NTNU | Norwegian University of Science and Technology

Department of Engineering Cybernetics

TTK4900 - Master's Thesis

Remote vessel survey using VR

Authors:
Jostein Sætra Schefte
Einar Lorentsen

June, 2021

Abstract

In this thesis, a VR application for conducting remote vessel surveys was developed in the Unity game engine. The task was given by the marine insurance company Gard. The COVID-19 pandemic has made it difficult for Gard to conduct physical surveys due to travel restrictions. In addition, conducting physical vessel surveys can be costly and leaves an environmental footprint because of airplane travel. Current methods used by Gard for conducting remote surveys were deemed not satisfactory, and a new method was therefore proposed. This method proposes to provide vessel crew with a LiDAR sensor and a 360° camera. The provided equipment could then be used to scan and photograph the vessel. The gathered material would then be sent to a surveyor that would explore a virtual environment generated from the material in VR.

By using a LiDAR sensor integrated in the latest mobile devices from Apple, accurate 3D models of vessels could be generated from LiDAR scans. The quality of the LiDAR scans are good for simple environments but becomes insufficient in more complex environments. To supplement the LiDAR model to provide a higher level of detail, 360° images from a GoPro max were used. The 360° images hold a high quality, and relevant details such as rust and cracks are visible from them. Taking a LiDAR scan and the necessary 360° images take only a few minutes. From the gathered material, a virtual environment can quickly be created. This environment can then be explored in the VR application. Tools that provide functionality necessary to perform surveys were developed and integrated in the application. Users can seamlessly switch between the LiDAR model and 360° images, collaborate and communicate in the virtual environment with others via online functionality, and document their findings with a virtual camera and audio recorder. In addition, users can interact with informational datapoints retrieving information about relevant objects from a database, measure distances in the LiDAR model and download other virtual environments from within the application. All implemented tools work as desired.

The developed application is demonstrated to surveyors at Gard. The surveyors find the technology promising. However, they think the quality of the LiDAR scans are too low. The surveyors think a method like the one presented in the application could be used for some clients as a sorting tool to decide which vessels are worth investigating in further detail physically. All things considered, the developed application is considered successful as it enables virtual walkthroughs of vessels combined with working surveyor tools. A video demonstrating the developed application can be seen in the following video link.

Sammendrag

I denne masteroppgaven ble en VR-applikasjon for eksterne skipsundersøkelser utviklet i spillmotoren Unity. Oppgaven ble gitt av det marine forsikringsselskapet Gard. Covid-19 pandemien har gjort det vanskelig for Gard å gjennomføre fysiske skipsinspeksjoner på grunn av reiserestriksjoner. Å gjennomføre en fysisk inspeksjon kan også være kostbart og forurensende siden inspektører vanligvis reiser med fly. Nåværende metoder Gard har brukt for eksterne undersøkelser har ikke vært ansett som gode nok. En ny metode ble derfor foreslått. Den nye metoden foreslår å tildele mannskap på skip en LiDAR sensor og et 360° kamera for å scanne og ta bilder av de relevante områdene på skipet. Materialet kan så sendes til en inspektør som genererer et virtuelt miljø fra det. Det virtuelle miljøet kan så utforskes i VR.

Ved å bruke en LiDAR sensor som er inkludert i nyere mobile enheter fra Apple, får man nøyaktige 3D modeller fra scannene. Kvaliteten på LiDAR scannene er bra når enkle omgivelser scannes, men blir fort dårligere hvis omgivelsene blir komplekse. For å ta hensyn til dette brukes 360° bildene til å supplementere LiDAR modellen og gi et høyere detaljnivå enn det LiDAR sensoren klarer. For å ta 360° bilder brukes et GoPro Max kamera. Kameraet gir bilder av høy kvalitet med gode detaljer. Det er mulig å se relevante detaljer som rust og sprekker fra bildene. Å ta en LiDAR scan og de nødvendige bildene av en omgivelse tar bare noen få minutter. Fra det innsamlede materialet kan man lage et virtuelt miljø av de skannede omgivelsene. Dette miljøet kan så utforskes av en inspektør i VR. Verktøy som gir nødvendig funksjonalitet for å gjennomføre en skipsinnspeksjon ble implementert og lagt til i den utviklede applikasjonen. Brukere kan sømløst bytte mellom LiDAR modellen og 360° bildene, samarbeide og kommunisere med andre via nettverksfunksjonalitet og dokumentere mulige funn med et virtuelt kamera. I tillegg kan brukere interagere med informasjonspunkter som henter info om relevante objekter fra databaser, måle avstander i LiDAR modeller og laste ned andre virtuelle miljøer fra applikasjonen. Alle de implementerte verktøyene fungerer som forventet.

Den utviklede applikasjonen ble demonstrert for inspektører på Gard. Inspektørene synes teknologien virker lovende, men synes kvaliteten på LiDAR scannene var for dårlig. Inspektørene tror at metoden som ble demonstrert kunne blitt brukt som et sorteringsverktøy for å bestemme hvilke skip som er verdt å undersøke nærmere fysisk for noen av klientene deres. Med alt tatt i betraktning blir den utviklede applikasjonen ansett som vellykket, da den gjør det mulig å undersøke skip virtuelt i tillegg til at alle de utviklede verktøyene fungerer som de skal. En video som demonstrerer den utviklede applikasjonen kan bli sett ved å trykke på den følgende video lenken.

Acknowledgements

We would like to thank our supervisors Sverre Hendseth at NTNU and Stephen Bibby at Gard. You have both been to enormous help. The thesis would not have held the same quality without your help. A special thanks will also go to Gard who provided us with the necessary equipment, Eivind Kværnø at NTNU who helped us acquire access to the VR Lab at Øya helsehus and to DNV for reaching out to shipping companies on our behalf. Thank you to Egil Ulvan Rederi and the friendly crew at Kristian With for allowing us on board to perform scans. We would also like to thank all our interview subjects for providing invaluable insight. Thank you to Christian Bauer-Nilsen who lent us his music stand that were used as a camera tripod.

Contents

Chapter 1

Introduction

1.1 Context and motivation

Vessel surveys are conducted to make sure that a vessel follow a set of guidelines necessary to operate safely at sea. Some examples of the tasks performed during a survey are examining the general condition of the vessel, and performing different tests of the equipment on board. These surveys are usually conducted physically on board the vessel, however due to the COVID-19 pandemic performing these surveys physically have been difficult due to travel restrictions. A possible solution to this problem is to perform remote surveys.

An advantage of performing remote surveys is that the environmental impact of the survey is reduced, as travel by airplane for the surveyors is avoided. Increasing the amount of remote surveys could be one of the means taken to reach the climate goal of reducing greenhouse gas emission by at least 50% by 2030 compared to the level in 1990. This goal is set by the Norwegian government (Miljødirektoratet, 2021). The cost associated with a survey could also be reduced by performing remote surveys. This is because accommodation cost and travel expenses are avoided. Physical surveys also takes a significant amount of time, usually an entire day, which may impact the vessel's operational availability which could result in a financial loss for the shipowner. By performing a remote survey this problem is removed as there is no need for the vessel to be inspected to be at port. This is because a surveyor can perform the survey with the help of the vessel crew remotely.

Virtual Reality (VR) is the use of computer technology to create an immersive simulated environment that can be explored in 360 degrees. VR is used in a wide variety of industries today. Some of these are education, entertainment, architecture, retail, and engineering. VR devices and development frameworks have seen huge technological improvements the last few years. This has further accelerating the demand for VR technology worldwide.

A LiDAR sensor measures the distance to objects, and stores all distances in a point cloud. Accurate 3D models of real life environments can be generated from a point cloud. This technology enables real world environments to be scanned efficiently in a matter of minutes.

Recent technological advances have made it possible to fit LiDAR sensors in modern tablets and smartphones, making 3D scanning available to everyone. It is therefore possible for everyone to create a detailed 3D model of any object or area desired. The generated 3D models from a LiDAR scan can be imported directly into VR applications, enabling real world environments to be explored virtually.

1.2 Problem description

The project task is given by Gard, one of the world's largest marine insurers. The task was to develop an application which enables virtual walkthroughs of vessels. After conducting several interviews with employees at Gard, it is found that developing an application which improves the currently used method for remote surveys would be the most helpful use-case for the application.

1.3 Project goal

The travel restrictions imposed by the COVID-19 pandemic has severely reduced Gard's ability to conduct physical vessel surveys. To reduce the amount of outstanding surveys, Gard has been experimenting with remote surveys. When performing remote surveys, a surveyor from Gard has sat down with the shipowner of the vessel to be inspected, and been presented photos from it. Gard has concluded that this way of conducting remote surveys is not satisfactory and has stopped performing them. The method was deemed not satisfactory because the surveyor could be presented biased information and photos from the shipowner that did not accurately reflect the current condition of the vessel to be inspected.

As a result, Gard now faces a large amount of outstanding vessel surveys. The goal of this thesis is therefore to implement a VR application that offers an improved method for conducting remote surveys compared to previous methods used by Gard. It is believed that by utilizing VR technology, a more realistic and interactive remote survey can be performed. To achieve this, the application to be developed should include several tools and functionality which makes important elements of a physical survey available in the virtual environment. In addition, the virtual environments to be explored must accurately represent the vessel to be surveyed. This thesis will therefore also investigate if LiDAR sensors and 360° cameras can be used to generate accurate virtual vessel models.

1.4 Methods

To enable a virtual survey, 3D models of vessels will be explored in VR. To generate the 3D vessel models, the LiDAR sensor present in modern iPhones and iPads will be used. This is because it is considered to be highly available both now and in the future as it is included in an increasing amount of Apple's devices. To achieve a higher level of detail in the virtual environment, 360° images will be used to supplement the 3D models. The Unity game engine and its XR Interaction Toolkit will be used to develop the VR application. This is to accelerate the development process as there is only 4 months available to implement the application. In order to develop useful surveyor tools for the application, several interviews with some of Gard's surveyors will be held to identify core features of the application.

1.5 Project overview

This thesis is divided into 7 chapters and 2 appendices. In addition, a video demonstrating the material gathering process and the application's functionality is also provided. This video can be viewed by clicking the following hyperlink: YouTube. A short description of the chapters and appendices are given below.

- **Chapter 2:** Contains theory about the technology used in the project. It also presents how this technology is already being used in various industries today. In addition, general information about Gard and vessel surveys will also be included.
- **Chapter 3:** Presents which tools will be used to solve the problem task. Which means taken to avoid VR sickness together with an overall structure of the application will also be presented.
- **Chapter 4:** Describes the implementation of the developed VR application. In addition, this chapter describes how material for the application is gathered.

- **Chapter 5:** Presents the results of the developed application. A video demonstrating the application's functionality is added as an attachment for this chapter.
- **Chapter 6:** Discusses the results from the previous chapter. These discussions include the quality of the developed virtual environments and the application's tools. In addition, alternative use cases and general reflections regarding the utilized technology is held.
- **Chapter 7:** Finalizes the thesis with a conclusion of the results.

- **Appendix A:** Contains information about how to download the project files of the developed VR application.
- **Appendix B:** Contains summaries of all conducted interviews and meetings for this thesis. These summaries will be used as references throughout the thesis.

Chapter 2

Background

2.1 Virtual Reality

Virtual Reality (VR) is the use of computer technology to create a simulated, immersive environment. The virtual environment can either simulate the physical real world or a fictional environment. Unlike traditional user interfaces through a two-dimensional screen, VR enables interaction inside a 3D world. VR technology provides a unique way to interact with an expanding digital landscape in a natural way. The purpose of VR is to create an environment that mimic how we interpret the world around us. When done correctly, a virtual reality experience can convince users that they are physically located within the virtual world.

To experience virtual reality, special glasses, speakers and motion sensors are used. These components are often included in a headset. In addition, hand controllers are often used to interact with the simulated environment. There exist several ways to display the simulated environment in virtual reality. Different displays include single large projection screens, multiple connected projection screens, stereo-capable monitors with desktop tracking, and head-mounted displays (HMDs).

2.1.1 Types of VR displays

A single large projection screen can be used to visualize 3D models on a screen. When wearing specialized glasses multiple users can view the same model, but from their own perspective. This can be seen in Figure 2.1a.

A richer experience can be achieved by using multiple connected projection screens as seen in Figure 2.1b. Here, the field of view is broadened, making it easier to obtain the full image. Two drawbacks of using these technologies are that they both require a significant amount of space for the projection screens in addition to their high costs.

A less stationary option is a head-mounted display as seen in Figure 2.1c. This type of display follows the heads movement. When a user of a head mounted display move around physically, the same movement is occurring in the simulated environment. Traditionally, head mounted displays has had one screen for each eye, which requires two screens to be refreshed every frame. New headsets however have one big screen which cover both eyes.

Oculus Quest 2 is a head-mounted virtual reality system created by Oculus, a brand of Facebook Technologies. The Quest 2 is capable of running as a standalone headset with an internal Android-based operating system, or externally powered by a PC when connected over USB. Running the headset externally requires a computer with a sufficient graphics card which can be expensive. When running internal applications, the Quest 2 is completely wireless as neither the headset or the controllers needs to be connected to a computer, making the Quest 2 easy to use wherever desired without any restrictions. When released, the Quest 2 received critical acclaim as a big leap forward in consumer VR technology (Lynch, 2020). The intuitive user interface and its simple set-up was especially praised, as it will make VR more accessible for everyone. The Quest 2's guardian system uses the headsets internal cameras to automatically set up a virtual cage around the user to prevent him from colliding with real world objects while using the headset. In addition, the headsets cameras can be used for automated hand tracking. The headset's display is a singular fast-switch LCD panel with a 1832×1920 per eye resolution, which can run at a refresh rate of up to 90 Hz.

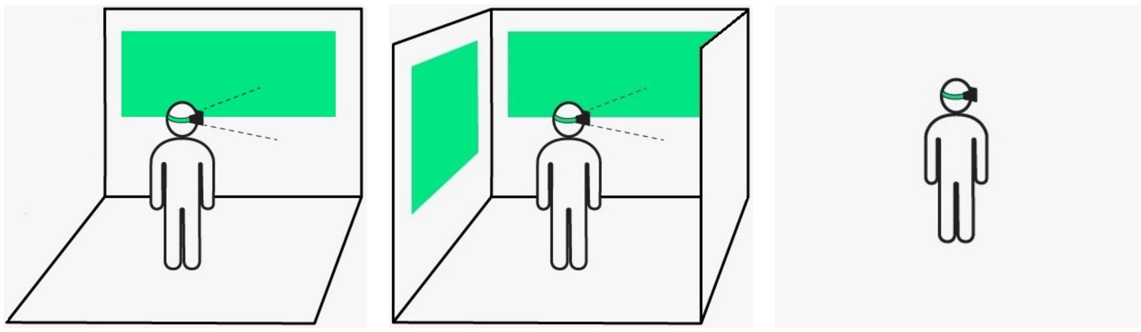

(a) A single projection screen (Delight XR, 2021). (b) A multiple projection screen (Delight XR, 2021). (c) A head-mounted display (Delight XR, 2021).

Figure 2.1: Different virtual reality display systems.

2.1.2 Controllers and tracking system

A critical component of VR applications is interacting with the virtual environment. Different tracking systems are used to enable the position and orientation of both head and hands of a user which immerses him in the application being run. Tracking can be performed using optical, magnetic, ultrasonic or intertial tracking, and is a crucial part of calculating the correct visual perspective viewed by a user (Berg and Vance, 2017). Hand-held controllers allow users to navigate and manipulate objects within the virtual world (Bowman et al., 2008). This also allows for haptic feedback which provide a stronger understanding of how objects in a virtual world physically interact. Other feedback VR systems can provide is vibration, wind, temperature, and pressure. This again enhance interactions if it is suitable for the application. Today, many head-mounted displays include cameras which allow automated hand tracking (Switzer, 2020).

2.1.3 VR today

VR technology is used in a wide variety of industries today. One common application of VR technology is in the entertainment industry for video games, movies and social interactions. Because of a growing market and cheaper computational power, virtual reality systems can be made at much lower costs today than what was possible a few years ago. VR technology is also used for educational purposes for simulating training scenarios, often performed on digital twins of real world physical objects. In architecture and engineering, VR technology can help to better envisage a project and present it to clients. The growth VR technology has experienced, has been greatly accelerated due to the recent covid-19 pandemic. The latest forecasts from IDC research predicts that the worldwide spending on AR and VR will experience a compound annual growth rate of 54.0%, resulting in 72.8 billion dollars in 2024 (IDC, 2020). The commercial use cases that are forecast to receive the largest investment growths by 2024 apart from video games are education, industrial maintenance, and retail showcasing.

By utilizing VR technology it is possible to enhance how virtual environments are perceived. In research conducted by Berg and Vance, 2017 for the National Science Foundation, people in the manufacturing and product design business were asked how they experienced production-stage prototypes in VR. One engineer described the experience of sitting in a production vehicle after experiencing it virtually: "I've been here, this is surreal.. if I turn around I'll see the radio, yep, there it is!". This highlights the power of VR, not only for the manufacturing and product design business, but for every business needing to visualise environments virtually. Virtual reality provides a natural way to interact with 3-dimensional data.

Virtual reality systems most commonly track only one viewpoint, allowing one user to control the experience. Head-mounted displays has one display which only the user can see. However, it is possible to share the perspective of what is displayed inside the head-mounted display to an external screen. This results in better communication between the user inside the VR system and other team members watching. VR systems can thus be used as an efficient collaboration tool where multiple users can interact. Another benefit is that a VR system can be connected to the internet. This allows users to connect with each other across boarders, making virtual collaboration possible.

2.1.4 Limitations and challenges

Traditionally, the programs run within a VR headset has been run externally from a computer connected to the VR headset via a cable (Barnard, 2019). A PC powerful enough to run these programs and send their outputs to the headset requires powerful graphics cards which are expensive. The headsets themselves does not come cheap either. Therefore, until recently, VR has been reserved for enthusiast PC gamers with a budget or industrial usage. Having the headset connected to a powerful PC significantly restrict where the headset can be used since it requires a powerful nearby computer. A users mobility is also restricted while using a headset connected to a computer by the connecting cable. However, as technology has progressed computers has become smaller and faster. This has allowed powerful computers to fit inside VR headsets. An example of this progress is the wireless headset series created by Facebook named Oculus Quest. This headset runs any program internally, proving that these limitations can be circumvented.

One major challenge of both externally and internally run VR headsets are the presence of motion sickness their users can experience (Thompson, 2020). VR motion sickness occurs when a users brain receives conflicting signals about movement in the virtual environment with respect to the users physical movement. In VR, this essentially means that if a user is standing still and the virtual environment is moving, the user's brain's equilibrium is disturbed and the user can start to feel nauseous. By using a high quality, 6-degrees of freedom, low latency VR headset and gradually increasing the time spent in VR, the experienced motion sickness can be decreased. As developers continue to gain a better understanding of what causes motion sickness in VR, future applications can be designed to minimize and maybe even remove the problem.

2.2 Modeling

Several techniques and methods can be used to create a virtual model of a real world object. One of these are generating the virtual representation using a LiDAR scanner, which will be presented in subsection 2.2.1. Another method for generating a virtual representation of a real world object is by using machine learning on floor plans to generate 3D models, as discussed in subsection 2.2.2. 360° imaging can be used as a supplement to the above modelling methods as presented in subsection 2.2.3.

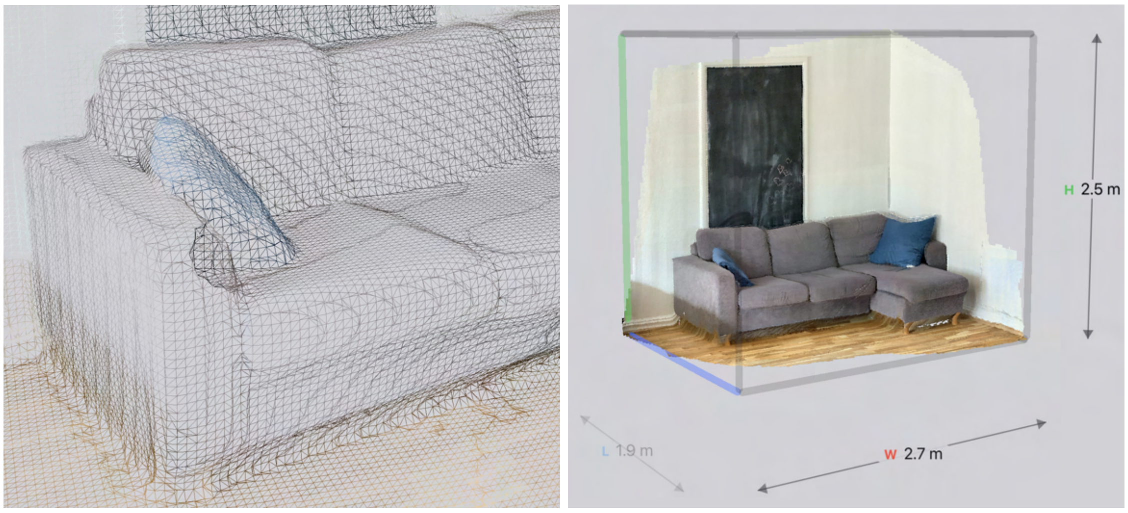
2.2.1 LiDAR

LiDAR is a remote detection and ranging method used to measure distances, and is short for Light Detection and Ranging. It is a sensor which emit infrared light pulses and measures how much time it takes the light pulses to return after hitting nearby objects. The time between the output laser pulse and the detection of the reflected pulse allows the LiDAR sensor to calculate the distance to each object with an accuracy depending on the sensor. This is possible because the speed of light is known, and thus the travel time seen in Equation 2.1.

$$d = \frac{c \cdot t}{2} \tag{2.1}$$

Where d is the distance, c is the light speed, and t is the time it takes the light to bounce back to the sensor. It is divided by 2 because the light has to travel the same distance twice as it returns.

Each second a LiDAR sensor captures millions of such distance measurement points. A point cloud which is a 3D matrix of detected points, is generated from a LiDAR scan. A 3d model of the scanned environment can be generated from the point cloud. Great accuracy can be achieved, and LiDAR sensors can provide reliable results over short and long ranges with an accuracy of millimeters.



(a) Point cloud generated by a LiDAR scanner. The points are connected to form vertices.

(b) A textured 3D model generated from the scans point cloud.

Figure 2.2: LiDAR scan of a sofa with 20mm accuracy.

A key benefit of using a LiDAR sensor is the ability to perform well under any light conditions. This is because a LiDAR sensor only measures the reflection of the infrared light pulses that is emitted, which behave the same regardless of existing lighting conditions. The resulting point cloud of a LiDAR scan can be converted into a 3D map of the scanned environment. A LiDAR scan can be supplemented with different sensory data, such as images, to get a better understanding of the scanned environment. The point cloud surfaces is then combined with images of the scanned environment, creating a textured 3D model. LiDAR sensors are currently used in many industrial applications, ranging from scanning entire buildings, as presented in subsection 2.3.1, to navigation in autonomous driving vehicles.

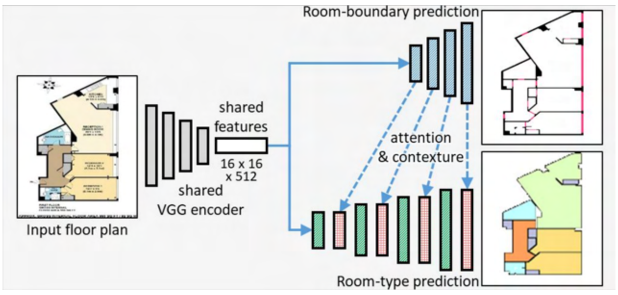
There exist two types of LiDAR sensors, mechanical scanning LiDARs and a solid-state LiDARs. A mechanical LiDAR can sample a large area simultaneously by rotating the sensor up to 360 degrees or by using a rotating mirror to steer a light beam. It provides a detailed mapping of a scanned environment. However, a large price, complexity, and reliability issues makes mechanical LiDARs an unatractive option. A solid-state LiDAR is built without any moving mechanical parts, and scans an environment incrementally. This makes it highly efficient at processing data, as well as being cheaper according to LeddarTech, 2020. A solid-state LiDAR scanner can be either stationary or handheld.

2.2.2 Machine Learning - 3D model from floor plan

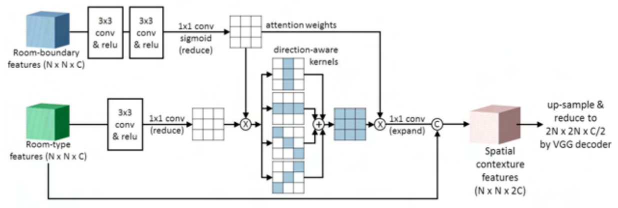
3D models of objects and environments can be achieved by generating the necessary parts of it using specialized algorithms. If enough data about the real life object to be visualized is known, algorithms can recreate the real life object in a virtual environment as discussed by Bjorn Mes in section B.3. Mes was able to accurately replicate a vessel virtually, using only data about the vessel and an algorithm.

Another method used for constructing 3D models of vessels is to construct a 3D model from their floor plan. By using a deep neural network to predict room-boundary elements, a digital representation used to reconstruct a 3D model is possible, as described by Zeng et al., 2019. The paper presents a new approach to recognize elements in floor plan layouts, such as walls, doors, windows, and different types of rooms. Since elements are recognized and labeled, a 3D model reconstruction of the floor plan can be made.

The architecture of the network created by Zeng et al., 2019 can be seen from Figure 2.3a. Here, a deeper version of a convolutional neural network called VGG is deployed to extract features from an input floor plan image. Then, the network is divided into two networks with different tasks. One predicts the room-boundary pixels, e.i. walls, doors, and windows. This is the floor-plan elements that separate room regions. The other task predicts the room-type pixels, i.e. dining room, kitchen, etc. The network thus learns the shared features common for both tasks, then use two separate VGG decoders to make different predictions.

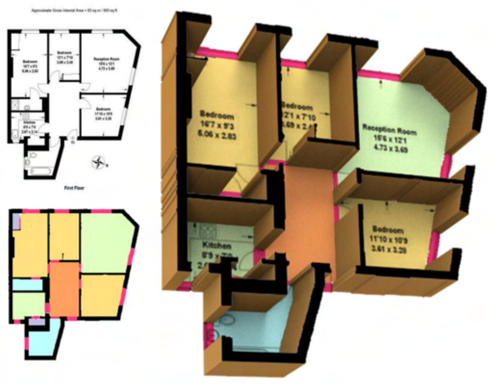


(a) Overall network architecture of the deep neural network created by Zeng et al., 2019.



Room-boundary features (N x N x C)
3x3 conv & relu
3x3 conv & relu
1x1 conv sigmoid (reduce)
attention weights
direction-aware kernels
Room-type features (N x N x C)
3x3 conv & relu
1x1 conv (reduce)
1x1 conv (expand)
Spatial contexture features (N x N x 2C)
up-sample & reduce to 2N x 2N x C/2 by VGG decoder

(b) The attention & contexture layer in Figure 2.3a to help make room-type predictions by Zeng et al., 2019.



(c) The floor plan recognition result (lower-left) and the reconstructed 3D model (right) by Zeng et al., 2019.

Figure 2.3: Dummy figure

To help with making room-type predictions, a spatial contextual module is created as seen in Figure 2.3b. Here, the input to the top branch is the room-boundary features from the top VGG decoder (see the blue boxes in Figure 2.3a and Figure 2.3b). The input to the bottom branch is the room-type features from the bottom VGG decoder (see the green boxes in Figure 2.3a and Figure 2.3b). The result from the different levels is the spatial contextual features, which help the features integration for room-type predictions (Zeng et al., 2019).

After the VGG network has been trained, correct predictions of room-boundaries and room-types from floor plans can be made. This can be used to create a 3D model based on the predicted results as seen in Figure 2.3c.

2.2.3 360° imaging

360° imaging, also known as omnidirectional imaging, uses cameras to create a high resolution 360° field-of-view which show the entire scene at hand. It works by combining several photos into a clear 360° view using advanced algorithms. Several different industries have applied this technology. The car industry use it to provide visual assistance to drivers. A clear benefit of using these cameras is the enhanced overview it gives the user.

There are two types of videos which can be seen in a 360° field-of-view, monoscopic videos and stereoscopic videos. Monoscopic videos are flat renderings, meaning that there is no depth perception, captured by the 360° cameras. This is the most common type of 360° media, and is captured using a single lens system. Monoscopic renderings are commonly used for mapping, for instance in Google Street View (90Seconds, 2020). Stereoscopic video is on the other hand captured using a twin lens system, mimicking how humans use their eyes to perceive depth and distance (Viewport, 2021). Stereoscopic video thus add another level of immersion by adding depth data between the foreground and the background. An example of the added depth perception of stereoscopic video can be seen from Figure 2.4.

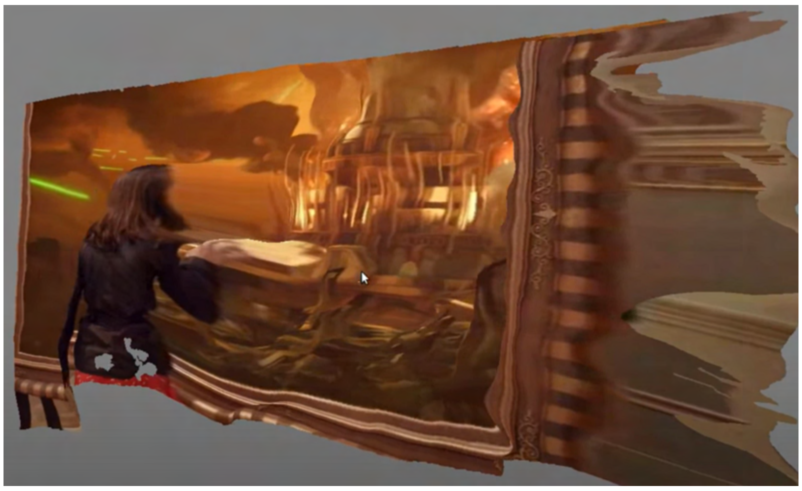


Figure 2.4: A stereoscopic image showing a woman closer to the camera than the image in the background (Terence Eden, 2013).

2.3 Related work

The recent advances in VR technology and LiDAR scanning have increased the adoption of the technologies for various use cases across several industries. As technological progress continues, it is likely to believe that this adoption will only accelerate. In subsection 2.3.1, the use of LiDAR to create a digital twin of a factory is presented. In subsection 2.3.2, use-cases for VR technology in the maritime industry is presented. These use cases are VR training for vessel crew and the use VR technology for conducting remote vessel surveys.

2.3.1 LiDAR scans and VR for factory expansions in the process industry

Erling Tønnesen, consultant at Sweco, one of Europe's leading firms in architecture and engineering consulting, has utilized LiDAR scanning in several of his projects. In some of the projects, VR technology was combined with the LiDAR scans to enable virtual exploration of them. In a meeting, Tønnesen discussed his views on the technologies and talked about two projects where he utilized LiDAR scanning (section B.1).

Hennig-Olsen Is, Norways oldest and now largest producer of ice cream, have conducted multiple LiDAR scans of their factory in Kristiansand, Norway. The LiDAR scans have an accuracy of milimeters and are being used by multiple teams working on different projects at the factory. Since the factory opened in 1960, the factory has seen multiple expansion and renovation phases. Each phase had their own floor plans and process documentation of various quality, which made it hard to get a complete overview over the documentation of the factory in its current state. Detailed LiDAR scans of the factory with high accuracy were therefore taken with the help of Tønnesen.

The scans made it possible to get an accurate digital twin of the factory that could be explored by anyone who had access to its online portal. This helped getting an overview of the details of the factory as well as speeding up future expansion and renovation phases. The factory could be explored through a UI which had different tools including the possibility of measuring distances and angles between desired points in the scan as seen in Figure 2.5. Measuring the diameter of a pipe or the width of a door through the LiDAR scans proved much faster than measuring them in real life. The scans are now being used for all new expansion and renovation phases by architects and other contractors.

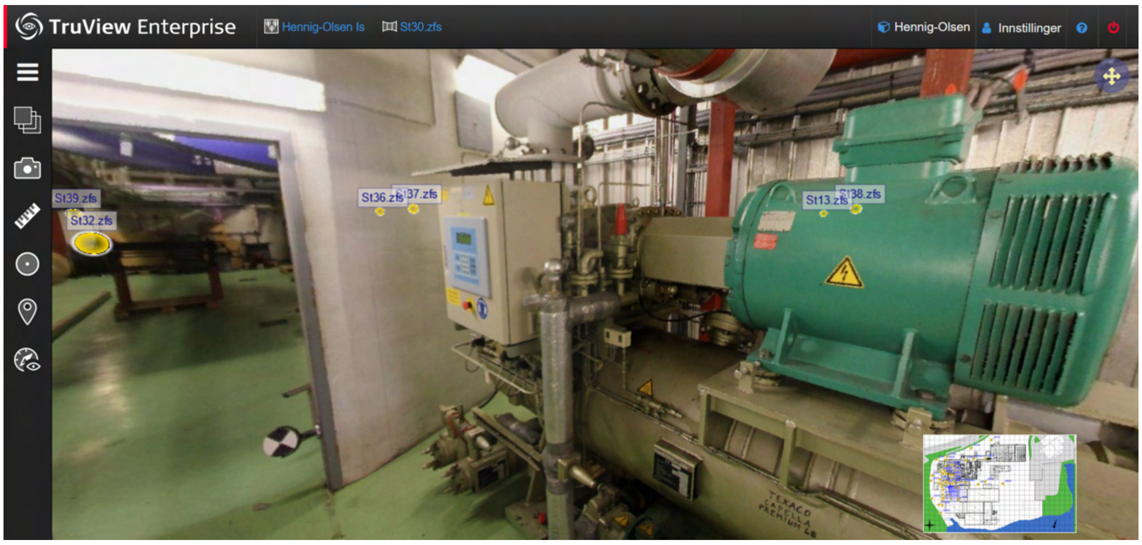


Figure 2.5: A screenshot from a LiDAR scan performed at Hennig-Olsen Is factory in Kristiansand displayed in the Trueview Enterprise platform.

After Tønnesen had performed his scans of the factory, Hennig-Olsen Is realized that their old documentation had several errors. For instance, one storage room where found to be 60cm narrower than what was stated in the original floor plan documentation. This shows the clear benefit of using LiDAR scans compared to virtual models in a designated computer program. A LiDAR scan gives an accurate within millimeters result about how the geometry of an object actually is instead of replicating the measured object manually in a computer program which is prone to human errors.

In 2020, Hennig-Olsen Is were laying new specialized pipes for transporting chocolate. In total, the new pipes were over 1km long. Before installing the new pipes, the chocolate pipes were drawn into the existing point cloud of the factory. The high resolution LiDAR scans were used to detect collisions the newly proposed chocolate pipes had with existing pipes in the factory. In addition to collision detection, the high resolution point cloud meant that parts for the pipes could be ordered with the exact measurements and partly preassembled, saving costs. This also significantly sped up the installation process, as there was less time spent on assembling the pipes.

Another application of the LiDAR scans taken by Tønnesen was to implement future production lines in their entirety in the point cloud before installing them in order to save costs, avoid collisions with existing material and speed up the installation process. In addition, the scans with the added production line could be explored in VR by production line workers to make sure the production line were optimized to best suit the existing building layout and production workflow. Tønnesen explored this on a project he did for GE Healthcare's department in Lindesnes. GE Healthcare is part of the american GE consern. GE Healthcares factory in Lindesnes is one of the worlds biggest producers of active substances for contrast agents. While working on a new production line at the Lindesnes factory, Tønnesen LiDAR scanned the area where the production line would be installed, added the proposed production line to the scan and then explored it in VR. This made it easy to visualize how the production line would be integrated in the factory. Virtual exploration of the LiDAR scans with the added production line could also potentially be used for educational purposes when training new production line workers in the future.

Tønnesen encountered several challenges working with LiDAR and VR technology. The challenges encountered were so limiting that Sweco today are no longer combining VR technology with LiDAR scans in their new projects. As a consulting firm, Sweco's goal is to give a customer the best possible solution for for the lowest possible price. Tønnesen states that at the moment, exploring LiDAR scans in VR on his customers projects is too expensive compared to the results a customer gets in return. Creating a 3D model of the object to be built in a 3D modeling program and presenting it to customers on a ordinary screen is cheaper, and according to Tønnesen gives almost as good results as exploring LiDAR scans in VR.

The main challenges and limitations Tønnesen encountered working with LiDAR scans and VR technology were:

- **Cost:** A professional LiDAR scanner is extremely expensive and requires significant expertise by the person operating it. These factors drives up the cost of LiDAR scanning, making them very expensive to conduct. Tønnesen concludes that at the moment, exploring LiDAR scans in VR on a project is too expensive compared to traditional methods used by Sweco.
- **Data size:** One experienced downside of using such high resolution LiDAR scans was the enormous amount of data points produced. The amount of points in the scans results in long processing times as well as enormous file sizes which takes significant time to download. To combat these challenges, a user can only navigate through the scans by moving between stationary points in the scan, which reduces the amount of points rendered at any given time. However, this feature also limits a users mobility as he can not move around freely within the scans.
- **Software:** Tønnesen states that one of the main challenge of exploring LiDAR scans in VR was the lack of proper and intuitive software that could integrate VR in an existing scan. As a result, Sweco ended up developing their own plugin to Autodesk's naviswork to be able to explore a scan in VR.

In order for VR exploration of LiDAR scans to be a viable option in the future, Tønnesen states that it must become as cheap as existing methods used today. However, he believes that exploration of LiDAR scans in VR offers several benefits over existing methods as it gives the user a better spatial representation of the objects to be explored. Tønnesen believe that the technology is still in its early stages and that it will be more widely adopted as soon as good solutions to the challenges he encountered are in place.

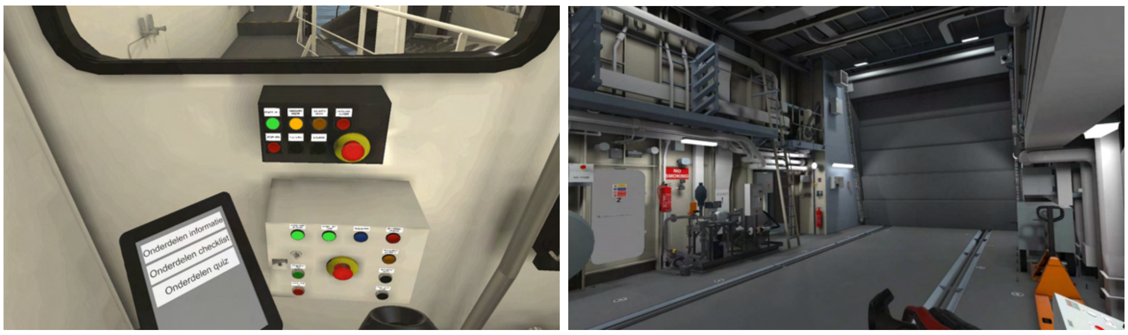
One significant future use case Tønnesen discusses is the ability for engineers and communal inspectors to quickly survey construction sites. Instead of requiring a communal inspector to travel to a construction site and conduct a survey before approving an application, he can simply put on his VR headset and inspect an accurate model of the building being applied for, placed in the exact position it is applied for.

2.3.2 Virtual reality in shipbuilding and remote cargo ship surveys

In addition to using VR technology in the process industry, there has been many proposed applications of VR in the maritime industry. These applications range from shipbuilding to vessel surveys. Damen Shipyards Group (Damen), a Dutch shipyard company, have worked on integrating VR training grounds for all ships they build. DNV, an international company within quality assurance and risk management for the maritime industry, has worked on how to best perform remote vessel surveys. A survey of a vessel can be conducted both physically or remote and is performed in order to investigate if a vessel adhere to specific standards and requirements. DNV ran a pilot project utilizing VR technology to conduct remote surveys. Underneath is an explanation on how VR technology is integrated in the two maritime industry projects, and how it helped the two companies.

Virtual reality in shipbuilding

Damen are actively working with VR/AR technology in the ship building and conversion arena. In 2018, Björn Mes, technical VR/AR specialist at Damen, investigated the use of VR for different training scenarios on vessels (Mes, 2018). In the project, Mes and his team built a virtual training ground aimed at running training scenarios for specific technical equipment on a vessel. Photos from the project can be seen in Figure 2.6a. In an interview conducted with Mes (section B.3), Mes explained that the project proved successful, and a pilot project was started soon after. In this project, an exact virtual replica of a 100 meter long ship was created, so that it could be inspected in VR. The goal of the project was to find out how large vessel models could be created and implemented in a VR environment. Both the training ground and the virtual replica of the ship was created by hand, which proved both time consuming and tedious. In the interview, Mes estimates that the process of creating an exact virtual model of the ship took around 3,500 hours. In the project, Mes and his team converted CAD models of the ship to 3D models supported by Unity. Using this method, they had to clean up all textures and simplify faces of the model in order to optimize it for the available VR technology at the time. This was to ensure that the application would be able to run smoothly on a VR headset.



(a) Virtual training for a specific technical equipment (Mes, 2018).

(b) Screenshot from the current Virtual Reality program developed by Damen Shipyards Group demonstrating a generated vessel model.

Figure 2.6: VR environments created by Damen Shipyards Group.

Even though the pilot project proved successful, Mes concluded that VR technology was not user-friendly enough to be a consumer ready project at that stage in 2018 (Mes, 2018). Damen concluded that the technology was promising and that they would start using full-fledged VR applications in two to five years. In an interview conducted with Björn Mes in 2021 (section B.3), Mes explains that Damen Naval today have a full-fledged VR application that Damen offer's as an optional package alongside all new vessels being built. The application is used for training vessel crew. A customer can choose from a wide variety of set or custom training scenarios including maintenance or equipment training. The training scenarios can be performed on an exact 3d model of the customers vessel which significantly improves the quality of the training. Simulated training platforms can be constructed far more cost-efficiently by creating a VR simulation in a virtual generated model of a vessel compared to building physical models that are commonly used by ship owners (Mes, 2018). In addition, these virtual scenarios can be accessed from anywhere, which means vessel crew can train for operations and familiarize themselves with the vessels layout before deployment. For some types of vessels like war ships there may only exist one vessel of a specific type which makes it unfeasible to perform training while it is operational. For such customers, having the ability to perform training virtually is extremely valuable.

After completing the the initial investigation and the pilot project, Mes wanted to reduce the time it took to create the virtual replicas of vessels. Since Damen Naval are building ships, they have all data describing the vessels they construct. Mes and his team realized that they didn't need to convert the vessel model from already generated CAD models of the vessel, but could instead generate a correct virtual replica of the vessel based on the vessel data Damen had available. As an example, Mes explained that what once took 2-3 weeks when converting and cleaning up a model now takes about a minute. However, Mes and his team still needs to mark and label objects in the virtual model manually.

With the currently used method, Damen Naval generate the 3D objects on a vessel by accessing a database using Python, and finding out what type of object it is. If the object is found to be a pipe, the XML file of the object is exported to the 3D modeling software Blender which generates the 3D model of the object based on its XML file. To convert difficult objects with complex geometry, Mes and his team use Pixyz, which is a plugin for Unity, to prepare and optimize the large CAD models. This is because they can't automate the generation of large 3D models easily. One example of a complex object is the main engine. If some of the object models are updated, Mes explains that Python keeps track of these changes and automatically exports a new object model. An example of a vessel environment automatically generated using the discussed method that utilizes vessel data can be seen in Figure 2.6b.

To optimize the VR application Damen uses to explore these vessel models, Mes explains that they keep the number of polygons in their models low. The level of detail in object textures is kept low in order to optimize performance. Mes and his team also divide ships into segments, use occlusion culling, and hide all objects that are not visible in a users view in order to boost performance. However, there are some edge cases that needs to be handled according to Mes. For instance, for long hallways from the back of a ship to the front, the methods used by Mes and his team will not work. Here, other measures and techniques must be utilized to render these large areas smoothly. Mes and his team also uses technology from the gaming industry to optimize their VR application. Some of these techniques include mipmaps, baking of light when building the application, keeping the special effects used to the minimum and avoiding any form of live lighting. This is to avoid frame rate problems, which can cause VR sickness, when exploring these large vessel models in VR according to Mes.

To avoid VR sickness, Mes explains that certain rules have to be obeyed. The rules Mes states are: *"Never control the camera of the player, . . . and keep the frame rate as high as possible. Make your models and textures optimized so that you can have at least 90 frames per second. . . . and don't make it an experience longer than 15-30 minutes."*.

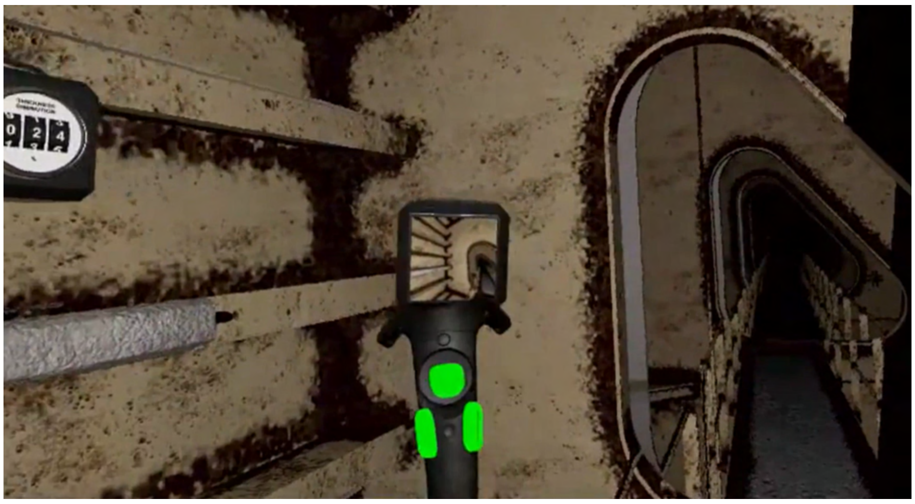
Before the construction of a new ship starts today, Damen transfers models of the proposed ship into a VR environment to finalize design parameters of it. This allows engineers to walk through a virtual model of the ship, and details such as sightlines, clearance and headroom will be understood more intuitively. By doing this, Damen are able to make decisions that would be difficult to imagine from a 2D or 3D drawing. As a result, this is found to make construction far more efficient (Mes, 2018). Another experienced benefit of using VR technology was how easy it was to collaborate in VR. Multiple users could simultaneously access the simulated VR environment, and the opportunities for interactions within a virtual ship was significant. Designers, engineers, project managers and end-users could see the ship from their perspective, and experiencing it at the same time. The physical location of the users is not a problem, as anyone can access the experience if they have the required equipment and an internet connection.

Virtual reality in remote cargo surveys

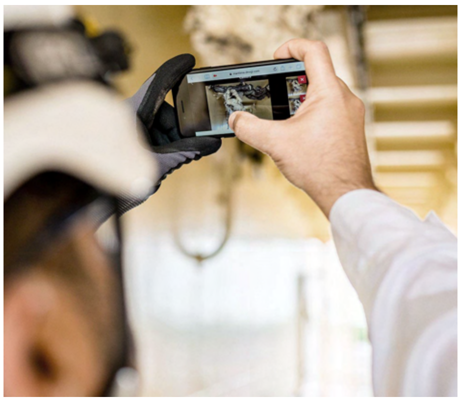
Another application of VR/AR technology in the maritime industry is for remote surveys of ships. During the COVID-19 pandemic VR technology has seen a growth in demand. Since governments around the world have applied constraints limiting peoples movement to stop the spread of the coronavirus, especially across boarders, there has been a higher demand for remote surveys (Hakirevic, 2020). DNV has delivered remote surveys of ships since 2018. During the pandemic the number of remote surveys DNV conducted rose by 33 per cent (DNV, 2020). According to Hakirevic, 2020, remote inspection devices could become commonplace in the future. A prediction is that this will either replace or assist the physical attendance of surveyors.

In 2018, the German office at DNV did a test project together with Corral Design on remote surveys using VR technology. Corral designed a virtual reality application with the purpose of allowing engineers to perform a safe and efficient remote surveys of cargo ships. A screenshot from a demonstration of the application can be seen in Figure 2.7a. The model of the ship was created using Autodesk Sketchbook Pro and Adobe Suite, and within the application there was options to take pictures, measure thickness, inspect corrosion, etc. The goal of the collaboration with Corral Design was to deliver a proof-of-concept prototype to show that VR has practical applications in the maritime engineering world.

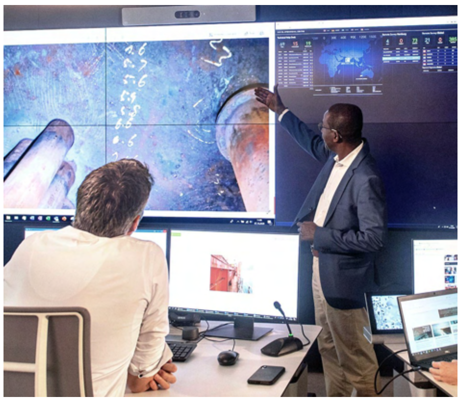
The prototype allowed users to move around a 3D representation of the vessel, and score and annotate images taken within the application. The user also had the option to refer to historical data from other vessels (Corral Design, 2018).



(a) Screenshot from a demonstration of the virtual remote survey application developed for HTC Vive by Corral Design (Corral Design, 2018).



(b) Workers taking documentation with their smartphone used for remote surveys (DNV, 2021).



(c) Operators using the footage captured in Figure 2.7b to discuss the condition of the ship (DNV, 2021).

Figure 2.7: Remote survey by DNV.

However, in a interview conducted with Stener Olav Stenersen (section B.4), Head of Services at the DNV GL Maritime Operational Centre in Høvik, DNV has not yet adopted the technology created for Corral's application. According to Stenersen it has not been a need for this type of virtual inspection, as they use different tools when conducting remote surveys. According to Senersen, starting to utilize this technology requires large investments in order to further develop the prototype project created by Corral Design.

Stenersen explains that DNV currently performs remote surveys by gaining access to documentation of the ship, work reports, pictures, and other material available and then discussing the material remotely as seen in Figure 2.7c. They also make the crew stream live video, showing certain parts of the ship as seen in Figure 2.7b. A common problem regarding this method is wifi-connectivity, as Stenersen says that having a stable internet connection at sea is a major problem. Certain parts of a vessel, e.g. the machine room, often lack internet connection as the signal gets blocked by the construction. However, Stenersen says that this is about to change, and this change has been accelerated by the recent COVID-19 pandemic. More ships are gaining stable internet connection, and several connection points are added to get better range. Another challenge is that many vessels does not have the necessary equipment on board. It is not given that a ship have a computer with a good enough webcamera or a smartphone available. Some vessels also contain flammable material, and as a consequence no electrical equipment is allowed on board.

According to Stenersen, conducting a survey remotely takes more time than a physical survey. This is because the crew has to be trained to properly handle a camera and set up streaming correctly. The crew also have to be guided around the ship, to allow the surveyors to see what they require. It is also difficult for a surveyor to gain a full overview of a vessel from a video. Good communication is therefore essential when performing remote surveys using video link, and this can sometimes be an issue.

Because of these difficulties, DNV has decided that the periodic yearly surveys they perform are to be performed physical. However, DNV want to increase the amount of remote surveys on the occasional surveys they perform. An occasional survey are typically shorter and less complex than yearly surveys. DNV's long term goal is that all occasional surveys are to be performed remotely in the future.

Stenersen believe that remote surveys can serve as temporally permissions to continue sailing when a problem occurs, provided that the vessel will sail to a harbour where a physical survey can be conducted. So instead of having to travel far and delay a boat even further, a temporally permission to sail can be applied by conducting a remote survey.

Currently, remote surveys at DNV are performed with a smartphone and a 360° camera by the ship crew if available. However, in the future Stenersen believe that specialized glasses with integrated cameras used in remote surveys will be used. This is because using glasses to capture video instead of using smartphones frees the hands of the crew member, so it becomes easier for the crew member to maneuver around the ship.

2.4 Game Engine

A game engine provides the software framework to build and create video games. The more advanced game engines are responsible for rendering graphics, collision detection, physics, sound, scripting, artificial intelligence, and networking. Two of the most popular game engines are Unreal Engine and Unity. Both these engines have its advantages and disadvantages, and a discussion regarding which game engine best serves the project can be seen in section 3.3.

2.4.1 Unreal Engine

Unreal Engine is a game development engine owned by Epic Games. By using the game engine, a wide variaty of 2D, 3D, and VR applications can be made. Unreal Engine 4 has support for more than 15 different platforms. In regard to VR development, Unreal Engine provides OpenXR, which is a toolkit that provide access to XR platforms and devices such as Oculus, SteamVR, and Windows Mixed Reality (Unreal Engine, 2021a). Extended Reality (XR) is an umbrella term encapsulating Augmented Reality (AR), Virtual Reality(VR) and Mixed Reality (MR). The engine is best known for its impressive graphical and lightning capabilities, and provide very realistic scenes. It also provides libraries of materials, objects, and characters which can help build game scenes. To program an Unreal engine projects game logic, C++ is used as a programming language, but a visual editor called *Blueprints* can also be used. Using the visual editor, no coding experience is required (Computer Hope, 2019).

Infrastructure

The important factors of Unreal engine's infrastructure consist of its asset store, documentation, and its community. The current number of assets in the Unreal Engine marketplace is around 10,000 (Circuit Stream, 2021). This marketplace allows users to purchase pre-made 3D models, objects, environments and much more.

Unreal Engine provide extensive documentation in relation to how to develop XR applications (Unreal Engine, 2021b). However, to resolve bugs or any problems encountered during an implementation, the engine's community can help solve the problem if the documentation is not found to be adequate. The Unreal Engine's forum has around 12,000 topics on C++ programming, and 4,600 threads on VR and AR development. The community is divided between non-programmers who use visual scripting and programmers using C++. The majority of the community consist of non-programmers as there are twice the number of discussions of visual based scripting compared to C++ discussions.

Rendering

The Unreal Engine was one the first game engines to support DirectX Ray tracing (DXR). Ray tracing is a technique that makes light in virtual environments behave like it does the physical world. It works by simulating actual light rays, using an algorithm to trace the path that a physical beam of light would take. By utilizing DXR, ray tracing only needs to be applied to the passes that need it. Traditional rendering methods like rasterization is used to render 3D objects. Ray tracing is computationally expensive, however using this feature ensures more accurately modelling of how light interact with the environment, creating more realistic scenes (Microsoft, 2018). Using DirectX Raytracing ensures both render quality and rendering speeds are met, and for every new update to Unreal Engine, Epic Games works to improve the engine's rendering speed and quality (Circuit Stream, 2021).

Networking

Unreal Engine provides synchronization of data and procedure calls between clients and servers by a process called Replication. This system provides a high-level abstraction along with low-level customization for simplifying implementation of networked functionality in applications where multiple simultaneous users interact.

Unreal Engine uses a client-server model, where one computer in the network acts as a server and hosts the application. The other computers who connect to the server is the clients. This is an integral part of the engine, and is easy to set up. For other clients to connect to a host, the public IP address of the host need to be known (Unreal Engine, 2021c).

Another option is to use a third-party company which can provide dedicated servers for networking. One of the options is Photon which provide rooms, and in-room communication for clients through servers maintained by Exit Games. By utilizing Exit Games servers, the servers are load balanced which gives a layer of scalability that enables the application to run on multiple servers if the incoming traffic requires it. In addition to providing dedicated servers managed by Exit Games, developers can also set up their own and connect them to Exit Games services. This gives developers full control over the server logic.

2.4.2 Unity

Unity is a cross-platform game engine developed by Unity Technologies. The engine was originally released as a MacOS exclusive in 2005 and has since evolved into supporting more than 25 different platforms. In regard to VR development, Unity support Oculus and Windows Mixed VR, but also support third-party platforms such as Google VR and Open VR. All of these platforms can be added to a VR application through Unity's XR Interaction Toolkit which enables cross platform VR interactivity. As of 2020, Unity is one of the most popular game engines available (Peckham, 2019). Unity offers powerful frameworks for developing applications in both 2D and 3D, as well as VR and AR. In recent years Unity has extended into other industries than game development (Unity Technologies, 2020). The engine is currently being used by the automotive industry, for engineering and construction, and in visual effects and animation for film production.

Infrastructure

The Unity Engine has its own asset store which is available directly through the engine. In the asset store, game developers can create and sell produced tools and assets to other developers. A wide variety of assets are available, including textures, 3D models, music, tutorials and Editor extensions. The current number of assets in Unity asset store is about 67 000. The majority of assets are 3D objects, which account for about 38 000 assets. The Unity asset store has a huge amount of available tools for everything from animation to AI which can be included in a Unity project.

Unity has an extensive documentation in relation to how to build XR applications on the platform (Unity, 2021a). However, Unity also relies on its community which is bigger than Unreal Engine's community. Unity's forum have 128 000 threads on scripting and 6 100 topics on VR development, compared to Unreal Engines 12 000 and 4 600 respectively. Unity developers use C# as their primary development language, and is not divided into a community of programmers and non-programmers. Thus, Unity has a larger and more coherent community with more available resources than Unreal engine.

As Unity use C# as its programming language, another advantage becomes apparent. Around 90% of AR/VR development companies use C#, which result in a large available code-base and tested solutions in the development community (Circuit Stream, 2021).

Rendering

A render pipeline performs a series of operations that take the contents of a Unity scene and displays them on the screen. At a high level, these operations are culling, which is the a process that prevents Unity from performing calculations on objects that are completely hidden from the view, rendering the data in the scene and applying post-processing to the rendered view.

Until 2018, only one render pipeline was available in Unity. This render pipeline offered two rendering paths: forward rendering and deferred rendering. However, in January 2018 Unity unveiled the Scriptable Render Pipeline (SRP) which gave the option to customize the rendering loop via C# scripting. There er currently two pre-built SRPs: High-Definition Render Pipeline (HDRP) and Universal Render Pipeline (URP). HDRP is a hybrid between deferred/forward rendering and tile/cluster rendering. The main advantage of this method is that it offers faster processing of lighting and reduce the bandwidth consumption compared to deferred rendering. It is a render pipeline that target high-end hardware like PC's and gaming consoles. The pipeline is designed to be used for high fidelity games, graphics demos and architectural renderers where highly realistic graphics is the goal. URP is a fast single-forward renderer, and is designed for lower-end devices lacking support for compute shader technology, such as older smartphones and XR devices (Unity, 2021b). It is designed to achieve a more performant rendering, and preprocess the light before building the application. This makes URP save computational cost. URP has many of the same features as HDRP, but stripped down to make it more performant on all platforms. HDPR offers more complex lighting features like realtime global illumination, volumetric lighting and raytracing. However, URP allows developers to customize the pipeline to further optimize their application for desired platforms. These two rendering techniques renders immersive graphics which results in a more immersive digital experience, as they both provide realistic materials to cinematic lighting effects.

Networking

Networking in Unity is currently under development, as the previous solution called UNet is being deprecated. Currently, Unity is working on a new solution called Unity Mid level API (MLAPI). To integrate networking capabilities in Unity projects, there are currently two methods available: GameObjects networking, which utilize MLAPI, and ECS-based networking. MLAPI is a high level networking library built to abstract networking implementation to reduce time spent to on low level networking frameworks and protocols (Unity, 2021c).

In the Unity's asset store, other networking tools are available. One of these are Photon Unity Networking (PUN), who re-implements and enhances the features of Unity's built in networking (Photon, 2021a). Unlike Unity's built in networking, PUN connects to a dedicated server which provide the same functionality as presented in section 2.4.1.

To optimize the connection between users, the Photon Cloud is organized in separate regions. Each region is separate from all others. This means that if a remote team will connect to the same environment, the team must connect to the same region.

Photon uses a master server that manages currently running rooms. The rooms are independent of each other and are identified by name. This means that users can only communicate with users that are in the same room as themselves. If a master client disconnects from a room, Photon automatically chooses another client to become the master (Photon, 2021b) of that room.

To synchronize interactable objects located in a networked room across the server, a PhotonView component needs to be added to the object. When a PhotonView component is added to a Unity object, data is sent over the internet describing the position, rotation and scale of the object to other clients connected to the same room.

Core concepts

Unity features a scripting API in C# as well as a drag and drop user interface. The scripting API features a wide variety of extended methods and Unity specific libraries on top of the native libraries in C#. New scripts can be created and attached to GameObjects from the Unity editor window seen in Figure 2.8. GameObjects are the base class for all entities in a Unity scene. A GameObject is a container for different attached components, which are extended classes that add functionality to the GameObject. Every GameObject has a transform component attached, which holds the position, rotation and scale of the object. Unity has a wide variety of built in components that can be attached to GameObjects. Some examples of built in components are mesh rendering, physics interactions and collisions. The Unity scripting API make it possible to create custom components, which gives developers great flexibility.

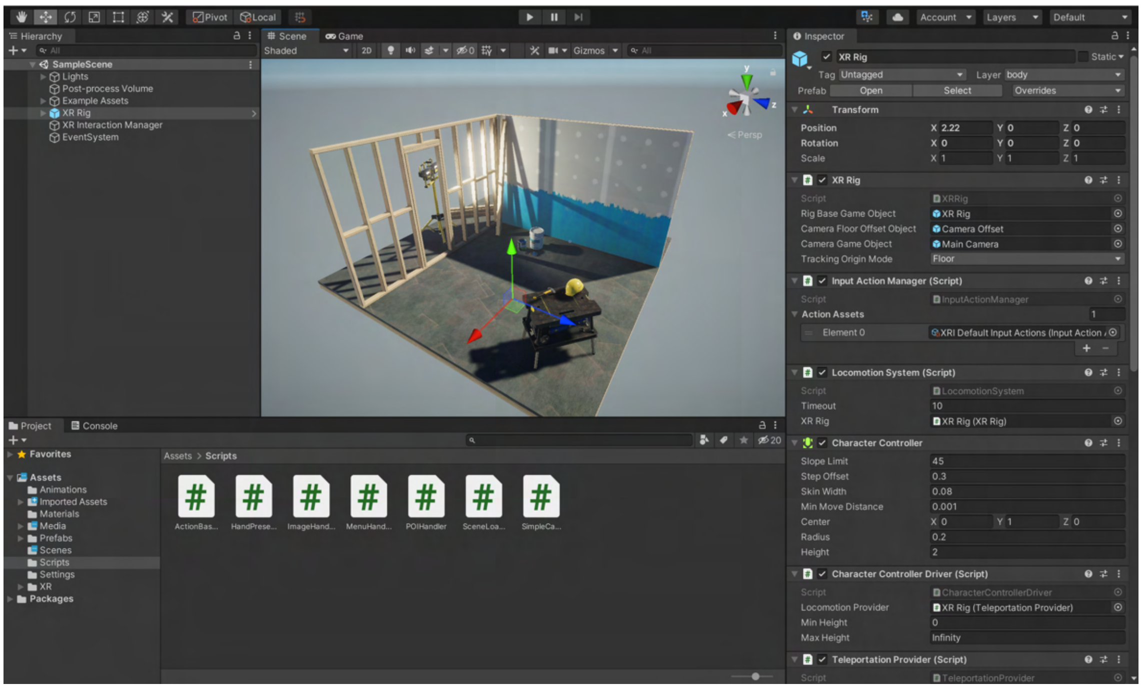
Editor Window

The Unity default editor window in Figure 2.8 consists of several panels displaying different data. The inspector window to the far right displays all components attached to the selected GameObject. All GameObjects in the current scene are shown in the hierarchy panel to the upper left. The scene panel in the middle allows developers to view the scene from different angles. The project panel containing all assets and packages for the project can be seen below the scene panel. Depending on which tasks are being performed by the developer, different panels may appear. It is possible to customize the editor windows layout to optimize it for specific tasks.

Monobehaviour class

MonoBehaviour is the base class every Unity specific script derives by default. When a C# script is created in Unity, it automatically inherits from MonoBehaviour, which provides a default template script. The MonoBehaviour class provides framework which allows a script to be attached to a GameObject in the editor. It also provides event related methods such as Start and Update. The Start() method is used for initialization, and is called once on the first frame a script is enabled. Update() is called every frame a script is active if MonoBehaviour is enabled.

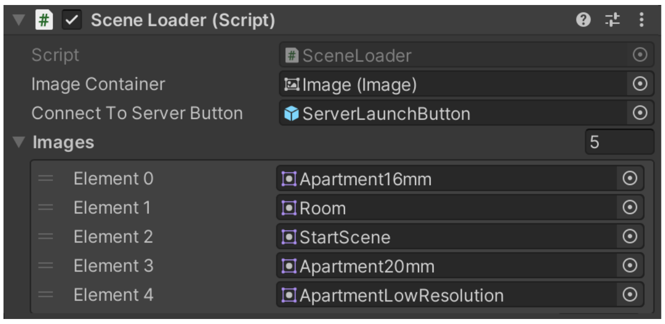
The MonoBehaviour class enables managing of coroutines, which are a way to write asynchronous code. Coroutines are functions that can be partially executed after certain actions are completed, such as timers and events. Other code can execute while a coroutine is yielding.



```
XR Rig                                   Static
Tag Untagged              Layer body
Prefab   Open    Select   Overrides
Transform
Position        X 2.22   Y 0   Z 0
Rotation        X 0      Y 0   Z 0
Scale           X 1      Y 1   Z 1
XR Rig
Script                      XRRig
Rig Base Game Object        XR Rig
Camera Floor Offset Object  Camera Offset
Camera Game Object          Main Camera
Tracking Origin Mode        Floor
Input Action Manager (Script)
Script                      InputActionManager
Action Assets                                  1
  Element 0                 XRI Default Input Actions (Input Action
Locomotion System (Script)
Script                      LocomotionSystem
Timeout                     10
XR Rig                      XR Rig (XR Rig)
Character Controller
Slope Limit                 45
Step Offset                 0.3
Skin Width                  0.08
Min Move Distance           0.001
Center                      X 0   Y 1   Z 0
Radius                      0.2
Height                      2
Character Controller Driver (Script)
Script                      CharacterControllerDriver
Locomotion Provider         XR Rig (Teleportation Provider)
Min Height                  0
Max Height                  Infinity
Teleportation Provider (Script)
Script                      TeleportationProvider
```

Figure 2.8: Overview of the Unity default editor window. The inspector panel is located to the right, the hierarchy to the left, the scene window is located in the middle and the project tab is placed at the bottom. All panels functionality are explained in section section 2.4.2. Screenshot from the Unity engine.

Public variables in classes derived from MonoBehaviour can be changed directly in the Unity inspector as seen in Figure 2.9. This allows developers to experiment and tune game variables without having to make changes in the script. It is also possible to create scripts in Unity that extend the C# base object class instead of MonoBehaviour. These scripts can still interact with other Unity scripts and vice versa.



```
Scene Loader (Script)
Script                      SceneLoader
Image Container             Image (Image)
Connect To Server Button    ServerLaunchButton
Images                                         5
  Element 0                 Apartment16mm
  Element 1                 Room
  Element 2                 StartScene
  Element 3                 Apartment20mm
  Element 4                 ApartmentLowResolution
```

```
public class SceneLoader : MonoBehaviour
{
    [SerializeField]
    private Image imageContainer;
    [SerializeField]
    private GameObject connectToServerButton;

    public List<Sprite> images;
```

Figure 2.9: Variables in a script that inherits from Monobehaviour and is made adjustable in the inspector. All public variables are automatically adjustable in the inspector in addition to private variables with the tag *[SerializeField]*.

Another useful method in the Monobehaviour class is Instantiate(). By creating a GameObject variable in the script and assigning a GameObject to it in the inspector, it is possible to access that exact GameObject through code. The Instantiate() method makes it possible to create and place desired GameObjects at runtime. Instantiate() takes three parameters; the GameObject to be instantiated, its world position and the GameObjects rotation.

XR Interaction Toolkit

The XR Interaction Toolkit package is a high-level, component-based, interaction system developed by Unity Technologies with the goal of standardizing and making XR development easier. It provides a framework that makes 3D and UI interactions available in XR from Unity input events. Before the *XR Interaction Toolkit* was released, a different set of frameworks were used to develop for each line of VR sets. The *XR Interaction Toolkit* provides a framework for cross-platform VR input as well as a cross-platform XR-rig consisting of a VR camera and VR controller tracking.

The XR Interaction Toolkit includes two types of XR Rig: Stationary and Room-Scale. The Stationary XR Rig uses the user's device as the tracking origin, allowing for three degrees of freedom as seen in Figure 2.10. This makes a good fit for projects where the user remains seated, and only make use of the head-mounted display's rotational movement. All devices capable of supporting XR will work on the Stationary Rig.

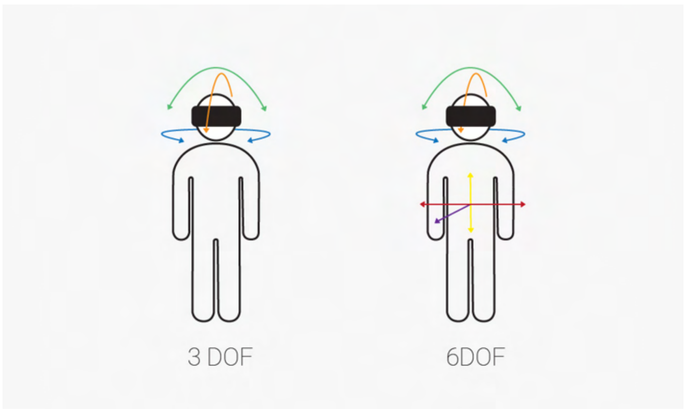


Figure 2.10: Three degrees of freedom (3DOF) compared to six degrees of freedom (6DOF) (Delight XR, 2021).

The Room-Scale XR Rig uses the floor of the user's physical space as the tracking origin, allowing for six degrees of freedom as seen in Figure 2.10. This means that the users rotation and positional movement is enabled, and allows the user to explore the virtual world at the scale of their physical environment. A headset capable of mapping a user's environment and tracking their motion within it is fit for the Room-Scale XR Rig (Unity, 2021d).

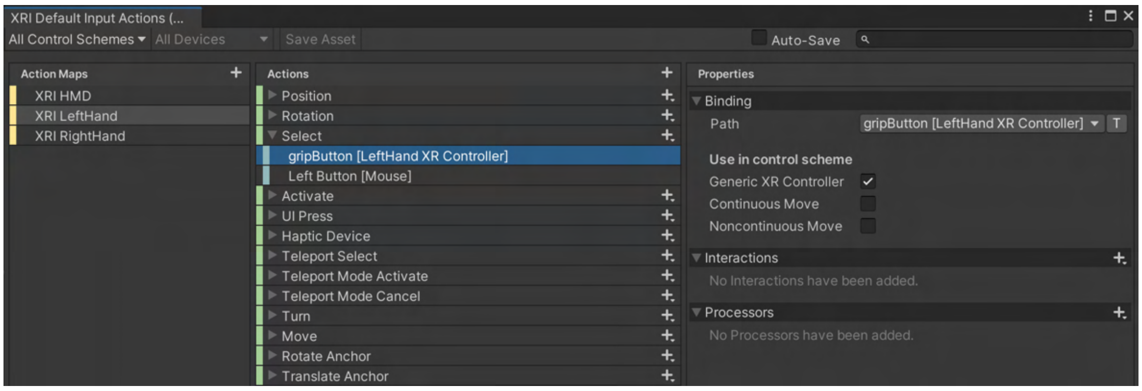
Device-based behaviour

There are currently two ways of reading input from controllers in Unity, device-based and action based behaviour. To read values from controls with device-based behaviour, the program has to listen to input from a specified input device. This is done through the *InputDevices.TryGetFeatureValue* function. As a result, different sets of code has to be written for reading input from different sets of controllers which results in redundantly written code.

Action-based behaviour

In contrast, action-based behaviour read input through defined actions that are bounded to one or more controls inputs. This makes it easier to allow cross-platform compatibility as an input code base is only written once. Another benefit is the use of the Action editor. Here, controls from all platforms can be assigned to specific actions by by binding them as seen in Figure 2.11a. For instance, the *Select* action can be binded to several input inputs from different devices. Still, both inputs from both controllers are listened to through the *Select* action. To listen to the inputs of a specific *type* from a script the following command can be used: *Select*.action.ReadValues<*type*>(). By utilizing action based behaviour, inputs from controllers, keyboards, joysticks and other devices can be bound to perform the same actions which removes the need for redundantly written code.

The different actions need to be enabled before they can react to input. When an application starts, the *OnEnable* function enables the desired action in order for it to react to its bound input. An action can also be disabled at any time using the *OnDisable* method. If desired, the Input Action Managers behavior can automatically enable and disable the actions in the *OnEnable* and *OnDisable* methods. The Input Action Manager script can be added to a GameObject in a Unity scene, which adds all Input Action Assets in the Action Assets list as seen in Figure 2.11b.



XRI Default Input Actions (...

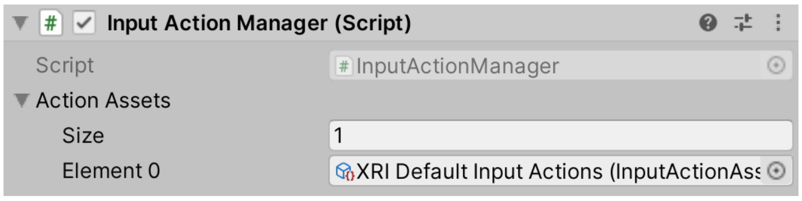
All Control Schemes | All Devices | Save Asset | Auto-Save

Action Maps:
- XRI HMD
- XRI LeftHand
- XRI RightHand

Actions:
- Position
- Rotation
- Select
  - gripButton [LeftHand XR Controller]
  - Left Button [Mouse]
- Activate
- UI Press
- Haptic Device
- Teleport Select
- Teleport Mode Activate
- Teleport Mode Cancel
- Turn
- Move
- Rotate Anchor
- Translate Anchor

Properties:
- Binding
  - Path: gripButton [LeftHand XR Controller]
- Use in control scheme
  - Generic XR Controller: ✓
  - Continuous Move: ☐
  - Noncontinuous Move: ☐
- Interactions: No Interactions have been added.
- Processors: No Processors have been added.

(a) The action editor allows bindings of different inputs to be added to different Actions. The path to the binding of the controller must be included. Here, *gripButton* refers to a specific button on a LeftHand XR Controller which is bounded to the `Select` action. The *Left Button* is also bounded to the same action



Input Action Manager (Script)

Script	InputActionManager
Action Assets	
Size	1
Element 0	XRI Default Input Actions (InputActionAss

(b) Input Action Manager script added to a GameObject. The actions created in Figure 2.11a is added to the Action Assets list. This will automatically enable and disable the actions created which allow responding to input from controllers.

Figure 2.11: Input Actions bindings are demonstrated in Figure 2.11a. These actions can be automatically enabled and disabled by acquiring the Input Action Manager as seen in Figure 2.11b.

2.5 Microsoft Azure

Formally released in 2010, Microsoft Azure is Microsoft's cloud computing service for building, testing, deploying, and managing applications and services through Microsoft-managed data centers. Microsoft Azure provides software, platform and infrastructure as a service and supports a wide variety of different programming languages, tools, and frameworks. Some of these services include computer services like virtual machines, data storage and management services, identity and security services, mixed reality, migration, AI, machine learning and blockchain tools. With this amount of services readily available in one place, an increasing number of companies have started partially or fully running their operations in Azure (Tracey, 2020). As a result of increasing digitization of existing operations, worldwide use and spending on cloud services like Azure are expected to increase significantly in the future.

By utilizing a feature called 'pipelines' in Azure, online code repositories can be connected to web apps in Azure. When code is pushed into the designated repository, the pipeline feature will automatically update the code base being run in the Azure web app if the repository code does not generate any errors or flags. Being able to automatically deploy code from repositories to web apps significantly speeds up development, as there is no time spent integrating newly developed code into an existing application.

2.6 Assuranceforeningen Gard

Figure 2.12: Gard's logo. (Gard, 2021a)

Gard, founded in 1907 in Arendal, where its headquarters is still located, is the largest protection and indemnety (P&I) insurer among the thirteen members of *the International Group of P&I Clubs*, which provide P&I liability cover for approximately 90% of the world's ocean-going tonnage (Gard, 2021a). Gard's clients include shipowners and operators, shipyards, companies involved in upstream oil and gas markets as well as windfarm operators. Today, Gard has 13 offices located around the world and employs around 500 people. In 2020, Gards yearly earned gross premium was 829 million US dollars. Gard's logo can be seen in Figure 2.12.

2.6.1 Protection and indemnity insurance

Protection and indemnity insurance, more commonly known as P&I insurance, is a form of mutual maritime insurance provided by a P&I club. Whereas a marine insurance company provides hull and machinery cover for shipowners, and cargo cover for cargo owners, a P&I club provides cover for open-ended risks that traditional insurers are reluctant to insure. These P&I covers can include risks for damage caused to cargo during carriage, risks of environmental damage such as oil spills and pollution, operational downtime risks and war related risks.

A P&I club is a mutual insurance association that provides risk pooling, information and legal representation for its members. Unlike a marine insurance company, which reports to its shareholders, a P&I club reports to its members. Traditionally, the assured part pays a premium to an underwriter for cover which lasts for a particular time. This time period could be for a travel, a week or a year. A P&I club member instead pays a *call*. This is a sum of money that is put into the club's pool. If, at the end of the year, there are still funds in the pool, each member will pay a reduced call the following year; but if the club has made a major payout (for instance a big accident or oil spillage) club members will immediately have to pay a further call to replenish the pool. Since a payout can be in the magnitude of billions of dollars, a single club member can not usually insure 100% of the amount that a shipowner wants assured. Therefore, the risk percentages are split between different club members in a bidding process.

To summarize, a P&I club is run as a non-profit co-operative and the insurance is financed by *calls*. Club members contribute to the club's common risk pool according to the pools rules. Only shipowners with acceptable reputations are allowed to join a P&I club and any P&I club member who incurs reckless or avoidable losses to the club may be asked to leave.

2.6.2 Vessel surveys

Before a shipowner is allowed to insure his ships at Gard, each ship has to have a required set of certificates and has to be manually inspected by surveyors to make sure the ships adhere to Gard's strict safety guidelines. As Gard is a is run as a non-profit co-operative, it is in Gard and all of its clients interest to minimize claims. In order to minimize future claims, Gard support their clients in all aspects of risk management, especially in prevention and handling of accidents and crises.

In a meeting with Gard's head of loss prevention Marius Schønberg, together with Bjarne Augestad and Per Haveland, two senior surveyors at Gard, Gards surveyor practice is discussed in further detail (section B.2).

In the meeting, Augestad and Haveland explains that Gard performs two types of surveys; condition surveys and entry surveys. Condition surveys can be performed as part of an ongoing campaign at Gard against specific types of vessels or if Gard suspects that a vessel does not adhere to Gards guidelines. If a vessel has filed for several claims that raise suspicion, a condition survey can also be performed. Gard is obliged to perform an entry survey when a shipowner wants to insure a vessel that has not been insured or inspected by Gard earlier. Both types of surveys checks a set type of requirements depending on the type of vessel to be inspected (Gard, 2021b). Having the forms online, creates transparency for both parts of the survey to be performed.

Before a survey is conducted, the surveyor conducts a pre-study of the vessel using both Gards internal and some external information. Augestad and Haveland explains that experience and intuition is two important factors in a good surveyor. For instance, if the vessel to be inspected is a bulk carrier ship that frequently departs from Brazil, an experienced surveyor would know that this vessel probably carries iron ore frequently. This means that the inspector should pay an extra close eye to the ships carrier tanks and its structural condition, since iron ore is a heavy cargo which put a lot of stress on the vessel's tanks.

Augestad explains that ships are in continuous operation and that repairs therefore often are conducted while the ship is operative. If for instance, a surveyor notices a disassembled auxiliary engine, this could either be a sign of neglect by the crew that should be reported, or a sign that the crew is following necessary precautions if it is part of an ongoing repair. A continuous discussion between the surveyor and crew members is therefore essential during an inspection. While conducting a survey, the surveyor will always be accompanied by a member of the vessel crew, usually the first mate. In addition to being able to discuss various findings, being accompanied by the first mate also creates transparency, because a surveyor always discloses his findings that will be reported to the first mate. This gives the crew member a chance to object and explain potential findings on the spot which creates a clear understanding for both parts about what to expect from the future filed report.

While conducting a survey, a surveyor will continuously take photos to use as documentation for the final report to be filed. Since an inspection is carried out in only one day, an inspector might overlook potential findings because of the large area to be inspected in the short amount of time. Gard employs surveyors from all over the world, and each surveyor measures quality slightly different. Good photographs from the inspection is therefore essential, as the photos becomes universal and objective information that can be further used in the report as documentation. Haveland explains that he numerous times has discovered findings from photographs that were missed during the physical inspection which proves the significance of good photographs.

Augestad explains that Gard insures around 12 000 large vessels, and many more if small vessels are included. Last year (2020), Gard performed around 200 surveys. Augestad estimates that only a minority of vessels are being run in the gray area. Combined with the shipowner and vessel record, Augestad estimates that an experienced surveyor will get a pretty good estimate whether a vessel is worth further investigation after a rough walkthrough of the ship.

One problem Augestad notes with documenting physical inspections, is that a vessel may have areas were photography is prohibited. Tank ships carrying gas and oil may have zones where electric equipment including cameras are prohibited because of explosion hazards. Augestad estimates that in some large tank vessels, these areas can be up to 20% of the total area of the vessel. A surveyor may also encounter situations where he wants to inspect areas which are off limits, for instance a tank that is full at the time of inspection. As Gard does not want to inflict financial losses on its members, a surveyor will have to wait until a vessel is unloaded if he wants to inspect a tank that is full or take photographs in an area that has photography restrictions.

During the covid-19 pandemic, performing physical inspections have been hard. In order to try to resolve an increasing amount of outstanding inspections, Gard has experimented with remote surveys. Here, the surveyor sat down with the shipowner and was presented photos from the vessel to be inspected. Gard has now concluded that this remote way of conducting surveys were not satisfactory and has stopped performing them. The remote surveys were not satisfactory because the surveyor could be presented biased information and photos from the shipowner that did not accurately reflect the current conditions at the vessel to be inspected. At the time of the meeting, all physical inspections were stopped because of the covid-19 pandemic. Augestad explains that Gard has had a good renewal of insurance policies in 2021 and that there also are a lot of new vessels that wants to sign up for insurance at Gard. As a result, Gard now has a high amount of outstanding surveys to be performed.

Chapter 3

Analysis and Design

In this chapter, an overview of the problem to be solved and its proposed application solution will be presented. Next, key equipment used for creating material to the application is discussed as well as the choice of game engine and render pipeline in section 3.2 and section 3.3 respectively. The means taken to avoid VR sickness is addressed in section 3.4. In the end, a high-level overview of the structure of the VR application is presented in section 3.5.

3.1 Specification

The task given by Gard is to create an application which allows for a virtual walk through of a vessel in VR. After conducting several interviews with employees at Gard and other actors in the market utilizing VR technology, it is concluded that the most useful use- case for a VR application is as a virtual sorting tool to decide which vessel need a physical survey. The proposed application will therefore be developed to help decide which vessels should and which should not be explored physically by surveyors. A VR application with this functionality could help reduce the amount of outstanding inspections Gard faces caused by the covid-19 pandemic.

3.1.1 Problem analysis

As briefly discussed in subsection 2.6.2, the covid-19 pandemic has greatly affected Gard's ability to perform physical surveys. Remote surveys have been attempted, but the current method of sitting down with the shipowner and being presented material has not been deemed good enough to adhere to Gard's strict guidelines. With a good policy renewal for 2021 and a significant amount of new vessels requesting insurance, Gard faces a record high amount of outstanding surveys to be performed.

Performing physical inspections require a surveyor to travel significant distances, usually by plane, which is both time consuming and expensive, especially when accommodation costs are included. As more and more companies are becoming more environmental conscious, limiting miles traveled by plane is one effective way of reducing a company's carbon footprint. In addition, physical surveys takes a significant amount of time, usually an entire day, which impacts the vessel's operational availability which could result in a financial loss for the shipowner. A surveyor may encounter challenges when conducting a survey, as discussed in subsection 2.6.2. During an inspection, a surveyor may want to inspect a tank that is full at the time of inspection or photograph some object in a zone where photography is restricted because of explosion hazards.

3.1.2 Requirements

As Gard's current way of conducting remote surveys was deemed not satisfactory, a new application is proposed. As discussed in section B.2, senior surveyors at Gard does not think a virtual inspection could ever fully replace a physical, as a physical inspection is a complex process. Bjarne Augestad, senior surveyor at Gard explained that after a rough walkthrough of a vessel and dialogue with the vessel's crew, an experienced surveyor could usually tell whether a vessel is worth further investigating or not. The proposed new application will therefore serve as a sorting tool to decide which vessels are worth investigating physically. Vessels that are deemed in satisfactory condition could potentially be approved for further operation. If the application proves successful as a sorting tool, it could significantly limit the amount of physical inspections needed to be carried out. This would help Gard to reduce the amount of outstanding inspections as well as to save costs spent on conducting physical inspections for both Gard and shipowners.

In order for the new proposed application to be successful, it should have the necessary tools to enable essential components of a physical inspection. The requirements of the proposed application is therefore:

- **High level of detail:** In order for a surveyor to accurately inspect a vessel, the virtual environment must represent a highly realistic model of the vessel. A surveyor must be able to notice small cracks and rust, which means the virtual model to be explored in the application must have a high level of detail.
- **Collaboration:** As discussed in subsection 2.6.2, a continuous discussion with a crew member during all parts of an inspection is essential because it gives the crew mate a chance to clarify potential findings. The application should therefore enable multiple people to walk through the vessel together and communicate via voice chat.
- **Documentation:** Good documentation in the form of photos is an important resource for a surveyor when writing the final inspection report and going through his findings for the day. In addition, photographs serve as universal information which is important since different surveyor might measure quality slightly different. The application should therefore allow for photographs to be taken by users within the virtual environment as documentation to be used in the survey report.
- **Cost saving:** Conducting a survey through the virtual environment in the proposed application should be cheaper than conducting a physical survey. This means that the equipment used to document the physical environment should be cheaper than paying for flight tickets and accommodation for a surveyor.
- **User friendly:** In order for the proposed application to be used by surveyors and vessel crew, it should be user friendly and intuitive. Both using the application and providing material to it should be a smooth experience and not require any technical knowledge.

3.1.3 Proposed solution

With the above requirements in mind, a VR application is proposed. This is to create an immersive experience, which mimics a physical survey. A VR application is thought to give a better result and be more interactive than existing methods used in the industry. The proposed application will enable exploration of 3d models of vessels combined with 360° imagery. By combining the 3d model of a vessel with imagery, it is easy for a surveyor to recognize where each image is taken aboard the vessel.

Generating material for the application

Since Gard does not have access to vessel metadata in the magnitude that Damen does (subsection 2.3.2), it is not possible to generate a vessel model automatically. This is because Damen, as a shipyard company, build the vessels they generate the models for, and therefore have access to all the metadata needed to generate a model of the ship. Gard on the other hand, as an insurance company, does not have access to this metadata.

Instead, a 3D model of a vessel has to be generated in a different way. Two methods for generating 3D models of environments are presented in section 2.2; generating models from scans taken by a LiDAR sensor and by using machine learning to generate 3D models from floor plans. Generating a 3D model from floor plans only generates a model of the walls and windows of a room. In addition, as experienced by Hennig-Olsen Is in subsection 2.3.1, old floor plans may contain inaccuracies. Generating a model from a LiDAR scan gives an accurate 3D model of an environments and the objects within it, and as proven by Hennig-Olsen Is, gives very accurate results. The 3D models will therefore be generated using a LiDAR sensor, as this method is thought to give the best results. The downside of this method is that the 3D model must be generated by the vessel crew and sent to a surveyor. This imposes an increased workload on the vessel crew. On the other hand the LiDAR sensor generate very accurate results, as discussed in subsection 2.2.1, and can be combined with textures. This result in accurate models that have a touch of realism because of the textures applied. Another benefit of using a LiDAR sensor is that a LiDAR model has a one-to-one relationship with the actual object being scanned with an accuracy equal to the resolution of the LiDAR scan.

Since a surveyor requires a high level of detail in order to properly survey a vessel, the LiDAR scans will be supplemented with 360° high resolution images. As seen in subsection 2.3.1, LiDAR scans have the possibility of generating very accurate results with almost picture perfect textures applied. However, these scans are performed with very expensive equipment by professionals and since the scans taken in the application are captured by the vessel crew, imperfect results are expected.

Features of the application

When walking around the geometrical model of the vessel in VR, the surveyor should be able to switch between the LiDAR model and 360° images of desired spots if he wishes to inspect an object in further detail. Using networking features, the surveyor should be able to connect his application to the internet and communicate with a crew member of the vessel to be inspected. Informational data points will be put within the model to be explored. For instance, if a surveyor is inspecting an engine, he could select the motors data point and get information about it, for instance what type of motor it is, when it was last serviced and it's damage history. To enable these data points, the proposed application must be able to connect to Gards databases in Azure, extract information from them and display them in VR.

Since the vessel model consists of both a LiDAR scan and 360° images, additional functionality will be implemented. The application should allow a surveyor to accurately measure distances within the geometric model of the vessel which can later be used by a surveyor as documentation. An important part of conducting surveys is gathering documentation, especially photographs, for the report to be filed. The proposed application should therefore have a virtual camera that can be used to gather documentation. Since the size of a high resolution LiDAR scan is large, it is desirable that surveyors will be able to download and manage the scans they need directly in VR. This will significantly reduce the file size of the proposed application.

3.2 Equipment and methods

Both a LiDAR scanner and a 360° camera will be used to create a realistic virtual model of a vessel. The LiDAR scans and 360° imagery will be taken by a crew member and sent to a surveyor that will explore the generated model in VR.

LiDAR Scanner

In order to acquire a 3D model that can be explored in VR, a scan performed by a LiDAR sensor is thought to give the best and most realistic results. There are a wide variety of LiDAR scanners available on the market today. Both handheld and stationary scanners are available with different resolutions. Since the LiDAR scanner used to gather material for the application will be used on ships which may be large in size and have a complex geometry, a handheld LiDAR scanner is thought to be the best option. However, many of the available handheld LiDAR scanners on the market today are expensive and difficult to operate.

In March 2020, Apple released its new iPad Pro series with a built in solid-state LiDAR as seen in Figure 3.1a. A LiDAR sensor was also included in the new iPhone 12 Pro series released on October 23, 2020. By this time, several new capabilities of the LiDAR sensor were added to the iOS operating system which demonstrates Apples commitment to the technology. The benefits of having a LiDAR in a phone or tablet includes better autofocus for cameras and more immersive AR. LiDAR sensors are believed to be a mainstream feature of phones and tablets in the future, making 3D scanning of rooms and buildings easily accessible. It is likely that LiDAR sensors in future devices will be of even higher quality than the ones available in devices today. In addition, it is likely that algorithms used to process a LiDAR point cloud, generating its 3D shape and texturing it will continue improving, resulting in even more realistic results in the future. By gathering material for the application with a LiDAR sensor already available in current or future phones, the cost for Gard of making sure each vessel has access to a LiDAR scanner could potentially be zero, since it is likely that at least one crew member has a high end mobile telephone. In addition to it's low cost, there already exists multiple apps on the App Store with a user friendly interface that allows users to scan and create 3D models and adjust scan parameters. This makes 3D scanning easily accessible. As a result, utilizing LiDAR sensors in current and future mobile devices to take scans used in the proposed application is regarded as the best option, as it gives a realistic result with a user friendly interface at a low cost.

360 camera

As the resolution of a LiDAR scan increases, the amounts of polygons in the generated 3D model increases accordingly. A model from a high resolution LiDAR scan will therefore require more computational power to render inside a VR headset and take up more space compared to a scan at a lower resolution. In order to make sure the proposed application runs smoothly on VR headsets, LiDAR scans will be done at a lower resolution than what is possible in the current sensors utilized. To supplement the LiDAR scans and give surveyors the level of detail they require, a high resolution 360° camera will be used. Rendering a high resolution 360° image is computationally cheaper than increasing the resolution of the LiDAR scan. Newer 360° cameras offer high resolution images at a reasonable cost. By utilizing both LiDAR scans and 360° imagery, it is believed that one can achieve a better result as the features from both technologies will complement each other.

Capturing 360° images for the proposed application will be achieved with the GoPro Max as it is believed to be the best option. The camera can be seen in Figure 3.1b. The GoPro Max has two 180° lenses on both sides of the camera as seen in Figure 3.1b, and uses an algorithm to stitch the two photos together into a complete 360° degree view. Since the GoPro Max is an action camera, it has a robust build and is waterproof, which is essential properties for a camera being used by non-technical crew aboard a vessel. The GoPro Max has the ability to capture both photos and videos at 6K resolution, which is more than enough to capture detailed images. It is easily controlled via its touch screen or via an app on the phone, which makes it easier to use for first time users.

When capturing material for a survey, a crew member could simply put the camera on their head, turn on record, and walk through the vessel. The surveyor can then later extract the photos they need from the video. Alternatively, a crew member can walk through each room of the vessel and manually capture a photo with the camera. If the captured material does not adhere to the surveyor's strict standards, he could request new material of desired objects or rooms until he is satisfied.

The GoPro max is priced between 400-500 USD, which is significantly cheaper than the expenses of flight tickets and accommodation required for performing a physical inspection. Assuming a crew member already has a high end mobile phone, the GoPro Max is Gard's only required expense in order to conduct a remote survey. By sending the GoPro max to a shipowner and utilizing the LiDAR sensor on a crew members phone, the crew has all the equipment they need in order to gather material for a highly detailed realistic virtual model of the vessel. This material can then be sent directly to the surveyor that can inspect the material in VR.

(a) iPad Pro (2020) with LiDAR scanner to the right of the dual-camera system (Apple, 2020).

(b) GoPro Max 360° camera (GoPro, 2019).

Figure 3.1: The modeling tools used to gather material for the application.

VR headset

Once a surveyor has received sufficient material of satisfactory detail, he can explore it in VR. As discussed in subsection 2.1.1, there are multiple types of VR displays available. A head mounted display type is deemed as the best option as they are significantly cheaper and more accessible than stationary displays.

The proposed application will be developed using toolkits that allow for cross-platform VR development. Even though the proposed application will be developed to work on other head mounted VR headsets, the preferred head mounted display for testing and developing the application will be the Oculus Quest 2 seen in Figure 3.2. The Oculus Quest 2 offers a fast and responsive VR experience at a reasonable price. In addition, as discussed in subsection 2.1.1 and by Bjorn Mes (section B.3), the Oculus Quest 2 is capable of running applications both internally and externally. This means that the Quest 2 is able to run without being connected to a an external computer. If increased computational power is desired, the Quest 2 can be connected to an external computer via a link cable and run applications from the computer.

Being able to choose whether to run an application internally or externally gives surveyors great flexibility. After a crew member has captured a high resolution LiDAR scan of a vessel, it is possible to run a simplification algorithm on the captured 3D model inside a scanner app to reduce its polygon count. When the crew member send both the original and the simplified 3D model to the surveyor, the surveyor can choose which model he wants to explore based on his current hardware situation. If the surveyor does not have a powerful computer next to him, he can explore the model with a reduced polygon count internally in his headset and still have a high frame rate. However, if he has a powerful computer available, he can take advantage of its increased computational resources and explore the high resolution model running the application externally from the computer.

Figure 3.2: The Oculus Quest 2 (Oculus, 2020).

3.3 Game engine and render pipeline

In order to be able to create a networked 3D VR-application in a few months, a game engine is required to speed up the development process. A game engine provides an abstraction layer which abstract away the low-level parts of an application. This includes low-level parts such as rendering and physics. This abstraction layer also enables faster prototyping, and since only about 4 months are available to develop a full-fledged VR application, a game engine is needed.

As the proposed VR-application will include features such as networking, voice chat, graphics rendering, and input from different devices there are many requirements to the game engine to be used. As the game engine will have such an integral part of the application development, it is important to choose a game engine that makes the best fit.

The following will be the requirements the game engine has to satisfy in order to develop the application successfully:

- **Virtual Reality:** An important part of the project is to be able to implement virtual reality specific features. Tools to enable virtual reality and input from different virtual reality headsets should be enabled by the game engine. These tools should be easy to use and allow a wide range of functionality.
- **3D Graphics:** The game engine must provide rendering of graphics and provide a realistic result. 3D graphics is essential to provide an immersive experience in VR. It is also important that the rendering is optimized to keep the refresh rate as high as possible to avoid VR sickness.
- **Networking:** The game engine should provide the option to connect to servers, so that multiple users can interact in the same environment. Ideally, the game engine should provide an abstraction level, so that low-level networking implementations is already provided and servers are available for connection.
- **Voice Communication:** The game engine should provide voice communication tools in order to facilitate collaboration with other users using the application while connected to its online services. Voice recordings should also be provided as a way for surveyors to make a note of what they see within the virtual environment.
- **Object Interaction:** The game engine should provide object interaction to make the application as interactive and realistic as possible.
- **Animation:** To enable interaction, the game engine must provide tools for animation.
- **User-friendliness:** Since there is only a scope of around 4 months to develop the VR-application the game engine should be both easy to learn and use.
- **Well documented:** The game engine should be well documented, so when a problem is encountered it is possible to read through both documentation and community forums to solve the problem.

Looking at the list above, both Unity and Unreal Engine satisfy the requirements to a certain degree. Both engines perform equally well on some of the requirements, however at others they differ.

Comparing subsection 2.4.1 against subsection 2.4.2 it is apparent that both game engines offer functionality regarding *virtual reality.* Also, a wide range of VR platforms are supported on both game engines. Using either XR Interaction Toolkit from Unity, or OpenXR from Unreal Engine provides access to a wide range of VR platforms and devices. In addition, both game engines also provide *object interaction* and *animation* tools.

With regard to *networking* capabilities, both game engines provide good options to choose from. A third-party company named Photon has also developed solutions to networking on both platforms. Servers are thus hosted by a third-party company, which is an advantage if many users connect to the application simultaneously since Photon offers on demand server scaling. Using Photons network solution also easily enables *voice communication.*

Another aspect considered is *3D graphics.* Traditionally, Unreal Engine has been superior compared to Unity for rendering high end graphics. However recently Unity has evolved its graphical capabilities, and today the same graphical results can be achieved with both Unity and Unreal engine (Circuit Stream, 2021). An advantage Unity possesses is the option to change the rendering pipeline to an optimized pipeline to run inside lower-end devices such as VR headsets.

When it comes to *user friendliness*, a big difference between the two engines is the programming language used for scripting. To create and destroy objects in C++, a developer needs to allocate and deallocate memory. In C# this is done automatically. Another advantage of using C# is that it is a much more protected language as it is not as close to the hardware compared to C++ where pointers can be used to access specific memory locations. However, this makes C++ faster compared to C# and applications built with C++ has smaller file sizes than those built with C#. This is because C# has a lot of overhead and includes libraries before compiling code, while C++ is much more lightweight. Compared to C++, C# gives clearer compiler errors and warning messages. In conclusion, C++ used in Unreal Engine is considered harder to work with compared to C# used in the Unity engine (Upwork, 2019).

As seen in subsection 2.4.1 and subsection 2.4.2, Unity have a bigger community compared to Unreal Engine. Where Unreal Engine has 12,000 topics on C++ programming and 4,600 threads about XR development, Unity has 128,000 topics on C# programming and 6,100 threads about VR development. As a consequence, it is more likely to encounter forum posts where developers have had similar problems and solved them, using Unity.

To conclude, Unity seems like a more user friendly alternative. The engine has an active community with many online resources available in addition to the engines documentation. Unity offer great tools to enable networking as well as voice communication, have an optimized rendering pipeline which ensures that the application runs smoothly on VR headsets, and has a toolkit which provides access to a wide range of VR platforms and devices. Since the application is being developed in a relative short amount of time, user friendliness is important. It is concluded that C# is preferred over C++ because of its simpler syntax and more intuitive and user friendly debug functionalities. As a result, C# will be faster to learn and utilize compared to C++. The Unity game engine is therefore chosen to develop the proposed application.

Render Pipeline

The Unity Game engine has several render pipelines available as discussed in section 2.4.2. Since realistic graphics and a high level of detail in the application is desirable, it is reasonable to believe that the High Definition Render Pipeline (HDPR) would be the best choice. However, since the application is being developed for cross platform VR headsets, standalone VR headsets like the Oculus Quest 2 has to be able to run the application internally at a high frame rate in order to avoid VR sickness. In addition, many modern VR headsets has one screen per eye, which means that every frame has to be drawn twice which can be computationally expensive. The Universal Render Pipeline (URP) is therefore considered the best option and will be used when developing the application since it will ensure a more efficient rendering which will result in higher frame rates on all VR headsets. The application will not be using computationally expensive visual effects which is best utilized with HDRP. It is therefore concluded that some minor graphical quality can be sacrificed in order to ensure a smooth experience for all users of the application regardless of platform by using the URP.

Input system

As Unity offer two types of input systems seen in section 2.4.2, the most convenient method has to be chosen. When using the device-based behaviour, string names referencing the input controller has to be manually set in the code in order to listen for input. This will cause problems, as string references can suffer from misspelling. The action-based controller on the other hand, has actions bounded to the controllers, and thus no need for string references. As the actions also can be bounded to several types of controllers, cross-platform input is easier to set up and the code becomes more organized when using action based input. This is because a developer does not need to listen to input from several controllers, but rather listen for an action that is bounded to several controllers. The action-based controller is thus chosen as the preferred input system.

3.4 Avoiding VR sickness

Since the proposed application is being developed for VR devices, measures must be taken in order to minimize VR sickness. As discussed in subsection 2.1.4, VR sickness occurs when a users brain receives conflicting signals about movement in the virtual environment with respect to the users physical movement. VR sickness can be reduced by following certain guidelines as discussed by Björn Mes, technical VR specialist at Damen Shipyard (section B.3). The following guidelines will be followed when developing the proposed VR application in order to minimize VR sickness:

- **Camera control:** Controlling a users camera immediately sends conflicting signals about movement to a users brain. The users camera will therefore only move when the user wants it to. In order to further minimize motion sickness, users should be able to choose how they want to turn in the virtual environment, and at what rate. When walking through the virtual environment, the camera should not wobble to create a realistic walking effect or replicate swinging movements from a vessel experiencing incoming waves, but instead float at a constant height. Teleportation features should also be added in order to allow users to quickly move around the virtual environment and reduce movement which may cause VR sickness.
- **Frame rate:** In order to minimize VR sickness, the frame rate must be kept high in the proposed application. If the frame rate is low, the lag between frame rates will cause VR sickness as the brain will perceive the time between frames as the player standing still, when in reality he could be moving. In order to make sure that the frame rate is kept high when the application is being run both internally and externally, a virtual environment will have two different versions of itself. One version with a higher quality LiDAR scan and the other with a lower resolution LiDAR scan. The smaller the size of the LiDAR scan is, less computation is required to render the frame which results in a higher frame rate. Utilizing URP to develop the application will make it more efficient to run which also helps in maintaining a high frame rate.
- **Length of the experience:** As prolonged use of VR may cause VR sickness, the proposed VR application should strive to have an efficient interface to reduce the time needed to get to the objects to be inspected. An efficient interface will be achieved by splitting parts of a vessel into different scans which can all be individually accessible via a user interface. If, for instance a surveyor wants to inspect the engine room, he can access it directly from a user interface instead of manually walking through the vessel model to get there. Once a surveyor has entered his desired scan, teleportation movement will allow the surveyor to quickly move around the scan, reducing the time spent in VR which in turn reduces VR sickness.

3.5 Structure of the VR application

In order to achieve an efficient and user friendly interface, each virtual environment will be put into individual Unity scenes. A Unity scene will consist of a 3D model generated from the LiDAR scan of a scanned area, it's 360 images and other assets including points of interest for that area. By interacting with a user interface, a user should be able to jump between the different scanned areas by switching between Unity scenes. For instance, a vessel could be an individual virtual environment, but if the vessel is large enough, areas of the vessel will be split into separate Unity scenes in order to minimize scene size and thus maintain a high frame rate. For a large vessel, important areas such as the deck, bridge and engine room may be split into separate Unity scenes.

The proposed VR application should have networking features. These should be activated or deactivated from a user interface. When activated, the application should connect to a server to allow multiplayer functionality. In order for users to be able to follow each other between different scenes, scene IDs will be used when communicating with the online server. If two or more users are connected to the server and have the same scene ID, they should be able to see and hear each other in the virtual environment. If the headset is connected to a WiFi, the application should automatically download information for all points of interest (POIs) in a scene based on the vessels ID when a scene is being initialized.

The simplified structure of the proposed VR application can be seen in Figure 3.3.

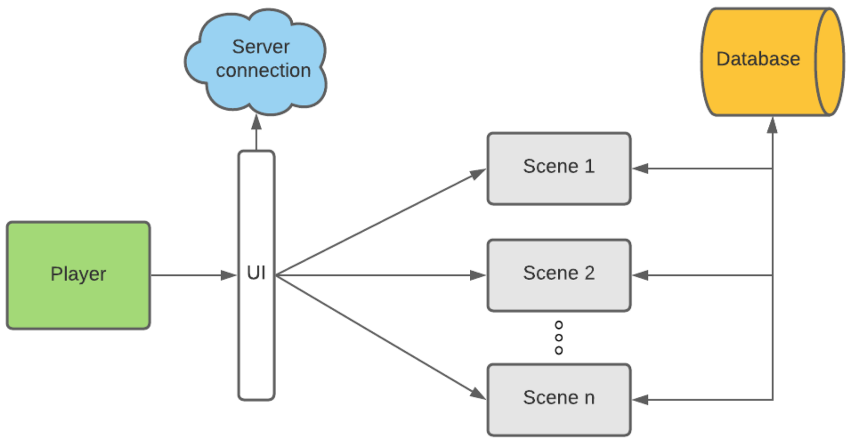


Figure 3.3: The simplified structure of the proposed VR application with n scenes.

Since the application is being developed in Unity with the Universal Render Pipeline (URP), all lights will be baked when building the application. This means that all light calculations are already completed when the application is started, resulting in low loading times when initializing the application. Because the Oculus Quest 2 is an android device, the built apk application file can be downloaded or sent directly to users who can run it immediately.

Chapter 4

Implementation

This chapter gives an in-depth overview of the implementation of the proposed VR application developed to create a platform for conducting remote surveys. To develop the application, the Unity Game engine was chosen as it had the necessary tools to implement the desired VR functionality as discussed in section 3.3. Key aspects of the methology used will also be presented. The overview will be given on a high level basis and assumes understanding of the C# language, but general programming knowledge will suffice.

4.1 Project setup

When creating a new Unity project, the first choice a developer makes is what type of project to create. In Unity, there are multiple template projects available for different kind of projects. Each template has their own set of pre-installed packages. One of the available templates utilizes the Universal Render Pipeline (URP) which includes a sample scene with some premade assets. This template will be used for setting up a URP project and serve as the application's template. The sample scene included in the URP template can be seen in Figure 4.1. It will be used as a testing ground while developing the different application features.

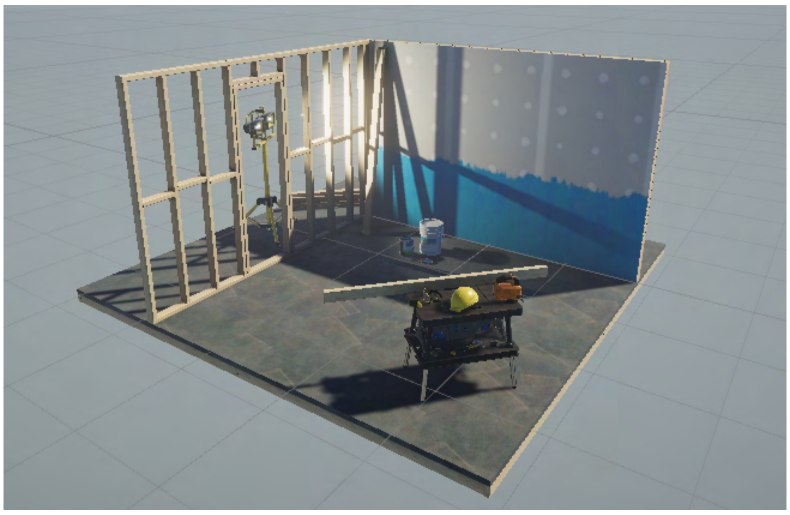


Figure 4.1: The integrated sample scene in a URP Unity project.

Once a URP project is set up, the necessary packages that are not a part of the template can be installed. There are two packages which are essential when developing a VR application. These are the *XR Plugin Management* package and the *XR Interaction Toolkit* package. The *XR Plugin Management* package provides management for XR plug-ins, and can be installed by going into

Edit > **Project Setting** > **XR Plugin Management** and then pressing install. The package helps with loading, initialization, settings, and build support for multiple devices such as Oculus and Windows Mixed Reality. The *XR Interaction Toolkit* (discussed in section 2.4.2) is at the time of writing in preview/beta access which means that bugs may be expected. In addition, the package's documentation is limited. The package can be installed by going into **Window** > **Package Manager**, and then enable preview packages by pressing **Advanced settings** and then ticking off on **Enable Preview Packages**. The *XR Interaction Toolkit* is then available in the **Unity Registry** packages, and can be installed. After all packages are installed, implementation of the application can begin.

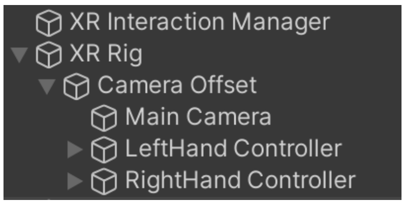
4.2 XR Rig

In order for a user to be able to explore an environment in VR, an object called the XR Rig is created. The XR Rig will work as the user's eyes, ears, and hands in the virtual world, and will allow users to move through the virtual environment. The XR Rig GameObject will read inputs and movement from the VR headset and its controllers. By doing this, an immersive experience can be created since a users movement in the real world will be translated to the application through the headset and controllers.

Every component and script attached to the XR-rig in this section is provided by the XR Interaction Toolkit. Each controller component will have independent colliders that can trigger physical interactions.

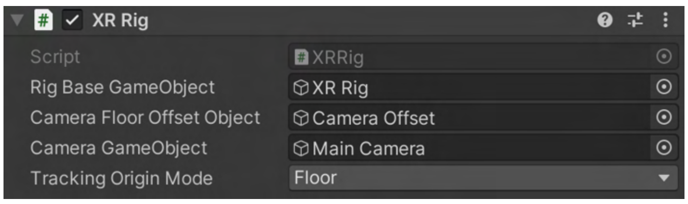
4.2.1 Structure

The XR Rig consists of the main camera and two controllers. The main camera will serve as the eyes of the user. It will be controlled by the VR headset, so if the user turns its head, the camera should turn as well. The two controllers will be controlled by the VR controllers. If the user moves or turns the controllers, so should the controllers in the virtual environment. A figure describing the structure of the XR Rig GameObject can be seen in Figure 4.2a.



XR Interaction Manager
- XR Rig
  - Camera Offset
    - Main Camera
    - LeftHand Controller
    - RightHand Controller

(a) Basic structure of the XR Rig GameObject. The *XR Interaction Manager* GameObject on the top enable interaction for the user.



XR Rig

Script	XRRig
Rig Base GameObject	XR Rig
Camera Floor Offset Object	Camera Offset
Camera GameObject	Main Camera
Tracking Origin Mode	Floor

(b) XRRig script attached to the XR Rig GameObject.

Figure 4.2: Structure of the XR Rig together with the script allowing for a Room-Scale XR Rig.

4.2.2 Configuration

The most important script component to attach to the XR Rig GameObject is *XRRig.cs*. This script is provided by the XR Interaction Toolkit. Several properties of the script has to be set as seen in Figure 4.2b. The *Rig Base GameObject* is the root object of the XR Rig hierarchy. This is the object being translated and rotated for movement. The *Camera Game Object* is used to render the scene, and serves as the eyes of the XR Rig. The camera is thus responsible for everything the user sees. To offset the camera so that it is placed at the user's height, the *Camera Floor Offset Object* is used. This option determines the distance between the ground and the user's eyes in the physical world and translates that height difference to the virtual environment. It is dependent on what *Tracking Origin Mode* is set to. The *Tracking Origin Mode* has five options: *Unknown*, *Device*, *Floor*, *Tracking Reference*, and *Unbounded*. If the *Device* option is chosen, the height of the offset can be manually set. However, by choosing the *Floor* tracking mode the offset will be automatically set to where to VR headset is physically located in relation to the floor. This option is only available for VR-headsets with six degrees of freedom, meaning a user can also physically move with the headset on and translate that movement to the virtual environment as seen in Figure 2.10. These two options result in Stationary and Room-Scale experiences respectively, as was presented in section 2.4.2. Since the application to be developed is best suited for users standing, a Room-Scale XR Rig is chosen. The *Tracking Origin Mode* is thus set to *Floor*. Being able to have the application adjust it's camera depending on a user's movement will make it more intuitive and immersive because a users movement will translate naturally into the virtual environment.

As the action-based behaviour input system is chosen as the preferred input system, the *Input Action Manager* script is added to the XR Rig. As described in section 2.4.2, this script will automatically enable and disable actions created in the action editor. By doing this, input from the controllers which control the specified actions are enabled. Alternatively, the actions used in the different scripts can be enabled and disabled in the *OnEnable()* and *OnDisable()* functions, but by providing the *Input Action Manager* script this is already taken care of. When the actions are enabled, the application can react to where the controllers and headset are located and to which buttons on the controllers are pressed.

It is possible to download a default set of input actions called *XRI Default Input Actions*. These input actions are bound to default controllers that are similar for several VR devices. Default controllers for head movement, controller movement, and other actions as seen in Figure 2.11a are thus added. The *XRI Default Input Actions* control scheme is set as the action asset, as seen in Figure 2.11b, and custom actions can be defined in the control scheme. These actions can then be used in the application.

To handle communication between scene objects available for interaction and the objects performing the interaction, the *XR Interaction Manager* GameObject is created as seen in Figure 4.2a. The XR Interaction Toolkit provide a script called *XRInteractionManager.cs* which enable interaction, and is attached to the *XR Interaction Manager* GameObject.

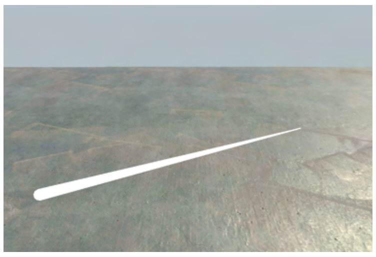
4.2.3 Main camera

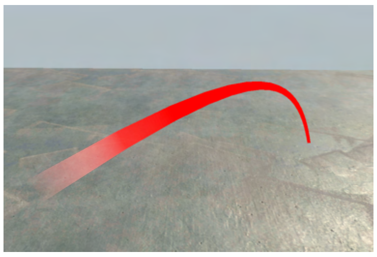
The main camera behaves as the eyes of the user. Here, several options for the camera projection can be changed. For instance, the camera's clipping planes can be decided. This option decides at what range an object is rendered. If it is too close or too far away from the main camera, the object will not be rendered. If an object is between these distances, it will be rendered. A distance from 0.1 - 1000 is set as it renders objects at close and long ranges.

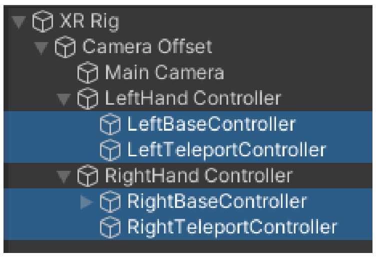
4.2.4 Controllers

The controllers are used for interaction with the virtual environment. This includes everything from interacting with the user interface (UI), interacting with objects in the virtual environment, locomotion and teleportation. To teleport, interact with the UI or objects at range, a ray can be used to help users know where interaction can occur. Interacting with the UI is easiest with a straight line, while teleportation is normally done with a curved ray. Both options for raycasts can be seen in Figure 4.3a and Figure 4.3b. As all these actions require different intuitive visual representations, a divided setup for the controllers is suggested. This is because the *XRRayInteractor.cs* script from the *XR Interaction Toolkit* can only be attached once to a GameObject. This is the script that decide whether the raycast should be straight or curved. The controllers are thus divided into a base controller that can do everything from grabbing objects to interact with the UI, and another controller that can teleport. The setup for the controllers can be seen in Figure 4.3c.

By adding a *Line Renderer* component, a *XRRayInteractor.cs* script, and a *XR Interactor Line Visual* component to the controller gameobjects, a visual line is cast from the controllers. The


(a) Straight line visual.


(b) Projectile curve visual.


XR Rig
Camera Offset
Main Camera
LeftHand Controller
LeftBaseController
LeftTeleportController
RightHand Controller
RightBaseController
RightTeleportController

(c) The controllers are divided into a base controller and a teleportation controller. This is to allow different ray curves attached to the same controller.

Figure 4.3: Line visuals added to the controllers.

curve of the line can be customized from the *XRRayInteractor.cs* script, where a straight line, projectile curve, or a bezier curve can be chosen. Each of the curves can be customized and and the curves reach distance can be set. The straight line is chosen on the *BaseController*, while the projectile curve is chosen on the *TeleportController*. This is done to optimize the controller ray curves based on a user's desired action.

Controller tracking

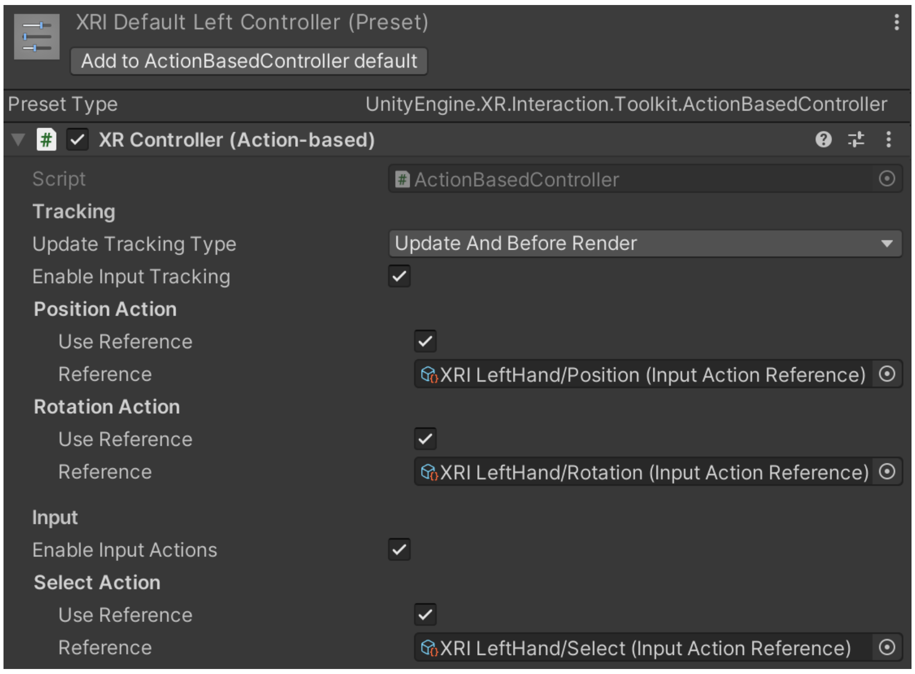
To enable tracking of the controllers and interactions with the virtual environment, the *ActionBasedController.cs* Unity script component is added. The components attached to this script describe the controllers movement, rotation, and basic input actions. When downloading and importing the *XRI Default Input Actions* from the XR Interaction Toolkit samples, there are default presets available. These presets describe what actions are bound to which controllers for different scripts providing locomotion and controller logic. Two of these are for the left and right hand controller, where the setup for the left hand controller can be seen in Figure 4.4. By adding this preset as the default, it will be automatically filled when adding the *ActionBasedController* to the left hand controller GameObject. However, the references to the actions can be customized. When all actions have references describing which controller sensors are attached to them, the tracking of the controller and actions are enabled.

Models and animation

To increase immersion, 3D models following controller movement will be added. Oculus offers a variety of packages containing different 3D models of controllers (Oculus, 2021a). By utilizing these packages, a developer gets access to all 3D models of the available Oculus device controllers, such as controller models for the Oculus Quest 2 and Oculus Go. Hand models are also available. The Oculus Hand Models package includes left and right hand models and animations for use with Autodesk Maya. Custom hand poses can thus be created, allowing for great flexibility in what hand poses best suits the application.

Hand models are considered to contribute to an immersive application experience, since the hand models can run animation corresponding to users pressing certain buttons on their controllers. To provide information to users about what the different buttons on the controllers do, models of the controllers can be used to display controller hints. To switch between the hand and controller 3D models, a GameObject with a script called *HandPresence.cs* is created. This object is used as the *Model Prefab* from the *ActionBasedController.cs* script which inherits the transform of where the controller physically is. This means that the 3D model's position and rotation will be mapped to the physical location of the user's controllers.

In *HandPresence.cs*, both the controller and hand model is instantiated. Initially, the hand model


XRI Default Left Controller (Preset)
Add to ActionBasedController default

Preset Type	UnityEngine.XR.Interaction.Toolkit.ActionBasedController
XR Controller (Action-based)	
Script	ActionBasedController
Tracking	
Update Tracking Type	Update And Before Render
Enable Input Tracking	✓
Position Action	
Use Reference	✓
Reference	XRI LeftHand/Position (Input Action Reference)
Rotation Action	
Use Reference	✓
Reference	XRI LeftHand/Rotation (Input Action Reference)
Input	
Enable Input Actions	✓
Select Action	
Use Reference	✓
Reference	XRI LeftHand/Select (Input Action Reference)

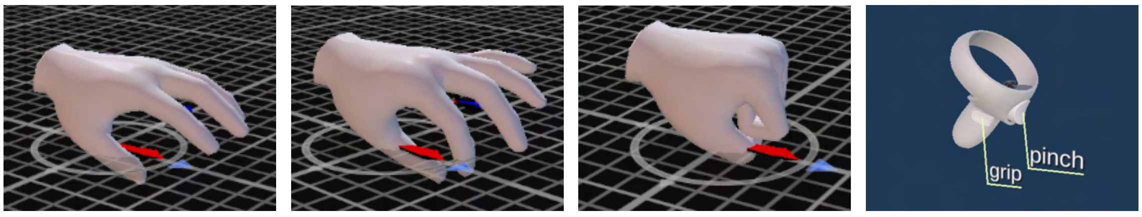
Figure 4.4: The default setup for the left hand action-based controller. It can be added to the ActionBasedController.

is enabled while the controller model is disabled. By pressing the *start* button on the controller, the controller model is enabled while the hand model is disabled. The opposite happens if the *start* button is pressed once again.

To create a more immersive experience, animation is added to the hand models. Three animations are created, a default hand pose, a pinch pose, and a grip pose. These animations can be seen in Figure 4.5. To manage these animations, an *Animator Controller* component is created and placed as the controller in the *Animator* component of the hand model. All the hand animations will be managed by two values, which will control the hand animations in a *Blend Tree*. To activate the different animations, the buttons on the controller seen in Figure 4.5d are chosen as the best fit. Since there are two values, the blend type is chosen to *2D Freeform Cartesian*. By using this blend type, there are four different scenarios that need to be handled: when none of the buttons are pressed, when each of the buttons are separately pressed, and when both buttons are pressed at the same time. Four motions are thus added. When none of the buttons are pressed, the default hand pose is shown. When one of the two buttons responsible for the animation is pressed, either the pinch hand pose or the grip hand pose is shown depending on which button is pressed. If both buttons are pressed at the same time the grip hand pose is shown. The buttons on the controller have float values ranging from 0 to 1. The float values from the buttons determines which stage of the animation will be displayed. For instance, when the button is fully pressed the animation will be at its final stage, but if it is halfway pressed it will be in the middle between the default hand pose and the animation corresponding to the button pressed. To listen to the input of the buttons the values is polled in the *Update()* function by calling *button*.action.ReadValue<float>() as described in section 2.4.2.

Physical interactions

As the tracking of the controller and controller models are present, the foundation for interactions in the virtual environment is enabled. To allow grabbing of objects with the controller, a script called *XRDirectInteractor.cs* from the *XR Interaction Toolkit* is added to the *RightHand Controller*. This is accompanied with a collider which will serve as a trigger. By having the trigger on both of the controllers, it is known when the controller 3D models are colliding with other objects. If such a collision is detected together with the desired controller input, it is possible to grab the object



(a) Default hand pose. (b) Pinch hand pose. (c) Grip hand pose. (d) Buttons on the controller corresponding to grip and pinch animation.

Figure 4.5: Hand animation of the Oculus Hand Model.

colliding with the controller triggers. The *LeftHand Controller* will interact with the UI, so it will have a straight raycast from the controller. The raycast is enabled by adding *XR Ray Interactor* to the base controller, which also enable interaction with objects from the ray. This means that by pointing the ray towards an object and pressing the *grip* button seen in Figure 4.5d which is chosen as the *Select* action, the object will be selected and arrive at the position of the controller.

However, to be able to interact with objects in the first place, a script enabling this action has to be added to the object to be grabbed. The Grab Interactable, available from the *XRGrabInteractable.cs* script from the *XR Interaction Toolkit*, allows a user to pick up, drop, throw, or place a GameObject by using either a *Direct* or *Ray Interactor* grab action. Another condition for an object to be grabbed is that it must have a *Collider* describing the physical boundaries of the object. If physical properties, such as gravity and weight are desired for an object a *Rigid Body* component has to be added to it.

4.2.5 Movement

The *Locomotion System* is the component that provides movement for the XR Rig. There are several ways to achieve locomotion. Some examples are teleportation, continuous movement, and turning. To move/turn a XR Rig, a locomotion provider first checks if the *Locomotion System* is available. If it is, it requests exclusive control of the system. If the request is granted, the XR Rig is moved or turned. When the action is finished, the locomotion action gives up its exclusive control.

There are four different components that are added to the XR Rig to give the user the option to decide how to move. These are *Continuous Move Provider*, *Teleportation Provider*, *Snap Turn Provider*, and *Continuous Turn Provider*. To move using the *Teleportation Provider* a valid area to teleport to needs to be created. This can be achieved by adding a *Teleportation Anchor* or *Teleportation Area* component to an object. However, this requires the object to have a collider attached, as the XR Rig must be prevented from falling off the area teleported to. To move using the *Continuous Move Provider*, a button on the controller has to be selected to control movement. The joystick on the left hand controller is the standard for moving a character in a video-game. This controller input is therefore decided to be used for continuous movement.

When it comes to turning the camera, which function as the users eyes, two options are provided from the XR Interaction Toolkit. One snaps at a decided angle, while the other have a continuous rotation.

To control how the XR Rig behaves when it collides with an object, the *Character Controller* component is added. Here, the radius and height of the XR Rig can be decided. The maximum steepness the character is allowed to ascend can also be set. However, when grabbed objects are placed inside the character boundaries decided by the *Character Controller*, problems arise. If an object is placed inside the characters boundaries the XR Rig will start moving in uncontrolled directions. To avoid this behaviour, the XR Rigs body and other scene objects available for grabbing have to separated in different *Layers*. The parent XR Rig GameObject is therefore changed from the *Default* layer to the new *Body* layer. The objects being interacted with is changed to the new *Grab* layer. By going into **Edit** > **Project Setting** and then to **Physics**, the option to turn of collision between the *Body* and *Grab* layer can be achieved in the layer collision matrix. After this change is applied, a user can safely pick up objects and move them towards the XR Rig's body without causing uncontrolled movement.

However, after applying the *Character Controller* the height of the XR Rig got an offset. The *Character Controller* has a static *height* component that push the camera of the XR Rig higher. A user thus feels like they are taller than they actually are in the virtual environment. To avoid this behaviour the *Movement.cs* scripts is created. Since the *Character Controller* has a static height that offsets the XR Rig, the solution is to dynamically set the height of the *Character Controller*, and thus the XR Rig, to what the height of the VR headset is. This can be achieved by first getting access to the *Character Controller* in the XR Rig in the *Start()* function. In the *Update()* function, the height of the *Character Controller* is set to the height of the VR headset by utilizing the *cameraInRigSpaceHeight* property of the XR Rig. The mass center of the *Character Controller* also has to be dynamically set. This is to make sure that the character's capsule is placed in the correct position such that its height value also is correct. The center of the capsule can be obtained by first getting access to the local position of the camera with respect to the XR Rig. This will serve as the x and z components of the capsule center. The local position of the camera with respect to the XR Rig can be found by using *transform.InverseTransformPoint(rig.cameraGameObject.transform.position)* where *rig* is the reference to the XR Rig. The y component, meaning the height, can be found by using the *height* found dynamically above, and divide this value by 2. By setting the *Character Controller*'s center to these values, the actual mass center of a user is set and the result can be seen in Figure 4.6.

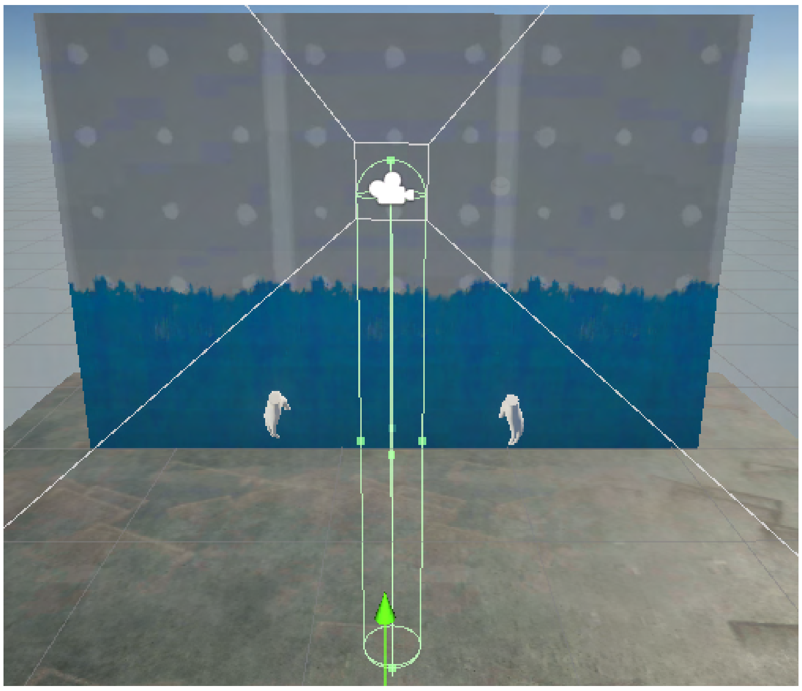


Figure 4.6: The *Character Controller* capsule. The height of the capsule is set to where the VR headset is placed with respect to the defined floor level. The mass center of the capsule is placed in the middle between the headset and the floor.

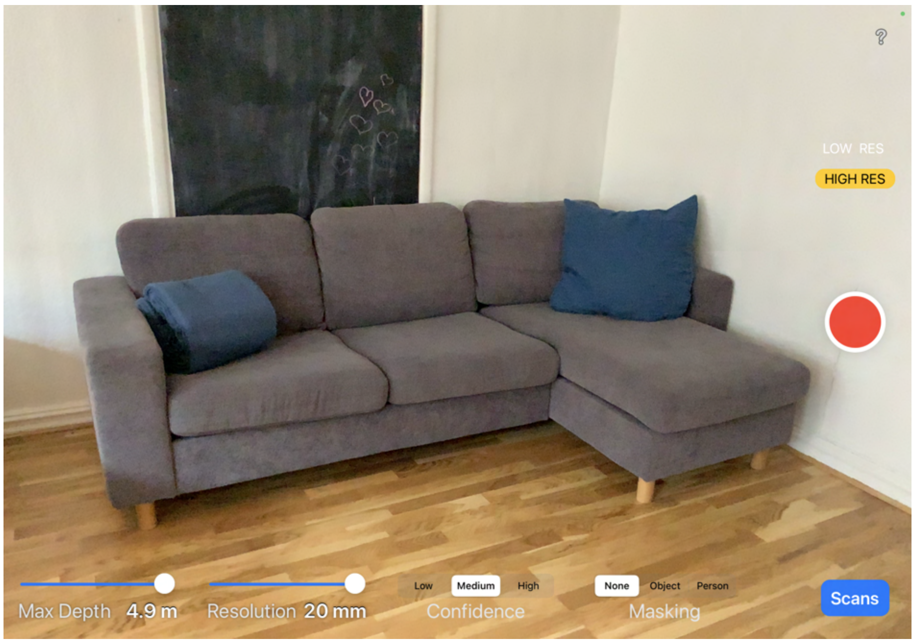
To set the direction of the *Character Controller* to where the user is looking, the *Character Controller* must be rotated with the head yaw. The head yaw of the user is accessed by using the *Quaternion.Euler()* function, and use the y-rotation of the XR Rig's camera as the input. The direction of where the user is looking can then be obtained, and the *Character Controller* can be rotated accordingly.

4.3 Virtual environment

After a working controllable XR Rig which reacts to input from VR hardware is implemented, the next step is to create a 3D environment that the XR Rig can explore. As discussed in section 3.2, a virtual model of an environment will be created by utilizing the LiDAR sensor of a second generation iPad Pro and 360° images captured by a GoPro max.

4.3.1 LiDAR scans

3D models generated from LiDAR scans will be the virtual environment users can explore in the VR application. There exists several apps on Apple's App Store that can create a 3D model from a LiDAR scan. After experimenting, it is found that the free '3d Scanner App (2021)' by Laan Labs gives accurate results and will therefore be used as the main tool for creating 3D models from LiDAR scans. Using the app, a user can perform a LiDAR scan, create a 3D model from its point cloud and export the finished 3D model. The app has an intuitive user interface which allows multiple scan parameters to be adjusted as seen in Figure 4.7. These parameters includes the accuracy of the scan in millimeters (mm), the max detectable scan depth, the confidence level for a point to be added to the point cloud and a masking mode which can mask out people or object shapes from the scan. After a scan has been completed, simple edits of the point cloud can be performed directly in the app. This includes cropping, scaling and rotation. It is also possible to preprocess the scans. This includes model simplification to reduce the polygon count of a model, scan smoothing and applying textures to the point cloud. In addition, the app features a low-res scanning option with preset scan parameters used to minimize scan size.



Max Depth 4.9 m
Resolution 20 mm
Confidence
Masking

Figure 4.7: The user interface in '3d scanner app (2021)' by Laan Labs when performing a high resolution LiDAR scan.

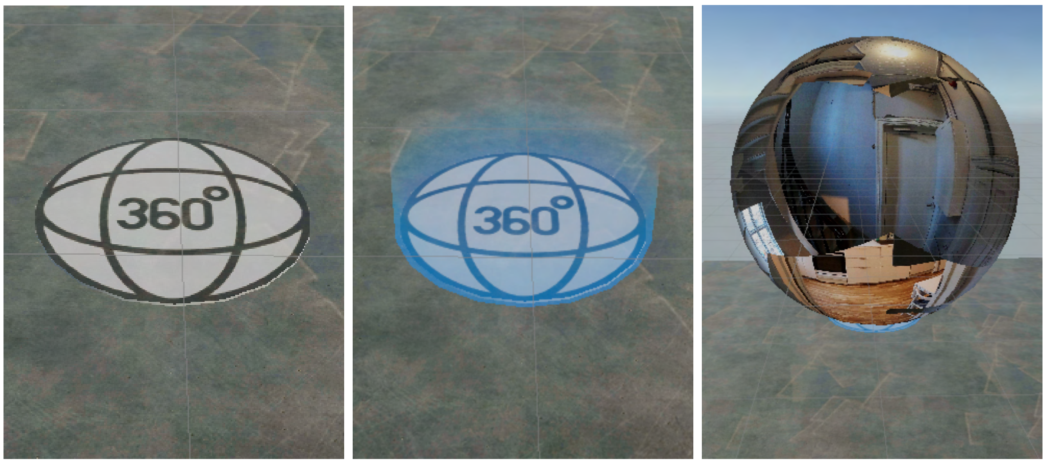
After a LiDAR scan has been taken and a 3D model is generated from it, it is possible to export the point clouds from conducted scans in a wide variety of formats. One of the options is exporting the model as an *.obj* file, which is a supported file format in Unity. By making the model an *.obj* file, it can be directly exported to Unity. In order for a user to be able to interact with the model, a mesh collider component is added to the imported 3D model. The mesh collider automatically generates colliders on the mesh of the imported model. Without a mesh collider, a user of the VR application would fall through the model. The generated collider makes the imported 3D model behave like a physical object that a user can interact with. After the mesh collider is added to the imported model, it is possible for a user to explore it in VR through the XR Rig.

4.3.2 Image Anchors

360° images captured by the GoPro Max will be used to supplement a 3D model generated from a LiDAR scan in order to provide a higher level of detail to the virtual environment. A user of the application should be able to toggle between exploring the 3D model and 360°images of specific parts of the model. For instance, if a user is standing in close proximity to a vessel motor in the 3D model, the user should be able to toggle a 360° image of the motor to achieve a more detailed look of it and its surroundings.

Structure

In order to detect if a user should be able to toggle a certain image, an *Image Anchor* object is created. The *Image Anchor* object consists of three child objects: the *anchor* object which is a platform at ground level, an *image* which is a sphere that will have the desired 360° image as a texture, and a *glow effect* used for user feedback. When a user of the application is standing on the *anchor* platform through the XR Rig, the *glow effect* should trigger and indicate to the user that the Image Anchors *image* can be toggled. When the user presses the designated toggle button, the *image* object with the desired image as a texture should be activated as seen in Figure 4.8. Since the *image* object is a sphere which completely surrounds the XR Rig, toggling the *image* object will seem to the user as a seamless switch between the 3D model and the 360° images. When a user does not want to inspect the 360° image, the user should be able to deactivate the *image* object by pressing the same button that was used to toggle the *image*. In order to make it easier for a user to move between different Image Anchors, users should be able to teleport between different *anchor* platforms.



(a) The Image Anchor platform when it is inactive.
(b) The platforms glow effect is triggered when a user is standing on it.
(c) A user can toggle a 360°image when standing on the Image Anchor Platform.

Figure 4.8: The three states an Image Anchor can be in. The Image Anchor is inactive in Figure 4.8a, in the hover state in Figure 4.8b and active in Figure 4.8c.

Implementation

To enable the above functionality, a script called *ImageHandler.cs* is created and attached to the *Image Anchor* object. To enable toggling of the photo and the *glow effect*, a *Capsule Collider* is added to the *Image Anchor*. When a user of the application through the XR Rig is colliding with the *Capsule Collider* because of the character controller on the XR Rig, the *glow effect* will trigger. The *Update()* method of the *ImageHandler.cs* script, will continuously listen for the input command for toggling between the LiDAR model and 360° photos. If the input command is detected and the XR Rig is colliding with the capsule collider of the Image Anchor, it is known that the user is inside the anchor platform and will be allowed to toggle the anchors photo. After the sphere object with the photo texture is toggled on, the LiDAR model is deactivated to save computational power. Four box colliders which creates a wall around the photo sphere is activated to make sure users can not move outside of the photo sphere. The wall collider structure is shown in Figure 4.9. When the *Update()* function detects the toggling input command again, the four box colliders and the photo sphere is deactivated and the LiDAR model is activated. In order for the toggling functionality to seem like a seamless switch between a 360°photo and the LiDAR model, the photo texture on the sphere must be rotated according to its environment.

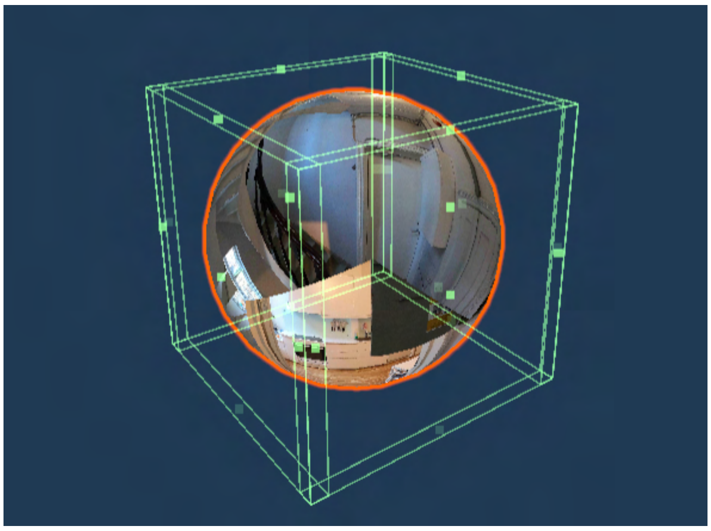


Figure 4.9: The colliders being activated around a photo sphere to make sure users can not move outside of the image view.

4.4 User interface

The user interface of the application should allow users to navigate between different environments, toggle network functionality, adjust controller options and provide assistance to new users of the application by displaying controller hints. The UI should have a simple and intuitive layout and be easily accessible.

4.4.1 Menu

In order for users of the application to be able to switch between different virtual environments and trigger network functionality, a menu system is implemented. The menu will be toggled by the press of a button and will be displayed in front of the user. The menu will also enable users to change parameters of their VR experience in an options tab.

A Canvas Unity object is the area that all UI elements are inside. All UI elements must be children of a Canvas object, where the Canvas object decide if the UI elements are rendered in screen space or world space. By using screen space, the UI is placed on the user's screen in front of the rendered scene. This means that the Canvas has a static position in front of the users field of view. By using the world space render mode the Canvas will behave like any other object in the scene. In this way, the Canvas can be placed at any desired location in the scene. A canvas object called *Menu canvas* is created as a parent object for all menu related buttons and text. When a user presses the menu button, a menu should appear before him. Since it is desired to not have the menu overlay users field of view, the render mode of the Canvas is set to *world space*. To enable the XR-rigs rigs controller rays to interact with the menu, a *Graphic Raycaster* component is added to the *Menu canvas*.

Creating an interactable menu

By creating children text objects to the menu canvas and adding a *button* component to them, a simple interactable menu is created. Each menu view is created as a separate child Gameobject to the canvas. The text buttons *OnClick()* method is used to detect clicks on different menu items. When a click on a specific menu item has been detected, the necessary menu views are toggled to allow menu navigation. A simple UI displaying the applications main menu can be seen in Figure 4.10.

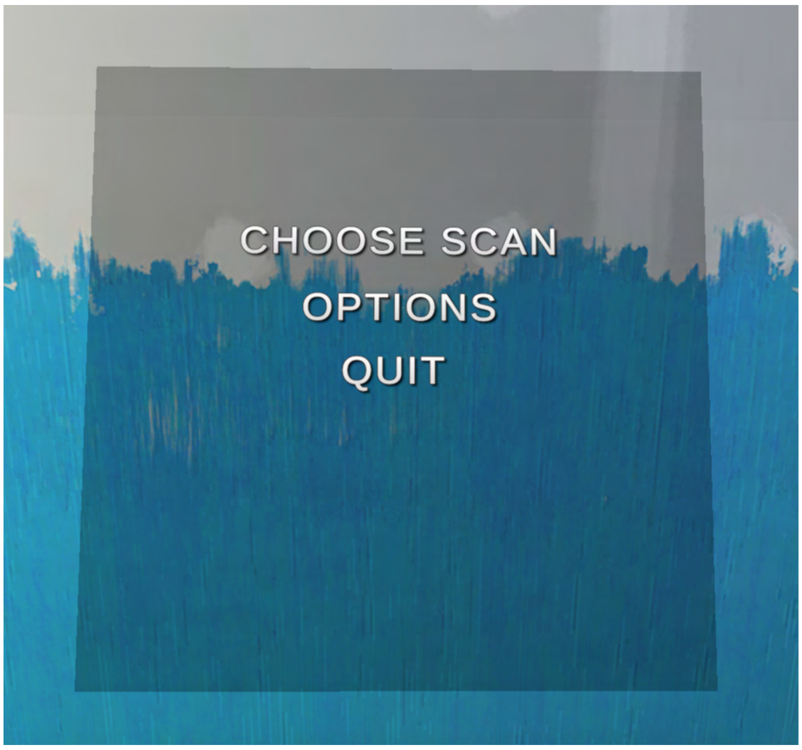


Figure 4.10: The main menu of the application.

Toggle and position the menu

To toggle the *Menu canvas* object and position it according to a users position and orientation, a script called *MenuHandler.cs* is created. The *MenuHandler.cs* script's *Start()* method initializes the Menu Canvas object by automatically setting it's event camera.

The main logic for toggling and positioning the menu canvas is implemented in the *MenuHandler.cs* script's *Update()* method. The *Update()* method will continuously listen for the designated input to toggle the menu. When the input is detected, the menus position is calculated based on the users position and orientation. When the calculation is complete, the menu will be toggled and appear in front of the user. Once the menu is activated, it will change its orientation to always face the user by using the *LookAt()* Unity function. When the user presses the designated menu input again, the menu will be deactivated.

Scene selection

In order to implement scene selection, a GameObject called *SceneList* is created. The *SceneList* object has a custom written component script called *SceneList.cs*. The *SceneList.cs* script contains a list of *public serializeable* C# class objects which stores information about all available scenes in the application.

In the menu view called "Choose Scan", users should be able to select available scenes and view an image of their environment. If a user has selected a scene and wants to switch to it, he can do so by pressing a designated button in the menu view to load the desired scene. A script called *AvailableScenesManager.cs* is created to help generate a button for each scene in the *SceneList* object and react to a user's button selection. When activated, the scene selection menu view called "Choose Scan" will run a function called *GenerateMenuIcons()* that automatically instantiates a button for each available scene in the *SceneList* object and display them in a scrollable list in the menu view as seen in Figure 4.11. Each instantiated button has a text containing the scene name of the scene they were generated from.

Once one of the instantiated scene buttons in the "Choose Scan" menu view are pressed, a function called *TaskOnClick()* will run. The *TaskOnClick()* function uses the string text on the pressed button to call a function called *PrepareScene()* from a script called *SceneLoader.cs*. In addition to loading scenes, the *SceneLoader.cs* script is responsible for keeping track over which scene a user has selected in the menu and displaying an image of it to provide user feedback. In order to do so, the *SceneLoader.cs* script has a public list containing images of the available scenes. The *PrepareScene()* function iterates through the list of scene images and displays the image with the same name as the selected scene in an image container next to the scrollable scene list as seen in Figure 4.11. When a user has selected a scene and presses the switch scene button, a function in *SceneLoader.cs* will run and load the selected scene.

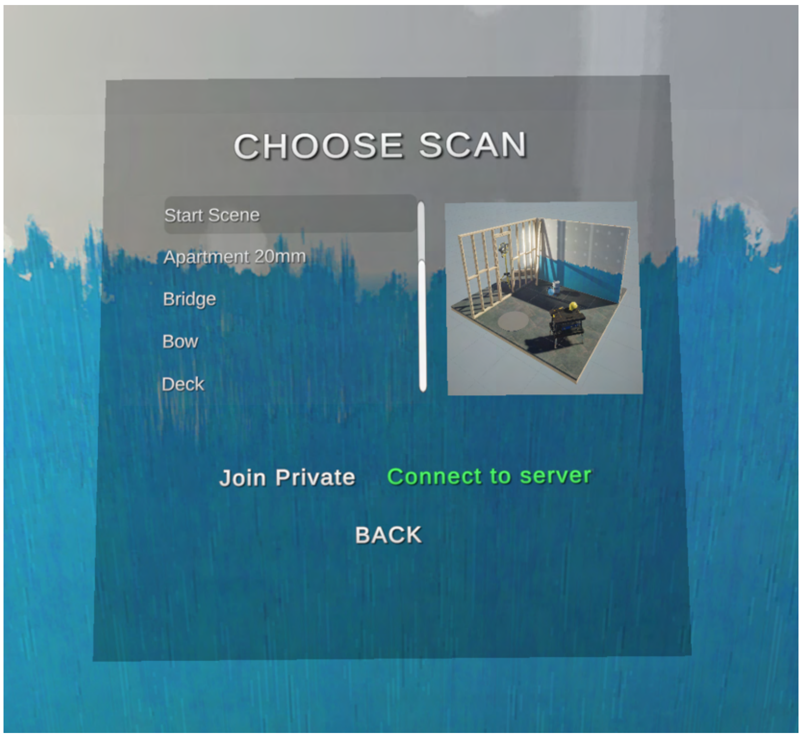


Figure 4.11: The menu view "Choose Scan" displaying available scenes and a photo of the selected choice. Users can choose to switch to a scene in offline or online mode which enables network functionality.

Options

To avoid VR-sickness, users of the application should be able to adjust the turning movement controller type. There are two main ways of turning in VR using the controllers, namely continuous turn and snap turn. When using the continuous turn option, a user rotates continuously with the joystick while the snap turn option rotates a user with a specified number of degrees. Users have individual preferences regarding which turn option feels best. Using a turn option which feels uncomfortable can induce VR sickness. The options tab in the created menu should therefore allow users to choose turning option and adjust the turn rate for both options.

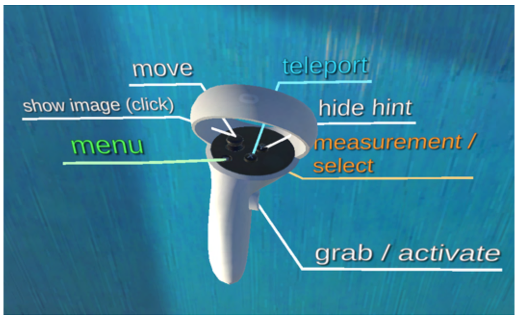
Since the selected options preferences has to be stored between different Unity scenes, an object called *Options Information* is created. The *Options Information* should store all selected settings. To implement this, a script called *OptionsInformation.cs* is created. The script stores the turn option and the desired turn rates for the two options. It also has set and get functions for the stored variables. When a new Unity scene is loaded, all objects from the previous scene is destroyed. In order to avoid this, the *Options Information* object is given the *DontDestroyOnLoad()* function in the *Start()* method of the *OptionsInformation.cs* script. This ensures that the *Options Information* object is not destroyed between scenes, and its information is kept.

To create the menu view that allows users to adjust turning properties, different UI elements are added. A checkbox is used to toggle between the two turn options. The continuous turn rate is adjusted with a slider, while the snap turn angle is adjusted with a drop down menu of selectable degrees. In order to react to inputs to these UI elements, a script called *OptionsHandler.cs* is created. The *OptionsHandler.cs* script is responsible for detecting changes to the UI elements in the Options menu view, storing them in the *Options Information* object and setting the set changes to the XR Rig. The *OptionsHandler.cs* script is added as a component to the main menu canvas. In its *Start()* method, the *OptionsHandler.cs* script gets access to the XR Rig and its turn providers. It also gets access to the information stored in the *OptionsInformation.cs* script on the *Options Information* object and sets the stored values from the script on the XR Rig. In order to achieve toggling between two turn modes, both turn modes are added as script components to the XR Rig. Switching between two turn modes is accomplished by toggling the two script components depending on which is selected in the menu view and setting the turn rates in the active component. When the UI turn adjustment elements detects a change in value, the changed UI element calls a function from the *OptionsHandler.cs* script which stores the new value in *OptionsInformation.cs* and sets the changed value in the XR Rig.

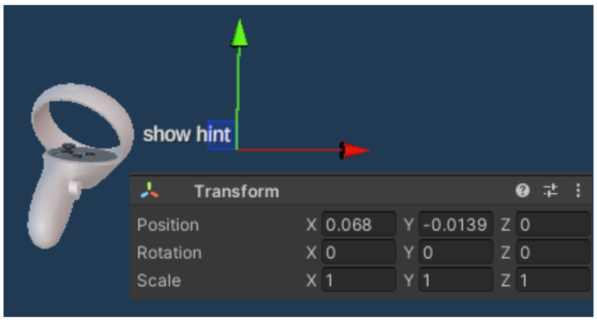
If a user presses the "Quit" button in the main menu view, the Unity function *Application.Quit()* which closes the application.

4.4.2 Controller hints

To be able to see what the different buttons on a controller do, controller hints are added. First, a GameObject is added to structure the controller hints. It is placed as the first child of the root controller GameObject. This is because it then inherits the transform of the controller GameObject, and thus follows the position and rotation of the controllers. Then a world canvas scaled down to the appropriate size containing the text hints is added. To visualize for the user which button corresponds to the actions written in the text, lines pointing in the direction of the button is added as can be seen from Figure 4.12a.



(a) Controller hints with text describing what the different buttons do.



Position	X 0.068	Y -0.0139	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

(b) The local position of one of the edges describing the line projection.

Figure 4.12: Controller hints.

The lines pointing at the individual buttons are added at runtime. To do this, a specified number of points describing the edges of the line projection are added to a *Line Renderer*. An example of one of these points can be seen in Figure 4.12b. To add the points to the *Line Renderer*, a script called *ControllerInformation.cs* is created to provide this functionality. First, the points indicating the edges of the line have to be specified and added. Then, the number of points in the *Line Renderer* is set to the number of points provided. At last, the position of the points is set to the local position of the points provided. By utilizing the local position, the actual position is decided with relation to the text and button instead of the global position. As the local position is small, instead of placing the edges at the origin of the global scene, the edges are placed with respect to the controllers as seen in Figure 4.12b.

4.5 Points of interest

A point of interest (POI) will serve as an informational and interactable object that provides users with relevant information about the objects they are investigating in the virtual environment. For instance, if a user is inspecting a motor, the user should be able to select the motor's POI which will display relevant data about the motor. This data could be the type of motor the user is looking at, its service history and possible claims related to it. In order to achieve this, data for each POI object in an environment must be retrieved from a database. The POI feature will therefore require an internet connection in order to work properly.

4.5.1 Structure

Gard has significant amounts of data about the vessels they insure. This data includes general information about vessels, their claims history and all vessels cards and certificates. Most of Gard's digital infrastructure is run in Microsoft Azure (discussed in further detail in section 2.5). The application must therefore be able to connect to Gard's relevant databases hosted in Azure in order to retrieve vessel data.

Database

There are two main ways to host a database in Azure. A database can be run as Infrastructure as a service (IaaS) or as Platform as a service (PaaS). By hosting a database as IaaS, a virtual machine is created in Azure. Operating system and a SQL server then has to be installed on the virtual machine in order to be able to access a database. By hosting a database in Azure as PaaS, hardware, operating system and type of SQL server is controlled by Microsoft. In other words, running a database in Azure as PaaS provides an abstraction layer between the users and the hardware. The benefit of hosting a database as PaaS is that there is no need for a user to set up a virtual machine, buy an SQL server license or maintain and update software on the virtual machine as this is done by Microsoft. It is also cheaper from a resource point of view. The database the application retrieves information from will therefore be set up as PaaS.

Unity Objects

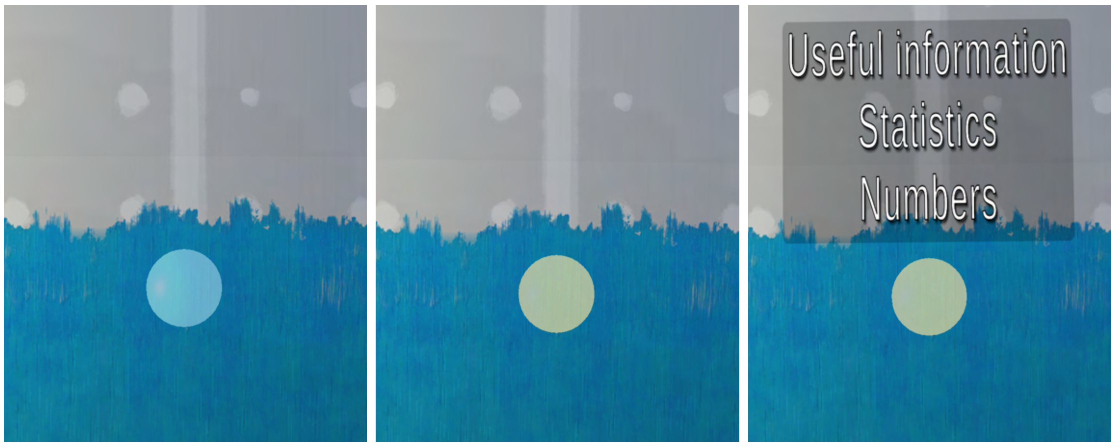
A POI object consists of two child objects, a panel that will display information retrieved from one of Gard's databases and a textured orb which will serve as a 3-dimensional button used to toggle the informational panel. In order for a Unity scene to be able to retrieve data from a database hosted in Azure, each scene has an additional object called *Database Manager.* Each time a Unity scene is initialized in the application, the *Database Manager* object will retrieve data from a database and feed it to all POIs in that scene.

4.5.2 Code logic

To retrieve data from a database hosted in Azure in a secure manner, the data will be fetched by connecting the application to a web server that retrieves the desired information if the IP address of the client is white-listed by Gard. By doing this, there is no need to have the database username and password in the application code, as these are stored in the web server script that are unavailable to the client. Getting data through an Azure web app also provides an additional level of security, as Gard has to white list all IP addresses that will be able to retrieve data from the database. If restricted data is to be displayed in the app, Gard can ensure that only approved users will be able to retrieve the information.

Controlling a virtual display

In order to enable toggling functionality of the informational panel in addition to visual user feedback when the orb is selected, a script called *POIHandler.cs* is created. Each POI will have a public enum variable called *TYPE* which describes of which type the POI is. A POI TYPE decides what kind of information it should display. For instance, if a POI is of type *VESSEL*, it should display general information about the vessel, but if it is of type *MOTOR* it should display motor relevant data. To provide feedback to a user that a POI can be interacted with, the *POIHandler.cs* script will switch the color material on the orb when a user hovers over it or selects it. When the orb detects that a user clicks on it, the *POIHandler.cs* script will activate the informational panel. If the orb detects another click, it deactivates the panel. Toggling of the informational panel and user feedback from the POI's orb is demonstrated in Figure 4.13. In order to make sure that the panel is clearly readable for the user when activated, the *LookAt()* function will run in the *Update()* method of *POIHandler.cs* to make sure that the informational panel is always rotated towards the XR Rig's main camera. By adding a *content size fitter* component to the panel displaying the retrieved data, the panel will scale automatically to fit the retrieved data.



(a) The POI when it is inactive. (b) The POI changes material when a user is hovers over it with the line interactor. (c) A user can toggle a POI by selecting it to display information about specific objects on a vessel.

Figure 4.13: The three states a POI can be in. The POI is inactive in Figure 4.13a, in the hover state in Figure 4.13b and active in Figure 4.13c.

Retrieving data from the database

A script called *DatabaseConnector.cs* is created and added to a scene's *Database Manager* object to enable the application to make GET requests to the web server running a PHP script in Azure. The *DatabaseConnector.cs* script has a public variable called VESSEL_ID which will be used as a primary-key by the web server to retrieve information about specific vessels from the database. An illustration showing the client-server relationship can be seen in Figure 4.14.

When a Unity scene in the application is initialized, the Start() method in the *DatabaseConnector.cs* script will find all objects in the scene with the tag POI and add them to a list. After all POI objects are found and stored, a URL will be constructed from a read-only string that is the URL to the web app hosted in Azure and the vessel ID parameter of the scan which is stored in the *Database Manager* object. For each POI in the scene, the POI's TYPE variable is added to the constructed URL. A GET request will then be sent to the web server using functions from the *UnityWebRequest* class. The GET request will then be handled by a PHP script hosted in the Azure web app, which will construct an SQL statement from the incoming VESSEL_ID and the POI type. For instance, if the TYPE variable of a request is MOTOR, an SQL statement to retrieve

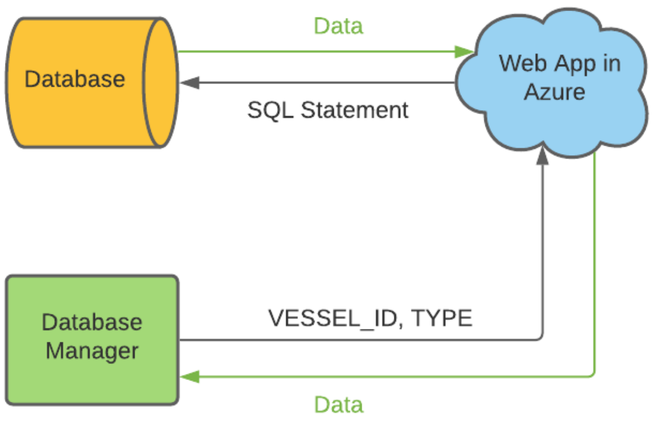


Figure 4.14: The data flow from the database manager object within the application (client) to the web server in Azure.

motor information will be constructed. If desired, the code can be further customized to allow for more variables to be sent with the GET request. This could be desirable in a situation where a vessel has more than one main engine. This would require the different POIs of TYPE MOTOR to distinguish between the different motors in order to retrieve correct information. An alternative implementation of the application to Azure web server communication could be to call different PHP scripts from Unity depending on the TYPE of each POI, and just pass the VESSEL_ID with the GET request.

After the PHP script hosted in the Azure web server has constructed an SQL statement from the incoming GET request sent by the *DatabaseConnector.cs* script in the application, the PHP script will run the query to the database. The returned information is then stored in an array in the PHP script. A string is created from the information in the array. The string is then returned to the application. When the *DatabaseConnector.cs* script detects a response to its GET request, it will update the informational panel on the POI that sent the request with the data that was received. As a result, if the application is connected to the internet and the IP address of the device is whitelisted by Gard, all POIs in a scene will retrieve their data during scene initialization.

4.6 Measurement tool

The measurement tool will allow users of the application to measure distances within the virtual environment. Because the virtual environment models are generated with a one-to-one relationship from LiDAR scans, it is possible to measure distances in them fast and with great precision. Being able to accurately measure distances virtually is useful for surveyors because it allows them to measure the length of potential cracks, areas with rust, or damaged areas. In addition, the measurement tool enables surveyors to measure specific distances between objects on a vessel that has to adhere to guidelines set by the company.

4.6.1 Structure

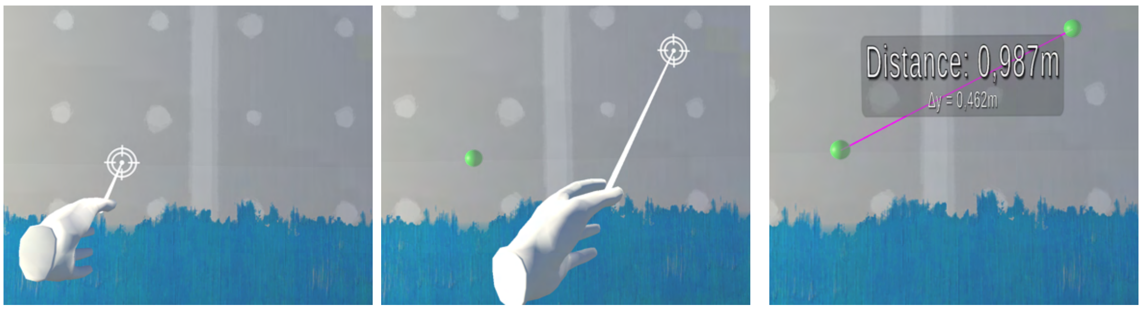
A measurement taken in a virtual environment generated from a LiDAR scan will have a potential inaccuracy equal to the resolution of the LiDAR scan. The goal of the measurement tool is to provide a fast and efficient way to conduct distance measurements in the application. It should be simple to use and provide users with feedback both during and after measurements are taken.

Taking measurements

The measurement tool will consist of two main components. The first component will be the object that performs the measurement. To provide a natural way of taking measurements, the measurement tool script will be added to the *left base controller* in the XR Rig. This will allow users of the app to take measurements by using their existing controller by utilizing the XR Rigs *XR Ray interactor*. If the ray on the left controller hovers over an object with a collider component, it is possible to detect the contact and use the collision for taking measurements. As a result, measurements can only be taken on objects that has a collider component attached.

User feedback

The second component in the measurement tool will be the measurement itself. In order to provide user feedback about the measured distance and the location of the points in the measurement, different assets are created. Each selected point should have an orb indicating the location of the point. When a user hovers over a surface with the tool, a marker indicating that the surface can be measured should be displayed. If two points are selected, a line between the two orbs should be displayed to indicate the measured distance. When two points are selected, an informational panel should display the measured distance in meters. In addition, the panel should also display the Δy component of the measurement to show the height difference between the two selected points. The user feedback from an example measurement taken using the measurement tool can be seen in Figure 4.15.



(a) A measurement marker indicates where a measurement point will be placed.

(b) A green orb is instantiated at a marked measurement point.

(c) When two measurement points are set, a line renderer is drawn between the two points and a panel displays the distance between the points as well as the measurements Δy component.

Figure 4.15: The three stages of taking a measurement.

4.6.2 Code logic

A script called *MeasurementTool.cs* is created to implement distance measurement functionality. The *MeasurementTool.cs* script is added as a component to the *Left Base Controller* on the XR Rig. The *MeasurementTool.cs* script has three public input variables, an input action from the controller it is attached to, a measurement marker prefab and a reticle prefab. Prefabs are a special type of component that allows fully configured GameObjects to be saved in a project for reuse. These assets can then be shared between scenes without having to be configured again. To keep track of registered measurement points and their orb markers, two lists are created to store their values.

A raycast is a ray that gets sent out from a position in 3D space and moves in a specific direction. In the *Start()* method of the *MeasurementTool.cs* script, access to the XR Ray Interactor from the *Left Base Controller* is achieved. This allows the *MeasurementTool.cs* script to utilize the raycast from the *Left Base Controller*. The *Start()* method also gets access to the informational panel that will display the measured distance and instantiates a reticle for the *Left Base Controller*. A line renderer is also created. The line renderer will enable a line to be drawn between two measurement points to illustrate the measured distance.

The measurement logic is implemented in the *MeasurementTool.cs* script's *Update()* function and is illustrated in Figure 4.16. The *Update()* function will continuously listen for the designated input to perform a measurement. When the input is detected, a raycast will be sent from the *Left Base Controller*. If the raycast hits an object with a collider, the coordinates for the hit point is stored in the list of measurement points and an illustrative orb marker is instantiated at the point. The orb marker object is stored in a separate list. When two measurement points are selected, a purple line illustrating the measured distance is generated between the two points. The distance and the points Δy component is then calculated. When the calculation is complete, the informational panel displaying the distance is updated and set at a calculated position between the middle point of the distance line and the main camera. When the informational display is toggled on, its rotation is being updated in the *Update()* method to ensure that it always faces the user. The next time a user presses the measurement input, the measurement is reset. This includes toggling the informational display off, removing all hit points and their markers and disabling the line renderer. After resetting all data from the previous measurement and disabling its markers and displays, the application is ready to perform a new measurement.

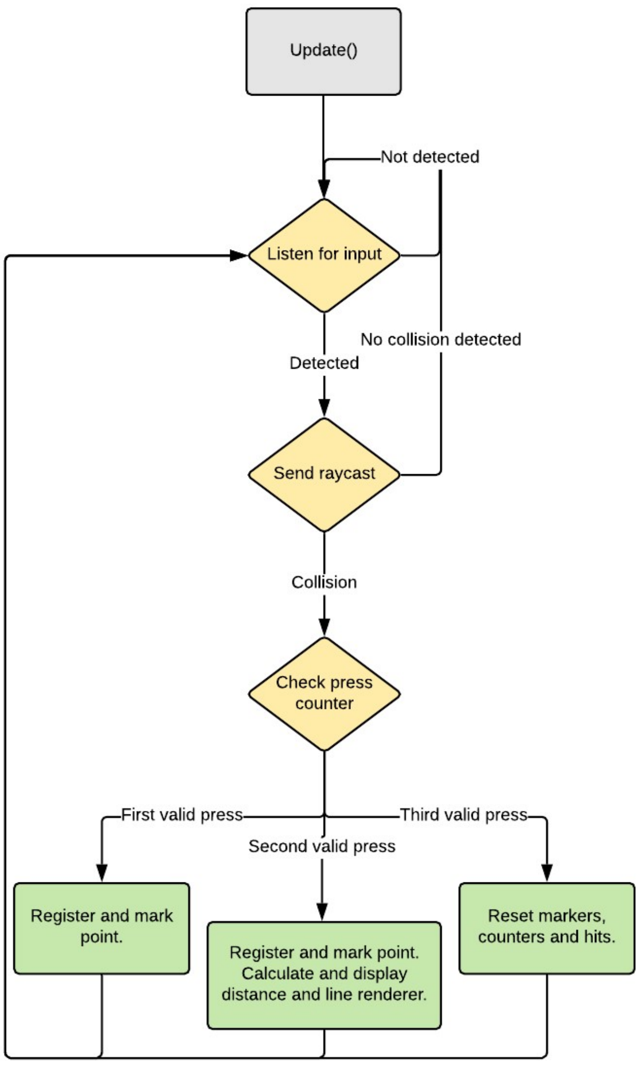


Figure 4.16: The main logic of the Update() function in the *MeasurementTool.cs* script

4.7 Networking

The application's networking functionality will enable clients to connect to the same server and collaborate in the virtual environments. As discussed in section 3.3, Photon Unity Networking (PUN) is used to enable this functionality. An explanation of how this package works is presented in section 2.4.2.

4.7.1 Setup

To enable Photon Unity Networking (PUN) it first has to be added to *My Assets* from the Unity Asset Store. The package then becomes available in the *Package Manager*, and can then be imported into the project. A Unity project has its own application identification code assigned from its connected PUN account. This is a unique identifier of the project, and is used for separate parties in the same region to connect to the application over the Photon Cloud. In the Photon Server Settings, available from **Window** > **Photon Unity Networking** > **Highlight Server Settings**, the application identifier (AppID) can be entered.

4.7.2 Server Connection

To connect to the Photon Cloud, the *PhotonNetwork.ConnectUsingSettings()* method has to be called. To know if the connection attempt succeeded or failed, PUN has implemented different callbacks that can be utilized. By using these, a developer knows which state the connection is in. One of the most convenient ways of generating callbacks is by extending the *MonoBehaviour-PunCallback* class instead of *MonoBehaviour*. This class exposes specific properties and virtual methods that are useful when implementing network features. Two of these virtual methods are *OnConnectedToMaster()* and *OnDisconnected()*. When the application is trying to connect to the server, these methods can be used to further proceed the network logic.

If the connection succeeds, the *OnConnectedToMaster()* method is called. Since it is desired that a client is connected to a room once connected to the server, the *JoinOrCreateRoom()* method is called after a successful connection is detected. A connected client will join or create a room with the same name as the name of the scene the client is in. A maximum of 20 users will be enabled in the same room. If the maximum number of clients are exceeded, another room will be set up.

4.7.3 Load networked scene

When a client has connected to a server and joined as presented in subsection 4.7.2, it is desirable to be able to join other rooms. From subsection 4.4.1 it is known that the *SceneLoader.cs* script keep track of which scene the client has selected from the menu. As the client presses the *Connect to server* button in the menu as seen in Figure 4.11, the *LoadScene()* function from the *Network-Manager.cs* script is called. Here, the current scene the client is present in is compared to the scene that has been selected in the menu and if they do not correspond, the *PhotonNetwork.LeaveRoom()* function gets called. Once the client has left the room, the avatar that represents the client in the room gets destroyed and the client returns to the Master Server where the client is able to join or create new rooms. Once connected to the Master Server, the client will try to join or create a new room by calling *CreateOrJoinRoom()*. If a client is already connected to the server, the client will once again fetch the scene selected in the menu from the *SceneLoader* script, and try to join or create a new room with the name of the selected scene. If there already exist a room with the same name, the client will join this room, otherwise the client will create the room. Once a room has been joined or created, the new scene is loaded.

4.7.4 Network client setup

To make a client visible to other connected users to the same room, an avatar is created. To make the avatar appear, the *PhotonNetwork.Instantiate()* method has to be called. This method has the avatar, and the position and rotation it should be instantiated at as input variables. What is important to remember is that the avatar is referenced using a static name placed in the *Resources* folder, and that a *PhotonView* component is added to the *GameObject*. A *PhotonView* component on a GameBbject synchronizes desired data of the GameObject over the network.

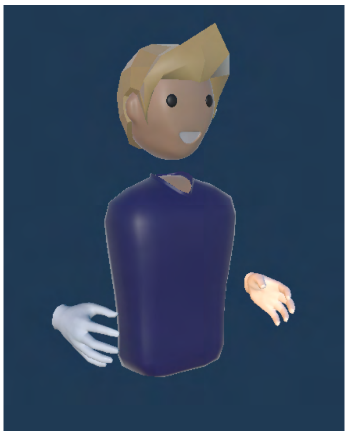


Figure 4.17: The avatar created by Kohncke, 2019 and its attached hands. These objects position and rotation is being sent over the Photon Network to other clients in the same room.

To create an interactive avatar, the body and hands of the avatar are created as child objects of the avatar GameObject. The body model is an avatar retrieved from Kohncke, 2019, while the hands are the same as described in section 4.2.4. The avatar with hands can be seen in Figure 4.17.

The next task is to make sure that the body and the hands of the avatar follow the headset and controller movement of the user. To make this happen, a new script is created and named *NetworkPlayer.cs*. First, the transform which holds the information about the position and rotation of the objects that are sent over the internet need to be accessed. The same applies to the transform of the headset and controllers. The transform of the headset and controllers can be found by first finding the XR Rig with the function *FindObjectOfType<XRRig>()*. Subsequently, the transform of the main camera, which is the head, and the base controllers, which hold the information about the transform of the controllers, can be accessed by finding the transform of the child GameObjects of the XR Rig by using *transform.Find()* function. This function can be used to find specific child or parent GameObjects within a GameObject, in this case the XR Rig. The head and controllers can be found by using exact string references holding the hierarchy information about the different GameObjects placement. This can also be achieved by applying tags to the GameObjects that are at interest and find them by using the *GameObject.FindGameObjectWithTag* method. When the transform of the headset and controllers are found, their position and rotation can be mapped over to the position and rotation of the body and hands of the avatar that is sent over the internet. This is done by creating the *MapPosition()* function which take both transforms as the input, and set the position and rotation of the body and hands of the avatar equal to the position and rotation of the headset and controllers. The *MapPosition()* function is then placed in the *Update()* method to have a continuous tracking of the device. To synchronize the position and rotation of the body and the hands over the network, the *Photon Transform View* component have to be added to all the transform parameters for the head and hands. By doing this, the *PhotonView* component in the root GameObject will automatically detect that it must send the transform of the GameObject that the component is attached to over the network.

To synchronize the hand animation presented in section 4.2.4, the *Photon Animator View* component must be added. As with the *Photon Transform View* component, the *Photon Animator View* component is also automatically detected to be sent over the network with the *PhotonView* component. Since the hand model prefab attached to the hand GameObject already contain the

Animator component responsible for the animation, this is already taken care of. The two values that are triggered by the two buttons seen in Figure 4.12b are responsible for the animation need to be set up. First, the two values have to be sent over the network by setting the parameters to *continuous* in the *Photon Animator View* component. Secondly, both buttons values have to be sent over the network and update the hand animation of the avatar accordingly. The float values from the buttons are thus polled and the animation is updated in the *UpdateHandAnimation()* function inside the *Update()* method.

However, since the *NetworkPlayer* is sent over the internet using *PhotonView*, multiple copies holding the information about the position, rotation, and animation of the other clients are created. This means that there are multiple *NetworkPlayers* in the scene. Since it is desired to only change the position of the avatar that belongs to a single client, it must be checked if the *PhotonView* component belongs to the client. This can be checked by utilizing the *PhotonView.IsMine* property.

4.7.5 Collaboration functionality

A key aspect of the network functionality is to be able to communicate with the other users present in the same room, and interact with the same objects. To be able to interact with objects in a scene, the same method described in section 4.2.4 are added to the objects. However, since object interactions are supposed to be sent over the network, changes has to be made. To synchronize the position of the object over the network, an *PhotonView* component have to be added to the object. To track the position and rotation of an object, the *Photon Rigidbody View* component is added. Using this component, the velocity and angular velocity of the object can be synchronized over the network. Additionally, to be able to move an object over the network, a client has to be the owner of the object. This can be achieved by changing the *XRGrabInteractable.cs* script and make the *PhotonView* component have its ownership transfer from *Fixed* to *Takeover.* By making a new script called *XRGrabNetworkInteractable.cs* and make this script inherit from the *XR Grab Interactable* class, the ownership of the object can be changed over the network. When a script inherit from the *XR Grab Interactable* class, all the functionality and methods generated from this class is transferred to the new script. By overriding the *OnSelectEntering()* script, which is called when an object is grabbed, the ownership of the object can be requested by calling *PhotonView.RequestOwnership().*

To add voice chat functionality enables communication with other clients present in the same room, another package from Unity Asset Store has to be imported. This is the *Photon Voice 2* package which is free. After the package is imported, a new application on the Photon website has to be created. This application is of type *Photon Voice*, and after it has been created an application identification (AppID) is provided. This AppID is then put into the *App ID Voice* field in the *Photon Server Settings.*

To be able to send sound over the network, a new GameObject called *Network Voice* is created. To enable voice chat, a *Photon Voice Network* component is added to the *Network Voice* object. This component requires a *Recorder* component, so this component is also added to the same GameObject. It is important that the *Recorder* component has the *Transmit Enabled* option checked to true, so that audio transmission is enabled. Each client needs to have a way of transmitting voice over the network. This is achieved similarly to what was done with the *PhotonView* component, however this time the *Photon Voice View* component provides the functionality. This component is then attached to the *Network Player* object which is the root GameObject of the avatar that is being sent over the network. The *PhotonVoiceView* component requires both a speaker and a recorder. Since the recorder has already been created, the *Use Primary Recorder* option can be checked. A speaker has to be added. This component can be added to the *Network Player* object as well. By adding both a *Speaker* component and an *Audio Source* component to the *Network Player*, voice can be transmitted over the network. To create an immersive experience for the clients connected to the same room, 3D voice can be added. This is done by changing the *Spatial Blend* from 2D to 3D, resulting in spatial attenuation being enabled. The final step to remember is to add the *Audio Listener* component to the XR Rig, which makes it possible to be able to hear audio that is transmitted over the network. This component is added to the *MainCamera* object that serves as the head of the XR Rig.

4.8 Scene downloads

If the application is to be used by multiple users each surveying different environments, the file size of the application will quickly become large. A usual model generated from a LiDAR scan takes up between 200 and 1500 MB depending on the scan resolution and the size of the environment being scanned. When a model is exported to Unity as an *.obj* file, this file size is significantly reduced. When Unity builds a project, project files can be further compressed by LZ4 or LZ4HC compression which significantly reduces the applications size. The difference between the two compression methods are their speed and compression rate. LZ4 is faster than LZ4HC, but LZ4HC has a higher compression rate. If enough scans are added, the application will still have a significant file size. To help reduce file size, the application should allow virtual environments to be downloaded from within the application. The maximum size of the application being built on the Oculus Quest is 1 GB (Oculus, 2021b). This emphasises the importance of being able to reduce the file size of the application, and being able to download additional scenes through extensions. By uploading pre-built and compressed Unity scenes, users can download the environments they need which significantly reduces the total file size of the application. Since the uploaded Unity scenes will be compressed, the download size for the individual scenes will be reasonably small and mainly less than 100 MB.

4.8.1 Creating Asset bundles

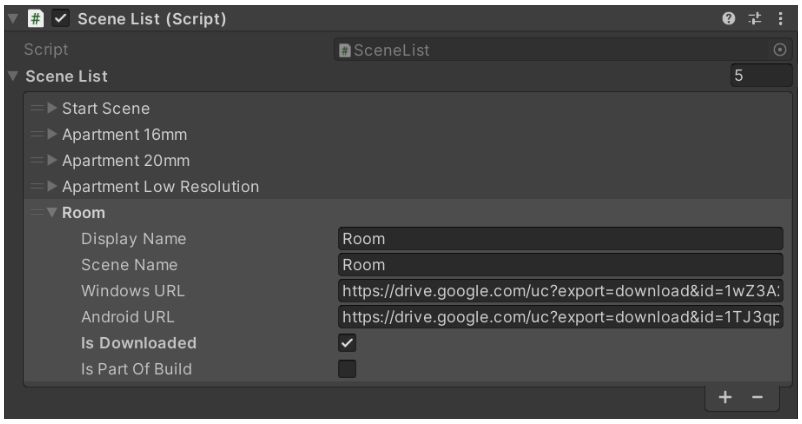
An asset bundle is is an archive file that contains platform-specific non-code Assets such as Models, Textures, Prefabs, and entire scenes that Unity can load at run time. In order to be able to download and run a scene at runtime, the downloaded scene must be a pre-built executable file. Because the downloaded file has to be executable, the downloaded file must be built for the specific target downloading and running it. Therefore, a PC and an Android based device such as the Oculus Quest 2 requires two different versions of an assetbundle in order to be able to run it. Since the Oculus Quest 2 can be run either standalone or via a connected PC (discussed in further detail in section 3.2), an asset bundled scene must be built two times, one for each platform.

In order to provide a quick way to pre-build Unity scenes, a Unity editor extension is implemented with a script called *CreateAssetBundle.cs*. The *CreateAssetBundle.cs* script sets up an assetbundle directory within the Unity environment and uses the Unity *BuildPipeline* class to build an assetbundle of a specified object. The *BuildPipeline.BuildAssetBundles()* function enables a user to specify build target, stored directory and other asset bundle options for an assetbundle. Once the *CreateAssetBundle.cs* script is implemented, it is possible to create assetbundles of desired objects directly in the Unity inspector. After a desired environment is pre-built for both platforms and placed in two individual assetbundles, they are ready to be extracted from the Unity project. The assetbundles can then be uploaded to any online platform like Google Drive or Azure. By providing the direct download link to the uploaded assetbundles, it is possible to access the uploaded scenes.

4.8.2 Scene structure

To keep track of which scenes are available for download online and which are already part of the application build, changes are made to the *SceneList.cs* script and *SceneList* object. Instead of just storing the scene name, a scene object in the *SceneList.cs* script now also stores the download URL for both the Android and PC version of the assetbundle and two boolean variables indicating if the scene object is part of the application build and whether it is downloaded or not. An example of a public scene object in the sceneList which is visible in the Unity inspector window can be seen in Figure 4.18.

Three new functions are implemented in the *SceneList.cs* script. These are *CheckAssetBundles()*, *CheckBuiltScenes()* and *RefreshSceneList()*. The *CheckAssetBundles()* function iterates through the *SceneList* and constructs a filepath for each scene which corresponds to the filepath where that scene would be stored if it was downloaded. If the function finds that the filepath exists, it means that the selected scene has in fact been downloaded and is therefore not a part of the



Scene List (Script)

Script SceneList

Scene List 5

Start Scene

Apartment 16mm

Apartment 20mm

Apartment Low Resolution

Room

Display Name	Room
Scene Name	Room
Windows URL	https://drive.google.com/uc?export=download&id=1wZ3A
Android URL	https://drive.google.com/uc?export=download&id=1TJ3qp
Is Downloaded	✓
Is Part Of Build	

Figure 4.18: A Scene object in the public Scene List. A Scene object has both a display name and a scene name in order to be able to display a different name for a scene in the menu than what its name string reference is.

build. The *isPartOfBuild* boolean variable for that scene is therefore automatically set to false and its *isDownloaded* variable is set to true. If the filepath does not exist, the scenes *isDownloaded* property is set to false. The *CheckBuiltScenes()* function gets the string name of all scenes in the application build. The function then iterates through the scene list and compares them to the string names of the application in the build. If the names match, it means that the selected scene is part of the application build. It's *IsPartOfBuild* variable is then set to true. If no match is found for a scene, its *IsPartOfBuild* variable is set to false. The *RefreshSceneList()* function is used to get an updated status on the variables of the scenes in the scene list. In order to do this, the *RefreshSceneList()* function runs both the *CheckAssetBundles()* and the *CheckBuiltScenes()* function.

After the *RefreshSceneList()* function has been run, a scene in the scene list is in one of three different states. For each state a different action must be taken. The first state is if a scene is part of the application build. If this is the case, no new action has to be taken. It's button icon will be generated as before in the scene selection menu view and it will be loaded as before when it is selected. The second state a scene can be in is if it is not a part of the build and it is not downloaded. If this is the case, the scene button icon in the scene selection menu view should have a different color indicating that it is unavailable and have a button next to it that should trigger a download for that scene. The third state a scene can be in is if the scene is not a part of the build but it is downloaded. If this is the case, the scene button icon in the scene selection menu view should look similar to the ones that are part of the build. In addition, it should have a button next to it that can be pressed in order to delete the downloaded version of the scene. If a scene is in the third state and a user wants to load it, it can not be loaded the same way as scenes that are part of the build since a downloaded scene is stored in a separate folder outside the built application file. The different scene button icons generated depending on scene state can be seen in Figure 4.19.

4.8.3 Downloading exported scenes

The *SceneList* object with the *SceneList.cs* script will serve as an extra security measure by providing access control. By manipulating the *SceneList* object of the application, one can easily decide which scenes should be available to different users. If for instance two different customers wants to use the application, they will both be able to use the same features but have access to completely different scenes depending on their version of the *SceneList* object in their application. The *SceneList* object can therefore help protect sensitive scans by only providing access to them for approved users. In addition, the *SceneList* object makes it easy to add or remove scenes from an existing application build.

When a user presses the download icon on a scan in the scene selection menu view, the icon will call a function in *AvailableScenesManager.cs* called *DownloadScan()* which initiates a coroutine.

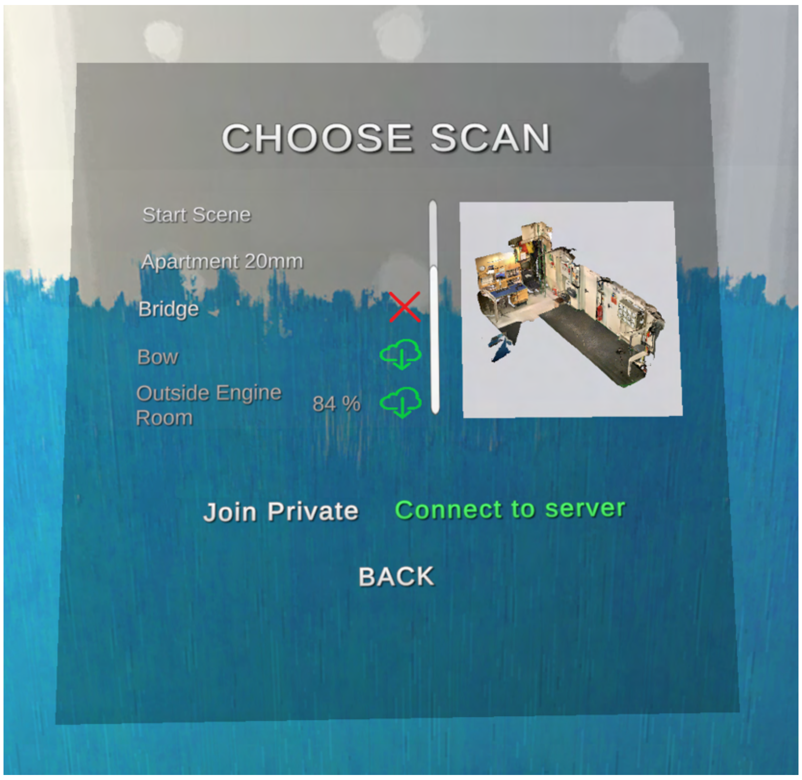


Figure 4.19: The three different generated button types. Scenes part of the build have no icon next to them. Scenes available for download have a download icon next to them which display a download progress text field when pressed. Downloaded scenes can be deleted by pressing the red cross icon next to them.

A coroutine (discussed in section 2.4.2) is a function that can suspend its execution (yield) until the given yield instruction finishes. The coroutine stores the string name of the scene whose download button was pressed. It then compares the stored string to the scene list. When the coroutine finds a match, it will run the *Application.RuntimePlatform()* function which returns the platform the application is running on. The coroutine then creates a GET *UnityWebRequest()* to either the Android or PC download URL of the scene, depending on the result from *Application.RuntimePlatform()*. A *DownloadHandler* object is added to the Web request to define how it should handle the HTTP response body data received from it. The GET request is then sent and the coroutine suspends its execution until it gets a result from the request. When the coroutine gets a result it runs a *Save()* function which creates a directory on the device running the application and uses the *File.WriteAllBytes()* function to store the received data. All downloaded scenes will be stored at the path *Application.persistentDataPath/AssetData/**sceneName**.unity*. The *Application.persistentDataPath* is a read only string that contains the path to a persistent data directory. The filepath of a downloaded assetbundle containing a scene therefore only depends on the name of the scene. After the assetbundle is downloaded, the *RefreshSceneList()* is run to update the status of the downloaded scene in the application scene list. New icons for the scene buttons in the scene selection menu view is then generated to display the changes using the *GenerateMenuIcons()* function.

In order to provide user feedback regarding the download status, a new coroutine called *DownloadProgress()* is created. It is called just before the GET *UnityWebRequest()* is sent. The coroutine takes the web request as an input and displays its download status in a text field on the scene button icon while it is downloading.

4.8.4 Loading and deleting assetbundled scenes

If a user wants to load a downloaded scene, it is not possible to do so using the method used on scenes that are part of the application build. This is because the downloaded scenes are stored in a separate directory outside of the built executable file that is the application. In order to load a downloaded scene, a coroutine called *LoadObject()* is implemented. The *LoadObject()* coroutine takes the scene name of the scene to be loaded as an input. It recreates its filepath from the scene name and starts loading it using the *AssetBundle.LoadFromFileAsync()* function which loads the assetbundle asynchronously. The assetbundle is loaded asynchronously to avoid the application freezing while the assetbundle is being loaded. The *LoadObject()* coroutine suspends its execution until the load is complete. Once the assetbundle is loaded, the *LoadObject()* coroutine extracts the scene from the assetbundle using the *GetAllScenePaths()* function which returns the scene asset paths within the assetBundle. The scene can then be loaded using the *SceneManager.LoadScene* function.

To delete a downloaded scene, a coroutine called *DeleteScan()* is implemented. When a user presses the delete icon on the scene button in the scene selection menu view, *DeleteScan()* is run. The coroutine uses the string name of the scene whose delete icon was pressed to recreate its file path. The *File.Delete()* function is then run which deletes the file at its input file path. To remove the assetbundle content that has been cached by the application, the *Caching.ClearCache()* function is run. After the assetbundle containing the scene has been deleted, the *RefreshSceneList()* function is run to update the status of the deleted scene in the application scene list. New icons for the scene buttons in the scene selection menu view is then generated to display the changes.

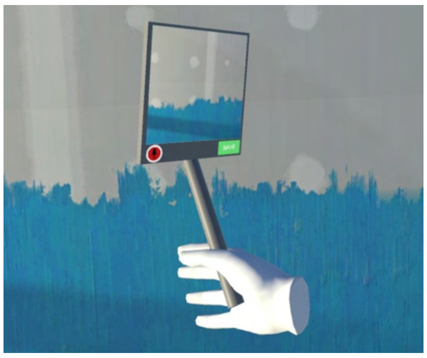
4.9 Documentation tool

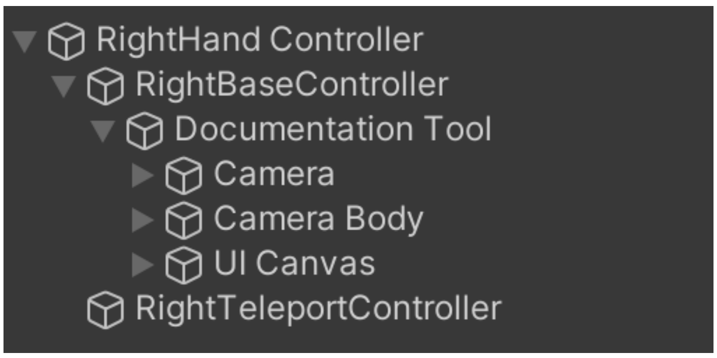
An important part of a survey is to be able to document findings so that they can be included in a report. To make documentation easier, a virtual camera is implemented in the application. As a mean to provide instant thoughts about findings, a recording device is also implemented. A user should have the option to save both an image and its audio recording to the device running the application, so that they can be reviewed later.

The documentation tool is thought to have the best visual appearance if it is a camera that is connected to the hand of a user via a stick as seen in Figure 4.20a. It is placed on the *Right-BaseController*. By doing this, the documentation tool will follow the position and rotation as the *RightBaseController*, meaning the right physical controller. The hierarchy of the objects that make the documentation tool can be seen in Figure 4.20b. In the middle of the stick's tablet is a *Camera* object that is able to take high-resolution images. Another *Camera* at the same position is able to take low-resolution images. The thought is that the low-resolution camera can capture images that is displayed on the tablet's screen in Figure 4.20a, much like a video camera. This is to aid the user to take good images, as the user is able to see the camera output on the tablet screen. The video is only captured once the camera trigger is pressed, and once the trigger is released a high-resolution image is taken by the high-resolution *Camera*. If the video camera were taking high-resolution images and displaying them on screen it would be too computationally expensive to render the images several times each second. Therefore, the tablet will only display the image captured by the low-resolution *Camera*.

4.9.1 Image capture

To provide logic for the documentation tool, the *MediaManager.cs* script is created. Since the documentation tool is attached to the hand of the XR Rig, the *MediaManager* script can be added to the root GameObject of the XR Rig. This is to easily locate where the script is attached, but the script could also have been attached to the *Documentation Tool* GameObject as seen in Figure 4.20b. In the script, the resolution of both the camera and the video camera is set to 1080x1080 and 128x128 pixels respectively in the *Start()* function. Then the *Texture2D* of both


(a) The documentation tool placed inside the right controller.


RightHand Controller
RightBaseController
Documentation Tool
Camera
Camera Body
UI Canvas
RightTeleportController

(b) The hierarchy of the Documentation Tool placed in the RightBaseController.

Figure 4.20: The hierarchy and visual representation of the documentation tool.

the captured camera and video image are set to the same resolution.

In the *Update()* function, the documentation tool is toggled on and off by pressing the *Secondary button* on the right controller. Once the camera is present, the *TakeImage()* function is called. In this function the video and image gets taken and displayed on the screen. The video only gets taken once the *Trigger* button on the right controller is pressed, and once it is released a high-resolution image is taken and displayed on the screen.

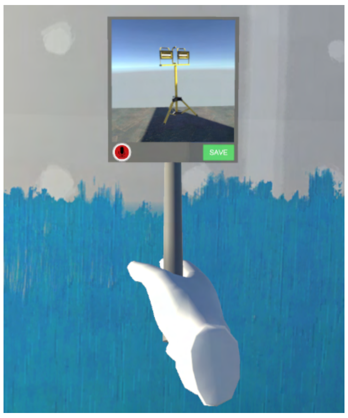
4.9.2 Audio record

To record audio, an *Audio Source* component must be present. This component was already attached to the XR Rig when implementing network functionality. In the *MediaManager.cs* script's *Start()* function a reference to this *Audio Source* component is achieved by calling *GetComponent<AudioSource>()*. This function assumes that an *Audio Source* component is attached to the same hierarchy layer as the script calling it. Once a user presses the microphone symbol seen in Figure 4.21a, an audio clip is captured. This is done by utilizing the *Microphone.Start()* function, which starts to record an audio clip as seen in Figure 4.21b. However, the length of an audio clip has to be set before it is started. Therefore a length of 200 seconds is set, and when the stop record button is pressed as seen in Figure 4.21c, the audio clip is stopped by calling *Microphone.Stop()*. After the recording is stopped, the resulting audio-clip is trimmed. Trimming the audio-clip is done by getting the length of the recording just before the it is stopped by calling *Microphone.GetPosition()*. A new audio-clip with a duration of *Microphone.GetPosition()* is then created. The trimmed audio clip is then set as the audio clip for the *AudioSource*.

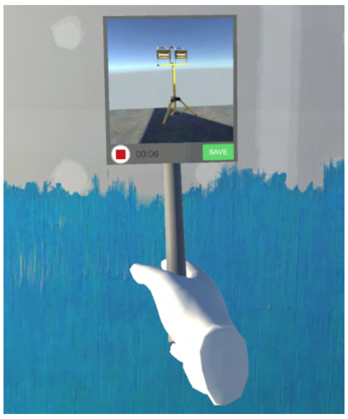
The length of the recording is also tracked visually through a timer as seen in Figure 4.21b - Figure 4.21d. This is a *Stopwatch* that is started once a recording starts. It is updated in the *Update()* function of the *MediaManager.cs* script through the *Stopwatch.Elapsed* property. This stopwatch is stopped once the recording stops. The audio recording can also be played in the application as seen in Figure 4.21d by using *audiosource*.Play() where *audiosource* is the *Audio Source* attached to the XR Rig.

4.9.3 Saving media

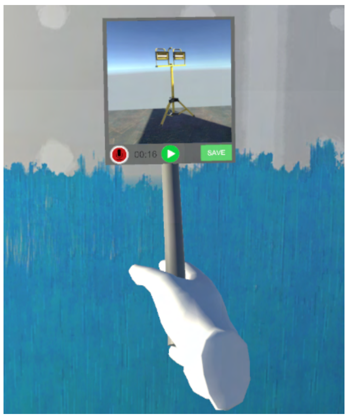
To save both the image and audio clip different methods are used. The image is saved as a PNG file, while the audio clip is saved as a WAV file. The image is first encoded to an array of bytes through the *EncodeToPNG()* function which can be called from a *Texture2D* object. The audio clip is saved by calling *SavWavSave()*, while the image is saved by creating a method called *SaveInDirectory()*. Both methods is dependent on a filepath being provided. The *SaveInDirectory()* method first creates a new directory if it does not already exist. Then, the bytes that represent the PNG image



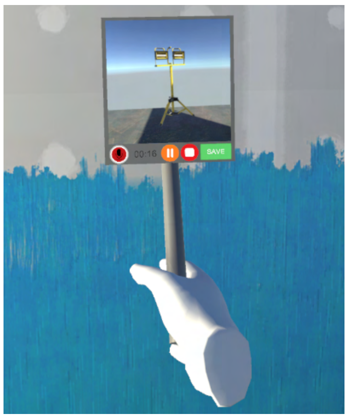
(a) The documentation tool after a picture is taken and before an audio record is started.



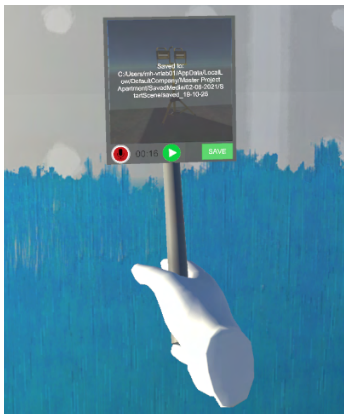
(b) The documentation tool during an audio record.



(c) The documentation tool after an audio record is taken.



(d) The documentation tool when the audio recording is played.


Saved to:
C:/Users/mh-vrlab01/AppData/LocalLow/DefaultCompany/Master Project Apartment/SavedMedia/02-06-2021/StartScene/saved_18-10-26

(e) The documentation tool showing the save path of the image and audio record on the device after *Save* is pressed.

Figure 4.21: Image series showing the different states of the documentation tool when a image is taken and an audio record is captured.

gets written to the directory path. This is done using *File.WriteAllBytes()*, and providing the directory path and array of bytes from the image.

A method to provide a filepath independent of which device the application is running on is by utilizing the *Application.persistentDataPath* method. By using this method a path to a persistent data directory which is a public directory on the device is obtained. The media is then placed inside another folder called *SavedMedia*. After this, the media is sorted by the day, month and year the media is captured, before it is sorted regarding at what scene the media was captured. At last, the media is placed inside a folder describing at what time it was captured. The file path that the media is saved at can be displayed on the screen of the documentation tool as seen in Figure 4.21e.

4.10 Final changes

Once all features are created, different measures are taken to simplify the process of implementing new scenes. In order for all implemented features to work in a scene, several *GameObjects* are required. The required *GameObjects* are:

- **XR Rig:** The collection of GameObjects receiving user input from controllers and the VR headset. An XR Rig is required in order for a user to move around and interact with the environment.

- **Menu Canvas:** The canvas holding all UI elements for the menu. This GameObject is required in order for the menu to be displayed in a scene.
- **Measurement Canvas:** The canvas holding all UI elements for the measurement tool. This GameObject is required in order to display the distance of a distance measurement in a scene.
- **Database Manager:** The Database Manager GameObject has the *DatabaseConnector.cs* script as a component which has the VESSEL_ID as a public variable. This GameObject is required for the scene to be able to connect to a database and retrieve information to all Points of Interest (POIs) in the scene based on the primary key VESSEL_ID.
- **Network Voice:** The Network Voice GameObject has a Photon Voice Network component. This GameObject is required in order for users to communicate via voice chat in a scene when network functionality is enabled.
- **XR Interaction Manager:** The XR Interaction Manager has an XR Interaction Manager Unity script as a component. It is required to enable interactions between all interactors and interactable objects in a scene.
- **Eventsystem:** The EventSystem GameObject has an Event System component and a Unity XRUI Input Module attached. It is required in order to manage user inputs properly in a scene.
- **SceneList:** The SceneList GameObject has the *SceneList.cs* script as a component. The SceneList GameObject is required in order for the current scene to keep track of other scenes statuses and availability.
- **Options Information:** The GameObject that holds information about a user's turning preferences across the scenes of the user. The GameObject is destroyed if another instance of the object is already present in a scene.

A GameObject called *Essential Assets* is created and prefabed in order to simplify the implementation of new scenes. The *Essential Assets* GameObject has all the above required GameObjects as child objects. This means that when a new scene is implemented, *Essential Assets* is the only GameObject required in a scene in order for all functionality to be enabled.

All necessary steps required for a new scene's virtual environment to be fully functional and ready to use is listed below:

1. Create a new Unity Scene and import the 3D model generated from a LiDAR scan into it. Add a mesh collider component to it to make it behave like a physical object and disable global illumination and shadows on the models mesh renderer in order to make it computationally cheaper to render.
2. Drag and drop all desired Image Anchors into the scene and add their corresponding 360° images as textures. Remember to rotate the image texture according to the Image Anchors surroundings to make their transitions smooth.
3. Drag and drop all desired POIs into the scene. Remember to set each POI objects TYPE variable to decide what type of information the individual POIs should display.
4. Add an *Essential Assets* prefab to the scene. Remember to set the VESSEL_ID of the Database Manager to decide which vessel the scene's POI information should be retrieved from.

These are all steps needed to set up a new virtual environment. By following these steps when creating a new scene, all features implemented in this chapter are enabled and fully functional in the new environment.

Chapter 5

Results

In this chapter, material gathered with the utilized 360° camera and LiDAR sensor will be presented. The resulting application implemented in the Unity engine will be shown. A video demonstrating the material gathering process and the developed applications functionality can be seen by clicking the following link: https://youtu.be/XkmnMDrFR0s or by clicking on the following hyperlink: YouTube. Hyperlinks to specific parts of the video demonstrating the different results will also be provided throughout the chapter. These hyperlinks will start the video from a designated timestamp and have their duration mentioned in text. It is therefore important to make a note of this when clicking the hyperlinks, as the video segment specified is important for the topic providing the link.

Since the developed application is such a visual product, a video is thought to be the best option to demonstrate the achieved results. The video will demonstrate general controls of the application and the implemented tools. In addition, the video will demonstrate the applications functionality by exploring a scan of an apartment and several scans taken aboard a vessel. The video is considered as an essential part of the results chapter.

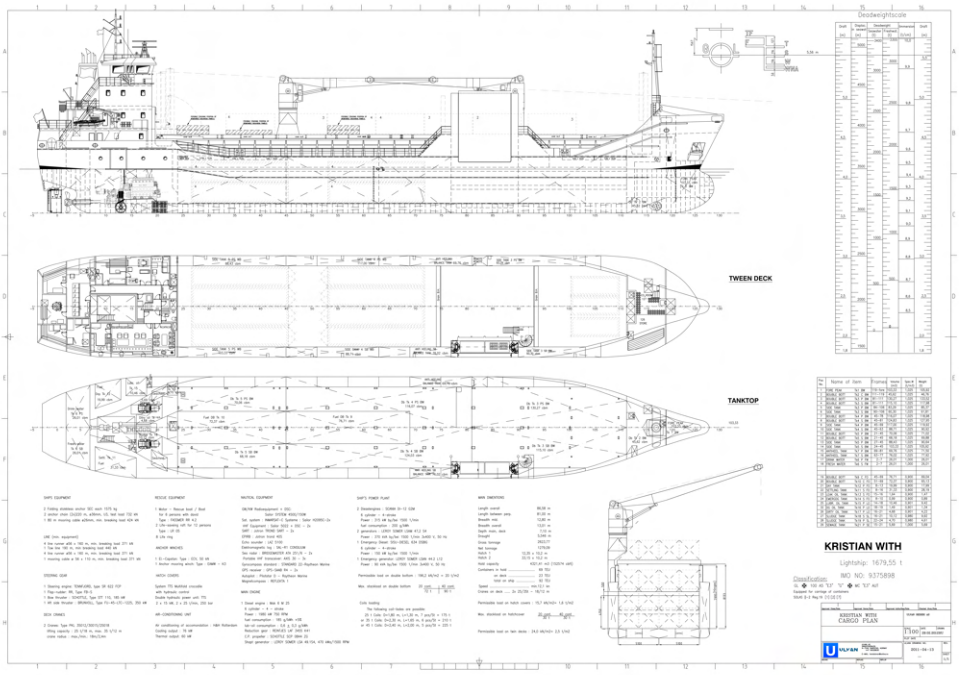
5.1 Modeling

A GoPro Max camera and an iPad Pro 2020 model was used to gather the material the virtual environments were generated from. The equipment has been tested in several conditions and in different environments. As the application is aimed at vessel surveys, parts of a general cargo vessel was scanned and imported into the application. This section will first present general information about the vessel that were scanned in subsection 5.1.1. The general results from the material gathering process is presented in subsection 5.1.2 and subsection 5.1.3.

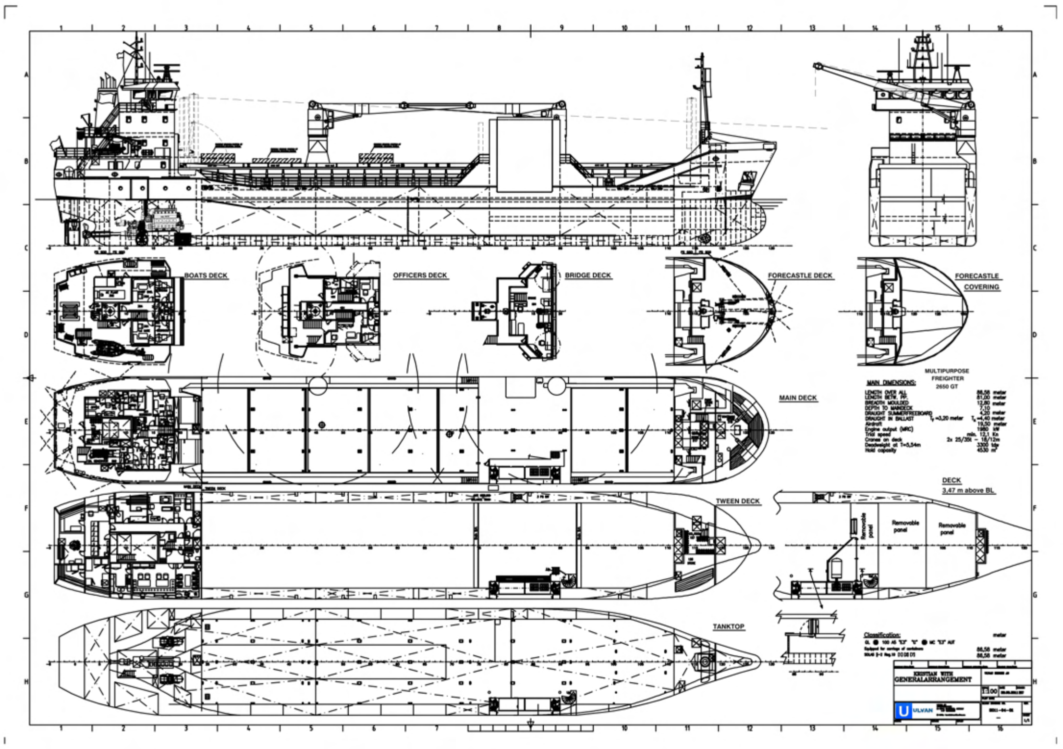
5.1.1 MS Kristian With

To test the capabilities of the developed application and scanning methods used, parts of a vessel is scanned and imported into the application. After a meeting with Stener Olav Stenersen at DNV, contact with Egil Ulvan Rederi was established. Egil Ulvan Rederi is a Norwegian shipping company with a total of 7 vessels. One of their vessels are MS Kristian With. Kristian With is a general cargo ship and is about 86.5 meters long and 12.8 meters wide as seen from its general arrangements drawings in Figure 5.1. After contacting Egil Ulvan Rederi, the enquiry of boarding Kristian With was accepted. The process of being able to board a ship would normally be simple, however the COVID-19 pandemic made this difficult to arrange. Thankfully permission to board and scan Kristian With were given. Before the vessel was boarded, a scan and prototype virtual environment of an apartment was created. scans of Kristian With was performed on the 6th of May 2021 while Kristian With was at bay in Trondheim.

In a meeting with Per Haveland and Bjarne Augestad (section B.5), both senior surveyors at Gard, the professional surveyors recommended scanning the vessels bridge, engine room and parts of the deck in order to test the capabilities of the application, as those areas are important to examine for a surveyor.



(a) Tank plan and capacity plan of Kristian With.



(b) General arrangement drawing of Kristian With.

Figure 5.1: General arrangements drawing of MS Kristian With.

The LiDAR scans and 360° images were taken of the vessel and exported to Unity. The vessel could then be explored in the application. A demonstration of the material gathering process, the application's functionality and the vessel scans being explored can be seen in the attached video. As seen in the video, all implemented objects and tools work as expected and the overall quality of most environments are satisfactory.

5.1.2 LiDAR

The LiDAR sensor on an iPad Pro 2020 model was utilized to perform scans of different environments. The '3d scanner app' by Laan Labs was then used to convert the generated point cloud into a textured 3D model which could be exported as a *.obj* file into Unity. The same LiDAR sensor is also available in the iPhone 12 Pro series. The captured results from Kristian With can be seen in Figure 5.2, while the scans taken from the apartment with different resolutions can be seen in Figure 5.3. As seen in the figures, ceilings are not present in the resulting LiDAR models. This is because it was concluded that the ceilings were not of interest and they were therefore not scanned.

5.1.3 360° images

The GoPro Max was used to capture 360° images. The captured photos supplements the LiDAR model in the virtual environment and provides a higher level of detail than what is possible with the LiDAR sensor. After experimenting, it is found that the best results are achieved when the camera is held between 160-170cm above floor level. This camera height gives the most realistic switch between LiDAR scan and image when a user is standing while using the application. One instance of a 360° image being triggered in the application can be seen in the following part of the video from 22:24 - 22:41. 360° images captured from different environments can be seen in Figure 5.4.

5.2 Application requirements

In subsection 3.1.2, a number of requirements for the application was set. For the application to be regarded as successful, the set criteria had to be met. This sub-chapter presents results from the application for each of the requirements.

High level of detail

In order for a surveyor to be able to accurately inspect a vessel, the virtual environments had to be presented with a highly realistic model of the scanned vessel. Scans taken for the application generally holds a high level of detail and are highly accurate. An example can be seen in the following part of the video from 31:13 - 31:38. In situations where the quality of the LiDAR scans are moderate because of complex environments, the 360° photos supplement the scans and provide a high level of detail. The combination of LiDAR scans and 360° photos is therefore regarded as successful in providing a high level of detail in the virtual environments.

Collaboration

Collaboration functionality was implemented to enable multiple users to explore the same areas in VR while communicating. A surveyor is always accompanied by a member of the vessel crew when conducting a physical survey. The application successfully enables collaboration functionality by sending data about users online so that they can see each others position in real time and communicate via voice chat. Being able to point at objects and explain findings via voice chat is

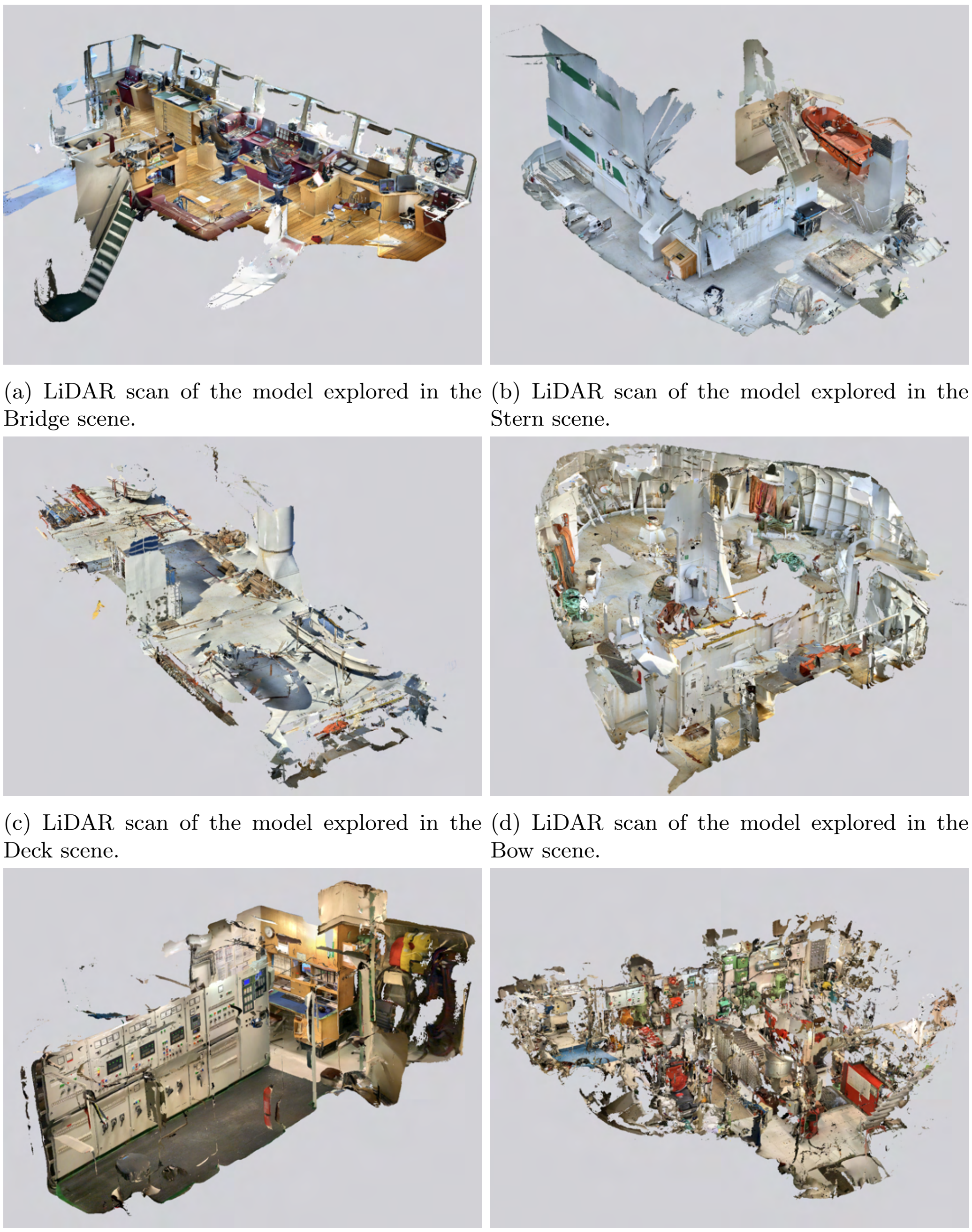

(a) LiDAR scan of the model explored in the Bridge scene.
(b) LiDAR scan of the model explored in the Stern scene.
(c) LiDAR scan of the model explored in the Deck scene.
(d) LiDAR scan of the model explored in the Bow scene.
(e) LiDAR scan of the model explored in the Outside Engine Room, i.e. control room scene.
(f) LiDAR scan of the model explored in the Engine Room scene.

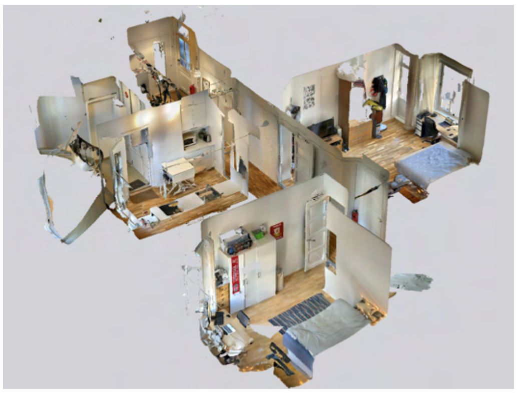
Figure 5.2: Models generated from different LiDAR scans taken aboard Kristian With.

useful when conducting surveys, and the implemented functionality works as expected. An example of how collaboration is used can be seen in the following part of the video from 23:08 - 24:20.

Documentation

A surveyor is continuously documenting his findings when conducting a physical survey. A documentation tool was therefore implemented in the application. The tool enables users to successfully documents their findings in the application by using a virtual camera to take photographs. A sound recording can be attached to each photo and then stored on the device running the application. After a virtual survey is conducted, a user can extract all photos and sound recordings from their

(a) LiDAR scan of the model in the Apartment Low Resolution scene.


(b) LiDAR scan of the model in the Apartment 20mm scene.

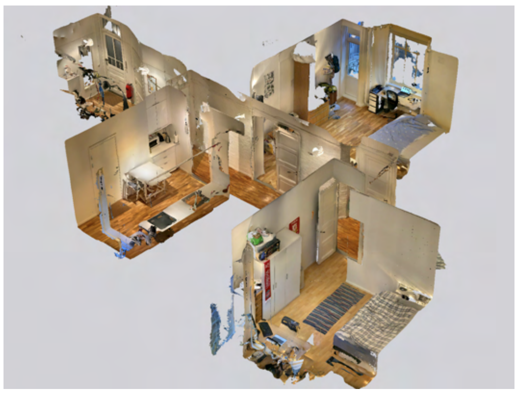

(c) LiDAR scan of the model in the Apartment 16mm scene.

Figure 5.3: The LiDAR model scans taken of an apartment with different resolutions.

device and add them to the report to be filed. The implemented documentation tool works as expected and is a useful resource for capturing documentation in the virtual environment. Use of the documentation tool is for instance demonstrated in the following part of the video from 30:24 - 30:51.

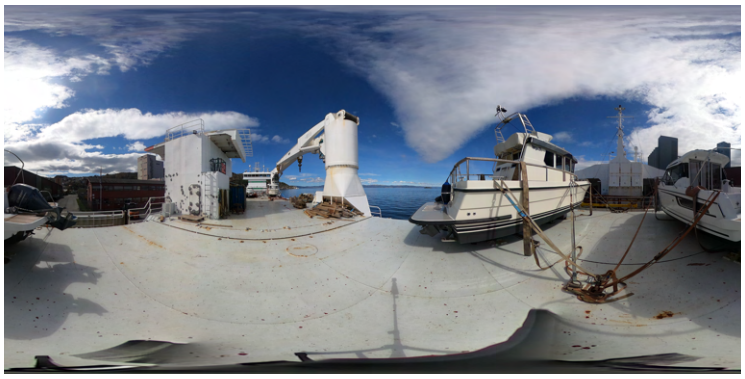
Cost saving

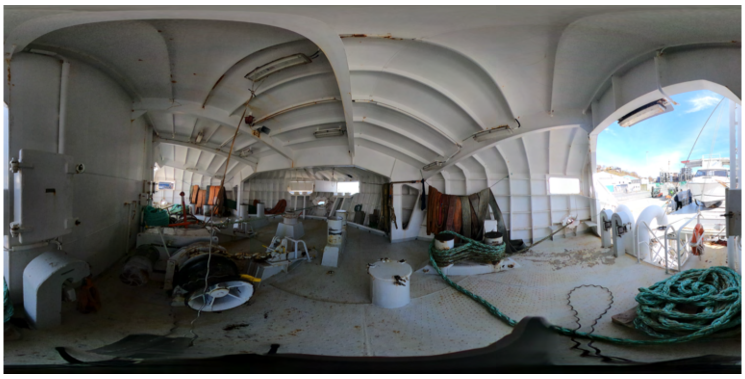
No further expenses than an iPhone or iPad with a LiDAR sensor and a GoPro Max camera were encountered when developing the project. If one member of the vessel crew already has an iPhone or iPad with a LiDAR sensor, the only expense required to be able to gather material for the application is the GoPro Max which costs between 400-500 USD. This is significantly cheaper than round trip plane tickets and accommodation for a surveyor conducting a physical survey. If Gard aqcuired several cameras, they could send a camera to a port where they know a specific vessel is heading. A crew member could then pick up the camera, use it and send it back at the next port. Over time this would significantly reduce costs, as one camera can be used by multiple vessels and crew just by paying the shipping costs of the camera. As a result, the developed application is considered successful when it comes to the cost aspect of it.

User friendliness

The developed application has virtual controller hints which can be accessed by users at any time to view what the different buttons on the controllers do. This can be seen in the following part of the video from 5:13 - 5:32. The menu system has a simple and intuitive user interface. A walkthrough of the menu system can be seen in the following part of the video from 13:35 - 15:01. Using hand

(a) 360° image from the bridge on Kristian With.


(b) 360° image from the deck on Kristian With.


(c) 360° image from the bow on Kristian With.

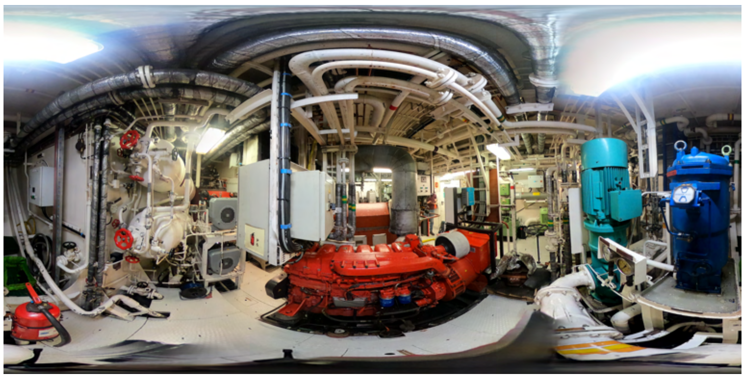

(d) 360° image from the engine room on Kristian With.

Figure 5.4: Several 360° images captured on Kristian With.

movement and physically moving or turning translates to the application which provides users with a natural way of interacting with the virtual environment. This can be seen in the following part of the video from 4:35 - 5:00. All things considered, the application is regarded as being user friendly.

5.3 Avoiding VR sickness

In order to avoid VR sickness (which is discussed in section 3.4), different measures are taken. The main factors causing VR sickness are unwanted camera movement, low frame rate and long VR experiences.

5.3.1 Control

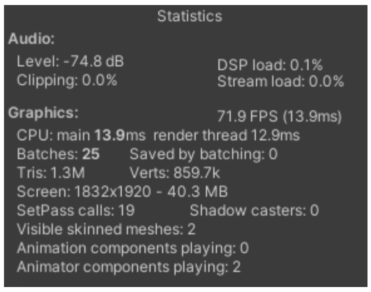
The application is implemented with two different methods for moving the camera. A user can either move and turn using the controllers as seen in the following part of the video from 5:32 - 5:54, or by turning and moving in physical space as seen in the following part of the video from 4:48 - 5:00. By allowing users to move both with controllers and physical movement, more natural interaction is achieved which minimizes VR sickness. Another element reducing VR sickness is that a users camera is never being controlled without their input. In addition, users can customize their controller turn rate and turn type in the "Options" menu view which allows users to find a comfortable turn rate and type which can reduce VR sickness. This can be seen in the following part of the video from 13:43 - 14:20.

5.3.2 Frame rate

Low frame rates can induce VR sickness, and as a result the frame rate of the application is kept as high as possible. When running applications on the Oculus Quest 2, the frame rate can reach a maximum of 72 FPS. To test how the frame rate of the application varies between different scenes, a test scenario is created. The test scenario will run the application and switch between multiple scenes with different rendering properties such as LiDAR model vertex size. The test scenario will start with the scenes with the lowest vertex size, and end at the most complex. This is to understand how LiDAR model vertex size affects frame rate. The rendering properties of the different scenes in the test can be seen in Figure 5.5. Here, the number of vertices and triangles in the different scenes can be found. A graph showing the timestamps of the different scene switches during the test can be seen in Figure 5.6. As seen from the figure, the frame rate drops every time a new scene is being loaded. It is also seen that the frame rate varies significantly from scene to scene. There are a clear correlation between scene vertex size and the achieved FPS. As the vertex size of a scene increases, the FPS of the application decreases.

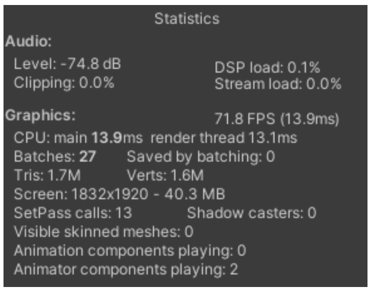
Another factor affecting the achieved FPS is the shader type used in Unity. As seen in Figure 5.7a, the shader type used has a big impact on the frame rate. Using a more complex shader than URP significantly decreases the achieved FPS. The graph data is generated by performing similar actions in the same scenes, and only changing the shader between the different runs. When using the standard shader, only three out of the six scenes could be loaded in the application. This is because an application file can only take up to 1 GB in file size when exporting applications to the Oculus Quest (Oculus Developer Center, 2021). When using URP, all six scenes could be loaded into the application. The application file is much larger when using the standard shader compared to using URP. As seen from these results, the shader type used to build the application has a massive impact on the file size and achieved FPS of the application which makes the URP optimal for this type of application.

As discussed above, the number of vertices rendered in a scene has a big impact on the frame rate as seen in Figure 5.7b. As seen from the figure there is almost a constant frame rate of 72 FPS when exploring models of up to 1.6 million vertices. When there is a slight increase of vertices up to 2 million, the frame rate is more unstable. This decrease in frame rate is an ongoing trend once the models are getting more complex. Increased scene complexity means that more vertices needs



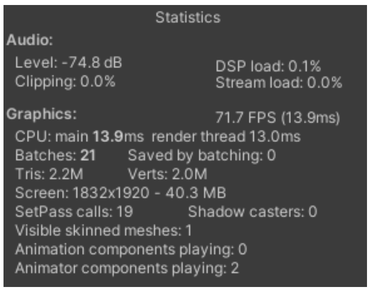
```
Statistics
Audio:
 Level: -74.8 dB                DSP load: 0.1%
 Clipping: 0.0%                 Stream load: 0.0%
Graphics:                       71.9 FPS (13.9ms)
 CPU: main 13.9ms  render thread 12.9ms
 Batches: 25     Saved by batching: 0
 Tris: 1.3M      Verts: 859.7k
 Screen: 1832x1920 - 40.3 MB
 SetPass calls: 19          Shadow casters: 0
 Visible skinned meshes: 2
 Animation components playing: 0
 Animator components playing: 2
```

(a) Apartment Low Resolution rendering statistics.



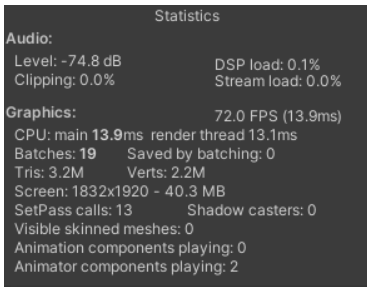
```
Statistics
Audio:
 Level: -74.8 dB                DSP load: 0.1%
 Clipping: 0.0%                 Stream load: 0.0%
Graphics:                       71.8 FPS (13.9ms)
 CPU: main 13.9ms  render thread 13.1ms
 Batches: 27     Saved by batching: 0
 Tris: 1.7M      Verts: 1.6M
 Screen: 1832x1920 - 40.3 MB
 SetPass calls: 13          Shadow casters: 0
 Visible skinned meshes: 0
 Animation components playing: 0
 Animator components playing: 2
```

(b) Deck rendering statistics.



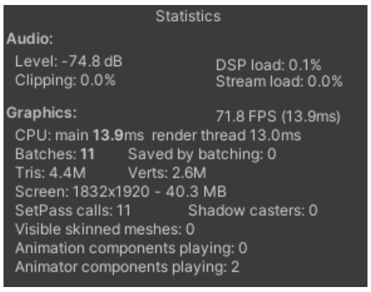
```
Statistics
Audio:
 Level: -74.8 dB                DSP load: 0.1%
 Clipping: 0.0%                 Stream load: 0.0%
Graphics:                       71.7 FPS (13.9ms)
 CPU: main 13.9ms  render thread 13.0ms
 Batches: 21     Saved by batching: 0
 Tris: 2.2M      Verts: 2.0M
 Screen: 1832x1920 - 40.3 MB
 SetPass calls: 19          Shadow casters: 0
 Visible skinned meshes: 1
 Animation components playing: 0
 Animator components playing: 2
```

(c) Engine Room rendering statistics.



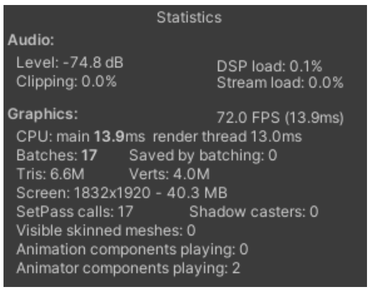
```
Statistics
Audio:
 Level: -74.8 dB                DSP load: 0.1%
 Clipping: 0.0%                 Stream load: 0.0%
Graphics:                       72.0 FPS (13.9ms)
 CPU: main 13.9ms  render thread 13.1ms
 Batches: 19     Saved by batching: 0
 Tris: 3.2M      Verts: 2.2M
 Screen: 1832x1920 - 40.3 MB
 SetPass calls: 13          Shadow casters: 0
 Visible skinned meshes: 0
 Animation components playing: 0
 Animator components playing: 2
```

(d) Stern rendering statistics.



```
Statistics
Audio:
 Level: -74.8 dB                DSP load: 0.1%
 Clipping: 0.0%                 Stream load: 0.0%
Graphics:                       71.8 FPS (13.9ms)
 CPU: main 13.9ms  render thread 13.0ms
 Batches: 11     Saved by batching: 0
 Tris: 4.4M      Verts: 2.6M
 Screen: 1832x1920 - 40.3 MB
 SetPass calls: 11          Shadow casters: 0
 Visible skinned meshes: 0
 Animation components playing: 0
 Animator components playing: 2
```

(e) Apartment 20mm rendering statistics.



```
Statistics
Audio:
 Level: -74.8 dB                DSP load: 0.1%
 Clipping: 0.0%                 Stream load: 0.0%
Graphics:                       72.0 FPS (13.9ms)
 CPU: main 13.9ms  render thread 13.0ms
 Batches: 17     Saved by batching: 0
 Tris: 6.6M      Verts: 4.0M
 Screen: 1832x1920 - 40.3 MB
 SetPass calls: 17          Shadow casters: 0
 Visible skinned meshes: 0
 Animation components playing: 0
 Animator components playing: 2
```

(f) Apartment 16mm rendering statistics.

Figure 5.5: Statistics about the different scenes being rendered from Figure 5.6. The most important information provided is the number of triangles (Tris) and vertices (Verts) for the scene.

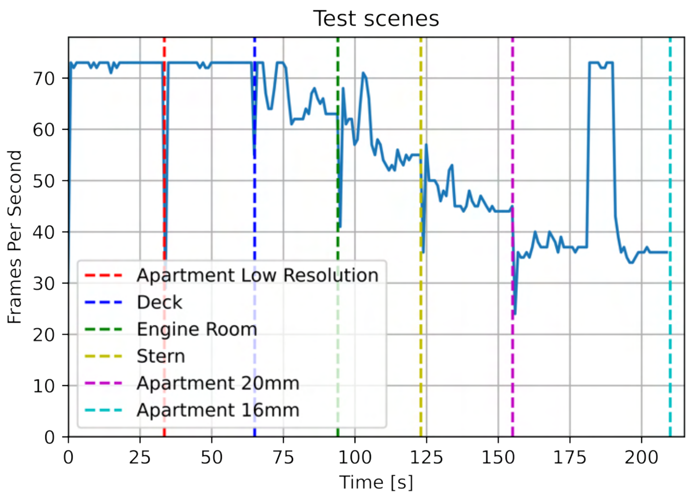
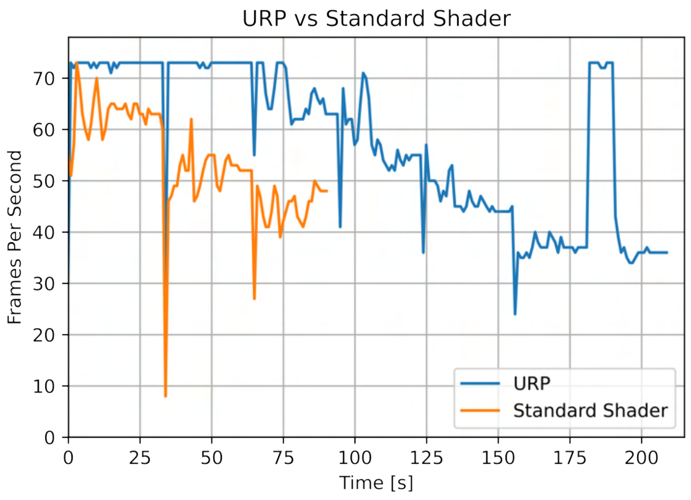


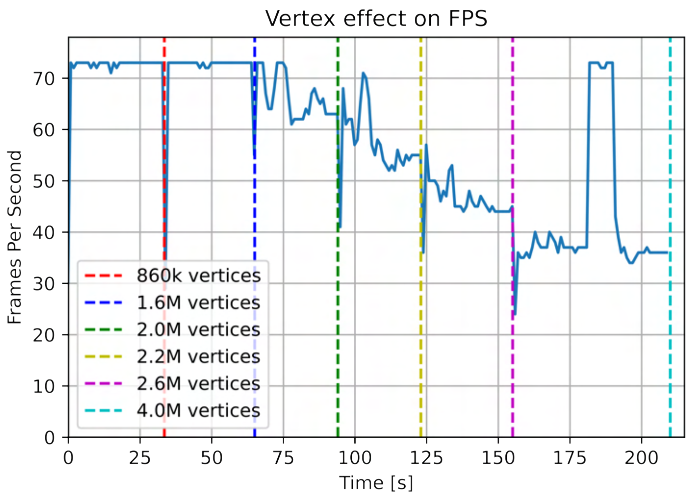
Figure 5.6: Scenes loaded to check their influence on the frame rate. The dotted lines separates the different scenes, where the first interval from 0-33 seconds show the FPS when inside the Apartment Low Resolution scene while the second interval from 33-65 seconds show the FPS when inside the Deck scene and so on.

to be rendered every single frame. In the last scene rendered, 4 million vertices are rendered in a single frame which drastically limits the FPS. However, there is an increase in frame rate from approximately 180-190 seconds. This is because an Image Anchor is toggled, which also disables the LiDAR model so that it is no longer rendered. As seen from the graph, disabling the LiDAR model when toggling Image Anchors significantly increases the achieved FPS.

When running the application externally from a computer, the frame rate drop experienced when using the Standard shader instead of URP (Figure 5.7a) is negligible. The frame rate is 72 FPS for every scene tested, even in scenes with LiDAR models with 4 million rendered vertices. This is because the application is ran from an external computer that is more powerful than the Oculus Quest 2.



(a) Graph comparing the URP shader against the Standard shader.



(b) Graph showing how the number of rendered vertices effect the FPS when running the application internally on the Oculus Quest 2.

Figure 5.7: How the shader used and vertex number of a scene influences the frame rate when running the application internally on the Oculus Quest 2.

In Figure 5.8 all the scenes from Kristian With is ran internally on the Oculus Quest 2, and the frame rate is recorded. As seen from the figure the frame rate is high at around 60-70 FPS.

5.3.3 Duration

Prolonged time spent in VR can induce VR sickness. To reduce the time spent in the VR for the application, a teleportation system is developed. By moving faster through large areas, time spent in the application can be reduced. This is for instance very useful on the deck scene, which is large in area compared to the other scenes. By teleporting, larger areas can be covered in a shorter time period. This is especially important when a current task is far away. An example of teleportation in the application can be seen in the following part of the video from 28:00 - 28:34.

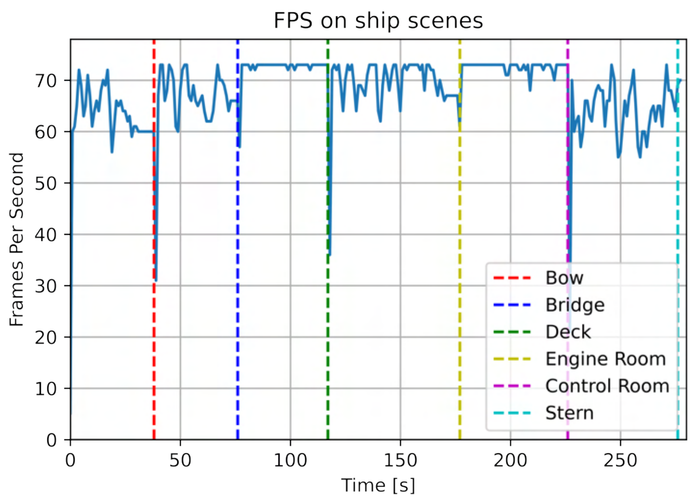


Figure 5.8: The frame rate when running the ship scenes of Kristian With internally on the Oculus Quest 2.

5.4 Surveyor feedback

The finished application is demonstrated to Marius Schønberg, vice president and head of loss prevention at Gard and Bjarne Augestad and Per Haveland, both senior surveyors at Gard. A summary of the demonstration and following interview can be seen in section B.6.

Overall, the surveyors are impressed with the quality of the 360° images in the virtual environments. They are able to view relevant details such as rust, cracks, oil spills, quality of isolation and how cargo is secured on deck from the 360° images. The surveyors thinks the available objects and tools are useful. Schønberg thinks the application is a useful tool to quickly and efficiently get an overview over a vessels condition. He also likes the idea of performing remote surveys as it is a more environmental friendly method of conducting surveys. Schønberg thinks methods like the one demonstrated could improve the quality of remote surveys in the future. He explains that having collaboration functionality like the type present in the application is essential in order to successfully conduct a remote survey. Augestad likes the combination of LiDAR scans and 360° photos used in the application. He thinks the informational datapoints available in the LiDAR scans are useful. He imagines these being further developed by connecting them to a vessels electronic safety system, which would give surveyors even more available information. Augestad would have preferred a higher quality on some of the LiDAR scans, especially Kristian With's engine room, as he thinks it's quality is too poor to be useful.

Augestad thinks a system like the one demonstrated could be used for some of Gard's customers in a screening setting as a sorting tool. For this to happen, Gard would have to create a new system for gathering virtual documentation and create clear public guidelines about how crew should gather material for the application. In addition, Gard would have to create guidelines about how surveyors should create a report from the virtual material. However, Augestad does not think that a method like the one demonstrated can substitute physical surveys. He explains that there are more than physical things that are observed during a survey. A surveyor should also investigate a "human factor" which is elements like work environment, safety and cargo handling. Augestad states that these factors are hard to investigate during a remote survey. He thinks however, that a screening report from an application like the one demonstrated could help give indications whether a vessel is worth investigating further physically or not.

After a survey is conducted, the shipowner may get a report with improvements he has to conduct. In addition to being used as a sorting tool and remote surveys, Augestad thinks the application could be useful for controlling whether these improvements have been implemented or not. If a scan of the vessel is taken during the remote survey, the surveyor may request a new scan in order to verify that the requested improvements have been made. According to Augestad, it would be much more efficient to verify this via the application instead of visiting the vessel physically. Augestad also thinks the application could be useful from a hull insurance perspective as the informational points could provide access to vital information about technical equipment aboard a vessel. Schønberg thinks that by having access to multiple scans of a vessel over its lifetime, a surveyor would be able follow a vessels evolvement and potential decay over time which would be useful as a preparation to a physical survey.

Haveland thinks the technology utilized in the application would be useful for educational purposes. By scanning a vessel, crew can familiarize themselves with the vessels layout before boarding which means that they are ready to work the moment they step aboard. He also thinks the technology demonstrated in the application could be useful for maintenance and reparations, as a repairman can put on his VR headset before coming aboard a vessel. The repairman can then have the machine manager of a vessel show him where the damaged equipment is on the vessel, point and explain what is wrong with it. The repairman could then guide the machine manager to fix the error, and if it is not successful, prepare the necessary equipment and start working when boarding the vessel. Haveland explains that having access to the kind of vessel models demonstrated in the application is useful to both shipowners and surveyors, but for separate purposes.

Chapter 6

Discussion

In this project, a VR application which could improve the way remote vessel surveys are conducted was developed. The developed application can serve as a sorting tool to decide which vessels are worth investigating physically and which can get temporary licenses to continue sailing. The application has several tools available which makes the application closely resemble a physical survey. To develop the application, the Unity engine and the Universal Render Pipeline was used. In this chapter, challenges, limitations and takeaways of this project will be discussed. In addition, an evaluation about potential improvements of the application together with some alternative use-cases will be held.

6.1 Modeling

An important part of the project is how well the modeling tools generate results. Since the application will serve as a way to conduct remote surveys, the material gathered must hold a high quality in order to be useful. In this section the different modeling tools will be discussed, and an analysis on how well they perform will be given.

6.1.1 LiDAR scan quality

This subsection will discuss the quality of the generated LiDAR scans and explain why the LiDAR sensor's performance varies between different environments.

Strengths

The iPad Pro 2020's LiDAR sensor performs well under many circumstances, even when light conditions are challenging. The sensor is intuitive and simple to use through the '3d scanner app' interface as demonstrated in the following part of the video from 0:37 - 1:06. The LiDAR sensors handles verticality well as seen in Figure 5.4d, where two floors are present in the same scan. As seen from the figure, the textures on the generated models accurately resembles the scanned environment and has a high level of detail.

Multiple examples of 3D models generated from LiDAR scans using '3d scanner app' are shown in Figure 5.2 and Figure 5.3. These models can also viewed in first person in the following part of the video from 15:07 - 37:29. As seen from both the images and the video, the LiDAR sensor is capable of scanning large environments and generating an accurate 3D model from them. If the geometry of an environment is not too complex, the LiDAR sensor delivers satisfactory results.

The LiDAR utilized has a lower accuracy than most industrial sensors. The reduced file size of the scans makes it possible for users to move freely around the generated models instead of moving between designated points, which was the case in the scans performed for Hennig Olsen-Is (discussed in subsection 2.3.1). The reduced file size makes applying textures to the point cloud model from a scan fast. Even though the LiDAR sensor has a lower accuracy than industrial sensors, the resulting 3D models from the scans are regarded as detailed. Most of the scans give a great representation of an area's geometry. Since the sensor is handheld, conducting scans does not take a lot of time. It is also easy to choose the resolution that gives the best results for a given area. In addition, it is simple to minimize scan size by using the "low-resolution" scan option. However, some experience may be needed to decide which resolution is optimal for a given area. All in all, learning how to use the scanner and performing a scan is fast as the LiDAR sensor is intuitive to use. As a result, an environment can be scanned in only a few minutes.

Limitations

As seen from the attached video, the quality of the LiDAR scans used to generate 3D models varies greatly. There are several factors impacting the quality of a performed scan. The main factors are:

- **Geometry of environment:** The more complex the geometry of an environment is, the harder it is for the person performing the scan. In order to cover surface area fully with the scanner, all objects in a scene has to be scanned from multiple angles. This can be hard to achieve if an environment has a complex geometry and is large in area because a person performing the scan has to remember which angles objects has and has not been scanned from. If an environment has a lot of minor details, for instance small tubes or pipes, the LiDAR may struggle to capture those details if the resolution of the scan is not sufficiently high. This can clearly be seen from the engine room scan seen in Figure 5.2f and in the following part of the video from 34:26 - 37:30. The engine room is large and has a complex geometry which makes it hard for the person performing the scan. In addition, the engine room has a lot of smaller tubes, valves and pipes which can only be captured properly if the LiDAR resolution is high. Since the engine room is large, it is not possible to make a scan of the entire room with a high resolution. This is because the size (number of vertices) of the resulting scan would be too large, which would significantly impact the experienced frame rate when exploring the scan in VR. This results in a lot of detail being missed in the resulting 3D model since the scan resolution has to be held low.
- **Reflectivity of surfaces:** If a surface is too reflective, the LiDAR sensor's pulse will be reflected away from the sensor instead of back to it. If a LiDAR sensor does not detect a reflection from a sent pulse, it will assume that the area it sent the pulse towards is further away than the sensor's range. This results in empty voids in areas of a generated model where reflective surfaces are present. If a glass window is being scanned, the incoming sensor pulse will pass through the glass instead of reflecting back to the sensor which also causes empty voids in the resulting model. This can clearly be seen in multiple scans in the attached video. Some examples are the mirrors and windows in the apartment scene, the windows in the bridge scan and certain areas in the engine room all have empty voids in them because of reflective surfaces or windows being scanned.
- **Distance to objects being scanned:** The LiDAR sensor used in the iPad Pro 2020 model and iPhone 12 Pro and Pro Max have a range of 5 meters The recommended minimum distance to an object being scanned is 1.8 meters for the best results. In cramped areas it can be hard to keep a minimum of 1.8 meters to the object being scanned which results in inaccurate results. This can clearly be seen in the engine room scene in the attached video. The engine room of Kristian With was tight and cramped, making it impossible to keep 1.8 meters distance to all objects being scanned. This resulted in an inaccurate model and certain objects were not being scanned properly. If the distance to an object being scanned is greater than 5 meters, it will appear as an empty void. This can clearly be seen in the deck scene in the attached video, were there is no water surrounding the vessel. This is because the distance from the deck to the sea was more than 5 meters.

The geometry of an environment will still be captured even if the environment is dimly or not lit because the LiDAR scan operates using infrared light pulses which behave the same regardless of lighting conditions. This will however result in dark textures on the scanned geometry which significantly reduce their utility. However, since a vessel is a workplace, most areas on a vessel where people are frequently located are usually well lit.

Realizations

Of all LiDAR scans taken, the engine room scene had the worst results. As seen from the list above this is because the engine room at Kristian With had all factors that reduce scan quality present. The engine room had a complex geometry with minor details which made it hard to scan properly. The area was large, which meant that it had to be scanned with low quality to reduce the resulting scan's file size. Reflective surfaces were present which resulted in empty voids and it was cramped which made it impossible to keep the recommended distance to all objects being scanned. As a result of these factors, the engine room model were missing detail and had poor quality.

On the contrary, one of the most accurate scans were the scans taken of an apartment and the bridge at Kristian With. The apartment had a simple geometry, were not too large and only had some areas with reflective surfaces. In addition, it was also possible to keep the recommended distance to most objects being scanned. The bridge also had a simple geometry, were not too large and it was possible to keep the recommended distance to most objects being scanned. The bridge's windows resulted in some empty voids in the scan, but apart from that the scan quality was good.

From the list above it becomes apparent that the environment that would give the best scan results is a medium sized confined area with walls and a simple geometry. In addition it has to be open to always allow the minimum distance between the sensor and a scanned object to be held. An optimal environment would not have many reflective surfaces. Not many areas aboard a vessel satisfy all of these conditions, however as demonstrated in the attached video, most non optimal environments will still result in adequate scan results.

6.1.2 360° image quality

This section will discuss the general quality of the 360° images used in the application. In addition, factors impacting image quality will be discussed.

Strenghts

The GoPro Max can be controlled from a phone through the *GoPro Quik* app which makes it simple and intuitive to use, even for inexperienced people. It is waterproof and has a sturdy build which is essential when being used aboard vessels. Different settings of the camera can be adjusted, to optimize the image quality for different lighting scenarios. For this thesis all photos were taken with settings set to 'automatic'. The GoPro Max delivers overall good image quality as seen in Figure 5.4. It handled all tested environments aboard Kristian With well and delivered images of high quality. The camera captures colors with great accuracy and its high resolution images makes it possible to capture minor details such as rust and cracks, even at distance. Even in VR, the images appear sharp and crisp because of the camera's 6K resolution.

Limitations

As seen from the attached video in the results chapter, the general quality of the 360° images are good. However after experimenting, it is found that the image quality of the GoPro max is noticeably reduced when more challenging environments and conditions are encountered. The main factors reducing image quality are:

- **Light availability:** The GoPro Max requires available light in order to capture photos of high quality. Once light conditions are restricted, noise are quickly introduced in photos and details in dark areas of a photo become unclear. Adjusting the camera's settings may improve how the camera handles low light situations, but all in all the image quality becomes poorer when light conditions are challenging.

- **Light contrast:** If a photo has a high contrast between the lighter and darker parts of it, those parts will either be over or under exposed. An example of this situation can be seen in Figure 5.4c. This is because the GoPro Max has no High Dynamic Range (HDR) functionality which allows cameras to take three photos at different exposures and combine them into one photo, making all parts of a photo well exposed.

Realizations

The day Kristian With was scanned and photographed, lighting conditions were optimal with sunny weather. Had Kristian With been visited in the evening or on a cloudy day, image quality on deck may have been worse. The GoPro Max was released in 2019. With camera technology advancing rapidly, there already exists newer 360° cameras on the market today with HDR functionality and better low light performance than the GoPro Max. Other available 360° cameras may perform and handle the above limiting factors better than the GoPro Max. Newer 360° cameras should therefore be considered. Alternatively, GoPro may release a Max 2 which could be a valid option for future use.

When scanning Kristian With, a tripod for the camera was used in order to insure optimal image quality. Alternatively, vessel crew could put the camera on their head, press record and walk around the vessel. A surveyor could then extract the images he needed from the resulting video. However, this requires a head mount which must be supplied with the camera. An optimal camera height for a seamless switch between LiDAR model and 360° image was found in subsection 5.1.3. By putting the camera on a crew member's head, the camera height will depend on the height of the crew member. However, mounting the camera on a crew member's head greatly simplifies the image capturing process, and it is therefore decided that some deviation from the optimal camera height is acceptable. The GoPro Max internal image stabilization ensures high quality photographs even if a person is moving while taking photographs. This is one of the benefits of using an action camera. If no head mount is present, vessel crew could simply walk around the vessel with the camera in hand and take photos. This would however result in the crew member blocking a certain part of each photo which is undesirable. A head mount is therefore regarded as the best option for future use because it is an efficient way of recording imagery of a vessel and gives great results.

6.2 Developed tools

The developed tools are an essential part of the application as they provide functionality that resembles physical surveys. To what degree the different tools succeed is discussed below.

6.2.1 Image Anchors

Image Anchors were developed to give users visual feedback to where 360° images could be toggled. These platforms are located where their corresponding images are taken. The images are rotated according to their surroundings so that they correspond with the LiDAR model. Although the platforms are made somewhat transparent, the platforms can make it hard to view the area below them. This is an undesired behaviour if the platforms are placed above an interesting area. An example of the use of Image Anchors can be seen in the following part of the video from 34:57 - 35:09.

An alternative way of implementing images in the virtual environments could be to discard the platforms, and show users the closest available 360° image when they try to toggle one. This could however cause a user to not immediately know from where an image is taken which would cause confusion. A possible solution is to place more images in the environment. However, this increases the workload on developers when creating new scenes for the application. Another consequence of adding more photos is increased file size of a scene. According to Unity, 2020a textures usually take up the most space in an application build. As there is limited space available for an application build to run on the Oculus Quest 2, the number of images placed in the scene have to be considered. A compromise can be to reduce the quality of the image textures, which results in reduced file sizes.

6.2.2 Documentation tool

To give surveyors the option to record what they see in the application, and report their findings, the documentation tool was developed. It is easy to toggle and quickly captures desired findings using the tool's camera. Being able to also use the microphone to give immediate thoughts about what is being observed is thought to give great value to surveyors. An example of the use of the documentation tool in the application can be seen in the following segment of the video from 30:31 - 30:50. The tool is deemed as an effective way of gathering and storing documentation. Going through saved material on the device after a remote survey, can be helpful when writing the survey report. If desired, captured images can be included directly in the report.

6.2.3 Points of interest

A point of interest displays relevant information about an object in the virtual environment. The displayed information is retrieved online from databases. The implemented POI functionality works as expected. By changing a POI objects TYPE variable, different SQL statements are run, using the VESSEL_ID as a primary key. However, the usefulness of the POI object depends on the quality and availability of relevant data for each vessel. This makes the POI object vulnerable as it is of no use in models of vessels with little or no recorded data. Manually placing POIs and setting their TYPE variable can be time consuming. However, once added to a scene with a significant amount of available data, the POI objects are regarded as highly useful. An example of the use of POI to display relevant data can be seen in the following part of the video from 25:58 - 26:27.

6.2.4 Measurement tool

The measurement tool gives a surveyor the option to measure distances in the LiDAR model. From a meeting with the head of loss prevention and senior surveyors at Gard (section B.2), the head of loss prevention demonstrated that surveyors measure damaged areas to give a perception of the size of the affected areas. Since the LiDAR scans have a reasonably high resolution, accurate measurements can be achieved. The tool is easy and fast to use. All in all, it is regarded as a useful addition to the developed application. An example of the use of the measurement tool in the application can be seen in the following part of the video from 15:52 - 16:07. One disadvantage of the tool is that the quality of the measurements depend on where the measurement points are placed. If millimeter or centimeter precision is wanted from of a measurement, it requires that the person conducting it has a steady hand.

The inaccuracy of a LiDAR model is equal to the resolution of the model's scan. Therefore, some inaccuracies will be present in the resulting LiDAR models. However, the measurement tool can give a precise indication of distances in the LiDAR models with only some minor inaccuracies.

6.3 Time spent gathering material

Gathering the necessary 360° photographs can be done in the time it takes to walk around the vessel. By putting the camera on the head of a crew member and pressing record, an entire vessel can be filmed by walking through it. The video can then be sent to the surveyor who can extract the images he needs. Alternatively, the surveyor could request images of certain objects or areas that can be sent to him in a matter of minutes.

One of the benefits of using the LiDAR sensor in an iPad Pro is that since the iPad is handheld, scanning environments is fast. Instead of moving a stationary LiDAR sensor around several points and waiting for it to scan each point, users simply move around the environment continuously "painting" it with the sensor by looking at the UI of the '3d scanner app'. After a few minutes of instructions, everybody should be able to make high quality LiDAR scans. In addition to being simple to use, the scanning process of using the sensor on an iPad Pro is fast. The time usage spent on the different scanning process stages is as follows:

- **Scanning an environment:** The time spent performing a LiDAR scan depends on what is being scanned. If a single object is being scanned, for instance a motor, this is done in a matter of seconds. If an entire environment is being scanned, it can take anywhere between a couple of minutes to ten minutes depending on the size of the environment. Large areas such as the deck and engine room at Kristian With took approximately 10 minutes. The bridge and stern however, took less than five minutes to scan. Even though large areas can take up to 10 minutes to scan, performing a scan is fast and efficient method of getting an accurate 3D model of an environment.
- **Processing the scan:** After a LiDAR scan is performed, the '3d scanner app' by Laan Labs generates a 3D model of the scan in a matter of seconds. The 3D model can then be processed directly in the app. The in-app processing adds more detail to the 3D model and textures to it. The model processing can be done with different accuracies. The most accurate choice is high definition (HD) processing which was used for all models present in the attached video. Even on the larger scans like the bridge and deck, HD processing of the 3D models only took a couple of minutes.
- **Importing the scan to Unity:** Importing the generated model into Unity only takes a couple of minutes, depending on the file size of the model. After a model is imported, the only necessary steps to be able to explore it in VR is to add a mesh collider to the model and add an *Essential Assets* gameobject to the scene. Both of these steps are done in a matter of seconds. The model is then ready to be explored in VR. The most time consuming part of implementing a new environment in Unity is adding the Image Anchors and POIs to the scene. When adding Image Anchors, all image textures has to be rotated according to its environment. In addition, all POIs TYPE variable has to be set.

As seen from the list above, using the LiDAR sensor in an iPad Pro makes the process of scanning and generating models extremely fast. In the bridge scene for instance, scanning the environment took about 4 minutes. HD processing of the model took less than two minutes. The model was then exported to a cloud service and later downloaded to a computer. Importing the scan into Unity took 2 minutes, and adding the mesh collider and *Essential Assets* GameObject only took a few seconds. With the current framework in place, all necessary steps from scanning the environment to being able to explore it in VR took less than 10 minutes which is very quick. If Image Anchors and POIs are added to a scene, this significantly increase the amount of time spent. Adding Image Anchors and POIs to the bridge scene took approximately 15 minutes which is 150% more time than all other steps taken in order to capture, generate, import and prepare the scan. Even though adding Image Anchors and POIs significantly increases the time spent preparing a scene, it is deemed to be worth it since the Image Anchors add great detail to the environment. All in all, being able to take a scan, process the model, import it to Unity and prepare it with Image Anchors and POIs in approximately 25 minutes is very fast and proves the power of the technology utilized.

6.4 VR Sickness

There has been many means taken to avoid VR sickness and make the application more comfortable to use. The three major factors are control over the camera, frame rate, and how long the duration of the VR experience is. How these factors help reduce VR sickness, and at which degree they succeed is discussed below.

6.4.1 Control

To always be in control of where the camera is looking is very important to avoid VR sickness. A mean taken to give the user full control over how the camera turns is thus implemented as described in section 4.4.1. This is because people have different preferences regarding turn rate and type. By giving the option of how the user will turn in the application the user can experiment with what suits best at the given time. This can be seen in the attached video from 13:43 - 14:20. After some time the preference can change, and this option is then provided. It is important that the turn feels natural for the user, and never exceeds what the user expect.

A way to make the ship experience more realistic can be to add natural sway in the camera because of the waves in the ocean. But because this interfere with the users physical movements, this was not implemented. In the interview conducted with Björn Mes (section B.3), he mentioned that a customer had this question. The customer was wondering where the sway of the boat was and asked why there was no movement on the boat. This would cause VR sickness, and is thus avoided in the application.

6.4.2 Frame rate

Another important factor is the frame rate. Each frame requires a certain amount of calculation from both the CPU and GPU of the device running the application. If the target FPS is 72, the calculations have to be repeated 72 times in a second. If a frame is supposed to be displayed, but the calculations for it is not finished, the system will stay on the previous frame, wait for the calculations to finish and then display the next frame.

One of the ways to improve the frame rate was to use a render pipeline that was targeted towards lower-end devices such as VR-headsets. As can be seen from Figure 5.7a this was a correct choice. In addition to providing smaller APK files, so that additional scenes could be loaded into the application, the frame rate was also considerably higher. One reason behind the higher frame rate could be because the URP preprocess the lights before building the application, thus saving computational cost at run time as explained in section 2.4.2.

When Unity build the application many shader variants from a single shader source file is compiled. The URP can exclude some shader variants, which result in smaller build sizes and shorter build times (Unity, 2020b). This can however lead to less realistic results if some shaders that are important for the application are excluded. These shaders could be important for how light is perceived in the application. If the URP generate acceptable results, URP is recommended to use, especially when targeting lower-end devices as was done in this thesis. Using the URP is therefore thought to be the best solution as a render pipeline in this application.

Another way to increase the frame rate is by be aware of how many vertices is being rendered in a scene. As seen in Figure 5.7b, the number of vertices being rendered in a scene have a large impact on the frame rate. In the two first scenes, the frame rate is almost constant at 72 FPS which is as high as Oculus Quest 2 can run applications. Once the number of vertices being rendered exceeds 2 million, the frame rate starts to vary between 60-70 FPS.

There are several factors that can be considered when trying to reduce VR sickness. The frame rate is one thing, but the consistency is another. According to Ubisoft, 2020 an unchanging frame rate which is between 30 and 60 FPS will feel smooth, while a varying frame rate between 35 FPS and 60 FPS will feel choppy. 24 FPS is the threshold in which the human eye can identify the passage of a frame to the next one.

As seen from Figure 5.7b, the upwards spike in frame rate from approximately 180-190 seconds occurs when toggling off the 3D model and turning on the 360° image. This demonstrates the importance of disabling the rendering of the LiDAR model once an image is toggled since it drastically reduces the number of vertices being rendered in a scene.

However, as mentioned above, consistency in the frame rate is also important to give the user a pleasant experience. As seen, by disabling the model the frame rate change from around 35-40 FPS to about 70 FPS. When the image is being disabled and the LiDAR model is activated, the sudden fall in frame rate is not optimal. However, always having the highest frame rate when looking at the 360° photos makes using the Image Anchors a smoother experience.

6.4.3 Duration

The teleportation functionality results in faster transportation in the application. It was added so that users can teleport to desired Image Anchors as described in subsection 4.3.2. However, teleportation could also have been enabled on all floor surfaces. This would result in even faster navigation in the environment. However, this was not added as many of the produced virtual environments have a large amount of Image Anchors in them, which resulted in many available teleportation points. If teleportation was added to all floor surfaces, it would not be as easy to teleport to an Image Anchor to quickly toggle them. By having teleportation only available at the Image Anchors, users knows that fast toggling of the different images are possible.

6.5 Development using Unity

Unity proved an essential tool to achieve the results described in chapter 5. Without the help of a game engine, which provide access to an extensive amount of resources, the implementation would have taken much more time. Since the project only lasted for about four months, the use of a game engine was deemed necessary. The application was developed from the scratch, using the Unity engine's rendering, physics, network functionality and XR Interaction Toolkit.

When developing the application, several bugs were encountered. Having a large active community which have encountered similar problems in the past, proved useful. Through Unitys forums or StackOverflow, discussions about solutions to different problems could be considered. Because of the large unified community Unity has, most problems encountered had a solution online.

The new action-based input system described in section 2.4.2 is proved both useful and efficient. Even though the documentation was lacking compared to the old device-based input system, it proved easy to use once it was properly understood. The input system made it easier to bind different actions to controllers of any kind in the action editor as seen in Figure 2.11a. The action editor contained whole libraries of allowed inputs which could be selected. This was however not without any difficulty. An example where difficulties were encountered was with the grip button on both controllers. The action editor only contained information about *gripButton* and *gripPressed*. These two properties only had two integer values, namely 0 and 1. As the animation of the controllers presented in section 4.2.4 needed float values between 0 and 1, this was not possible. However, when experimenting it was found that when changing the text path from *gripButton* or *gripPressed* to *grip*, float values could be extracted. This presents a problem using the XR Interaction Toolkit. Since the package is still in preview/beta mode and not completely finished, unexpected problems can arise. However, as the problems could be fixed after tweaking some values or experimenting with other methods, the package is deemed successful.

Some other problems was encountered when using the Unity game engine. The newest available version of Unity was decided to be the best option for the project since the application was depending on preview/beta packages to implement the desired functionality. However, the latest Unity release was not always as stable as expected. Several times when the application was flashed to Oculus Quest 2, Unity would allocated several gigabytes as *shader cache*. Shader cache remembers the result of the previous compilation, but after changing the application and flashing once more the shader would continue to grow unbounded. As there is limited storage on computers, the only way to make this this allocation stop was to close Unity, delete the shader cache, and reboot Unity. It is unknown if this could be avoided if a more stable Unity release was used, but it was only a minor concern. As new versions was released during development of the application, the problem occured less frequent, but was still present.

Connecting the application to a database was no big challenge, as a GET request could be sent from the application to request data using Unity specific methods. To maintain security, a PHP script hosted in an Azure web app served as an intermediate part, communicating with the application, retrieving data from a database and sending it back to the application. The hard part was writing the server side scripts retrieving the requested data. Any server side programming language like Node.js could have been used, however PHP was used in this scenario because of available documentation in online forums.

6.6 Alternative use-cases

In addition to the possible use cases mentioned by the surveyors in section 5.4, there exists alternative use cases for the application and demonstrated technology outside the marine industry. This section will discuss some of the possible use cases.

6.6.1 General training and familiarization

Since object interactions and physics is implemented in the application, it is possible to develop different training scenarios using the already implemented framework. In addition, by utilizing the implemented network functionality it is possible to get instructions and demonstrations from an instructor online. By using the available functionality, one could implement any situation or equipment training scenario and receive instructions online using the available functionality. Since object data is sent online, an instructor could first demonstrate the task to be performed for a student, and then later observe the student repeat the task. Using the LiDAR sensor, the training scenario could take place in a model generated from a real world environment to increase the realism and immersiveness of the training.

In addition to training scenarios, the application can be used for a number of familiarization situations. By creating a virtual model of a real world environment and exploring it in VR, users are able to train on navigating through the real world environment without being there physically. This could be useful for a number of scenarios, for instance for new employees in factories or offices needing to familiarize themselves with their workplace layout. If a new employee has practiced the workplace layout before entering the workplace physically, the employee will be ready to work the moment they arrive. In addition, fire drills and other emergency evacuations, which significantly impacts daily operations, can be held in VR. This would save any company money as there would be no need to disrupt daily operations at the workplace to conduct emergency drills.

6.6.2 Consumer retail and housing rental market

Since a LiDAR scan has a one to one relationship with the object being scanned, the application could be used in consumer retail. By scanning a product from multiple angles, users of the application would be able to explore a virtual twin of the scanned product. Viewing a 3D model of the product in VR would give a more accurate representation of it compared to viewing images of it online. This is especially true for clothing, where clothing size and fit may be hard to see from a photograph when shopping online. By using the measurement tool, users could for instance measure the arm length of a sweater or or the length of it. In addition, viewing a virtual model of a piece of clothing would give a better understanding of its size since it has a one-to-one relationship with the real life product. Creating 3D models of single objects with the LiDAR sensor in the iPad Pro is very fast, taking less than a minute which demonstrates the power of the technology utilized.

As demonstrated in the results chapter's attached video, the LiDAR scanner performs well when scanning indoor areas with simple geometry. Instead of exploring vessels, the application could be used to explore apartments or student accommodation. A common problem in areas with a pressed housing market is landlords lying about accommodation quality and area in online advertisements to attract tenants. Unity allows any project to be built as a WebGL file, which makes it possible to run it directly in a web browser. By changing the application so that it is controllable via a keyboard and building it as a WebGL project, it should be possible to run the developed application directly in a web browser. Setting up keyboard input would be easy because the action-based input system discussed in section 2.4.2 is currently used in the application which means that actions can be bound to several input controls. By setting up additional keyboard controls to the already defined actions, the functionality of the application will be the same when using a keyboard. By attaching a link in an online add on Finn or hybel.no, users would be able to explore the accommodation being rented out in a web browser, assuming the landlord made a LiDAR scan of it using an iPad or iPhone. Since taking a scan, importing it to the application and uploading it only takes a few of minutes, a landlord would have no good excuses not to do so. By using the measurement tool, students could easily acquire the square meter size of the accommodation, which is something a landlord can lie about to attract tenants. Users could also measure other distances in the accommodation to see if their currently owned furniture like beds and desks would fit in the accommodation available for rent. In addition, LiDAR scans of indoor environments with simple geometry like accommodations, are of high standard so that general accommodation quality can be assessed from the scans.

6.7 Reflections

This project has been a learning experience on many different levels. A VR application aimed at helping remote vessel surveys and working as a sorting tool were developed. In addition, several people across different industries were interviewed about the technologies utilized to develop the application.

VR technology

After speaking to different actors across various industries about VR technology it became apparent that the VR revolution has already started. Companies are actively using VR and developing VR solutions today. VR technology has seen enormous progress in the last two years with the Oculus Quest and Quest 2 being released. The Quest 2 in particular makes VR accessible for anyone as the headset gives a great VR experience at a reasonable price without the need of a powerful computer. In addition, powerful tools for VR development are currently being released. The toolkit used for this project is Unity's XR Interaction Toolkit. It is still in development, but once released it will make VR development simpler and more accessible to everyone.

LiDAR technology

Capable LiDAR sensors have been available for several years and is nothing new. However, having a LiDAR sensor integrated in a tablet or phone is something that has not been seen before. These sensors opens up a new world of possibilities as LiDAR scanning and 3D modeling is made available to everyone. For instance, game developers can scan objects and instantly create a 3D model of it. The model can then be imported into a game engine instead of having to manually create the object in a 3D modeling software like Blender. This is significantly faster than manually creating a 3D asset, and may result in an asset with better quality than one that is manually created, if the artist is not particularly skilled. Even though the sensor's quality is not the best, it is revolutionary that anyone can make a 3D model of an object or environment in a matter of seconds or minutes. With time, it is reasonable to believe that these sensors and software will continue improving, resulting in even better 3D models. While this project was being developed, Apple released a new iPad Pro (2021) model with the new M1 processing chip which according to Apple (Apple, 2021) offers 50% increased CPU performance and 40% faster GPU performance compared to the previous iPad Pro model that was used for gathering material for this project. With this increase in computing power, it is likely to believe that the new iPad Pro (2021) would process and texture scans significantly faster than the model utilized in this project.

With these technological advances in mind, it is safe to say that developing an application like the one presented in this project would have been significantly harder or maybe not possible a couple of years ago due to equipment and software limitations. With the technological development only accelerating, it is likely that the equipment and software utilized for this project will only continue improving.

6.8 Future work

A number of improvements to the application were proposed by Bjarne Augestad and Per Haveland, two senior surveyors at Gard and Marius Schønberg, Vice President at Gard and Head of Loss Prevention section B.6. The key improvements proposed by the surveyors, together with some additional improvements proposed by the authors are presented below:

- **Adding vessel floor plans:** Schønberg and Haveland would like to have floor plans of the vessels explored implemented in the application. They suggests implementing a feature similar to the ones found in many video games where the floor plans could serve as a 2D map for the users position. Users can then navigate around the scan by utilizing the floor plan map which tells them their exact position in the floor plan. In addition, this would give users extra information about objects they are looking at in the environments. For instance, if a user is moving towards a motor, the user can look at the virtual map and see that the motor they are moving towards is an auxiliary engine.
- **More detailed photos of objects:** Although Augestad thinks the general quality of the 360° images are good, he would like to have additional photos within the 360° images. For instance if a user is looking at a 360° photo where life saving equipment is visible, users of the application should be able to click the life saving equipment to view more detailed 2D photos of it. This would make it easier for users of the application to investigate important objects in further detail. However, this would significantly increase the time spent implementing the Unity scene of an environment, as these interactive points and additional photos would have to be manually placed for each photo and object.
- **Live streaming of scanned material:** Schønberg suggests implementing functionality where a surveyor can put on his VR headset and command vessel crew to receive scans of desired objects or rooms in real time. This would make it easier for the surveyor to receive materials of the areas he is interested in and harder for vessel crew to try avoid scanning unflattering areas. This would however require a fast and responsive internet connection for both parts which may be hard at sea. In addition this would require writing algorithms to automatically set up a Unity scene, import the scan taken, place Image Anchors, build the scene and export it online. With the current used method for gathering material and application architecture, this may be difficult.
- **Connecting the application to vessels electronic systems:** Most modern vessels today have on board electronic systems. Augestad suggests connecting the application to these so that the Points of Interest can extract more detailed information about technical equipment, valves and motors maintenance and operational history. This would result in vessel scans having digital twins of technical equipment and systems on board. However, this would require that the vessel to be scanned has an internet connection in order to retrieve updated information when a surveyor wishes to inspect a certain scan. In addition, different vessels may have completely different electronic systems which would make it hard to implement good solutions working for the different systems.
- **Automate image placement and rotation:** The most time consuming part of creating a virtual environment in Unity is to place the 360° images and rotate them in a such way so that

they correspond to the 3D environment. Using AI and computer vision to recognize matching patterns between the 360° images and the LiDAR scan could help place and rotate the Image Anchors automatically. However, this would require significant technical knowledge about computer vision and AI which are two difficult subject areas.

- **Different scan quality depending on the device:** There are many available VR devices in the market with different processing power. Since better scan quality leads to more expensive rendering, the quality of the loaded scans could be automatically chosen to provide optimal frame rate for the device being used. VR sickness could then be reduced.

All things considered, adding vessel floor plans seems like the most useful proposed feature as this would significantly help users navigate around the virtual environment and navigate to desired objects. Implementing such a feature does not come with any disadvantages and does not increase the time spent setting up new virtual environments. Implementing functionality which detects device hardware specifications and loads environments according to them would also be a useful feature. By having this functionality, users would be presented environments optimized for their devices automatically which would increase the achieved frame rate and therefore also overall user experience. Implementing such a feature would not impose any great challenges. It is therefore highly recommended to implement vessel floor plans and map functionality as well as automatic device specification detection in the application.

Chapter 7

Conclusion

Conducting physical vessel surveys has been difficult for Gard during the COVID-19 pandemic. As a result, Gard faces a large amount of outstanding vessel surveys. Gard's current way of conducting remote surveys was deemed not satisfactory. The goal of this thesis was therefore to develop a VR application for conducting remote vessel surveys. The application should offer a better solution for conducting remote surveys compared to existing methods used by Gard. By making a VR application, a more realistic and immersive experience compared to traditional methods through a 2D screen is achieved. In addition, exploring vessels in virtual reality makes it possible to observe and investigate findings in a more natural and interactive way. Important details that can be missed through a traditional screen are more easily observed in VR. The main benefit of conducting remote surveys is that they can be performed from anywhere in the world as it is no longer required for surveyors to physically board the vessel being inspected. It is more environmental friendly and cost effective since there is no need to pay flight tickets and accommodation for surveyors. In addition, the inspected vessel's operational availability is not affected when surveys are being performed remotely. The only requirement for conducting remote surveys using the proposed method is that the vessel crew has the necessary equipment to gather material for the survey.

7.1 Equipment

The virtual environments explored in the developed application are created by combining LiDAR models with 360° images. The equipment used was the LiDAR sensor present in modern mobile devices from Apple and the GoPro Max to capture 360° images. The LiDAR sensor from the latest mobile devices from Apple was chosen because of its current and future availability. This makes it possible for everyone to create accurate 3D models of objects and environments. The GoPro Max was chosen as the preferred 360° camera because of its high image resolution, user friendliness and robust build.

The LiDAR scans captured in this project gives an accurate representation about the scanned environment. However, in complex environments the accuracy of the LiDAR sensor used in this project is deemed insufficient. This results in a general lack of detail in the resulting model's geometry and texture. The 360° images from the GoPro Max are considered to hold a high quality. From the images, it is possible to see minor but relevant details aboard a vessel such as cracks and rust. However, it is not possible to create a virtual model from the images alone as they lack spatial information. By combining the two methods to create a virtual environment, the limitations of the separate methods can be avoided. What the LiDAR model lacks in detail is achieved in the 360° images and what the 360° images lack in spatial information is provided by a LiDAR model. In this way the combination of the two methods complete each other resulting in virtual environments of high quality.

7.2 Application

All implemented tools and objects in the application work as desired. Users can seamlessly switch between 3D models and 360° images in the virtual environment to view it in greater detail. An important part of a survey is to be able to communicate with a member of the vessel crew in order to clarify findings. As a result, network functionality was implemented to allow users to communicate and interact online. In addition, users can interact with points of interest to view information about different objects in the virtual environment. If a surveyor detects damaged areas on a vessel, it is useful to be able to measure the extent of the damage. Thus, a measurement tool which allows users to measure distances in the virtual environment is implemented. Measuring distances is possible because the LiDAR sensor capture the actual distances in the environment. Another important element of a survey is to be able to document findings for the resulting survey report that describe the condition of the vessel. A documentation tool is therefore implemented to allow users to document their virtual findings. In addition, users can download virtual environments stored in the cloud which saves device storage space.

All the tools and objects are included to give surveyors the feeling of conducting a physical survey, but with additional advantages. Gathering material for the virtual environments are quick and can be done in a few minutes. The equipment used to gather material is user friendly and can be used by anyone after a short introduction. A link to a video demonstrating the material gathering process and the application's functionality can be seen in the following hyperlink: YouTube.

After demonstrating the finished application to two of Gard's senior surveyors and Gard's head of loss prevention, the feedback was that the technology is promising. However, the surveyors find the quality of the LiDAR models to be insufficient. At this time, there are no plans to replace physical surveys with a solution like the one presented at Gard. However as one surveyor stated, an application like the one presented could serve as a sorting tool for some of Gard's clients. In addition, the surveyors think that the developed application could be useful for checking if requested improvements have been made aboard a vessel. Also, the surveyors think the application could be used for crew training. All things considered, the developed application achieves the goals set in section 1.3. It successfully enables virtual walkthroughs of vessels, which was the task given by Gard (section 1.2), and the added tools and functionality makes it a useful instrument for conducting remote surveys.

The technology demonstrated in this thesis proves that 3D scanning and modeling are accessible to everyone. Having the ability to quickly scan an environment and being able to explore it in VR after a few minutes is revolutionary and will provide new opportunities for VR applications in the future. Since the LiDAR sensor utilized in the project is Apple's first generation LiDAR sensor, it is reasonable to believe that future mobile devices will be equipped with even better LiDAR sensors. Also, the software used to process the scans has seen significant improvements since the project started. Future upgraded hardware and software could significantly increase the quality of the achieved 3D models which would create more realistic environments for the application. Increased model quality would improve remote vessel surveys using the proposed method. It is therefore suggested that new scans are performed when future iterations of LiDAR sensors in mobile devices are improved.

Bibliography

Miljødirektoratet. (2021). *Klima*. https://miljostatus.miljodirektoratet.no/miljomal/klima/

Lynch, G. (2020). *Oculus quest 2 review*. Retrieved 26th January 2021, from https://www.techradar.com/reviews/oculus-quest-2-review

Delight XR. (2021). *Xr glossary*. Retrieved 28th April 2021, from https://delight-vr.com/xr-glossary/

Berg, L. P. & Vance, J. M. (2017). Industry use of virtual reality in product design and manufacturing: A survey. *VIRTUAL REALITY*, *21*, 1–17. https://doi.org/10.1007/s10055-016-0293-9

Bowman, D. A., Coquillart, S., Froehlich, B., Hirose, M., Kitamura, Y., Kiyokawa, K. & Stuerzlinger, W. (2008). 3d user interfaces: New directions and perspectives. *IEEE Computer Graphics and Applications*, *28*, 20–36. https://doi.org/10.1109/MCG.2008.109

Switzer, E. (2020). *Quest 2 hand tracking is a neat trick, but far from game ready*. Retrieved 25th January 2021, from https://www.thegamer.com/oculus-quest-2-hand-tracking-waltz-wizard/

IDC. (2020). *Worldwide spending on augmented and virtual reality forecast to deliver strong growth through 2024, according to a new idc spending guide*. Retrieved 26th January 2021, from https://www.idc.com/getdoc.jsp?containerId=prUS47012020

Barnard, D. (2019). *History of vr - timeline of events and tech development*. Retrieved 18th February 2021, from https://virtualspeech.com/blog/history-of-vr

Thompson, S. (2020). *Motion sickness in vr: Why it happens and how to minimise it*. Retrieved 18th February 2021, from https://virtualspeech.com/blog/motion-sickness-vr

LeddarTech. (2020). *Why lidar*. Retrieved 25th January 2021, from https://leddartech.com/why-lidar/

Zeng, Z., Li, X., Yu, Y. K. & Fu, C.-W. (2019). Deep floor plan recognition using a multi-task network with room-boundary-guided attention.

90Seconds. (2020). *What is a 360 camera?* Retrieved 26th January 2021, from https://90seconds.com/what-is/360-camera/

Viewport. (2021). *Monoscopic vs sterescopic*. Retrieved 28th April 2021, from https://viewport.com.au/post/monoscopic-vs-sterescopic

Terence Eden. (2013). *Converting stereoscopic images from hsbs movies into 3d models*. https://www.youtube.com/watch?v=YGJ4qdoAfAw

Mes, B. (2018). *Virtual and augmented reality in shipbuilding*. Retrieved 18th February 2021, from https://www.damen.com/en/magazines/2018/01/virtual-and-augmented-reality-in-shipbuilding

Hakirevic, N. (2020). *Shipping industry sees growth in remote surveys in times of coronavirus crisis*. Retrieved 22nd February 2021, from https://www.offshore-energy.biz/shipping-industry-sees-growth-in-remote-surveys-in-times-of-coronavirus-crisis/

DNV. (2020). *Remote annual surveys – offering a digital alternative*. Retrieved 22nd February 2021, from https://www.dnvgl.com/expert-story/maritime-impact/Remote-annual-surveys-offering-a-digital-alternative.html

Corral Design. (2018). *Virtual reality: Remote cargo ship surveys*. Retrieved 21st February 2021, from https://www.corralldesign.com/vr-ship-surveys

DNV. (2021). *Seafarers zoom in on remote surveys of ships*. https://www.dnv.com/expert-story/maritime-impact/Seafarers-zoom-in-on-remote-surveys-of-ships.html

Unreal Engine. (2021a). *Openxr*. Retrieved 26th April 2021, from https://docs.unrealengine.com/en-US/SharingAndReleasing/XRDevelopment/Openxr/index.html

Computer Hope. (2019). *Unreal engine*. Retrieved 14th April 2021, from https://www.computerhope.com/jargon/u/unreal-engine.htm

Circuit Stream. (2021). *Unity vs unreal engine for xr development: Which one is better?* Retrieved 14th April 2021, from https://circuitstream.com/blog/unity-vs-unreal/

Unreal Engine. (2021b). *Xr development*. Retrieved 14th April 2021, from https://docs.unrealengine.com/en-US/SharingAndReleasing/XRDevelopment/index.html

Microsoft. (2018). *Announcing microsoft directx raytracing!* Retrieved 15th April 2021, from https://devblogs.microsoft.com/directx/announcing-microsoft-directx-raytracing/

Unreal Engine. (2021c). *Networking and multiplayer*. Retrieved 15th April 2021, from https://docs.unrealengine.com/en-US/InteractiveExperiences/Networking/index.html

Peckham, E. (2019). *How unity built the world's most popular game engine*. Retrieved 27th October 2020, from https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/?guccounter=1

Unity Technologies. (2020). *Real-time solutions, endless opportunities*. Retrieved 27th October 2020, from https://unity.com/solutions

Unity. (2021a). *Xr*. Retrieved 14th April 2021, from https://docs.unity3d.com/Manual/XR.html

Unity. (2021b). *Choosing and configuring a render pipeline and lighting solution*. Retrieved 15th April 2021, from https://docs.unity3d.com/Manual/BestPracticeLightingPipelines.html

Unity. (2021c). *Multiplayer and networking*. Retrieved 15th April 2021, from https://docs.unity3d.com/Manual/UNet.html

Photon. (2021a). *Pun*. Retrieved 15th April 2021, from https://www.photonengine.com/pun

Photon. (2021b). *General documentation*. Retrieved 28th April 2021, from https://doc-api.photonengine.com/en/pun/v1/general.html

Unity. (2021d). *Configuring an xr rig with the xr interaction toolkit*. Retrieved 29th April 2021, from https://learn.unity.com/tutorial/configuring-an-xr-rig-with-the-xr-interaction-toolkit#

Tracey, D. (2020). *Who's using microsoft azure?* Retrieved 26th January 2021, from https://www.contino.io/insights/whos-using-microsoft-azure-2020

Gard. (2021a). *About gard*. Retrieved 22nd April 2021, from https://www.gard.no/web/about-gard

Gard. (2021b). *Forms*. Retrieved 26th April 2021, from https://www.gard.no/web/forms

Apple. (2020). *Ipad pro*. https://www.apple.com/ipad-pro/

GoPro. (2019). *Gopro max*. https://gopro.com/en/no/shop/cameras/max/CHDHZ-202-master.html

Oculus. (2020). *Quest 2*. https://www.oculus.com/quest-2/

Upwork. (2019). *C vs. c++: Which language is right for your software project?* Retrieved 22nd April 2021, from https://www.upwork.com/resources/c-sharp-vs-c-plus-plus

Oculus. (2021a). *Packages of type misc*. Retrieved 5th May 2021, from https://developer.oculus.com/downloads/misc/

Kohncke, E. (2019). *Low-poly male avatar 02 vr*. https://sketchfab.com/3d-models/low-poly-male-avatar-02-vr-87bb06b3d4ac48c0ae8d5d76bf0d6211

Oculus. (2021b). *Upload apps for oculus quest*. https://developer.oculus.com/distribute/publish-uploading-mobile/?locale=nb_NO

Oculus Developer Center. (2021). *Upload apps for oculus quest*. Retrieved 22nd May 2021, from https://developer.oculus.com/distribute/publish-uploading-mobile/?locale=nb_NO

Unity. (2020a). *Reducing the file size of your build*. Retrieved 24th May 2021, from https://docs.unity3d.com/Manual/ReducingFilesize.html

Unity. (2020b). *Shader stripping*. Retrieved 24th May 2021, from https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/shader-stripping.html

Ubisoft. (2020). *Fps or frames per second*. Retrieved 23rd May 2021, from https://www.ubisoft.com/en-gb/help/connectivity-and-performance/article/fps-or-frames-per-second/000062726

Apple. (2021). *Apple presenterer ny ipad pro med banebrytende m1-chip, lynrask 5g og praktfull 12,9-tommers liquid retina xdr-skjerm*. Retrieved 23rd May 2021, from https://www.apple.com/no/newsroom/2021/04/apple-unveils-new-ipad-pro-with-m1-chip-and-stunning-liquid-retina-xdr-display/

Appendix A

Downloading project

The project is available at Google Drive. The folder on Google Drive contains the project that can be exported to Unity, the scripts used to provide the functionality, and the application file (*.apk*). The *.apk* file can be transferred to the VR headset by following this guide. The scripts are attached as a separate folder, in case only the scripts are at interest when reading chapter 4.

If it is desired to open the project and build the application in Unity, an explanation of how to do so is provided below:

1. Download Unity Hub.
2. Go to **Installs** > **ADD** and download Unity version: 2020.3.11f1. If this version is not available locate the **download archive** link, and install Unity version 2020.3.11f1. Alternatively the newest stable release of Unity can be downloaded.
3. Go to **Projects** > **ADD** and locate the folder downloaded from Google Drive and add the folder named *Master Project*.
4. Select the project in Unity Hub and run the project. To be able to run the project as intended, a VR device is required. Also, developer mode must be enabled on the device. A guide to how this can be done on the Oculus Quest can be seen in the following link.
5. To build the project and run it on the Oculus Quest, make sure that the platform is set to *Android*. This can be done by locating the build settings in **File** > **Build Settings...** and switch the platform from *PC, Mac & Linux Standalone* to *Android*.
6. To flash the project to the VR device, press **Build and Run** in the build settings window, or press **File** > **Build and Run**. Give a name to the *.apk* file and press **Save**.

Appendix B

Interviews

B.1 Intervju med Erling Tønnesen ved Sweco - 19.02.2021

Møte med Erling Tønnesen i forbindelse med masteroppgave

Til stede: Jostein Sætra Schefte, Einar Lorentsen, og Erling Tønnesen.
Mål for møtet: Erling Tønnesen er til daglig konsulent i Sweco, et av Europas ledende selskap på området arkitekt- og ingeniørrådgivning. Gjennom arbeidet har han jobbet på flere prosjekter der spesielt LiDAR scanning, men også VR integrasjon er tatt i bruk. Målet for møtet er å få innsikt i hvordan han har jobbet med disse prosjektene, hva de har gått ut på og hvilke utfordringer som ble møtt på.

Fortell om hvilke prosjekter du har jobbet på der LiDAR og VR teknologi er tatt i bruk.

Tønnesen forteller at han har jobbet mye med 3D scanning og 3D modellering. Dette er mye brukt teknologi i Sweco. En ide han hadde hatt var å utforske disse i VR. Teknologien ble testet ut i flere prosjekter og endte opp i stilige prototyper. Tønnesen trekker frem to spesifikke prosjekter på utbygging i prosessindustrien. Et prosjekt på Hennig-Olsen is sin fabrikk i Kristiansand og et på GE Healthcare's avdeling Lindesnes. Tønnesen diskuterer prosjektene og spesifiserer at for deres del så man først og fremst på utforsking av LiDAR skanner i VR som et mulig verktøy for å bedre kunne visualisere hva som foregår inne i en fabrikk i for eksempel en produksjonslinje istedet for et verktøy for byggplanlegging for arkitekter, da det finnes mer egnede verktøy for dette. Utforsking av LiDAR skanner i VR kan være et godt verktøy for å bedre kunne visualisere for kundene hvordan endringer inne i fabrikken påvirker eksisterende materie. En nyttig fordel av teknologien er muligheten til for eksempel å kunne inspisere en fremtidig produksjonslinje i VR, for så å kunne endre og modifisere og optimere den i sanntid. Inspeksjon i VR gir ofte en bedre romfølelse enn det ville gjort å se samme materie gjennom en tradisjonell skjerm.

Hvilket utstyr og software ble brukt?

Tønnesen forteller at LiDAR skannere av høy kvalitet og nøyaktighet ble brukt. Utstyret var så dyrt at det på tidspunktene de to prosjektene ble gjennomført, at det ble leid inn et eksternt firma for å foreta scannene og bearbeide dataene. Utstyret har nøyaktighet på under millimeteren, men avstanden måleren står fra det målte punktet vil påvirke dette. I dag har Sweco kjøpt inn eget skanneutstyr og bearbeider scandata selv. Sweco utviklet en egen plugin til Autodesk's naviswork for å interagere med scannene i VR.

Hvilke utfordringer ble møtt på i arbeidet med prosjektene?

Tønnesen forteller at det ble møtt på flere utfordringer, og at det i dag ikke jobbes videre med utforsking av LiDAR scannere i VR på Sweco. Som et konsulentselskap skal best mulig resultat leveres for lavest mulig pris til kunder. Da må prislappen på arbeidet kunne forsvare hva kunden sitter igjen med. Tønnesen mener den ekstra kostnaden VR funksjonaliteten kombinert med LiDAR skanning medførte ikke forsvarte hva kunden satt igjen med da et nesten like bra og mye billigere alternativ var 3D modellering av eksisterende materie som utforskes i et vanlig modelleringsprogram på skjerm. Tønnesen påpeker flere begrensende faktorer for prosjektet. Hovedutfordringen er prislappen selve scanningen og VR implementasjonen medfører da scanneren i seg selv er ekstremt kostbar. I tillegg krever det ekspertise for de som skal utføre selve scanningen. En annen utfordring var de enorme datamengdene man sitter igjen med etter en scan av høy nøyaktighet. For å lettere håndtere datamengden ble de ferdige modellene delt opp i mindre deler da størrelsen på filene gjorde de nesten uhåndterlige. Denne dataen må behandles for å kunne gjenskape en nøyaktig modell av den skannede materien. Dette var tidskrevende og krever mye datakraft. Tønnesen påpeker at de også følte de manglet god software for å integrere scannene med VR. Det fantes ingen intuitiv og god tilgjengelig software for dette med god nok kvalitet. For at VR utforskning av LiDAR skanner skal bli brukt på større skala må det være minst like billig og gi bedre resultater enn for eksempel å gå igjennom en 3D modell av byggmaterie med kunden på en skjerm. Tønnesen er klar på at denne integrasjonen og skanningen er fremtiden, men inntil utfordringene nevnt ovenfor er håndtert, er denne typen prosjekter satt på vent i Sweco da det ikke er lønnsomt. Foreløpig står ikke krav av ekspertise og kost står ikke i samsvar med resultater det gir.

Fremtidige bruksområder for teknologien.

I tillegg til de eksisterende bruksområdene, ser Tønnesen for seg at teknologien i fremtiden kan brukes som et alternativ til å dra på befaring på en byggeplass. En kommuneansatt som skal godkjenne en byggesøknad kan enkelt befare byggeplassen uten å forlate kontoret for for eksempel å se hvordan en ferdig hytte på en bestemt holme vil se ut. Tønnesen ser også for seg at VR briller også kan brukes som et verktøy mens man er fysisk til stede på en befaring. Ved å ta på seg VR brillene kan den på befaring se hvordan det ferdige produktet vil se ut på den eksisterende tomten og bevege seg rundt i det. Tønnesen legger til at han tror teknologien vil brukes i mye større grad i fremtiden og at det kun er fantasien som setter grenser for dens bruksområder. Tønnesen påpeker at han følger spent med på utviklingen av håndholdte 3D skannere, men tror at de for mange industrielle formål der millimeterpresisjon er nødvendig, ikke vil bli gode nok.

B.2 Møte med Head of Loss Prevention på Gard og senior surveyors - 09.03.2021

Møte med Marius Schønberg, Bjarne Augestad og Per Haveland i forbindelse med masteroppgave.

Til stede: Jostein Sætra Schefte, Einar Lorentsen, Marius Schønberg, Bjarne Augestad og Per Haveland

Mål for møtet: Marius Schønberg er vice president og head of loss prevention på Gard. Loss prevention hos Gard inkluderer blant annet gjennomføring av inspeksjoner på skip, gjennomføre trening og kurs for klienter og sørge for at klienter har tilgang til oppdatert informasjon og tekniske råd. Bjarne Augestad og Per Haveland er senior surveyors hos Gard. Jobbene deres går blant annet ut på å gjennomføre inspeksjoner på skip enten som del av rutine eller som en såkalt «entry survey» som gjennomføres hvis det er et skip som ikke er inspisert eller har tegnet forsikring hos Gard tidligere. Målet for møtet er å få klarhet i hvordan skipsinspeksjoner gjennomføres i dag, hvilke utfordringer de fører med seg samt en diskusjon rundt og en demonstrasjon av en VR-applikasjon for virtuell inspeksjon av LiDAR skanner.

Når gjennomføres en inspeksjon av et skip?

Augestad og Haveland forklarer at det finnes to typer inspeksjoner Gard gjennomfører dag; entry surveys og condition surveys. Inspeksjonene foretar seg de samme kravene og skjemaene, men grunnen til at inspeksjonene utføres er forskjellige. En condition survey kan gjennomføres dersom Gard f.eks. kjører en kampanje mot visse typer skip eller har skjellig grunn til mistanke om mangler varslet fra noen som har vært om bord på et skip eller innsendte claims som vekker mistanke om mangler. En entry survey er Gard forpliktet til å gjennomføre når et reder ønsker å tegne forsikring hos Gard skip som ikke har vært forsikret tidligere. Før skipet får tegne forsikring hos Gard må det ha alle sertifikater de krever i orden og bestå en entry survey gjennomført av inspektører fra Gard.

Hvordan gjennomføres en inspeksjon av et skip?

Augestad forteller at før avreise bestilles hotellopphold og flybilletter. Inspektøren som skal gjennomføre inspeksjonen vil i forkant gjøre en forstudie. Han vil sette seg ned og gå over intern informasjon Gard sitter på om rederiet og skipet som for eksempel skadehistorikk på skipet. Viktig informasjon er hvilke type skip det er og hvordan det trader. For eksempel hvis det er en bulk båt som går ut jevnlig fra Brasil, sier Augestad at han da vet at skipet frakter jernmalm ofte og at han derfor bør ned i tankene på skipet og sjekke ekstra nøye siden dette er tung last.

Etter forstudien vil inspektøren ta kontakt med en agent som vil gi alle tillatelser og tilganger nødvendige for å komme om bord på skipet. Inspektøren flyr så ned og oppsøker skipet der det kommer i havn. Inspektøren vil så gå om bord i skipet å ha et kort møte med kapteinen for å forklare hvem han er, og hva han vil se. Kapteinen vil så klargjøre skipet for inspeksjon, som for eksempel å åpne og lufte ut de nødvendige tankene. Under inspeksjonen går inspektøren alltid sammen med ansatte på skipet, ofte en førstestyrmann eller maskinsjef for å ha en kontinuerlig diskusjon angående funn og for å ha åpenhet rundt eventuelle avvik som vil havne i rapporten.

Hva ser en inspektør etter i en vanlig inspeksjon?

Augestad og Haveland forteller at det er bestemte ting som skal ses etter ut ifra Gards krav til de forskjellige typer skip som vil tegne forsikring hos dem. Inspeksjonen forutsetter at skipet på forhånd har alle de nødvendige sertifikater Gard stiller krav til. Listen over hva inspektøren kommer til å se etter avhenger av type skip som skal inspiseres, og ligger åpent på nett på https://www.gard.no/web/forms så alle parter er klare over hva som skal sjekkes. Augestad trekker spesielt frem at erfaring og intuisjon innen skipsfart hos den som utfører inspeksjonen er viktig. Det viktigste å avklare på et skip er strukturelle feil, da disse potensielt kan føre til at et skip synker, som vil gi store tap for Gard. Elementer som kan si mye om skipets strukturelle tilstand er sprekker, lekkasjer, rust, midlertidige reparasjoner, pumper, tanker, luker og rør. I tillegg skal mannskapets rutiner og kunnskap sjekkes.

Augestad forteller at skip er i kontinuerlig drift, og derfor utføres også reparasjoner kontinuerlig. At for eksempel en hjelpemotor kan ligge demontert betyr ikke nødvendigvis at det er noe galt, kanskje heller at mannskapet gjør jobben sin ved å sørge for godt vedlikehold. Augestad forteller at god kommunikasjon derfor er viktig under en inspeksjon. En inspektør vil alltid ha med seg for eksempel en førstestyrmann (på dekk) eller en maskinsjef under en inspeksjon og ha en kontinuerlig dialog med dem for å kunne avklare funn og ha åpenhet hvis det avdekkes mangler som vil komme i inspeksjonsrapporten. De om bord må forstå hva som kritiseres og hvorfor.

Haveland forteller at en inspektør kontinuerlig tar bilder under hele inspeksjonen, også av dokumenter og loggbøker. Bilder underveis er veldig viktig siden Gard bruker mange inspektører fra hele verden og alle vurderer kvalitet forskjellig. Haveland forteller at han ofte ser ting på bilder som ikke har blir sett under inspeksjonen. Siden en inspeksjon kun skal ta en dag, og Gard forsikrer mange store skip, er det mye som skal inspiseres på kort tid som kan føre til at ting blir oversett når inspektøren er på skipet.

Inspeksjonsrapporten vil fylles inn både underveis inspeksjonen og etterpå. Etter rapporten er ferdig og mangler er ført inn sendes en defektliste til rederiet for kommentarer. Her får rederiet en sjanse til å forsvare manglende og fortelle hvordan de vil utbedre de. Etter saksbehandlingen er ferdig vil skipet få en risikorating på forskjellige områder, som konkluderer i en grønn, gul eller rød rating på hele skipet. Hvis det oppdages kritiske mangler, vil dette gi begrensinger i hva Gard vil dekke. For eksempel kan man velge å nekte skipet dekke for våtkrav hvis det er funnet kritiske mangler på luker.

Hvilke utfordringer møter en inspektør på under og etter inspeksjoner?

Augestad påpeker at shippingbransjen er en konservativ bransje og at det er mye som enda ikke er digitalisert. For eksempel stilles det mye krav til at man har de rette papirene for å få tilgang til skipene. Viktige elementer som skal inspiseres får man ofte ikke mulighet til å inspisere, da det ikke alltid er mulig å komme seg ned i de tankene man ønsker siden disse kan være lastet med noe. På tankskip er det også store begrensinger på hvor man har lov til å ta bilder siden elektriske apparater kan være forbudt i visse soner på grunn av eksplosjonsfare. Augestad anslår at det på et stort tankskip kan inkludere opptil 20% av arealet på skipet. Gard ønsker ikke å påføre sine medlemmer økonomiske tap, så om en vil ned i en bestemt tank som er full eller ønsker bilder av noe i en sone med fotografiforbud, må inspektøren vente til skipet er på verksted eller ikke lenger er lastet med gass.

Hvordan har inspeksjoner blitt gjort under covid-19?

Augestad forteller at Gard i år har hatt en god fornyelse av tegnede forsikringer i tillegg til mange nye skip som har tegnet forsikring. Covid-19 har gjort det vanskelig å få gjennomført fysiske inspeksjoner. Gard har gjort forsøk med virtuelle inspeksjoner. Da har inspektøren satt seg ned med reder og gått igjennom deres dokumentasjon av eget skip, det vil si stillbilder. Gard har nå konkludert med at denne type inspeksjon ikke var en tilfredsstillende og stoppet dem, siden reder kan velge hva han vil vise inspektør, eller vise gammelt materiale. Ved tidspunktet møtet holdes har Gard stoppet alle, både fysiske og virtuelle inspeksjoner i påvente av at det skal bli mulig å gjenoppta fysiske inspeksjoner. Siden mange nye skip har tegnet forsikring i tillegg til en god nytegning av eksisterende forsikringer sitter Gard med et stort etterslep av utestående inspeksjoner på grunn av covid-19.

Demonstrasjon av VR applikasjon. Hva er inspektørenes tanker om bruk av VR for å gjennomføre virtuelle inspeksjoner?

Augestad og Haveland forteller at Gard gjennomførte 200 inspeksjoner i fjor. Gard forsikrer 12 000 store skip, og mange flere hvis man regner med de mindre. Augestad anslår at rundt 10% av skip drives i gråsonen. Dette vil si at 90% av alle skip Gard forsikrer kun vil ha mindre eller ingen avvik ved inspeksjon. Ved å ha en rask tur ombord vil en inspektør raskt få inntrykk av om skipet driftes godt eller om det bør undersøkes nærmere. Augestad og Haveland tror ikke virtuell inspeksjon vil kunne erstatte fysiske inspeksjoner, men mener det kan være et godt sorteringsverktøy for å avgjøre om et skip er verdt å besøke eller ikke siden de fleste skip er i god stand og en virtuell applikasjon kan gi si mye om det generelle inntrykket av skipets tilstand.

Haveland forteller at de har begynt å eksperimentere med 360 bilder (Insta 360 One X2), og synes kombinasjonen av LiDAR og 360 graders bilder virker lovende. Haveland og Augestad påpeker at de er nysgjerrige på kvalitetene av bildene og scannene. Hvor enkelt er det for eksempel å se rust og sprekker? De er også bekymrede for hvor store datamengdene på scannene kan bli siden Gard forsikrer mange store skip. Haveland og Augestad lurer også på hvor lett det vil være å scanne inn et skip på en god måte, og presiserer at applikasjonen må være brukervennlig.

Haveland og Augestad ser i tillegg til det å være et sorteringsverktøy for skip, opplæring som et godt nytteområde for VR applikasjonen. Før et mannskap på et skip får lovt til å begynne å jobbe må han være kjent med layouten på skipet, skipets sikkerhetsrutiner, rømningsveier og ha gjennomført en brannøvelse. Augestad påpeker at med en virtuell applikasjon kan mannskap gjennomføre og trene på dette før de er om bord på skipet slik at de kan begynne å jobbe i det øyeblikket de går om bord. Siden mange skip har høy rotasjon på mannskap, brukes det i dag mye tid på opplæring og trening om bord. Haveland tror også en virtuell applikasjon kan brukes for mer spesifikk scenariotrening, som for eksempel ved å demonstrere diverse verktøy om bord, og hvor de befinner seg.

B.3 Interview with Björn Mes at Damen Naval - 15.03.2021

Meeting with Björn Mes about VR/AR development at Damen Schelde Naval Shipbuilding.

Meeting participants: Einar Lorentsen, Jostein Sætra Schefte and Björn Mes.
Agenda: Björn Mes works as technical specialist for VR/AR at Damen Schelde Naval Shipbuilding, a Dutch shipyard company providing maritime solutions through design, shipbuilding, ship

repair and related services. Mes is a team leader for a small development team creating VR/AR applications.

Can you tell us more about the project the article (Virtual and Augmented Reality in Shipbuilding) was about?

Mes tells that the project was a proof-of-concept to find out if VR could be used for training at different scenarios. In the project they built a virtual training ground for specific technical equipment at a selected ship. The project proved successful, and a pilot project was started soon after. For the pilot project, Mes explains that his team created a replica of a 100-meter long ship in VR in order to test building bigger ship models. This was done by hand and was very tedious. However, the result was a finished product which Damen today sells to its customers.

What is provided in the VR package Damen sell to costumers?

Mes tells that the VR models are used for training purposes. They provide maintenance training which can be specialized for the costumers for individual ships. When a ship is sold, a customer can choose from a wide variety of VR training packages in addition to requesting custom scenario packages.

How do you build the models of the vessels used in the virtual reality platform?

Mes explains that the models created in the article was done by hand, which was both time consuming and tedious. Mes's team converted CAD models of the ship to 3D models supported by Unity. When doing this, the team had to clean up the textures and faces in order to make sure that it was possible for the technology at the time to run the application. Mes estimates that it took about 3500 hours to convert the models.

After completing the two projects, Mes wanted to reduce the time it took to convert the models. Since Damen Naval are building the ships they provide VR packages for, Damen has all data which describes each vessel. Mes's team realized that they did not need to manually extract 3D models from the CAD files, but could rather automatically generate the correct models based on the data already available to Damen. As an example, Mes tells that what once took 2-3 weeks to convert and clean up all the models of an engine room manually now takes about a minute. However, his team still have to mark and label objects manually in addition to some minor manual clean up of the vessel model.

Mes explains that his team generate the models automatically by accessing a database using Python, and find out if the object is a pipe, steel or equipment. If it is a pipe, they don't export the geometry, but rather get the XML file and generate a model in Blender based on the this data. To convert difficult models they use Pixyz, which is a plugin for Unity, used to prepare and optimize large CAD models used in Unity. This is because they can't automate big models easily, like the main engine, so they have to use other tools to help according to Mes. If some of the models are updated, Mes tells that Python keeps track of these changes and exports a new model.

Another advantage of having access to the metadata describing the ship is that you can automatically link systems that belong to eachother togheter. By doing this they can highlight whole systems, which from a didactic point-of-view very interesing according to Mes.

Are there any main challenges or difficulties to overcome when creating the models and using VR?

Mes tells that when they started using VR they had to connect the VR-headset to expensive gaming computers, and this limited the movement that was possible inside the model without using teleportation. This problem is now fixed by the Oculus Quest VR-headset which run wireless. Another problem encountered was system confusion caused by light. Infrared light was used to track the player, and with reflection of light when there were many windows present caused problems. This problem is also fixed as inside out tracking is used instead of outside in tracking.

Another problem is that it takes years to build and deliver vessels. As time goes by VR-headsets are updated, and the headset the application is meant for is not available anymore.

There are also problems related to the human factor tells Mes: “How can you make sure that everyone understands the interface? How can you make sure you don’t overfeed them with stuff?”. Another problem is related to motion sickness in VR, in which Mes tells that you have to abide certain rules. They rules Mes states are: “never control the camera of the player, . . . and keep the frame rate as high as possible. Make your models and textures optimized so that you can have at least 90 frames per second. . . . and don’t make it an experience longer than 15-30 minutes.” Mes has also experienced doubt related to the realism of VR. An example he comes up with is the absence of both sway that is present on a ship and the smell of oil that is not experienced in a virtual model.

Challenges Mes tells that Damen Naval have now is related to remote sensing, and how the latency effect this and what happens if the connection is lost. The safety of data strings sent from a ship to shore for a remote sensing session is also a challenge.

How do you optimize the models so it’s possible to run them smoothly?

Mes tells that they keep both the number of polygons low and the level of detail in the textures low to optimize performance. They also divide their ships into segments, use occlusion culling, and hide everything that is not visible. However, there are some edge cases that needs to be handled according to Mes. For instance, at a long hallway from the back of the ship to the front the tricks they use will not work. They also use game technology such as mipmaps, bake light before running the application to avoid live lighting, and keep the effects to the minimum. This is to avoid frame rate problems according to Mes.

Do you have any new VR projects you are working on?

Mes tells that they work on the scientific reasoning to use VR instead of real training. Another area they are interested in is adding simulations that behaves like a real system. This can be extended to help suppliers test their equipment in a VR environment.

What sort of display are you using to experience VR (head-mounted display, a single projection screen or multiple projection screen)?

Mes tells that they only use head-mounted displays. Before they used HTC Vive, but now they are using Oculus Quest 1 & 2. If a costumer needs a really specific training scenario that needs a lot of power, they have the option to use the Link cable with Oculus Quest, which connects the Quest headset to a computer in order to utilize its power.

Mes follows up by explaining that they don’t use projection screens. This is because if a multi-user experience is required, only one can be the master controlling what can be seen for the other users which become the slave. This caused a bad experience according to Mes. The systems also cost a lot of money compared to head-mounted displays.

Do you use VR in the shipbuilding phase?

Mes tells that they do, because VR is good to visualize something that hasn’t been built yet. So they use VR in the design phase as well. One time Mes remembers that they had a meeting in VR with a client where they went through the ship to discover if the dimensions of the ship was satisfactory. Mes tells that: “you can see stuff in VR that you can easily miss on the 2D plane.”. Once they also had a VR session with engineers, and they discovered things they missed on the drawing because with VR you have more sense of the depth of size and your own location tells Mes.

Do you experience any increased interest from your costumers regarding VR?

Mes tells that they experience increased interest, especially by the northern European countries, and the NATO clients.

How realistic do you feel the models is compared to the real ship?

The shapes constructed are located at the exact same place and at the original size as the real ship, which cause realism according to Mes. Since the distances, sizes and positions of the buttons are correct, you will recognize this in the real ship even though the ship doesn't have the same lightning or textures. Mes tells that the textures aren't like photographs yet, but they try to make the textures as real as possible. He explains that he once got onboard a vessel he had worked on in VR for an extended period of time, and that it was a weird feeling knowing the ins and outs of the ships layout even though he had never been on it before.

B.4 Intervju med Stener Olav Stenersen hos DNV - 15.04.2021

Møte med Stener Olav Stenersen som jobber i DNV.

Til stede: Einar Lorentsen, Jostein Sætra Schefte, og Stener Olav Stenersen
Mål for møtet: Stener Olav Stenersen er ansvarlig for maritime enheter i drift hos DNV. DNV jobber med kvalitetssikring og risikohåndtering til blant annet den maritime sektoren. De leverer klassifisering, sertifisering, teknisk risiko- og pålitelighetsanalyse sammen med programvare, datahåndtering og uavhengig ekspertrådgivning til blant annet den maritime sektoren. DNV har også utviklet ulike metoder for eksterne undersøkelser av skip, og leverer tjenester for dette. Målet for møtet er å finne mer ut hvordan DNV utfører eksterne undersøkelser av skip, og høre mer om prosjektet de hadde om bruk av VR briller til eksterne undersøkelser beskrevet i artikkelen (Virtual Reality: Remote cargo ship surveys).

Hva slags inspeksjoner gjennomfører DNV?

Stenersen forteller at DNV maritim har en rekke regler som designere av båter må følge. Tegningene må bli godkjent av DNV, og deretter følger DNV opp at båten faktisk blir bygget i henhold til tegningene. Disse inspeksjonene blir gjennomført av en «sight-surveyor», og hvis det blir godkjent får båten et klassesertifikat som er gyldig i 5 år, i tillegg til andre sertifikater som båten trenger. Disse sertifikatene har også en gyldighet på 5 år, og sertifikatene forutsetter at det blir gjennomført en undersøkelse av båten årlig. Disse undersøkelsene gjøres om bord et skip eller ved bruk av andre hjelpemidler, der man ser om alt er i orden og fungerer i henhold til minimumskravene.

I et ideelt tilfelle er DNV om bord skipet en gang i året, men det skje ting som: skader på skip, skader forårsaket av skip, skade på maskineri og utstyr når skip blir eldre. Dette skal rapporteres til klassen, og da skal DNV undersøke hva som har skjedd, om det er greit å seile med, og om det har blitt reparert.

Hvorfor startet DNV med eksterne undersøkelser (remote surveys)?

Stenersen forteller at dette ble gjort for å gjennomføre «occasional surveys», altså de undersøkelsene som kommer mellom de periodiske årlige undersøkelsene. Disse undersøkelsene tar optimalt sett kun noen timer, og de må reise langt for å se på forholdsvis enkle ting. Dette ble gjort remote, og det ble gjennomført med hjelp av dokumentasjon, arbeidsrapporter, bilder, og andre ting som var tilgjengelig. Tilbudet har blitt oppgradert til å også inkludere streamet video på visse ting.

I år kan Stenersen fortelle at DNV har prøvd å utføre større eksterne undersøkelser gjennom å streame video fra skipene. Dette medfører at de må lære opp folk om bord til å utføre inspeksjonen slik DNV vil, noe som igjen medfører at inspeksjonen i seg selv tar lenger tid.

Hvordan går det med VR-prosjektet det ble skrevet om i artikkelen (Virtual Reality: Remote cargo ship surveys)?

Stenersen forteller at det var en test utført av DNVs tyske kontor i 2018, men som ikke DNV har gått videre med. Ifølge Stenersen har det ikke vært behov for denne typen inspeksjon enda, og det er knyttet store investeringer til prosjektet. DNV har et godt utbygd nettverk både i Norge og resten av verden i knutepunktene for skipstrafikk, så det har ikke vært et problem å få tak i surveyere som kan utføre inspeksjoner.

Hovedutfordringen Stenersen nevner med å gjøre ting eksternt for øyeblikket er at skipene ikke har utstyr om bord til å utføre undersøkelser. Man kan ikke ta det for gitt at skipet har en bærbar PC med webkamera eller en smarttelefon som fungerer. Men, Stenersen sier at dette er i ferd med å endre seg, særlig på grunn av pandemien. Andre problemer er at diverse flaggstater og havnestater som fremdeles noe skeptisk, i tillegg til at å DNV har erfart at å gjøre ting eksternt tar ganske lang tid. Stenersen forteller at i en survey så ser man ikke på absolutt alt, men heller en overflatesjekk der man går videre inn på ting hvis det er et problem. Så en survey er ikke det samme som en inspeksjon forteller Stenersen.

Et annet problem Stenersen nevner med remote surveys er at man oppholder flere arbeidere på båten enn man ville gjort i en fysisk inspeksjon, der man kun som oftest går med ett crew-medlem. Dette gjør at det plasseres en ekstra byrde som legges på crewet som må vurderes.

Stenersen forteller også at de får en bedre oversikt og kommunikasjon hvis man er fysisk om bord et skip, så han mener at næringen ikke er helt moden enda.

Hva skal til for at næringen skal bli moden for en overgang til remote surveys?

Statusen i DNV er at de vil fysisk om bord et skip en gang i året, men utover det har de lyst til å gjøre mer eksternt. Stenersen mener at det som skal til for at næringen skal bli klar for en overgang er tilgjengelig internettilgang slik at man kan sende bilder og signaler i tillegg til data.

Med data mener Stenersen at man kan se data fra sensorer som forteller noe om f.eks. eksostemperaturen og forbruket på en motor. Hvis dette skulle vært noe rart med dataen de får, må de sjekke nærmere opp i hva som er galt istedenfor å uansett sjekke om det er noe galt med motoren.

Er det noen begrensninger på størrelsen på skipet når dere skal gjennomføre en remote survey?

Ifølge Stenersen er det type båt som bestemmer. En stor containerbåt på 300-400 meter har masse systemer overalt, inkludert heiser, og det er vanskelig å komme til pga. containere. Dette er utfordringene på de type båtene ifølge Stenersen. En annen type skip er tankskip, der det mest avanserte er motorrommet, brobygning og pumpeutstyr. Resten består av stål, og skipene i seg selv er veldig store. Jobben til DNV, spesielt på 5-årige surveys, er blant annet å se om «coating» er intakt, om det er sprekker osv. Mindre skip, som avanserte supplybåter i Nordsjøen, er omtrent 100 meter lange og inneholder masse utstyr. Så Stenersen nevner at det er veldig mye på disse båtene, men kan ikke gi et godt svar på om det er noen begrensninger knyttet til gjennomføring av remote survey.

Hvilken oppgave har DNV i motsetning til klassiske forsikringsselskaper?

Stenersen sier at en klasse oppstod som følge av at man trengte en uavhengig tredjepart som kunne si om båten var i orden. Forsikringsselskapene startet klassevirksomheten, og har en lignende oppgave som EU-kontroll har på biler.

Gjennomfører DNV fysiske inspeksjoner nå, eller har det blitt satt på vent pga. COVID-19?

Stenersen forteller at fysiske inspeksjoner har blitt gjennomført, og at det spesielt i mars har økt. Fysiske inspeksjoner har blitt gjennomført hele tiden, men det er pga. DNV opererer i hele verden og ved å følge lokale helsemyndigheters retningslinjer har det vært mulig. Stenersen sier at DNV ikke klarer å erstatte årlige fysiske inspeksjoner med eksterne inspeksjoner, men at de har prøvd med en håndfull eksterne inspeksjoner.

Har DNV sett på muligheten til å bruke LiDAR sensorer på inspeksjoner?

Stenersen forteller at DNV har sett på det, og synes det er interessant å bygge en modell som et sammenligningsverktøy på tilstanden til båten. Det å kunne lage slike modeller på en enkel og billig måte tror Stenersen er viktig for at det skal bli brukt, og er det Stenersen tenker er mest interessant. Det at hvem som helst kan lage en modell av båten kan bli veldig nyttig. Men dette mener han hvordan båten har forandret seg i løpet av årene, hvordan så skipet ut før skaden osv. Bilder kan også utgjøre sammenligningsgrunnlaget ifølge Stenersen.

Stenersen forteller at DNV har noe de kaller Veracity, som er en datadelingsplattform. Tanken er at en modell av skipet kan deles på denne plattformen fra eieren av skipet, og modellen kan dermed brukes som en referansemodell for å se på endringer som har skjedd.

Stenersen sier at DNV ikke tror at remote surveys kan erstatte fysiske inspeksjoner fullstendig, men at mange inspeksjoner utenom de årlige inspeksjonene kan bli erstattet.

Stenersen sier også at remote surveys kan f.eks. fungere som en midlertidig tillatelse til å seile til den har kommet til en havn der en fysisk inspeksjon kan foregå. Så istedenfor å reise langt og oppholde båten, så kan man gi en midlertidig tillatelse hvis forholdene tillater det.

Hva slags utstyr bruker dere når dere gjennomfører remote surveys?

Stenersen sier at de bruker mobiltelefoner, men at de også kan bruke GoPro løsninger også. I framtiden spår Stenersen at briller også kan bli brukt, ettersom å holde en telefon i lengden kan være slitsomt og at to hender er optimalt når man beveger seg i skipet.

Hva slags trening tilbyr DNV?

Stenersen forteller at DNV har en modell som heter SuSi (Survey Simulator), som ble laget i Polen. Det var knyttet veldig store forventninger til simulatoren, fordi man kunne øve på å være en surveyor. Stenersen forteller derimot at det har blitt mindre snakk om simulatoren med tiden.

B.5 Intervju med surveyors 2 - 20.04.2021

Møte med Bjarne Augestad og Per Haveland i forbindelse med masteroppgave.

Til stede: Jostein Sætra Schefte, Einar Lorentsen, Bjarne Augestad og Per Haveland

Mål for møtet: Bjarne Augestad og Per Haveland er senior surveyors hos Gard. Jobbene deres går blant annet ut på å gjennomføre inspeksjoner på skip enten som del av rutine eller som en såkalt «entry survey» som gjennomføres hvis det er et skip som ikke er inspisert eller har tegnet forsikring hos Gard tidligere. Målet for møtet er å gi Bjarne og Per en demonstrasjon av applikasjonens funksjonalitet, få feedback og finne ut hvilke områder som er mest interessante for dem å skanne på et skip for senere å kunne få en full demo av dette.

Hva er status på gjennomføring av fysiske inspeksjoner?

Haveland forteller at fysiske inspeksjoner nå gjennomføres igjen, men i svært begrenset grad på grunn av covid-19. De har fortsatt et voksende antall utestående inspeksjoner.

Hva er mest interessant å få scannet inn på et skip for å teste applikasjonen fra et ansvars og kaskoforsikringsperspektiv?

Augestad trekker nevner spesifikt tanker når det gjelder ansvarsforsikring. Han er nysgjerrig på hvor godt rust vil vises på LiDAR scan og 360 bilder. For kaskoforsikring blir maskinrom trukket frem som det mest attraktive. Augestad og Haveland nevner at man som studenter sannsynligvis ikke får lov til å forstyrre skipets operative drift, og at det derfor ikke blir mulighet til å gå ned i tanker. I tillegg vil det være begrenset tid om bord i båten, som vil si at visse områder må prioriteres. De ønsker derfor skanner av maskinrom, bro og dekk. Broen har en del måleapparater som Augestad og Haveland er nysgjerrige på hvor godt vil vises på scannene og bildene. Maskinrommet har en kompleks geometri med mye detaljer og er et rom som alltid sjekkes i en inspeksjon. På dekk er surveyorne nysgjerrige på hvor godt det blir mulig vurdere skipets livsikrings og brannslukkingsutstyr som for eksempel livbåter og annet redningsutstyr. De foreslår også å sette seg inn i tilgjengelig dokumentasjon på skipet som skal undersøkes før man går om bord.

B.6 Intervju med surveyors 3 - 20.05.2021

Møte med Marius Schønberg, Bjarne Augestad og Per Haveland i forbindelse med masteroppgave.

Til stede: Jostein Sætra Schefte, Einar Lorentsen, Marius Schønberg, Bjarne Augestad og Per Haveland

Mål for møtet: Marius Schønberg er Vice president og head of loss prevention hos Gard. Bjarne Augestad og Per Haveland er senior surveyors hos Gard. Målet for møtet er å gi Marius, Bjarne og Per en live demonstrasjon av den ferdige applikasjonens og få tilbakemeldinger på resultatet.

Marius, Bjarne og Per får en live demonstrasjon av applikasjonen. Jostein og Einar kjører applikasjonen i VR brillene og streamer synsfeltet sitt slik at Marius, Bjarne og Per kan dirigere dem rundt i applikasjonen så de får undersøkt det de vil se.

Tilbakemeldinger underveis i demonstrasjonen

Augestad kommenterer at han synes 360 graders bildene har høy kvalitet. Han legger til at han synes kombinasjonen av LiDAR skanner og bilder er fin, selv om han i enkelte scener som for eksempel i maskinrommet på Kristian With, synes LiDAR scannen blir for grovkornet. Ifølge Augestad kan han tydelig se relevante detaljer fra 360-bildene som f.eks hvordan lasten er sikret på dekk, kvaliteten på isolasjon i brennstoffrør, lekkasjer, oljesøl, rust, sprekker og andre detaljer er relevante for surveyere. Augetsad synes informasjonspunktene virker nyttige, da de kan gi rask og oversiktlig informasjon om objektene brukere ser. Schønberg legger til at applikasjonen virker som et nyttig hjelpemiddel for å få overblikk over situasjonen på et skip. For Schønberg er også bærekraftperspektivet viktig, da man med et slikt verktøy kan undersøke fartøy uten å reise fysisk. Han synes det er interessant å se ideer til metoder som gir bedre eksterne undersøkelser og kanskje på sikt kan bli like gode som å ha en mann om bord.

Hvilke bruksområder ser dere for dere at denne applikasjonen og teknologien kan brukes til?

I tillegg til å kunne brukes for å få oversikt over situasjonen på et skip nevner Augestad bruk av applikasjonen fra et inspeksjonsteknisk ståsted. Ofte får et rederi rapporter etter endt inspeksjon med defekter som må utbedres eller repareres, som f.eks at lukepakninger er for dårlige og må skiftes. Han mener applikasjonen vil kunne brukes til å kontrollere at utbedringene har blitt utført. Dette kan gjøres ved at man har en scan fra utført inspeksjon der feilene er til stede og ber om nytt materiale når rederiet sier utbedringene er gjennomført. En surveyor vil da kunne verifisere at utbedringene er utført virtuelt i stedet for å fysisk oppsøke skipet, som vil være en umiddelbar og intuitiv måte å kontrollere det på. Schønberg legger til at man ved å sitte på flere skanner vil kunne følge et skips utvikling over tid, som f.eks slitasje og generelt forfall.

Augestad synes også kombinasjonen med LiDAR scan og bilder er interessant i et kaskoperspektiv. Ved å bruke informasjonspunkter som henter informasjon om objekter om bord fra databaser vil inspektører raskt få relevant informasjon om utstyr på skipet. Haveland sier at når det gjelder bruksområder er det kun fantasien som setter grenser. Han ser for seg at applikasjonen og teknologien vil fungere godt til opplæring og familisering for mannskap. Hvis f.eks en reperatør skal om bord på et skip kan han ta på VR brillene, koble applikasjonen online og snakke med maskinsjefen som kan peke og forklare hvor utstyret det er noe galt med er lokalisert og hva som er galt med det. Haveland påpeker at slike virituelle miljøer som blir utforsket i applikasjonen kan være like nyttige for både rederier og surveyere, men til forskjellig formål.

Hvilke endringer eller forbedringer ville dere gjort på applikasjonen?

Schønberg ønsker en videreutvikling der plantegninger over skipet blir lagt inn i applikasjonen. Da kan brukere ha tilgang til et kart mens de beveger seg rundt i LiDAR modellen og til enhver tid se posisjonen sin i plantegningene. En bruker vil da også få oversikt over objekter i miljøet han befinner seg i, f.eks hvis en bruker er i maskinrommet og beveger seg mot en motor kan han se på plantegningene og se at motoren han beveger seg mot er en hjelpemotor eller pumpen han beveger seg mot er en ballastpumpe. Haveland er enig i denne ideen og påpeker at han ser for seg lignende funksjonalitet som i et dataspill der en bruker til enhver tid har et lite kart tilgjengelig i synsfeltet.

Schønberg ser også for seg en videreutvikling der en inspektør kommuniserer med et medlem av mannskapet og får tilsendt/strømmet skanner han ber om i sanntid av mannskapet. Dette vil hindre mannskap i å lage fordelaktige skanner da surveyoren vil få tilsendt det han ber om. Schønberg mener kommunikasjon over nett er en forutsetning for at en remote survey skal bli bra. Ulempen med dette er at en slik løsning stiller store krav til dekning og båndbredde til både surveyor og mannskap, som kan være vanskelig og dyrt da det ofte er dårlig eller ingen dekning til sjøs og å sende data via satellitt fort blir dyrt.

Augestad ønsker i større grad mer detaljerte bilder. For eksempel hvis en bruker befinner seg i et bilde av et livredningsutstyr og omgivelsene til det ønsker han at man inne i dette bilde vil kunne trigge et nytt, mindre og mer detaljert 2-dimensjonalt bilde(r) av livreddingsutstyret. Dette vil gjøre bildepunktene mer interaktive og gi surveyere lettere tilgang til detaljert oversikt over enkelte objekter i et eksisterende bilde. En annen ide Augestad har er å koble applikasjonen opp mot skips eksisterende elektroniske sikkerhetsystemer slik at man til enhver tid har tilgang til alle enheters data og historikk. Dette vil gi en digital kopi av alle tekniske elementer og ville ifølge han selv vært fantastisk.

Er LiDAR scanen nødvendig for å få et godt overblikk over skipet?

Augestad forteller at han synes 360-bildene ga mer informasjon og var lettere å orientere seg i enn LiDAR scannen. Han synes LiDAR scannen i flere tilfeller, og da spesielt scannen av maskinrommet var for grovkornet, men synes informasjonspunktene i scannen virker veldig nyttige. Han påpeker også at LiDAR scannen enkelt gir informasjon om hvor de forskjellige bildene er tatt som kan være vanskelig å finne ut av hvis man kun blar igjennom bilder på en PC uten noen rammer å sette dem inn i. Alt i alt er kombinasjonen mellom LiDAR scan og bilder fin, men LiDAR scannene skulle gjerne vært bedre.

Gir applikasjonen i sin nåværende tilstand godt nok grunnlag for å kunne fungere som et sorteringsverktøy for å bestemme om et skip er verdt å undersøke fysisk eller ikke?

Augestad forteller at en inspeksjon er kompleks og består av flere elementer og risikovurdering. Han trekker frem at et system som ligner det som blir demonstrert etter hans mening vil kunne benyttes for enkelte av deres kunder i en «screening» setting. Det vil i midlertidig kreve en raffinering av Gards system for dokumentasjon og klare instrukser fra Gard til manskap om bord på skip om hvordan de skal utføre scanningen før det kan bli tatt aktivt i bruk.

Augestad trekker frem at det er andre enn fysiske ting som blir observert under en inspeksjon, som for eksempel en «human factor» som inkluderer arbeidsmiljø, sikkerhet og lastbehandling. Han sier at dette vil være vanskelig å monitorere digitalt, men at en screening rapport fra en slik type applikasjon vil kunne gi indikasjoner på om et skip er verdt å undersøke fysisk eller ikke.