging way to understand the operation of thrusters, improving retention and understanding.

1.5 Structure of the Report

This report follows a standard template for clarity. The methods are divided into two main chronological parts: Making the Communication App and Visualization in VR/AR, along with several smaller sections placed where they fit best. The methodology section details our decision-making process and the development of solutions. The results section presents and analyzes our findings, demonstrating the effectiveness of our approaches. The discussion section examines our methods and results, addressing strengths, weaknesses, challenges encountered, and future possibilities. Additionally, the recommendations section provides practical advice for anyone looking to replicate or continue this project. Since the project is written for Kongsberg Maritime, some of the Software Block Diagram images are blurred due to confidentiality.

2 Theory

2.1 Visualisation

2.1.1 Blender

”Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. Advanced users employ Blender’s API for Python scripting to customize the application and write specialized tools; often these are included in Blender’s future releases. Blender is well suited to individuals and small studios who benefit from its unified pipeline and responsive development process.”[3]

Figure 3: Blender Logo

2.1.2 Unreal Engine

”Unreal Engine (UE) is a series of 3D computer graphics game engines developed by Epic Games, first showcased in the 1998 first-person shooter video game Unreal. Initially developed for PC first-person shooters, it has since been used in a variety of genres of games and has been adopted by other industries, most notably the film and television industry. Unreal Engine is written in C++ and features a high degree of portability, supporting a wide range of desktop, mobile, console, and virtual reality platforms.”[22]

Figure 4: Unreal Engine Logo

2.1.3 Unity

Unity is described as ”a cross-platform game engine developed by Unity Technologies first developed as a Mac OS X game engine, but has now gradually extended to support a variety of desktop, mobile, console and virtual reality platforms. Particularly popular for iOS and Android mobile game development for making ”indie games”. The game engine is considered as an engine that is easy for beginner developers to use.

The engine can be used to create three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering, construction, and the United States Armed Forces.”[21]

Figure 5: Unity logo

2.1.4 Threepp

Open-source C++ library for 3D visualization [16], developed by NTNU Ph.D. Lecturer Lars Ivar Hatledal. Built upon the Three.js [20]JavaScript 3D Library.

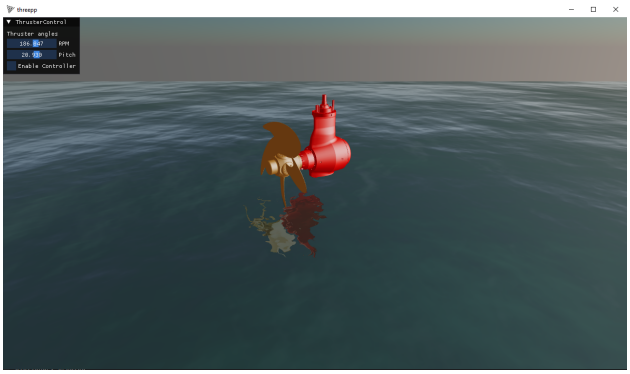


Figure 6: Threepp example of our 3D-Thruster made in the Pre Project

2.1.5 Autodesk 3DS Max

”3ds Max is used to model, animate and render detailed 3D characters, photorealistic designs and complex scenes for film and TV, games and design visualisation projects. 3ds Max is used by 3D modellers, animators and lighting artists for game development, film and TV productions, and design visualisation projects” [2]

Figure 7: Autodesk 3DS MAX Logo

2.1.6 Meta Quest 3 VR/AR Headset

The newest headset Meta provides today and is a cost efficient VR/AR headset that can provide mixed reality. It is used in this project to visualize the thruster movements in augmented reality (AR). This headset brand was formerly known as Oculus but was bought up by Meta in 2014 changing the headsets name from the Oculus to Meta.[12]

Figure 8: Meta Quest 3 Mixed Reality Headset [12]

2.2 CAF

The Common Application Framework (CAF) is a software framework developed by Rolls-Royce to interact with their control systems.

2.3 OPC-UA

OPC-UA (Open Platform Communications Unified Architecture) is a versatile and platform-independent protocol designed for industrial automation, developed by the OPC Foundation. It supports important features like security, organizing data in a complex structure, and two types of communication: publisher/subscriber and server/client. OPC-UA uses TCP/IP for communication, making it widely compatible and straightforward to implement across different systems.

2.4 Software Version Table

2.5 Package Table

Software	Version
Python	3.9
Unity	2022.3.20f1
Unreal Engine	5.3
Blender	4.0
Autodesk 3DS Max	2024
Pycharm IDE	2023.3.1.
CAF	1.17.5.7

Table 1: Software Version Table

Software	Package	Version	Description
Python	aiohttp	3.8.4	A popular asynchronous HTTP client/server framework for Python [1]
Python	pyinstaller	5.12.0	Converts Python applications into stand-alone executables, under Windows, Linux, and Mac
Python	colorama	0.4.6	Makes ANSI escape character sequences work under Windows
Python	aiohttp	3.8.4	Supports asynchronous request/response handling
Python	asyncua	1.0.2	An asynchronous client/server implementation of the OPC UA protocol
Unity	Meta All In One SDK	64	Asset used to incorporate VR/AR functions with Meta Quest goggles
Unity	Json.NET Converters	1.1.2	Library in Unity for JSON parsing

Table 2: Package Table

2.6 OpenBridge Design System

”OpenBridge Design System offers a collection of tools and approaches to improve implementation, design and approval of maritime workplaces and equipment. OpenBridge Design Guideline is a prominent part of OpenBridge Design System that explains how to design OpenBridge user interfaces.

It is a free resource built on modern principles of user interface and workplace design, adapted to maritime context and regulations.”[15]

2.7 Simple WebSocket Client

Simple WebSocket Client is a Google Chrome add-on that allows users to easily open a WebSocket connection and send requests. The response is displayed and continuously updated with new lines as data is received.

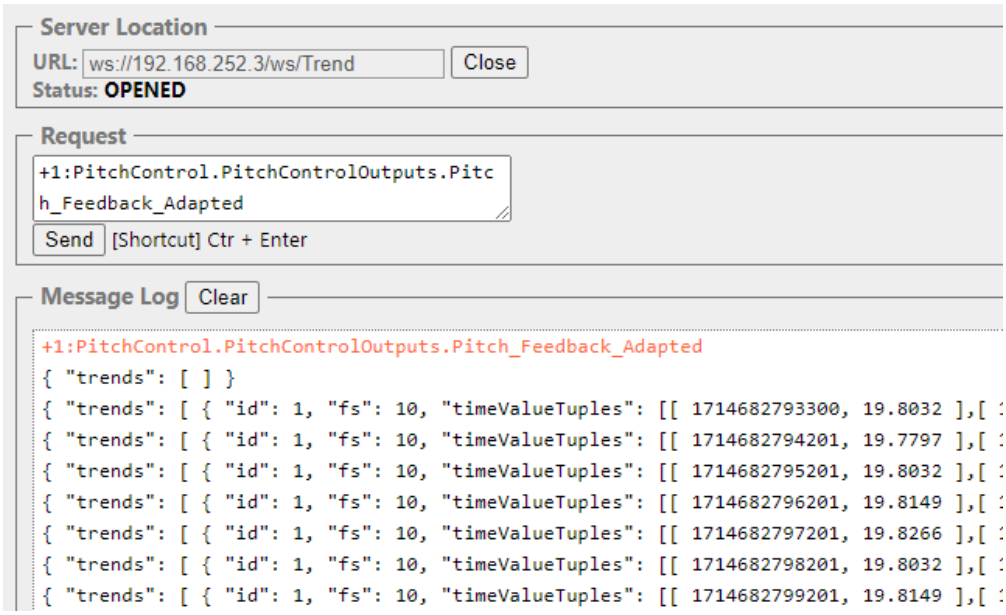


Figure 9: Example of Simple Web Socket Client

2.8 Pitch on Propeller

In this report, ”pitch” usually refers to the angle of the thruster blades.

2.9 Interpolation

Interpolation is a mathematical method used to estimate unknown values within a known range of data points. For example, it can increase the number of points between two data samples to make a simulation appear smoother.

2.10 Unity Asset Store

”The Unity Asset Store contains a library of free and commercial assets that Unity Technologies and members of the community create. A wide variety of assets are available, including textures, models, animations, entire project examples, tutorials, and extensions for the Unity Editor.”[17]

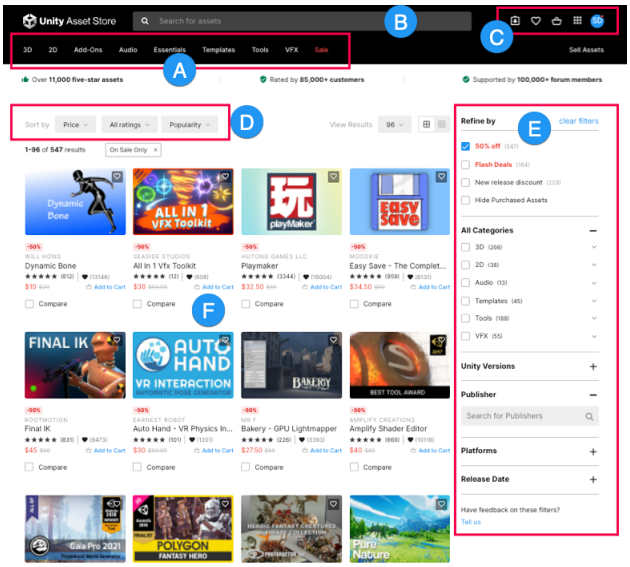


Figure 10: The Asset Store website, picture taken from the Unity Manual [17]

2.11 Tunnel Thruster

”Tunnel thrusters provide side force to the ship to improve manoeuvring capability in port or to provide additional station keeping power when dynamic positioning. Well proven, robust and reliable design.”[8]

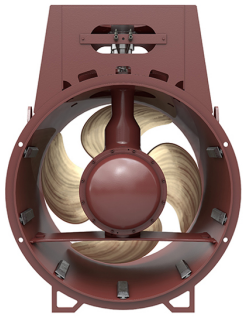


Figure 11: KM’s Tunnel Thruster, courtesy of Kongsberg Maritime [8].

2.12 TCNS - Swing-up Azimuth Thruster

”The TCNS type azimuth thruster offers all the advantages of a retractable thruster within an exceptionally compact design. When operational, the thruster provides high bollard pull, ensuring efficient positioning and manoeuvring of the vessel. For transit and operations in shallow waters, the thruster can be swung up above the baseline and retracted into its housing.” [9]

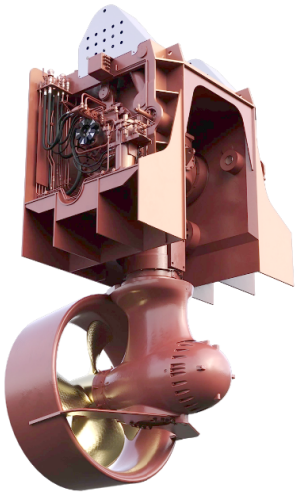


Figure 12: KM's TCNS Swing-Up Azimuth Thruster, courtesy of Kongsberg Maritime [9].

2.13 Azipull Thrusters

"The Azipull thruster is an azimuthing and pulling thruster with the propeller in front of the gear housing, providing the following advantages:

- The propeller's location ensures uniform water flow, which is ideal for a propeller designed for high efficiency, low pressure excitations on the hull and low underwater noise.
- A slender shape is possible using high precision gear wheels, this minimises hydrodynamic drag.
- A tail fin increases the rudder effect and reduces course correction losses while in transit."[7]

Figure 13: Azipull thruster

2.14 MQTT

”MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.”[14]

2.15 UDP

”User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as UDP/IP suite. Unlike TCP, it is an unreliable and connectionless protocol. So, there is no need to establish a connection before data transfer. The UDP helps to establish low-latency and loss-tolerating connections over the network. The UDP enables process-to-process communication.”[6]

2.16 Courotines

”Couroutines are multitasking methods where functions can pause execution and yield control back to Unity’s runtime which then resumes them later in the game”[17]

2.17 Parsing

”To examine computer data and change it into a form that can be easily read or understood.”[4]

3 Methodology

The initial step in enhancing the training environment was to identify key areas that needed attention. As discussed in section 1.2, the current training setup contains an excess of non-essential information, which complicates the learning process for trainees. This issue was addressed in the preliminary project outlined in section 1.3, where software block diagrams were presented in a SCADA-like format using a program called Ignition. However, the communication component of this solution was unstable, often crashing and failing to start reliably. It also struggled with connecting to multiple thrusters simultaneously.

The lead training instructor at Kongsberg Maritime Ålesund noted that some courses are designed specifically for the configurations of particular vessels, providing training directly relevant to the vessels that the attendees operate. This customization, however, introduces several challenges. Different vessels may not share the same functionalities, leading to variations in the software block diagrams. In addition, not all vessels use the latest versions of CAF, resulting in different communication standards. Ideally, the GUI should be tailored to each specific vessel, which would involve auto-generating software block diagrams. Although this is beyond the scope of this project, adapting the program to accommodate various CAF versions is feasible by developing a flexible communication program.

To make the program more interactive, the team wished to implement duplex communication in order to change the parameters and see the response from the control system, without having to change between CAF and the visualization. This approach not only streamlines the workflow but also significantly reduces the time and effort required for training. By allowing seamless parameter modifications and instant observation of the outcomes, the training process becomes more intuitive and effective. Users can experiment with different settings in a controlled and interactive environment, quickly understanding the effects of their adjustments, thereby accelerating the learning curve and improving the overall experience with the system.

3.1 Design and structure of the program

To facilitate the objectives outlined above, we adopted a modular approach based on a client-server relationship. This structure maintains a stable connection between the GUI and our program, while permitting client modules to dynamically select and connect with the appropriate server system. This flexibility allows the communication program to act as a versatile intermediary between the thruster control system and various GUI platforms. In addition, this program can be tailored for other use cases beyond the initial scope of this project. Such adaptability not only supports multiple clients, ensuring both backward and forward compatibility, but also facilitates easy integration with systems beyond the CAF.

The planned structure of the new training environment is shown in figure 14. The blocks on the left represents the controllers for each thruster on a vessel, ranging from one to multiple thrusters depending on the vessel type. These controllers contains a software called CAF which is interfaced with using the communication methods Rest API and Websocket. In this figure the data gathering app communicates via the websocket client module, this(grey box) is where other clients would be implemented to make the program work with other CAF versions. Which then retrieves the messages from the modules in to a OPC UA server (Red block). Thereafter distributing the topics and values coming from the control system to the visualizations (pink block). But in order to visualize in Unity the program needs a method to send the values from the OPC UA server through UDP messages (the dark blue blocks).

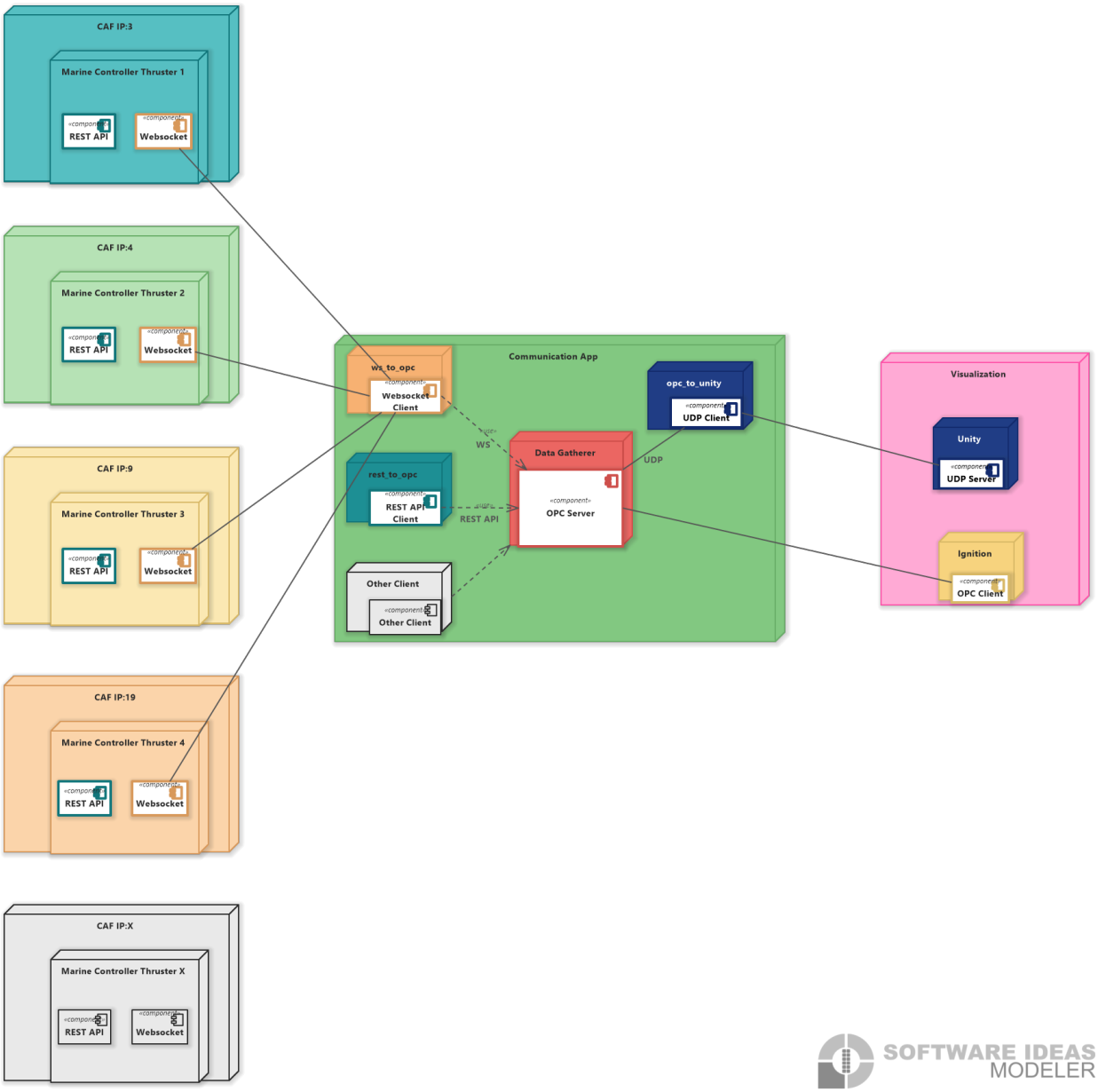


Figure 14: UML Deployment Diagram programs structure

3.2 Collecting data from CAF

While exploring viable solutions for data collection, the project team had a meeting with developers from Kongsberg Maritime, discussing three potential available methods: Websocket, MQTT, and REST API. Each approach was considered viable, but the REST API was ultimately favored. The preference for REST API stemmed from its flexibility in request frequency, as it is not limited by software constraints. Furthermore, the REST API supports both GET and POST operations, offering more comprehensive data interaction capabilities. It is also noteworthy that REST API and Websocket operate at different levels. REST API allows for requests of entire components, whereas Websocket is limited to individual data variables.

The option of using MQTT was discarded after careful consideration. The primary drawback of MQTT for this project was the need for an additional microcontroller to facilitate connections, complicating the system architecture. The team prioritized maintaining system simplicity and accessibility, leading to the decision against not incorporating MQTT.

Furthermore, the documentation provided was limited to accessing the REST API through the following endpoint: http://'ip-address'/api/spec. This documentation outlines how to use the CAF REST API.

3.2.1 Testing the collection methods

According to the REST API specifications, only sub-objects can be requested to obtain the data values we are interested in. This means that instead of requesting the entire "PitchControl" object, which includes both "PitchControlInputs" and "PitchControlOutputs" along with several other sub-objects, we had to request each sub-object separately. To confirm that the methods work, some tests were performed to confirm the number of variables and the update frequency that can be retrieved from the server. Starting with **REST API**, the test(image 15) revealed that one network session needs approximately 0.4 seconds to return data on one of the components.

```
GET request to http://192.168.252.3/api/components/SignalPool/protos/PitchControlInputs/fields took 0.47703027725219727 seconds
GET request to http://192.168.252.3/api/components/PitchControl/protos/PitchControlInputs/fields took 0.7701060771942139 seconds
GET request to http://192.168.252.3/api/components/PitchControl/protos/PitchControlTestpoints/fields took 1.1610445976257324 seconds
GET request to http://192.168.252.3/api/components/PitchControl/protos/PitchControlOutputs/fields took 1.5539674758911133 seconds
GET request to http://192.168.252.3/api/components/PitchControl/protos/PitchControlParams/fields took 1.974207878112793 seconds
GET request to http://192.168.252.3/api/components/LoadControl/protos/LoadControlInputs/fields took 2.366039276123047 seconds
GET request to http://192.168.252.3/api/components/LoadControl/protos/LoadControlOutputs/fields took 2.660085678100586 seconds
GET request to http://192.168.252.3/api/components/LoadControl/protos/LoadControlTestpoints/fields took 2.953977108001709 seconds
GET request to http://192.168.252.3/api/components/LoadControl/protos/LoadControlParams/fields took 3.3614249229431152 seconds
GET request to http://192.168.252.3/api/components/LoadControl/protos/LoadController1Testpoints/fields took 3.6528525352478027 seconds
GET request to http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandInputs/fields took 3.961491346359253 seconds
GET request to http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandOutputs/fields took 4.351402521133423 seconds
GET request to http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandTestpoints/fields took 4.649383306503296 seconds
GET request to http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandParams/fields took 4.958368301391602 seconds
GET request to http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandInputs/fields took 5.347954273223877 seconds
GET request to http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandOutputs/fields took 5.6581950187683105 seconds
GET request to http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandTestpoints/fields took 5.953319549560547 seconds
GET request to http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Testpoints/fields took 6.347946882247925 seconds
GET request to http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Outputs/fields took 6.65700888633728 seconds
GET request to http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Testpoints/fields took 6.842984676361084 seconds
GET request to http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandParams/fields took 7.154978513717651 seconds
GET request to http://192.168.252.3/api/components/AlarmInterface/protos/LoadControlAlarms/fields took 7.55516791343689 seconds
Gathering data took: 7.571168661117554 seconds
```

Figure 15: Test of REST API with one session using "restapi_single_session.py"

Since the server contains multiple components and sub components, REST API seemed too slow. In an attempt to

reduce the time, multiple network sessions were introduced, each of which took a third of the topics, but according to the test(image 16) it is not supported by the server because it takes roughly the same amount of time to gather the data.

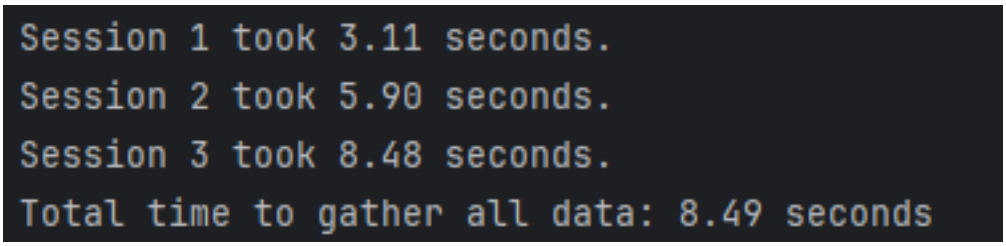


Figure 16: Test of REST API with multiple sessions. Using ”restapi_test_multiple_sessions.py”

From our discussion with KM developers, we gained insights into establishing a WebSocket connection with their server. Firstly, the connection had to be initiated via the WebSocket address ws://192.168.252.3/ws/Trend. Secondly, to subscribe to topics, a request has to begin with ”+1:” followed by the CAF path to the desired topic, such as +1:PitchControl.PitchControlOutputs.Pitch_Feedback_Adapted. We also learned that each data transmission included ten timestamp-value pairs, which introduced a one-second delay into our system. Given the slow operational nature of the system we were monitoring, this additional delay was considered acceptable. Verification through the Google Chrome extension ”Simple WebSocket Client” confirmed our understanding, although we noted that the number of pairs per response varied, sometimes more than doubling the expected amount.

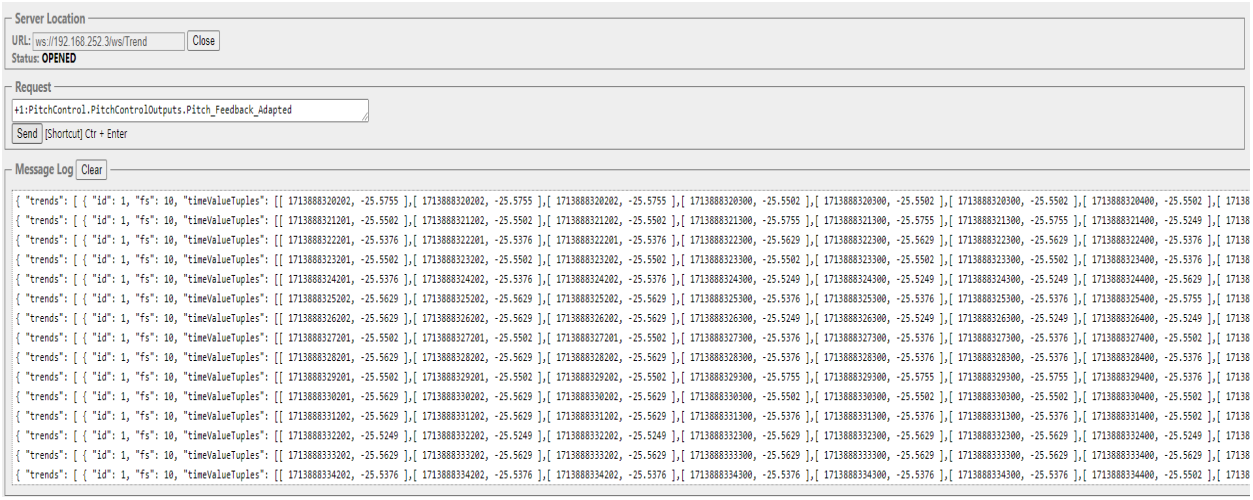


Figure 17: Test Using the Google Chrome Extension ”Simple WebSocket Client”

To find the limit of subscribable topics a test script was written in Python 8.2.3. This script runs 2 tasks receive_message(producer) and do_stuff(consumer) to get and print data. Currently do_stuff prints a line of hashtags to separate batches of data and receive_message attempts to open 200 connections. This test showed clearly (image 18) that 100 simultaneous connections is the maximum number of topics to be sent from the marine controller. Because of the round number of ”active connections”, the team believed that this had to do with server limitations.

```
 CONNECT: number active connections:  93
Connected to websocket server
 CONNECT: number active connections:  94
Connected to websocket server
 CONNECT: number active connections:  95
Connected to websocket server
 CONNECT: number active connections:  96
Connected to websocket server
 CONNECT: number active connections:  97
Connected to websocket server
 CONNECT: number active connections:  98
Connected to websocket server
 CONNECT: number active connections:  99
Connected to websocket server
 CONNECT: number active connections:  100
 ######################NEXT BATCH OF DATA#########################
Received message from topic  +1:PitchCommand.PitchCommandTestpoints.Pitch_Order_Digital_Scaling :  { "trends": [ ] }
```

Figure 18: Testing max websocket connections. Using code: ”WS_test_multiple_clients.py”

Comparing the two methods. From these tests we found that REST API can gather approximately $\frac{1}{0.4} \equiv 2.5 \approx 2$ objects per second, while websockets could collect 100 topics per second. Meaning that if the average 2 objects contained less than 100 topics, websockets were better suited to use for this project. From 2 random samples (image 19 and 20), the objects contained less than 50 topics. ChatGPT was utilized as a effective way to count the topics faster instead of counting all the topics by hand. This realisation of the objects containing less than 50 topics strengthened the teams initial beliefs, that websockets would be the most suitable option for this specific use-case.

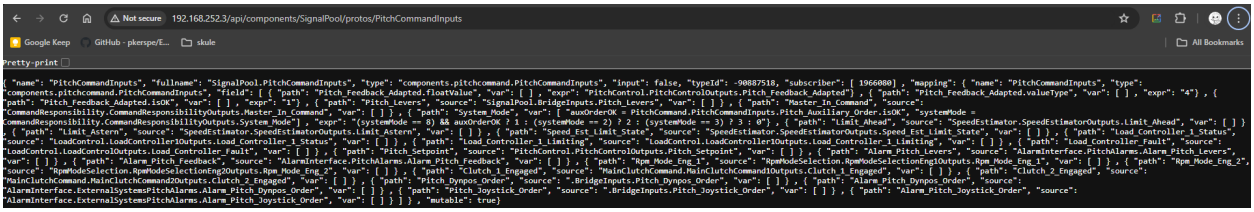


Figure 19: Component 1 containing 23 topics. Counted with ChatGPT

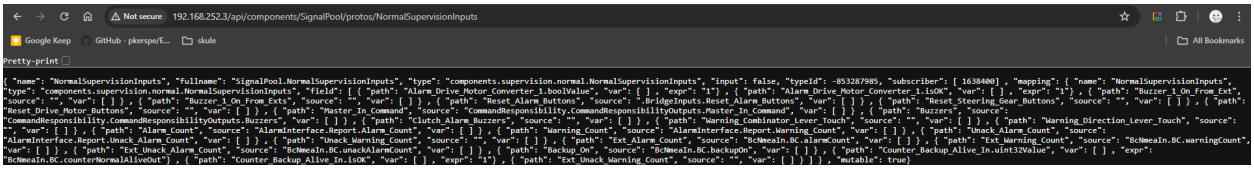


Figure 20: Component 2 also containing 23 topics. Counted with ChatGPT

3.3 Distributing Data

After collecting the data to the OPC-UA server the data had to be distributed to the GUI and visualization platforms. It was decided to continue using OPC-UA like the pre-project based on its scalability, security features and seamless integration with Ignition, which was displayed in the live software blockdiagrams made in the pre-project.

To learn more about visualization and attempt to cooperate with a professional firm, the team conducted a meeting with Morild Interactive, a local simulation firm. Project Engineer Per Magne Dalseth and Vice President(VP) for Services & Technical Support Geir Olav Otterlei attended the meeting with us. During this session, it was decided that Morild would integrate a UDP component and a visualization into their offering, with the specific UDP message format to be confirmed via email. Morild also promised to send a formal proposal detailing the costs involved.

Subsequently, our project group began developing the UDP client. However, upon receiving Morild's proposal, we found the costs too high, leading us to discontinue the collaboration. Despite this, the discussions had highlighted the potential of using Unity and UDP, which inspired our team to independently pursue this technology. We proceeded by configuring the UDP client to subscribe to our OPC-UA server and relay data to Unity via UDP.

3.4 Making The Communication App

The application for this project was created in multiple steps to minimize the room for errors. The team began by coding the program in C++, but realized quickly that this was too slow. Although creating the program in C++ would be beneficial in regards to efficiency and speed, we decided a switch to code in Python in order to rapidly test, develop and adapt our code. After switching to Python, we started by recreating the functionality that already existed in the pre-project, but developing the new program with an object oriented approach. This was done by creating the `start_opc_ua` and `websocket_to_opc` scripts as separate programs and launching them one by one. The same method was used to create the rest of the components listed in table 3 as well as several client classes which will be explained in detail later in the report. Additionally, it was decided to make the program asynchronous to increase the communication efficiency.

Script name	**Function**
App	Main entry point to the program.
start_opc_server	Contains everything to do with the OPC-UA server.
ws_to_opc	Utilizes the websocket and OPC clients to retrieve data from CAF and store it on OPC-UA
rest_to_opc	Demonstrates the possibility of having additional clients. Utilizes the CAF restApi and OPC clients to retrieve data from CAF and store it on OPC-UA
opc_to_unity	Utilizes the OPC and UDP clients to send data from OPC-UA to the IP address and port described in the connection data.json file.
functions	Contains various helper functions used throughout the other scripts for data handling and processing.
Simulate_thrust_load_rest	A program to simulate load and upload it to CAF
opc_client	A client to upload or download data from the server.
restapi_client	A class to use CAF restApi
ws_client	A class to retrieve data from CAF using websocket

Table 3: Overview of Code Components in the System

3.4.1 OPC-UA Server

The server leverages the Asyncio library for asynchronous functionality and the Asyncua library for OPC-UA functions. This approach allowed for the division of the program into separate parts, enhancing modularity and maintain-

ability. While the server setup drew inspiration from an example in the Asyncio documentation([5], modifications were made to accommodate new functionalities, such as a method to add variables to the server from the client side. This dynamic variable addition ensures that only existing variables on the CAF servers are incorporated, rather than relying on a predetermined list.

Notably, all security options were intentionally excluded by the line `server.set_security_policy([ua.SecurityPolicyType.NoSecurity])` to minimize potential communication errors and streamline the setup processes. Given that communication primarily occurs within local networks, the absence of security measures is deemed acceptable, considering the lower risk of unauthorized access.

The **OPC-UA Client Class** was created to interface with the OPC-UA server in a simple way. During its creation, it was noted that using the client module of the asyncua library was easier, therefore the opc-ua client was excluded in all parts of the code except some sample code used for learning and testing.

3.4.2 Websocket To OPC-UA

Websocket to OPC-UA interaface with CAF using websockets. The program takes two inputs, `connectionData` and a path to the topics text file. The main function extracts the data from `connectionData` and creates clients equal to the amount of topics in the text file. The program then starts several tasks asynchronously. It starts `receive_message2` once per topic and `process_queue` once to parse and upload the data to the server.

Websocket Client Class The websocket client class was created to simplify and objectify the usage of websocket. A websocket client object takes an ”uri” as a parameter and the class contains many functions used for testing and learning, but mainly 3 functions that are used in the final product: connect, receive_message2 and close.

`receive_message` and `receive_message2` differs in that `receive_message2` sends a new request after every response, in an attempt to get more topics than clients. This seemed to work, but was unreliable and some of the responses were mixed between topics. Therefore `receive_message` was implemented as a simplification which is limited to 100 topics. To see logic for `receive_message` look at the flowchart in figure 21.

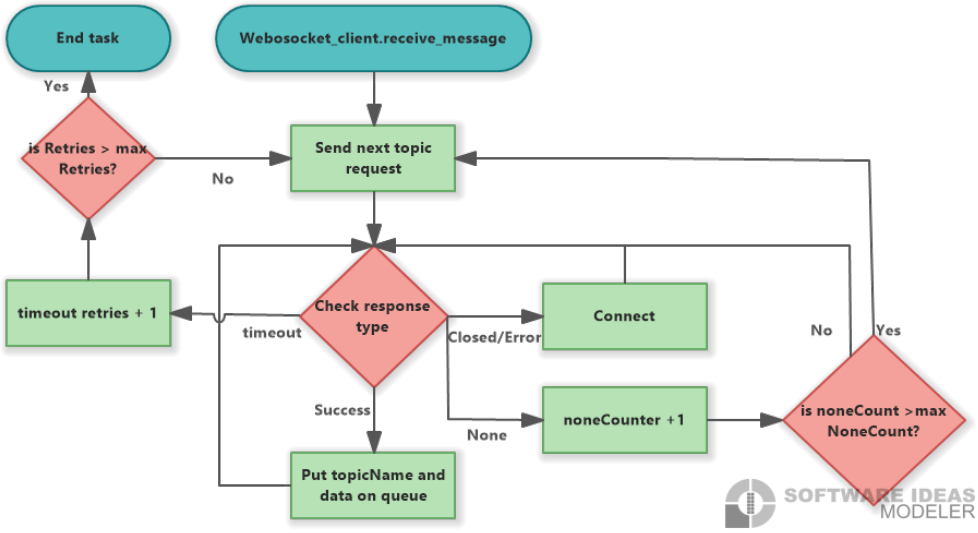


Figure 21: Simplified flowchart of `receive_message` function

3.4.3 Rest To OPC-UA

The program that leverages CAF's REST API to collect data was called `rest_to_opc`. This program uses a producer consumer pattern with 2 tasks where the function `receive_message`(producer) retrieves data using the aiohttp context manager, while `upload_to_opc` uploads the data to the OPC-UA server using the OPC client class of the asyncua library. The `rest_to_opc`'s main entry point takes two parameters. The first is a tuple with three values where the first is thruster connection data, second is OPC-UA connection data, and the last is the thruster name. The second parameter is the path to the text file with topics. The main loop of this program starts by storing the parameters as variables with understandable names before creating the queues, connecting to the OPC-UA server and starting the tasks.

`Receive_message`, the producer in this code runs a continuous loop while the client is connected and `connect ion_retries` is less than the retry limit. To achieve this, the "aiohttp" context manager is wrapped in a "try-except" block to track connection failures. If the connection succeeds, `connection_retries` is set to 0 and a `get` request is sent to the server. After this, the response status is checked. If the response status is 200(GET request successful), the data is processed and put on a queue to be shared with other tasks. If the connection fails, an exponential back-off strategy is used to reconnect the server. This strategy is used to give the system time to cool down in the case of overheating.

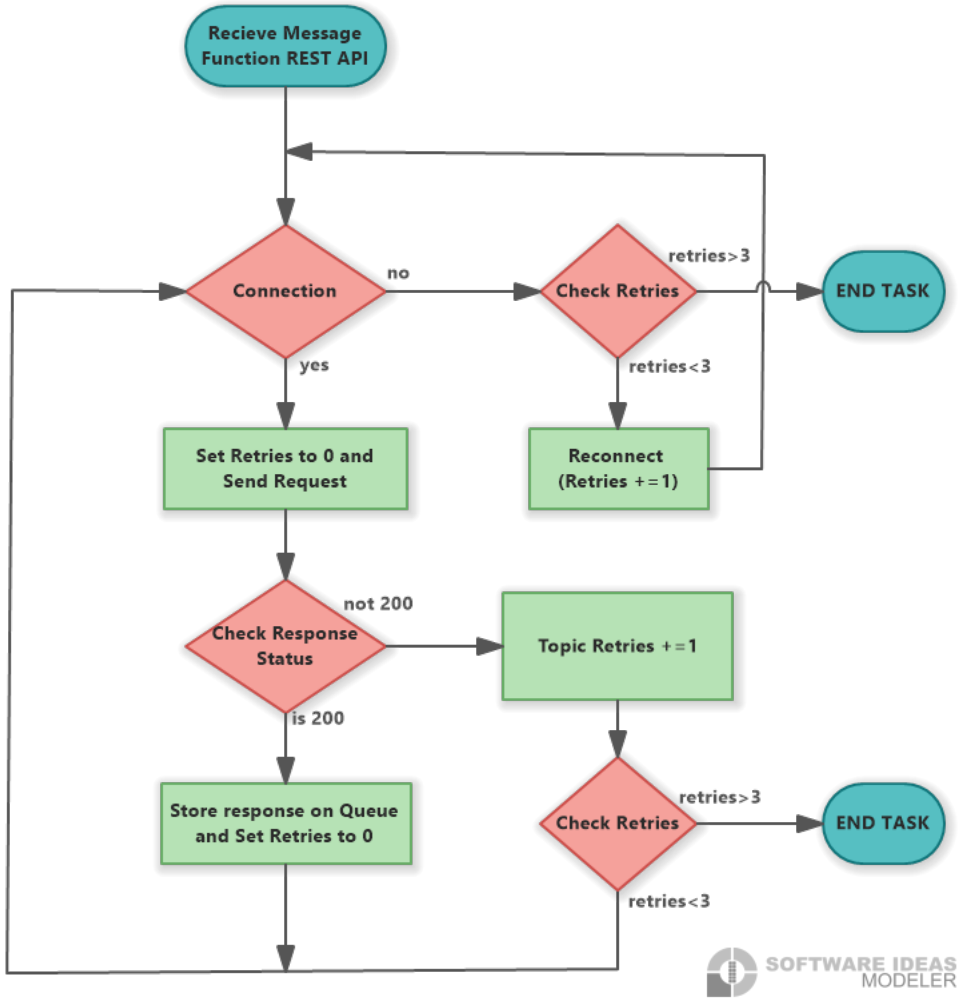


Figure 22: Simplified flowchart of `recieve_message` function

`upload_to_opc`, the consumer in this code also runs a continuous loop and takes an opc-ua client, a queue, and the object node ID as parameters. If the queue is empty the loop simply waits one second and checks again. If the queue contains data, the data is extracted. If the extracted data contains a value, a function called `determine_opcua_datatype` checks which datatype the value is before a test is done to check if the variable already exists on the server. If it does not exist, the variable is added. If it does exist then the value is simply updated.

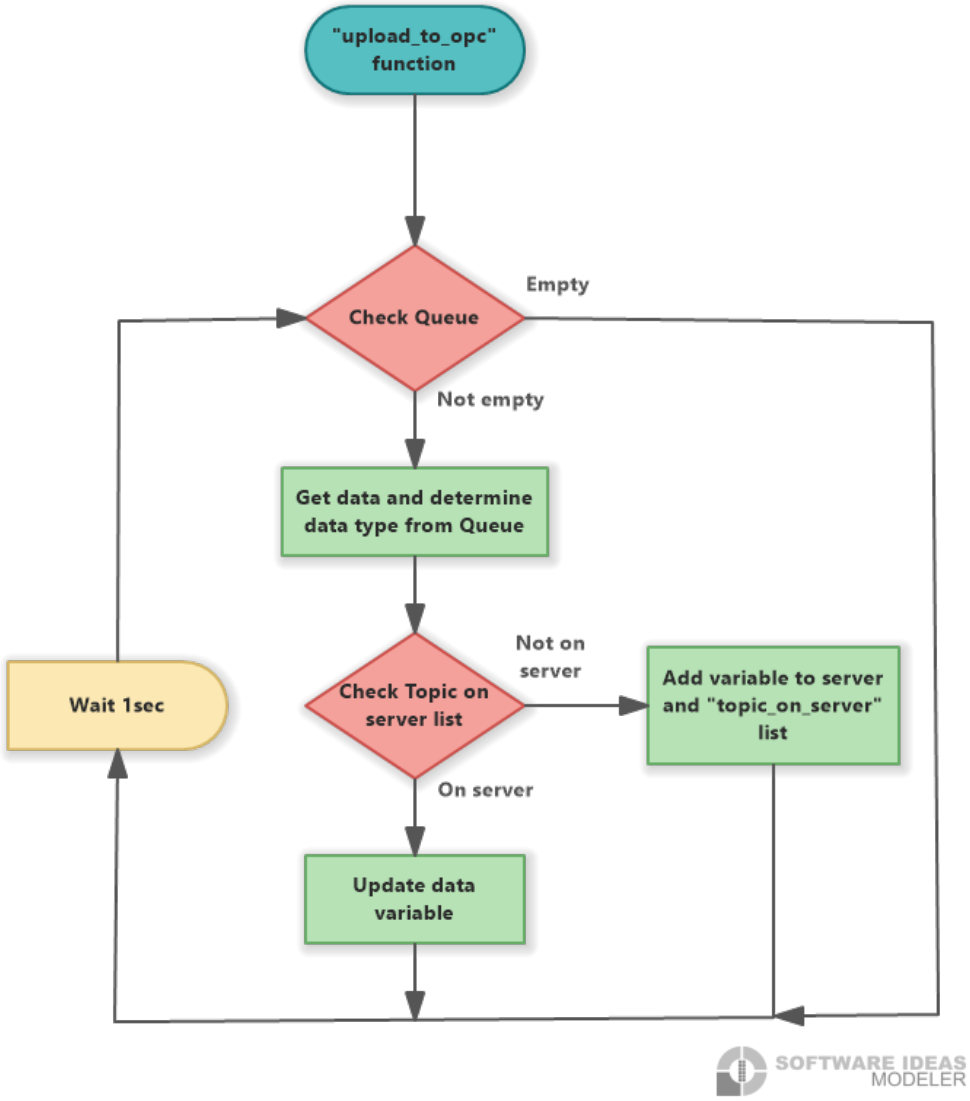


Figure 23: Simplified flowchart of upload_to_opc function

3.4.4 OPC-UA To Unity

The `opc_to_unity.py` script is structured to perform continuous data retrieval from the OPC-UA server and data transmission via UDP. This is facilitated by an asynchronous loop within the 'main' function, utilizing asyncio for non-blocking operations. Data is fetched, processed into a JSON format with specific key-value pair structuring, and sent through the 'UDPClient' instance. The 'UDPClient' class in '`udp_client.py`' manages a UDP socket, ensuring reliable data delivery to the specified IP address and port. The `opc_to_unity.py` script facilitates UDP communication. It retrieves data from the OPC-UA server and sends it, formatted as JSON, to the specified IP address. Data is then organized as pair of name values, with the thruster name followed by an underscore and then the variable name. For example:

```
{
    'thruster1_Pitch_Feedback_Adapted': -82.1145,
     'thruster1_Propeller_Rpm_Feedback_In_Rpm': 152.081
 }
```

UDP Client Class The UDP client class contains of 2 functions: `send_data` and `receive_data`. Using this class, a UDP client object can be made by simply giving the IP-address and port of the server or client to communicate with parameters. The class consists of important functions such as `send_data` and `receive_data`. The latter function has not been used, but was created preemptively for duplex communication with the GUI or visualizations.

3.4.5 Integration of System Components via the Main Application Module

The main application module, `app.py` shown in figure 24, serves as the the primary entry point of the program, effectively coordinating the interaction between different modules. Upon initiation, the application parses the `ConnectionData.json` file, which contains essential configuration settings and parameters. Based on the `client_to_use` variable, the application dynamically selects and initiates specific communication modules that interface with the OPC-UA server. These modules, forexample `ws_to_opc` are responsible for retrieving data from the CAF system and uploading it to the server.

Figure 24: Simplified flowchart of the main application

The initialization of these modules is determined by data specified in the `ConnectionData.json` file, particularly the details provided for each thruster. A loop constructs and task for each thruster, leveraging modules such as `ws_to_opc` for handling WebSocket communications and uploading the data to the OPC-UA server.

In scenarios where the `client_to_use` is set to "REST API", the system modifies the topic to simplify integration. For instance, a detailed topic path such as `+1:PitchControl.PitchControlTestpoints.Pitch_Setpoint_Deviation` is condensed to a format that incorporates the thruster name followed by a the description of the topic, like `ThrusterName_Pitch_Setpoint_Deviation`.

Following the configuration and initialization of the necessary modules, the `opc_to_unity` module is added to tasks to facilitate the data synchronization with Unity-based visualizations. The collective tasks, encompassing data retrieval and upload routines, are concurrently executed using the `asyncio.gather(*tasks)` function, driven by `asyncio.run()` to manage asynchronous operations efficiently.

This structured approach ensures that all system components are integrated smoothly and function cohesively to maintain the reliability and responsiveness of the data flow between the CAF system, the OPC-UA server, and the Unity visualizations.

3.4.6 Simulating Thrust Load

To make the training environment more realistic, the group decided that a simulation of the load on the thrusters could be beneficial. Thrusters in the real world would be affected by variables such as, wind, cargo, waves, and currents. The current thruster control simulator on the ”pelicase” has a direct connection between the setpoint and thruster feedback. A more sophisticated approach would involve breaking this direct link and routing the setpoint through a mathematical model that simulates environmental effects before updating the feedback signal, like shown in figure 25.

Before developing the mathematical model, the team believed it would be more prudent to implement the two-way communication system first. In the list below are several methods to implement the communication for the simulator:

- Connecting an external PLC to the I/O modules.
- Incorporating the simulation directly within the existing PLC.
- Utilizing REST API.
- Employing WebSocket communication.

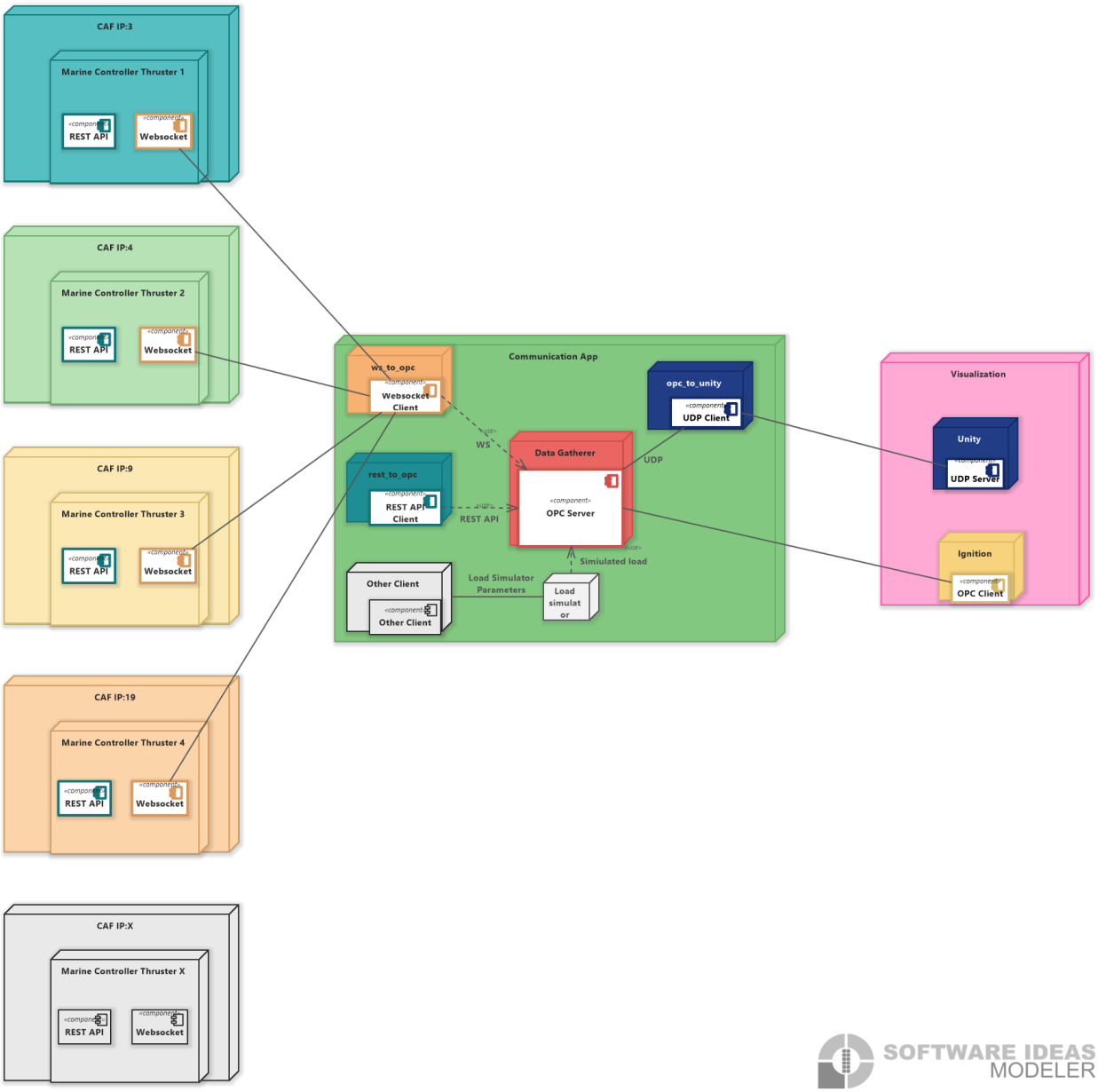


Figure 25: UML diagram with load simulator on one client

Given the group's prior experience with REST API and WebSocket communication, and considering no data had yet been sent back to the Control Application Framework (CAF), these methods were prioritized. Initial discussions with Kongsberg Maritime (KM) developers revealed limitations in the WebSocket server's support for two-way communication. Therefore, the team considered it wise to ensure the communication system's functionality before investing time in a mathematical model whose eventual use was dependant on the two way communication.

3.4.7 Challenges With Two-way Communication

Attempting to implement data feedback to CAF using REST API introduced several challenges. According to the API specifications, there were two viable methods for updating the data: "PUT" and "PATCH". The PUT method would overwrite all existing protobuff data and required sending the complete proto as a nested JSON. In contrast, the PATCH method only updated fields that were included in the request, with omitted fields retaining their previous values. Our first objective was to alter a single value which meant that PATCH was more suitable. However, testing revealed complications:

- Sending the complete message retrieved from a GET request back to the server via a PATCH request resulted in a 202 response code and caused CAF to crash(image 26).
- Similarly, attempts to modify and send back only the intended value using PATCH also triggered a 202 response and led to a subsequent crash.
- Also, testing with the PUT method yielded the exact same response as PATCH, indicating no difference in the outcome between these methods(image 27).

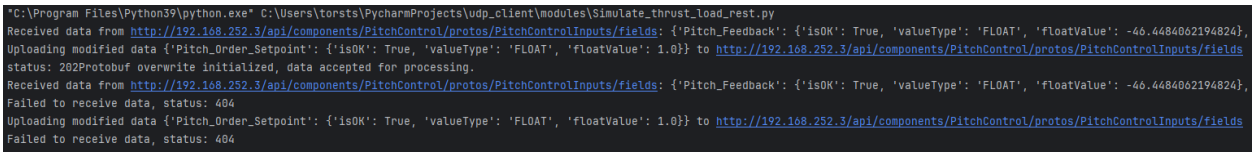


Figure 26: Attempt using the PATCH method. Code `Simulate_thrust_load_rest.py` was used in this test.

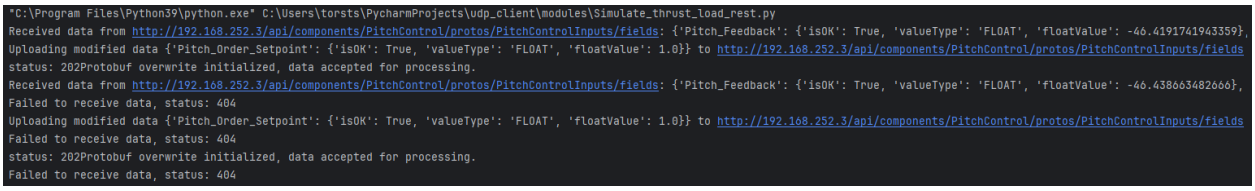


Figure 27: Attempt using the PUT method. Code `Simulate_thrust_load_rest.py` was used, but PATCH in line 53 was replaced with PUT.

The 202 response status meant, according to the specifications, that the server had successfully received the data and it was sent to the back end for computation. Given that we received a successful response but CAF crashed, we assumed that the message was wrongly formatted. Unfortunately there was not enough time available to explore this further as other parts of the project was prioritized.

3.4.8 Making the App more accessible

To make the program more accessible and more likely to be used, it was decided to convert the file into a distributable program that can be used without installing Python or any packages. To accomplish this, we utilized the PyInstaller library. The process began by creating a `app.spec` file, which specifies configuration details such as the executable name, included files/folders, and other advanced options(image 28). We chose to create this file using ChatGPT, instead of searching the web for a template. After creating the file the team added the readme file and Data folder manually.

Once the ”spec” file was prepared, the following commands were executed from the command prompt (CMD) within the project folder:

```
cd path\to\projectfolder
pyinstaller app.spec
```

This process resulted in the creation of a distributable folder containing the executable. However, initial attempts to run the executable revealed incorrect file paths. To resolve this, we modified the system directory path at the beginning of the Python script to ensure correct path references using the OS module from the Python standard library(image 29). Specifically, the script checks if the application is running as a bundled executable (using sys.frozen) and sets the current working directory to the folder where bundled assets are unpacked (sys._MEIPASS). If not running as a bundled executable, it is set to the directory where the script file is located. This ensures that file paths are correct even if the project is distributed and the project path is changed.

app.spec

```
# -*- mode: python ; coding: utf-8 -*-

block_cipher = None

a = Analysis(
    ['app.py'],
    pathex=[],
    binaries=[],
    datas=[('Data', 'Data'), ('readme.md', '.')],  # Including readme.txt at the root, and the Data folder
    hiddenimports=['executables'],
    hookspath=[],
    hooksconfig={},
    runtime_hooks=[],
    excludes=[],
    win_no_prefer_redirects=False,
    win_private_assemblies=False,
    cipher=block_cipher,
    noarchive=False,
)
pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)

exe = EXE(
    pyz,
    a.scripts,
    [],
    exclude_binaries=True,
    name='ngtec-v1.2',
    debug=False,
    bootloader_ignore_signals=False,
    strip=False,
    upx=True,
    console=True,
    disable_windowed_traceback=False,
    argv_emulation=False,
    target_arch=None,
    codesign_identity=None,
    entitlements_file=None,
)
coll = COLLECT(
    exe,
    a.binaries,
    a.zipfiles,
    a.datas,  # This line ensures that readme.txt, along with the Data directory, gets included in the final bundle
    strip=False,
    upx=True,
    upx_exclude=[],
    name='app',
)
```

Figure 28: App.spec file.

```
torsts
async def main():
    if getattr(sys, 'frozen', False) and hasattr(sys, '_MEIPASS'):
        os.chdir(sys._MEIPASS)
    else:
        os.chdir(os.path.dirname(os.path.abspath(__file__)))
```

Figure 29: using OS module to change directory.

3.4.9 Customizing OPC-UA Topics to Reuse Old GUI

Since parts of the Ignition GUI were developed prior to this project (as explained in 1.3), it was essential that the topics conform to a specific format. This was necessary to avoid the cumbersome task of rebinding all displays.

The format for the Ignition tags was specified as follows:

`[default]PitchCommand/_1:PitchCommand_PitchCommandTestpoints_Pitch_Order`

In this format:

- `'PitchCommand'` represents the folder where the tag is listed in the "Tag Browser" in Ignition(figure 30).
- `'_1:'` denotes the thruster name set in `connectionData.json`.
- `'PitchCommand_PitchCommandTestpoints_Pitch_Order'` indicates the topic path on CAF, meaning that the format should be: `folder/thrusterName+topicPath`.

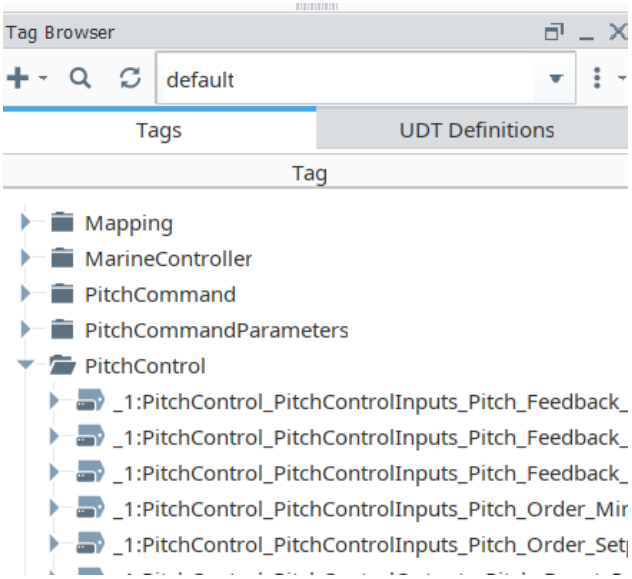


Figure 30: Ignition "Tag Browser" from pre project.

The thruster name must be `'_1'` for the current Ignition GUI to function as intended, see figure 30.

Initially, the program was uploading data in the format `"thrusterName_topicName"`. We believed the name could be changed to `"_1:"`, with the digit referring to which thruster it was, and the data could be manually placed into the correct folders. However, after our attempts, it became apparent that including a colon (`':'`) in the thrusterName caused issues during data upload to the OPC-UA server. We encountered an error message stating:

```
'Error parsing string _1:PitchCommand.PitchCommandTestpoints.Pitch_Setpoint_Protected',
```

accompanied by a `ValueError`:

```
"invalid literal for int() with base 10: '_1'"
```

This suggests that the server's parsing mechanism misinterprets the colon within the identifier, leading to upload failures. Attempts to cast the thrusterName into a string format were unsuccessful.

To resolve this problem, the team first imported all the topics into Ignition using the built in OPC-Browser. This renewed the OPC-UA item paths to use the string node identifiers. Then we exported the topics as a JSON file and edited all the topics names from starting with "_1" to start with "_1:", before re importing the file in to Ignition again.

3.5 Testing the Ignition Program with the App

After successfully reusing the Ignition GUI, tests were conducted to find the limitations and functionality of the program. Testing involved using two thruster simulation "pelicases" along with a test rig at NMK(figure 1 and 31, resulting in connections with two azimuth thrusters and two tunnel thrusters. This setup introduced a variety of test conditions. The main aspects of testing are outlined below:

1. **IP Address Configuration:**
    - Each thruster controller was assigned a unique IP address.
    - The `connectionData.json` file was updated with these new IP addresses.
    - Upon starting the script, all operations proceeded smoothly with all expected topics successfully uploaded to OPC-UA and displayed on the Ignition GUI.
2. **Disconnection and Reconnection Tests:**
    - Tests were conducted by manually disconnecting the ethernet cable from one of the thrusters.
    - It was observed that if the disconnection period was **shorter** than the preset timeout duration in the web-socket client, reconnection occurred automatically.
    - This revealed that the `try/except` block in the script did not trigger as expected, indicating that the timeout mechanism alone adequately handled the reconnections.
    - If the disconnection lasted longer than the timeout duration, an error message was generated, suggesting problems with adding topics. This issue likely stemmed from the program's attempts to re-add topics to the server, which is unintended and generates an error.

These findings suggest that while the websocket client efficiently manages short-term disconnections via its time-out feature, longer disconnections pose challenges that should be investigated.

Figure 31: Ferry installation test rig.

After adding the OPC-UA server connection to Ignition's connection settings, it automatically subscribes to all the data available and makes them available to use in the Ignition Designer editor.

To test that the communication with multiple thrusters was actually working. A test-page in Ignition was created. This page displays both thrust and azimuth angle for the 4 thrusters we used for testing. As shown in figure 32 the two azimuth thrusters located at the bottom has an angle, while the tunnelthrusters provide sideways force. The edge between blue and white signifies the thrust direction and power, meaning that if the edge is in the middle, there is no thrust. This might not be the most intuitive solution but it worked to test the communication.

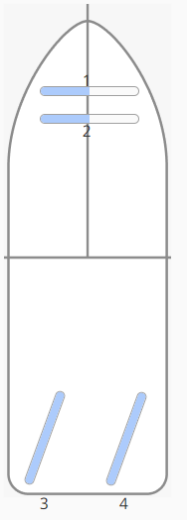


Figure 32: New test-page in Ignition showing multiple thruster functionality.

3.6 Visualization in VR/AR

This section outlines the methodology and tools employed to visualize thruster movements in augmented reality. We will detail the various methods, software packages, and equipment chosen for this project and discuss the reasons behind selecting these specific resources.

3.6.1 Tools and considerations

In order to find out what programs and equipment to incorporate in the project, the group had to figure out what the intended goals and objectives for the project were. The main goal of the project was to create an innovative and immersive real-time training experience, enhancing the learning outcome for the users. The decision was made to use augmented reality (AR) to visualize thruster movements to improve the training experience. Several factors were taken into consideration before deciding which software and equipment to use. In order to find the most suitable option for the intended scope of the project, extensive research was conducted. This research provided us with information about various software and hardware options available for AR.

3.6.1.1 Selection of Visualization Tools

There were several options of hardware and visualization programs available to use, towards our decision making about selecting the most suitable programs. The options discovered were game engines or programs such as Unity, Unreal Engine, Godot, Construct 3 and Blender game engine. Although Godot and Construct 3 are popular game engines, they did not align with the specific requirements and goals for the project. Our objectives necessitated the use of advanced 3D models from Kongsberg Maritime (KM) to create a visualization in augmented reality. Consequently, Unity, Unreal Engine and Blender seemed to be the best programs to use for visualizing the thrusters movements in real time, aligning with our projects requirements and goals.

3.6.1.2 Hardware and Software Considerations

Implementing augmented reality to the visualization in the project depended on selecting the appropriate hardware capable of providing such an environment. A consequence of this, was that the focus shifted towards headsets supporting mixed reality. Given that augmented reality is a relatively new technology, there were limited alternatives available. For this project, the feasible choices for headsets that fit the intended scope included the HTC VIVE XR Elite VR Headset, Meta Quest 3 (formerly known as Oculus), and Apple Vision Pro. Each of these options features pass-through functionality, essential for creating the augmented reality environment wanted for this project. The decision to select Meta Quest 3 was mainly influenced by cost-effectiveness. At the time this project was conducted, the HTC VIVE XR Elite VR was priced at NOK 16,690, the Meta Quest 3 at NOK 6,890, and the Apple Vision Pro at NOK 38,463, for a more structured comparison look at table 4. Given the significant price disparity, the Meta Quest 3 offered a practical balance between functionality and affordability. Not only did Quest 3 provide cost effectiveness, but it also scored very well in various tests, making it a promising headset to use in the project. Additionally, the biggest reason for not choosing the Apple Vision Pro over the Quest 3 was solely the fact that it was not possible to buy in Norway. Another important factor for deciding to use Quest 3 was the simplicity of developing on the headset in order to integrate with the Unity game editor. Therefore, choosing a well known headset brand with a lot of documentation and tutorials could prove as beneficial to complete the visualization.

Headsett	**Price in NOK**	**Price in USD**
Meta Quest 3	6890kr	626.78$
HTC VIVE XR Elite VR	16 690kr	1518.29$
Apple Vision Pro	38 463kr	3499$

Table 4: Headsett Pricing 2024

After evaluating various game engines, the choices were narrowed down to two, each with its own advantages and disadvantages. In order to make an informed decision, extensive research and some testing on the two engines was conducted. Ultimately, the decision to utilize Unity for the visualization was significantly influenced by its versatility and widespread adoption. This was evidenced by its extensive use within the community, used by companies like Morild Interaktiv AS—a marine shipping simulator company. As mentioned previously in chapter 3.3, prior to making our decision on which game engine to use, we visited Morild Interaktiv AS to discuss various aspects of visualization and simulation. Their valuable insights greatly influenced our choice to use Unity. Additionally, another important aspect worth to mention was the documentation and tutorials available for Unity compared to Unreal Engine, which strengthened our decision to proceed the visualization using Unity.

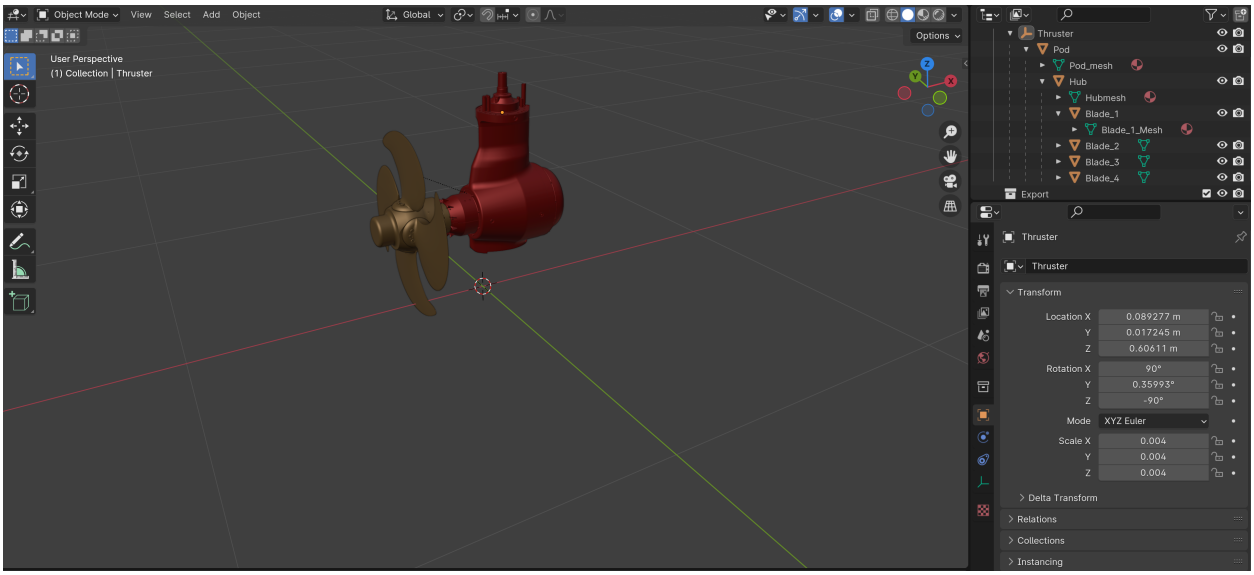
3.6.2 Acquiring and Preparing 3D Models

The visualization of the thrusters movements depended on acquiring and preparing the thrusters and UT-Boat 3D models in a software program that is capable of editing a variety of different file formats. For this project, the decision to utilize Blender as the 3D model editor was made due to the project group's previous experience with the application. The preparation of the 3D models regarded editing the colors, hierarchy and names of the objects which were essential for applying the models into Unity.

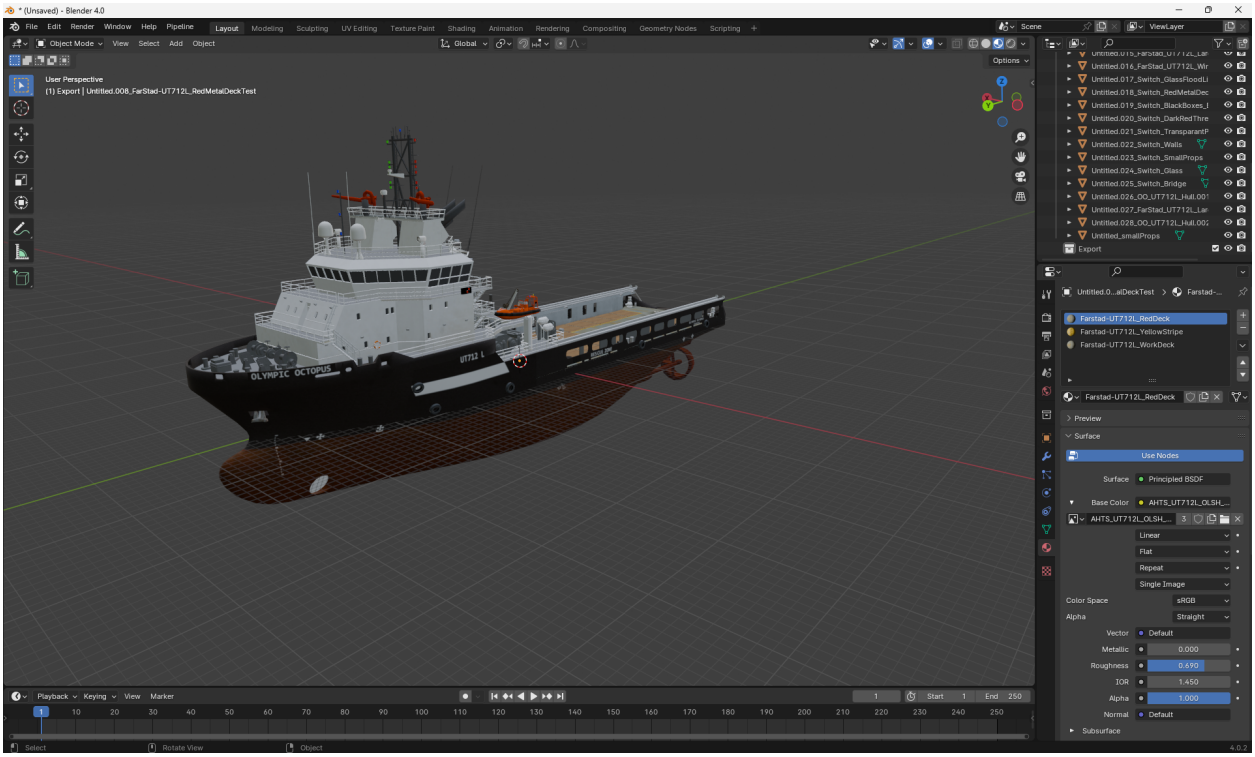
Since the goal of this project was to improve the training environment for Kongsberg Maritime and the overall system understanding for employers and customers, making our own 3D models was not the priority of this thesis. For

this reason, the group engaged in collaborative efforts with various departments within Kongsberg Maritime (KM) as well as with NTNU to acquire some models that were suitable to use for the visualization. Specifically, we obtained the BB-Octopus, see figure 33b from NTNU and the UT-7138 ship from KM's ship design department, see figure 39. The tunnel thruster used for this project was acquired during our preliminary project from last semester, which the team already had acquired from Kongsberg Maritime's department in Ulsteinvik, see figure 33a.

These models made it possible to add the thrustermovements directly on to the thruster design, which was a concept we already had proved in the first prototype visualization using Threepp, look at figure 6. The provided models were given to us as a "3DSMax" file. Since Unity does not support "3DSMax", we had to change the file. The conversion was accomplished by opening the models in Autodesk 3DSMax and exporting the models in a format called "FBX", that Blender and Unity supports. Adjustments to the models regarded such as editing the centerpoint, the naming and hierarchy of the child objects within the object and deleting unused and unwanted objects that were not essential for our visualization. This included objects like cranes, wires, camera animations and lights that had no relation to our visualization. The thruster objects structure and hierarchy was named logically with names suitable for each object instead of having part numbers to be each objects name in order to make the object easier to understand and use. Look at figure 33a with the hierarchy and naming to the right in the editor. In the provided figure one can see the structure of the thruster object which is the parent object to all the child objects within the thruster. Under the thruster one can see the pod which is the parent to the "hub" object which then is the parent to the propeller blades.



(a) Tunnel thruster in Blender editor with object hierarchy to the right



(b) Olympic Octopus Boat Model in Blender editor received from NTNU

Figure 33: Detailed Blender Models for the Visualization

3.6.3 Integrating 3D Models into Unity

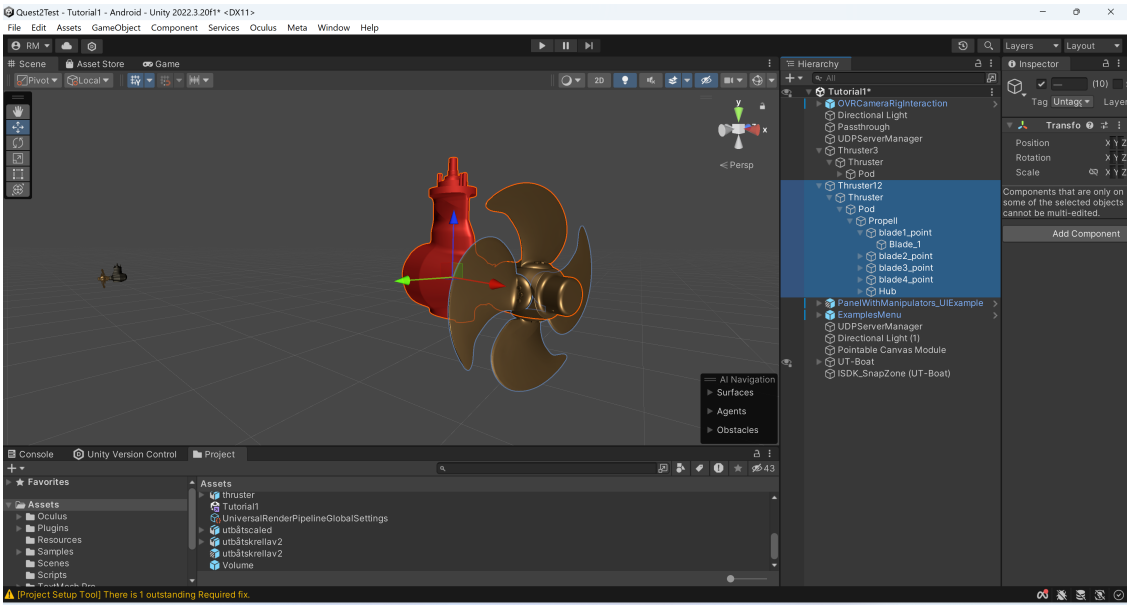
As mentioned in section 3.6.2, the hierarchy in the "*GameObject*" plays an important role in how the movement logic is implemented. In order for the blades to rotate with the correct pitch angle on the tunnel-thruster, some minor changes were necessary to edit the objects so that the centerpoint of each blade was correctly placed. The solution for this was to create an empty GameObject and define the centerpoint to be at the desired rotation point of each blade. Thereafter, structuring the new GameObject called `blade_point` as a parent to the blade objects, look at figure 34a to see the hierarchy and structure of the thruster object in Unity and figure 34b to see the location of the centerpoint for the `blade_point` and how it was positioned. In this picture the green arrow represents the Y-axis, the red represents the X-axis and the blue represents the Z-axis. Therefore, in order to rotate each blade's pitch angle one would need to rotate the `blade_point` object around the Y-axis.

3.6.4 Data Integration and Movement Simulation

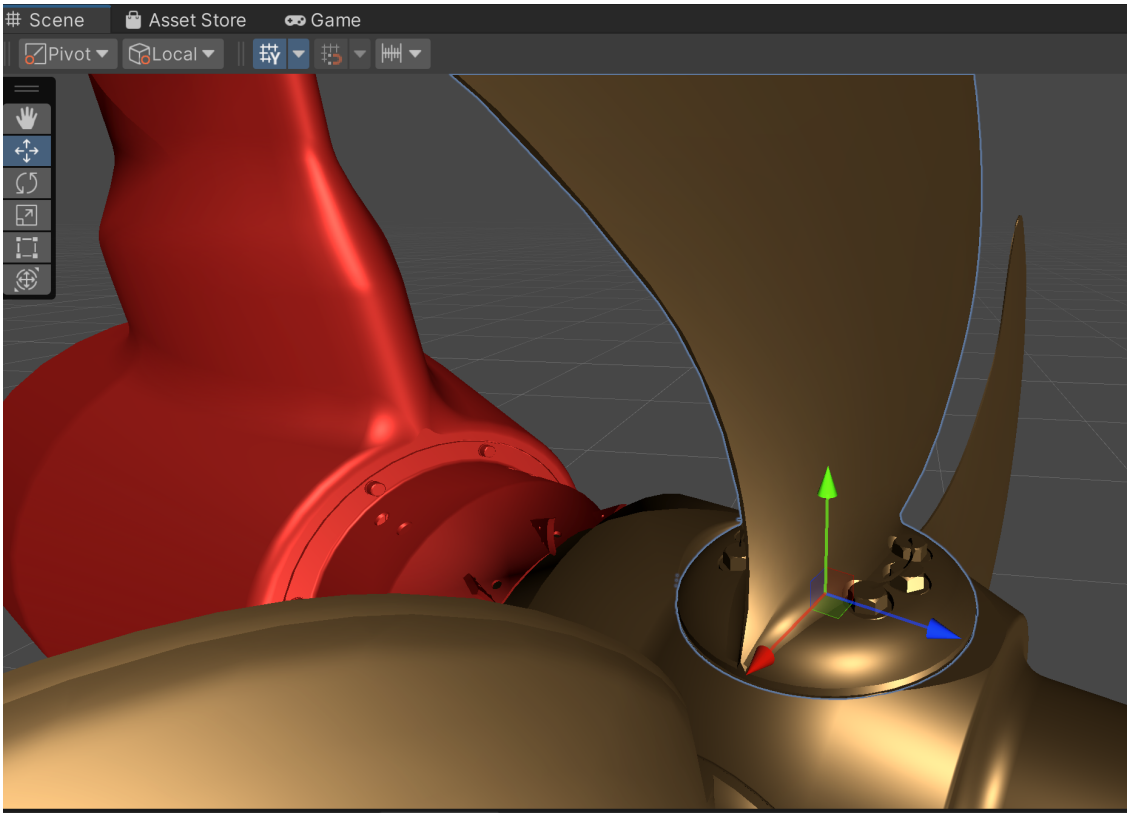
For the thruster to be able to move in accordance with the data coming from the marine controller, the development mentioned in 3.4 and the creation of a script that acts as a client to receive the data values was necessary. Additionally, the next step was creating a movement script that listened to the communication script in order to visualize the rotations and thruster movements.

3.6.4.1 Data Communication Setup

Implementing the communication script involved several considerations. The first step was to determine what communication protocol to use. The initial idea was to utilize OPC UA as the main communication protocol. OPC UA is also the communication used for the data gatherer program, which additionally communicates directly with the Software Block Diagram (SWBD) visualization on Ignition, as shown in figure 14. Therefore, a similar approach was looked upon as highly beneficial, but unfortunately Unity did not have built in support for OPC UA communications. For this reason, integrating OPC UA communication into Unity would require downloading a third-party library from the asset store or creating a custom solution that interfaced with OPC UA servers. Since the team had little to no experience using Unity and the price of using a third-party library from the asset store, the team figured out another solution that would be suitable for the project's needs. As mentioned in section 3.4.4, UDP was chosen as the communication protocol due to its built-in support and comprehensive documentation. Hence, making a client script in our data gatherer program which is described in section 3.4.4, that sends the data variables from our OPC UA server to the client in Unity. See figure 14 for a visual representation for the structure of the programs and figure 35 to see the flowchart of the UDPClient.



(a) Tunnel Thruster in Unity Editor Showing Hierarchy



(b) Blade Point to Correctly Rotate Pitch Angle of the Blade.

Figure 34: Visual Representations in Unity and Blade Mechanics

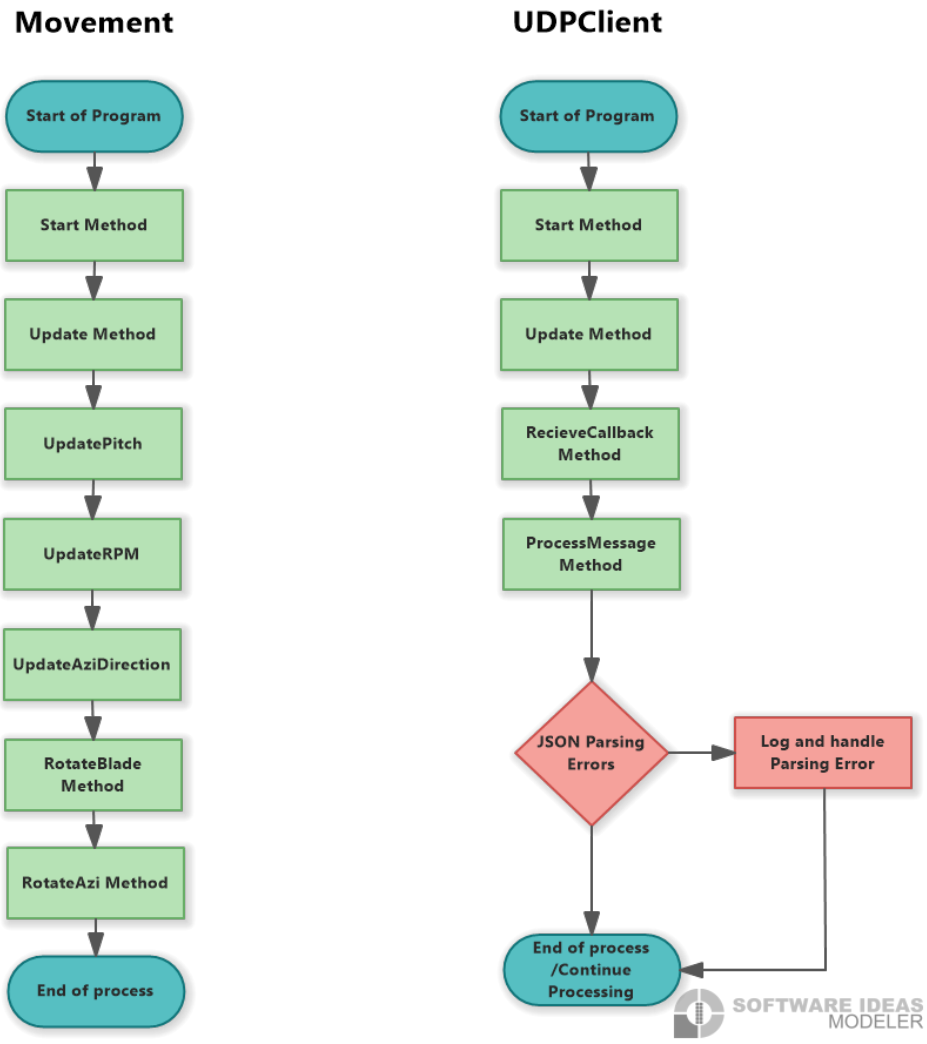


Figure 35: Flowchart diagram for Movement and UDPClient scripts in Unity

The inbuilt library used for implementing UDP was the .NET library, see the top of the code in listing 8.3.1. These packages are used to implement and receive the variables through UDP communication. In order to attach the class to a game object in Unity the class needs to inherit Monobehaviour[17]. Allowing the script to use Unity-specific functionality such as ”coroutines” to respond to game events by implementing specific methods like ”Start()” and ”Update()”. In this class, the `serverSocket` is defined using the .NET library to handle UDP networking communications. The port number (in this case, 5000) specifies the port the client listens on for incoming messages. Additionally, a thread-safe queue called `messageQueue`, which stores incoming messages until they can be processed, is created as a private member variable. Furthermore, the `UDPClient` class has the functions `void Start()` and `void Update()`. The `Start` function is called before the `Update` function, initializing the `UDPClient` to listen on the specified port and begin receiving data, thus starting the communication. Thereafter, the `Update` function is called every frame and processes all the messages that have been received and stored in `messageQueue` using the `ProcessMessage` function.

Additionally, the client has a callback method to receive and store the messages. The function `ReceiveCallback` is a callback method for `BeginReceive`, which reads data from the network and converts the byte array into a string using UTF-8 encoding putting the string into the queue and looping it back again by calling `BeginReceive`. In order to apply the messages and use data coming from the data gatherer program on the GameObjects, the program depends on a method to process and parse the messages. The function `ProcessMessage` processes each dequeued message from `messageQueue` and parses the JSON formatted string into a ”*JObject*”. The function parses the JSON object to extract the thruster ID, enabling the use of multiple thrusters within the same visualization. Additionally, it processes incoming messages to update pitch, RPM, or azimuth angle values based on the topic from the data gatherer program's UDPClient. These values are then applied to the correct GameObject using the thruster ID to define which GameObject to move the JSON data received by UDP.(see Figure 36).

3.6.4.2 Movement Mechanics Implementation

The visualization depends on a `Movement` class, see code in section 8.3.2 and 35. This class manages the rotations of the propeller on the thruster. Rotations such as, RPM, pitch angle and azimuth angle according to the values received from the UDPClient. The script is designed to smoothly interpolate the values to remove ”choppy” thruster movements. As mentioned in the section above the `Movement` class has the `MonoBehaviour` which enables the class to be a component that can be attached to a *GameObject*, see right side of figure 36 where the `Movement` component is attached to ”*Thruster3*”. Within this class, public variables as shown in the figure above were created in order to apply the GameObjects to the component.

The `Propeller` variable has the *Propell* object attached to it, and similarly, each blade has its own `blade_point` object attached to the `Movement` component. The movement script uses the functions `Start()` and `Update()`, and initializes both the initial blade and azimuth rotations for each individual thruster. The `Update` function performs the interpolation of RPM, pitch, and azimuth values using `Math.Lerp`, linearly interpolating between the current and target values based on the *interpolationSpeed* variable. Furthermore, each value has its own `UpdatePitch()`, `UpdateRPM()`, and `UpdateAzi()` functions to update the variables to their latest values. Additionally, the pitch and azimuth angles have `RotateBlade()` and `RotateAzi()` functions, respectively, which rotate the *GameObjects* along their correct axes, as mentioned in Chapter 3.6.2. These functions are then used in the `Update()` function in the movement class.

To add the `Movement` script to the thruster, a simple drag-and-drop of the `Movement` script component from the project's asset folder was performed. Then, the respective GameObjects were dragged and dropped from the scene hierarchy onto the movement component to apply the thruster movement to a thruster object. Figure 36 in the inspector to the right shows how the `Movement` component was implemented with the thruster.

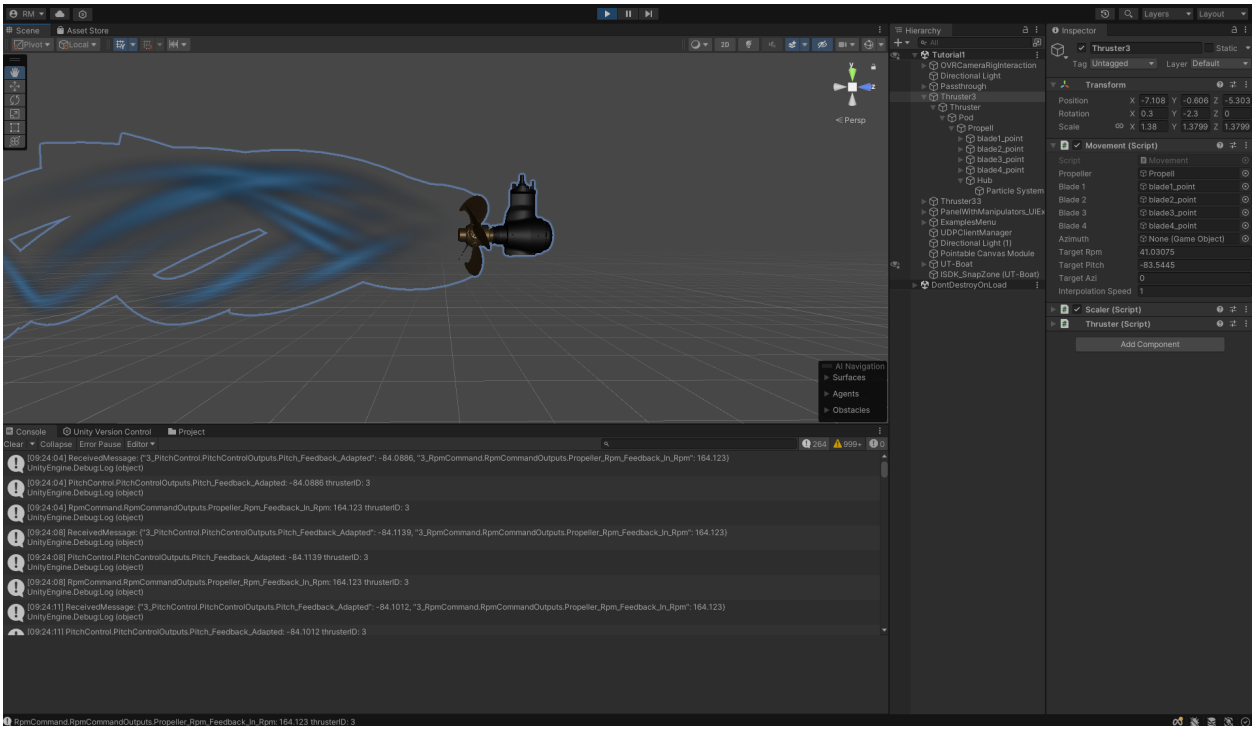


Figure 36: Tunnel thruster running in editor with thrust direction receiving values

3.6.5 Augmented and Virtual Reality Setup

The integration of Augmented Reality (AR) and Virtual Reality (VR) necessitates the implementation and installation of essential software components. According to the Unity Asset Store, the OVR All-in-One package emerges as the most up to date and comprehensive solution for this purpose.

3.6.5.1 Setup of Meta Quest Features

In the early stages of the project the team acquired a virtual reality (VR) headset from NTNU, to develop and test and implement the visualization of the thruster objects in VR. The reason for starting with the Quest 2 headset was that the team had not acquired the more modern Quest 3, but wanted to prove that it was possible to get the values from the data gatherer program in to Unity and to a virtual environment. Shortly after proving that it was possible to visualize the thruster movements in VR using Quest 2, Kongsberg Maritime agreed to acquired a quest 3 headset so that we could develop the AR portion of the project.

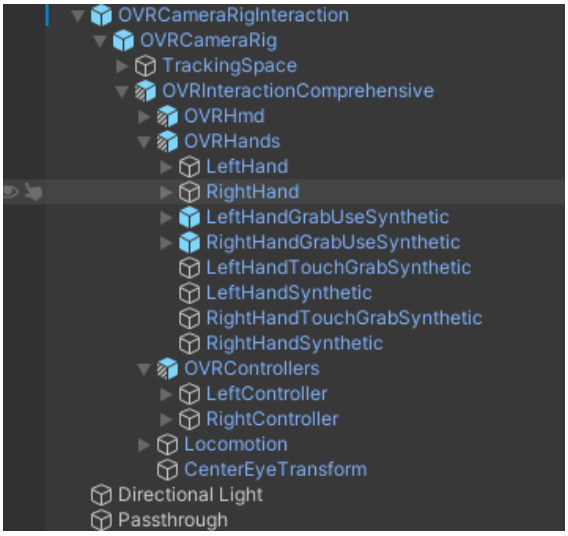
To demonstrate the feasibility of utilizing the Quest 2 headset for visualizing thruster movement, the team followed a YouTube tutorial from the "ValemTutorials" channel [19]. This tutorial served as an introduction to developing the

VR environment in Unity. To implement the VR functionality in to the game scene the ”Meta All In One SDK”, had to be installed from the ”Asset Store”, from Unity’s web-page. Thereafter, importing the asset and applying it on the GameObjects in the scene. At first the team used the tutorials on YouTube as a guide to manually integrate the Meta All-In-One SDK into Unity. These tutorials provided a detailed walkthrough for setting up a mixed reality scene, introducing features like hand and controller tracking and other VR/AR functionalities like for example ”pass-through”. While this method was informative, some of the videos were deprecated. Therefore, it became apparent after developing in Unity that relying on tutorials to utilize the SDK lacked efficiency and scalability. This led the team to explore alternative approaches. Upon discovering Unity’s pre-built tools and building blocks designed specifically for the SDK, the team decided to start using these resources instead of the YouTube tutorials. This change made development easier and more efficient, and minimized the risk of errors in the visualization.

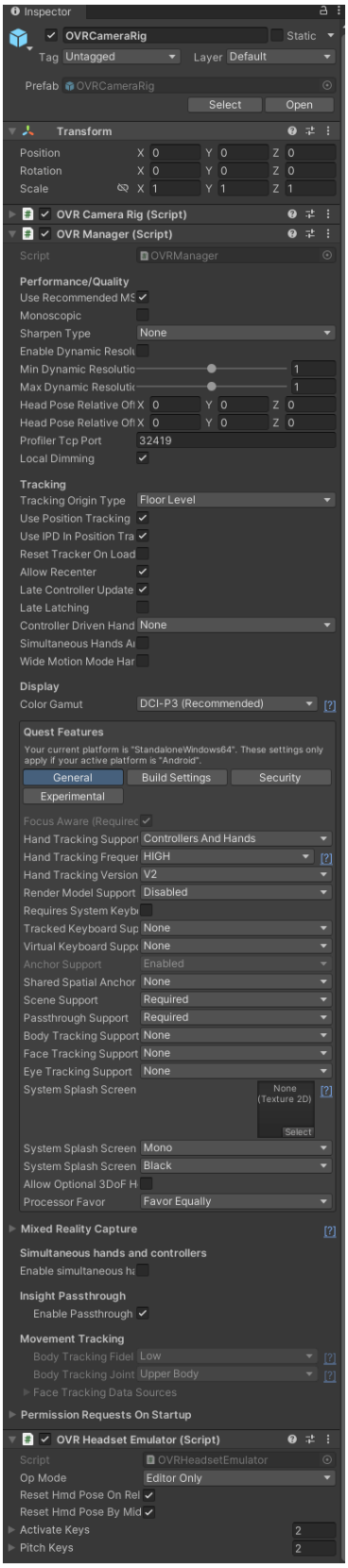
Within the scene hierarchy, see figure 37a, the VR and AR functionality requires a `OVRCameraRigInteraction` block that contains all the logic for the VR and AR functionalities, features like pass-through, hand tracking, and controller tracking the Meta Quest headsets provide. Within this block there is the `OVRCameraRig`, this block sets up the camera rig and manager which is important in order to apply the Meta Quest features within the visualization, see figure 37b. Within the `OVRCameraRig` block there is a block that provides the interaction logic with *GameObjects*, like the thruster within the game scene named `OVRInteractionComprehensive`. This block is the parent to the blocks that contains the Meta controllers and hand tracking features. In order to enable the both hands and controllers tracking features the team enabled them in the OVRManager seen in 37b, making the users controllers and hands to be intractable with other GameObjets. To grab an object, the object must have a ’grabbable’ and ’rigidbody’ component. In this instance, a grabbable component was applied to the *Pod* object because the team wanted to enable grabbing only on the pod, as the propeller was going to be rotating.

3.6.6 Learning about Water Physics

The teams first visualization prototype using Unity was creating an environment using the Quest 2 with a black test environment background and a simple plane to illustrate the ”ground” in the VR environment. After proving that visualizing the thruster movement was possible, the team shifted its focus towards enhancing the visualization by illustrating the ”wash” of the propeller blades to observe the thrust direction on the thruster. Consequently, attention was directed towards incorporating water physics into the visualization. The initial plan was therefore to create a ”pool” of water in the visualization prototype and make the water interact with the propeller blades of the thruster.



(a) OVR Hierarchy



(b) OVRCameraRig Inspector

Figure 37: Meta Quest Features Implementation

Upon the implementation of water physics within the game, it became clear that managing water dynamics was more complex than initially anticipated. Resulting in a time consuming research and test period to figure out the different methods of implementing water dynamics to the visualization. The first methods the team found for implementing water dynamics was either to create a ”water shader” ourselves, or download and import a water ”asset” from the ”asset store”. After considering these options, the decision was made to use a water asset from the asset store, recognizing its benefits. However, the cost associated with these assets redirected our focus towards seeking other alternatives. The team found another free alternative called ”HDRP Water Sample Scenes” from Unity-Technologies, available on their GitHub page [18]. By following the readme instructions, downloading, and importing the sample scenes into our visualization project, water dynamics could be implemented. Consequently, placing the thruster under water to test if the water would interact with the propeller blades, the outcome was unexpected. The water did not move, splash or respond in accordance with the propeller movement and thrust direction at all. This discovery led the team to realize that particles were needed to effectively illustrate the movement, as shown in figure 38.

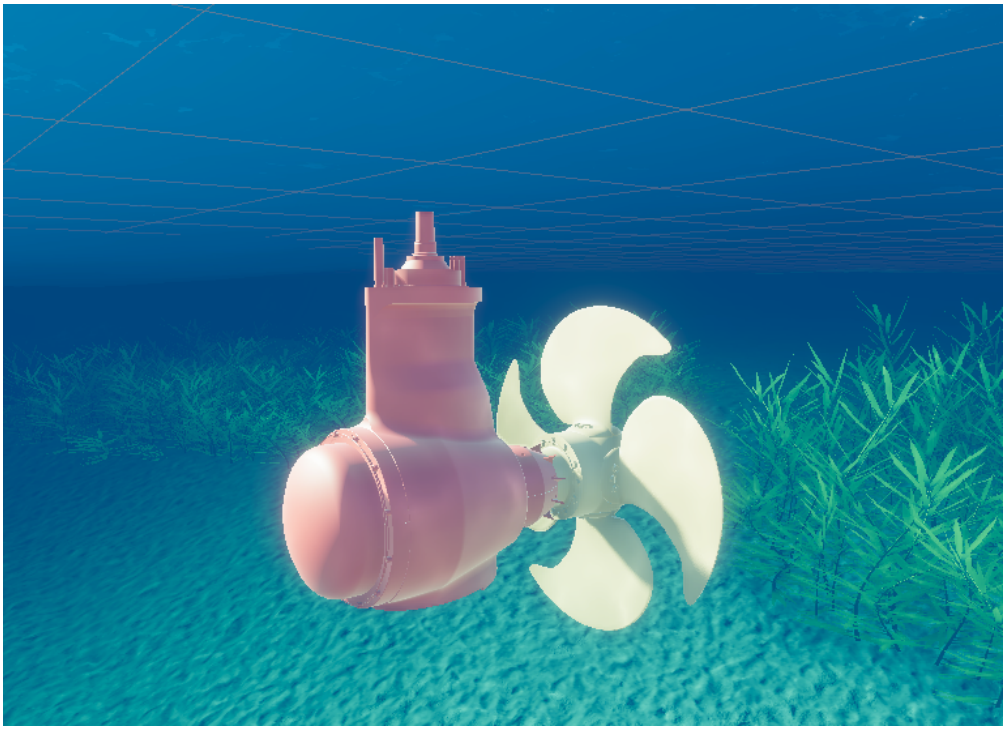


Figure 38: Thruster Under Water VR Using Quest 2

Unity features a robust Particle System where you can simulate moving liquids, smoke, clouds, flames, magic spells, and other effects [11]. In the visualization, the particle system uses the cone shape to emulate the thrust from the propeller, but in order to get the thrust direction shape both negative and positive direction, the angle on the cone shape was modified into a ”cylinder”. Directing particles outwards or inwards in the appropriate direction, as illustrated in Figure 36.

To ensure the particles accurately represent the propeller’s thrust direction, a script was developed to adjust the particle system’s orientation based on the thruster’s pitch angle. This script dynamically aligned the particle emission with the thruster’s movement changing the rotation and speed of the particles, look at the code in Section 8.3.3.

The `PropellerParticleSystemController` class uses the movement script, mentioned in Section 8.3.2 and 3.6.4.2, as a public variable and refers to the modules in the particle system as private variables in order to move the particle system according to the thruster's movement. The *Start()* method is called on the frame before any of the *Update* methods are called for the first time.

The particle controller script moves the `propellerParticles` by acquiring the *ParticleSystem* component attached to the same *GameObject*. Additionally, it then checks if the `propellerParticles` component and `movementScript` are properly assigned within the Unity scene, and if not, it logs an error and disables the script to prevent further execution errors. The most important method to make the particles follow the propeller blades' movement is retrieving the modules `velocityOverLifetime` and `shape` from the `propellerParticles` in order to adjust these modules to move simultaneously in accordance with the pitch value coming from the movement script.

3.6.7 Transitioning to Augmented Reality

Transitioning to augmented reality from the established virtual reality setup proved to be straightforward, thanks to the preliminary work done with Meta Quest features, mentioned in Section 3.6.5.1. By enabling the pass-through functionality, the team was able to shift from a black background to a transparent one, effectively removing the need for the "HDRP water sample" scenes and visualizing an environment for AR. This process involved adjustments to the existing VR elements to accommodate AR-specific requirements, enhancing the immersive experience by integrating real-world elements seamlessly with the virtual thrusters. Mentioned in section 3.6.5.1, the OVR tool within Unity was helpful to easily convert the existing VR visualization in to a augmented reality environment.

3.6.8 Developing the UT-7128 Ship Design Scene

Similar to the import process for the tunnel-thruster object, additional editing was required within the game scene during the import of ship objects in the game environment. Given that both models of ships were acquired concurrently, the team opted to prioritize the UT-7138 design (figure 39), which was the latest of the two designs. The BB-Octopus model was retained as a supplementary option or backup, providing flexibility for potential future use or requirements. As mentioned in chapter 3.6.2, it was also necessary to edit the game object within the Unity editor. Unwanted objects that did not have any relevancy for the ship were removed and the two orange "Davit" lifeboats positions were moved from the bridge to their respective position on the side of the ship's hull. Subsequently, the team incorporated the tunnel thruster model obtained from Ulsteinvik, substituting it for the boat's existing thrusters. This replacement was necessary because the initial thrusters of the UT-Design lacked segmented blades, resulting in all attached thrusters being fixed pitch. Additionally, the team also replaced the original azimuth propeller with the already made tunnel thruster propeller blades, opting for simplicity, despite it not being entirely accurate, since creating a new 3D-object would not align with our team's intended scope of the project.

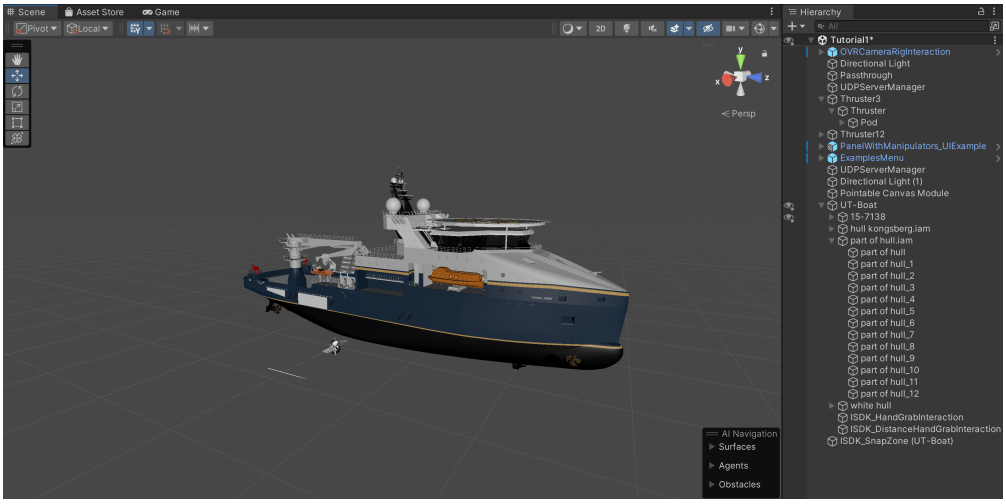


Figure 39: UT-Boat in Unity Editor

3.6.8.1 Adding Multiple Thrusters

In order to add more thrusters to the scene in Unity the team simply copied and pasted the tunnel thruster and applied it to the UT-Design as mentioned in the previous chapter. In chapter 3.4.4, one can see how the parsed message coming from the OPC server looks. It sends the thruster ID first in the message, this way when multiple thrusters are sending values that have the same topic name to be able to differentiate between the values. Thereafter changing the name of each *ThrusterObject* to their respective ID number. So for example, a thruster with id $\bar{3}$ the name of the object needs to be named ”*Thruster3*” to be able to move the object, look at figure 36.

3.6.8.2 Azimuth Movement

When implementing the multiple thrusters, the azimuth rotation also had to be implemented. As mentioned in 3.6.4.2 the `Movement` class moves the azimuth thrusters according to messages coming from the data gatherer server. But to get the azimuths to rotate the correct way some changes to the scene hierarchy had to be made. These changes were applied to the azipulls on the UT-Ship design, but could also be applied to the TCNS located towards the bow of the UT-Ship.

Changes such as creating a new GameObject and positioning the centerpoint to the correct position aligning with the azimuths centerpoint. Thereafter, changing the GameObject name to ”Thruster” + ”ID” was necessary to apply the movement to the object. Next adding the newly created GameObject to correctly placing it in the hierarchy to be a child object of the UT-Design. Thereafter, removing the original blades as mentioned in 3.6.8 and adding the tunnel thruster propeller to the azimuth object, look at Figure 40. In addition to placing the azimuth object as a child within the new created *ThrusterObject* to ensure a correct hierarchical structure.

Figure 40: Azimuth Thrusters on UT Ship Design

3.6.9 Unity Version Control Problem

To more easily share the visualization project between computers, Unity's cloud based version control was utilized. This addition to the project had the unfortunate effect that the communication stopped working. At first, the group believed this was because we lacked an understanding of the tool. We therefore embarked on the lengthy process of double checking every setting, code component and the 3D models. Due to some misunderstandings with the version control tool, many of these seemed to have changed. The group changed these back to the functioning versions manually, but the communication was still not working. Because of this, the group started to suspect the IT-department at "KM"(whos offices we use). The team tested with personal computers and a direct ethernet connection. When this did not work, we experimented using different ports which was the solution.

3.6.10 Making the Unity Visualization More Accessible

The team wanted to make the concept more user friendly, and a factor to making it more user friendly is to make the program easy to launch and use. Therefore, making a executable instead of having to use the Unity editor to start the program seemed to be beneficial. In order to make a runnable program, the team had to go to the "build" and "run" settings within Unity and switch the platform to Windows. Thereafter, building the program and placing it in a desirable folder or creating a "Runs" folder on the computer. This made program one that was possible to run directly

from the desktop on the Quest Headset when connected via ”Quest Link” on the computer.

3.6.11 HUD-Panels Integration

A plan to implement Head-Up Display (HUD) panels was initiated by utilizing pre-existing assets from an example scene within the Meta All In One SDK toolkit. The team proceeded to integrate these panels into the scene, ensuring they were interactive and capable of being grabbable. This served as an illustrative demonstration of the potential appearance of the panels. However, due to time constraints, the team was unable to implement additional features such as the ability to spawn/delete objects or display values directly on the panels.

4 Results

The final product enhances the communication program by providing adaptability and improved robustness. Data from the Common Application Framework (CAF) is visualized by overlaying data on both software block diagrams and hardware drawings, through the Ignition platform. Furthermore, the thruster movements are visualized in augmented reality (AR) to provide a realistic illustration of the system's scale and operational mechanics. This implementation establishes a foundation for future innovations and improvements, including the integration of exploded views, AR flowcharts, and other advanced visual aids.

4.1 Communication Program

The current communication program is designed for dynamic operation, supporting multiple thrusters and Graphical User Interface (GUI) connections concurrently. It has been successfully tested with up to four thrusters and works with both the REST API client and the websocket client, integrating with both Ignition and Unity visualization platforms. After checking that both cliens work, most of the testing and demonstrating has been done using the websocket client.

When using the websocket client, the amount of topics gathered per thruster is limited to 100 because of server limitations, as explained in chapter 3.4.2. Configuration is facilitated through several files: `connectionData`, `topics` and `endpoints_restapi`. The *connectionData* file, structured in JSON format, is used to list essential elements such as connection information, including URLs and IP addresses. This format simplifies and enhances the efficiency of reading, parsing, and editing data. Meanwhile, the `topics` file is straightforward, with one topic per line. These files specify the topics to extract from the thruster and the ones to send to the Unity visualization program. The `endpoints_restapi` file lists the endpoints to use when gathering data with the REST client.

When using the REST API client, there is no limit to the number of topics that can be requested. However, it takes approximately 0.4 seconds (see Figure 15) to retrieve data from a single sub-object, which contains a limited number of topics. Unfortunately, our testing and the API specifications indicate that there is no way to receive batches of multiple objects simultaneously. Additionally, when requesting larger objects, only metadata about the object is received, rather than the full data set.

An additional configuration file has also been implemented to a limited degree. This file contains parameters such as ”max_retry_attempts” and ”retry_delay”, to allow some configurations to be made by the user without editing the code directly. How to use these configurations and which parameters are currently in use, are further explained in the programs *readme* file.

Robustness features have been incorporated to enhance system reliability. If a thruster specified in `communi cationData` is not detected, the program logs this issue to the console and continues to connect to the available thrusters while maintaining operational stability. Similarly, if a topic from the `topics` list is missing, this discrepancy is noted on the console, but the program remains functional, ensuring uninterrupted performance. However, despite these measures, the robustness occasionally fails, causing the system to get stuck in a repeating loop and sometimes crashing the control system and CAF.

Given that the program is built using an object-oriented approach, it offers flexibility and compatibility across different versions and systems, if the necessary components are created. This architecture allows for both backward and forward compatibility and can be easily adapted to interface with various systems through the addition of new client modules designed to communicate with the appropriate data servers. This method streamlines maintenance and simplifies the process of adapting the program to accommodate new technological environments. However, better

encapsulation of the various components and functionalities could further enhance the maintainability and scalability of the program, making it easier to manage and extend.

The current websocket module implemented in the program assumes that topics in the topics list are formatted as follows: +1:GuiInterface_1.GuiFeatheringMode.Feathering_Status, where ”GuiInterface_1” is the object, ”GuiFeatheringMode” is the subobject, and ”Feathering_Status” is the variable to be retrieved.

Similarly, the REST API module utilizes a text file akin to that used in the websocket module to determine which objects to retrieve. However, the endpoints are formatted as: PitchControl/protos/PitchControlOutputs/fields, where ”PitchControl” is the object and ”PitchControlOutputs” are the subobject. The system then parses and uploads all data within the subobject to the OPC-UA server.

4.2 Using Ignition to Visualize Live Diagrams

The GUI created in Ignition displays a system with max configured software block diagrams with data from CAF as an overlay and is mostly the same as the project baseline, shown in Figure 44, 45 and explained in section 1.3. In addition we have now implemented live system drawings that show the current states of connections such as IO modules and CAN connections. The initial goal was to introduce two-way communication to enable the Ignition GUI to change parameters on CAF, but this was never successfully implemented.

To enhance the live software block diagram visualization using Ignition, the team concentrated on leveraging the new OPC-UA server object structure. This integration allows for improved interaction with Ignition. Refer to Figure 41 to view the thrusters within the system’s tag browser. This works, but appears to be a bit slower in real life when updating the topic values than the first prototype because interpolation has not been made in the new server. Despite having slower updated topic values using the new data-gatherer for Ignition. The new prototype is much more stable and supports multiple thruster integration which is beneficial for training purposes. The result of using Ignition gives a training program that serves scalability and ease of use for developers to expand the program if wanted.

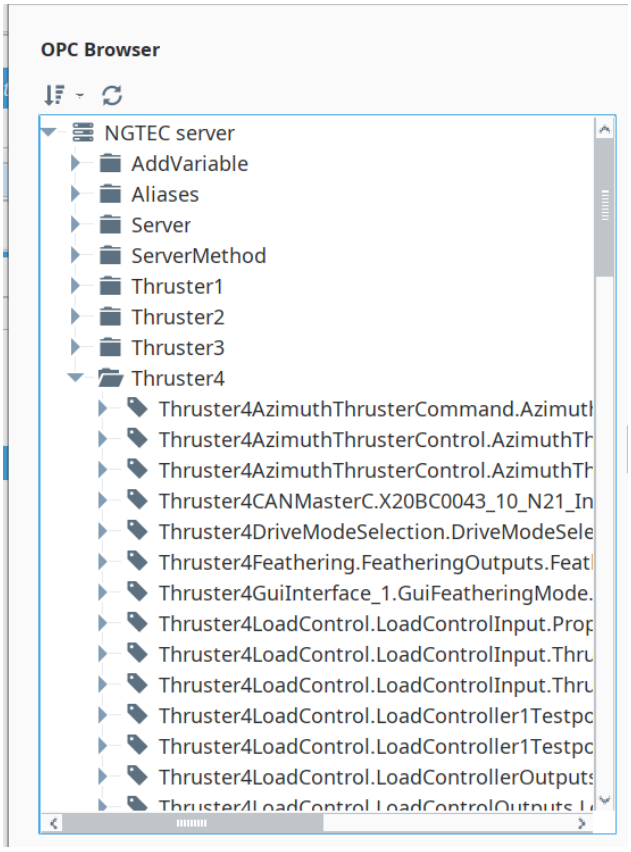


Figure 41: New OPC-UA server shown in Ignition OPC browser

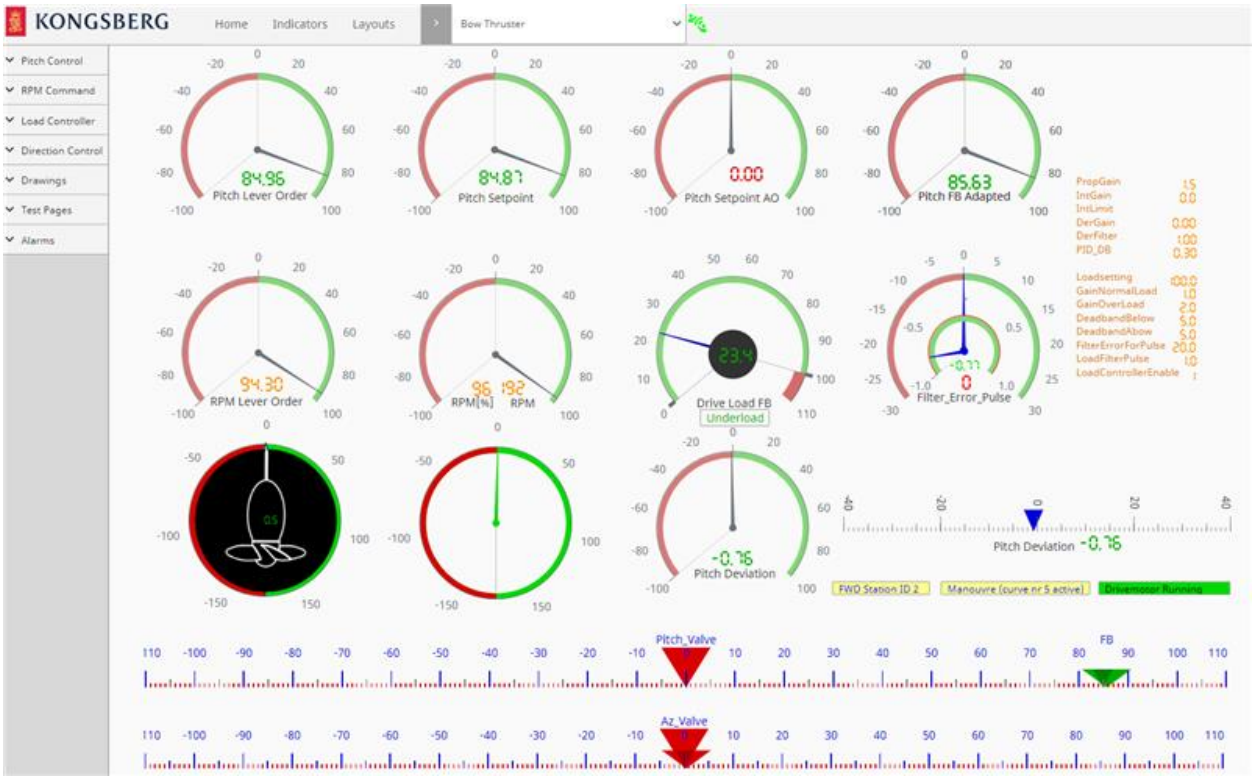


Figure 42: Indicators in the Ignition GUI

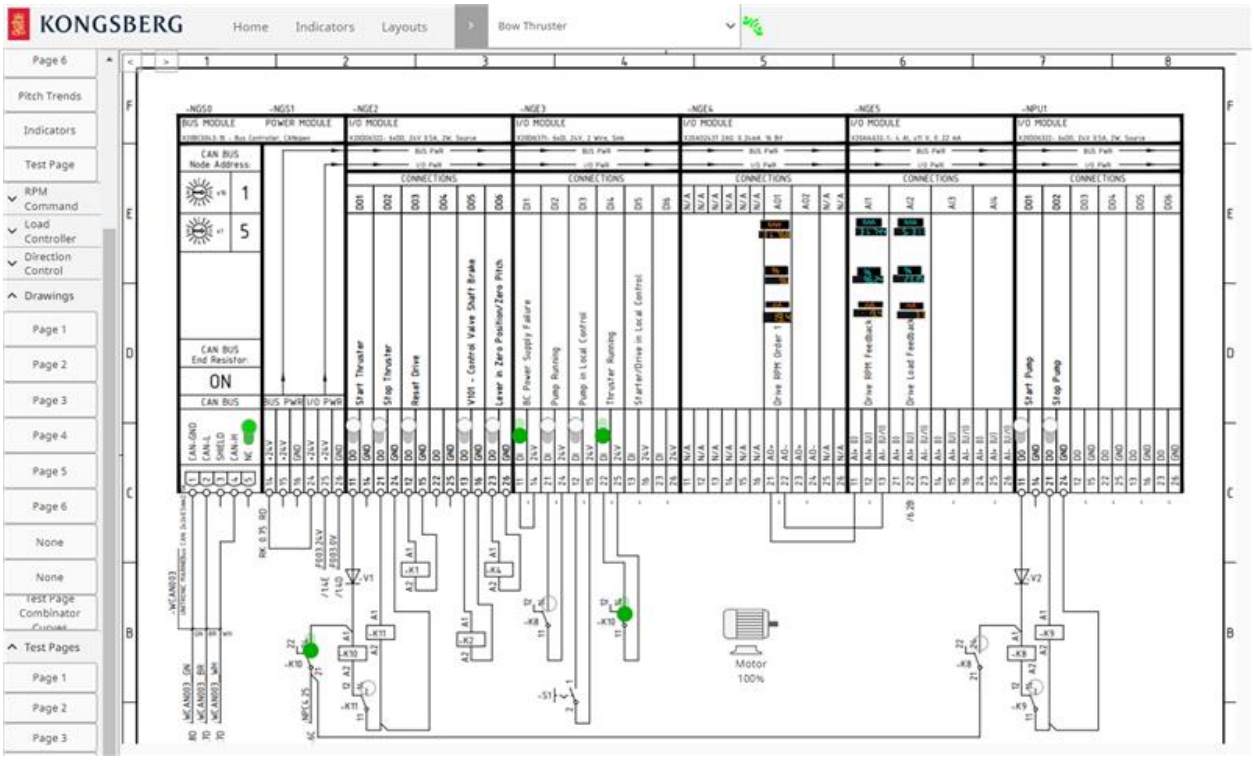


Figure 43: Real-Time Technical Drawings with live indicators and switches on IO-Modules

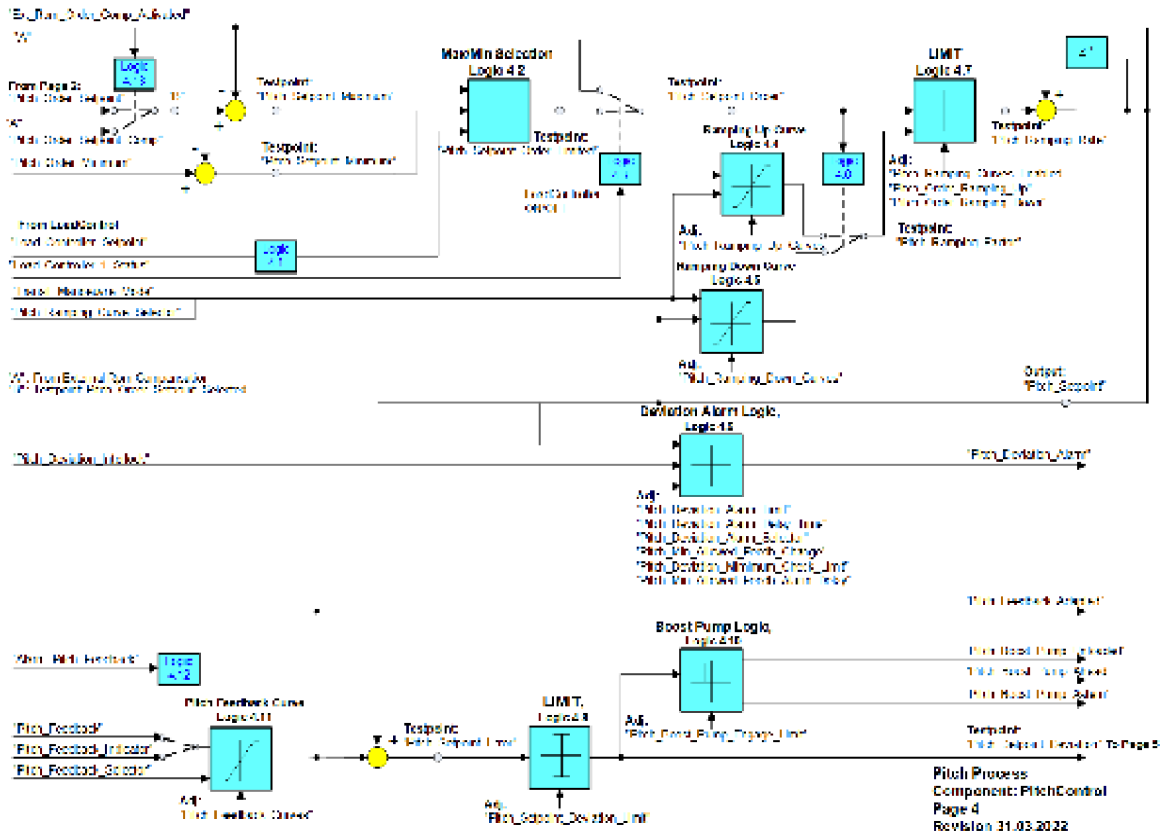


Figure 44: Pitch Control Page Ignition, *Blurred due to confidentiality*.

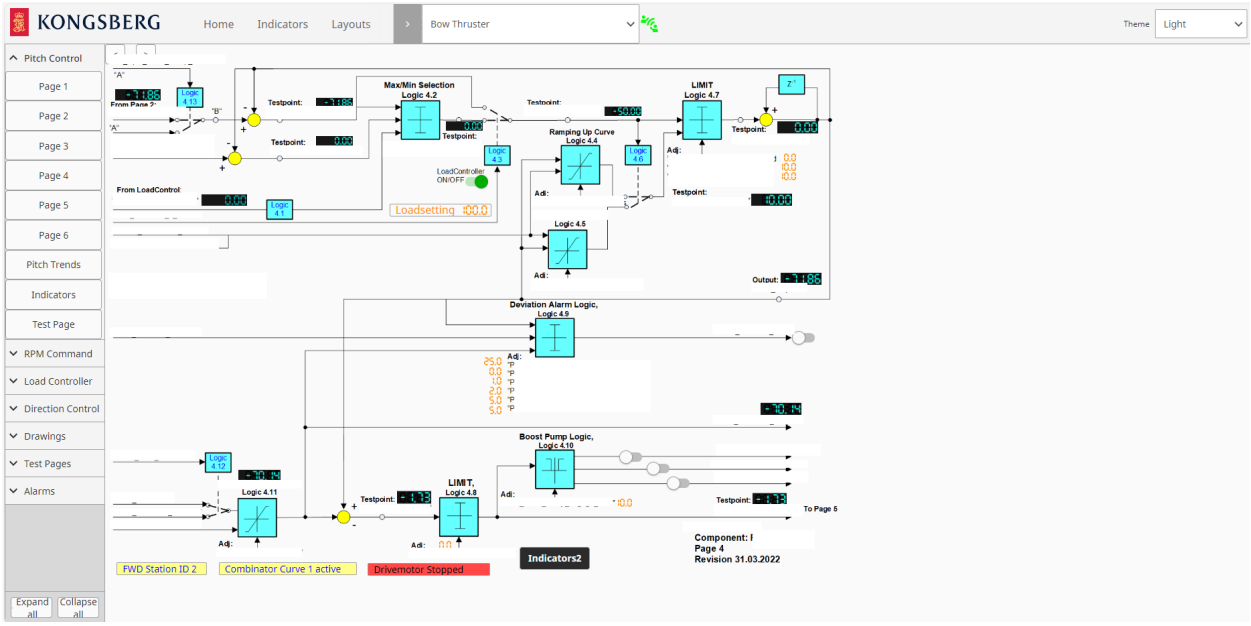


Figure 45: New Pitch live SWBD GUI, *Blurred due to confidentiality*.

4.3 Augmented Reality in Maritime Training

As a part of our project's aim to enhance the training environment for Kongsberg Maritime systems, Unity was utilized to create an immersive and interactive 3D visualization in augmented reality. Utilizing Unity allowed the team to have access to the comprehensive visualization tools within the program, such as its advanced rendering capabilities and support for mixed reality allowing the development of a detailed and responsive thruster visualization. This section will discuss the substantial results achieved through the implementation of Unity, reflecting on if the visualization improved overall system understanding and interaction with Kongsberg Maritime's control systems.

The use of the Unity game engine enabled the project team to visualize complex system data into a visually intuitive format, facilitating a more effective and immersive learning experience for users. Applying real-time movement from a marine controller to a 3D Thruster has primarily benefited overall system understanding and learning, particularly for new employees and those lacking in-depth system knowledge. For example, individuals who are unfamiliar with the pitch on the propeller blades, the correlation between RPM and pitch, and the scale and size of a real thruster can greatly benefit. The finished visualization is shown in Figures 46, 47, and the video provided in the appendix. The visualization in Unity also allows the users to directly see the movement of the thrusters enabling the possibility to study the movements in case of deviations or faults within the hardware/software. Additionally, the visualization gives the instructor the opportunity to more effectively explain details and movements the control system might have when operating in a certain way to their participants. This integration serves as an invaluable tool for enhancing comprehension, providing practical insights into system dynamics. It allows employees to visually understand adjustments that directly the affect thruster behavior, bridging comprehension gaps and accelerate the learning process.

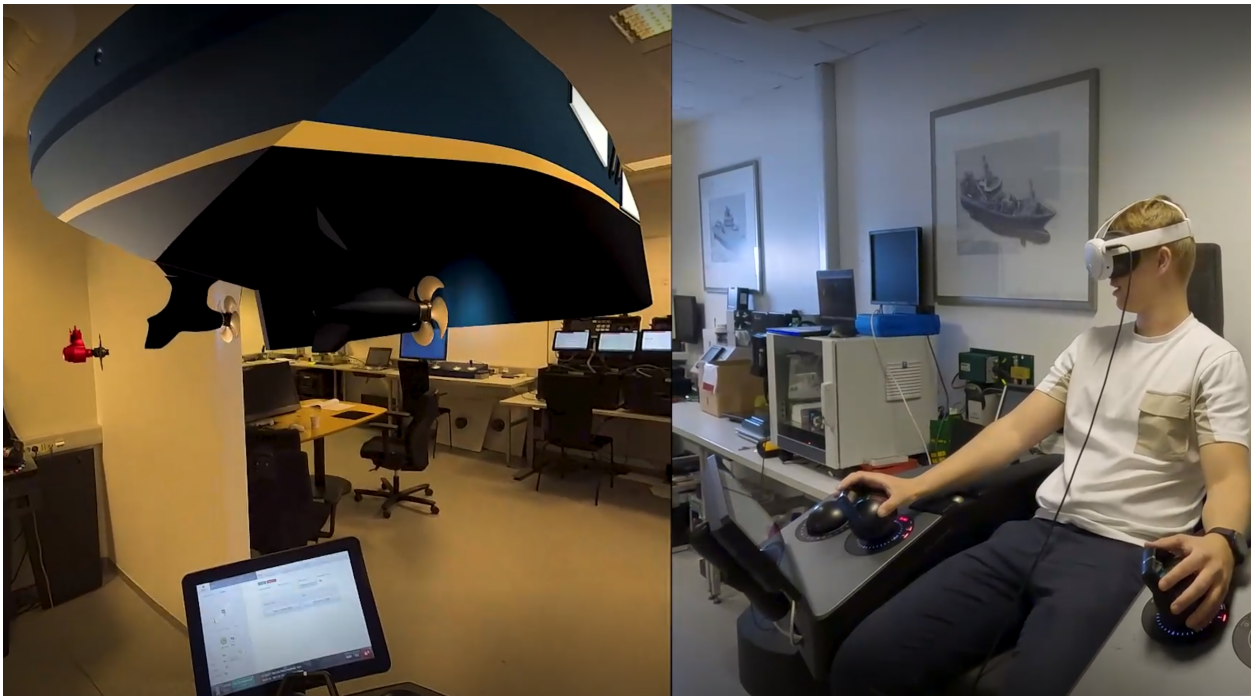


Figure 46: Result of AR visualization

Figure 47: AR Visualization of Thrusters in Action: Real-time displaying active thrusters controlled by a system, complete with dynamic thrust direction indicators.

4.4 Qualitative Findings

This section presents results derived from qualitative analysis of interviews and observations conducted during the thesis. Through thematic analysis, key themes were identified, highlighting current challenges and opportunities within the existing training environment.

1. **Need for Interactive Learning Tools**
   Questioning around 50 employees at Kongsberg Maritime Ålesund revealed a consensus on the benefits of the new interactive training concept. They emphasized its intuitive access to information and data, crucial for system comprehension. Geir Olav Otterlei, VP Services & Technical Support, and Per Magne Dalseth, Technical Engineer and Instructor, underscored the potential benefits for the company, suggesting that an interactive program could enhance employees' system understanding, thereby boosting efficiency and overall operations.

2. **Testing Concept During Training Course**
   Per Magne Dalseth incorporated the new training concept, featuring real-time software block diagrams (SWBD), during a course. This adjustment resulted in notably positive outcomes, particularly in re-engaging participants who had previously struggled with traditional methods.

3. **Feedback and Utility of the Product**
   Feedback from service engineers, training instructors, and participants indicated strong potential for the product in training. Of the eight individuals interviewed, five were mainly positive, highlighting improved engagement and understanding due to the features in the visualization. One participant was highly enthusiastic about the current features and the future possibilities the concept could offer. However, one participant expressed a negative view, not towards the product itself but regarding the typically lengthy setup time required for solutions like these. Despite the communication program occasionally crashing the control system, two service engineers expressed interest in utilizing the solution during service missions, suggesting substantial promise with continued development.

4. **Augmented Reality (AR) in Training**
   While AR visualization has been showcased in courses, participants sometimes prioritize the experience itself over detailed thruster movements. Experimentation with scaling adjustments has shown some improvement, but enhancing functionality and user-friendliness could further enhance effectiveness. More than five sales representatives have also shown interest, indicating potential utility for sales, with one expressing explicit interest in using it for an exhibition scheduled for September in Iceland.

Overall, the findings in this analysis signify that the product has potential in multiple departments at Kongsberg Maritime (KM). However, it is important that the solution is easy to set up to ensure its successful implementation and widespread adoption.

5 Discussion

Completing this task has been quite demanding, requiring us to determine the best approach to achieve our objectives. This challenge led the project group to delve into a significant amount of new subject matter, resulting in a steep learning curve, particularly regarding Kongsberg Maritime systems. The process of acquiring the necessary experience and knowledge was both frustrating and rewarding. Reflecting on the project execution, the team recognized that the knowledge gained benefited us from the start. This section discusses several issues and possibilities encountered during the project.

5.1 Communication Issues Caused by Unity Version Control

After resolving the communication issue caused by the Unity version control, mentioned in Chapter 3.6.9, the group developed several theories about why the problem occurred. Since changing the port number from 5000 to 5002 seemed to fix the issue, the group believed it was highly likely that the version control system uses port 5000 for its cloud communication. Upon further investigation, the group discovered that port 5000 is often a default port, suggesting that port numbers should be chosen with more consideration. Furthermore, the group concluded that using port 5000 initially was not ideal and that selecting a different port from the start could have prevented such issues.

5.2 Stand-alone AR Visualization on Headset

The team considered building the AR visualization as a stand-alone app to be used on the headset without requiring a connection to a PC. However, this idea was abandoned for several reasons:

- Strict Wi-Fi connection in the development area.
- Difficulty assigning a static IP address to the headset.
- Wi-Fi was not fast enough when using hotspot on Phone.
- A wired connection with the quest link was preferred during development due to limited battery life.

5.3 Interpolating

Since the communication program only receives data once per second from CAF, the group considered interpolating the data before adding it to the server to avoid choppy animations on the visualization platforms. However, this approach was not implemented for two main reasons:

1. Simple interpolation would introduce additional time delay, making the system less real-time.
2. Significantly more data would have to be transferred to clients.

This means that the visualizations using data from our server would have to interpolate data locally if necessary.

5.4 Data Gathering Issues

As explained in chapter 3.4.2, we are currently limited to managing 100 topics concurrently. Potential improvements were considered but have not yet been successfully implemented. These include using a combination of Websocket and REST API, asynchronously retrieving multiple topics per Websocket connection, or utilizing an MQTT server, which would require an additional micro marine controller.

During our attempts to circumvent the limitation of 100 concurrent Websocket connections, there were instances where the system appeared to function correctly, only to later reveal significant issues like data being assigned to the wrong topic. These partly successful attempts suggested that it might be possible to request multiple topics per connection, despite the server's response rate of once per second. Although promising, we have yet to pinpoint the issues and have decided to maintain the 100-topic limit for this prototype.

Additionally, we explored various methods for making batch requests. These methods included requesting a list of topics, using comma-separated topics, formatting topics in JSON, and modifying the identifier from "+1:" to "+2:", with commas separating the two topics. Most of these attempts yielded no results. However, changing the identifier initially seemed promising as it appeared to support two simultaneous requests, potentially allowing us to double the number of topics received. Subsequent attempts to implement this functionality consistently resulted in only the first topic being received. This inconsistency raises concerns about either the reliability of our initial testing or possible changes in the versions of our libraries or the CAF system. Regrettably, the code from these tests was discarded, and further tests using the Simple Web Socket Client (a Google Chrome extension) were unsuccessful in replicating the earlier results regarding batch requests.

5.4.1 Why two-way communication was not implemented

Two-way communication was initially deprioritized because we focused on retrieving all data from the server first. Unfortunately, this approach was time-consuming and ultimately unsuccessful. After abandoning the websocket and REST API methods for data retrieval, we attempted to implement load simulation as described in section 3.4.6, but since a load simulator was dependant on two-way communication the implementation of the load simulator was unfortunately not created.

5.5 Visualization

5.5.1 Water Physics

Focusing on implementing water in Unity proved to be a time-consuming task, which slowed the progress of the project. Through this focus, we realized that water implementation in games is often simplified and does not encompass all the physics and dynamics of real-life water. Features like buoyancy, ripples, and effects are often implemented as component scripts applied to game objects, making them behave like water rather than actually interacting with game objects. Implementing real water physics and dynamics, where water interacts with other objects, would introduce a heavy load on the GPU processing power because the water would consist of millions of tiny particles acting as fluid. This challenge is present in Unity, but some game engines, like Unreal Engine, have implemented more advanced interactive water systems.

Although we considered transitioning to Unreal Engine to utilize its water system, we decided to stick with Unity due to the time required to learn a new game engine. However, if our focus had been on enabling the thruster to be

dipped in a pool and interact with water, Unreal Engine might have been a better choice due to its more optimized water interaction capabilities. Consequently, we decided to focus on other aspects of the project. As a result of this decision, the project's particle system was introduced, implemented as explained in section 3.6.6. Refer to Figure 36 to see the thruster running with particles flowing in the thrust direction.

5.5.2 Enhancing User-Friendliness in AR

A challenge the team encountered while visualizing in AR was making the visualization user-friendly, especially considering that the users of the solution will often be individuals who have never tried VR or AR before. Making the visualization intuitive for everyone to use was harder than expected. Some users were naturally adept and understood how to use the AR solution, while others needed some practice before getting accustomed to the technology. Although some users needed practice, all individuals testing it achieved a basic understanding. Taking this into consideration, the AR visualization could have been further optimized by improving user-friendliness, but it works nicely as a demonstration of the potential.

5.5.3 Positive Feedback and Future Potential

Despite the challenges, the team received enthusiastic feedback from KM employees and training participants. This validation highlighted the system's ability to make complex data accessible and engaging, reinforcing its value in training environments. KM's interest in using the solution for training and exhibitions, including an upcoming event in Iceland, underscores its practical utility. Additionally, the feature allowing interaction with multiple thrusters enhanced the system's functionality and provided a more comprehensive training tool. The positive reception and demonstrated potential suggest that continued development, focusing on user-friendliness and expanded functionality, could further increase the system's impact and utility across various departments at Kongsberg Maritime.

5.5.4 Autogenerated Software Block Diagrams in Ignition

An enhancement to the Ignition visualization would involve developing a method to automatically configure the SWBDs and technical drawings by utilizing the configuration file for a vessel. This would enable the program to be utilized in the field on a vessel. Presently, the program is only configured with the software block diagrams of a maximally configured vessel, where many of the blocks are not utilized for specific vessels, particularly for the pelicase.

To make the product useful in field applications such as fault-seeking or commissioning, in addition to training, implementing auto-generated software block diagrams (SWBDs) is essential. This feature would significantly improve the application's usability and adaptability, allowing it to meet the specific needs of different vessels and scenarios, thereby enhancing its practical utility and effectiveness.

However, auto-generated SWBDs were not implemented due to the project's time constraints and the complexity of developing such a feature. Creating auto-generated diagrams requires sophisticated algorithms and extensive data integration, which were beyond the project's current scope. Future development could prioritize this feature to fully realize the potential of the application in diverse operational contexts.

5.6 Different Use-cases for the Solution

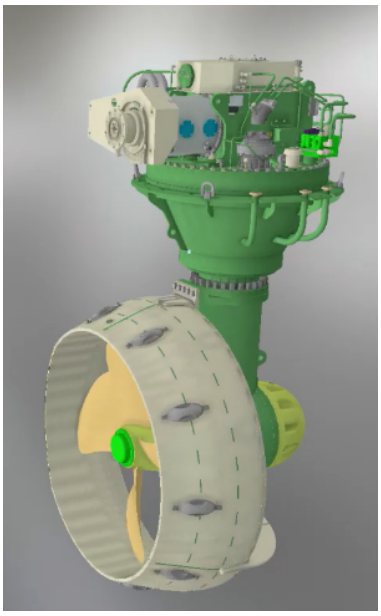
For complete newcomers we believe that the AR visualization is the most helpful in comprehending terms like ”pitch,” ”rpm,” and other technical terminologies. It provides a visual representation that aids in understanding the size and appearance of various components, particularly the thrusters, offering a tangible sense of their size.

On the other hand, the SCADA visualization serves as a valuable tool for teaching the system as a whole. It facilitates a deeper understanding of signal flow, system parameters, and overall system functionality. By visually illustrating the utilization of blocks within the system and their respective functions, the program developed in Ignition proves to be a superior choice for comprehensive learning. But using both programs in combination provides a greater view of the system.

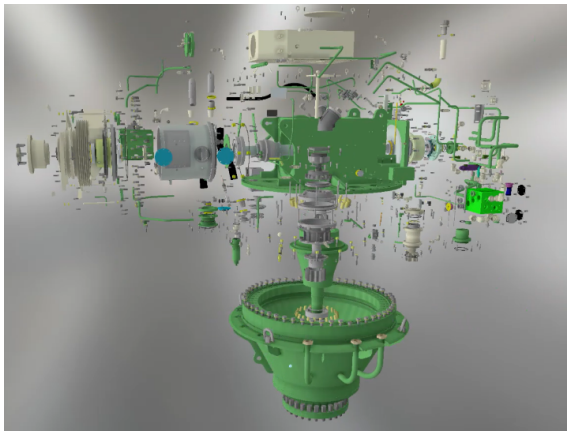
5.6.1 Future Possibilities

As discussed in the report, this project showcases several exciting opportunities. Throughout the development of the program, we have envisioned numerous enhancements that could significantly elevate our product. These potential improvements include, but are not limited to:

- An exploded view of complete equipment in motion using augmented reality (AR), such as a thruster with its components like gears and pinions dynamically illustrated, look at Figure 48.
- A fully interactive bridge in AR, equipped with operational handles and levers, enhancing user interaction and realism.
- With further development in robustness and comprehensive testing, the live software block diagrams could be effectively utilized during commissioning and service missions.
- Auto generated SWBD’s and topics by using vessel specific configuration files.
- Creating more client modules to support more CAF versions and potentially other systems.
- Make it possible to switch between communication modules using the configuration files or a GUI.
- Create a HUD-display for users to more easily interact with the AR visualization.
- Make a ”how to” guide that prompts new users in the AR visualization. Can be integrated within HUD-menu.
- Create a pool for the users to ”dip” the thruster in the water and make splash effects for seeing the thrust direction.
- Improve Grabbing logic.



(a) Thruster before exploded view.



(b) Exploded thruster view showing components.

Figure 48: Thruster example in AutoCAD demonstrating how the exploded view would look. Courtesy of Kongsberg Maritime.

These enhancements could significantly broaden the utility of the program across various divisions within Kongsberg Maritime, including sales and commissioning. In sales, the ability to demonstrate the product using an AR headset or a PC could revolutionize client presentations by displaying objects like thrusters, ship designs, and bridge consoles in an immersive and interactive manner. For commissioning, the detailed overview provided by the program can enable technicians to more efficiently verify system data and diagnose faults, thereby improving operational efficiency and reducing downtime.

To effectively implement these advanced features, there are critical considerations regarding the development capabilities required. The project team currently lacks expertise in complex game development engines like Unity, which are essential for creating sophisticated AR experiences. There are two primary paths to address this gap:

- **Training the Existing Team**: Invest in comprehensive training for the current project group to learn Unity or another suitable game engine.
- **Hiring Specialist Developers**: Alternatively, the company could hire experienced game developers who already possess the necessary skills in these technologies, such as those from Morild Interactive AS.

Additionally, the company could consider passing the project on to the next group of students. These students could build upon our current findings to potentially enhance and expand the project's capabilities further.

5.7 Evaluating the Qualitative Research Methods

In our research on effective methods, a qualitative study was conducted to explore potential improvements in the learning experience. One key finding was that the team should have conducted a study in the early stages of the project

to measure the overall learning outcomes when using the product. To enhance the training experience, a proposed approach could involve implementing a two-part methodology:

- **Initial Baseline Study**: Conduct a study at the beginning of the project to establish a baseline of participants' knowledge and skills. This would involve pre-training assessments and interviews to understand their current level of understanding and identify specific areas for improvement.
- **Post-Training Evaluation**: After implementing the training program, conduct follow-up assessments and interviews to measure changes in knowledge, skills, and overall learning outcomes. Comparing these results with the initial baseline data would provide valuable insights into the effectiveness of the training and highlight areas that need further enhancement.

By adopting this two-part methodology, the project team could have more effectively measured the impact of the training program and made data-driven decisions to optimize the learning experience. This approach would also allow for continuous improvement, ensuring that the training remains relevant and effective for participants.

6 Conclusion

This thesis has provided valuable insights that will benefit the project group in future projects. Despite our best efforts, some decisions and methods used during this project may not have been optimal.

Our communication solution is notably more stable than the initial prototype. It supports multiple thrusters with up to 100 topics per thruster and a one-second delay. While we are satisfied with this improvement, further expansion to handle more topics is necessary for a comprehensive system visualization. Additionally, our communication program demonstrates stability by managing short disconnections and minor configuration file issues, thereby enhancing reliability.

In terms of robustness, the communication program is dependable when configuration files and connections are correct. However, issues can arise if a connection is faulty during the program's initialization.

Regarding visualization, we are pleased with the outcome, especially given our limited prior experience in game development. The visualization accurately depicts thruster movements, thrust directions, and a UT-Ship design in augmented reality. It effectively communicates with the marine controller through the communication program, allowing thruster movement in response to lever adjustments.

The SCADA visualization using Ignition successfully retrieves data from the OPC server in the communication program and displays the values. Although it functions similarly to the previous project, we have improved the GUI and added functionality to display multiple thrusters simultaneously.

In conclusion, our solution significantly enhances the training environment, particularly with the software block diagrams proving to be the most beneficial. We believe that further development by someone with game development expertise could make the 3D visualization an invaluable tool, especially if complete models of the system, including internal components like tubes and gears, are integrated. For future development, expanding the scope to include other systems beyond thruster controls—such as steering, sails, automation, and more—could enable a comprehensive visualization of the entire vessel and its subsystems.

7 Recommendations

For those interested in advancing the development of our training environment, we offer the following recommendations to facilitate further improvements and ease future enhancements. First, it is advisable to thoroughly read this report and reassess our design choices to ensure that previous limitations are not perpetuated. Additionally, gaining a strong understanding of our communication program is essential, and to that end, we recommend the following steps:

1. Begin by deepening your understanding of the existing code, focusing on its object-oriented aspects. This initial step is crucial as it not only helps you grasp the overall architecture and operational logic but also allows you to identify potential areas for refinement.

2. Work on refining the object-oriented elements of the code. Improving these aspects will not only clarify and streamline the codebase but also enhance its modularity. This is essential for simplifying both the integration of new features and the ongoing maintenance of the system. For instance, consider implementing new functions or classes specifically designed for retry mechanisms, which can improve the robustness and reliability of operations.

Secondly, for those looking to enhance the visualization capabilities of our system, consider the following approaches could be considered:

- Look into the possibilities regarding auto configured drawings
- Evaluate the potential integration of Open Bridge Design System's 2D or AR components to enhance user interaction and realism.
- Is Unity and Ignition the best software to complete the task?
- Consider developing a new Unity module that subscribes to the OPC-UA server to streamline data flow and visualization.

8 Appendices

8.1 Python communication program

8.1.1 Main application

```
from modules import ws_to_opc, start_opc_server, opc_to_unity, rest_to_opc
from misc.functions import makeListFromTextfile

import asyncio
import json
import os
import sys
from colorama import Fore

path_to_topics = 'Data/topics.txt'
path_to_endpoints_restapi = 'Data/endpoints_restapi.txt'
path_to_connection_data = 'Data/Connection data.json'
client_to_use = ('websocket')


async def main():
    if getattr(sys, 'frozen', False) and hasattr(sys, '_MEIPASS'):
        os.chdir(sys._MEIPASS)
    else:
        os.chdir(os.path.dirname(os.path.abspath(__file__)))

    thrusters_connectionData = []
    tasks = []
    with open(path_to_connection_data, 'r') as file:
        connectionData = json.load(file)
        opc_connection_data = connectionData['opc_ua']
        tasks.append(start_opc_server.main(connectionData))

    if client_to_use == 'websocket':
        print("Using Websocket client")
        # start udp messages to unity visualization
        tasks.append(opc_to_unity.main(connectionData, path_to_topics))

        # start ws communication once per thruster
        try:
            for thruster_name, thruster_data in connectionData['thrusters'].items():
                print("thruster_name: ", thruster_name)
                print("thruster_data: ", thruster_data)
                connection_detail = (thruster_data, opc_connection_data, thruster_name)
                thrusters_connectionData.append(connection_detail)

                print("Started ws communication for: ", thruster_name)
                tasks.append(ws_to_opc.main(connection_detail, path_to_topics))

            print("Length of thrusters_connectionData: ", len(thrusters_connectionData))
```

```
46
47
48          except Exception as e:
49              print(f"Error in main: {e}")
50              await asyncio.sleep(5)
51
52
53      elif client_to_use == 'restapi':
54          print("Using Rest API client")
55          try:
56              list_of_topic_lists = []
57              topics = makeListFromTextfile(path_to_topics)
58              for thruster_name, thruster_data in connectionData['thrusters'].items():
59                  print("thruster_name: ", thruster_name)
60                  print("thruster_data: ", thruster_data)
61                  thrusters_connectionData.append((thruster_data, opc_connection_data,
    thruster_name))
62                  modified_topics = []
63
64                  # changing topic to become thrusterName + topic name. Removes everything before
    the second '.'
65                  for topic in topics:
66                      parts = topic.strip().split('.')
67                      new_topic = thruster_name + '_' + '.'.join(parts[2:])
68                      print("new_topic: ", new_topic)
69                      modified_topics.append(new_topic)
70                  list_of_topic_lists.append(modified_topics)
71              tasks.append(opc_to_unity.main(connectionData, path_to_endpoints_restapi,
    list_of_topic_lists))
72
73          except Exception as e:
74              print(f"Error in main restapi: {e}")
75              await asyncio.sleep(5)
76          for connectionData in thrusters_connectionData:
77              print("Started rest communication")
78              tasks.append(rest_to_opc.main(connectionData, path_to_endpoints_restapi))
79
80      try:
81          await asyncio.gather(*tasks)
82      except Exception as e:
83          print(f"{Fore.YELLOW}app: Error in main: {e}")
84
85
86  if __name__ == "__main__":
87      asyncio.run(main())
```

8.1.2 ws_to_opc module

```
1  import asyncio
2  from asyncua import ua, Client
3  from colorama import Fore, init
4  import json
```

```
5
6  init(autoreset=True)
7  from clients.ws_client import WebsocketClient
8  from misc.functions import replace_topic_prefix, determine_opcua_data_type_stringonly,
       makeListFromTextfile
9
10 variables_on_server = []
11 lock_variables_on_server = asyncio.Lock()
12
13
14
15 async def upload_to_opc(opcClient, object_node, variables_on_server, updated_topic_name,
       data_type_string, value):
16     method_node_id = "ns=2;s=AddVariableMethod"
17     method_node = opcClient.get_node(method_node_id)
18
19     variable_node_id = f"ns=2;s={updated_topic_name}"
20     if updated_topic_name not in variables_on_server:
21
22         print(f"Adding variable {updated_topic_name} with type {data_type_string} and value {
       value} to server")
23         result = await object_node.call_method(method_node, updated_topic_name, data_type_string,
        value)
24         if result:
25             variables_on_server.append(updated_topic_name)
26         #print(f"Exisiting variables: {variables_on_server}")
27
28     else:
29         var = opcClient.get_node(variable_node_id)
30         try:
31             #print(f"Updating variable {updated_topic_name} with value {value}")
32             await var.write_value(ua.Variant(value, getattr(ua, data_type_string)))
33         except Exception as e:
34             print(f"Error updating variable {updated_topic_name}: {e}")
35
36
37
38
39 async def process_data_queue(opcClient, dataQueue, object_node_id, thruster_name):
40     object_node = opcClient.get_node(object_node_id)
41     variables_on_server = []
42     topic = ""
43     while True:
44         try:
45             while dataQueue.qsize() > 0 :
46                 topic, topicData = await dataQueue.get()
47                 dataQueue.task_done()
48
49                 result = determine_opcua_data_type_stringonly(topicData)
50                 #print(f"result  from topic {topic} is {result}")
51                 if None in result:
52                     print(f"{Fore.YELLOW}Unsupported data type for topic {topic}")
```

```
                    continue

                _, _, data_type_string, value = result

                #print(f"Topic: {topic} became {topicName}")
                updated_topic_name = await replace_topic_prefix(topic, thruster_name, ":")  # changing topic to avoid confusion with other thrusters

                await upload_to_opc(opcClient, object_node, variables_on_server, updated_topic_name, data_type_string, value)

            await asyncio.sleep(1)

        except Exception as e:
            print(f"process_data_queue error: {e} topic is {topic}")

async def variable_exists(client, node_id):
    try:
        var = client.get_node(node_id)
        await var.get_value()
        return True
    except Exception as e:
        return False


async def check_opc_connection(client):
    while True:
        try:
            await client.connect()
            print("Connected to OPC server.")
            return True
        except Exception as e:
            print(f"{Fore.YELLOW}Error connecting to OPC server: {e}")
            await asyncio.sleep(1)


async def main(connectionData, path_to_topics):
    try:
        thruster_connection_data = connectionData[0]
        opc_connection_data = connectionData[1]
        thruster_name = connectionData[2]
        topics = makeListFromTextfile(path_to_topics)

        ws_uri = thruster_connection_data['websocket']['ws_url']
        opc_url = opc_connection_data['opc_url']
        object_node_id = str(thruster_connection_data['object_node_id'])

        with open('Data/config.json', 'r') as file:
            config = json.load(file)
        #receiver_function_name = config['ws_to_opc']['receiver_function_name']

        print(f"opc_url: {opc_url}", f"ws_uri: {ws_uri}", f"topics length: {len(topics)}",
```

```
              f"object_node_id: {object_node_id}")

        PRODUCERS = min(len(topics), 100)

        parsed_response_queue = asyncio.Queue()
        topics_queue = asyncio.Queue()
        [await topics_queue.put(topic) for topic in topics]
        print(f"topics_queue size: {topics_queue.qsize()}")

        ws_clients = [WebsocketClient(ws_uri) for _ in range(PRODUCERS)]
        await asyncio.gather(*(client.connect() for client in ws_clients))

        opc_client = Client(url=opc_url)
        while True:
            try:
                await opc_client.connect()
                print("Connected to OPC server.")
                break
            except Exception as e:
                print(f"{Fore.YELLOW}Error connecting to OPC server: {e}")
                await asyncio.sleep(1)

        tasks = []
        tasks.extend([check_opc_connection(opc_client)])


        for i in range(PRODUCERS):
            client_index = i % len(ws_clients)  # Cycle through ws_clients
            #Change from receive_message2 to receive_message for the original version which only use the first 100 topics
            tasks.append(
                ws_clients[client_index].receive_message(topics_queue, parsed_response_queue))

        tasks.append(process_data_queue(opc_client, parsed_response_queue, object_node_id, thruster_name))

        await asyncio.gather(*tasks)

    except Exception as e:
        print(f"{Fore.YELLOW}ws_to_opc: Error in main: {e}")
        print("Shutting down...")
```

8.1.3 rest_to_opc module

```
import asyncio
from asyncua import ua, Client
from colorama import Fore, init
import json

init(autoreset=True)
from clients.ws_client import WebsocketClient
from misc.functions import replace_topic_prefix, determine_opcua_data_type_stringonly,
```

```
    makeListFromTextfile

variables_on_server = []
lock_variables_on_server = asyncio.Lock()



async def upload_to_opc(opcClient, object_node, variables_on_server, updated_topic_name,
    data_type_string, value):
    method_node_id = "ns=2;s=AddVariableMethod"
    method_node = opcClient.get_node(method_node_id)

    variable_node_id = f"ns=2;s={updated_topic_name}"
    if updated_topic_name not in variables_on_server:

        print(f"Adding variable {updated_topic_name} with type {data_type_string} and value {
    value} to server")
        result = await object_node.call_method(method_node, updated_topic_name, data_type_string,
     value)
        if result:
            variables_on_server.append(updated_topic_name)
        #print(f"Exisiting variables: {variables_on_server}")

    else:
        var = opcClient.get_node(variable_node_id)
        try:
            #print(f"Updating variable {updated_topic_name} with value {value}")
            await var.write_value(ua.Variant(value, getattr(ua, data_type_string)))
        except Exception as e:
            print(f"Error updating variable {updated_topic_name}: {e}")




async def process_data_queue(opcClient, dataQueue, object_node_id, thruster_name):
    object_node = opcClient.get_node(object_node_id)
    variables_on_server = []
    topic = ""
    while True:
        try:
            while dataQueue.qsize() > 0 :
                topic, topicData = await dataQueue.get()
                dataQueue.task_done()

                result = determine_opcua_data_type_stringonly(topicData)
                #print(f"result  from topic {topic} is {result}")
                if None in result:
                    print(f"{Fore.YELLOW}Unsupported data type for topic {topic}")
                    continue

                _, _, data_type_string, value = result

```

```
                #print(f"Topic: {topic} became {topicName}")
                updated_topic_name = await replace_topic_prefix(topic, thruster_name, ":")  # changing topic to avoid confusion with other thrusters

                await upload_to_opc(opcClient, object_node, variables_on_server, updated_topic_name, data_type_string, value)

            await asyncio.sleep(1)

        except Exception as e:
            print(f"process_data_queue error: {e} topic is {topic}")

async def variable_exists(client, node_id):
    try:
        var = client.get_node(node_id)
        await var.get_value()
        return True
    except Exception as e:
        return False


async def check_opc_connection(client):
    while True:
        try:
            await client.connect()
            print("Connected to OPC server.")
            return True
        except Exception as e:
            print(f"{Fore.YELLOW}Error connecting to OPC server: {e}")
            await asyncio.sleep(1)


async def main(connectionData, path_to_topics):
    try:
        thruster_connection_data = connectionData[0]
        opc_connection_data = connectionData[1]
        thruster_name = connectionData[2]
        topics = makeListFromTextfile(path_to_topics)

        ws_uri = thruster_connection_data['websocket']['ws_url']
        opc_url = opc_connection_data['opc_url']
        object_node_id = str(thruster_connection_data['object_node_id'])

        with open('Data/config.json', 'r') as file:
            config = json.load(file)
        #receiver_function_name = config['ws_to_opc']['receiver_function_name']

        print(f"opc_url: {opc_url}", f"ws_uri: {ws_uri}", f"topics length: {len(topics)}",
              f"object_node_id: {object_node_id}")

        PRODUCERS = min(len(topics), 100)
```

```
        parsed_response_queue = asyncio.Queue()
        topics_queue = asyncio.Queue()
        [await topics_queue.put(topic) for topic in topics]
        print(f"topics_queue size: {topics_queue.qsize()}")

        ws_clients = [WebsocketClient(ws_uri) for _ in range(PRODUCERS)]
        await asyncio.gather(*(client.connect() for client in ws_clients))

        opc_client = Client(url=opc_url)
        while True:
            try:
                await opc_client.connect()
                print("Connected to OPC server.")
                break
            except Exception as e:
                print(f"{Fore.YELLOW}Error connecting to OPC server: {e}")
                await asyncio.sleep(1)

        tasks = []
        tasks.extend([check_opc_connection(opc_client)])


        for i in range(PRODUCERS):
            client_index = i % len(ws_clients)  # Cycle through ws_clients
            #Change from receive_message2 to receive_message for the original version which only use the first 100 topics
            tasks.append(
                ws_clients[client_index].receive_message(topics_queue, parsed_response_queue))

        tasks.append(process_data_queue(opc_client, parsed_response_queue, object_node_id, thruster_name))

        await asyncio.gather(*tasks)

    except Exception as e:
        print(f"{Fore.YELLOW}ws_to_opc: Error in main: {e}")
        print("Shutting down...")
```

8.1.4 opc_to_unity module

```
import asyncio
from asyncua import Client
from clients.udp_client import UDPClient
from misc.functions import makeListFromTextfile, replace_topic_prefix
import json
from colorama import Fore, init

init(autoreset=True)
opc_url = "opc.tcp://localhost:4840/freeopcua/server/"
opc_namespace = "http://NGTEC.io"


```

```
async def main(connectionData, path_to_topics, list_of_topic_lists=[]):
    with open('Data/config.json', 'r') as file:
        config = json.load(file)
        startup_delay = config['opc_to_unity']['startup_delay']

    await asyncio.sleep(startup_delay) #to give time for the other tasks to add all topics to the server.
    num_thrusters = len(connectionData['thrusters'])
    print(f"ConnectionData: {connectionData}")
    print(f"{num_thrusters} thrusters found in connection data.")

    if list_of_topic_lists == []:
        topics = makeListFromTextfile(path_to_topics)
        for thruster_name in connectionData['thrusters'].keys():
            modified_topics = [await replace_topic_prefix(topic, f"{thruster_name}", ':') for topic in topics]
            list_of_topic_lists.append(modified_topics.copy())
            for topic in modified_topics:
                print(f"Modified topic: {topic}")

    udp_ip = connectionData['udp_data_target']['ip_address']
    udp_port = connectionData['udp_data_target']['port']
    udp_client = UDPClient(udp_ip, udp_port)

    max_retries = 1  # Maximum number of retries
    retry_delay = 1/len(topics)  # Delay between retries in seconds. #limiting it to 1 second per retry
    attempt = 0

    #Check which topics are available. Stop checking unavailable topics
    while attempt < max_retries:
        data_to_send = {}
        try:
            async with Client(url=opc_url) as opcClient:
                attempt = 0  # Resetting connection attempts on successful connection
                while True:
                    await asyncio.sleep(1)
                    nsidx = await opcClient.get_namespace_index(opc_namespace)
                    for i, thruster_name in enumerate(connectionData['thrusters']):
                        topics_to_remove = []
                        for topic in list_of_topic_lists[i]:
                            retries = 0
                            while retries < max_retries:
                                try:
                                    var = opcClient.get_node(f"ns=2;s={topic}")
                                    value = await var.read_value()
                                    data_to_send[topic] = str(value)
                                    #print("success reading value for ", topic)
                                    break
                                except Exception as e:
                                    print(f"{Fore.YELLOW}opc_to_unity: Error reading value for {topic}: {e}")
```

```
                                    retries += 1
                                    await asyncio.sleep(retry_delay)
                            if retries >= max_retries:
                                print(f"{Fore.RED}Failed to read value for {topic} after {
    max_retries} retries. Removing topic.")
                                topics_to_remove.append(topic)
                        for topic in topics_to_remove:
                            list_of_topic_lists[i].remove(topic)
                        #print("Data to send: ", data_to_send)
                        json_data = json.dumps(data_to_send)
                        udp_client.send_data(json_data)
            except Exception as e:
                print(f"Attempt {attempt + 1} failed with error: {e}")
                attempt += 1
                await asyncio.sleep(retry_delay)

        if attempt == max_retries:
            print("Failed to connect after maximum retries.")
```

8.1.5 start_opc_server

```
import asyncio
import logging
import json
from colorama import Fore


from asyncua import Server, ua
from asyncua.common.methods import uamethod
from misc.functions import determine_opcua_data_type


@uamethod
def func(parent, value):
    return value * 2


@uamethod
async def add_variable_method(parentNodeId, name, data_type, initial_value):
    #myobj = server.get_node(parent)
    try:
        parentNode = server.get_node(parentNodeId)
        print("object used in add_variable_method: ", parentNode)
    except Exception as e:
        print(f"{Fore.LIGHTYELLOW_EX}opc server: Error getting parent node: {e}")
        return False

    if data_type == "Int32":
        ua_data_type = ua.VariantType.Int32
    elif data_type == "Double":
        ua_data_type = ua.VariantType.Double
    elif data_type == "String":
```

```
        ua_data_type = ua.VariantType.String
    elif data_type == "Boolean":
        ua_data_type = ua.VariantType.Boolean
    else:
        raise ValueError(f"Unsupported data type: {data_type}")

    try:
        data_type_node_id = ua.NodeId(ua_data_type.value)
    except Exception as e:
        print(f"{Fore.LIGHTYELLOW_EX}opc server: Error getting data type node id: {e}")
        return False

    try:
        node_id = ua.NodeId(name, parentNode.nodeid.NamespaceIndex)
        variable_node = await parentNode.add_variable(
            node_id,
            name,
            str(initial_value),
            datatype=data_type_node_id)
        await variable_node.set_writable()
        return True
    except Exception as e:
        print(f"opc server: Error adding variable: {e}")
        return False


async def process_topic(client, topic, timeout=10):  # Timeout in seconds
    try:
        # Enforcing a timeout
        data = await asyncio.wait_for(client.process_message_and_return_value(topic), timeout)
        if data:
            message = data[-1]
            _, data_type, value = determine_opcua_data_type(message)
            return topic, data_type, message
    except asyncio.TimeoutError:
        print(f"Timeout reached for topic {topic}")
    except Exception as e:
        print(f"Error processing message for topic {topic}: {e}")
    return None


async def main(connectionData):
    _logger = logging.getLogger(__name__)
    print("main")

    opc_endpoint = connectionData['opc_ua']['opc_endpoint']
    uri = connectionData['opc_ua']['opc_namespace']
    # setup our server
    global server
    server = Server()
    await server.init()
    server.set_security_policy([ua.SecurityPolicyType.NoSecurity])
```

```
    server.set_endpoint(opc_endpoint)

    idx = await server.register_namespace(uri)

    with open('Data/connection data.json', 'r') as file:
        data = json.load(file)
        for thruster_name, thruster_data in data['thrusters'].items():
            # Extract thruster name from the tuple
            thruster_name_str = thruster_name
            print("Added ", thruster_name_str, " to the server. node idx is ", idx)
            await server.nodes.objects.add_object(idx, thruster_name_str)
            await asyncio.sleep(0.5)

    await server.nodes.objects.add_method(ua.NodeId("AddVariableMethod", idx), "AddVariable",
    add_variable_method,
                                          [ua.VariantType.String, ua.VariantType.String, ua.
    VariantType.String],
                                          [ua.VariantType.Boolean])

    await asyncio.sleep(1)

    await server.nodes.objects.add_method(
        ua.NodeId("ServerMethod", idx),
        ua.QualifiedName("ServerMethod", idx),
        func,
        [ua.VariantType.Int64],
        [ua.VariantType.Int64],
    )

    _logger.info("Starting server!")
    async with server:
        while True:
            await asyncio.sleep(1)


if __name__ == "__main__":
    print("start")
    logging.basicConfig(level=logging.DEBUG)
    asyncio.run(main(), debug=True)
```

8.1.6 Simulate_thrust_load module

```
import asyncio

import aiohttp


async def gather_data(endpoint, queue):
    while True:
        async with aiohttp.ClientSession() as session:
            async with session.get(endpoint) as response:
                if response.status == 200:
```

```
                    data = await response.json()
                    print(f"Received data from {endpoint}: {data}")
                    await queue.put(data)
                else:
                    print(f"Failed to receive data, status: {response.status}")


async def modify_data(raw_data_queue, modified_data_queue):  # simulate load in this func
    modify = True
    while True:
        modified_data = {}
        try:
            data = await raw_data_queue.get()
            if modify:
                if "Pitch_Order_Setpoint" in data:
                    value = data["Pitch_Order_Setpoint"]
                    dir = 1 if value < -90 else -1 if value > 90 else 1
                    value += float(dir * 1)
                    modified_data = {
                        "Pitch_Order_Setpoint": {
                            'isOK': True,
                            'valueType': 'FLOAT',
                            'floatValue': value
                        }
                    }
            else:
                modified_data = data  # Passing data unchanged
            await modified_data_queue.put(modified_data)
        except Exception as e:
            print(f"Error modifying data: {e}")


async def upload_data(queue, target_url):
    while True:
        async with aiohttp.ClientSession() as session:
            while not queue.empty():
                data = await queue.get()
                queue.task_done()
                print(f"Uploading modified data {data} to {target_url}")
                async with session.patch(target_url, json=data) as response:
                    if response.status == 202:
                        print(f"status: {response.status}Protobuf overwrite initialized, data
    accepted for processing.")
                    else:
                        print(f"Failed to overwrite data, status: {response.status}")
                await asyncio.sleep(1)


async def main():
    raw_data_queue = asyncio.Queue()
    modified_data_queue = asyncio.Queue()
    endpoint = "http://192.168.252.3/api/components/PitchControl/protos/PitchControlInputs/fields
```

```
    "
    tasks = [gather_data(endpoint, raw_data_queue), upload_data(modified_data_queue, endpoint),
             modify_data(raw_data_queue, modified_data_queue)]
    await asyncio.gather(*tasks)


if __name__ == "__main__":
    asyncio.run(main())
```

8.1.7 UDP client

```
import socket

def make_message(topic, value):
    return f"{topic}:{value}"

class UDPClient:
    def __init__(self, server_address, server_port):
        """Initialize the UDP client with server address and port."""
        self.server_address = server_address
        self.server_port = server_port
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        # Set the timeout for the socket
        self.timeouttime = 1
        self.socket.settimeout(self.timeouttime)

    def send_data(self, topic: str, data=None):
        try:
            if data is None:
                message = topic
            else:
                message = make_message(topic, data)
            sent = self.socket.sendto(message.encode(), (self.server_address, self.server_port))
            print(f"Sent {sent} bytes to {self.server_address}:{self.server_port}.")
        except Exception as e:
            print(f"Error sending data: {e}")

    def receive_data(self, buffer_size=1024):
        self.socket.bind((self.server_address, self.server_port))
        while True:
            try:
                # Attempt to receive response within the timeout period

                data, server = self.socket.recvfrom(buffer_size)
                print(f"Received: {data} from {server}")

            except socket.timeout:
                print(f"No response received within {self.timeouttime} sec.")
            except Exception as e:
                print(f"Error receiving data: {e}")

    def close(self):
```

```
        """Close the socket."""
        self.socket.close()
```

8.1.8 Websocket client

```
import asyncio
import aiohttp
import json
from colorama import Fore, Style, init

init(autoreset=True)
import time


class WebsocketClient:
    def __init__(self, uri, keepalive_interval=30):
        self.uri = uri
        self.keepalive_interval = keepalive_interval
        self.session = None
        self.websocket = None
        self.topics = []
        self.topics_data_received = {}
        self.connection_established = False

    async def connect(self):
        self.session = aiohttp.ClientSession()
        max_retry = 5
        retry_count = 0
        while not self.connection_established and retry_count < max_retry:
            try:
                self.websocket = await self.session.ws_connect(self.uri)
                self.connection_established = True
                print(f"Connected to WebSocket at {self.uri}")
            except Exception as e:
                print(f"Error connecting to WebSocket: {e}")
                self.connection_established = False
                retry_count += 1
                await asyncio.sleep(1)  # delay before retrying
                if retry_count >= max_retry:
                    print(f"Maximum retries reached. Connection could not be established.")
                    break
        if not self.connection_established:
            print("Failed to establish connection after maximum retries.")

    async def parse_ws_message(self, response):

        try:
            data = None
            data = json.loads(response.data)
            if data:
                trends = data.get('trends', [])
                if trends:
```

```
                    time_value_tuples = trends[0].get('timeValueTuples', [])
                    _, value = time_value_tuples[-1]  # -1 is the latest pair (timestamp, value)
                    return value
            else:
                print(f"{Fore.YELLOW}parse_ws_message: No data in response")

        except Exception as e:
            print(f"{Fore.YELLOW}parse_ws_message error processing data: {data}\n{e}")

    async def receive_message2(self, topics_queue, queue):#Attempt to request new topic every iteration
        while True:
            topic = await self.getTopic(topics_queue)
            await self.sendRequest(topic)
            noneCounter = 0
            max_none_response_retries = 3
            try:
                while True:
                    response = await asyncio.wait_for(self.websocket.receive(), timeout=5)
                    try:
                        response_value = await self.parse_ws_message(response)
                        if response_value is None:
                            noneCounter += 1
                            if noneCounter >= max_none_response_retries:
                                print(
                                    f"{Fore.YELLOW}Received None response {max_none_response_retries} times consecutively for topic '{topic}'. Trying next topic.")
                                topic = await self.try_next_topic(
                                    topics_queue)  # retry without adding topic back to queue. Removing it from the process.
                                break
                        else:
                            response_value = await self.parse_ws_message(response)
                            await queue.put((topic, response_value))
                            # (f" put topic '{topic}' to queue")
                            topic = await self.getTopic(topics_queue)
                            await self.sendRequest(topic)
                            break

                    except asyncio.TimeoutError:
                        print(f"{Fore.YELLOW}Timeout waiting for message for topic '{topic}'. Retrying...")
                        continue
            except asyncio.TimeoutError:
                print(f"{Fore.YELLOW}Timeout waiting for message for topic '{topic}'. Retrying ...")
                continue



    async def getTopic(self, topics_queue):
        try:
```

```
            topic = await topics_queue.get()
            topics_queue.task_done()
            return topic
        except Exception as e:
            print(f"Error getting topic from topics_queue {e}")
            print(f" Topics_queue size: {topics_queue.qsize()}")
            return None

    async def sendRequest(self, topic):
        while True:
            try:
                await self.websocket.send_str(topic)
                print(f"Sent topic '{topic}' to WebSocket.")
                break
            except Exception as e:
                print(f"{Fore.YELLOW}ws_client: Error sending message for topic '{topic}': {e}")
                continue

    async def try_next_topic(self, topics_queue):
        topic = await self.getTopic(topics_queue)
        await self.sendRequest(topic)
        return topic
    async def receive_message(self, topics_queue, queue):

        topic = await self.try_next_topic(topics_queue)
        while True:
            noneCounter = 0
            timeout_retries = 0
            max_timeout_retries = 999
            max_none_response_retries = 5
            try:
                response = await asyncio.wait_for(self.websocket.receive(), timeout=99)
                if response.type in (aiohttp.WSMsgType.CLOSED, aiohttp.WSMsgType.ERROR):
                    print(f"Connection closed or errored for topic '{topic}'. Reconnecting...")
                    self.connection_established = False
                    await self.connect()
                    continue  # Attempt to reconnect within the outer loop

                response_value = await self.parse_ws_message(response)
                if response_value is None:
                    noneCounter += 1
                    if noneCounter >= max_none_response_retries:
                        print(
                            f"{Fore.YELLOW}Received None response {max_none_response_retries}
    times consecutively for topic '{topic}'. Trying next topic.")#retry without adding topic back
     to queue. Removing it from the process.

                        topic = await self.try_next_topic(topics_queue)
                        continue
                else:
                    await queue.put((topic, response_value))

```

```
            except asyncio.TimeoutError:
                print(f"{Fore.YELLOW}Timeout waiting for message for topic '{topic}'. Retrying
    ...")
                timeout_retries += 1
                if timeout_retries >= max_timeout_retries:
                    break
                else:
                    topic = await self.try_next_topic(topics_queue)
                    await asyncio.sleep(1)
                    continue  # Continue to retry receiving on the same topic
            except Exception as e:
                print(f"{Fore.YELLOW}Error processing message for topic '{topic}': {e}")
                continue

    async def close(self):
        if self.websocket:
            await self.websocket.close()
        if self.session:
            await self.session.close()
        self.connection_established = False

    async def close(self):
        if self.websocket:
            await self.websocket.close()
        if self.session:
            await self.session.close()
        self.connection_established = False
```

8.1.9 functions

```
import json
from asyncua import ua


def makeListFromTextfile(filepath: str):
    with open(filepath) as f:
        content = f.readlines()
    content = [x.strip() for x in content]
    return content


def makeListFromTextfile_limit(filepath: str, limit: int):
    topics = []
    with open(filepath, 'r') as file:
        for i, line in enumerate(file):
            topics.append(line.strip())
            if i + 1 >= limit:
                break
    return topics


def makeListFromTextfile_restapi(baseurl: str, filepath: str):
```

```
    topics = []
    try:
        with open(filepath, 'r') as file:
            for line in file:
                cleaned_line = line.strip()
                if cleaned_line:
                    full_url = baseurl + cleaned_line
                    topics.append(full_url)
                    print(f"Added URL: {full_url}")
    except FileNotFoundError:
        print(f"Error: The file '{filepath}' does not exist.")
    except Exception as e:
        print(f"An error occurred: {e}")
    return topics


async def clear_queue(dataQueue):
    while not dataQueue.empty():
        await dataQueue.get()
        dataQueue.task_done()


def determine_opcua_data_type(value):
    try:
        if value:

            if isinstance(value, str):
                return ua.VariantType.String, ua.ObjectIds.String, "String", value
            elif isinstance(value, float):
                return ua.VariantType.Double, ua.ObjectIds.Double, "Double", value
            elif isinstance(value, (dict, list)):
                value = json.dumps(value)  # Convert complex types to JSON string
                return ua.VariantType.String, ua.ObjectIds.String, "String", value
            elif isinstance(value, int):
                return ua.VariantType.Double, ua.ObjectIds.Double, "Double", float(value)

            else:
                raise ValueError(f"Unsupported data type for value: {type(value)}")
        else:
            return None, None, None, None
    except Exception as e:
        print(f"Error determining data type: {e}")
        return None, None, None, None

def determine_opcua_data_type_stringonly(value):
    from opcua import ua
    import json

    try:
        # Convert value to string if not already
        if isinstance(value, str):
            str_value = value
```

```
        elif isinstance(value, (dict, list)):
            str_value = json.dumps(value)
        else:
            str_value = str(value)


        return ua.VariantType.String, ua.ObjectIds.String, "String", str_value

    except Exception as e:
        print(f"Error determining data type: {e}")
        return None, None, None, None

async def replace_topic_prefix(topic, new_prefix, divider):
    try:
        # Split the topic at the first divider and replace everything before it with the new_prefix
        return new_prefix + topic.split(divider, 1)[1] if divider in topic else new_prefix + topic
    except Exception as e:
        print(f"replace_topic_prefix error: {e}")
        return topic
```

8.2 Python Test Codes

8.2.1 REST API Single session

```
import asyncio
import aiohttp
import time

async def fetch_data(url, session):
    try:
        start_time = time.time()  # Record start time
        async with session.get(url) as response:
            if response.status == 200:
                await asyncio.sleep(0.1)
                #print(response.status)
                duration = time.time() - start_time  # Calculate time taken
                print(f"GET request to {url} took {duration} seconds")
                return url, await response.json(), duration  # Return URL, data, and time taken
            elif response.status == 404:
                print(f"Specified component not found at url: {url}")
            else:
                print(f"Unexpected status {response.status} received from {url}")
    except aiohttp.ClientError as e:
        print(f"Client error occurred when fetching data from {url}: {e}")
    return None, None, 0  # Return None, None, and 0 time if request fails

async def main():
    urls = [
        "http://192.168.252.3/api/components/SignalPool/protos/PitchControlInputs/fields",
        "http://192.168.252.3/api/components/PitchControl/protos/PitchControlInputs/fields",
```

```
        "http://192.168.252.3/api/components/PitchControl/protos/PitchControlTestpoints/fields",
        "http://192.168.252.3/api/components/PitchControl/protos/PitchControlOutputs/fields",
        "http://192.168.252.3/api/components/PitchControl/protos/PitchControlParams/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadControlInputs/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadControlOutputs/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadControlTestpoints/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadControlParams/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadController1Testpoints/fields"
    ,
        "http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandInputs/fields",
        "http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandOutputs/fields",
        "http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandTestpoints/fields",
        "http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandParams/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandInputs/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandOutputs/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandTestpoints/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Testpoints/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Outputs/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Testpoints/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandParams/fields",
        "http://192.168.252.3/api/components/AlarmInterface/protos/LoadControlAlarms/fields"
    ]

    delay = 1  # seconds between each iteration of the loop

    async with aiohttp.ClientSession(connector=aiohttp.TCPConnector(ssl=False), trust_env=True)
    as session:
        while True:  # This will loop forever, use a condition to break loop if necessary
            time_start = time.time()
            tasks = [fetch_data(url, session) for url in urls]
            results = await asyncio.gather(*tasks)
            duration = time.time() - time_start
            print(f"Gathering data took: {duration} seconds")

            await asyncio.sleep(delay)  # Wait for `delay` seconds before next iteration

if __name__ == "__main__":
    asyncio.run(main())
```

8.2.2 REST API Test Multiple Sessions

```
import aiohttp
import asyncio
import time


async def fetch_data(url, session):
    try:
        async with session.get(url) as response:
            if response.status == 200:
                data = await response.json()
                return url, data
```

```
            else:
                return url, None
    except aiohttp.ClientError as e:
        return url, None


async def handle_urls(urls, session):
    start_time = time.perf_counter()
    tasks = [fetch_data(url, session) for url in urls]
    results = await asyncio.gather(*tasks)
    end_time = time.perf_counter()
    duration = end_time - start_time
    return results, duration


async def main():
    urls = [
        "http://192.168.252.3/api/components/SignalPool/protos/PitchControlInputs/fields",
        "http://192.168.252.3/api/components/PitchControl/protos/PitchControlInputs/fields",
        "http://192.168.252.3/api/components/PitchControl/protos/PitchControlTestpoints/fields",
        "http://192.168.252.3/api/components/PitchControl/protos/PitchControlOutputs/fields",
        "http://192.168.252.3/api/components/PitchControl/protos/PitchControlParams/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadControlInputs/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadControlOutputs/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadControlTestpoints/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadControlParams/fields",
        "http://192.168.252.3/api/components/LoadControl/protos/LoadController1Testpoints/fields"
    ,
        "http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandInputs/fields",
        "http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandOutputs/fields",
        "http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandTestpoints/fields",
        "http://192.168.252.3/api/components/PitchCommand/protos/PitchCommandParams/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandInputs/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandOutputs/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandTestpoints/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Testpoints/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Outputs/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandEng1Testpoints/fields",
        "http://192.168.252.3/api/components/RpmCommand/protos/RpmCommandParams/fields",
        "http://192.168.252.3/api/components/AlarmInterface/protos/LoadControlAlarms/fields"
    ]
    # Split URLs into 3 chunks
    chunks = [urls[i::3] for i in range(3)]

    total_start_time = time.perf_counter()  # Start total timer

    # Create and manage sessions
    async with aiohttp.ClientSession() as session1, aiohttp.ClientSession() as session2, aiohttp.
    ClientSession() as session3:
        sessions = [session1, session2, session3]
        tasks = [handle_urls(chunks[i], sessions[i]) for i in range(3)]
        all_results = await asyncio.gather(*tasks)
```

```
        combined_results = [item for sublist in all_results for item in sublist[0]]  # Access results
        for i, session_results in enumerate(all_results):
            print(f"Session {i + 1} took {session_results[1]:.2f} seconds.")

    total_end_time = time.perf_counter()
    total_duration = total_end_time - total_start_time
    print(f"Total time to gather all data: {total_duration:.2f} seconds")
    print(combined_results)

asyncio.run(main())
```

8.2.3 Websocket Test Multiple Clients

```
import asyncio
import aiohttp
import json
from colorama import Fore, Style, init
from misc.functions import makeListFromTextfile_limit
import time

class WebsocketClient:
    def __init__(self, uri, keepalive_interval=30):
        self.uri = uri
        self.keepalive_interval = keepalive_interval
        self.session = aiohttp.ClientSession()
        self.topics = []

    async def keepalive_ping(self, websocket):
        while True:
            await asyncio.sleep(self.keepalive_interval)
            await websocket.ping()

    async def receive_message(self, topics, queue, number_active_connections):
        try:
            async with self.session.ws_connect(self.uri) as websocket:
                number_active_connections.put_nowait(1)
                print("Connected to websocket server")
                print(f"{Fore.GREEN} CONNECT: number active connections: ", number_active_connections.qsize())

                if not topics.empty():
                    topic = await topics.get()
                    topics.task_done()
                    await websocket.send_str(topic)
                    while True:
                        response = await websocket.receive()
                        print(f"{Fore.RESET}Received message from topic ", topic)
                        try:
                            if response.data:
                                value = await self.process_message(response)
                                await queue.put((topic, value))
```

```
                        except Exception as e:
                            print(f"Error processing message: {e}")
        except Exception as e:
            con = number_active_connections.get()
            number_active_connections.task_done()
            print(f"{Fore.RED} DISCONNECT: number active connections: ",
    number_active_connections.qsize())
            await asyncio.sleep(5)


    async def process_message(self, data):
        try:
            data = json.loads(data.data)
            trends = data.get('trends', [])
            if trends:
                time_value_tuples = trends[0].get('timeValueTuples', [])
                return(time_value_tuples[-1])
        except Exception as e:
            print("Error processing data:", data)
            print(e)


async def do_stuff(queue):
    try:
        while True:
            while not queue.empty():
                stuff = await queue.get()
                queue.task_done()

            print(f"{Fore.LIGHTMAGENTA_EX} ####################NEXT BATCH OF DATA
    ##########################")
            await asyncio.sleep(1)
    except Exception as e:
        print("Error doing stuff:", e)


async def main():
    uri = 'ws://192.168.252.3/ws/Trend'

    topics_Q = asyncio.Queue()
    active_ws_connections = asyncio.Queue()
    ws_clients = []
    NUM_CLIENTS = 200

    for i in range(NUM_CLIENTS):
        ws_clients.append(WebsocketClient(uri))

    max_topics = NUM_CLIENTS
    topics = makeListFromTextfile_limit('../Data/topics.txt', max_topics)
    for topic in topics:
        topics_Q.put_nowait(topic)

```

```
    client = WebsocketClient(uri)

    dataQueue= asyncio.Queue()
    tasks = []
    tasks.extend([client.receive_message(topics_Q, dataQueue, active_ws_connections) for _ in
    ws_clients])
    tasks.extend([do_stuff(dataQueue) for _ in range(1)])

    await asyncio.gather(*tasks)


asyncio.run(main())
```

8.3 Unity codes in C#

8.3.1 UDP Client script

```
using System;
using System.Collections.Generic;
using UnityEngine;
using System.Net;
using System.Net.Sockets;
using System.Text;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;

public class UDPClient : MonoBehaviour
{
    private UdpClient serverSocket;
    private int port = 5002;
    private Queue<string> messageQueue = new Queue<string>();

    void Start()
    {
        serverSocket = new UdpClient(port);
        Debug.Log("UDP server listening on port " + port);
        serverSocket.BeginReceive(new AsyncCallback(ReceiveCallback), null);
    }

    void Update()
    {
        while (messageQueue.Count > 0)
        {
            string message;
            lock (messageQueue)
            {
                message = messageQueue.Dequeue();
            }
            ProcessMessage(message);
            Debug.Log($"ReceivedMessage: {message}");
        }

```

```
    }

    private void ReceiveCallback(IAsyncResult ar)
    {
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        byte[] receivedBytes = serverSocket.EndReceive(ar, ref sender);
        string receivedMessage = Encoding.UTF8.GetString(receivedBytes);
        Debug.Log($"ReceivedMessage: {receivedMessage}");
        lock (messageQueue)
        {
            messageQueue.Enqueue(receivedMessage);
        }
        serverSocket.BeginReceive(new AsyncCallback(ReceiveCallback), null);
    }

    private void ProcessMessage(string message)
    {
        try
        {
            var jsonData = JObject.Parse(message);
            foreach (var property in jsonData)
            {
                string name = property.Key;
                var value = property.Value;
                char thrusterID = name[1];
                name = name.Substring(2); // Removes the first two characters from the string 'name' to fit the JSON-message
                Debug.Log(name + ": " + value + " thrusterID: " + thrusterID);
                GameObject thrusterGameObject = GameObject.Find("Thruster" + thrusterID);
                if (thrusterGameObject != null)
                {
                    Movement movement = thrusterGameObject.GetComponent<Movement>();
                    if (movement != null)
                    {
                        if (name.Contains("Pitch_Feedback_Adapted"))
                        {
                            movement.UpdatePitch((float)value);

                        }
                        else if (name.Contains("Propeller_Rpm_Feedback_In_Rpm"))
                        {
                            movement.UpdateRPM((float)value * 0.25f);
                        }
                        else if (name.Contains("Direction_Feedback_Adapted"))
                        {
                            movement.UpdateAziDirection((float)value);
                        }
                    }
                }
            }
        }
        catch (JsonReaderException e)
```

```
        {
            Debug.LogError("JsonReaderException: " + e.Message);
        }
        catch (Exception e)
        {
            Debug.LogError("Exception: " + e.Message);
        }
    }

}
```

8.3.2 Movement script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Movement : MonoBehaviour
{
    public GameObject Propeller;
    public GameObject Blade1;
    public GameObject Blade2;
    public GameObject Blade3;
    public GameObject Blade4;
    public GameObject Azimuth;
    public float targetRpm = 0.0f;     // Target RPM that we want to interpolate towards
    public float targetPitch = 0.0f;   // Target pitch that we want to interpolate towards
    public float targetAzi = 0.0f;     // Target azimuth angle that we want to interpolate
    towards

    private float currentRpm = 0.0f;   // Current RPM used for interpolation
    private float currentPitch = 0.0f; // Current pitch used for interpolation
    private float currentAzi = 0.0f;   // Current azimuth used for interpolation

    private Quaternion[] initialBladeRotations;
    private Quaternion initialAzimuthRotation;

    public float interpolationSpeed = 1.0f;

    void Start()
    {
        // Store the initial rotations of the blades
        initialBladeRotations = new Quaternion[4];
        initialBladeRotations[0] = Blade1.transform.localRotation;
        initialBladeRotations[1] = Blade2.transform.localRotation;
        initialBladeRotations[2] = Blade3.transform.localRotation;
        initialBladeRotations[3] = Blade4.transform.localRotation;

        // Store the initial rotation of the Azimuth
        initialAzimuthRotation = Azimuth.transform.localRotation;
    }

```

```
    void Update()
    {
        // Interpolate the RPM, Pitch, and Azi values "smoothly" towards their targets
        currentRpm = Mathf.Lerp(currentRpm, targetRpm, Time.deltaTime * interpolationSpeed);
        currentPitch = Mathf.Lerp(currentPitch, targetPitch, Time.deltaTime * interpolationSpeed);
        currentAzi = Mathf.Lerp(currentAzi, targetAzi, Time.deltaTime * interpolationSpeed);

        // Calculate rotation amount based on interpolated rpm
        float rotationAmount = currentRpm * Time.deltaTime * 6f; // Convert rpm to degrees per second
        Propeller.transform.Rotate(Vector3.forward, rotationAmount);

        // Rotate each blade based on interpolated pitch
        RotateBlade(Blade1, currentPitch, Vector3.up, 0);
        RotateBlade(Blade2, currentPitch, Vector3.up, 1);
        RotateBlade(Blade3, currentPitch, Vector3.up, 2);
        RotateBlade(Blade4, currentPitch, Vector3.up, 3);

        // Rotate azimuth based on interpolated azi angle
        RotateAzi(currentAzi);
    }

    public void UpdatePitch(float newPitch)
    {
        targetPitch = newPitch; // Set target pitch
    }

    public void UpdateRPM(float newRPM)
    {
        targetRpm = newRPM; // Set target RPM
    }

    public void UpdateAziDirection(float newAzi)
    {
        targetAzi = newAzi; // Set target azimuth direction
    }

    void RotateBlade(GameObject blade, float pitchValue, Vector3 axis, int index)
    {
        if (blade != null)
        {
            // Calculate the angle based on the pitch value (mapped to a suitable range if needed)
            pitchValue = pitchValue * 0.35f; // To make the blades stop at 35 degrees
            float angle = Mathf.Clamp(pitchValue, -100f, 100f);


            // Rotate the blade around the specified axis relative to its initial rotation
            Quaternion targetRotation = initialBladeRotations[index] * Quaternion.AngleAxis(angle, axis);
            blade.transform.localRotation = targetRotation;
```

```
        }
        else
        {
            Debug.LogWarning("Blade object is not assigned.");
        }
    }

    void RotateAzi(float aziValue)
    {
        if (Azimuth != null)
        {
            Quaternion targetRotation = Quaternion.AngleAxis(aziValue, Vector3.up);
            Azimuth.transform.localRotation = Quaternion.Lerp(Azimuth.transform.localRotation,
    initialAzimuthRotation * targetRotation, Time.deltaTime * interpolationSpeed);
        }
        else
        {
            Debug.LogWarning("Azimuth object is not assigned.");
        }
    }
}
```

8.3.3 Propeller Particle System Controller script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PropellerParticleSystemController : MonoBehaviour
{
    public Movement movementScript;
    private ParticleSystem propellerParticles;
    private ParticleSystem.VelocityOverLifetimeModule velocityOverLifetimeModule;
    private ParticleSystem.ShapeModule shapeModule;

    void Start()
    {
        propellerParticles = GetComponent<ParticleSystem>();
        if (propellerParticles == null)
        {
            Debug.LogError("ParticleSystem component not found!");
            this.enabled = false;
            return;
        }

        if (movementScript == null)
        {
            Debug.LogError("Movement script is not assigned!");
            this.enabled = false;
            return;
        }

```

```
        velocityOverLifetimeModule = propellerParticles.velocityOverLifetime;
        shapeModule = propellerParticles.shape;
        velocityOverLifetimeModule.enabled = true;
        propellerParticles.Play();
    }

    void Update()
    {
        if (velocityOverLifetimeModule.enabled)
        {
            float speedModifierValue = movementScript.targetPitch / 100.0f;



            // Conditionally adjust the X-rotation of the shape module
            if (speedModifierValue < 0)
            {
                shapeModule.rotation = new Vector3(180f, 0f, 0f);  // Rotate 180 degrees around the X-axis
            }
            else
            {
                shapeModule.rotation = Vector3.zero;  // No rotation
            }
            // Update the speed modifier
            velocityOverLifetimeModule.speedModifier = new ParticleSystem.MinMaxCurve(Mathf.Abs(speedModifierValue)); // Set length of speed to propeller
        }
        else
        {
            Debug.LogWarning("VelocityOverLifetimeModule is not enabled.");
        }
    }
}
```

8.4 Pre-project planning report

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
Next Generation Training Environment

KANDIDATNUMMER(E):			
Torstein Ditlev Skare **Robert Alan Moltu**			
DATO: **25.01.2024**	EMNEKODE: **AIS2900**	EMNE: **Bacheloroppgave ingeniørfag**	DOKUMENT TILGANG: - Åpen
STUDIUM: **AUTOMATISERING OG INTELLIGENTE SYSTEMER**		ANT SIDER/VEDLEGG: 12/4	BIBL. NR: - Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
Robin T. Bye (NTNU) Ottar Osen (NTNU) Håkon Lunheim (Kongsberg Maritime)

OPPGAVE/SAMMENDRAG:
Formålet med dette prosjektet er å utvikle den best mulige løsningen for visualisering av verdier og justering av parametere innenfor Kongsberg Maritime (KM) sine systemer. Ved å bygge videre på ideen og prototypen som ble laget i sommeren 2023 og videre jobbet med i ett industriprosjekt høsten 2023. Bachelor oppgaven har som mål å videreutvikle prototypen med nye og innovative løsninger for å skape ett produkt som kan bli brukt både i felt og i opplæringssammenheng. I tillegg vil oppgaven gå ut på å ytterligere forbedre systemet og revurdere metodene og programmene som skal brukes.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

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

INNHOLD

1 INNLEDNING

Dette prosjektet har som mål å forbedre prototypen som ble utviklet under sommer/industriprosjektet 2023. Produktet har et potensiale som kan forbedre både opplæring og forenkle servicejobber ved å vise verdier og parameter til sluttbruker på en mer intuitiv måte. Historisk sett har opplæringskursene fulgt en konvensjonell tilnærming, der forelesere bruker PowerPoint-presentasjoner og deler ut omfattende papirdokumenter, ofte på over hundre sider, for å forklare systemene til Kongsberg Maritime. Denne bacheloren vil fokusere på å lage en innovativ og nyskapende løsning som gjør at brukeren av produktet vil lære systemene til KM på en mer effektiv måte. Ved å utvikle et produkt som gjør det lettere å følge signalflyten i systemet samtidig gjøre det enklere å finne relevant informasjon om systemet.

2 BEGREPER

- KM – Kongsberg Maritime
- Pelicase – Treningskoffert utstyrt med Mcon styresystem
- SCRUM – Metode for prosjektstyring
- Sprint – Periode i en SCRUM plan
- Gantt - Metode for prosjektstyring

3 PROSJEKTORGANISASJON

3.1 *Prosjektgruppe*

Studentnummer(e)
554668 – Torstein Ditlev Skare 522003 – Robert Alan Moltu

3.1.1 Oppgaver for prosjektgruppen – organisering

Gruppen har bare 2 medlemmer og kjører en flat struktur ettersom alle medlemmene står på omtrent samme nivå i beslutningshierarkiet. Ved uenighet vil gruppen samarbeide med veiledere eller oppdragsgiver for å komme fram til en enighet der det er forventet at gruppemedlemmene skal være åpne for nye forslag.

3.1.2 Oppgaver for prosjektgruppen

Skrive møtereferat
Oppdatere fremdriftsplan
Utføre prosjektarbeid
Utføre rapport

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Torstein Skare (Student)

Robert Moltu (Student)

Håkon Lunheim (Oppdragsgiver)

Ottar Osen (Veileder)

Robin T. Bye (Veileder)

4 AVTALER

4.1 Avtale med oppdragsgiver

Vi har avtalt å benytte Kongsberg Maritime lokaler og ressurser til prosjektarbeidet. Dette inkluderer testanlegg (Pelicase og trainingrom) samt PCer.

4.2 Arbeidssted og ressurser

- Bruker Kongsberg lokaler på NMK
- Menneskelige og fysiske ressurser er tilgjengelige på NMK i arbeidstid
- Konfidensielle dokumenter og bilder vil bli brukt
- Det er avtalt å ha møte med oppdragsgiver/veileder hver 14. dag. Alle i gruppa skal delta.
- Oppdatering av fremdriftsplan skal skje minst 2 ganger ukentlig (hver mandag og torsdag).

4.3 Gruppenormer – samarbeidsregler – holdninger

For den som gjennomfører individuelt prosjekt, vil avsnittet begrenses til den siste delen.

- Følge normale arbeidstider (8-16, mandag-fredag) så lenge prosjektet går etter planen.
- Ekstra arbeid utover normale arbeidstider kan/vil oppstå dersom nødvendig.
- Dokumentere kontinuerlig gjennom utførelsen av prosjektet.
- Jobbe målrettet og holde gode arbeidsrutiner.
- Utøve godt samarbeid med arbeidsgiver/veileder og gruppemedlemmer.
- Ha en profesjonell holdning til alle involverte i prosjektet.
- Levere en prosjektoppgave som blir til nytte for oppdragsgiver.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Problemstilling

MCON systemet til Kongsberg Maritime bruker CAF til å lagre og vise verdier og parameter fra gjeldende system. CAF er tungvint å bruke fordi det viser alle verdier og parameter i listeformat som gjør det vanskelig å få oversikt.

Software blokk diagrammene er vanskelige å forstå siden man må finne riktig PDF og side for å få lest om logikken, og fordi man må lete gjennom CAF dersom man skal se de faktiske verdiene på ett hvert punkt i diagrammet. All denne letingen resulterer med at det tar lang tid å finne de verdiene man er interessert i, noe som fører til at både service og opplæring tar lenger tid. Samtidig fører det til unødvendig forvirring og dårlig helhetsbilde av systemet som kan føre til at folk mister fokus på kurs eller på service.

Hensikt

Hensikten med prosjektet er å lage ett produkt som gjør service og opplæring enklere. Dette skal vi gjøre ved å harmonisere "realtime" verdier med Software blokk diagrammene. For å ytterligere utvikle løsningen skal prosjektet også inneholde en digital tvilling og simulator av en thruster/båt. Dette vil gi en reell visuell og interaktiv opplæring samt gi servicepersonell nyttig innsikt i systemet som de kan bruke til feilsøking, idriftsetting og annet.

Målsetting

Målsetningen for prosjektoppgaven er å følge 3 hovedpunkt som er:

1. Robusthet for datasamlingsprogram
    - Gjøre programmet robust nokk til at det ikke krasjer CAF
    - Implementere tiltak for cybersikkerhet for å sikre dataintegritet og systemets pålitelighet.
2. Utvidelse og optimalisering av funksjonalitet og tilgjengelighet
    - Implementere toveis kommunikasjon, med et første fokus på CAF.
    - Automatisk konfigurering av IP-adresser fra config-fil i marinecontroller.
    - Mer effektiv datahenting ved å utvikle en ny server
        - Dele opp programmet i henting, beregning og lagring, slik at beregninger skjer i server og ikke i GUI (Ignition).
    - Gjøre produktet mer brukervennlig.
        - Lage .exe fil
        - Oppstarts script
3. Nye teknologiske løsninger
    - Bruke en PLS til å lage en thruster simulator, slik at vi kan fjerne hardware fra nåværende testrigg (Pelicase) (Låne foreksempel Wago Compact100 fra NTNU).
    - Digital tvilling av thruster/båt i 3D og/eller i VR/AR

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Målet er å lage ett produkt som kan brukes til opplæring og service. For å oppfylle dette målet må følgende krav være oppfylt:

1. Datasamlingsprogrammet starter og kjører i 7 timer uten at CAF eller programmet krasjer. Testing må inneholde fysisk fjerning og tilbakekobling av signalkabel.
2. Data som blir lastet opp på server har mindre enn 2 sekund forsinkelse.
3. Visualiseringen bruker data fra serveren i "realtime".
4. Digital tvilling oppfører seg likt som en ekte thruster.
5. Digital tvilling har flere elementer som beveg
6. Simulatoren fungerer på samme måte som dagens "fysiske thruster simulator" i treningskofferten (Pelicasen).

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

Vi planlegger å lage ett Gant-diagram som viser planlagt fremdrift for å ha klare arbeidsmål og milepæler. Arbeidet skal deles opp i mindre deler for å enklere gjennomføre prosjektet. Dette vil gi oss en visuell oversikt over prosjektets tidslinje, inkludert milepæler for de ulike fasene, samt hvordan de overlapper og påvirker hverandre.

For utvikling og styring av prosjektoppgaven har gruppen bestemt seg for å bruke metoden Scrum. Denne metoden vil tillate oss å tilpasse oss raskt til eventuelle endringer og prioritere arbeidet effektivt. I Scrum vil vi organisere arbeidet i sprinter på to uker hvor et bestemt sett med oppgaver skal fullføres. Hver sprint vil avslutte med en oppsummering og planleggingssesjon for å definere målene og oppgavene som skal gjennomføres samt reflektere på det som er blitt gjort. Daglige stand-up møter vil sikre kontinuerlig kommunikasjon innad i teamet og gi en mulighet til raskt å løse eventuelle hindringer som oppstår.

Kombinasjonen av Gantt-diagrammet og Scrum vil gi oss en balanse mellom en klar, strukturert planlegging og fleksibilitet til å tilpasse oss dynamiske prosjektbehov. Mens Gantt-diagrammet vil gi oss et langtidsperspektiv og en oversikt over prosjektets fremdrift, vil Scrum-metoden gi oss fleksibilitet og effektivitet i den daglige gjennomføringen. Denne tilnærmingen sikrer at vi ikke bare holder oss til tidsfrister og milepæler, men også at vi kan levere kvalitetsarbeid gjennom kontinuerlig tilbakemelding og iterativ utvikling.

5.4 Informasjonsinnsamling – utført og planlagt

Prosjektoppgaven vil være en videreutvikling av en prototype som ble laget i løpet av sommeren 2023 og industriprosjektet som vi hadde forrige semester. Under denne perioden har gruppen allerede drøftet og kommet frem til nye ideer som kan utføres under prosjektoppgaven. Kunnskap som å bytte kodespråk fra Python til et raskere og mer effektivt kodespråk er blitt bestemt. En enkel metode for visualisering har også blitt laget, men skal bli revidert for å bli bedre. Visualiseringsverktøy som Unreal Engine eller Unity vil være aktuelle programmer for å løse dette. Gruppen har kjennskap til at systemløsningen ikke er robust og er derfor ikke klar til å bli brukt kommersielt med unntak av opplæring/kursing.

Det er gjennomført en analyse av eksisterende løsninger for å adressere problemstillingen, basert på samtaler med oppdragsgiver samt opplæringsansvarlige fra Kongsberg Maritimes avdelinger i Ålesund, Ulsteinvik og Kongsberg. Tilbakemeldingene fra disse samtalene indikerer en positiv holdning til produktets nødvendighet. Fremtidige løsninger vil bli undersøkt på samme måte, og det er planlagt et møte med programvareingeniører fra Kongsberg Maritime for å kartlegge nåværende og potensielle løsninger.

I forprosjektene har gruppen utviklet prototypen som består av 2 deler. Første delen henter data fra CAF og lagrer dataen på en OPC-UA server. Den andre delen bruker data fra serveren og visualiserer den med å bruke Ignition. Prototypen fungerer, men den er uforutsigbar og ustabil. Signalforsinkelsen varierer mellom 1 til 20 sekunder, og programmet må noen ganger startes flere ganger før det vil virke. I tillegg fører ofte programmet til at CAF krasjer, slik at kontrolleren må bli startet på nytt. Dette ville vært ett kritisk problem på en båt i drift. I tillegg er der ingen metode for to-veis kommunikasjon

og tilbakekobling dersom signalet forsvinner. Det er derfor planlagt å utbedre dette for å sikre robusthet og brukervennlighet i produktet.

5.5 Vurdering – analyse av risiko

Det er mulig at gruppen har store mål, men vi tror at prosjektet er gjennomførbart. På server siden er vi mest skeptisk på hvordan cyber security skal innføres, da gruppen har lite/ingen erfaring med dette. Angående simulator og visualiserings delen av prosjektet er gruppen optimistiske, fordi gruppen har noe erfaring med dette fra tidligere prosjekter.

Vurdering av hva som vil være særlig viktig for å lykkes, og hva som anses å være trusler mot suksess er blant annet:

- Ikke følge fremdriftsplan
- Ha for store planer
- Unngå unnasluntring
- Sykdom
- Skade
- Disiplin
- Feilprioriteringer
- Dødsfall

Vurdering av mulige risikoelementer, sikkerhetsaspekter og eventuell virkning på miljø knyttet til ønsket løsning (prosjektresultat):

- Cybersikkerhet
- Informasjonssikkerhet
- Konfidensialitet

Eventuell virking på miljøet er at produktet vil gjøre det lettere og mer interaktivt for en kursdeltaker å lære seg systemene til KM. Resultatet av dette gjør at service ingeniøren kan systemet bedre og vil for eksempel tune parametere på en båt mer korrekt som kan spare tid og drivstoff. Programmet har/vil også bidra til mer effektiv feilsøking dersom det skal brukes som ett service verktøy.

RISIKOANALYSE

Enhet/Institutt:	Department of ICT and Natural Science, NTNU
Prosjektnavn:	Next Generation Training Enviroment
Prosjektnummer:	Bachelorgruppe nr.7
Veileder (navn):	Ottar Osen, Robin T. Bye
Deltakere (navn):	Robert Alan Moltu, Torstein Ditlev Skare
Periode	Våren 2024

Beskrivelse av den aktuelle aktiviteten, området mv.:

Denne risikoanalysen gjelder prosjektarbeid inne i DP-training rommet til Kongsberg Maritime ved NMK i Ålesund. Her skal det prosjekteres og utvikles et nytt interaktivt training verktøy som skal effektivisere opplæring og arbeid i felt. I løpet av prosjektet vil det bli benyttet komponenter og materiell for testing. Derfor vil fysisk arbeid som oppkobling av Pelicase/testrigg foregå i Kongsberg Maritime og NTNU sine lokaler.

Aktivitet/ arbeidsoppgave	Mulig uønsket hendelse	Eksisterende tiltak	Sannsynlighet (S) (1-5)	Vurdering av konsekvens (K) *Menneske skal alltid vurderes* Menneske (1-5)	Materiell (1-5)	Ytre miljø (1-5)	Risikovurdi (S x K)	Forslag til forebyggende og/eller korrigendene tiltak	Sannsynlighet etter tiltak	Konsekvens etter tiltiak	Restrisiko etter tiltak (S x K)
Oppstart											
Tilstedeværelse på lab/lokale	Søle vann/kaffe på utstyr	Holde drikkevarer med god avstand til utstyr	2	0	2	0	4				4
Manipulering av pelicase/testrigg	Støt fra elektriske komponenter	Koble uten strøm på, Måle spenning før arbeid starter, Legesjekk ved uhell	2	1	1	0	2				2
Manipulering av pelicase/testrigg	Klemfare og løfteskade	Ingen	2	0	2	0	4				4
Montere/koble komponenter	Klemfare/ løfteskade/ støt / fysiske skader / materielle skader	Koble uten strøm på, Måle spenning før arbeid starter, legesjekk ved uhell.	2	2	3	0	6	Sikre arbeidsplass før start, dobbelsjekk at koblinger er riktig før man setter spenning på komponentene.	1	3	3
Løfting og montering av pelicase/testrigg	Klemfare og løfteskade	Påbudt verneutstyr	2	3	0	0	6	Bruke tilegnet utstyr som hindrer eventuelle skader	2	2	4
Demo/Eksperiment											
Program krasjer marinecontroller i pelicase/testrigg	Marinecontroller må restartes	Ingen	2	0	0	0	0				0
Program krasjer marinecontroller i på båt	Marinecontroller/båt må restartes og mister kontroll	Ingen	2	2	5	5	10	Ikke kjøre uferdig program/ utbedre robusthet	1	1	2
Nedrigging											
Løfting og montering	Klemfare og løfteskade	Påbudt verneutstyr	2	3	0	0	6	Bruke tilegnet utstyr som hindrer eventuelle skader	2	2	4
Kjemikalier											
Ikke aktuelt							0				0
Datasikkerhet											
Utvikle kode	Pcen kræsjer eller blir		2	0	4	0	8	Jevnlig ta backup av koden, gjerne med versjonskontroll, som f.eks. GitHub	1	2	2
Koble opp utstyr mot ethernettporter	Pelicasen blir feiltagelig koblet til internett og blir	Sjekke hvilke porter som er koblet til internett før man bruker de	1	0	4	0	4				0
Smittevern											
Dersom sykdom oppstår	Smittet av COVID-19 / influensa eller	Følge retningslinjene til NHI og NTNU	3	3	0	0	9	Renslighet, Vaske hender, muligheter for å jobbe hjemmefra	1	4	4

Risikotabell

		Ubetydelig konsekvens 1	Liten konsekvens 2	Moderat konsekvens 3	Alvorlig konsekvens 4	Katastrofal konsekvens 5
Svært høy sannsynlighet	5	Moderat (5)	Moderat (10)	Høy (15)	Høy (20)	Katastrofal (25)
Høy sannsynlighet	4	Lav (4)	Moderat (8)	Høy (12)	Høy (16)	Høy (20)
Moderat sannsynlighet	3	Lav (3)	Moderat (6)	Moderat (9)	Høy (12)	Høy (15)
Liten sannsynlighet	2	Lav (2)	Lav (4)	Moderat (6)	Moderat (8)	Moderat (10)
Svært liten sannsynlighet	1	Lav (1)	Lav (2)	Lav (3)	Lav (4)	Moderat (5)

5.6 *Hovedaktiviteter i videre arbeid*

Hovedaktiviteter og delaktiviteter er beskrevet i Gantt diagrammet som ligger vedlagt. Det er ikke forventet noen kostnader, da gruppen bruker får bruke fasiliteter som allerede er innkjøpt. På grunn av dette er kostnader foreløpig ikke inkludert.

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Hovedplanen er i grove trekk som vist i listen under. Se vedlagt Gantt diagram for mer detaljer om de forskjellige punktene.

1. Planleggingsfase
2. Utbedre/fikse C++ server.
3. Utvidelse av funksjonalitet i systemet.
4. Forenkle oppstart av server (.exe fil?)
5. Simulere og visualisering (digital tvilling)

5.7.2 Styringshjelpemidler

Oversikt over hjelpemidler en ønsker å bruke i arbeidet med å styre prosjektet:

- Microsoft Project/Planner.
- Wiki.
- Excel.

5.7.3 Utviklingshjelpemidler

Oversikt over hjelpemidler en vil ha behov for eller ønsker å bruke i arbeidet med å gjennomføre prosjektet (Dette kan ses i sammenheng med punkt 9):

- ChatGPT, Copilot og andre AI-tjenester.
- Clion (C++)
- VScode (Python)
- Pycharm (Python)
- Unreal Engine
- Ignition
- Putty
- WinSCP

5.7.4 Intern kontroll – evaluering

Gruppa tenker å bruke testlister for å evaluere underveis og til slutt. Dette inkluderer: testlister for robusthet, sikkerhet, funksjonalitet og simulator.

Det planlegges å bruke Git for versjonskontroll under programvare utvikling.

Gruppa har planlagt ukentlige SCRUM møter for å reflektere og planlegge arbeid.

Dele oppsummerings rapporter sprint møter når vi møter med veileder/oppdragsgiver.

5.8 Beslutninger – beslutningsprosess

Informasjon om hvordan beslutninger om avgrensning / presisering av oppgaven og andre sentrale beslutninger har blitt tatt under arbeidet med forprosjektet.

Oversikt over hvordan viktige beslutninger planlegges tatt under arbeidet med hovedprosjektet. Dette gjelder hovedområder og viktige avgjørelser som skal/må tas underveis i arbeidet, slik det er forutsatt i hovedplanen (5.7.1).

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Planleggingsfase:

- Lage Gant diagram
- Lage Ukeplan
- Lage Scrum plan
- Planleggingsmøte med veiledere/oppdragsgiver

Utføringsfase:

- Skrive rapport underveis i prosjektet.
- Ukentlig oppdatering av fremdriftsplan
- Sprint rapporter hver 14. dag
- Møte med veiledere/oppdragsgiver hver 14. dag.

Avslutningsfase

- Leveranse av prosjekt

For å opprettholde gode rutiner skal vi følge planene og normene som bestemt i punkt 4.3 og 7.1
For distribusjon/kopiering skal gruppa følge Kongsberg Maritimes regler for å opprettholde konfidensialitet. Dokumenter blir oppbevart på Microsoft Teams. Distribusjon til sensor/veiledere vil skje på Inspera eller E-post.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 *Møter*

7.1.1 Møter med styringsgruppen

- Annenhver uke torsdager kl. 10:00-10:30 (Justerer iht. veiledere)

7.1.2 Prosjektmøter

- Planlagte møtedatoer/tidspunkt – hensikt
- Gruppa planlegger å møtes hver arbeidsdag som ikke er avsatt til emnet INGA2230.
- Ordinær arbeidstid for bachelor blir 08:00 til 16:00.
- Stand-up møte hver morgen
- Sprint oppsummering og planlegging hver fredag

7.2 *Periodiske rapporter*

7.2.1 Framdriftsrapporter (inkl. milepæl)

Annenhver uke skal en framdriftsrapport (sprint rapport) bli lagt frem på møte med veileder/oppdragsgiver. Milepæler vises i Gant diagram og vil bli nevnt i framdriftsrapportene.

8 PLANLAGT AVVIKSBEHANDLING

Dersom prosjektet ikke går som planlagt tenker gruppen å arbeide utover normal arbeidstid. Dersom dette ikke er tilstrekkelig planlegger gruppen følgende:

Ansvarsområde

- Det er forventet at gruppemedlemmene skal begrunne hvorfor ansvarsområdet ikke går etter planen. Slik at gruppa kan finne ei løsning. Eventuelt kan også veileder eller oppdragsgiver konsulteres for å finne en løsning eller endre prioriteringer.

Planlagt prosedyre for endringer

- Dersom endringer i prosjektet oppstår, skal gruppa konsultere veiledere/oppdragsgivere.

Ansvar

- Det er forventet at gruppemedlemmene tar ansvar for prosjektet og egne ansvarsområder, samt vise engasjement. Det er også forventet å kontinuerlig samarbeide og ha god kommunikasjon.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

- Peliacse (KM)
- MCON-simulator (KM)
- Wago PLS (NTNU)
- Arbeidslokaler (KM)

10 REFERANSER

VEDLEGG

Vedlegg ligger i en egen mappe i forprosjekt mappen.

Vedlegg 1	Industriprosjektrapport
Vedlegg 2	Risikovurdering
Vedlegg 3	SPRINT UKE 5-6.
Vedlegg 4	Gantt diagram

8.5 Gantt Chart using MS Planner

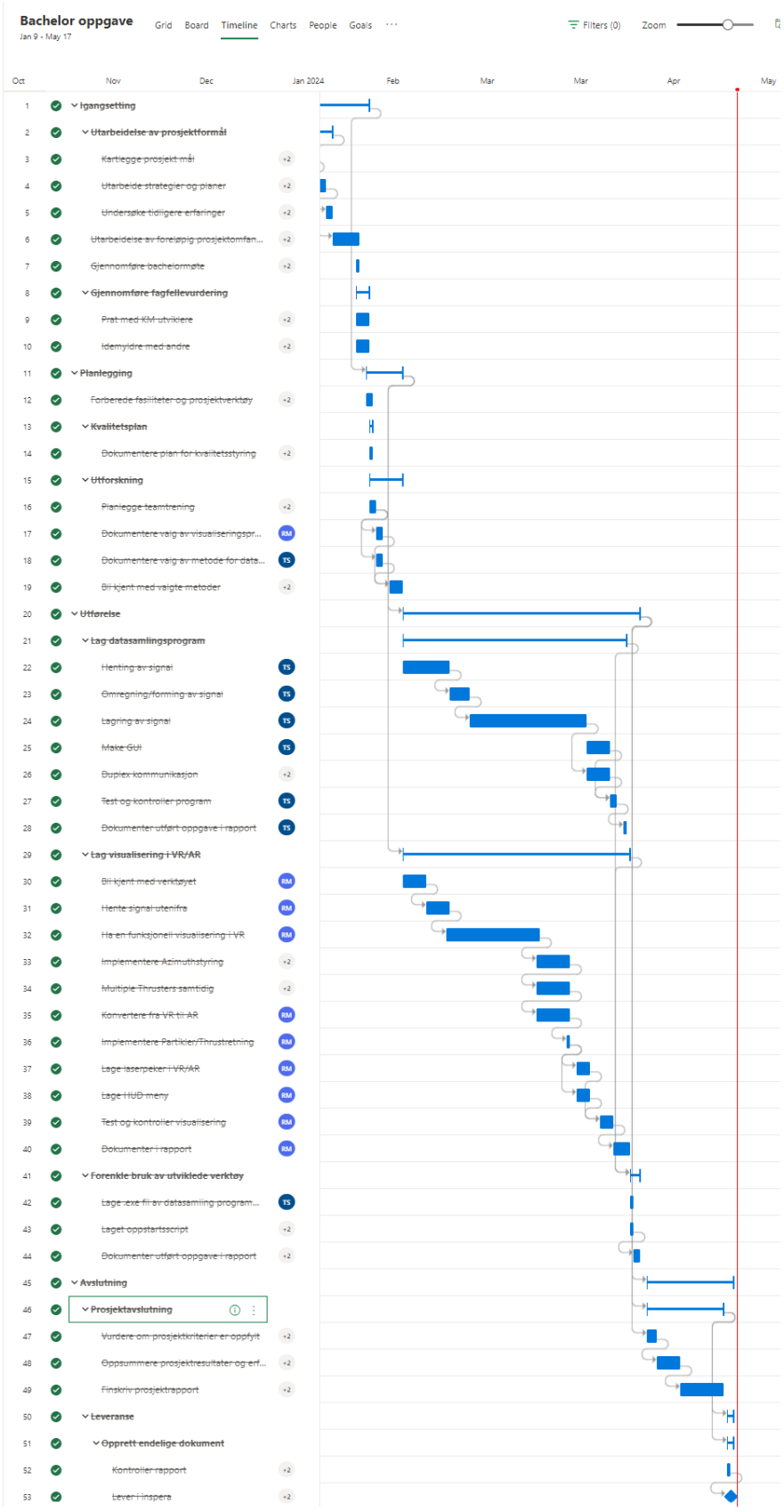


Figure 49: Gantt Diagram of Bachelor Project Showing Timelines and Milestones.

8.6 Timesheet in Excel

Frem til 13.mars er man,tir,ons brukt til INGA

Timeplan

uke	Dag	Dato	Torstein	Kommentar	Logg	Robert	Kommentar	Logg
1	Mandag	08.01.2024	4,00			4,00		
	Tirsdag	09.01.2024	7,50	*Riggbygging*		7,50	*Riggbygging*	
	Onsdag	10.01.2024	7,50	*Riggbygging*		7,50	*Riggbygging*	
	Torsdag	11.01.2024	7,50	*Forprosjektrapport*		7,50	*Forprosjektrapport*	
	Fredag	12.01.2024	0,00	*INGA*		0,00	*INGA*	
	Lørdag	13.01.2024						
	Søndag	14.01.2024						
2	Mandag	15.01.2024						
	Tirsdag	16.01.2024						
	Onsdag	17.01.2024						
	Torsdag	18.01.2024	7,50	*Tur på båt*		7,50	*Tur på båt*	
	Fredag	19.01.2024	7,50			7,50		
	Lørdag	20.01.2024						
	Søndag	21.01.2024						
3	Mandag	22.01.2024	3,00			3,00		
	Tirsdag	23.01.2024	4,00			4,00		
	Onsdag	24.01.2024						
	Torsdag	25.01.2024	7,50	*Forprosjektrapport ferdigstillielse*		7,50	*Forprosjektrapport ferdigstillielse*	
	Fredag	26.01.2024	8,50			8,50		
	Lørdag	27.01.2024						
	Søndag	28.01.2024						
4	Mandag	29.01.2024						
	Tirsdag	30.01.2024						
	Onsdag	31.01.2024						
	Torsdag	01.02.2024	7,50	*Visualisering*		7,50	*Server*	
	Fredag	02.02.2024	7,50	*Visualisering*		7,50	*Server*	
	Lørdag	03.02.2024						
	Søndag	04.02.2024						
5	Mandag	05.02.2024						
	Tirsdag	06.02.2024						
	Onsdag	07.02.2024	4,00	*Riggbygging*		4,00	*Riggbygging*	
	Torsdag	08.02.2024	7,50	*Møte med KM utvikling* *Teste REST API*	*Fant ut hvordan REST API skal brukes oppmot serveren på marinecontrolleren.*	7,50	*Riggbygging/Visualisering*	
	Fredag	09.02.2024	7,50	*Lage python kode som henter data med REST API*	*Fikk koden til å fungere*	7,50	*Unreal Visualisering*	
	Lørdag	10.02.2024						
	Søndag	11.02.2024						
6	Mandag	12.02.2024						
	Tirsdag	13.02.2024						
	Onsdag	14.02.2024						
	Torsdag	15.02.2024	7,50	*Møte med Morild* *Rapportskriving*	*Skreve innledning og litt teori*	7,50	*Møte med Morild*	
	Fredag	16.02.2024	7,50	*Rapportskriving. Visualisering*	*Enkel AR app på mobilen*	7,50	*Visualisering*	Lastet ned Unity og prøvde å lære litt programvaren. Fikk testet en enkel AR app på telefon.
	Lørdag	17.02.2024	7,50	*Lage c++ kode som bruker REST API*	*Fikk ikke teste rèelt pga sykdom*			
	Søndag	18.02.2024	7,50					
7	Mandag	19.02.2024	7,50			7,50	*Visualisering*	
	Tirsdag	20.02.2024	7,50			7,50	*Visualisering, thruster movement*	Laget movement script som beveger thruster modellen slik som threepp, begynt smått på kommunikasjon mellom marine controller og unity men ingenting som funker endå
	Onsdag	21.02.2024	7,50			7,50		
	Torsdag	22.02.2024	7,50	*Kommunikasjon/koding*	*Vi har bestemt oss for å lage produktet med Python først for å prototype enklere. Laget en UDP client i python og fått kommunikasjon med Unity på en annen pc. Laget en OPC_UA server. har problemer med å laste opp til serveren.*	7,50	*Visualisering/Kommunikasjon*	
	Fredag	23.02.2024	7,50			7,50		
	Lørdag	24.02.2024						
	Søndag	25.02.2024						
8	Mandag	26.02.2024	7,50			7,50		
	Tirsdag	27.02.2024	7,50			7,50		
	Onsdag	28.02.2024	7,50			7,50		
	Torsdag	29.02.2024	7,50	*Kommunikasjon/koding*	*Fikk til å laste opp til fra client serveren. Løsningen var å lage alle objekter på server-siden å bare sette verdiene fra clienten. Fikk også til å laste opp "live" verdier fra CAF ved bruk av websocket. Har bestemt å bruke websocket til verdier som skal oppdateres raskt og Rest api til verdier som kan ta lenger tid. Dette er fordi websocket bruker for mye prosessorkraft når den skal hente alle topics. Rest Api klarer effektivt å hente all data, men bruker omtrent 0.3 sekund per objekt. Siden vi har 22 objekter blir forsinkelsen rundt 6 sekund som er litt tregt.*	7,50		
	Fredag	01.03.2024	7,50	*Laget klient som sender data fra OPC_UA serveren til unity ved bruk av UDP. Meldingen sendes som en touple (string,value)*	*Fikset bug i komunikasjonskoden. heiltall virket ikke. string,bool,array osv. virker fortsatt ikke.*	7,50		
	Lørdag	02.03.2024	7,50					
	Søndag	03.03.2024	7,50					
9	Mandag	04.03.2024	7,50			7,50		
	Tirsdag	05.03.2024	7,50			7,50		
	Onsdag	06.03.2024	7,50			7,50		
	Torsdag	07.03.2024	7,50			7,50		
	Fredag	08.03.2024	7,50			7,50		
	Lørdag	09.03.2024		*Koding/Kommunikasjon*	*Endret server koden slik at den oppretter alle variabler som riktig data type. Innser at ditte kanskje ikkje va lurt, da vi kjem til å koble oss opp mot forskjellige thrusterer samtidig i fremtiden, og da vil det være bedre om clienter kan lage variabler selv.*			
	Søndag	10.03.2024						
10	Mandag	11.03.2024	7,50			7,50		
	Tirsdag	12.03.2024	7,50			7,50		

	Onsdag	13.03.2024	7,50			7,50		
	Torsdag	14.03.2024	7,50			7,50	Se og lære hvordan vannfysikk fungerer i spill/visualisering	
	Fredag	15.03.2024		jobbreise		7,50	Teste forskjellige "water" assets i unity	
	Lørdag	16.03.2024		jobbreise				
	Søndag	17.03.2024		jobbreise				
11	Mandag	18.03.2024		jobbreise		7,50	Teste forskjellige "water" assets i unity	
	Tirsdag	19.03.2024		jobbreise		10,50	Teste forskjellige "water" assets i unity	Fant ut at vannshader til nyeste versjon av Unity i HDRP var det beste alternativet i VR.
	Onsdag	20.03.2024		jobbreise		7,50	Lage Thrust Partikler fra Thruster	
	Torsdag	21.03.2024	7,50			7,50	Lage Thrust Partikler fra Thruster	
	Fredag	22.03.2024	7,50			7,50	Lage Thrust Partikler fra Thruster	
	Lørdag	23.03.2024						
	Søndag	24.03.2024						
12	Mandag	25.03.2024	7,50			7,50	Bestille Meta Quest 3 for AR-implementasjon	
	Tirsdag	26.03.2024	7,50			7,50	Anskaffe Ny Ut-Båt design, legge inn på Unity	Importere modell fra Autodesk 3DS Max til en FBX fil slik at vi kan åpne fila i Blender for å editere og "skrelle" modellen for ubrukte objekt.
	Onsdag	27.03.2024	7,50			7,50	Editere UT-Båt 3D-modell, fjerne ubrukte ting inne i Blender.	
	Torsdag	28.03.2024	7,50			7,50		
	Fredag	29.03.2024	7,50			7,50	Plassere Thrustere på UT-Båt	
	Lørdag	30.03.2024						
	Søndag	31.03.2024						
13	Mandag	01.04.2024	7,50			7,50	Plassere Thrustere på UT-Båt	
	Tirsdag	02.04.2024	7,50			7,50		
	Onsdag	03.04.2024	7,50			7,50	Anskaffe AR briller og Teste, konvertere fra VR til AR.	
	Torsdag	04.04.2024	7,50			7,50		
	Fredag	05.04.2024	7,50			7,50		
	Lørdag	06.04.2024						
	Søndag	07.04.2024						
14	Mandag	08.04.2024	7,50			7,50		
	Tirsdag	09.04.2024	7,50	Koding/Kommunikasjon	Fikk til kommunikasjon med flere thrustere. Suksess i å formatere melding som json og sende over udp til Unity. Klarte å konvertere programmet til kjørbar .exe fil. la til data om target udp i communicationData.json	7,50		
	Onsdag	10.04.2024	7,50			7,50		
	Torsdag	11.04.2024	9,50	Koding/Kommunikasjon	Fikset kliss i kommunikasjonen ved å endre funksjonen som henter data. Siden vi nå begrenser oss til noen få variabler som vi skal visualisere, kan vi bruke 1 ws connection per topic. det ser ut til å fungere fint.	7,50		
	Fredag	12.04.2024	7,50			7,00		
	Lørdag	13.04.2024						
	Søndag	14.04.2024						
15	Mandag	15.04.2024	7,50			7,50		
	Tirsdag	16.04.2024	7,50			7,50		
	Onsdag	17.04.2024	7,50	Koding/Kommunikasjon	restapi og websocket clientene virker og kan kjøre separat	7,50		
	Torsdag	18.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Fredag	19.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Lørdag	20.04.2024						
	Søndag	21.04.2024						
16	Mandag	22.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Tirsdag	23.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Onsdag	24.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Torsdag	25.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Fredag	26.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Lørdag	27.04.2024						
	Søndag	28.04.2024						
17	Mandag	29.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Tirsdag	30.04.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Onsdag	01.05.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Torsdag	02.05.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Fredag	03.05.2024	7,50	Rapportskriving		7,50	Rapportskriving	
	Lørdag	04.05.2024						
	Søndag	05.05.2024						
18	Mandag	06.05.2024	12,00	Rapportskriving.		12,00	Rapportskriving.	
	Tirsdag	07.05.2024	12,00			12,00		
	Onsdag	08.05.2024	12,00	Testing	Teste program på stasjonær pc. Installere unity versionskontroll for å enklere flytte programmet	12,00	Testing	Teste program på stasjonær pc. Installere unity versionskontroll for å enklere flytte programmet
	Torsdag	09.05.2024	12,00	Feilsøking	Fikse problem som oppstod med unity version control. Feilen var at vi hadde brukt port 5000, som ble opptatt av UVC	12,00	Feilsøking	Fikse problem som oppstod med unity version control. Feilen var at vi hadde brukt port 5000, som ble opptatt av UVC
	Fredag	10.05.2024	12,00	Testing/forbereding	Teste att programmet virker. Koblet opp 2 pelicase og fergeinstallasjon mot systemet vårt. Planla litt film. Møte med foredragsholder Eli Anne Tvergrov for tips til endelig presentasjon og film	12,00	Testing/forbereding	Teste att programmet virker. Koblet opp 2 pelicase og fergeinstallasjon mot systemet vårt. Planla litt film. Møte med foredragsholder Eli Anne Tvergrov for tips til endelig presentasjon og film
	Lørdag	11.05.2024						
	Søndag	12.05.2024						
19	Mandag	13.05.2024	9,00	Planlegging/Forbereding	Finne utstyr til filming. Planlegge film. Skrive manus	9,00	Planlegging/Forbereding	Finne utstyr til filming. Planlegge film. Skrive manus
	Tirsdag	14.05.2024	9,00	Filming. Lydopptak. Redigering	Filming. Lydopptak. Redigering	9,00	Filming. Lydopptak. Redigering	Filming. Lydopptak. Redigering
	Onsdag	15.05.2024	9,00	Dokumentering	redigering. Rapportskriving	9,00	Dokumentering	redigering. Rapportskriving
	Torsdag	16.05.2024	9,00	Dokumentering	Rapportskriving/finpussing	9,00	Dokumentering	Rapportskriving/finpussing
	Fredag	17.05.2024						
	Lørdag	18.05.2024						

	Søndag	**19.05.2024**	7,50	*Finpuss*	*finpussing på rapport. Legge til alle appendix. Lage endelig dokument*	7,50	*Finpuss*	*finpussing på rapport. Legge til alle appendix. Lage endelig dokument*
20	Mandag	**20.05.2024**	5,00	*Avslutning*	*Levere endelig dokument. Forberede fremføring*	5,00	*Avslutning*	*Levere endelig dokument. Forberede fremføring*
	Tirsdag	**21.05.2024**	7,50	*Avslutning*	*Fremføring. Feiring*	7,50	*Avslutning*	*Fremføring. Feiring*
	Onsdag	**22.05.2024**						
	Torsdag	**23.05.2024**						
	sum		644,00			644,50		

8.7 Minutes of the meeting with Morlid Interactive

Meeting with Morild
Date: February 15, 2024
Location: Daeskogen
Participants: Torstein Skare, Robert Moltu, Per Magne Dalseth, Geir Olav Otterlei

The meeting was convened to discuss how the Morild's offshore simulator and expertise could be leveraged in our bachelor's thesis. After brainstorming, it was decided that Morild would develop a front-end visualization application that can send and receive signals. Additionally, it was agreed that the project team will provide the 3D models to be visualized and specify the command requests to be sent via UDP to retrieve the values. Since the Morild could not deliver the application until early may, it was also mentioned that the project group should create a basic visualization using unity in order to test the communication.

8.8 Meeting minutes with Supervisors

Stikkord fra møtet:

- Sikkerhet og robusthet
    - Med målsetning om at verktøyet skal brukes i felt
- Utvidelse og optimalisering av funksjonalitet i programmet
    - To-veis kommunikasjon
    - Automatisk konfigurering fra configfil
    - Mer effektiv datahenting fra CAF
    - Gjøre programmet executable / mer brukervennlig
- PLS simulator for sammenkobling og mer reell opplæring
    - Visualisering

- Tenk om igjen på ting som er blitt gjort (Drøft, sett opp alternativ)
- Nyskapende (Visualisering)
- OSI-modell (nettverkskommunikasjon)
- Eige data (omregninger i server istedenfor Ignition)
- WAGO Compact100 (snakk med anders) / IOT-Edge Controller Wago

Møtereferat:

Etter møter med både KM og faglærer, har bachelor gruppen kommet frem til ett par viktige punkt som er ønskelig å oppnå i løpet av prosjektet. Tema som robusthet/sikkerhet, utvidelse og optimalisering av programmet, PLS-simulator for sammenkobling og visualisering har vært nevnt som viktige punkt å gå videre med i prosjektet. Gruppen har som mål ved endt bachelor at programmet skal være mulig å bruke i felt, samtidig som at det skal være et nyskapende og teknologisk produkt/program. Dette planlegger vi å få til ved å utbedre prototypen og forsvare/endre noen av valgene som er blitt gjort, valg som f.eks. å bruke OPC-UA som server.

Planen er nå å følge 3 hovedpunkt som er:

- Sikkerhet og robusthet
    - Gjøre programmet robust nokk til at det ikke krasjer CAF
    - Noe cyber security
- Utvidelse og optimalisering av funksjonalitet i programmet
    - To-veis kommunikasjon (først med CAF)
    - Automatisk konfigurering av ip-adresser fra config-fil i marinecontroller.
    - Mer effektiv datahenting (Få C++ serveren til å virke)
        - Dele opp programmet i henting, beregning og lagring, slik at beregninger skjer i server og ikke i GUI(Ignition).
    - Gjøre programmet mer brukervennlig.
        - Lage .exe fil?
    - Oppstarts script?

- Bruke en PLS til å lage en thruster simulator, slik at vi kan fjerne hardware fra nåværende testrigg (Pelicase) (Låne foreksempel Wago Compact100 fra skulen?).
    - Digital tvilling av thruster(Unreal Engine? Fortsette med Threepp? Codesys?)

Møtereferat

Dato: 26.1.24
Lokasjon: NMK "Heland"
Deltagere: Torstein, Robert,
Ottar, Robin, Håkon

Spørsmål til møte:

- Håndskrevne spm.

Stikkord fra møte:

- Få frem akademisk med gode tester for robusthet
- Autogenererte config-filer
- Testing siden,
- Svart boks for mange,
- Design, bør være likt som system til KM
- Etikk, ulovlig distrubering slik at folk får tak i det, PM mista jobben sin.
- LCA, Gjør programmet uavhengig slik at fleire produkt kan brukast. (Generaliserbarhet) bruke på 1000 ting.
- Bærekraftighet, reise, effektiv tuning osv.
- Logging: Bruken av programmet, kor, kor ofte, kor lenge, ka folk trykk på osv. (Måle ting, om folk blir flinkare, maskinlæring) GDPR? Lære om læringen.
- Møter annenhver uke, Torsdager 11:30 til 12:00 HELAND/ANDRE ROMM
- Savner opplæringsbehovet, å vise fornuftigheten.
- Testing av kurs service folk
-

Referat fra møte:

I løpet av dette møte har prosjekt gruppen fått nyttig informasjon fra veiledere og oppdragsgiver. Ett gjennomgående tema er å få dokumentert arbeid som er blitt gjort på en akademisk måte, samtidig som at det forklarer behovet og fornuftigheten bak prosjektet. Det å ta ett skritt tilbake og vurdere hva som er blitt gjort og hvordan prosjektet kan bli løst på best mulig måte er viktig. Når det kommer til utvidelse av funksjonalitet til produktet, er det blant annet snakket om muligheten for autogenererte flyt diagram fra config-filer. I tillegg er tanken om å redegjør tester for programmet god, å lage tester i programmet og sette krav til hvordan programmet bør funke er smart. Videre funksjonalitet som design og bør være gjennomtenkt og bør se ut som et KM-produkt og ha en type «Standard».

Iløpet av møtet ble det også snakket om etikk, bærekraftighet og LCA. For etikk så er tema som at kursholdere kan få mindre arbeid, ulovlig distribuering og bedre opplæring på steder som vanligvis ville fåt dårlig opplæring eksempel på etiske problemstillinger til produkter. I tillegg ble en ide om å lage en metode for logging av bruken av programmet, som forteller hvor ofte, lenge eller hva folk trykker på i programmet et alternativ som gruppen kan jobbe med å legge til i prosjektet. I denne logginga kan man f.eks også bruke maskinlæring for å få nyttig informasjon. Et annet viktig tema for oppgaven er å få med behovet og vise fornuftigheten og behovet til produktet. Det er planlagt at fysisk demo/testing av produktet i ett kurs vil være kjempenyttig for å få feedback fra kursdeltagere og kursholdere. Og ett annet kurs på slutten av bacheloren for å få et godt resultat bit å skrive om. Det ble også nevnt å holde møter

Møtereferat

Dato: 26.1.24
Lokasjon: NMK “Heland”
Deltagere: Torstein, Robert,
Ottar, Robin, Håkon

annenhver uke på torsdager kl.11.30 til 12:00 dersom gruppen trenger mer tid på møte skal vi bli enige om å ha ett utvidet møte- i forkant av møte.

«Formål og mål, med delmål på veien. Og hvordan man adresserer oppgavene.»

Spørsmål til neste møte:

Stikkord fra møte:

- Tenk gjennom metodevalg og oppgaven
- Hva tjener man ved å bruke WS eller Proto (RestAPI) eller MQTT.
- Visualisering, simulere verdier til thrusteren for å få smooth signal (Ha en ide om ka som er behov å trene på)
- System som lagrer data (Playback funksjonalitet)
    - Er det live data fra en båt
    - Eller data fra koffert
    - Kor data kjem fra
    - Koble seg til en database istedet for å koble seg til en koffert
- Kor man bør ligge i løpet
    - Husk å lever gantt diagram og fyll ut skjema ligg på blackboard til veiledere. Legg det i innkallingen.
- Zoome litt ut få overblikk.
- Ka betyr resultatet, vise litt forståelse (når det kjem til rapport)
-

Referat fra møte:

I løpet av møtet fikk gruppen en bedre forståelse og mer overblikk over hvordan selve bachelor oppgaven bør gjennomføres. Gruppen ble også kjent med at til neste møtet skal det leveres inn relevante dokumenter som viser fremgang til veiledere (Gantt diagram og div) i tillegg til dette ble vi enige om at å både benytte Gantt diagram og SCRUM ikke var nødvendig, derfor ble det bestemt på møtet å benytte kun Gantt diagrammet for prosjektstyring. Før møtet hadde gruppen kartlagt litt potensielle metoder for datahenting til serveren der valget mellom å utnytte websockets eller å benytte Rest API(Protobuff) må bli dokumentert i rapporten. For visualisering ble vi enige om at visualiseringen skal dekke behovene til training og ha en ide på hva som er behovet å trene på.

En ide om å ha et system som lagrer data (Playback funksjonalitet) var også diskutert som en god ide som en funksjonalitet til training programmet. Dette for å lagre data som har vært live på enten en båt eller en koffert. Slik at man kan se hvor dataen kommer fra og spille tilbake det som har skjedd. Dette vil også åpne for en mulighet for at vi kan koble oss til en database trådløst istedet for å koble seg fysisk til en koffert/båt.

8.9 Periodic reports

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 0 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 1 av 3
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24

Main goal/purpose for these periods work

- Discover options and solutions for communication and visualization

Planned activities this period

- Discuss solutions with Kongsberg developers
- Discuss with other relevant people
- Improve facilities and tools
- Make plans for quality control
- Create program to retrieve data

Actually conducted activites this period

- Conducted meeting with KM developers
- Conducted meeting with Morild Interaktiv

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

-

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

- After having a meeting with Morild our plans for the visualization part of the project changed our plan a little. Initially the plan was to utilize Unreal Engine as the main game engine for developing the visualization, but after the meeting with Morild the group decided it was best to utilize Unity for making the thruster visualization. The primary reason for transitioning towards Unity was to utilize a platform that Morild was using in case of a partnership between us therefore using the same platform seemed beneficial, but also the group got some good input from them on how to go through with the visualization. Additionally, another big reason to transition towards Unity was because of the documentation for this game engine compared to Unreal Engine. Although Unreal Engine might be the newest and the best graphical looking engine Unity has been out for the longest period and is therefore the most used engine, making it easier for us to look up documentations, tutorials, and forums for us to complete our desired goals.
- A new method to collect data has been found and made in Python. The team has found that REST API seems to look beneficial for our desired goals instead of using websockets to collect data. This is because REST API allows us to retrieve objects containing multiple datapoints which we can separate from the object on the client end. Further testing of REST API is our next plan of action.

Main experience from this period

- Meeting with Morild has given the group a further understanding of game/visualization development and changed some of our opinions on the project's visualization part. It has also given us useful knowledge and connections within the business/industry.
- Talking to developers within Kongsberg Maritime led us to test REST API which has increased our knowledge about websockets and REST API and the differences between them.
- Visualization development has proven to be more effective after transitioning to Unity. The process of implementing the movement in Unity was similar to my approach in Threepp. The logic governing the rotation of each object was similar between the two methods.

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 0 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 2 av 3
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24

Main purpose/focus next period

- Set up communication between Unity and the Marine controller

Planned activities next period

- Create a functioning visualization in Unity
  - Functioning with values from marine controller
  - 3D-Visualsation on a screen
  - Look at adding water effects
  - Look at adding Olympic Octopus in and placing the thruster on the ship.
- Further test and develop our REST API client.
  - Check latency with many topics (~400)
- Create a program to convert the received data to appropriate datatypes/values. For example, booleans are received as integers 0 and 1. Another use case could be to interpolate data.

Other

- Since the meeting was cancled, we chose include some additional questions below

Wish/need for counceling

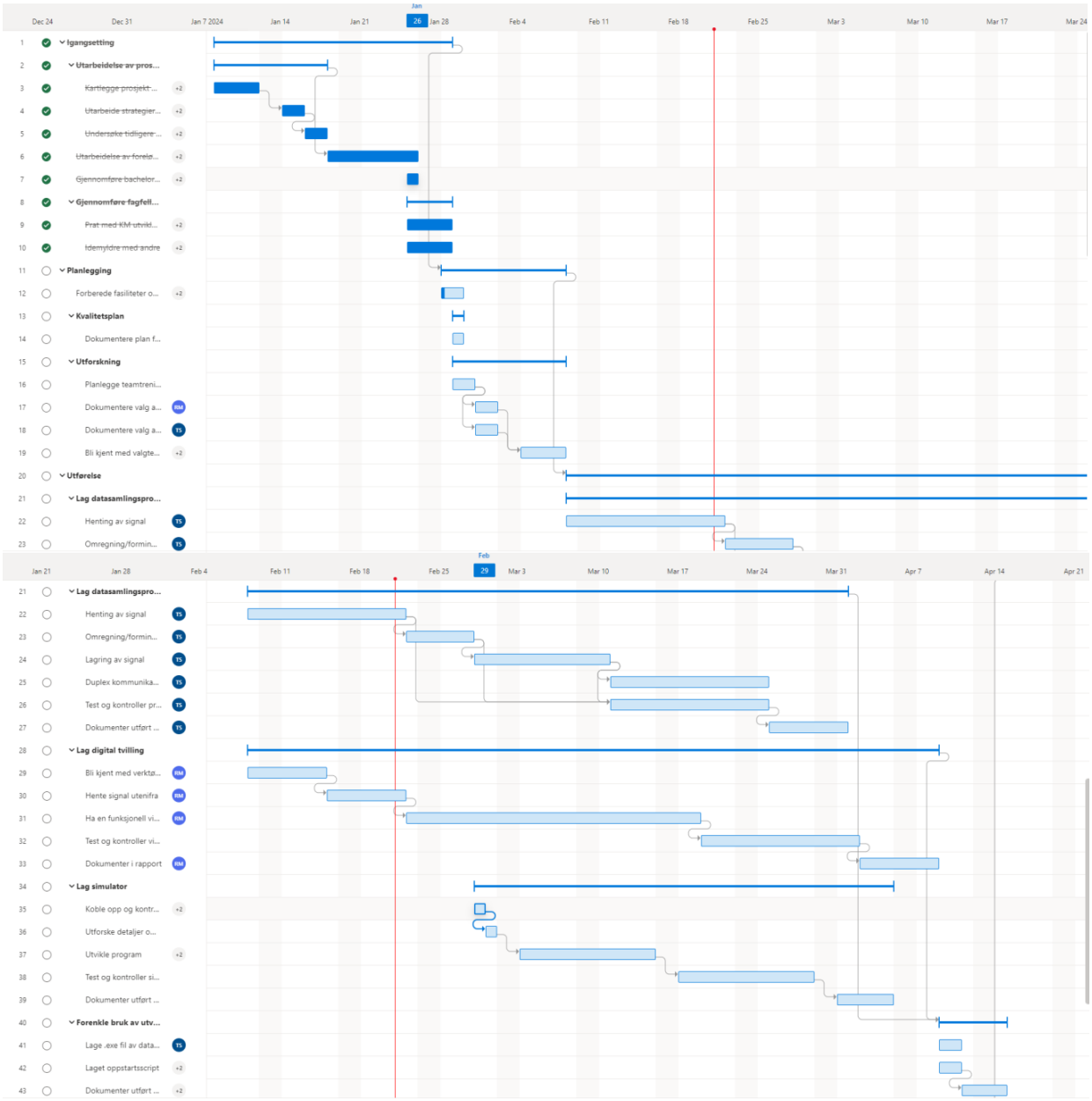
- Citation for logos and copying from Wikipedia for the theoretical part of the report.
- Does NTNU have Meta Quest 3? We are planning to test AR (Augmented Reality) visualization for the digital twin, which depends on us having the correct equipment to test.
- Should we focus on developing rapidly using Python, or to create an efficient program with C++? We believe that testing with Python is smart.
- Should we write the next report in Norwegian?

Approval/signature group leader	Signature other group participants
Torstein Skare Robert Moltu	-

GANNT-Diagram showing the period that is relevant for the coming periods:

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 0 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 3 av 3
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24



1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 1 av 4
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24

Main goal/purpose for these periods work

- Use discoveries from last period to connect communications script with visualization and apply it to VR.

Planned activities this period

- Develop thruster visualization on computer screen
    - Thruster movement
    - UDP-communication
    - Get 3D model of UT-Boat
    - Utilize Meta Quest 2 for visualization
        - Make Objects Grabbable
        - Make Objects Scalable (Sub-optimal but partially functionable)
    - Interpolate values coming from communication script
    - Document findings in report
- Develop server for storing data from CAF.
- Make client that can move data from CAF to our server.

Actually conducted activites this period

- Scheduled periodic meetings with supervisors
- Meeting with Arne Styve to get Meta Quest 2 VR headset
- All points that were planned for this period were conducted except documenting the findings in the report.
- Created simple OPC_UA server
- Created a client that uses websocket to retrieve a small list of topics from CAF and store the data on OPC_UA server.
- Created a client that retrieves data from OPC_UA and sends the data by UDP

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

- The progress within the project has been very good this period and a lot has been accomplished, but because of a report and an exam in another course (INGA) large parts of this period have gone in to studying for this subject. Which is why documenting the findings in the report has been de-prioritized. This will be done as soon as possible.

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

- We have decided to make the communication part of the project in separate modules such as server, websocket client and RESTAPI client. This is because neither of the methods are designed to deliver all the data we need at “realtime.” Using websockets allows us to gather and send data faster than RestAPI and will be used for the most important data coming from the marine controller. Therefore, will websockets be used for the visualization part of the project Unity/Ignition, while the rest of the data can be retrieved using RestAPI. RestAPI introduced around 0.3 seconds of delay per object (we are interested in around 20 objects. 0.3*20=6seconds total)

Main experience from this period

- Visualization development has proven to be effective in Unity. By using the Meta XR all in one SDK allows us to be able to develop a VR visualization.

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 2 av 4
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24

- Achieved communication between Marine Controller – Communication Script – Visualization, which was a good experience.
- Mostly experienced how the CAF server works. When using websockets to retrieve data from the server, it seems like ~40 topics is the limit and anything above does not get updated at a frequent interval. This limit could be increased to 80 because the server supports batch subscriptions, but the limit is 2.

Main purpose/focus next period

- Look at a solution to get communication from more than one marine controller
- Improve visualization
  - Scaling of object
  - Method for multiple marine controllers
  - Applying thruster model to UT-Boat
- Try to get from KM/NTNU
  - Getting in new models from KM(Azimuth/TCNS)
  - Meta Quest 3
- Improve communication
  - Make every relevant datatype work.
  - Make GUI to start/stop and to set information such as IP-addresses and number of connections.

Planned activities next period

- Create a program to convert the received data to appropriate datatypes/values. For example, booleans are received as integers 0 and 1.
- Search for a method to know which variables must be updated
- Make a new script that uses the other scripts as modules. (Hard because of asynchronous programming)
- Fix suboptimal scaling feature in visualization.
- Look at the possibility for multiple marine controllers/thrusters running simultaneously.
- Request a Meta Quest 3 headset
- Apply thruster models to a UT-Boat

Other

- Postponed creation of a simulator to march.14. So we can finish our current tasks.

Wish/need for counceling

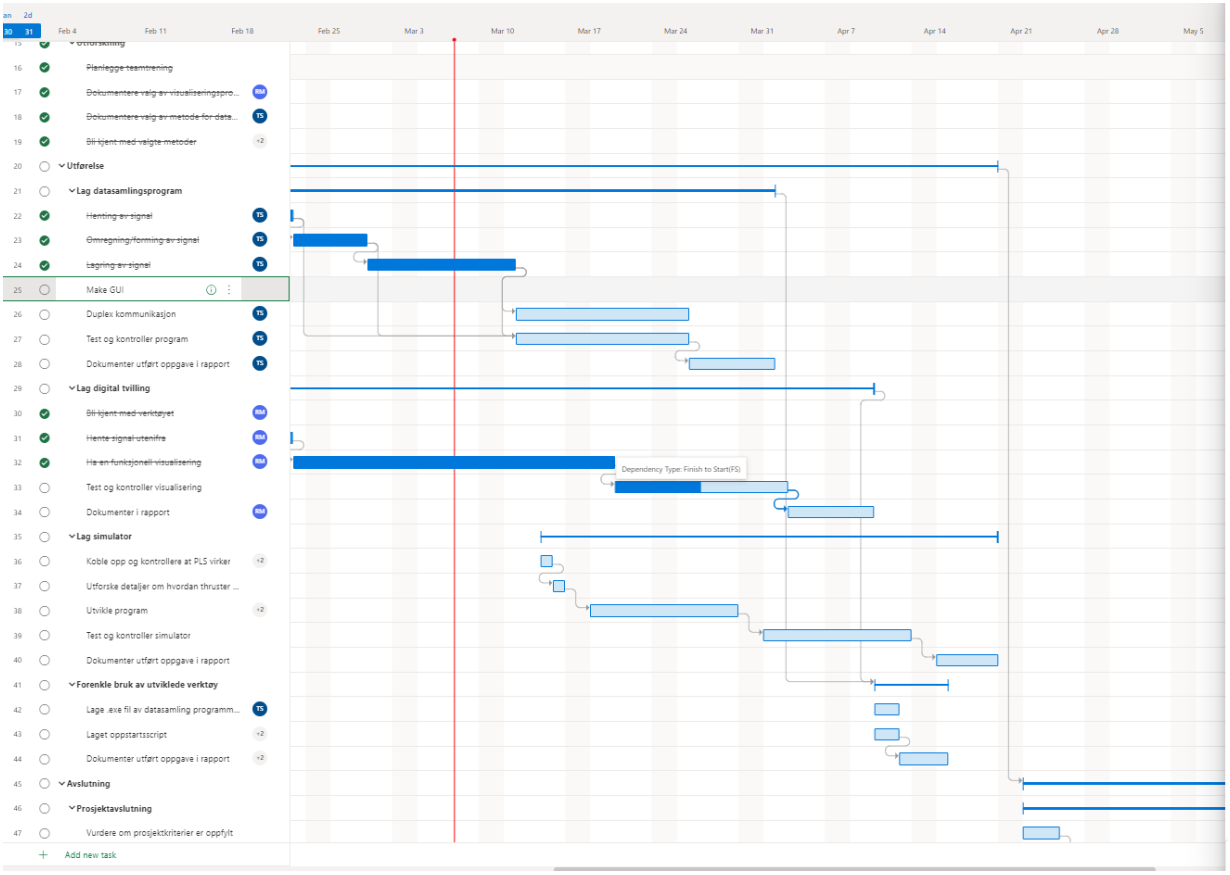
- Now that we have a working VR model, we want a Quest 3 to test AR

Approval/signature group leaders	Signature other group participants
Torstein Skare Robert Moltu	-

GANTT-Diagram showing the period that is relevant for the coming periods:

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 3 av 4
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24



1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 4 av 4
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24

Status Charts from Microsoft Planner

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 1 av 4
Progress report	Period/week(s) 3	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24

Main goal/purpose for these periods work

- For visualization this period aimed to research the possibility for water physics and flow/flush visualization from the thruster in live movement.
- For backend development this aimed to introduce multiple thrusters and a GUI

Planned activities this period

- Look at a solution to get communication from more than one marine controller
- Improve visualization
    - Scaling of object
    - Method for multiple marine controllers
    - Applying thruster model to UT-Boat
- Try to get from KM/NTNU
    - Getting in new models from KM(Azimuth/TCNS)
    - Meta Quest 3
- Improve communication
    - Make every relevant datatype work.
    - Make GUI to start/stop and to set information such as IP-addresses and number of connections.
    - Duplex communication

Actually conducted activites this period

- Received Meta Quest 3 headset and set it up with our current VR project.
- Received a new UT-boat model from Kongsberg Ship Design and removed unnecessary details and objects. Converted to file to a format usable in Unity
- Received UT-boat model from NTNU (Olympic Octopus)
- Discovered a problem with using the Rest-Api. Only sub-objects can be retrieved. This makes this method too slow when collecting many topics.
- Discovered problems in server code. It looks like 320 topics are added, but only 95 are added.
- Made a function that checks which datatype data collected from marine controller is, and use that information to upload the data correctly to our OPC-UA server..

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

- The progression this period has not been as great as last week's period. This is partially due to one of the weeks being set off to an exam in (INGA), preparation for a bachelor's midway presentation, and another being set off to easter holidays.
- A simple GUI has been created, but there were problems running the programs. We believe this is because PyCharm (the IDEwe use) automatically configures some settings that we now must do manually.
- We have not added functionality to add multiple thrusters/marine-controllers because of the problems discovered on the server and GUI. We believe fixing the problems should come first.

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

- The discovery that Rest-Api can only retrieve sub-objects introduced many more objects than we believed initially and increases the time to retrieve data multiple times. This makes us revert to the initial plan of using websockets to collect most of the data. Since our

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

visualization does not require a lot of separate variables RestAPI could be used to collect the data we desire to be the most real-time.
- Extra functionality on the server such as duplex communication and GUI
- For the visualization, the period regards researching and testing several types of assets regarding water physics to try to understand how water works in "games". Additionally importing the UT-boat models to Unity has been tested and applied with live movement.

Main experience from this period

- Water physics turned out to be more complex than we initially thought. This is why in most games water is rendered as a blend of materials and shaders with no physical or interactive logic to it to reduce the computational load. Where effects like bouncy, ripples and splashes are made by using self-made particle systems are used to accomplish this. This is why a particle system was made this week that follows the thrust direction on the propeller.
- The only options available to collect data without extra hardware are Rest-Api and websockets.
- Experiments from last period regarding the number of topics retrieved with websocket seems to be wrong. Because our current program retrieves over 300 values. This is without using batch subscription

Main purpose/focus next period

- Improve server
- Make visualization from VR to AR.
- Improve control over the objects in VR/AR
- Document progress so far in the report.

Planned activities next period

- Make sure that all values we believe to be uploaded to the server, actually are uploaded uploaded to the server
- Connect to multiple marine controllers.
- Make GUI or startup script
- Use received Meta Quest 3 and research possibilities for AR
- Implement laser pointer to grab objects in visualization
- Research implementation of HUD (Head up display) for visualiztion
    - Spawning objects
    - Display values
    - Etc more ideas?

Other

-

Wish/need for counceling

- Removing the creation of a simulator completely from the project. This serves no purpose to further improve the training environment and it already exists. In addition, it seems like we would be short on time. This can be discussed during the meeting.

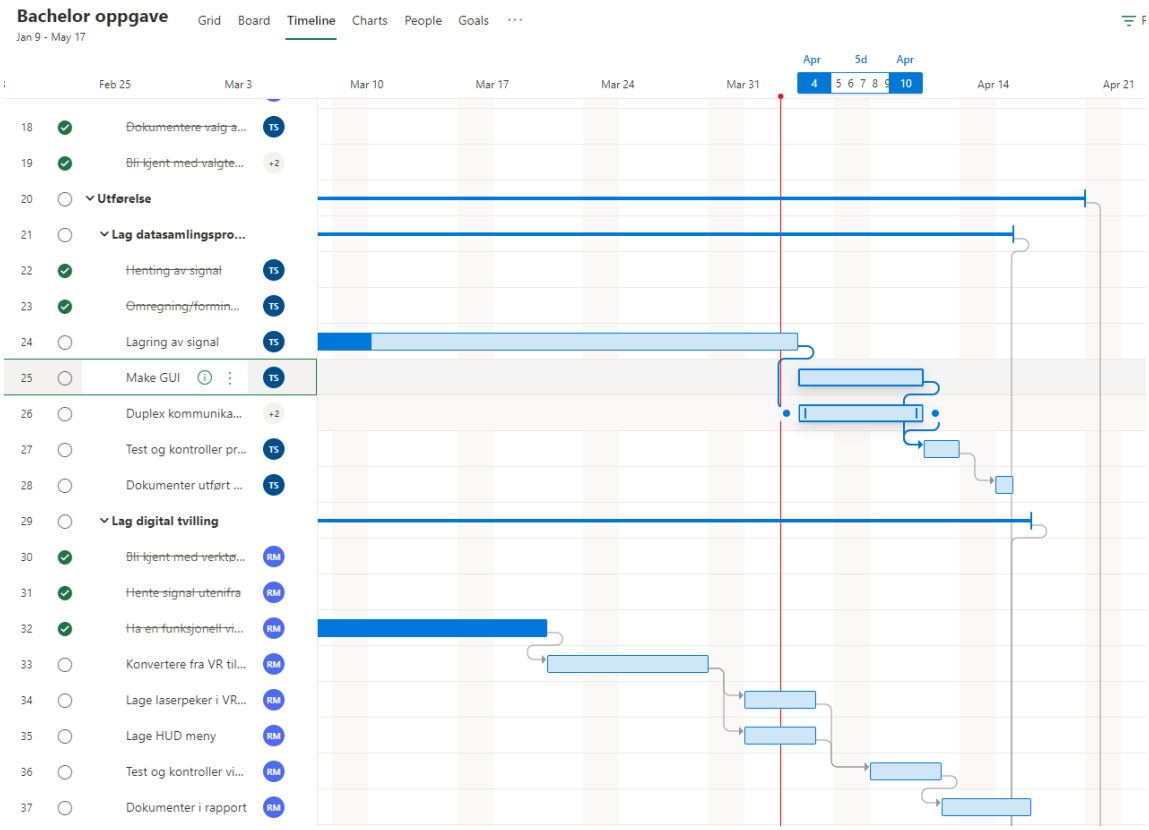
Approval/signature group leaders	Signature other group participants

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 3 av 4
Progress report	Period/week(s) 3	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24

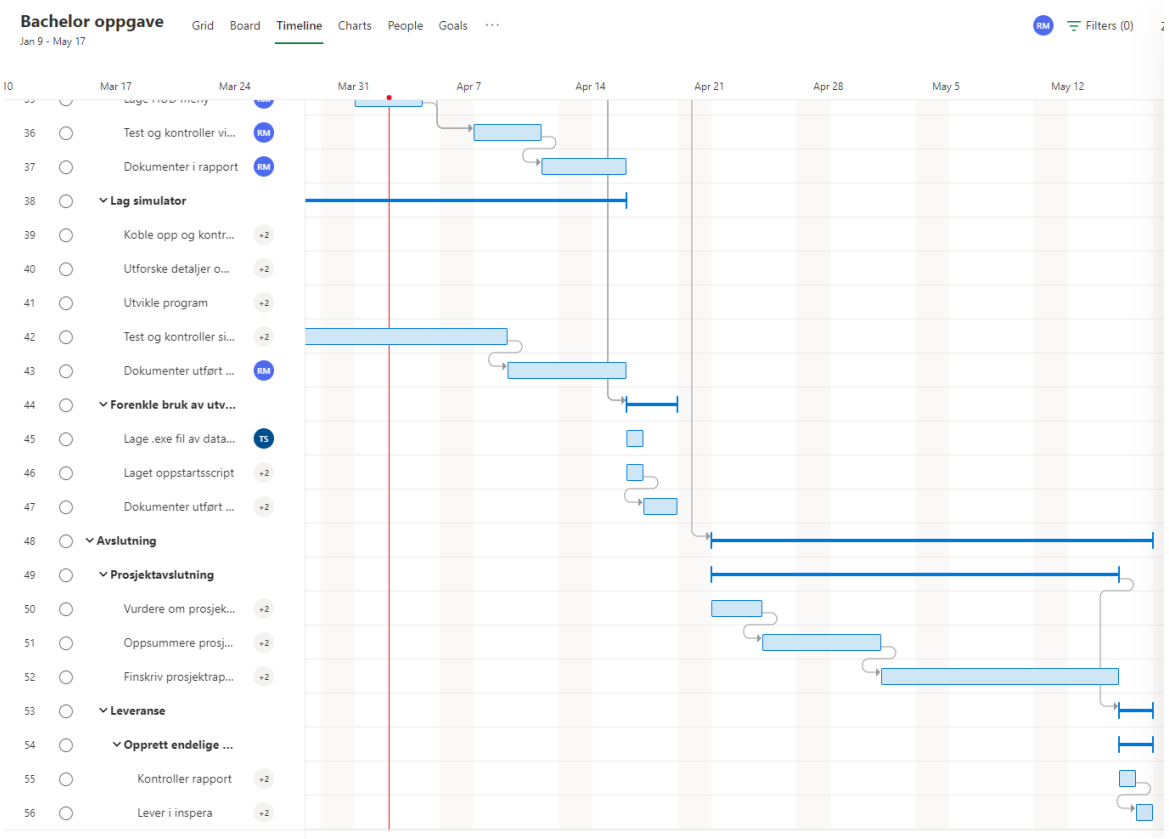
Torstein Skare Robert Moltu	-

GANTT-Diagram showing the period that is relevant for the coming periods:

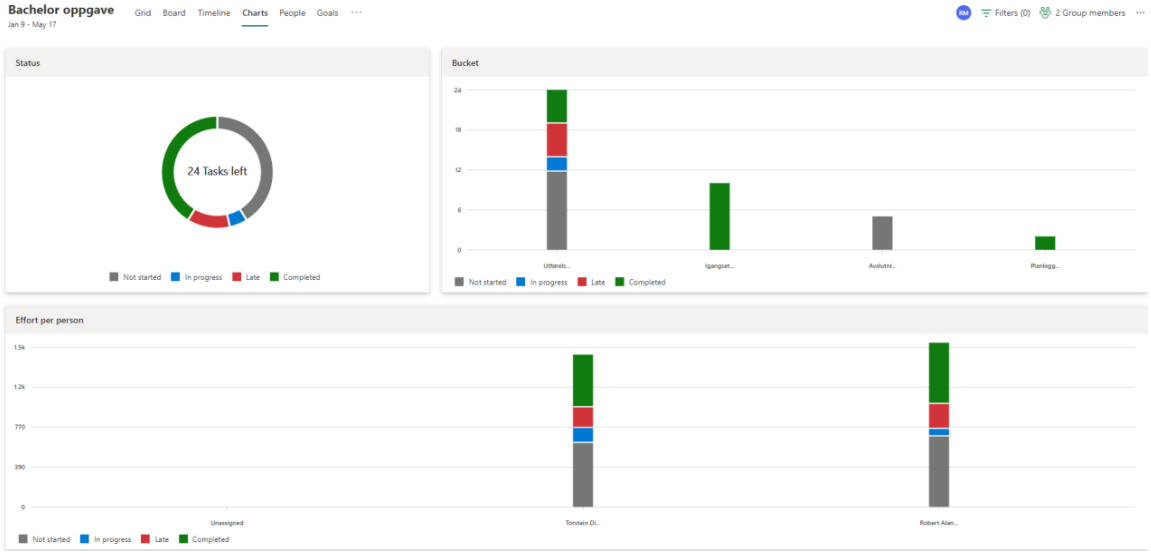


1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 3). 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 4 av 4
Progress report	Period/week(s) 3	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 22.02.24



Status Charts from Microsoft Planner



1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 2. 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 1 av 4
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 4.18.24

Main goal/purpose for these periods work

- Make visualization from VR to AR
- Improve user controlls in VR/AR
- Improve server

Planned activities this period

- Make sure that all values we believe to be uploaded to the server, actually are uploaded to the server
- Connect to multiple marine controllers.
- Make GUI or startup script
- Use received Meta Quest 3 and research possibilities for AR
- Implement laser pointer to grab objects in visualization
- Research implementation of HUD (Head up display) for visualiztion
    - Spawning objects
    - Display values

Actually conducted activites this period

- Successfully adapted the VR environment to AR
- Successfully connected 4 thrusters simultaneously and visualized movement in AR
- Made server robust enough to handle handle wrong connection data.
- Successfully retrieved and sent signals using restapi.
- Worked on modularization of the code.

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

- Making a HUD display for the Visualization
    - This period's focus has been implementing and converting the visualization from VR with the Quest 2 headset to AR with a new Quest 3 headset that supports AR. Additionally, the focus has been to make the AR visualization more user-friendly focusing on improving grab, distance grab and making models interactable. Implementing the movement of multiple thrusters from multiple marine controllers including movement of azimuth thrusters has been a priority. The result of these priorities has been that a HUD has not been successfully implemented and is also discussed as more of a "game development" task rather than a task for our criterias for our education.
- Making GUI for the communication program has been deprioritized because we have made config files that configure all the necessary communication settings and what data is gathered. In addition, a readme file and an executable to explain how to use the program and to run and distribute it more easily.

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

- It was discovered that while we successfully gathered 300 topics from the server, the data was jumbled, meaning that the topic/value pairs were sometimes wrong. We believe this was happening on the KM server.
- Previous plan to get all the relevant signals from CAF was to use websocket and restAPI in combination. While this is possible to do, it would take some time and still be relatively

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

slow. From testing we believe that the KM (Kongsberg Maritime) server has to be upgraded with new functionality to enable realtime data gathering of all the data. As a temporary solution we have limited the program to around 10 topics for visualization.

Main experience from this period

- To have a stable data gathering solution, KM servers must be upgraded.
- To make the user experience in AR better, skills in game development are needed.

Main purpose/focus next period

- Write bachelor report
- Look into the possibilities regarding implementation of a load simulator on the current thruster simulator (pelicase).

Planned activities next period

- Write most of the report
- Check the possibilities of implementing a load simulator

Other

-

Wish/need for counceling

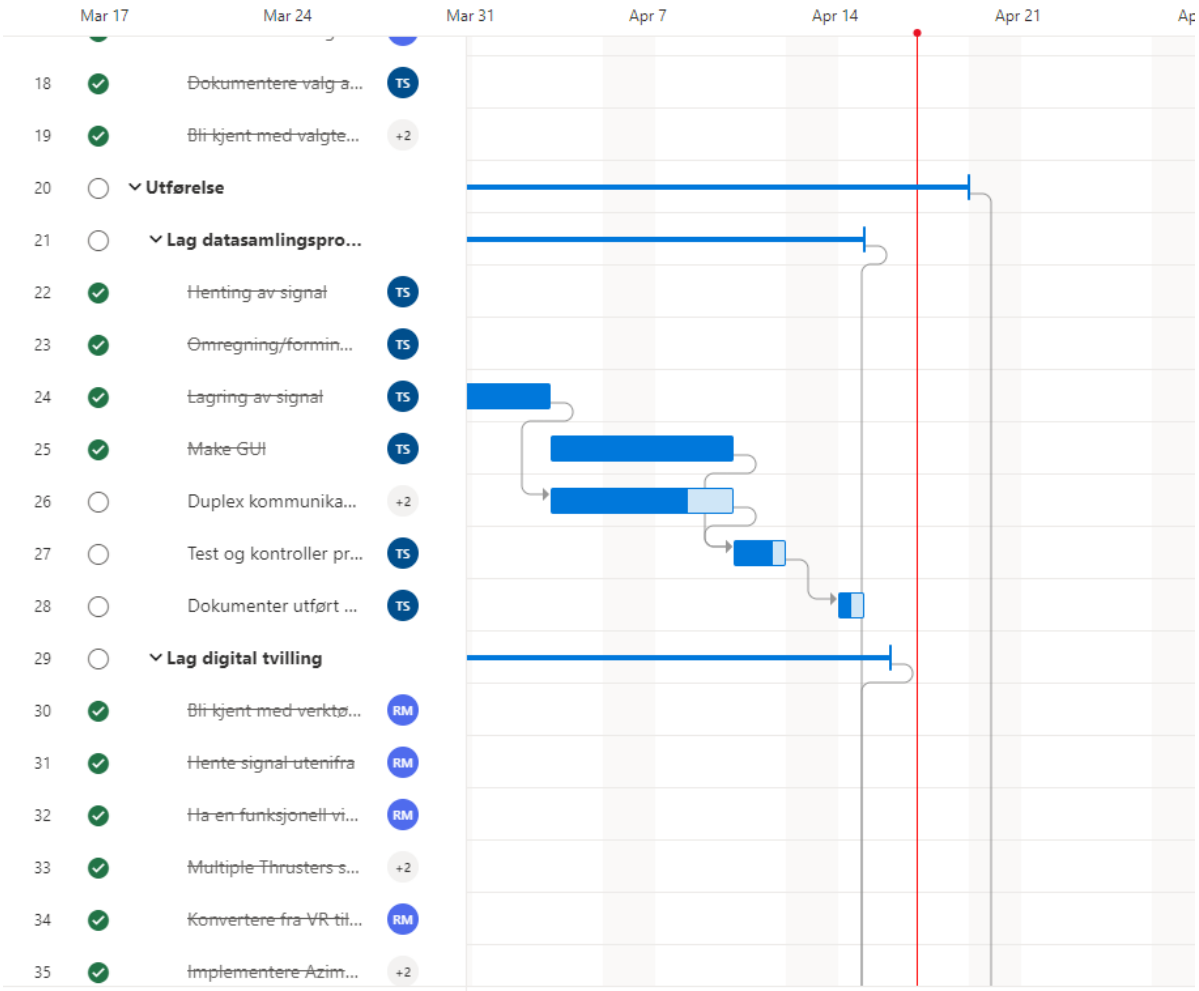
- Looking into implementing a load simulator to expand and lift the project's complexity and get some math, physics e.g. Need some counselling for this part. Do we have a sufficient amount of time? What should we prioritize?

Approval/signature group leaders	Signature other group participants
Torstein Skare Robert Moltu	-

GANTT-Diagram showing the period that is relevant for the coming periods:

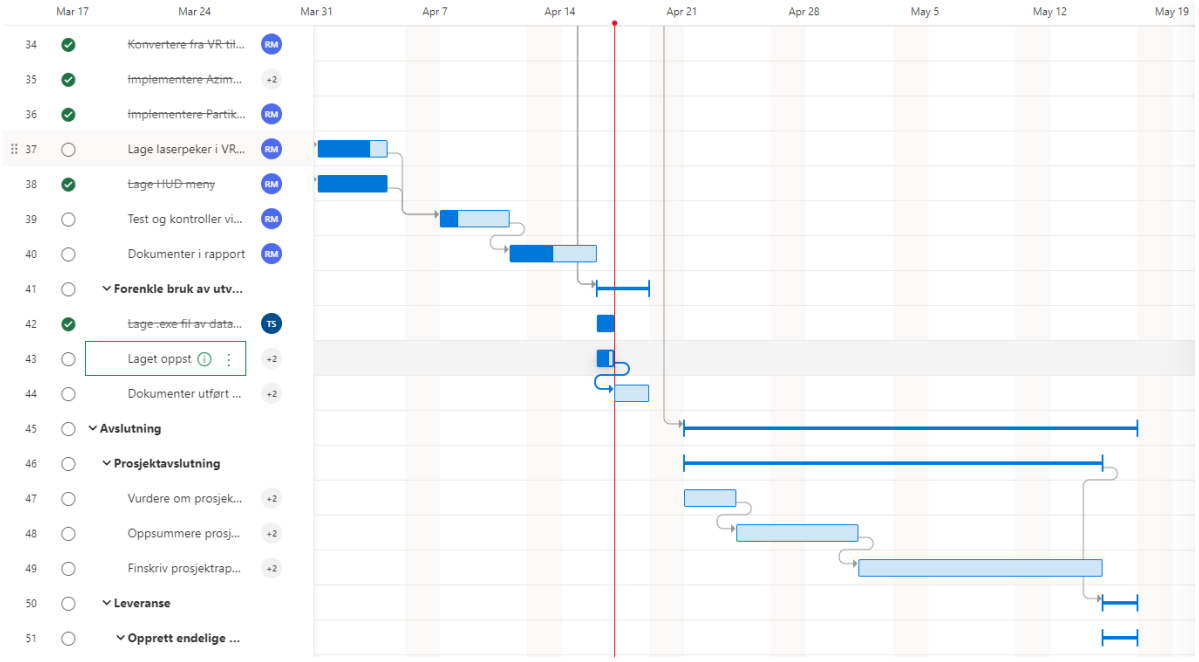
1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 2. 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 3 av 4
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 4.18.24

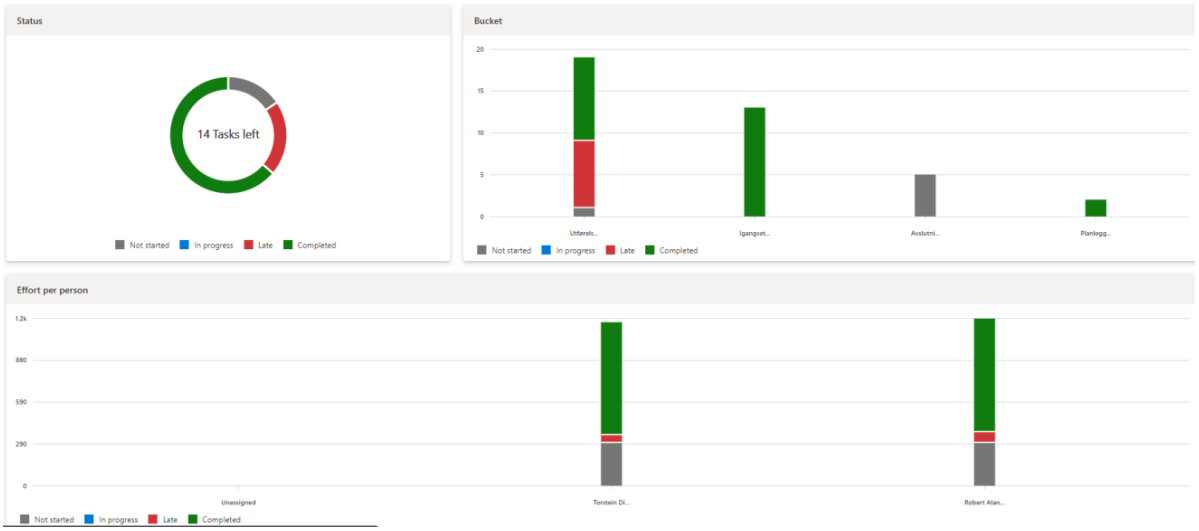


1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 2. 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 4 av 4
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 4.18.24



Status Charts from Microsoft Planner



1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 2. 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 1 av 3
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 4.18.24

Main goal/purpose for these periods work

- Begin writing report
- Write Theory and Methodology part of report

Planned activities this period

- Write most of the report
- Check the possibilities of implementing a load simulator

Actually conducted activites this period

- Theory and Method portion of the report mostly finished.
- About 50% of report has been written.
- Started on writing result parts
- Looked into implementation of load simulation.

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

- The user-friendliness of the visualization could be even better and more logical to grab and hold objects. It has been tried to be improved but has been set on “pause” to give time to focus on writing the thesis.

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

- Have not began to implement a thruster load sim, focus on writing report atleast 80% finished before implementing new parts to the project.

Main experience from this period

- It seems possible to use restapi to both get and reupload the load after calculations, but every attempt so far have crashed CAF.
- When users test the AR part of the project, they are too excited by the AR technology to focus on the thrusters and notice the thruster movements. We have reduced the size of non-moving objects to make the users focus on the thrusters, which seemed to work.
- Noticed that the user-friendliness for the visualization could be improved after demo testing.

Main purpose/focus next period

- Finish the report

Planned activities next period

- Deliver report

Other

-

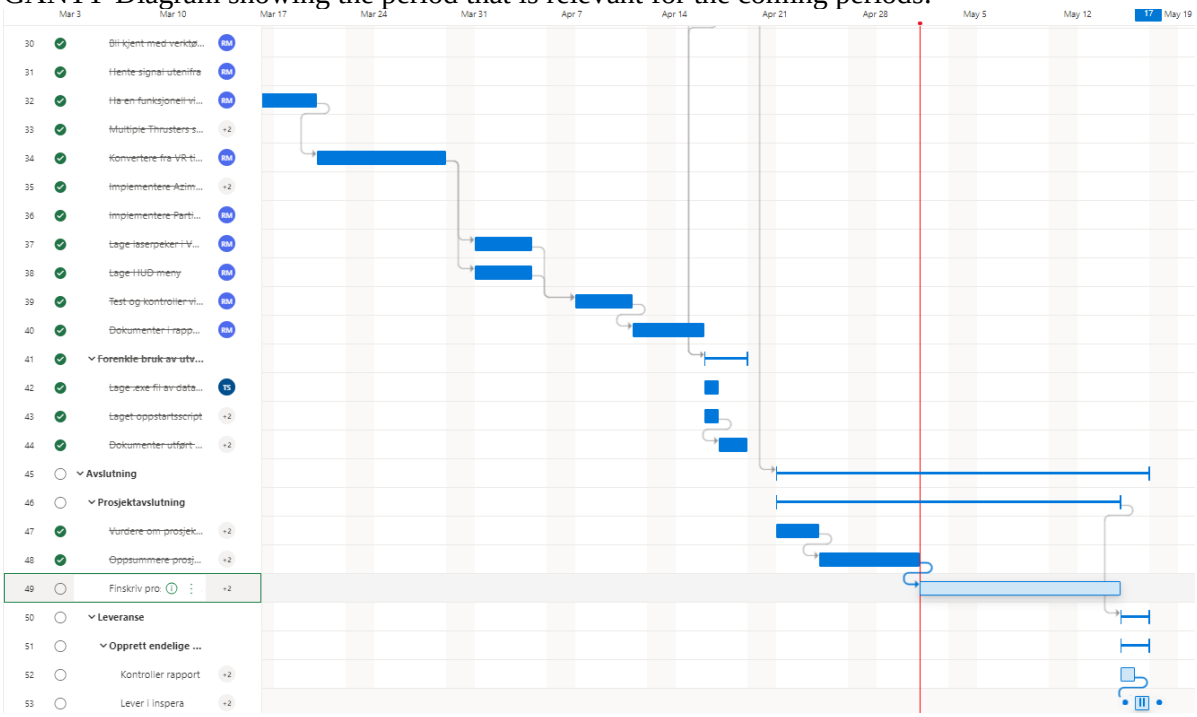
Wish/need for counceling

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 2. 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 2 av 3
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 4.18.24

Når vi skal referere til et spesifikt kapittel i en manual, skal vi referere heile manualen eller kapittelet også korleis gjer vi det?	
Approval/signature group leaders Torstein Skare Robert Moltu	Signature other group participants -

GANTT-Diagram showing the period that is relevant for the coming periods:



1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

AIS 2900 Bacheloroppgave ingeniørfag	Project Next Generation Training Environment for Kongsberg Maritime	Number of meeting this period 2. 1 planned	Firma - Oppdragsgiver NTNU-Ålesund / Kongsberg Maritime	Side 3 av 3
Progress report	Period/week(s) 2	Number of hours this period. (from log) Approx. 75h	Prosjektgruppe (navn) Gruppe 7	Dato 4.18.24

Status Charts from Microsoft Planner

-

1) Noter her kort tilbakemelding om antall møter – fordelt på typer (interne, styringsgruppe, møte med veileder) - i denne rapportperioden

9 Bibliography

[1] Aiohttp. *Welcome to AIOHTTP*. URL: https://docs.aiohttp.org/en/stable/index.html.

[2] Autodesk. *Autodesk 3ds Max: Create massive worlds and high-quality designs*. URL: https://www.autodesk.co.uk/products/3ds-max/overview?term=1-YEAR&tab=subscription.

[3] Blender. *The Software*. URL: https://www.blender.org/about/.

[4] Cambridge Dictionary. *Parsing Definition*. URL: https://dictionary.cambridge.org/dictionary/english/parsing.

[5] opcua-asyncio contributors. *A Minimal OPC-UA Servers*. URL: https://opcua-asyncio.readthedocs.io/en/latest/usage/get-started/minimal-server.html.

[6] GeeksforGeeks. *User Datagram Protocol (UDP)*. URL: https://www.geeksforgeeks.org/user-datagram-protocol-udp/.

[7] Kongsberg Maritime. *Azipull thruster*. URL: https://www.kongsberg.com/maritime/products/propulsors-and-propulsion-systems/thrusters/azipull/.

[8] Kongsberg Maritime. *Standard tunnel thruster*. URL: https://www.kongsberg.com/maritime/products/propulsors-and-propulsion-systems/thrusters/standard-tunnel-thruster2/.

[9] Kongsberg Maritime. *Swing-up Azimuthing Thruster*. URL: https://www.kongsberg.com/maritime/products/propulsors-and-propulsion-systems/thrusters/swing-up-tcnsc-azimuthing-thruster/.

[10] Kongsberg Maritime. *Technologies for sustainable oceans*. URL: https://www.kongsberg.com/maritime/.

[11] Unity Learn. *Introduction to Particle Systems*. URL: https://learn.unity.com/tutorial/introduction-to-particle-systems#.

[12] Meta. *Expand your world with Meta Quest 3*. URL: https://www.meta.com/no/en/quest/quest-3/.

[13] Software Ideas Modeler. *CASE tool for diagrams, software design & analysis*. URL: https://www.softwareideas.net/.

[14] MQTT. *MQTT: The Standard for IoT Messaging*. URL: https://mqtt.org/.

[15] Open Bridge. *About*. URL: https://www.openbridge.no/home/about.

[16] Johannes Valøy Robert A Moltu Torstein D Skare. *Industriprosjekt AIS2221*. URL: https://www.overleaf.com/project/64f858effe92ab46ada54cd5.

[17] Unity. *Unity User Manual 2022.3 (LTS)*. URL: https://docs.unity3d.com/Manual/UnityManual.html.

[18] Unity-Technologies. *HDRP Water Sample Scenes*. URL: https://github.com/Unity-Technologies/WaterScenes.

[19] ValemTutorials. *ValemTutorials Youtube Channel*. URL: https://www.youtube.com/@ValemTutorials.

[20] Wikipedia. *Three.js*. URL: https://en.wikipedia.org/wiki/Three.js.

[21] Wikipedia. *Unity*. URL: https://en.wikipedia.org/wiki/Unity_(game_engine).

[22] Wikipedia. *Unreal Engine*. URL: https://en.wikipedia.org/wiki/Unreal_Engine.