Sondre Mikalsen-Schwenke

Methods for visualizing spatial and temporal space in marine environments using AR and VR

Masteroppgave i Marin Teknikk
Veileder: Asgeir J. Sørensen
Juni 2023

Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for marin teknikk

Sondre Mikalsen-Schwenke

Methods for visualizing spatial and temporal space in marine environments using AR and VR

Masteroppgave i Marin Teknikk
Veileder: Asgeir J. Sørensen
Juni 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for marin teknikk

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MASTER OF TECHNOLOGY THESIS DEFINITION (30 SP)

Name of the candidate:	Sondre Mikalsen-Schwenke
Field of study:	Marine Cybernetics
Thesis title (Norwegian):	Metoder for visualisering av romlig og tidsmessig plass i marine operasjoner ved hjelp av AR og VR.
Thesis title (English):	Methods for visualizing spatial and temporal space in marine environments using AR and VR

Background

The marine domain of travel, transport, and research is showing an ever-increasing level of autonomy and automation. Marine operations using multiple agents of homogenous and heterogenous networks of vehicles in multiple layers on the observational pyramid require a method of observation that allows for a thorough understanding of the current situation as well as a method of adjusting the operational parameters for agents in the field. External operational conditions such as extreme weather or a sudden high-traffic area will change the priorities of autonomous systems. To keep control and a sufficient understanding of any situation, a new method of visualization and control using a combination of Virtual and Augmented Reality, and current platforms can be used to raise the awareness of an operator. A case study of current and new visualization tools for monitoring operations using an autonomous underwater vehicle (AUV) supported by and autonomous surface vehicle (ASV) will be conducted.

Scope of Work

1. Perform a background and literature review to provide information and relevant references on:
   - Coordinated Navigation of AUV and ASV
   - Methods for ocean mapping with combined ASV and AUV platforms
   - Communication challenges for navigation aiding of AUVs
   - Methods of visualization in marine operations

   Write a list of abbreviations and definitions of terms and symbols, relevant to the literature study and project report.
2. Set up and learn about the simulation platform used for testing and implementation of the control system, including the LSTS toolchain (Dune, Neputs, IMC), ROS, and Linux.
3. Designing a software framework based on the LSTS toolchain for use in a new method of visualizing the situation using the Unity game engine and a VR headset
4. Design a method for integrating the seabed in the situation for better visualizing the situation.
5. Describe the experimental set-up for a combined ASV and AUV operation to show the visualization capabilities of the system.
6. Document the results in a scientific report and discuss the setup and further work.

Specifications

<u>The student shall at startup provide a maximum 2-page week plan of work for the entire project period, with main activities and milestones. This should be updated on a monthly basis in agreement with supervisor.</u>

<u>Every weekend throughout the project period, the candidate shall send a status email to the supervisor and co-advisors, providing two brief bulleted lists: 1) work done recent week, and 2) work planned to be done next week.</u>

The scope of work may prove to be larger than initially anticipated. By the approval from the supervisor, described topics may be deleted or reduced in extent without consequences with regard to grading.

The candidate shall present personal contribution to the resolution of problems within the scope of work. Theories and conclusions should be based on mathematical derivations and logic reasoning identifying the steps in the deduction.

The report shall be organized in a logical structure to give a clear exposition of background, problem/research statement, design/method, analysis, and results. The text should be brief and to the point, with a clear language. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts., and it is not expected to be longer than 70 A4-pages, 100 B5-pages, from introduction to conclusion, unless otherwise agreed. It shall be written in English (preferably US) and contain the elements: Title page, project definition, preface (incl. description of help, resources, and internal and external factors that have affected the project process), acknowledgement, abstract, list of symbols and acronyms, table of contents, introduction (project background/motivation, objectives, scope and delimitations, and contributions), technical background and literature review, problem formulation or research question(s), method/design/development, results and analysis, conclusions with recommendations for further work, references, and optional appendices. Figures, tables, and equations shall be numerated. The contribution of the candidate shall be clearly and explicitly described, and material taken from other sources shall be clearly identified. Chapters/sections written together with other students shall be explicitly stated at the start of the chapter/section. Work from other sources shall be properly acknowledged using quotations and a Harvard citation style (e.g. natbib Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct, which is taken very seriously by the university and will result in consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed.

The thesis shall be submitted with an electronic copy to the main supervisor and department according to NTNU administrative procedures. The final revised version of this thesis definition shall be included after the title page. Computer code, pictures, videos, data, etc., shall be available for NTNU, for education and research purposes, and be included electronically with the report.

Start date: 15 January, 2023 **Due date:** 26 June, 2023

Supervisor: Asgeir J. Sørensen

Signatures: [illegible]

Abstract

This Master Thesis covers the development and use of tools and methods for integrating artificial reality with current methods of controlling and observing autonomous vessels such as AUVs, ASVs, and UAVs. Autonomous vessels are capable of completing a wide variety of tasks with minimal to no human interaction. However, at some point the vessels need to be monitored, found, and collected. To anyone who has attempted to find floating objects in the sea or spot a drone flying overhead, this is often a challenging task, even if you know roughly where to look.

Today, autonomous systems use a range of methods and systems for monitoring and control. Most of them are based on 2 dimensional maps on a flat screen, which plots the vessels position and heading, with some also giving a 3D view. Using AR, another method of viewing becomes available to enhance the users capabilities without hindering them from using traditional methods of tracking and control by allowing the user to still see the real world.

A method to bridge vehicle information from the LSTS toolchain to Unity has been deviced and made, allowing for seamless dynamic connection between the two platforms. This method is based on the compatibility between the IMC-messages used by LSTS, and the ROS Robotic Operating System, which makes it possible to translate relatively seamlessly between the two platforms. Once the ROS network mirrors the IMC network, the ROS messages can be transmitted to Unity via TCP, allowing untethered connection between the simulation software and the AR headset.

Using data from GeoNorge, a method has been shown how to extract depth data and converting it into a 3D model useable by Unity. This model can then be placed in full scale underneath the user to give a better understanding of the layout of the seabed near the user or the vehicles by both showing the general structure and depth information based on coloring of the object.

In Unity, a method of giving the user a direct real-time view description of the vessels position, heading, and path by both displaying a model in the approximate actual position of the vessel, as well as giving a 3D model of the area in the users hand to show a more top down view as well as a 3D understanding of the vessels position in relation to the seabed. This method solves the problem of poor visibility while at sea, and allows for more realistic training scenarios while on land.

Sammendrag

Denne masteroppgaven dekker utviklingen og bruk av verktøy og metoder for integrering av kunstig virkelighet (AR) for nåværende metoder for kontroll og observasjon av autonome fartøy som AUV-er, ASV-er og UAV-er. Autonome fartøy er i stand til å utføre en bred variasjon av oppgaver med minimal eller ingen menneskelig interaksjon. Imidlertid må fartøyene på et tidspunkt overvåkes, lokaliseres og hentes. For enhver som har forsøkt å finne flytende objekter i havet eller få øye på en drone som flyr i luften, er dette ofte en utfordrende oppgave, selv om man har en omtrentlig idé om hvor man skal se.

I dag bruker autonome systemer ulike metoder og systemer for overvåking og kontroll. De fleste av dem er basert på todimensjonale kart på en flat skjerm, som viser fartøyets posisjon og retning, og noen gir også en tredimensjonal visning. Ved bruk av AR blir en annen visningsmetode tilgjengelig for å forbedre brukerens evner uten å hindre dem i å bruke tradisjonelle metoder for sporing og kontroll, ved å la brukeren fortsatt se den virkelige verden.

En metode for å koble droneinformasjon fra LSTS-verktøykjeden til Unity er utviklet og implementert, slik at det blir en sømløs og dynamisk tilkobling mellom de to plattformene. Denne metoden er basert på kompatibiliteten mellom IMC-meldingene som brukes av LSTS og ROS (Robotic Operating System), noe som gjør det mulig å oversette relativt sømløst mellom de to plattformene. Når ROS-nettverket speiler IMC-nettverket, kan ROS-meldingene overføres til Unity via TCP, noe som muliggjør trådløs tilkobling mellom simuleringsprogramvaren og AR-hodesettet.

Ved å bruke data fra GeoNorge er det vist en metode for å trekke ut dybdedata og konvertere det til en 3D-modell som kan brukes i Unity. Denne modellen kan deretter plasseres i full skala under brukeren for å gi en bedre forståelse av utformingen av sjøbunnen i nærheten av brukeren eller fartøyene, ved å vise både den generelle strukturen og dybdeinformasjonen basert på fargen til objektet.

I Unity er det utviklet en metode for å gi brukeren en direkte sanntidsvisning av fartøyets posisjon, retning og bane ved å vise en modell omtrentlig i den faktiske posisjonen til fartøyet, samt gi en 3D-modell av området i brukerens hånd for å vise både en mer ovenfra-visning og en tredimensjonal forståelse av fartøyets posisjon i forhold til sjøbunnen. Denne metoden løser problemet med dårlig sikt mens man er til sjøs og åpner for mer realistiske treningscenarier på land.

Preface

This Master thesis has been written over the course of the spring 2023 and concludes the five-year Master of Science in Marine Technology with a specialisation in cybernetics at the Norwegian University of Science and Technology. This thesis has been created independently with the aim of creating a user-friendly, innovative, and original implementation to solve future problems. The original ideas and description of the thesis were discussed and solidified with the help of my main supervisor, Professor Asgeir J. Srensen, at the beginning of the semester.

The original idea for the project came to me in the fall of 2022 as i was assisting NTNU PhD candidate Jens Bremnes in his testing of multiple simultaneous AUV and ASV deployments in the Trondheimsfjord. In the active and crowded fjord, there were multiple occasions when the location of the drones was unclear or mistaken. With the trafficked environment and the varying levels of rough seas, a method of quickly, easily and intuitively locating the drone position in the water could be enormously helpful during operation.

Ideally the system would have been tested with real hardware in the fjord, but this was unfortunately not possible. I hope to be able to test this or a similar system in the future and verify the useability and usefulness of the setup. It has been a thorougly interesting semester making it work.

Trondheim, 2023-06-26

[illegible]

Sondre Mikalsen-Schwenke

Acknowledgment

I would like to take this opportunity to thank my Supervisor Asgeir J. Sørensen for his aid in deciding what the focus of this project should be. During the start phase of the project, he showed positivity, creativity, and eagerness to help that helped to inspire and solidify the project into an interesting opportunity to apply my skills and learn more in a great way. The enthusiasm he showed by connecting me to the right people and pushing me to gain more experience in the field has been extremely helpful in starting the project.

Secondly, i would like to show my thanks to the man who showed me the ropes and gave me first-hand experience in a use case of the project, my cosupervisor Ph.D. candidate Jens E. Bremnes. His positive attitude and eagerness to share experiences and opinions were of great help in understanding the setup I would have to create for the project and its individual components. The opportunity to join Jens in his multiple agents test in the field was a great experience and helped solidify my understanding of why the project would be useful now and more so in the future.

I also would like to thank my friends in the university office as well as my partner at home for allowing me to brainstorm ideas and solutions, as well as being helpful, supporting and keeping spirits high in the last year of studies.

Sondre Mikalsen-Schwenke
Trondheim, Norway
June, 2023

Contents

List of Figures

Acronyms

UUV	Unmanned Underwater Vehicle
ASV	Autonomous Surface Vehicle
AUV	Autonomous Underwater Vehicle
LAUV	Light Autonomous Underwater Vehicle
DR	Dead Reckoning
DOF	Degrees-of-Freedom
IMU	Internal Measuring Unit
DVL	Doppler Velocity Log
USBL	Ultra Short Base Line
HRI	Human Robot Interface
SA	Situational Awareness
AR	Augmented Reality
VR	Virtual Reality
XR	Mixed Reality, covers AR and VR
MQ2	Meta Quest 2
HMD	Head Mounted Display
FOV	Field of view
LSTS	Underwater Systems and Technology Laboratory
Dune	Uniform Navigation Environment

IMC	Inter-Module Communication
SNAME	Society of Naval Architects and Marine Engineers.
ROS	Robotic Operating System
TCP	Transmission Control Protocol
RTC	ROS-TCP connector
UDP	User datagram protocol
SCL	Shore Control Lab
AUR-Lab	Applied Underwater Robotics Laboratory

Chapter 1

Introduction

1.1 Background and motivation

The marine domain of travel, transport, and research is showing an ever-increasing level of autonomy and automation. Marine operations using multiple agents of homogenous and heterogenous networks of vehicles in multiple layers on the observational pyramid require a method of observation that allows for a thorough understanding of the current situation, as well as a method of adjusting the operational parameters for agents in the field. External operational conditions, such as extreme weather or a sudden high-traffic area, will change the priorities of autonomous systems. To keep control and a sufficient understanding of any situation, a new method of visualization and control using a combination of Virtual Reality and current platforms can be used to raise the awareness of an operator. Using the existing backbone structure found in autonomous systems today, such as the LSTS toolchain, and ROS, the new visualization tool will be constructed to allow for a more efficient and reliable method of knowing the positions and actions of the autonomous agents surrounding the operator. A case study of current and new visualization tools for monitoring operations using an autonomous underwater vehicle (AUV) supported by an autonomous surface vehicle (ASV) will be conducted.

1.2 Previous Work

This master project is a continuation of the project work done in the autumn of 2022.Mikalsen-Schwenke 2022 This thesis covers a lot of the same subject with several additions and changes in method and will therefore contain sections containing similar information as the original project report.

1.3 Project Objectives

The main objective of this project is to develop a method for translating the current setup for control and monitoring of many autonomous systems into an XR platform.

1. Set up an environment to test the AR system using a realistic positional system. (LSTS Toolchain)
2. Create a pipeline for transferring the positional data from the LSTS toolchain to Unity
3. Make a Unity program that can interpret the data and display the positions of the agents run using the LSTS toolchain.
4. Integrate depth data to make the seabed visible to the user in both AR and VR mode.
5. Implement other methods of visualising the data coming from the LSTS toolchain using AR/VR.
6. Demonstrate the system using two simulated cases, as well as a video showing the capabilities of the system.
7. Propose improvements to the setup and uncover useful data on the requirements of such a solution using AR/VR.

At the end of the project, a platform allowing dynamic connection between the LSTS toolchain and Unity shall be built. The setup will be a proof of concept and a platform that can be used for further development and expansion in terms of scope in the future. stability, usability, and functionality of the system.

1.4 Contributions

To the authors' knowledge, this project is one of very few instances where AR has been actually applied and tested in combination with the LSTS toolchain. The setup and methods used in the project contribute to a clearer understanding of how to setup and build such a connection between the LSTS Toolchain, ROS, and an AR view that allows further work and improvements to be done in the future. A method for generating 3D objects of bathymetric charts of the seabed that are useable by Unity, as well as a method to translate coordinates between reference frames for the correct placing on the seabed. The thesis also shows methods of conveying information in Unity using shaders to display depth, and the use of lights and multiple objects to catch the users' attention.

1.5 Thesis Outline

The thesis is setup in the following configuration.

- Chapter 1. Introduction
- Chapter 2. Background and Preliminaries: Covers some of the relevant topics and theory discussed in this thesis. This includes information on autonomy, AR and VR, and reference frames used in the thesis.
- Chapter 3. AR System Components and Setup: Describes the components and setup for the individual components of the project. This includes each component of the LSTS toolchain, ROS libraries, and the unity software running on the MQ2.
- Chapter 4. Results: Displays the systems uses by the means of two simulation cases, as well as discussing a demonstration video.
- Chapter 5. Discussion: This chapter describes the capabilities of the software and discusses the results as well as potential points of improvement.
- Chapter 6. Conclusions and further work: The final chapter summarizing the results and discusses future work that can be done to continue the work from the project.
- Bibliography
- Appendix A: Containing the link to the video demonstration as well.
- Appendix B: Containing link to the GitHub repository with the source code and relevant code snippets used in the project.

Chapter 2

Literature Review and Preliminaries

Today, several systems are used for the autonomous control of marine targets. LSTS Toolchain is a collection of tools for use with the monitoring and control of targets in the air, on the surface and below the surface of the ocean. All of these systems have varying levels of autonomy and monitoring. This chapter covers a overview of some of the relevant concepts and terminology used in this thesis.

2.1 Autonomus Systems Today

2.1.1 Unmanned Underwater Vehicles

Unmanned underwater vehicles (UUVs) are a rapidly expanding area of research with numerous applications. They are used for a variety of tasks, including bathymetric surveys, pipeline inspections, cable maintenance, marine archaeology, and marine biology studies. There are two main classifications of UUVs, remotely operated vehicle (ROV) and autonomous underwater vehicles (AUV). Unlike AUVs, ROVs are tethered and usually controlled by an operator located onshore or on a surface vessel. These vehicles are commonly equipped with manipulator arms that enable them to perform interactive tasks such as interacting with valves, aiding in underwater constructions and inspections.

AUV

Autonomous underwater vehicles (AUVs) represent a category of unmanned underwater vehicles (UUVs) that operate independently, without the need for human intervention, making them particularly suitable for underwater surveys. A typical AUV design includes a stern propeller for controlling forward speed and two fins for managing pitch and heading, ensuring exceptional maneuverability and the ability to cover extensive distances. See Figure: 2.3. AUVs often have a hydrodynamical shape, allowing them to be flexible, reach high speeds, and operate in complex environments. They carry their power supply onboard and are often underactuated, which means that they are not controllable at all degrees of freedom. Although this limits the duration and geographical extension of operations and excludes tasks that require precise control of all degrees of freedom.

AUVs can be equipped with a variety of sensors such as GNSS, Ultra Short Baseline (USBL), Doppler Velocity Log (DVL), IMUs, and water quality sensors, making them suitable for environmental monitoring, hydrography, and search and recovery (Kongsberg 2020). Current applications of AUV include the acquisition of high-resolution maps of the deep sea floor, the temporal and spatial presence of the ocean to survey oceanographic states such as salinity and temperature(Bellingham 2009). There are also several military applications, such as mine detection and clearance, and long-range reconnaissance (Bae and Hong 2023).

Some of the disadvantages that AUVs have are a lack of consistent communication; as nearly all electro-magnetic radiation is blocked by the watercolumn. The only method of communication is via acoustic methods, often combined with the USBL system. However, this method, compared to radio and satellites, lacks data bandwidth and makes navigation less accurate than systems with GNSS capabilities. This also leads to an increased probability of loss of the AUV, as a mistake in navigational data or a loss of power in the system can over time lead to great discrepancies in the estimated position of the vessel and the actual position.

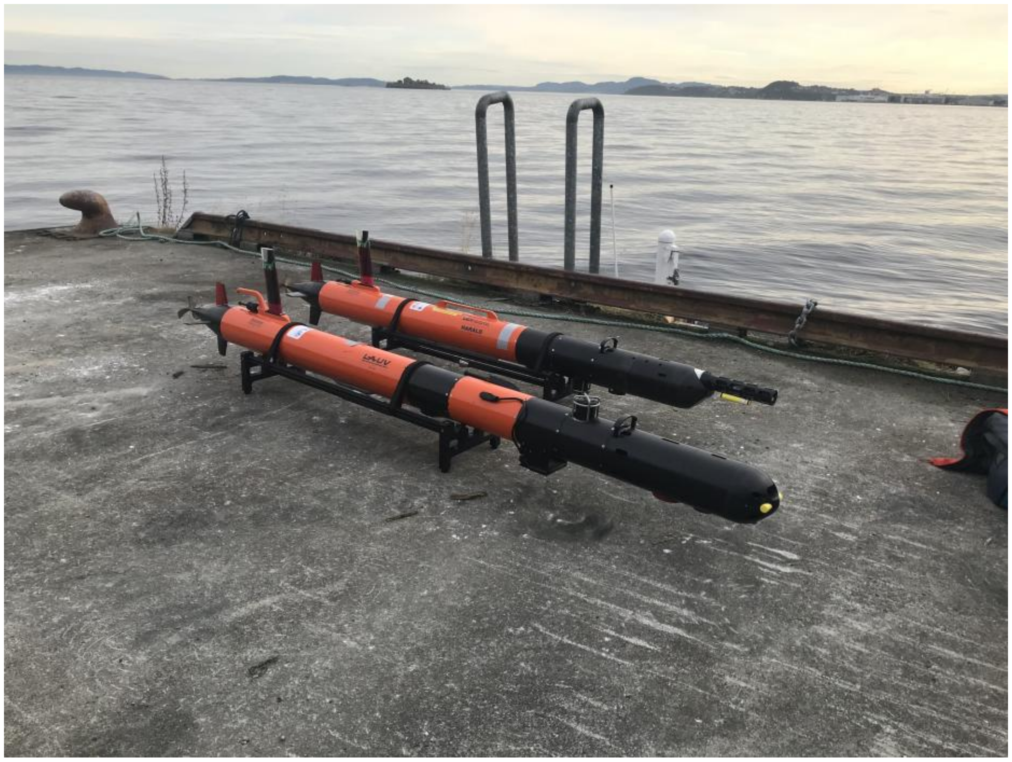


Figure 2.1: Two of AUR-Labs LAUV vehicles.

Autonomous Surface Vehicle

ASVs are a category of unmanned surface vehicles that can operate independently, without human control or external methods of power delivery or production. Because the ASVs operate on the surface they are able to receive and send constant data to and from the user using either radio transmission, 4G networking, or satellites, and are able to maintain a constant accurate positional update via GNSS. This allows ASVs to operate with sub-metre accuracy in their positioning no matter the length of the mission.

ASVs are considerably useful for a wide range of tasks such as environmental monitoring, wild life tracking, large scale seabed and riverbed bathymetry, and search and rescue missions.

2.1.2 Unmanned Areal Vehicle

Unmanned Aerial Vehicles (UAVs), commonly known as drones, are autonomous or remote controlled aircraft that operate without an onboard human pilot. These aircrafts are often equipped with various sensors, cameras, and navigation systems that allow them to fly and perform tasks without direct human intervention. UAVs come in different sizes, ranging from small hand-held drones to larger fixed-wing aircraft, and can be powered by electric batteries, combustion engines, or even solar energy. UAVs are widely used in aerial photography and videography, surveying and mapping, agriculture, infrastructure inspection, search and rescue operations, and even military applications. UAVs offer significant benefits, such as cost effectiveness, increased safety, and the ability to access difficult-to-reach or hazardous areas. They can be programmed to follow predefined flight paths, perform complex manoeuvres, and capture high-resolution imagery or sensor data. However, UAVs also raise concerns related to privacy, security, and airspace regulations, as their widespread use requires careful consideration of ethical and legal implications. The drawbacks of UAVs are limited payload capacity, limited operational length, vulnerability to wind and weather, as well as risk of loss of vehicle.

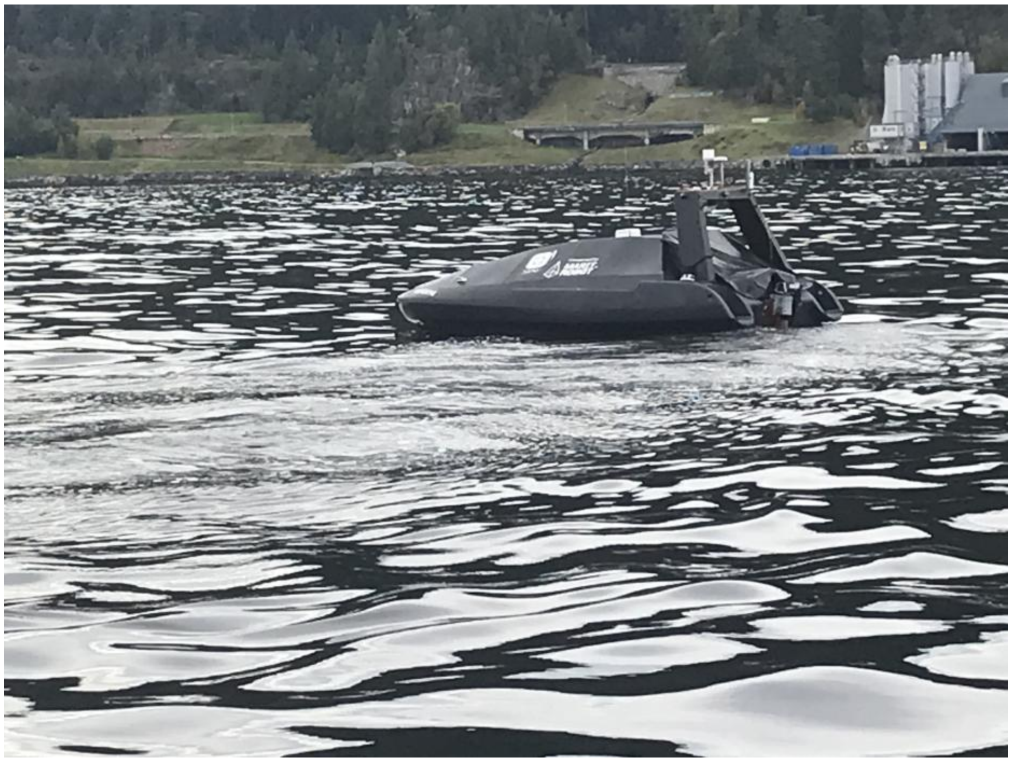


Figure 2.2: AUR-labs ASV, Grethe.

Figure 2.3: UAV fra NTNU AUV Lab. Foto: Asgeir J.Sørensen

2.2 The Future of AUVs and ASVs

An idea is to deploy a fleet of autonomous underwater vehicles (AUVs) and autonomous surface vehicles (ASVs) in the ocean to perform various tasks collectively. They can collaborate on tasks such as underwater surveillance, mapping the ocean floor, and identifying targets. If one vehicle discovers a target, this information can be shared with other AUVs and ASVs in the vicinity. Communication can be achieved using acoustic or radio signals. The AUVs and ASVs can be seen as a decentralised system with mobile nodes operating in the ocean. In the event of a vehicle malfunction or loss, the remaining vehicles retain the majority of the information and continue the mission.

Exciting AUV and ASV Applications in the Ocean

- Underwater surveillance of critical areas and marine assets.
- Search and rescue missions for lost objects or individuals in the ocean.

- Environmental monitoring and research, including studying marine life, water quality, and ecosystem health.
- Exploration and mapping of the ocean floor and underwater geological features.
- Disaster response and recovery operations in the aftermath of marine incidents or natural disasters.
- Military applications, such as underwater reconnaissance and mine detection.

The military has a strong interest in using AUVs and ASVs for various maritime operations. These vehicles can replace the need for manned patrols along coastlines and monitor restricted areas. With advanced algorithms and sensors, they can navigate complex underwater environments, detect underwater threats, and gather valuable intelligence. AUVs and ASVs can also be used in coastal management to monitor pollution levels, detect oil spills, and protect marine ecosystems. In addition, they can contribute to scientific research by collecting data on ocean currents, temperature, salinity, and other oceanographic parameters. Using specialised sensors, AUVs and ASVs can identify underwater archaeological sites, locate underwater resources, and aid in underwater construction projects.

The future of AUVs and ASVs in the ocean holds great potential for a wide range of applications, from maritime security to environmental protection and scientific exploration.

2.3 AR in navigation today

Today, AR has not yet been applied as a normal tool in the maritime sector, with a few exceptions starting to emerge. With the launch of multiple incresingly complex headsets from Meta, Microsofts several AR headsets such as the Hololens as well as the soon to be released Apple Vision Pro, there is reason to believe that in the coming years an increase in interest and demand will happen in the maritime sector.

2.3.1 Uses of AR in maritime sector.

Raymarine ClearCruise AR.

Raymarine has been proactive and launched its first AR solution for maritime use in 2019 with the name ClearCruise (Raymarine 2020). This system uses HD cameras and a processing unit to display information on a 2D screen for the user. The system provides a augmented reality view on the screen, allowing users to see critical navigation objects, surrounding automatic identification system (AIS) traffic, navigation markers, and waypoints in sync with real-world visuals.

The system does, however, not support any sort of AR headset or glasses, and little information is known about how the system works specifically.

Equinors use of Hololens

Although not directly used in navigation, Equinor has since its digitalisation been integrating AR solutions into its development process (Equinor 2021). Using the Hololens and Unity game engine, the workers are able to visualise digital models overlaid onto the physical environment, enabling them to easily navigate complex structures and identify specific components. According to Equinor, this greatly increases the efficiency in installing new systems on oil rigs, and saves cost by allowing the integrators to catch mistakes before construction physically begins.

2.3.2 Augmented Reality and its use cases

The use of AR in maritime navigation addresses issues such as limited visibility, lack of landmarks, and increasing maritime traffic. Using AR technology, operators can visualise their route on the water, identify obstacles that are difficult to perceive, and display real-time information on sea conditions directly in their field of view.

AR systems in maritime navigation are designed to reduce the cognitive load on human operators and increase safety. With the increasing amount of information available from sensors and instrumentation on ships, even simple tasks can become complex and distracting. AR helps by providing visual overlays of relevant information, allowing sailors to better perceive and understand their surroundings. For example, AR can paint navigation information, such as routes and obstacles, directly onto the sea surface, taking advantage of the higher vantage point on ships. This integration of visual information into the maritime environment through AR can significantly improve situational awareness and decision making, ultimately improving safety and reducing the stress experienced by sailors.

The following subsections cover some of the most widely used AR / VR headsets today and cover some differences, strengths, and weaknesses.

Meta Quest 2

For this thesis, the headset used for AR/VR is the Meta Quest 2 (MQ2), formerly known as Oculus Quest 2 Fig: (2.4). The headset is designed to run simulations and games natively in the headset, without the need for an external computer to run code and generate graphics. The headset uses a combination of four cameras as well as internal measuring units to track the user's position in space. The headset can use camera feeds to recreate the view in front of the headset to give the user a view of the world through the headset. However, the view is quite grainy black and white because the cameras only film in the infrared spectrum, and were initially designed only for tracking the user in space. However, for the purposes of this thesis, the view is good enough to give a valid platform for discussing the validity of a future set-up.

Figure 2.4: Meta Quest 2. AR/VR headset

Meta Quest Pro

Another headset developed by Meta is the Meta Quest Pro headset. This headset has the benefit of using the same framework as the widely used Quest 2, but comes with a number of additional features. Like the MQ2, the Quest Pro uses cameras to both track the world around the user and to show the user a realistic view which is displayed on the screens in front of the users eyes. This headset however has dedicated AR cameras which gives a full-color high resolution videofeed to view the world in front of the user making it easier to complete both complex and mundane tasks while wearing the headset.

Microsoft Hololens 2

Microsoft was early in their development of AR solutions when they unveiled the original Hololens in 2016. Hololens differs from Metas Quest platform in that the users view of the outside world is never blocked. This allows the headset to project 3D objects and "holograms" in front of the user without the loss of sight which makes operating outside equipment and reading screens while wearing the headset not a problem. The headset is compatible with Unity game development, and it is therefore relatively easy to convert the software developed in this thesis to run on the Hololens. However, the headset does come with a pricetag of $3500 and is therefore not available for use in this project.

2.4 Reference frames

Reference frames are a method of establishing the relationship between two bodies and are a crucial tool in most navigational methods. For normal navigation and control, atleast two reference frames are necessary for positional coordination. For this thesis, the reference frames use the same notation as Dune, which is the SNAME notation for NED and body frames (Pérez and Blanke 2023). The frames used in this thesis are shown in Fig. 2.5.

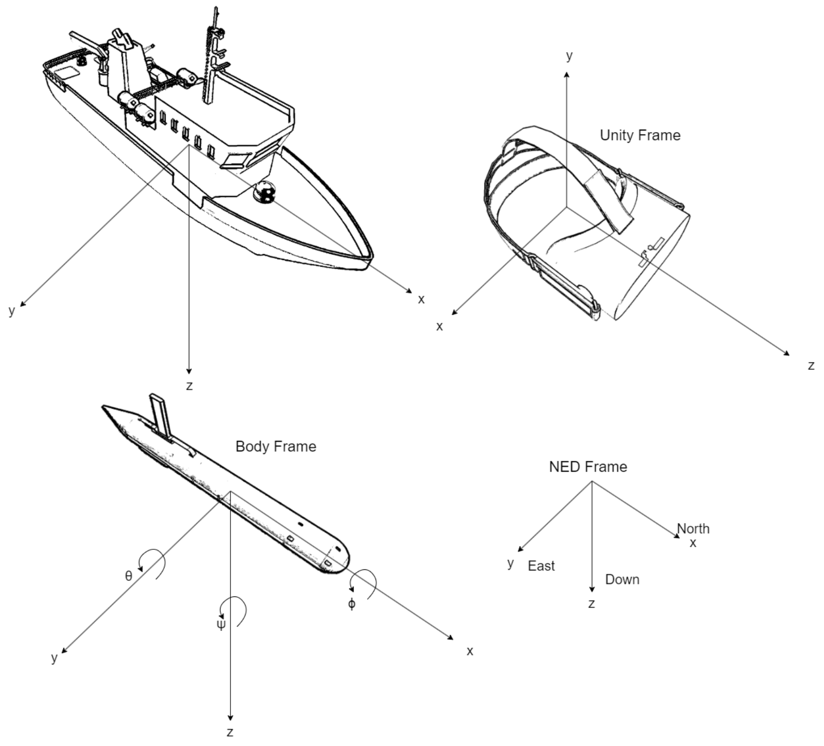


Figure 2.5: Representation of the reference frames used in this thesis

2.4.1 NED Frame

The Earth-fixed NED(North-East-Down) frame is used for the positioning of the drones in the EstimatedState messages and is centered at the starting point of the drones. The x, y, and z axes always point in the North, East, and Downward directions, respectively. The frame can be represented as a 6-DOF (Degrees-of-Freedom). system that includes roll, pitch, and yaw rotations, using the representation shown in Figure 2.6.

$$\eta = [p \quad \Theta]^T = [N \quad E \quad D \quad \phi \quad \theta \quad \psi]^T \in \mathbb{R}^6 \tag{2.1}$$

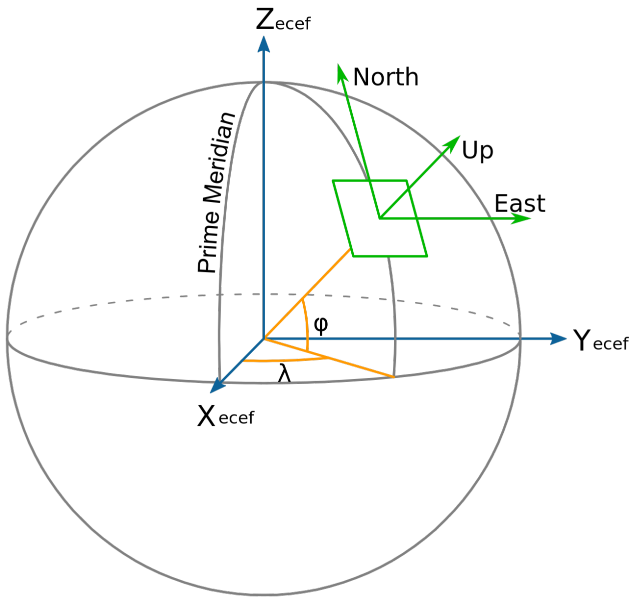


Figure 2.6: Representation of NED frame.(Wikipedia 2023)

2.4.2 Body frame

The body frame is a frame that follows each individual vessel and is centred in the centre of gravity. The axes of the frames follow the vessel rotation, with the x-axis of the frame pointing towards the front of the vessel, the y-axis points towards the starboard position, and the z-axis points downwards. Additionally, the velocity values for the vessel are given in the body frame as the velocity v and the angular velocity ω (Pérez and Blanke 2023).

$$v = [p \quad \omega]^T = [u \quad v \quad w \quad p \quad q \quad r]^T \in \mathbb{R}^6 \tag{2.2}$$

2.4.3 UnityFrame

The Unity frame refers to the coordinate system within the Unity programme. At launch, the programme sets the default origin of the frame to be the location of the VR headset. This can be reset at any time by using a "recenter/"adjust view" functionality allowing frame to be rotated in reference to either north or the body frame of a ship depending on the situation.

2.4.4 WGS 84, World Geodetic System

To coordinate the positioning of the user, drones, and seabed, the use of an Earth-fixed coordinate system is required. For this thesis, the coordinate system used is the WGS 84 standard. The WGS 84 standard is the current iteration used in GPS navigation and is used both by LSTS Toolchain and Google when referring to latitude and longitude. The standard has its origin in Earth's centre of mass and uses two angles and height to define positions on the Earth's surface with the sea.

Latitude ϕ States the degrees from the equator. It is equal to 90° at the north pole, −90° at the south pole, and it is 0° at the equator.

Longitude λ States the degrees of rotation from the IERS reference meridian, placed 102 metres east of the historical Greenwich meridian.

2.4.5 translation

Translation between the Earth-fixed WGS 84 frame and the Earth-fixed NED frame is done using the AlvinXY algorithm as outlined in (Murphy and Singh 2010). The algorithm, which is also used by som ROS packages, defines some latitude (ϕ) and longitude (λ) orign to be equivalent to some x and y coordinates. For the 3D model of the seabed used, this was set as shown in 2.3. These values need to be synced for every 3D seabed or land model used in such a setup.

$$\begin{aligned} \phi &= 63.44254823 \\ \lambda &= 10.35537962 \\ x &= 2000 \\ y &= -4000 \end{aligned} \tag{2.3}$$

using this as a reference, the translation was implemented as such.

```

public class LatLongTranslater
{

    public static double[] zeroLatLong = { 63.441924, 10.406475 };
    public static double[] zeroUnityZX = { 2000, -4000 };

    private static double zLatRad =(zeroLatLong[0]) * Math.PI / 180;
    private static double zLongRad = (zeroLatLong[1]) * Math.PI / 180;
    private static double mDegLon = (111415.13 * Math.Cos(zLatRad))
    - (94.55 * Math.Cos(3.0 * zLatRad)) - (0.12 * Math.Cos(5 * zLatRad));
```

```
        private static double mDegLat = 111132.09 - 566.05 * Math.Cos(2 * zLatRad)
        + 1.20 * Math.Cos(4.0 * zLatRad) - 0.002 * Math.Cos(6.0 * zLatRad);


        //This method takes a LatLon double and translates it into this zone's game
    ↪   world coordinates
        public Vector3 GetUnityPosition(double[] latLonPosition)
        {
            double z = zeroUnityZX[0]+((latLonPosition[1] - zeroLatLong[1]) * mDegLon);
            double x = zeroUnityZX[1]+((latLonPosition[0] - zeroLatLong[0]) * mDegLat);
            Vector3 unityCoord = new Vector3((float)z, 0f, (float)x);
            return unityCoord;
        }

        // This method takes a vector of unity positions and converts it into latitude
    ↪   and longitude coordinates
        public double[] GetLatLonPosition(Vector3 unityPos)
        {
            double lon = unityPos[1] / mDegLon + zeroLatLong[1];
            double lat = unityPos[0] / mDegLat + zeroLatLong[0];
            double[] endLatLong = { lat, lon };
            return endLatLong;
        }
    }
```

Chapter 3

AR System Components and Setup

This chapter provides an in-depth exploration of the simulation platform components and the augmented reality setup. It offers a comprehensive description of each component used in the setup, including its requirements and its specific role within the system. The setup is made to be used either on board of a stationary vessel or in a on-shore control lab setting. Figure 3.1 shows the overall setup of the system when in use. During testing, the LSTS toolchain components as well as the ROS nodes are running on a dedicated laptop running Linux Ubuntu, while the unity instance is running natively standalone on a Meta Quest 2 headset.

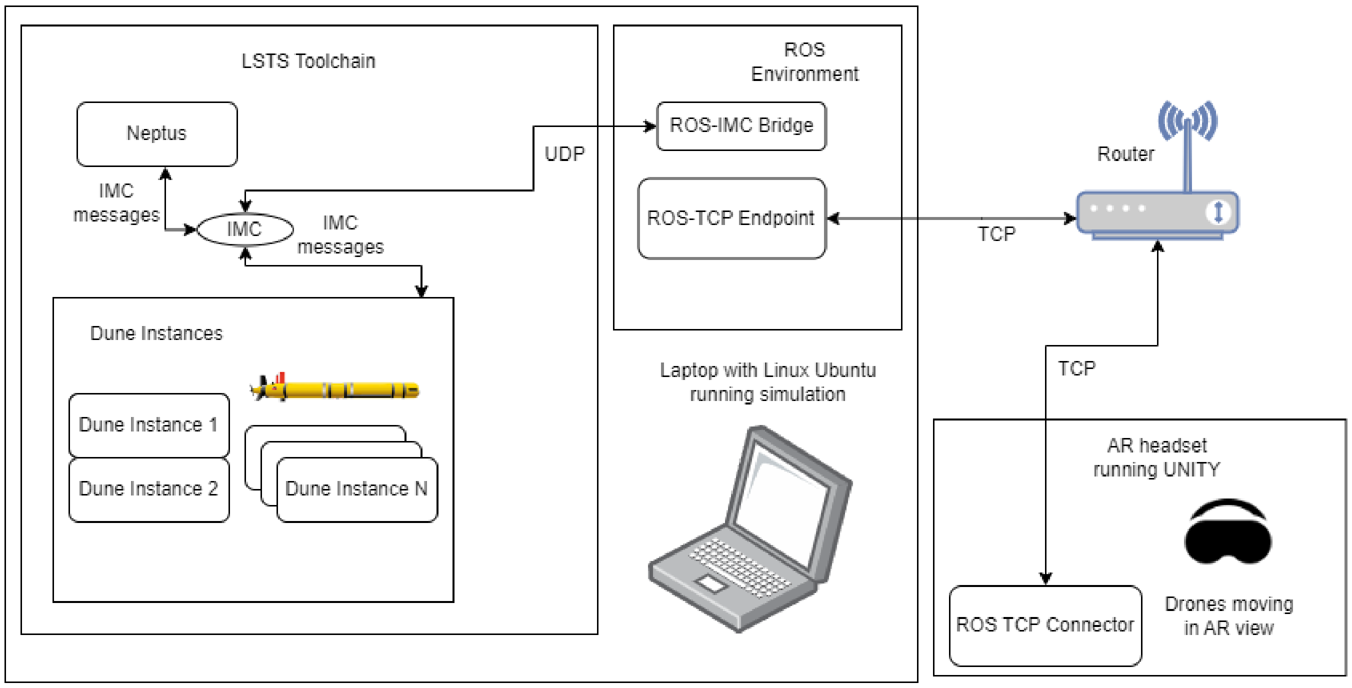


Figure 3.1: Setup used for testing the AR setup.

3.1 Components used

3.1.1 LSTS Toolchain

The LSTS software toolchain is a framework designed for the control, management, and supervision of autonomous ocean and air vehicles operating in challenging environments. (Ferreira et al. 2017) It addresses the increasing complexity that arises from the use of heterogeneous multi-vehicle teams in maritime operations. The most useful components of the toolchain for this project are Dune, which simulates AUV, ASV, and UAV vehicles, the IMC message protocol for communicating between the components, and Neptus for visualising and controlling the drones.

Dune, Unified Navigation Environment

Dune serves as a comprehensive operational environment, facilitating internal communication and control among various components within an autonomous vessel. These components include the positioning system (GPS, USBL, IMO), Doppler Velocity Log (DVL), communication modules and actuators (Ferreira et al. 2017). By utilising tasks, Dune enables seamless information sharing between individual components. A task, which acts as a fundamental subprogram, has the capability to subscribe to and publish information in the form of IMC messages (Fyrvik 2022).

One notable feature of Dune is its ability to simulate autonomous underwater vehicles (AUVs) used by the AUR-Lab (Autonomous Underwater Robotics Laboratory). Through the initiation of one or multiple Dune instances, the default drone models such as "lauv-simulator-1s" and "lauv-xplore-1s" or "caravela" for Autonomous Surface Vehicle (ASV) testing can be simulated.

IMC Inter-Module Communications

The Inter-Module Communications (IMC) component within LSTS plays a vital role in facilitating seamless communication among heterogeneous vehicles, sensors, humans, and local components. Serving as a robust communication protocol, IMC establishes a standardised framework comprising various data types and messages that can be exchanged between tasks within Dune and Neptus(Fyrvik 2022). IMC operates on the basis of a bus structure, enabling Dune-defined tasks to access and interact with all messages transmitted through the system via subscribe and publish functions. Each instance initiated by Dune possesses a unique IMC ID, ensuring that the source of every message can be precisely traced back to the corresponding Dune instance.

Neptus

Neptus serves as the graphical user interface for LSTS. It provides users with an interface to efficiently monitor enabled Dune instances (targets) and the current status of IMC messages within the system. (Dias et al. 2005) The Neptus interface offers a straightforward approach to planning various operations, including setting waypoints, defining row patterns for; searching, configuring loitering states, and more. Additionally, Neptus facilitates the process of uploading and executing these plans for one or multiple vehicles. Neptus is capable of using log files to replay older real or simulated missions in real time to allow for further review and analysis of older data. Historically, software tools such as Neptus have been crucial in effectively tracking LAUVs and ASVs through top-down 2D maps.

3.1.2 Robotic Operating System (ROS)

ROS (Robotic Operating System) is a popular open-source software platform widely used in the development of robotics applications. It offers a comprehensive range of tools and libraries that facilitate the management and coordination of various robot components, including sensors, actuators, and control algorithms. One of the key advantages of ROS is its modular design, which allows developers to incorporate new functionality into a robot by creating and integrating new code modules. This modularity, along with its strong online presence and pre-existing components designed for seamless integration with Unity, makes ROS an attractive choice for robotics projects.

ROS offers a wide range of capabilities and features that contribute to its popularity in the robotics community. One notable aspect is its support for a variety of programming languages, including C++, Python, and more, allowing developers to choose the language with which they are most comfortable. Additionally, ROS provides a rich collection of preexisting packages and libraries, offering ready-to-use solutions for common robotic tasks such as mapping, localisation, perception, and navigation. This extensive ecosystem of packages, combined with the active and supportive ROS community, makes it easier for developers to leverage existing tools and collaborate with others in the field.

In the context of message transmission, ROS follows a structure similar to Dune and IMC messages. Each type of IMC message can be converted into a ROS message, and instead of exchanging messages between Dune tasks, ROS Nodes communicate by publishing and subscribing to ROS topics. This communication system shares similarities with the subscription and publishing mechanisms used by IMC between Dune tasks. In particular, any ROS node can subscribe to or publish to any ROS topic, enabling the creation of a highly scalable system for building robotics applications.

IMC-ROS Bridge

The IMC-ROS Bridge package played an essential role in facilitating seamless communication between the IMC and ROS frameworks by enabling the translation of IMC messages into ROS messages(SMaRC n.d.). By initiating an ROS node, this package establishes a connection and subscribes to the IMC messages generated by Dune, harnessing them to publish equivalent ROS topics. This bidirectional functionality extends to controlling Dune instances or Neptus from ROS nodes as well, providing a versatile interface between the two systems. Specifically, in the context of this project, the IMC::EstimatedState and IMC::SimulatedState message types were used to track the estimated and simulated positioning data, respectively. These messages are then translated and made available as /IMC/EstimatedState and /IMC/SimulatedState topics within the ROS ecosystem.

ROS-TCP Bridge

To enable the transmission of data from the ROS environment, including the Dune vehicle positions, to the HMD running Unity, the ROS-TCP Bridge package is employed. The ROS-TCP Bridge package, developed by Unity, facilitates the transmission of ROS topics to Unity using a TCP connection. This package requires the IP-address of the laptop and a designated port to establish the TCP connection and transmit the ROS topics to Unity. For seamless integration, the ROS-TCP Bridge package is designed to work in conjunction with the Unity package called ROS-TCP Connector. The ROS-TCP Connector enables Unity to both subscribe to and publish ROS topics over TCP directly from the HMD, providing a seamless communication channel between the ROS environment and the Unity simulation.

3.1.3 Unity Game engine

The Unity game engine, developed by Unity Technologies, is a versatile and widely used cross-platform engine that is known for its application in various interactive applications, most notably video games and, in more recent years, VR/AR experiences. Alongside game development, Unity also caters to the creation of simulations, mobile apps, and other interactive content. Its popularity comes from its intuitive interface and high-quality graphics. Unity features a graphical user interface with a drag-and-drop component-based approach to development and uses C# as its scripting language. This combination makes Unity accessible to beginners and smaller teams, providing an easier learning curve for developers. Additionally, Unity boasts a vibrant and supportive community, offering extensive resources such as tutorials, documentation, and forums, empowering developers to dive into Unity and achieve their creative visions. In the context of this thesis, the Unity game engine was specifically chosen due to its proven track record as simulation software, as demonstrated in projects such as Autoferry Gemini (Vasstein et al. 2020), as well as some previous experience in using Unity for the development of a VR experience during the Experts in Teamwork course, highlighting its ability to create immersive virtual reality environments. To integrate augmented reality capabilities, the Oculus Integration SDK serves as the foundation for AR integration within Unity. Furthermore, to establish a connection with the ROS environment, the ROS-TCP Connector package is used, enabling seamless communication between Unity and ROS over TCP.

Oculus Integration SDK

The Oculus Integration SDK, initially developed by Oculus VR and now under the umbrella of Meta (formerly Facebook Technologies), is a comprehensive software development kit specifically designed to empower developers in creating virtual reality (VR) applications for the Meta HMD platform. This SDK comprises a collection of tools, libraries, and resources that streamline the development process and facilitate seamless integration with Meta's VR hardware. With the Oculus Integration SDK, developers gain access to a range of features, including virtual reality camera control, controller tracking, hand tracking, and an essential component known as passthrough AR that allows AR development. The SDK is accompanied by detailed documentation, which greatly simplifies the integration process compared to many other more general SDK approaches.

Unity objects

To populate the scene, a GameObject called ROSConnection based on the ROS-TCP Connector is added to the unity scene. This is then connected to the ROS environment over TCP using the ip-adress and port set by the ROS-TCP Endpoint. For this project, the IP of 10.53.0.207:11355 was used in conjunction with port forwarding to reach the LSTS network from anywhere. The object is now able to see, publish, and subscribe to all the topics in the ROS environment described in the message-type list. ROSConnection, can then be used as input in scripts embedded in other game objects such as information boards and other objects. ROSConnection was also the main movement generator in Unity. For every Drone found, a Sphere GameObject is spawned around the user using the body frame coordinates given in the /imc/estimated_state_imc topic in a realistic distance in relation to the user. A board of entity information is also placed and moved with the sphere to give a description of the vehicles distance and depth. This is shown in figure 3.2. More views from the AR perspective as well as default simulation condition from third person view can be found in the appendix.

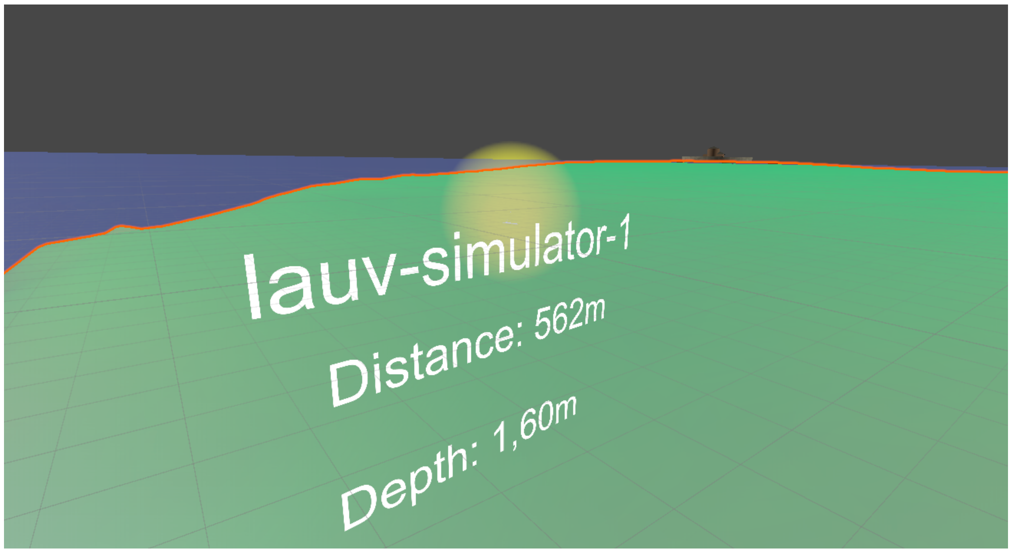


Figure 3.2: AR view of Drone1.

3.2 The Implementation

3.2.1 LSTS Toolchain

Dune Instances

For Dune instances, there was no need to write any new code. For each LAUV type that is in use, it is necessary to specify the types of IMC messages to send over UDP, specifically Announce, EstimatedState, SimulatedState, and StateReport. In addition, the port for the broadcasted UDP messages needs to be close to the port of the IMC_ROS_bridge. For this setup, the port of the IMC_ROS_bridge is set to 30106, so a good port for the Dune targets in the range 30101 - 30115.

In addition, each Dune vehicle has a configuration file used in the simulations that needs to be edited to send the desired IMC message types to the right ports. This was done by either editing the vehicle.ini files or creating a clone and naming it vehicle-saved.ini file, which has priority. The file is then edited by adding three subsections to set the desired values.

"Transports.Announce" enables the broadcast of announce messages and to which ports to broadcast. The "Announce" messages contain the names of the vehicles, such as "lauv-xplore-1".

The section "[Transports.UDP]" contains information on which IMC message types to be broadcast over UDP and which port to attempt to listen and broadcast from.

The "[Transports.Discovery]" section contains a list of ports that are available to the target to broadcast to. If the set port in "Transports.UDP" is taken, this is the list of available ports it will attempt to use instead. Ideally, this would be edited in the Task.cpp file to be capable of taking an integer range instead of integers, but for the purposes of this thesis it was not prioritised.

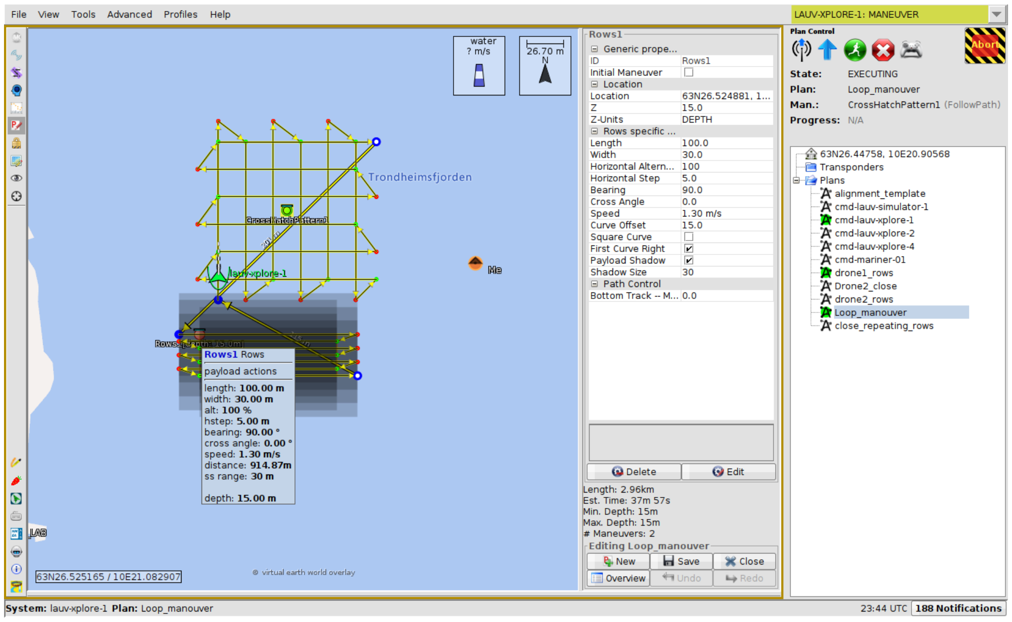
```
[Transports.Announce]
Announcement Periodicity = 10.0
Enable Broadcast = true
Enable Loopback = true
Enable Multicast = true
Multicast Address = 224.0.75.69
Ports = 30100, 30101, 30102, 30103, 30104, 30105, 30106, 30107, 30108, 30109, 30110,
↪  30111, 30112

[Transports.UDP]
Always Transmitted Messages = Abort, SimulatedState, EstimatedState, StateReport
Announce Service = true
Communication Range = 0.0
Contact Timeout = 30.0
Dynamic Nodes = true
Local Port = 30101
Print Incoming Messages = false
Print Outgoing Messages = false

[Transports.Discovery]
Multicast Address = 224.0.75.69
Ports = 30100, 30101, 30102, 30103, 30104, 30105, 30106, 30107, 30108, 30109, 30110,
↪  30111, 30112
Print Incoming Messages = false

```

In order to start a new drone using Dune, an instance is created. This is done by starting the ./dune executable and specifying the vehicle type such as Lauv-simulator-1. In addition, the instance can run with the -p flag, which allows the user to specify the profile, determining the type of vehicle that is being run. Typically, this is either Hardware or Simulation. For this project, only Simulation was used. An example of a Dune instance is shown below.

```
./dune -c lauv-simulator-1 -p Simulation
```

Neptus

Neptus is, as mentioned above, the graphical user interface used to monitor and control drones run by Dune instances with IMC messages. To control the vehicles simulated by the Dune instances, the Neptus console was used to plan and execute routes and missions for the vehicles to complete. An example of this can be seen in Figure: 4.2. In addition to simulating new scenarios, the Neptus MRA (Mission, Review and Analysis) tool will also be used to rerun older scenarios gathered from real-world tests with multiple vehicles. This is done by running the Data.lsf file generated during the real-world tests in MRA, and then starting "Mission Replay" and enabling "Network replay" as shown in Figure: 4.6.



Rows1

⊟ Generic prope...	
ID	Rows1
Initial Maneuver	☐
⊟ Location	
Location	63N26.524881, 1...
Z	15.0
Z-Units	DEPTH
⊟ Rows specific ...	
Length	100.0
Width	30.0
Horizontal Altern...	100
Horizontal Step	5.0
Bearing	90.0
Cross Angle	0.0
Speed	1.30 m/s
Curve Offset	15.0
Square Curve	☐
First Curve Right	✓
Payload Shadow	✓
Shadow Size	30
⊟ Path Control	
Bottom Track -- M...	0.0

Length: 2.96km
Est. Time: 37m 57s
Min. Depth: 15m
Max. Depth: 15m
Maneuvers: 2

Editing Loop_manouver

Rows1 Rows
payload actions
length: 100.00 m
width: 30.00 m
alt: 100 %
hstep: 5.00 m
bearing: 90.00 °
cross angle: 0.00 °
speed: 1.30 m/s
distance: 914.87m
ss range: 30 m

depth: 15.00 m

State: EXECUTING
Plan: Loop_manouver
Man.: CrossHatchPattern1 (FollowPath)
Progress: N/A

63N26.44758, 10E20.90568
Transponders
Plans
- alignment_template
- cmd-lauv-simulator-1
- cmd-lauv-xplore-1
- cmd-lauv-xplore-2
- cmd-lauv-xplore-4
- cmd-mariner-01
- drone1_rows
- Drone2_close
- drone2_rows
- Loop_manouver
- close_repeating_rows

System: lauv-xplore-1 Plan: Loop_manouver

Figure 3.3: View of the planning feature of Neptus.

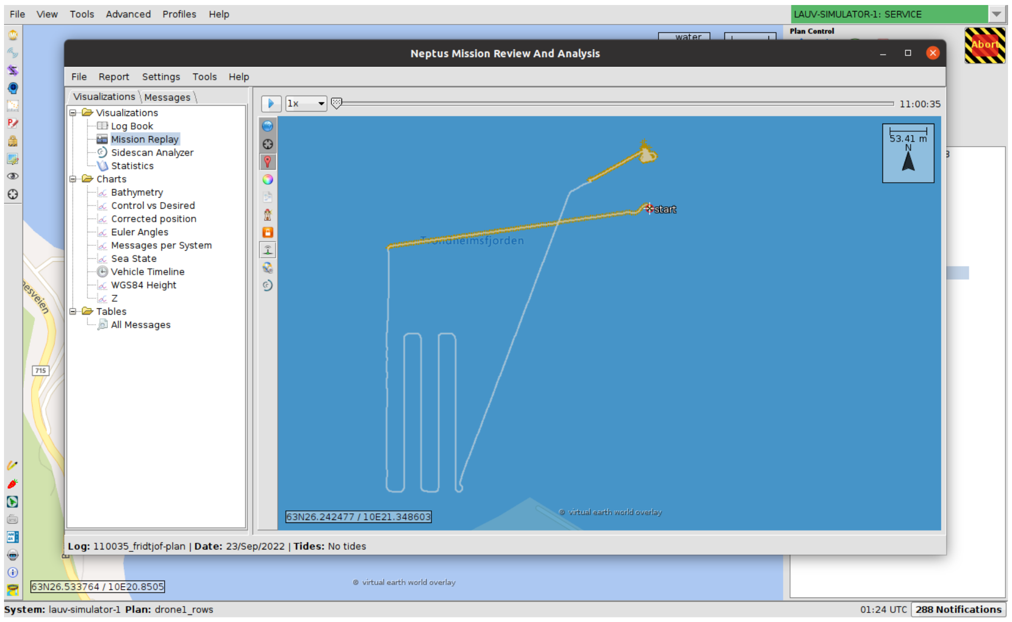

File View Tools Advanced Profiles Help
LAUV-SIMULATOR-1: SERVICE
Plan Control
Abort
Neptus Mission Review And Analysis
File Report Settings Tools Help
Visualizations Messages
Visualizations
Log Book
Mission Replay
Sidescan Analyzer
Statistics
Charts
Bathymetry
Control vs Desired
Corrected position
Euler Angles
Messages per System
Sea State
Vehicle Timeline
WGS84 Height
Z
Tables
All Messages
1x
11:00:35
53.41 m
N
start
Trondheimsfjorden
63N26.242477 / 10E21.348603
virtual earth world overlay
Log: 110035_fridtjof-plan | Date: 23/Sep/2022 | Tides: No tides
63N26.533764 / 10E20.8505
System: lauv-simulator-1 Plan: drone1_rows
01:24 UTC
288 Notifications

Figure 3.4: View of the Mission Replay feature of Neptus MRA.

3.2.2 ROS

For the ROS implementation, most of the work lies in understanding the components of the system and then implementing the transmission of the desired IMC messages. Since ROS uses a subscribe/publish relationship with its nodes, the IMC network can be translated nearly 1 to 1 by setting each IMC message type to be the same as an ROS topic. This is done by the imc_ros_bridge.

imc_ros_bridge

By default, the IMC_ROS_bridge was set up to transmit information from ROS to IMC, not the other way around. In order to allow messages such as EstimatedState to be translated from IMC to ROS, a EstimatedState.cpp file had to be added to the imc_to_ros folder containing the information in the message being transfered, as well as the source of the message, used for syncing info on the Unity side 3.2.2. In addition to the default normal contents of the messages, it is also necessary to include the source of the messages in order to handle messages from multiple drones at the same time. Specifically, the heading information src_id is fetched in the .cpp file of each message type and is added to the ros_msg topic before any other information 3.2.2.

```cpp
#include <imc_ros_bridge/imc_to_ros/EstimatedState.h>

namespace imc_to_ros {

template <>
bool convert(const IMC::EstimatedState& imc_msg, imc_ros_bridge::EstimatedState&
 ↪  ros_msg)
{
    ros_msg.src = imc_msg.getSource();
    ros_msg.lat = imc_msg.lat;
    ros_msg.lon = imc_msg.lon;
    ros_msg.height = imc_msg.height;
    ros_msg.x = imc_msg.x;
    ros_msg.y = imc_msg.y;
    ros_msg.z = imc_msg.z;
    ros_msg.phi = imc_msg.phi;
    ros_msg.theta = imc_msg.theta;
    ros_msg.psi = imc_msg.psi;
    ros_msg.u = imc_msg.u;
    ros_msg.v = imc_msg.v;
    ros_msg.w = imc_msg.w;
    ros_msg.vx = imc_msg.vx;
    ros_msg.vy = imc_msg.vy;
    ros_msg.vz = imc_msg.vz;
    ros_msg.p = imc_msg.p;
    ros_msg.q = imc_msg.q;
    ros_msg.r = imc_msg.r;
    ros_msg.depth = imc_msg.depth;
    ros_msg.alt = imc_msg.alt;

    return true;
}
}
```

Then the EstimatedState.h header file containing the variable types needed to be inserted into the /include/imc_ros_bridge/imc_to_ros/ folder.

```cpp
#ifndef IMC_TO_ROS_ESTIMATEDSTATE_H
#define IMC_TO_ROS_ESTIMATEDSTATE_H

#include <imc_ros_bridge/imc_ros_bridge_server.h>
#include "imc_ros_bridge/EstimatedState.h"
```

```
#include <IMC/Spec/EstimatedState.hpp>
#include <geometry_msgs/Pose.h>

namespace imc_to_ros {
bool convert(const IMC::EstimatedState& imc_msg, imc_ros_bridge::EstimatedState&
↪  ros_msg);
} // namespace imc_to_ros


#endif // IMC_TO_ROS_ESTIMATEDSTATE_H
```

Then for EstimatedState and Announce, the message type file .msg was already created, however for the StateReport which is used for latitude/longitude checking, the message file needed to be added. 3.2.2

```

add .msg file here

```

And finally to start the necessary processes to check for the messages and link everything together, the bridge_node.cpp file needs to be edited to start the new bridge component. For EstimatedState, this was done by adding the line as shown below 3.2.2.

```

Add bridge node lines here.

```

ROS-TCP-Endpoint

For the ROS-TCP-Endpoint, the only task it has is to read all the topics available in the ROS network and transmit them to the ROS-TCP-Connector running in Unity. Therefore, the setup consists of setting the correct ip adress and port in the endpoint.launch file like shown 3.2.2

```

<launch>
    <arg name="tcp_ip" default="192.168.1.11"/>
    <arg name="tcp_port" default="11355"/>

    <node name="unity_endpoint" pkg="ros_tcp_endpoint"
    type="default_server_endpoint.py" output="screen">
        <param name="tcp_ip" type="string" value="$(arg tcp_ip)"/>
        <param name="tcp_port" type="int" value="$(arg tcp_port)"/>
    </node>
</launch>

```

3.3 Unity

Unity is a game engine, designed for 2D, 3D, and XR games. For this project, the starting point used is the example scene made by the oculus SDK for showcasing Passthrough, which consists of an empty scene with some floating objects in an augmented reality space. From there all assets in the scene except for the camera model and the object manipulator GameObject were deleted. This allows for a simple starting point when it comes to enabling the Passthrough AR mode. The GameObject connections used in the project can be seen in

3.3.1 Connecting to ROS

ROSConnection GameObject

In order to connect to the ROS network, the "ROSConnection" object is created and added to the scene as an invisible entity. The component "ROS Connection" from the package ROS-TCP-Connection can then be added to the GameObject "ROSConnection", and the default ip and port information can be set in the editor. The game object will now automatically connect to the ros network on progam start, either in the editor or in the headset. However, in order to use the connection and the rostopics the custom component Ros_drone_movement is created. The component functions as the ros message handler, as well as starting some of the other startup functionalities of the application such as initial positioning in relation to the seabed.

Generating drones

As stated above, the RosConnection object handles the reading and processing of ROS messages. In this thesis, the two main types of messages are:

- EstimatedState, which contains positional information on each drone.

- Announce, which contains naming information on the drone.

When a message is received, the source of the message is read first, then if no match is found in the array of current targets, a new GameObject is created based on the preface GameObject AUV. For this project, all received messages are shown as the same type AUV by the programme, however, this could be changed type of vessel specified by the Announce message. When the object has been created and added to the scene, the relevant information from the message is added to the object as well as the source of the message. If the message is an EstimatedState, the position of the drone is moved to the corresponding position stated by the message. If the message type is Announce, the name of the object is displayed by the text box below the drone (Figure: 3.2). If no Announce message is received, the name of the drone is set as "Drone(ID of message source)", eg "Drone64".

Syncing drones names with positions

Using EstimatedState as a source of drone positions makes for a relatively simple method of reading the drone positions relative to the starting position as well as the depth. It is, however, an issue with syncing the read positions from the EstimatedState message with the specific relevant drone, as there is no drone name in the EstimatedState message. The solution is to use the source (src) data point from the header of the message to pinpoint the message source, then this source can be used on the Announce message to find the name of the drone from the sys_name parameter.

syncing drone position to lat long

Now that the source of every EstimatedState message is known, the position of every drone can be placed in the simulated world. Using the StateReport IMC message, a known lat, long is given as the origo in a NED reference frame. By placing the Ned frame in the game world which has a default origo position of NTNU Biological Station using the lat long in StateReport, the drone and user positions can be placed in the game world.

Input fields

To enable a dynamic IP and Port useage that can vary based on the dynamically set ip of the server PC, two input fields have been inserted into the scene. These will be used for the set IP and Port that the ROSConnection object uses to connect to the ROS environment. In order to interact with the fields using the VR controllers, the ObjectManipulator object script was rewritten to look for trigger input from the controller and then look for interactions with the input field.

To make ROS messages compatible, the imc messages ".msg" files were added to a rosMessages folder. From there message types, type classes were autogenerated into, for example, EstimatedStateMsg.cs files. For some reason, the class names are generated incorrectly to "RosMessages/EstimatedState" and needs to be edited into "imc_ros_bridge/EstimatedState"

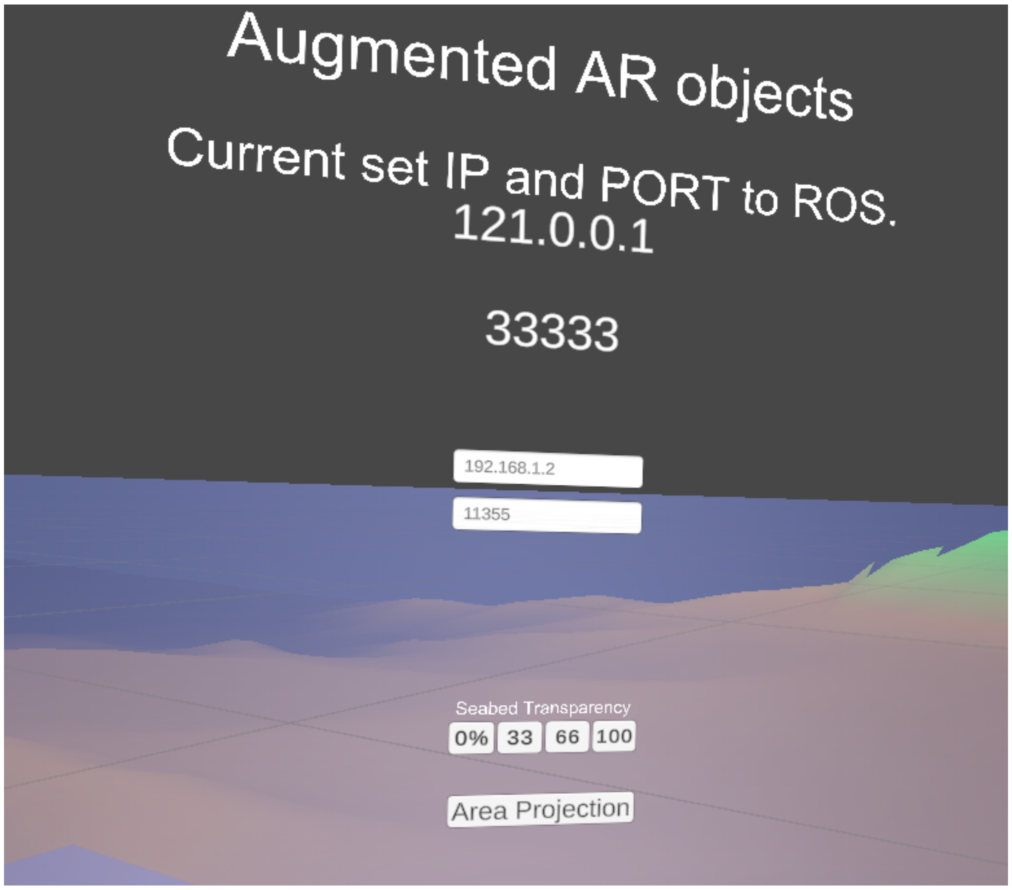


Figure 3.5: Interface for transparency control, ip/port input, and Hand-projection view.

Importing the seabed

In order to import the seabed, multiple methods were attempted, and there surely are many methods of gathering the data, this is the method used for this simple example using only Trondheimsfjorden. First, the depth data of the seabed is collected from GeoNorge(GeoNorge 2023) in the 3D object format ".tif". These data are then imported into a programme called QGIS software which is able to read the.tif file and display it as a 3D map. Then using the imported QGIS plugin "Qgis2threejs" it is possible to export the map as a ".gltf" file. This ".gltf" file can then be imported into the unity project and placed as a GameObject in the scene using the ".gltf" import plugin "GLTFUtility". It is now possible to interact and manipulate the map of the seabed as any other 3D object.

The imported ".gltf" model of the seabed, now a GameObject in the scene, is then applied with a passthrough filter, allowing it to be transparent to the real world on command. The level of transparency can then be adjusted using the seabed transparency buttons in the world edit pane. When a button is clicked, the passthrough opacity level is adjusted using the "update-Transparency" script attached to the map object and the transparency is adjusted to the percentage specified on the button 3.5.

Visualizing depth on Model using shaders

In the earlier tests of the seabed model, it became apparent that understanding distance and depth based on eye sight to the model was harder than expected. To handle this, an automatic coloring shader has been created to paint the 3D model according to depth.

A shader is a script used to contain mathematical calculations and algorithms to calculate the output colour and style of each pixel rendered on the object, based on the input of lighting, the material configuration, and the shape of the object to be rendered (Unity 2023). For the colour-changing shader script, four colours are chosen as input, one for ground colour (anything above -1m), and then three colours for the depth gradient. The minimum and maximum depth are also entered into the component in the editor; however, this can be changed to dynamically changing based on the size of the object being rendered. The shader then goes over every rendered pixel in the objects, checks the height of the point, and then gives a shade of the three input colours based on the height. Upon viewing the seabed model from above, it became quickly apparent that seeing where the land ended and the sea began is nearly impossible from the shape alone. Therefore, in addition to the colour change with depth, any piece of the 3D model above -1m is coloured brown. The result can be seen in the following images taken from a distance. Figure: 3.6, 3.7

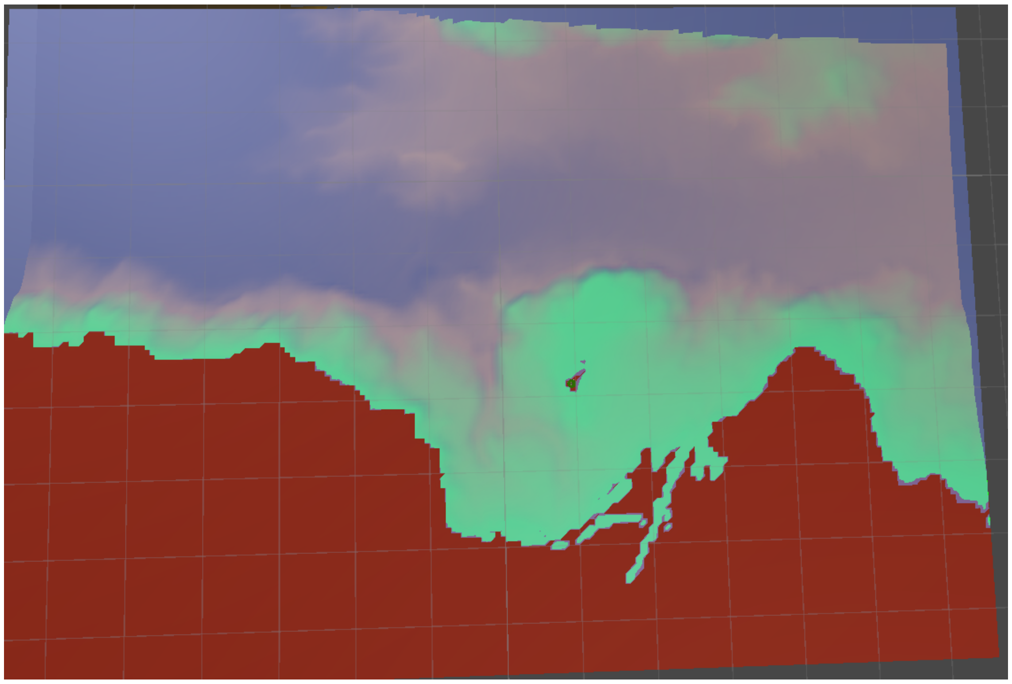


Figure 3.6: Top-Down view of Trondheimsfjorden.

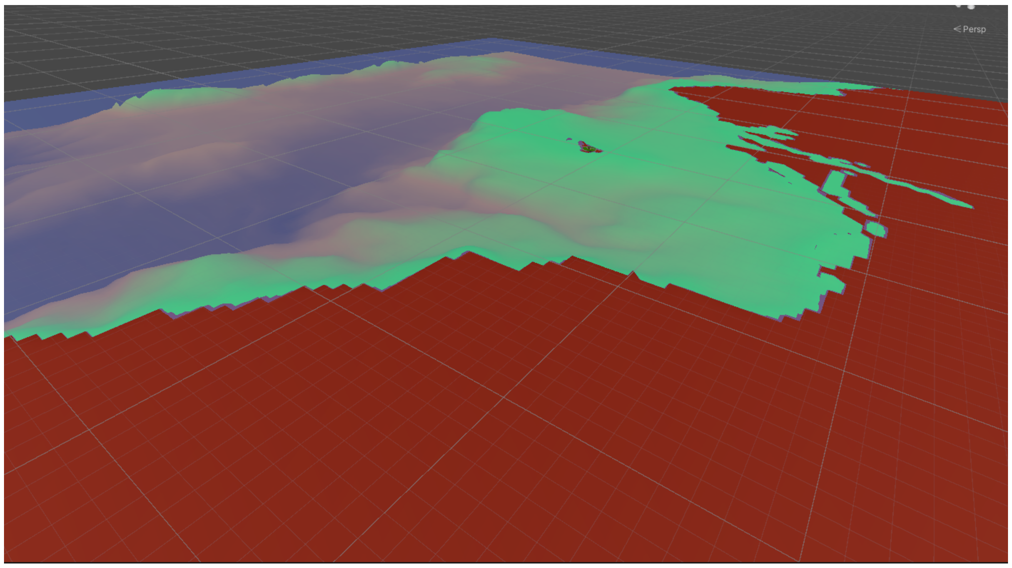


Figure 3.7: Isometric view of Trondheimsfjorden.

Positioning in reference to seabed

The positioning of the player and the targets can then be placed in reference to the seabed surface using the AlvinXY translation implemented in the LatLongTranslate script. 2.4.5

Hand projection/top-down view

With the full-scale seabed functioning and positioned correctly in relation to the given latitude and longitude of the user, the next thing that became apparent while testing was the user's desire to move around the scene and/or get a top-down view of the environment and simulation. This would of course be possible by either moving or scaling the seabed model around the user, however, this would remove the user from the current situation in case of an ongoing encounter and would also increase the risk of motion-sickness for inexperienced AR/VR users. Instead, a method of projecting the simulated state onto the top of the user hand has been devised 3.8. To do this, a default, empty smaller version of the fjord is running with a disabled render status from the start of the programme. When the "Area Projection" button (see Figure: 3.5) is pressed, the GameObject changes to a visible render status, making it visible to the user, and begins updating its position to the top of the user's hand. The user is then able to view the situation from multiple angles and rotate using the controller direction as the projection rotation direction.

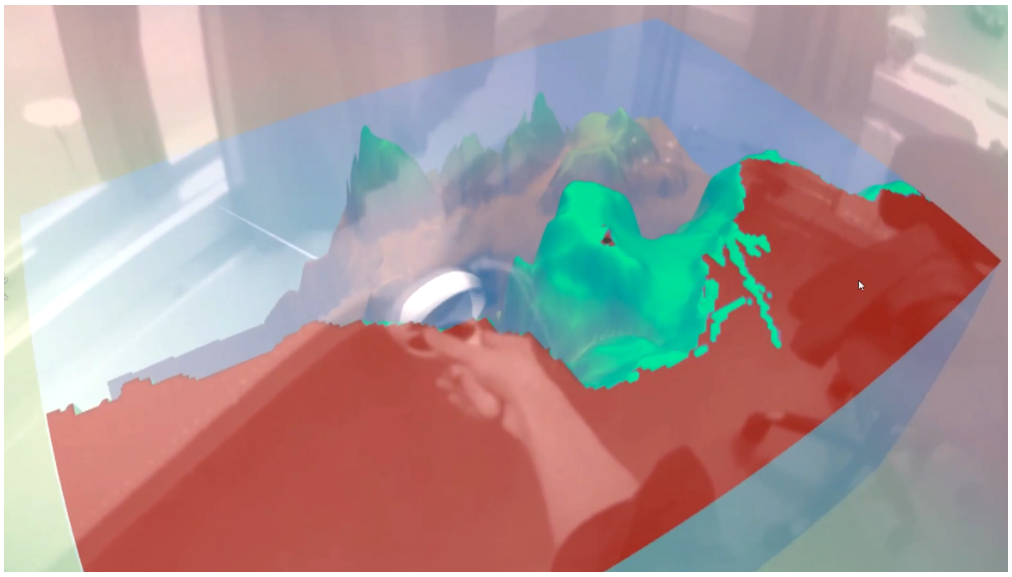


Figure 3.8: Projection view of Trondheimsfjorden, projected to user hand.

Chapter 4

Results

4.1 Results

The project setup was tested using two situations where simulated LAUVs were used using Dune run on a laptop running Linux. In case 1, One ASV and one LAUV is running simultaneously in front of the user. The idea is to see how easy it is to differentiate between the surface ASV and the submerged LAUV. In case 2, two LAUVs are running in two patterns at two depths to show the possibility to keep track of multiple submerged LAUVseven at long distances using the e.

The project setup was tested using several cases in which simulated targets ran simulated cases by running plans made in Neptus. Two cases are shown in this results section, as well as a demonstration of the system, shown in video given in Appendix A.1.

4.1.1 Case 1

For the first case is focused on showing the system working with one surface running agent, and one submerged agent. Running the pattern shown in figure 4.1. During the pattern, the LAUV, named Drone1 in unity was set to move a depth of 15m depth in a row pattern of length 300m and width 75m, while the ASV, named Drone2 was to move ontop of the first drone and maintain position to simulate aiding in positioning using USBL. Both drones were moving at a velocity of 2.5m/s

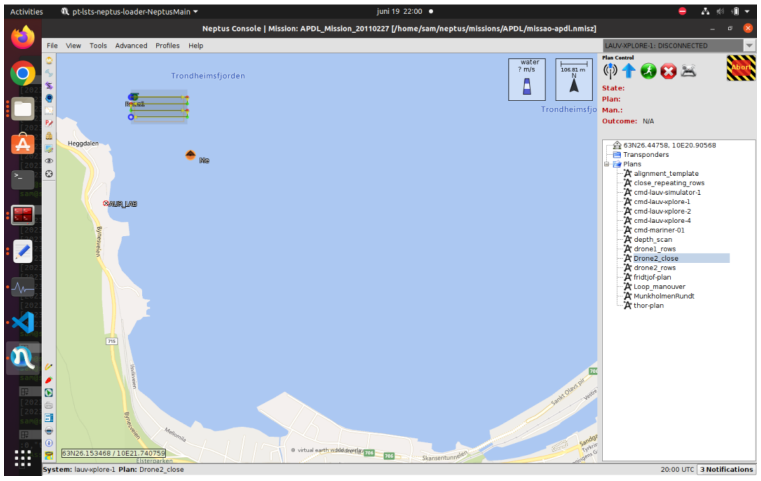


Figure 4.1: Path used in case 1.

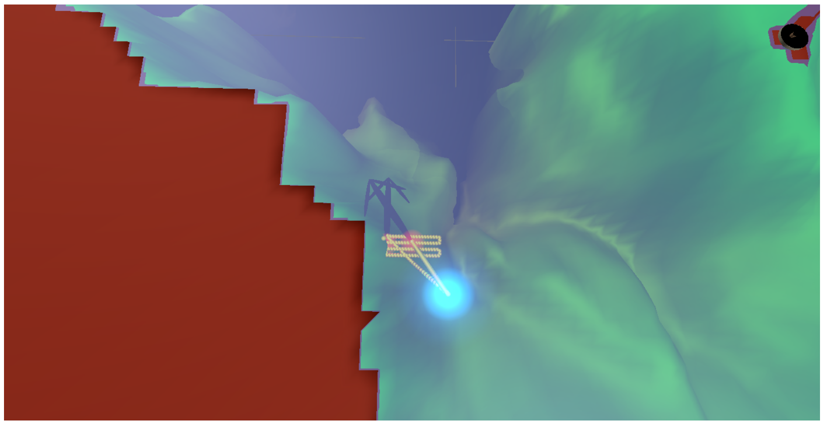


Figure 4.2: Top down view of the drone movements in Unity.

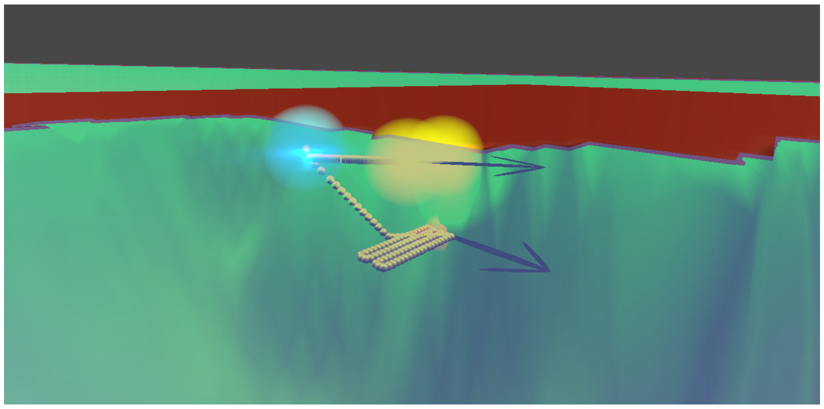


Figure 4.3: Side-view of the drone movements in Unity.

The tracking of the drones can be seen as the yellow dots in the 3D object. The case shows the drones moving in a correct manner to in accordance with the simulation.

4.1.2 Case 2

The second test shows the system's ability to display the movements of multiple drones in a larger area. There are two LAUV drones running two larger routes simultaneously, as shown in Fig. 4.4. The first drone is moving along a route around the user and moves in a simulated scan by running a crossing pattern of 100m at a depth of 90m. The drone then returns to the south of the user. The second drone moves simultaneously in a pattern around munkholmen. At the north side of munkholmen it runs a simulated scan by moving in a row pattern at 90m with a length of 300m, and a width of 100m. The drone then continues on its path around munkholmen to end up south of the user.

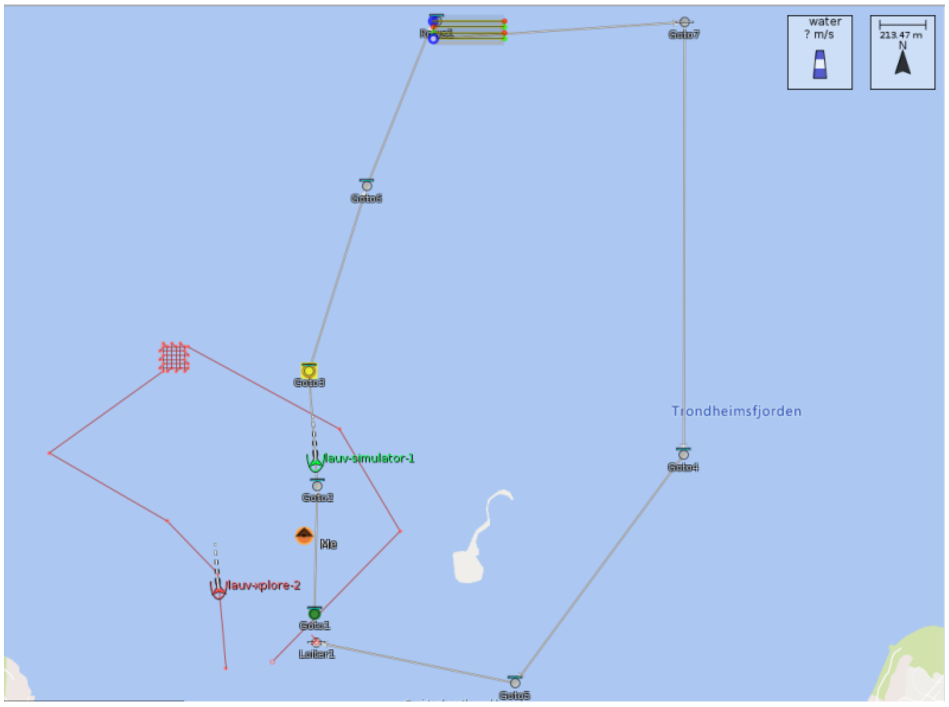


Figure 4.4: Path used in case 2.

As shown both vessels were sucessfully tracked along the entire path as shown by the yellow dots. Both images are taken of the view given by the Area Projection feature of the software.

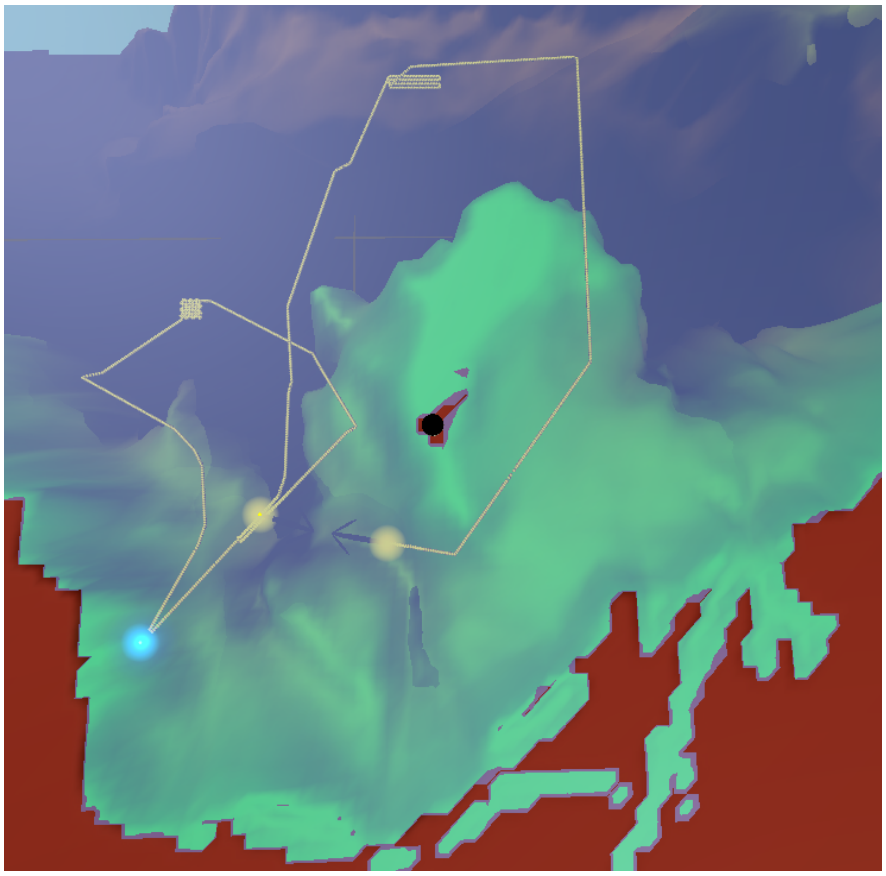


Figure 4.5: Top down view of the drone movements in Unity.

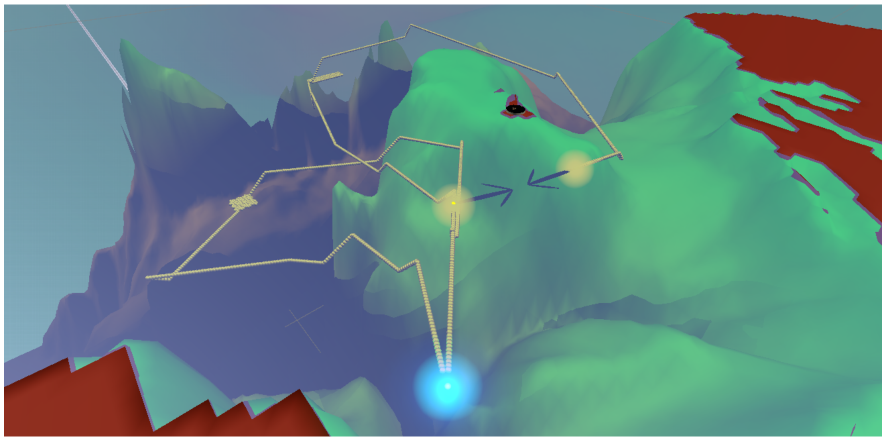


Figure 4.6: Side-view of the drone movements in Unity.

Chapter 5

Discussion

This thesis has shown a method of connecting the LSTS toolchain to a AR/VR platform in order to give a new and improved method of tracking Unmanned Vehicles in a marine setting. The setup was then tested with three cases, showing the versitility in the setup, as well as the strengths. The method of setting up the connection is to use an ROS network to function as a unifying bridge that can be used more easily to make the system compatible with other platforms than the LSTS toolchain, such as Gazebo, without having to do major changes in the Unity software.

By making the system an application running natively on the OQ2 headset, the necessary hardware for running the setup is drasticly simplified when compared to running the software on a PC tethered headset. The LSTS Toolchain instances and ROS are running on a Dell laptop from 2013 running Ubuntu with only two cores, no graphical processor, and 6Gb of RAM. A PC-tethered AR/VR setup will require a substantially more powerful computer to run, and would in most cases require the user to be physically tethered to the compouter at all times with either a USB-C or DisplayPort cable. Due to the simplicity of the setup, it is easy to move the setup to new locations, requiring only the laptop running Ubuntu, a router, and the OQ2. If the laptop is running port forwarding, then the user only needs an internet connection to use the virtual reality headset and use the visual components of the setup. This simplicity was an integral part of the idea of the system.

For the setup used in the thesis, the port range 30100 -30114 was used to broadcast UDP messages, with the IMC_ROS_Bridge holding the 30112 port. This is because the Dune system was not designed to run a large number of instances at the same time, leading to the port range having to be hardcoded to every dune instance individually. This is, however, not a perfect solution, as sometimes the Dune instance is still not able to establish a connection, at which point the solution is to restart the instance, which works most of the time. There is no documentation that i have found that describes this problem, but the issue lies most likely in the networking solution of the Dune instances.

Originally, the method used to translate the coordinates was based on the conversion of one number range to another. However, this required more points of information by syncing two points in unity to two latitude-longitude coordinates, as well as more complex math to reach the translation. This was changed to the AlvinXY method, which only needs one comparison of latitude-longitude to unity coordinate conversion. AlvinXy is accurate as long as the distances in use are not much longer than a few kilometers, and since the accuracy of translation matters the most when the drone is close, this is acceptable.

The experiments were carried out in a controlled testing environment under predetermined conditions. The absence of real-world environmental variations, such as varying sea states, underwater currents, and weather conditions, may limit the generalizability of our results. The performance of our system may differ when deployed in more dynamic and unpredictable environments.

To import the seabed, a multitude of processes and methods were tested. The final solution, using the .tif file from GeoNorge in QGIS is a valid method as long as you are willing to do the steps manually and then make a synchronisation point by hand. However, if a larger area is to be used or the system is to be tested in a range of environments, then it is possible to improve the workflow. One method would be to automate the process of extracting depth data and converting it to a .gltf file on demand. This can be done using API calls to extract mapdata and file conversion scripts such as Simblis Seacharts (Blindheim 2021) to generate a 3D file. However, for this thesis and for most use cases where the target area is not too large, the manual method is sufficient.

The resolution seabed bathymetry was set to 50m. This was the highest resolution i were able to obtain with my access and need as a student and was for the most part enough for this thesis. Only when the drone was near the seabed or vertical walls did it sometimes appear beneath the seabed. There is however a 5m resolution bathymetric scan available from GeoNorge that requires the same steps for implementation. With the right clearence and usecase, this higher resolution would give a more realistic view of the seeabed and give a clearer view of the drones position in relation to the seabed and points of interest near the drone such as shipwrecks and natural formations.

Chapter 6

Conclusions and further work

6.1 Conclusions

For this project, the objectives set was set in order to make a platform for further development of a bridge between the internal setup of the LSTS toolchain and a AR system to be used for control and monitoring. During this project the following objectives were achieved

- A literature revue was conducted explaining the basics of the project.
- An environment was set up to run simulations using the LSTS toolchain using Dune instances and neptus.
- A pipeline method was set up to transfer data from the IMC format through ROS and then to a Augmented Reality headset running Unity.
- A Unity program was created that could run natively in the headset, automatically connect to the IMC-network, and display the drones simulated in Dune in real time.
- The resulting system was then tested as seen in case 1 and case 2, as well as the appended video demoing the system. A.1

ROS connections were stable and gave continuous data on drone positions during case tests, showing the stability of both the ROS-TCP connector bridge and the IMC-ROS bridge. The text displaying information on the drones was at times hard to read, which could and should be fixed by either dynamic color correction or the ability to set the color by user input. The setup is shown to be functioning in the setup shown for the project. Even though the setup is of simple form, the concept is shown to be functional and is a proof of concept for further work going forward.

6.2 Further Work

Further work is needed to make this solution a viable software for real use, however the basic platform has been set up and is ready to be developed in the future. Concepts that could be focused on could be:

- The current AR view is of a very simple, and all of the drones use the same LAUV model. This could be fixed by adding a set of 3D models to the AR program and use the IMC ID to select the proper drone model based on the vehicle_type parameter.
- The current version of the Unity software logs the movements that have been recorded since the software started. It is however possible to at least also plot the plans of the individual vehicles using the planDB IMC-message type. Using this type, the waypoints given by the plan could be placed in the Area Projection model and connected to give a better understanding of the vehicles current route.
- A method for controlling the drones using AR could also be implemented using the planDB IMC message type. Using the Area Projection 3D model for reference, a new set of op waypoints could be selected on the projection which would then generate a new plan which could be sent back to the Dune instance.
- Navigation Uncertainty Visualisation. For the tests running the older tests run by Jens, the parameters NavigationUncertainty were given as a calculated uncertainty in the x and y directions. This could be implemented in the Unity model as a coloured or otherwise marked area surrounding the drone, representing the possible locations it could be with the given uncertainty.
- There were times when the signal between LSTS and Unity was lost, either due to simulation errors or net packet losses between the connectinos. This meant the drones in Unity froze until contact was re-established. Applying State Estimation either in ROS environment or in Unity by implementing a Kalman filter using the input from the sensor models to estimate future states in the event of signal loss or noise.
- When it comes to the transformation between WGS 84 and the NED frame, ideally for a more general solution the middle of the seafloor 3D object would have a known latitude and longitude, which would then be set to correspond to x = y = 0. But for this thesis, such a generalisation was not prioritised.
- For a future version of the software, the ability to change perspective would be beneficial to gain a better view and understanding of the drones prospective. With the combination of a normal or a 360° camera, the feed can be sent to the user and displayed in front/around the user to immerse the user further.

Bibliography

Bae, Inyeong and Jungpyo Hong, (May 2023). “Survey on the Developments of Unmanned Marine Vehicles: Intelligence and Cooperation”. In: *Sensors* 23, p. 4643. DOI: 10.3390/s23104643.

Bellingham, J.G., (2009). “Platforms: Autonomous Underwater Vehicles”. In: *Encyclopedia of Ocean Sciences (Second Edition)*. Ed. by John H. Steele. Second Edition. Oxford: Academic Press, pp. 473–484. ISBN: 978-0-12-374473-9. DOI: https://doi.org/10.1016/B978-012374473-9.00730-X. URL: https://www.sciencedirect.com/science/article/pii/B978012374473900730X.

Blindheim, Simon, (2021). *Python API for reading and manipulating polygon data of Electronic Navigational Charts (ENC)*. https://github.com/simbli/seacharts. [Accessed 3-June-2023].

Dias, P.S., S.L. Fraga, R.M.F. Gomes, G.M. Goncalves, F.L. Pereira, J. Pinto, and J.B. Sousa, (2005). “Neptus - a framework to support multiple vehicle operation”. In: *Europe Oceans 2005*. Vol. 2, 963–968 Vol. 2. DOI: 10.1109/OCEANSE.2005.1513187.

Equinor, (2021). *Shaping the future with HoloLens*. URL: https://loop.equinor.com/en/stories/shaping-the-future-with-hololens (visited on Apr. 15, 2023).

Ferreira, António Sérgio, José Pinto, Paulo Dias, and João Borges de Sousa, (2017). “The LSTS software toolchain for persistent maritime operations applied through vehicular ad-hoc networks”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 609–616. DOI: 10.1109/ICUAS.2017.7991471.

Fyrvik, Torbjørn Reitan, (June 2022). *Methods for Ocean Mapping with Combined ASV and AUV Platforms*.

GeoNorge, (2023). *GeoNorge Kartkatalog Dybdekart*. Depth map of Trondheim seabed. Datapoints retreived from GeoNorges 50m resolution scan. URL: https://kartkatalog.geonorge.

no/metadata/dybdekart/9285f93e-d6d7-498c-a289-35700d795fc1 (visited on Mar. 15, 2023).

Kongsberg, (2020). *Autonomous Underwater Vehicle (AUV), Hugin.* URL: https://www.kongsberg.com/maritime/products/marine-robotics/autonomous-underwater-vehicles/AUV-hugin/ (visited on June 5, 2023).

Mikalsen-Schwenke, Sondre, (Dec. 2022). *Improved Situational Awareness For USV/AUV Operations.*

Murphy, Chris and Hanumant Singh, (Oct. 2010). "Rectilinear coordinate frames for Deep sea navigation". In: pp. 1–10. DOI: 10.1109/AUV.2010.5779654.

Pérez, Tristan and Mogens Blanke, (June 2023). "Simulation of Ship Motion in Seaway". In.

Raymarine, (2020). *CLEARCRUISE AUGMENTED REALITY.* URL: https://www.raymarine.com/en-us/our-products/marine-cameras/augmented-reality (visited on Apr. 10, 2023).

SMaRC, (n.d.). *Minimal library for bridging ROS and IMC messages.* https://github.com/smarc-project/imc_ros_bridge. [Accessed 4-Dec-2022].

Unity, (2023). *Materials, Shaders & Textures.* Documentation on shaders in Unity. URL: https://docs.unity3d.com/560/Documentation/Manual/Shaders.html (visited on June 7, 2023).

Vasstein, Kjetil, Edmund Brekke, Rudolf Mester, and E Eide, (Nov. 2020). "Autoferry Gemini: a real-time simulation platform for electromagnetic radiation sensors on autonomous ships". In: *IOP Conference Series: Materials Science and Engineering* 929, p. 012032. DOI: 10.1088/1757-899X/929/1/012032.

Wikipedia, (2023). *Local tangent plane coordinates — Wikipedia, The Free Encyclopedia.* http://en.wikipedia.org/w/index.php?title=Local%20tangent%20plane%20coordinates&oldid=1156666928. [Online; accessed 2-June-2023].

Appendix A

Appendix

A.1 Video of Project Demo

A video made to display the capabilities of the system is shown here:
https://www.youtube.com/watch?v=2DMdZkv5JeQ

A.2 Code

A.2.1 GitHub repository

The github repository used during the development of this master thesis:
https://github.com/SondreM-S/master_project.git

The project uses LSF for handling the large files of the Unity software and also requires the user to install the components of the LSTS toolchain in the conventional manner prior to use. The version of Ubuntu used is 20.04, with noetic as the ROS distribution version used. Unity development was done on a Windows 10 machine, using Unity 2022.1.16f1.

A.2.2 Shader Code

```
// Original base script created by Mark Johns - https://twitter.com/Doomlaser
// Adapted to support multicolor shading and adjustable depth by Sondre
 ↪  Mikalsen-Schwenke
Shader "Custom/HeightColorBlendRelative"
{
    Properties
    {
        _Color ("Color", Color) = (1,1,1,1)
        _SecondColor ("Secondary Color", Color) = (0.5, 0.5, 0.5, 0.5)
        _MaxColor ("Color in Maxmal", Color) = (0, 0, 0, 0)
        _GroundColor ("Color of ground above 1m depth", Color) = (0.5, 0.1, 0.02, 1)
        _MinDistance ("Min Distance", Float) = 0
        _MaxDistance ("Max Distance", Float) = -500
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
        _Glossiness ("Smoothness", Range(0,1)) = 0.5
        _Metallic ("Metallic", Range(0,1)) = 0.0
    }
    SubShader
    {
        Tags { "RenderType"="Opaque" }
        LOD 200

        CGPROGRAM
        // Physically based Standard lighting model, and enable shadows on all light
         ↪  types
        #pragma surface surf Standard fullforwardshadows

        // Use shader model 3.0 target, to get nicer looking lighting
        #pragma target 3.0

        sampler2D _MainTex;

        struct Input
        {
            float2 uv_MainTex;
            float3 worldPos;
            float4 pos : POSITION;
            float3 screenPos;
            float4 color : COLOR;
        };

        half _Glossiness;
        half _Metallic;
        float _MaxDistance;
        float _MinDistance;
        fixed4 _Color;
        float4 _SecondColor;
        float4 _MaxColor;
        float4 _GroundColor;


        // Add instancing support for this shader. You need to check 'Enable Instancing'
         ↪  on materials that use the shader.
        // See https://docs.unity3d.com/Manual/GPUInstancing.html for more information
         ↪  about instancing.
        // #pragma instancing_options assumeuniformscaling
        UNITY_INSTANCING_BUFFER_START(Props)
            // put more per-instance properties here
        UNITY_INSTANCING_BUFFER_END(Props)

        void surf (Input IN, inout SurfaceOutputStandard o)
        {
           // float3 localPos = IN.worldPos -  mul(unity_ObjectToWorld,
            ↪  float4(0,0,0,1)).xyz;
           float3 localPos =  mul(unity_WorldToObject, float4(IN.worldPos,1)).xyz;
           half4 dist = localPos.z;
           float distFloat = localPos.z;

           half4 weight =  (dist - _MinDistance) / (_MaxDistance - _MinDistance);
           float weightFloat = (distFloat - _MinDistance) / (_MaxDistance -
            ↪  _MinDistance);
           //Go through multiple colors based on weight, _Color to _SecondColor to
            ↪  _MaxColor
           // Weight goes from 0 to 1, _Color to _SecondColor to _MaxColor
           half4 distanceColor = lerp(_Color, _MaxColor, weight);
           if (weightFloat < 0.5)
           {
               distanceColor = lerp(_Color, _SecondColor, weight*2);
           }
           else {
               distanceColor = lerp(_SecondColor, _MaxColor, (weight - 0.5) * 2);
           }
           if (distFloat > -1) { // Color everything above 1m depth brown as ground
            ↪  reference
               distanceColor = _GroundColor;
           }

            // Albedo comes from a texture tinted by color
            fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Color;
            o.Albedo = IN.color.rgb * distanceColor.rgb ;
            // Metallic and smoothness come from slider variables
            o.Metallic = _Metallic;
            o.Smoothness = _Glossiness;
            o.Alpha = c.a;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```