Teleoperacija mobilnog robota korištenjem reprezentacije digitalnog blizanca temeljenoj na virtualnoj stvarnosti

Paladin, Mateo

Master's thesis / Diplomski rad

2025

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:817106

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-23**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 754

MOBILE ROBOT TELEOPERATION USING A VIRTUAL REALITY-BASED REPRESENTATION OF A DIGITAL TWIN

Mateo Paladin

Zagreb, February 2025

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 754

MOBILE ROBOT TELEOPERATION USING A VIRTUAL REALITY-BASED REPRESENTATION OF A DIGITAL TWIN

Mateo Paladin

Zagreb, February 2025

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 30 September 2024

MASTER THESIS ASSIGNMENT No. 754

Student: **Mateo Paladin (0036525128)**

Study: Computing

Profile: Network Science

Mentor: prof. Lea Skorin-Kapov, PhD

Title: **Mobile Robot Teleoperation using a Virtual Reality-based Representation of a Digital Twin**

Description:

A mobile robotic platform is a type of robot that can move and navigate through an environment, typically equipped with motors to enable movement, various sensors to perceive the environment (e.g., cameras, lidar, or ultrasonic sensors), computing systems, and communication modules that allow remote control (teleoperation) or integration into larger systems. Digital Twins (DT) are synchronized virtual representations of real-world entities and processes. An emerging approach for visualizing and interacting with DTs is the use of Virtual Reality (VR), enhancing the human ability to interact with and manage DTs. Your task is to design and implement a VR-based visual representation of the DT of a simple real-world environment containing a mobile robotic platform. Your task will be to design a VR-based interface for intuitive teleoperation of the robotic platform based on the Robot Operating System (ROS) Noetic. The interface should enable the VR user to use VR controllers or hand gestures to execute commands for real-time navigation and manipulation of the remote, and consequently, virtual robot, and to visualize a real-time video stream from the remote robot's camera in the virtual environment (VE). The position and movements of the robotic platform in the real-world environment should be synchronized with the virtual representation of the same platform in the VE. Your task will further include evaluation of the interface performance in terms of latency, movement smoothness, and Quality of Experience.

Submission date: 14 February 2025

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 30. rujna 2024.

DIPLOMSKI ZADATAK br. 754

Pristupnik:	**Mateo Paladin (0036525128)**
Studij:	Računarstvo
Profil:	Znanost o mrežama
Mentorica:	prof. dr. sc. Lea Skorin-Kapov

Zadatak: **Teleoperacija mobilnog robota korištenjem reprezentacije digitalnog blizanca temeljenoj na virtualnoj stvarnosti**

Opis zadatka:

Mobilna robotska platforma je vrsta robota koja se može kretati i navigirati kroz okruženje, obično opremljena motorima za omogućavanje kretanja, raznim senzorima za percepciju okoline (npr. kamerama, lidarom ili ultrazvučnim senzorima), računalnim sustavima i komunikacijskim modulima koji omogućuju daljinsko upravljanje (teleoperaciju) ili integraciju u veće sustave. Digitalni blizanci (engl. Digital Twin, DT) su sinkronizirani virtualni prikazi stvarnih entiteta i procesa. Novi pristup za vizualizaciju i interakciju s digitalnim blizancima je korištenje virtualne stvarnosti (engl. Virtual Reality, VR), koja poboljšava ljudsku sposobnost interakcije i upravljanja digitalnim blizancima. Vaš zadatak je dizajnirati i implementirati vizualni prikaz digitalnog blizanca jednostavnog stvarnog okruženja koje sadrži mobilnu robotsku platformu, koristeći VR. Nadalje, Vaš zadatak je dizajnirati VR sučelje za intuitivnu teleoperaciju robotske platforme korištenjem programske podrške Robot Operating System (ROS) Noetic. Sučelje treba omogućiti korisniku u VR-u da koristi VR kontrolere ili geste ruku za izvršavanje komandi za navigaciju i manipulaciju u stvarnom vremenu, kako daljinskim, tako i posljedično virtualnim robotom, te da vizualizira video prijenos u stvarnom vremenu s kamere daljinskog robota u virtualnom okruženju. Pozicija i kretanje robotske platforme u stvarnom okruženju trebaju biti sinkronizirani s virtualnim prikazom iste platforme u virtualnom okruženju. Konačno, Vaš zadatak će uključivati evaluaciju performansi sučelja u smislu latencije, glatkoće kretanja i iskustvene kvalitete.

Rok za predaju rada: 14. veljače 2025.

I would like to thank my mentor, Prof. Lea Skorin-Kapov, for her never-ending guidance and immense support throughout my studies. She helped me tremendously in becoming a better student and person.

I would also like to thank the professors and assistants who motivated and taught me during my academic journey, with a special mention to Prof. Mirko Sužnjević, who played a pivotal role in finding my passion for Virtual Reality.

Additionally, I would like to thank Damir Kljajić for the opportunity to work with ENT and for the provided equipment and guidance.

I am deeply grateful to my parents for their constant support in all my endeavors, not only in my academic career but in all aspects of my life.

Lastly, I want to thank all my friends and colleagues that made my academic journey educational, fun, and unforgettable.

Contents

Introduction 1

1. Theoretical Background 3

1.1. Mobile Robotics and Teleoperation 3

1.2. Digital Twins 3

1.3. Virtual Reality and Immersive Environments 4

1.4. Robot Operation System 5

2. Design of the VRobo prototype 7

2.1. Prototype Overview 7

2.1.1. Architecture Design 7

2.1.2. Communication Overview 8

2.2. VR Scene Design 10

2.3. ROS Design and Functionalities 11

3. Development of VRobo 13

3.1. Used Technologies and Tools 13

3.1.1. Unity 14

3.1.2. ROS 14

3.1.3. Additional Hardware 15

3.1.4. Connection via a private 5G network 16

3.2. System Architecture Implementation 17

3.3. VR Scene Implementation 18

3.3.1. Scene Modeling 18

3.3.2. VR Controller remapping 20

3.3.3. First Person View Mode 22

3.4. Implementation of ROS Functionalities 23

3.4.1. Teleoperation 23

3.4.2. Sensor Data 24
3.4.3. Error Handling Mechanisms 27
3.5. ROS-Unity Communication Implementation 27
3.5.1. Establishing a Connection 28
3.5.2. Data Transmission Protocols 31
3.5.3. Latency Optimization 33
3.6. Prototype Limitations 34
4. User Study 36
4.1. Methodology 37
4.1.1. Laboratory Setup 37
4.1.2. Test scenarios and procedure 38
4.1.3. Collection of subjective and objective metrics 40
4.1.4. Participants 41
4.2. Results and Discussion 42
4.2.1. Subjective metrics data 43
4.2.2. Objective metrics data 50
4.2.3. Limitations 53
Conclusion 55
References 56
List of Figures 59
List of Tables 61
Abbreviations 62
Appendix A. User Study Form 63
Abstract 75
Sažetak 76

Introduction

Robot teleoperation enables humans to control robots remotely. It is mostly used in environments where direct human presence is not possible or where human presence can be hazardous or impractical. Mobile robotic platforms, capable of navigating dynamic environments, are widely used in logistics [1], disaster responses [2] as well as in healthcare both in surgeries [3] and in helping people with disabilities [4]. Due to dynamic environments, automatization of robots to complete given tasks can be exceptionally hard and very often requires intuition of an expert human to solve them [5]. To achieve effective teleoperation, robot platforms rely on real-time sensor feedback and data, fast and precise algorithms, real-time actuation as well as a consumer-friendly interface that enhances the operator interaction with the robot [6]. Traditional teleoperation interfaces are often unable to provide an immersive user experience, which can condition operator performance in high-stakes scenarios such as disaster responses during fires or lifesaving surgeries. Virtual Reality (VR) technology can help further improve immersion for users using 3D environments that correspond to real space, enhancing situational awareness and reducing cognitive load [7],[8]. Digital Twin (DT) is a virtual model of a physical system, with the potential to enhance robot teleoperation by synchronizing a real-world robot and its virtual counterpart [9]. This helps the operator in better understanding distant scenarios, reduces risk, and enables predictive analysis [10].

This thesis explores the integration of VR and DT technologies for mobile teleoperation using Robot Operating System (ROS) Noetic [11] and a private 5G network. This thesis aims to design a VR-based interface for intuitive teleoperation of the robotic platform based on the ROS Noetic. The interface enables a VR user to use VR controllers to execute commands for real-time navigation and manipulation of the remote, and consequently, virtual robot, and to visualize a real-time video stream from the remote robot's camera in the Virtual Environment (VE). The thesis provides a review of existing research on mobile robot teleoperation, DTs, and VR-based interfaces. Furthermore, the design and implementation of a VR-based DT system using Unity and ROS for real-time interaction with a robot is presented. Finally, a user study is conducted to evaluate latency, movement smoothness, and Quality of Experience (QoE) for the developed prototype. This

work aims to contribute to the field of human-robot interaction by demonstrating the potential of VR and DTs in mobile robot teleoperation, bridging the gap between upcoming technologies with great promise and practical applications.

The thesis consists of six chapters and is organized as follows. The introductory chapter provides an overview of the thesis and is followed by a chapter that presents the theoretical background, including mobile robotics, teleoperation, DTs, VR, and ROS. The third chapter explains the design of the prototype, focusing on architecture, communication framework, and the VR scene. Chapter four focuses on the development of the prototype, creating the VR scene, used technologies and tools, and Unity-ROS integration. The following chapter is centered around a user study conducted to evaluate QoE, interface intuitiveness, and potential latency-related issues. The thesis is concluded by a summary that is based on analysis of the results and potential areas for future improvement. Finally, a list of references, figures, and abbreviations used within the thesis are also provided, along with the questionnaire that was used in the study and can be found in the appendix.

The thesis was completed in part in the scope of the project XR Communication and Interaction Through a Dynamically Updated Digital Twin of a Smart Space – DIGIPHY, funded by the European Union – NextGenerationEU, Grant NPOO.C3.2.R3-I1.04.0070. The work was conducted in collaboration with the company Ericsson Nikola Tesla, who is a partner in the DIGIPHY project.

1. Theoretical Background

1.1. Mobile Robotics and Teleoperation

Mobile robotics refers to robotic platforms that are capable of navigating and operating in different environments without the restraint of staying in a fixed location [6]. These systems are equipped with motors that enable their movement as well as different sensors that help with the perception of the environment around them. Robots are equipped with many different types of sensors, ranging from video cameras and ultrasonic sensors to advanced sensors such as lidars, enabling robots to better perceive the environment and help the operator in interacting with it [13].

Teleoperation is a method of controlling a robot remotely, usually by a real-time human operator [12]. It is mostly used in dangerous or inaccessible environments, such as disaster zones or outer space, where direct task solving is not possible for humans [2]. A teleoperated robot relies on a safe and fast communication link over which it receives commands from the operator while simultaneously sending sensor feedback back to the operator to help in understanding the situation and the environment better. Recent advancements in teleoperation systems have incorporated haptic feedback, allowing operators to feel the forces experienced by the robot, further enhancing precision and control [14]. Integration of teleoperation into mobile platforms introduces challenges such as high latency, situational awareness, and synchronization. Solving these challenges helps ensure stability in an unpredictable distant environment. To overcome these challenges, advanced movement algorithms, sensor fusion, and interactive user interfaces are created and used. They cannot completely solve the problems these challenges present but can come remarkably close to giving precise real-time insights to the operator about the robot's environment and its own status in it.

1.2. Digital Twins

A DT is a virtual replica of a physical entity that aims to represent the physical entity as close as possible. According to the Digital Twin Consortium, a DT is defined as "*an integrated data-driven virtual representation of real-world entities and processes, with*

synchronized interaction at a specified frequency and fidelity.." [44]. It can use monitoring, simulations, predictions, and optimizations in the created space to accurately represent events without having to execute the event in the physical space [15]. Furthermore, the use of DTs can give us the ability to test and simulate system performance in various scenarios and under various conditions. Originally created for engineering purposes, DTs have expanded into domains such as healthcare, urban planning and robotics [16]. DT is also a key factor in mobile robotics simulations as a data-dependent virtual model of a robotic platform and its environment can enable operators to have precise testing options due to its ability to visualize, interact, and simulate actions. Essential components of a DT are sensors and other Internet of Things (IoT) devices, which provide real-time data from the physical system. Using these components and machine learning inside DTs, it is possible to conduct complex simulations such as predicting robot behavior, evaluating control strategies and creating path simulations without the risk of damage on a physical robot [17].

1.3. Virtual Reality and Immersive Environments

Virtual reality (VR) is a computer-generated environment that lets users immerse themselves in a simulated space, usually using a head-mounted display (HMD) to separate them from the real world [18] [19]. VR environments enable users to interact with digital objects and spaces as though they were physically present in them, using visual, audio, and lately haptic stimuli portray the real-world sensations as close as possible. Immersion is a psychological state in which the user feels completely surrounded by, included in, and engaged with a VE that delivers a constant flow of stimuli and experiences [20].

Immersive environments are environments that engage users by simulating or augmenting reality, often using technologies such as VR, augmented reality (AR), and mixed reality (MR), all of which fall under the category of extended reality (XR). User interaction and perception may in certain cases be enhanced by combining various immersive technologies (e.g., AR, VR) rather than using just one [21]. By combining VR or other immersive realities with DTs, operators can control robots within a virtual representation of their real-world space. These technologies together can offer high levels of situational awareness, reducing cognitive load on users and improving their task speed and completion rate [7].

Relevant studies show that many teleoperation tasks over a VR interface can be successfully completed. However, they often take a longer time and struggle when it comes to high precision tasks comparing to direct human contact [39] [40]. Tasks that require high precision (e.g., block stacking of ten three-by-three centimeter blocks on top of each other [39]) had a low rate of completion in VR due to small errors made in VR being stacked until the tower falls. For easier tasks, the studies showed that participants were twice as slow in proximal teleoperation, and four times slower in VR teleoperation from a distant place compared to solving tasks without teleoperation. However, users improve their performance over time as they become more proficient with the VR interface. These insights emphasize the need and possibilities for further improvements in the VR interface to enhance speed and precision. They also show the importance of designing accessible and intuitive interfaces that enable users to learn and improve task solving skills faster and easier.

1.4. Robot Operation System

Robot operating system (ROS) is an open-source robot operating system that provides a structured communications layer above the host operating system [22]. It is used for development and deployment of robotic applications. It also provides a collection of tools, libraries and communication protocols that help users create a modular and scalable robot system.

Most notable features of ROS include a modular architecture that enables easy organization of applications with packages and nodes, as well as a communication framework that uses a publish-subscribe model to efficiently exchange data with other applications and systems. It also provides community support and Unity integration for creating virtual simulations of a robot's system and actions [23]. This connection enables real-time data streaming from the robot sensors to the VE and back in terms of a robot's movements. All this together with a VR DT can ensure accurate synchronization. It can also support teleoperation functions such as navigation and inspection of spaces. Using more advanced sensors can also mimic virtual actions in real life without action errors even in dynamic environments.

ROS supports a wide variety of hardware interfaces as well as microROS. MicroROS is a variant of ROS that runs natively on embedded microcontrollers running real time

operating systems [31]. ROS uses tools called ***topics*** and messages to help developers connect actuators and sensors with a robot's control system. All of the communication can be recorded using ROS bag files or logs for easy testing and quality assurance. All of this lets ROS work with anything that has a software interface.

ROS has evolved from ROS 1 to ROS 2, adding several improvements in performance, flexibility, and scalability of systems [31]. ROS 1 has been widely adopted due to its simplicity and community support. It relies on a centralized master for communication that can create problems in distributed or other complex systems. ROS 1 was used as a part of this thesis as upgrades that ROS 2 provides were not needed in the scope of this thesis. ROS 2 offers improved security features and safer deployments in critical applications. In future endeavors, the prototype created as a part of this thesis can be upgraded to ROS 2 for implementation of new improvements.

2. Design of the VRobo prototype

2.1. Prototype Overview

The VRobo prototype is designed to connect a physical robot and its environment together with a VR robot and a DT of the environment, providing real-time synchronization between the physical robot and its virtual replica. The prototype uses a ROS-Unity based architecture and makes sure that the physical and virtual components are consistently connected and that they exchange real-time data [24]. The Unity side uses a Meta Quest 3 headset and controllers while the ROS side uses a Ubiquity Magni robot[1].

2.1.1. Architecture Design

The prototype enables bidirectional communication between the VR scene (accessed by a remote user via an HMD) and the robot, enabling teleoperation and sensor feedback. The physical robot is equipped with *ultrasonic sensors*[2] and battery sensors that stream data to the VE in order to help the operator better understand the physical environment around the robot. The robot contains scripts for calculating and sending data as well as scripts for parsing received data and sending commands to the robot. The virtual scene ensures that the operator can move the robot around and sends movement data to the physical robot in order to synchronize the two robot entities. Communication between the real-world and the VR headset can be seen in **Figure 2.1**.

[1] https://www.ubiquityrobotics.com/products-magni/

[2] https://learn.ubiquityrobotics.com/noetic_magnisilver_sonars

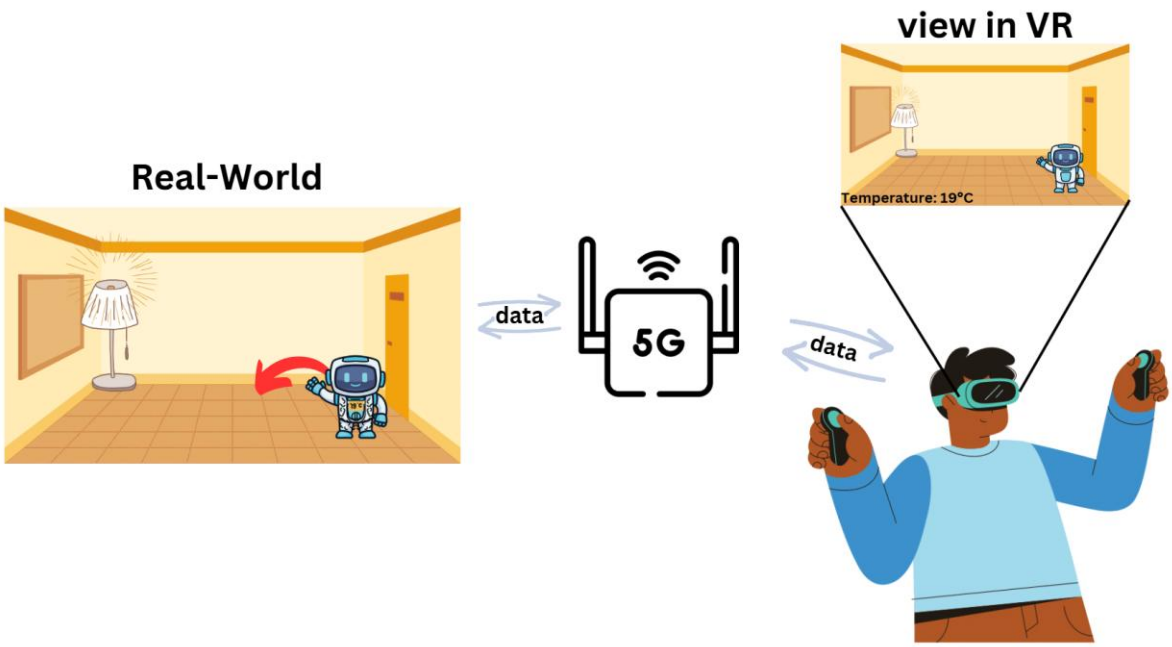

Figure 2.1: Real-world and VR application communication over a 5G network.

The left side of **Figure 2.1** shows a picture of the real-world that is a room with a lamp and a robot equipped with a temperature sensor. The right side shows a VR user that sees a visual representation of the DT of the space with the information from the sensor. The user can move the robot in real space using VR controllers, and the position of the robot will be updated in the virtual environment that it sees. All the communication is happening over a private 5G network.

2.1.2. Communication Overview

The data from Meta Quest 3 controllers' inputs is translated to **Twist** messages. These messages match the movement of the robot in the virtual scene and are simultaneously used to move the physical ROS Robot [25]. The designed architecture consists of three main components: the physical robot, the Unity scene and the communication framework that connects them, as shown in **Figure 2.2**

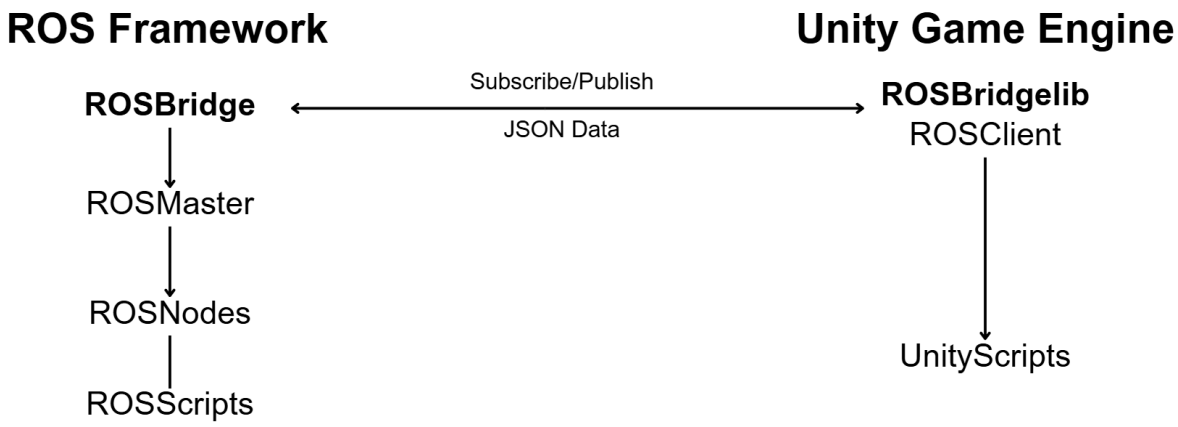

Figure 2.2: ROS - Unity Communication link overview, adapted from [24].

The connection between the Unity game engine and ROS is created by Unity connecting to the ROS Bridge server using ROS Bridge external libraries and ROS Client. It can then subscribe and publish *JSON*[3] data. ROS Bridge communicates with the ROS Master which is used to control and connect all nodes and scripts running on the ROS. It can generate data from the robot that will be sent over the ROS Bridge to Unity and can use received data from Unity to manipulate the robot. This architecture enables fast communication between the virtual and real-world robots, ensuring immersive interaction and improving spatial awareness of operators in VR.

[3] https://www.json.org/json-en.html

2.2. VR Scene Design

The VR scene in VRobo is designed to integrate the virtual replica of a physical robot inside a VR-based representation of the DT of the environment where the physical robot is located. The scene shown in **Figure 2.3** is a VR-based representation of the DT of the physical room shown in **Figure 2.4**. The VE replicates the physical environment in which the robot operates. The virtual room and all virtual objects in it have the same width, length, and height compared to the robot as they do in the real world. Using this and Unity scene colliders [26] we can create an environment that very precisely corresponds to the real world environment.

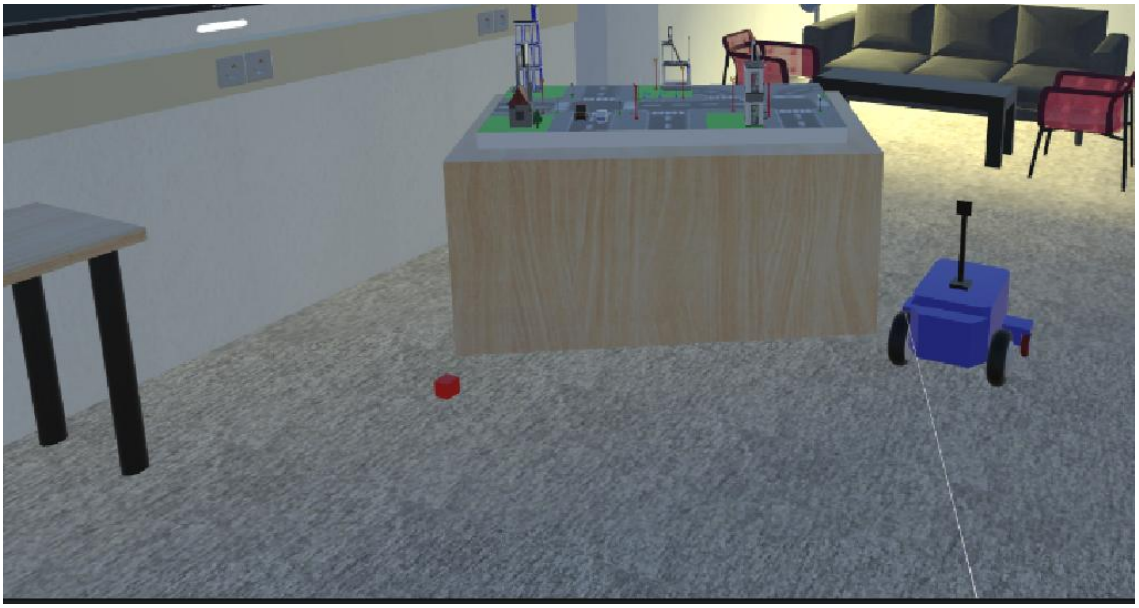

Figure 2.3: Virtual scene of the room.

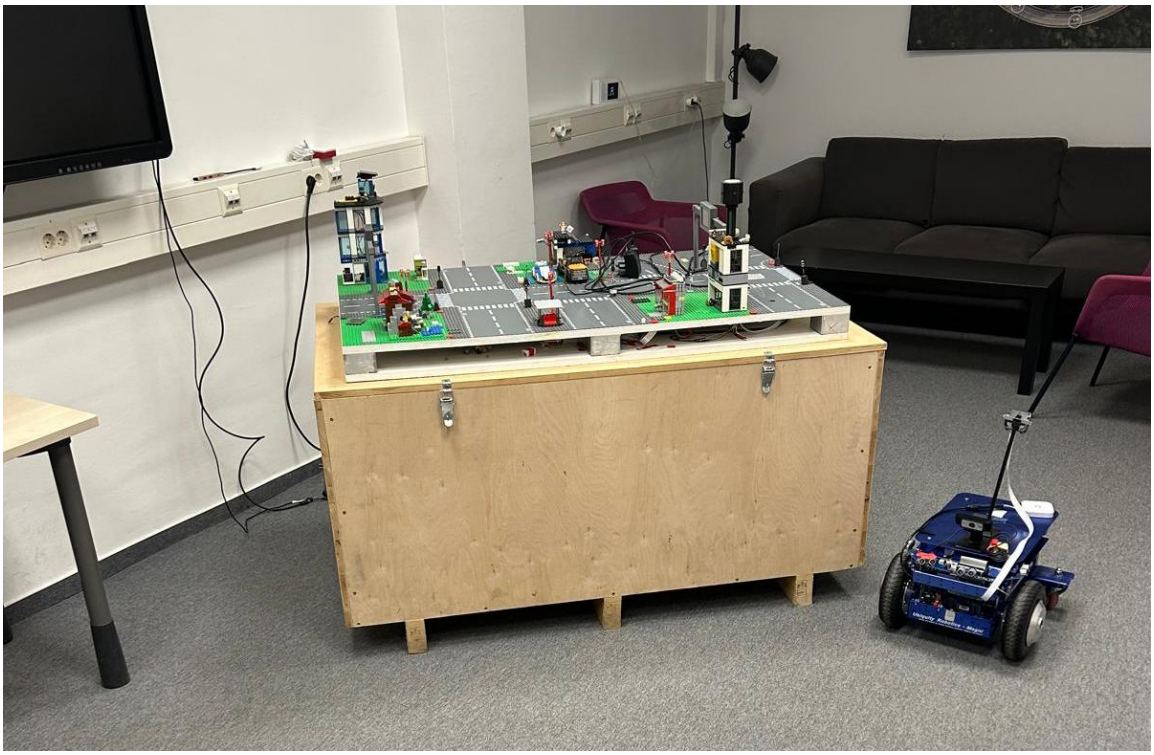

Figure 2.4: Picture of the physical room.

Interaction inside the scene is done using a Meta Quest 3 VR interface. The user can see the virtual robot from a third person view in the environment but can also use the robot's front and back camera to view a real-time camera feed from the real world from a first-person point of view.

Camera view position can be adjusted to fit the operator's HMD screen, or the camera can be mounted on the robot and move with it inside the virtual space. The scene also contains a panel where the operator can see all the real-time data from the sensors in order to understand the robot's operational status. The scene also has the option that lets the user interact with different virtual objects in the scene. These objects can later be connected to real life objects using actuators so not only robot's movements can be controlled using the VR interface. Most of the dynamic devices inside the DT can be used as long as they have a connected device that can modify them.

2.3. ROS Design and Functionalities

The Robot Operating System serves as the middleware for managing communication and control in the VRobo prototype. It is responsible for handling real-time data exchange from

both sides of the communication channel. The prototype has ROS nodes and scripts that collect data from the robot's sensors and cameras, then encrypts and sends the data over the channel to the Unity side. The ROS Bridge library is used inside Unity to handle the data before it is used in Unity scripts. ROS also has nodes and scripts that start and adjust camera settings, as well as those for motor control, enabling the operator to teleoperate the robot through transformed commands from the controllers.

ROS topics are used for publishing and subscribing, ensuring modularity and scalability of the system. Topics are named buses over which nodes exchange messages. All the nodes and scripts are managed and registered with the ROS Master in order to find and communicate with each other. This enables smooth synchronization and also adds to the modularity and scalability of the system.

3. Development of VRobo

The VRobo prototype integrates physical mobile robot teleoperation with a virtual reality-based representation of a DT, allowing real-time teleoperation through VR controllers and feedback from the robot's sensors. Prototype development includes multiple technologies: Unity for the VE and moving interface, ROS for communication and control of the robot, and additional hardware such as sensors and a 5G router for more precise and faster data transfer. This section dives into the development process, with a focus on how each component of the prototype was created, integrated and transformed to achieve precise, fast and synchronized teleoperation.

The mobile robot teleoperation prototype was designed to be integrated into the laboratory setup being developed in the scope of the project DIGIPHY. The goal of the **DIGIPHY project** is to research and design technologies that enable immersive and intuitive **eXtended Reality (XR)** inter-personal **communication** and **interaction**, as well as the remote presence and interaction of persons and objects in the visual representation of a **dynamically updated digital twin (DT)**, spatially and temporally synchronized with a physical smart space (equipped with sensors and actuators).

3.1. Used Technologies and Tools

Various technologies and tools were used to enable real-time interaction between the physical and the virtual robot. The main technologies were: Unity for the Meta Quest 3 application and VR scene creating, ROS as the operating system for moving and managing the robot, 5G router for enabling connection to a private 5G core network, cameras and sensors as additional hardware to help provide operators in VR with a higher level of immersion and more precise control with lower latency over the network [27]. The physical robot that was used in this prototype is a *Ubiquity Robotics Magni Robot*[4] that is being managed and programed by a *Raspberry Pi*[5] on which the ROS is run.

---

[4] https://learn.ubiquityrobotics.com/noetic_overview_magni_key

3.1.1. Unity

Unity is a game engine that is primarily used for 2D and 3D applications and often games [28]. It is also a widely used tool for developing VR applications due to its powerful 3D engine and highly flexible framework with many external libraries for building interactive and immersive VR applications [29]. It allows developers to model virtual environments, simulate physical forces and integrate advanced interfaces that can later be deployed to HMDs such as Meta Quest 3 that is used in this thesis. Unity also has external packages that can easily be imported and configurated when connecting to ROS using the ROS Bridge server.

Meta produced XR packages that contain Meta integration software development kits (SDKs) and Meta XR SDKs for easy connection and configuration of the Meta Quest 3 and its controllers [32]. All the packages can be imported and edited inside Unity using the C# programming language [30]. C# is a high-level, general-purpose programming language that supports multiple paradigms. It is also used in the scope of this thesis to create custom scripts which enable interaction between objects and components in the scene.

3.1.2. ROS

As previously mentioned, Robot Operating System is an open-source framework widely used in robotics applications and mostly for handling complex robot control systems [31]. ROS is used as a key part in robots communication with external hardware and controllers. It uses a modular structure that breaks tasks into nodes. Each node operates independently and handles a specific function. While certain nodes use sensor data (e.g., a node that starts and keeps the camera stream alive), there are also nodes that are used to move the robot. Important topics include the following:

- */battery_state*: used to publish the current state of the robot's battery. It is important as it lets the operator know the battery charge level, so the robot never runs out of battery and stops working during important tasks

---

[5] https://learn.ubiquityrobotics.com/noetic_overview_raspberrypi

- */cmd_vel:* used to publish velocity commands to the robot's base. The Unity application sends commands to move the robot to this topic, using *geometry_msgs/Twist*[6].
- */odom*: represents the robot's position and orientation over time. It is used to track robot's position relative to the starting point in order to reduce movement errors that build up over time.
- */motor_state*: used to publish state of the motors. It publishes speed, torque and power status. It is used for diagnostics and to ensure proper functioning of the motor.
- */pi_sonar:* has five separate topics named *Sonar_x* where x represents a number from zero to four, each topic is related to a different ultrasonic sensor on the robot. Each topic publishes data from the connected sensor that shows proximity of the obstacle to the sensor. Data from those sensors is used for object detection and collision prevention.

ROS Master is a critical part of the whole ROS; it is used to track and connect all active nodes and their topics with each other. ROS Master in the VRobo prototype is used to register nodes from the physical robot and connect them to their topics. Unity can then subscribe or publish to those topics in order to receive real-time sensor data and send robot movement commands back to the robot. This approach lets users very easily add new or replace old sensors. By adding new nodes and topics that are not dependent on other system components, external applications or controllers can then subscribe to the newly created topic and be notified with new sensor data.

3.1.3. Additional Hardware

Additional hardware refers to important system components which help stabilize and elevate the entire prototype to a new level that is faster and more precise. The front camera that is used is a *Logitech Webcam C930e*[7] while the back camera used is a *Raspberry Pi*

---

[6] https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html

[7] https://www.logitech.com/en-ae/products/webcams/c930e-business-webcam.960-000972.html

Camera Module[8]. Sonars used are a part of the Ubiquity Robots board that contains five ultrasonic raspberry sonars[9]. Data that sensors provide is crucial in creating an accurate DT of the real space in VE as they are the only part of this prototype that can provide real-time data about the dynamic changes that might occur in the real world environment of the robot, which can then be used to recreate those changes in the DT.

Meta Quest 3 is the VR headset that is used to run the VR application, and which lets the operators interact inside the VR DT of the space [33]. It offers six degrees of freedom for motion tracking, ensuring that the user's head and hands positioning and movements are always reflected in the virtual world. Meta Quest 3 controllers are used to control the robot and other objects in the virtual scene. Some of the buttons are remapped to send movement data to the real robot in order to synchronize it with the virtual robot.

3.1.4. Connection via a private 5G network

Private 5G networks offer organizations a secure, customizable, and high-performance solution when creating networks [35]. These networks use a dedicated infrastructure, ensuring high level of security, reliability, and control. With low latency, high bandwidth, and support for high device density, private 5G networks are used in advanced applications that use IoT, real-time analytics, and autonomous systems. The architecture of the private 5G network set up at Ericsson Nikola Tesla and used to enable network communication in the scope of the VRobo prototype can be seen in **Figure 3.1**.

[8] https://www.raspberrypi.com/documentation/accessories/camera.html

[9] https://learn.ubiquityrobotics.com/noetic_magnisilver_sonars

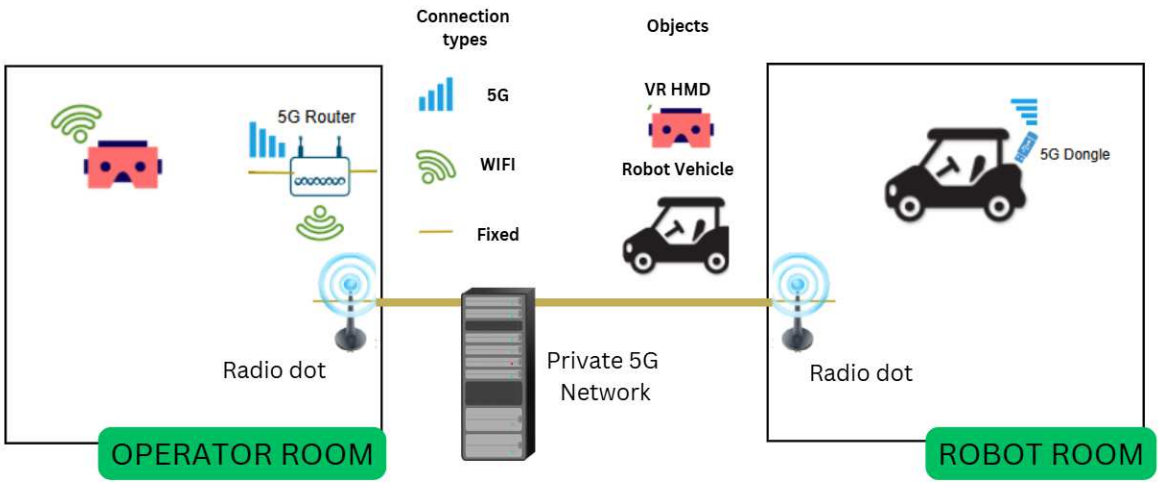

Figure 3.1: Private 5G network architecture example for this prototype.

In this example, a robot vehicle connects to 5G via a dongle, ensuring reliable and fast connection for real-time operations. It connects to the Radio dot used to extend 5G coverage and ensure good signal strength. Radio dot is connected to the private 5G network over a fixed connection. The operator room consists of VR HMD that connects to the 5G router directly over Wi-Fi[10]. Router is connected to the radio dot in that room and that is how it gets access to the private 5G network.. 5G dongle is a crucial part, it enables robots to connect and communicate over a 5G network without built-in 5G hardware. Every part of this architecture has a purpose in creating and maintaining a low latency, high bandwidth connection required to teleoperate a robot in real-time.

3.2. System Architecture Implementation

The system architecture was designed and developed to allow for fast and efficient communication between the virtual and physical world. The architecture is made modular in the sense that it lets developers add new sensors and remap controllers easily, ensuring flexibility and scalability of the system.

The VRobo prototype follows a client-server model, where the Unity application and the virtual user act as a client, while the ROS that is running on the robot acts and operates as a server. The physical robot is kept synchronized with Unity and the virtual scene using the ROS Bridge, which is the communication layer between the Unity game

[10] https://en.wikipedia.org/wiki/Wi-Fi

engine and the ROS Master. A private 5G network is used to achieve low latency and high bandwidth communication. The ROS does not support 5G connectivity by itself, so it uses a *5G NR USB DONGLE*[11] that acts as a gateway and lets the ROS connect to the 5G network [34]. It plugs into a USB port on the robot and acts as a modem, allowing the robot to connect to the network using the 5G cellular network. Meta Quest 3 connects to 5G network router using Wi-Fi. All data is transferred over this network using ROS nodes that read and publish sensor data continuously. Unity application also acts as a ROS node by subscribing to the data it needs as well as publishing data for the robot's movements. Bidirectional communication over a 5G network allows the prototype to reflect real-time changes and elevated levels of synchronization [35].

3.3. VR Scene Implementation

The VR environment is not only used as a visual representation of the robot's real-world surroundings but also acts as a teleoperation environment. The virtual scene is a precise DT of the real-world to enable operators' precise teleoperation even without seeing a robot's camera or having other ways of seeing where the robot really is inside the real world.

3.3.1. Scene Modeling

Using Unity's modeling and physics tools, the environment was created with the same dimensions and characteristics as the real world. Special attention was given to the placement of all objects and obstacles to match the position of them in the real world, and exact colliders were added to all objects in order to prevent unrealistic behavior during collisions. The scene was modelled as a copy of the static real world. Dynamic changes to the real world are not being updated inside the virtual scene which is why the robot has sensors and two cameras, front and rear. These cameras provide operators inside VR with extra information in situations where the environment might have changed dynamically during the applications use. Both camera views on the panel above the robot inside VR are portrayed in **Figure 3.2**.

[11] https://www.askey.com.tw/products-detail/ndq1300/

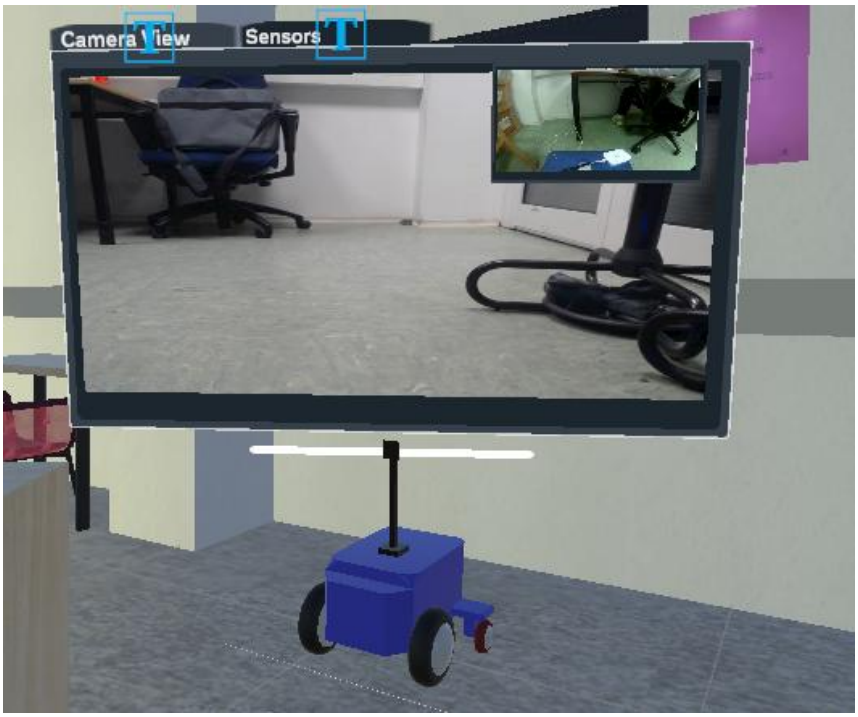

Figure 3.2: Front and back camera view on the panel above the robot.

Static changes in the real-world before starting the application need to be updated in the scene accordingly to achieve synchronization and an immersive feeling for the user. In addition to scene modeling, scene's lighting, textures and physics were optimized for VR use, lowering detail and exactness of some virtual objects that were deemed less important in the scene, as shown in **Figure 3.3**. All this together with lightmaps baking and use of low-poly models was done in order to optimize Meta Quest 3 required processing power as it is limited and could cause frame drops that lead to VR sickness in some users.

Figure 3.3: Low-poly robot and other objects in the scene.

3.3.2. VR Controller remapping

In the Unity application, the Meta Quest 3 controllers were remapped to translate user actions into robot commands. Unity's XR Input system that uses Meta XR SDK was used and changed to capture all user inputs from both controllers, such as thumbstick movements and clicks together with any button presses. These inputs are converted into ROS Twist messages, which are published to ROS Topics and then used to control the robot's movement using linear and angular velocities. Internal testing of moving the robot was created in order to choose the combination of the easiest and most intuitive controls, as well as best buttons for controlling objects in the scene. A guide for using robot controls can be found inside the virtual scene on the right side of the operator's spawn point and can be seen in **Figure 3.4**.

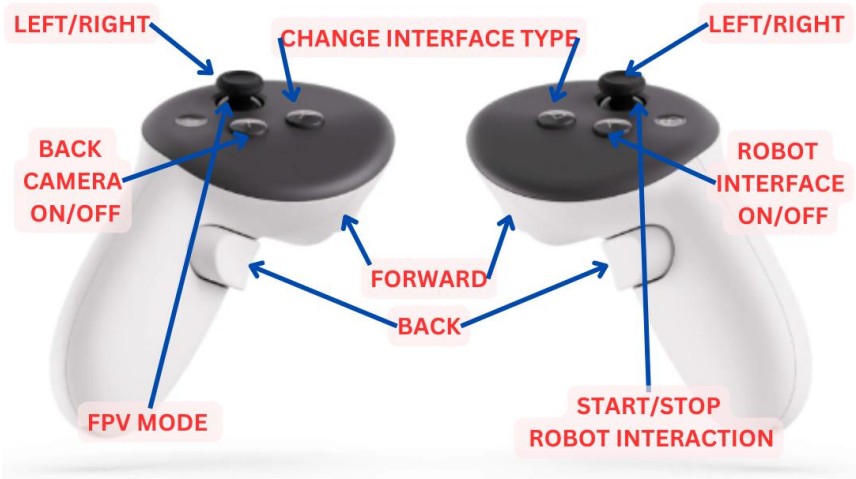

Figure 3.4: Mapping of VR controller buttons to robot control actions in the VRobo prototype.

Each of the two thumbsticks as well as index and grip buttons that are shown in **Figure 3.4** as forward and back actions, are mapped with sensitivity of the pressing or moving action. This means that pressing a button or moving the thumbstick halfway will move the robot with half of the power, both in VR and in the real world. This is important as it enables operators to easily adjust speed in scenarios when it is needed. Code that maps the sensitivity of the buttons can be seen in **Figure 3.5**.

```
_pos = OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick);
_pos2 = OVRInput.Get(OVRInput.Axis2D.SecondaryThumbstick);
if (_pos2.x > _pos.x && _pos2.x >0){
      _pos.x = _pos2.x;
}
if (_pos2.x < _pos.x && _pos2.x < 0){
      _pos.x = _pos2.x;
}
```

Figure 3.5: Code responsible for mapping sensitivity of thumbstick to motor actions.

Both `pos` variables for moving are an array of two float numbers minus one zero and one, depending on the position of the thumbstick in the two axis. These thumbsticks are then mapped to give priority to the controller with more power (e.g., the controller that moved the thumbstick further from the center will be the one that is sent to the robot).

3.3.3. First Person View Mode

To enhance operator immersion and enable operators to control the robot from first person point of view using the robot's front camera, a special panel with canvas elements was added that shows the robot's camera in front of the user in the VE. This panel can be seen in **Figure 3.6**. Using FPV mode, the operator can navigate the real world without seeing the robot's position in the DT. The operator can however use its position to gain extra information and have a more precise way to navigate the environment. This feature is particularly useful when the synchronization between real world robot and the virtual robot encounters a bigger error or when an obstacle appears in the real world that was not predicted and placed in the DT of the space.

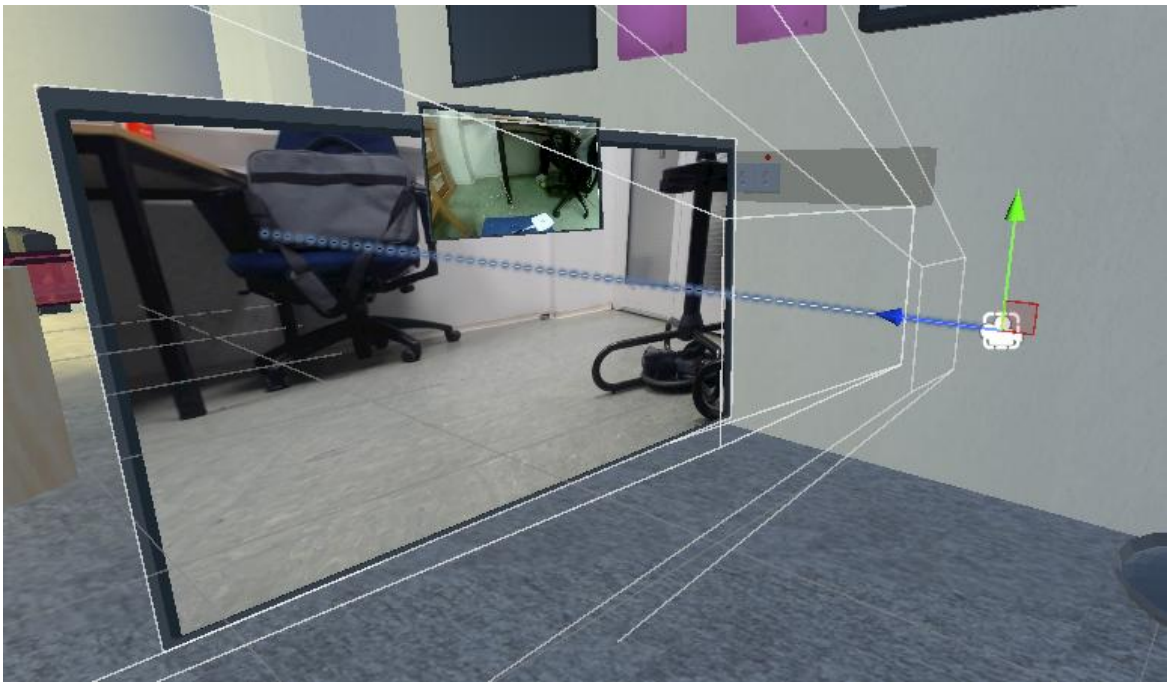

Figure 3.6: First Person view panel inside the DT with both cameras turned on.

FPV mode should also be used when doing any task in which the robot's camera feedback is of great importance and can provide the operator with more information than the DTs position.

3.4. Implementation of ROS Functionalities

ROS provides the main functionalities required for teleoperation, sensor data management, error handling mechanisms and maintaining communication with the Unity application.

3.4.1. Teleoperation

Teleoperation is the key functionality of the whole VRobo prototype, allowing users to control the real-world robot using the VR controllers. Unity constantly reads Meta Quest 3 controller inputs and translates them into twist messages that contain linear and angular velocity commands for the robot. The code responsible for reading controller inputs can be seen in **Figure 3.7**, while the code responsible for transforming those inputs into twist messages can be seen in **Figure 3.8**.

```
_indexSpeedFW = OVRInput.Get(OVRInput.Axis1D.SecondaryIndexTrigger);
_indexSpeedBW = OVRInput.Get(OVRInput.Axis1D.SecondaryHandTrigger);
_indexSpeedFW2 = OVRInput.Get(OVRInput.Axis1D.PrimaryIndexTrigger);
_indexSpeedBW2 = OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger);

if (_indexSpeedFW2 > _indexSpeedFW){
        _indexSpeedFW = _indexSpeedFW2;
        }
if (_indexSpeedBW2 > _indexSpeedBW) {
        _indexSpeedBW = _indexSpeedBW2;
        }
controllerInput.SendToRobot(-_pos.x, _indexSpeedFW);
```

Figure 3.7: Reading input from Meta Quest 3 VR controllers.

OVRInput.Axis1D.PrimaryHandTrigger is a method from Meta Quest SDK that takes a one-dimensional control such as a trigger and reports its floating-point state into a variable. This variable is represented in C# as a float number between 0 and 1. A button that is not pressed would return 0, while a fully pressed button returns 1. This lets the user choose sensitivity of the press similar to rotation shown in **Figure 3.5**. The second part of the script shows choosing a controller with higher value as a choice of movement power. This happens by comparing values on both controllers for both forward and backward and assigning the higher value to the variables that will later be sent to the robot.

```
public TwistMsg twistMessage;
void Start() {
      twistMessage = new TwistMsg();
      }
public void SendToRobot(float leftSpeed, float rightSpeed) {
      twistMessage.linear.x = rightSpeed * linearSpeedMultiplier;
      twistMessage.angular.z = leftSpeed * angularSpeedMultiplier;
      }
```

Figure 3.8: Transforming controller inputs into twist messages.

A new twist message is created at the start. Inputs from the controllers are sent to the script where they are saved and sent to the correct part of the linear and angular vector that together represent a twist message. Each of the values is multiplied by a speed multiplier, making sure the robot moves at the chosen speed. Robot's power forward is sent as a linear x variable, while the rotation of the robot is sent as an angular z variable. Both of the variables have to be between minus one and one. The robot moves its wheels according to the received twist messages and this enables the operators to teleoperate the robot. A float value of one indicates maximum power in a chosen direction. The virtual robot moves inside the DT of the space by the same amount it moves in the real-world.

3.4.2. Sensor Data

Sensor data is continuously collected and sent to the Unity application from the ROS. The two most important sensors in the VRobo prototype are the front and back camera. They are used to let the operator know what the robot's surroundings are and if there are any unpredicted objects or events taking place in the real world that the DT might not be aware of. Besides the cameras, ultrasonic sensors and battery level sensors data is sent to the Unity application letting the operator know if any obstacles or terrain are close to the robot as well as letting the operator know the battery levels of the robot. All sensor information can be seen on the panel above the robot by pressing the predetermined button for that function which is *'button B'* on the right controller and *'button X'* on the left controller. The sensor data panel can be seen in **Figure 3.9**.

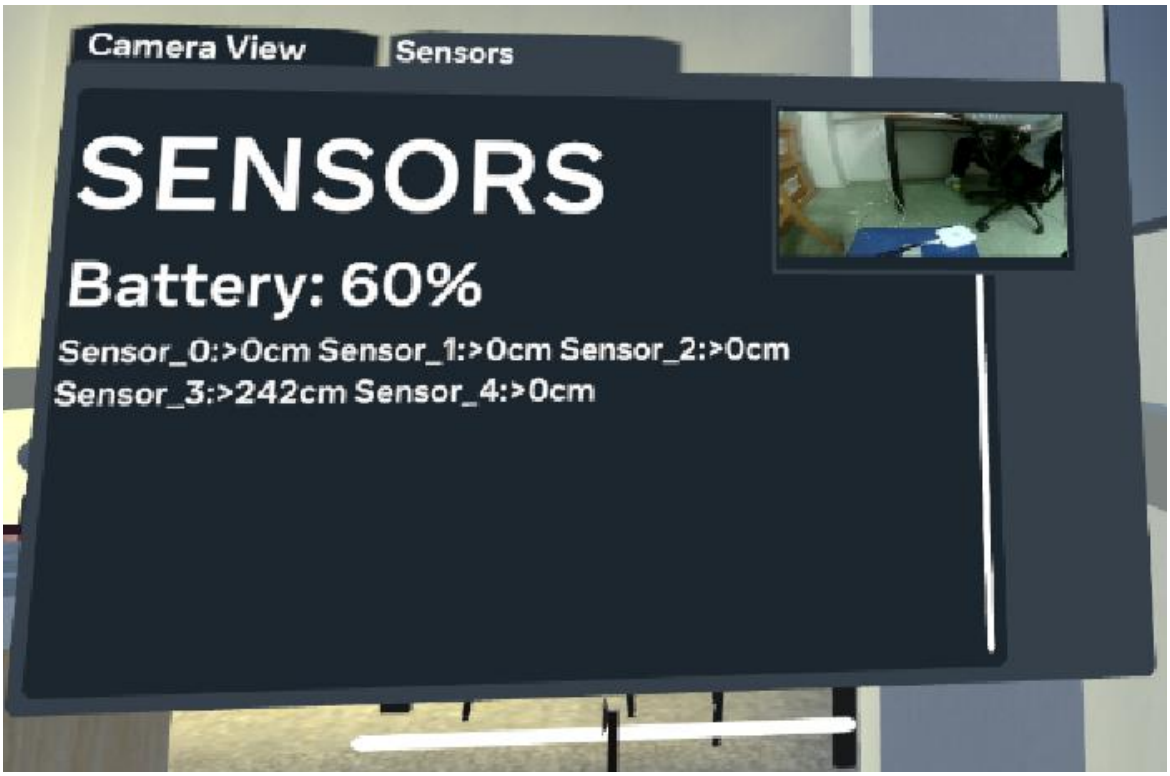

Figure 3.9: Panel above the robot showing battery percentage and ultrasonic sensor readings.

Data from the robot's sensors is published by the ROS to the topics that the Unity application subscribed to. This prototype contains battery power data as well as the ultrasonic sensors data. More sensors and data can be added to the panel as well as showing more information about the sensors that are connected. For the purpose of this thesis, it was decided to use battery data to make sure the robot does not run out of battery and proximity of object for sensors to make sure the operators do not hit obstacles. Subscribing and reading battery percentage data from the topic inside Unity can be seen in **Figure 3.10**.

```
private _topicBattery = "/battery_state";
void Start(){
      ros = ROSConnection.GetOrCreateInstance();
      ros.Subscribe<BatteryStateMsg>(_topicBattery,
      BatteryStateReceived);
      StartCoroutine(ShowTopicInfoEverySeconds(topicRefreshRate));
}
IEnumerator ShowTopicInfoEverySeconds(int i){
      while (true){
            UpdateTopicData();
            yield return new WaitForSeconds(i);
      }
}
void UpdateTopicData(){
      batteryTextField.text = _batteryData;
}
void BatteryStateReceived(BatteryStateMsg batteryStateMessage){
      _batteryData = "Battery Percentage: " +
      (batteryStateMessage.percentage * 100).ToString("F2") + "%";
}
```

Figure 3.10: Code used for subscribing to and parsing battery data in Unity.

Unity script connects to a ROS instance and uses a *Subscribe* method to get data from the ROS topic */battery_state*. After that, the script starts with a coroutine that updates the panel inside the VE. *ShowTopicInfoEverySeconds* takes an int variable and calls the *UpdateTopicData* method with a pause of given seconds. ROS */battery_State* topic is configured to send battery data every few seconds to save data. *BatteryStateReceived* method is called each time ROS publishes new data to the topic. It takes the data and parses it into percentages that are then shown on the panel to the user. Other sensors like cameras and ultrasonic sensors do not have a wait time as it is very important for them to be received in real time. Battery data is something that changes very slowly so the wait time is an optimization technique used to lower used data over the network.

Other sensors that are used are wheel encoders that measure the rotation of each wheel. These sensors can later be combined with an inertial measurement unit or radars to create very precise odometry information. ROS uses *tf package*[12] to calculate and

[12] https://wiki.ros.org/tf

determine the robot's location in the world. This odometry data is sent over the */odom*[13] topic and is then read by Unity. When combined with other sensors, it can provide very precise information and can be used in error handling mechanisms in case the robot's position in the virtual world does not correspond to the position of the robot in the real world.

3.4.3. Error Handling Mechanisms

The VRobo prototype implements some error handling mechanisms to ensure reliability of the prototype. ROS nodes are created in a way that if a sensor disconnects or a network issue occurs, the system triggers immediate recovery protocols, such as reconnecting to the sensor and sending feedback to the user. Unity has similar protocols for all external connections, making sure to retry connecting whenever the connection is lost. This is also important as camera feed resolution can be changed dynamically while the system is online. Unity application can also reconnect to the new video stream by itself.

Different types of grounds can impact wheel slippage and can create movement errors between the real and the virtual world. Non-motorized wheels are also a big issue as they do not have wheel encoders to correct errors that happen when the robot is moving [38]. Without use of those sensors, and adding *inertial measurement unit* [14] or radars to the VRobo's prototype, there currently is no opportunity for sensor fusion. Another way to manage small errors from the odometry and wheel slippage is by using *AMCL*[15]. It uses an adaptive *Monte Carlo method* [16] to find the location of the robot at any given time but requires some type of lidar to understand where in the space it is located so it can calculate and fix positional errors.

3.5. ROS-Unity Communication Implementation

Communication between the physical and virtual robot must be low latency and high bandwidth (high bandwidth due to streaming video form the robot camera). With this in

---

[13] https://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom

[14] https://docs.clearpathrobotics.com/docs/ros/config/yaml/sensors/imu/

[15] https://learn.ubiquityrobotics.com/noetic_quick_start_navigation

[16] https://en.wikipedia.org/wiki/Monte_Carlo_method

mind, ROS and Unity communication is created to ensure real-time data exchange and system responsiveness. This section discusses how communication between the systems was established and optimized.

3.5.1. Establishing a Connection

The connection between Unity and ROS is established using the ROS Bridge WebSocket protocol implemented by using *rosbridge_suite*[17]package that provides non-ROS programs. Unity uses this package to import a JSON *application programming interface (API)*[18] to ROS functionalities. The Robot starts a ROS server the moment it is turned on together with all its scripts. The Unity application acts as a client and connects to the ROS server. The Unity window for ROS connection taken from the ROS package can be seen in **Figure 3.11**.

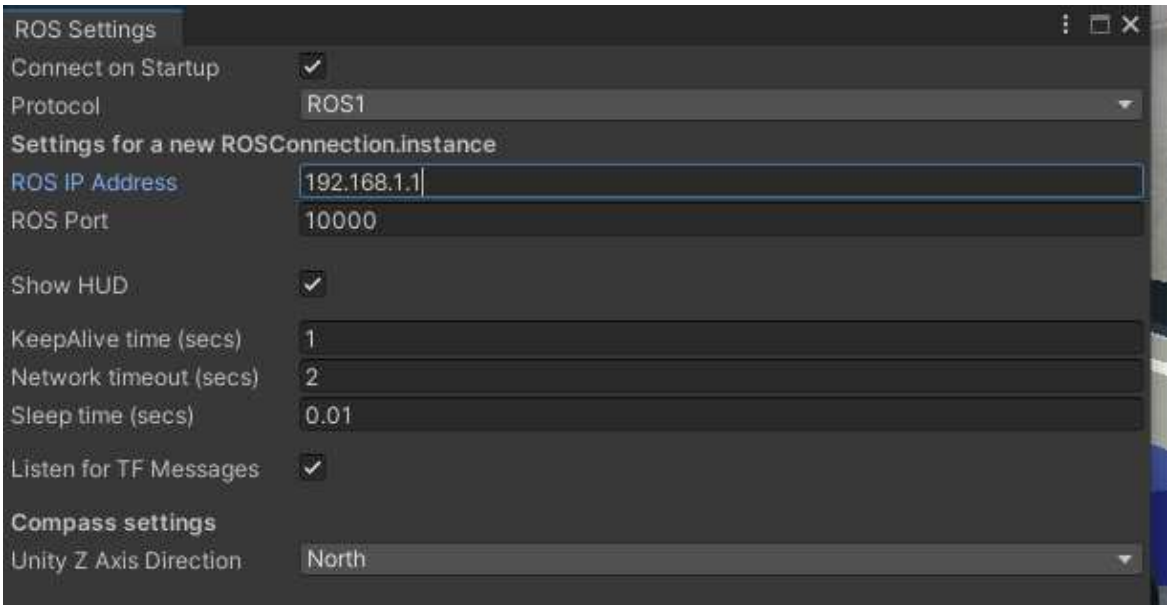

Figure 3.11: Unity window with settings used for connecting to the ROS.

The IP address highlighted in **Figure 3.11** represents the robot's IP address and uses a placeholder in this example. Port 10000 is a default ROS Master port for outside connections to the ROS. Unity connects to the ROS master gaining access to and information about all the topics and nodes. After connection, the Unity subscribe and publish scripts are run to create data channels for data that Unity plans on receiving or

[17] https://wiki.ros.org/rosbridge_suite

[18] https://en.wikipedia.org/wiki/API

sending during the teleoperation period. An example of unity code that is used to subscribe to data using a ROS topic can be seen in **Figure 3.10**, while the code that publishes that data in ROS so Unity can subscribe to it can be seen in **Figure 3.12**. Unity code that is used to publish data to a ROS topic can be seen in **Figure 3.13**.

```
class BatteryPublisher:
      def init(self):
            rospy.init_node('battery_publisher_node', anonymous=True)
            self.battery_pub = rospy.Publisher('/battery_state',
            BatteryState, queue_size=10)
            rospy.Subscriber('/battery', Float32MultiArray,
            self.battery_callback)
            self.rate = rospy.Rate(10)
            self.battery_voltage = 0.0
      def battery_callback(self, msg):
            self.battery_voltage = msg.data[0]
            …
      def publish_battery_state(self):
            while not rospy.is_shutdown():
                  full_charge = 29.4
                  low_charge = 21.0
                  …
                  battery_msg = BatteryState()
                  battery_msg.percentage = max(0.0, min(100, 100 *
                  (self.battery_voltage - low_charge)/(full_charge -
                  low_charge)))
                  self.battery_pub.publish(battery_msg)
                  …
                  rospy.loginfo("Battery Voltage: {:.2f} V, Percentage:
                  {:.2f}%".format(self.battery_voltage,
                  battery_percentage))
                  self.rate.sleep()

if name == 'main':
      try:
            magni_battery_publisher = MagniBatteryPublisher()
            magni_battery_publisher.publish_battery_state()
      except rospy.ROSInterruptException:
            pass
```

Figure 3.12: ROS code that publishes battery percentage data.

Code creates a class that is used for publishing battery data to the */battery_state* topic. It subscribes to the ROS provided battery topic where it gets all the data about the battery and then transforms that data into information important to the user (code shows the part that sets battery percentage). The script first creates a new publisher with all the default settings and then subscribes to the */battery* topic. Every time a new message is published by the battery the script takes the information and saves it into the *battery_voltage* variable. 10 Hz represents how often the script publishes data (ten times per second in this case). The script converts saved voltage to percentage, publishes it to the topic, and logs information into the system.

```
public string topicName = "/cmd_vel";
void Start(){
      ros = ROSConnection.GetOrCreateInstance();
      ros.RegisterPublisher<TwistMsg>(topicName);
}
private void Update(){
      timeElapsed += Time.deltaTime;
      if (timeElapsed > publishMessageFrequency){
            if (ros != null && !string.IsNullOrEmpty(topicName) &&
            ci.twistMessage != null){
                ros.Publish(topicName, ci.twistMessage);
            }
            timeElapsed = 0;
      }
}
```

Figure 3.13: Unity code that published to the /cmd_vel topic.

This Unity code is used to send velocity commands to the wheels. It first uses *RegisterPublisher* method to register itself as a publisher on a ROS topic using *TwistMsg*. Depending on the *publishMessageFrequency* the code publishes messages received from the *ControllerInput* script seen in **Figure 3.8**. ROS uses the received data to turn the robot's wheels and move the robot according to the virtual robot.

3.5.2. Data Transmission Protocols

Data is transmitted between ROS and Unity using the ROS *messages*[19], which structure the data into easily interpretable formats. One of the examples is Unity using twist messages to send movement data to ROS and control the robot's motors. Similarly, ROS sends sensor data such as battery percentage using *sensor_msgs BatteryState*[20]which Unity reads using its subscriber script from **Figure 3.12** to process the message and change the VR environment or in this case change the VR battery percentage panel above the robot.

Video feed from the cameras is being continuously published to an IP address using a *python*[21] script that takes a camera feed and makes it available to all devices on the network. Parts of this script can be seen in **Figure 3.14**, while the Unity script that reads the video stream from the URL can be seen in **Figure 3.15**.

```
global capture
StreamProps = ps.StreamProps
StreamProps.set_Page(StreamProps, HTML)
address = ('0.0.0.0', 9001)
capture = cv2.VideoCapture(1)
…
capture.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
capture.set(cv2.CAP_PROP_FPS, 30)
…
server = ps.Streamer(address, StreamProps)
```

Figure 3.14: Parts of python script used to start and maintain video stream from the cameras.

Python script takes the camera feed, encodes it into *Motion JPEG*[22] format and sends it to a chosen URL [36]. The python script takes the video feed from a device connected to port 1, sets the default resolution to 480p and framerate to 30 frames per second. The last part of the script starts the **PyShine**[23] server on the chosen IP and port.

---

[19] https://wiki.ros.org/msg

[20] https://docs.ros.org/en/jade/api/sensor_msgs/html/msg/BatteryState.html

[21] https://www.python.org/about/

[22] https://en.wikipedia.org/wiki/Motion_JPEG

[23] https://www.pyshine.com/

This python script also lets the user choose the quality of the front camera video stream using a Web interface. Users have the option to choose between 480p or 720p. The quality of the stream changes dynamically on the robot so users can choose to change the quality in real-time depending on what task they are doing. The script for the second camera is similar, using different default settings since it is a different camera type.

```
string defaultStreamURL = "http://ip:port/stream.mjpg";
public void StartStream(string url, bool isFW){
      StopStream(isFW);
      isStreaming = true;
      int threadID = randu.Next(65536);
      if (isFW){
            workerFW = new Thread(() =>
            ReadMJPEGStreamWorker(threadID, url, frameQueueFW));
            workerFW.Start();
      }
      else{
            workerBW = new Thread(() =>
            ReadMJPEGStreamWorker(threadID, url, frameQueueBW));
            workerBW.Start();
      }
 }
```

Figure 3.15: Unity script that reads the video stream from a URL.

Unity uses *WebRequest* [24] and a 'GET' method to connect and read the stream from the given URL. It does this using two separate threads for the two streams, continuously reading and showing the camera feed on the panels inside the VR environment. A new thread is created for each of the streams, whereby threads are used to decode the incoming stream. They receive Motion JPEG packages and decode them into *Texture2D*[25] that are then placed on *RenderTextures*[26] inside the VR.

[24] https://learn.microsoft.com/en-us/dotnet/api/system.net.webrequest?view=net-9.0

[25] https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Texture2D.html

[26] https://docs.unity3d.com/6000.0/Documentation/ScriptReference/RenderTexture.html

3.5.3. Latency Optimization

Teleoperation requires low-latency and real-time responsiveness. Because of this, data packets were minimized in size and the back camera is using 480p resolution. The front can use 480p or 720p depending on operator preference, but never uses 1080p that it could use. The reason for this is that using such high quality would require more time to process and would raise latency considerably. Motion JPEG was used in encoding the video stream. It was the fastest encoding we managed to create, requiring higher bandwidth but lower encoding times with the video packets compared to other options considered such as H.264[27] or H.265[28] [36]. In video compression testing conducted in [41], Motion JPEG outperforms x265[29] (implementation of the H.265 standard) being 7 times faster and outperforms H.264 being 22 times faster in terms of encoded frames per second as can be seen in **Table 3.1**. The speed of encoding comes at the cost of space and network bandwidth, for the purpose of the thesis we had a fast network and decided this was the best approach.

[27] https://en.wikipedia.org/wiki/Advanced_Video_Coding

[28] https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding

[29] https://en.wikipedia.org/wiki/X265

Bitrate (kb/s)	PSNR (dB)	e fps	Average e fps
MJPEG2000			4,385751163
51191,2	41	4,42071826	
56440	42	4,42931659	
61804	43	4,32082794	
66876,8	44	4,37937063	
71894,4	45	4,37937063	
HEVC HM 8.0			0,045371194
12304,1208	41,7618	0,0506142	
16492,4944	43,053325	0,04696132	
22238,9016	44,5064	0,03929415	
H.264			0,20509065
15008,9096	41,4060375	0,21948211	
19866,8176	42,77075	0,20101795	
26242,1592	44,2503125	0,19552559	
x265			0,645226762
34997,33	45,439	0,7	
27725,38	43,81	0,60245319	
20613,83	42,222	0,6	
15731,47	40,597	0,68497842	

Table 3.1: Performance comparison of video compression standards: MJPEG2000, HEVC HM, H.264, and x265 taken from [41].

The system also implements buffers to handle small network delays without them affecting the user experience. Using buffers on control inputs and sensor data ensures that the robot does not move in unpredicted ways. Control inputs from controllers are processed in their own thread and immediately sent to the robot while the sensor data is updated in Unity in near real-time. Use of a 5G network is crucial to enable all of this as it allows us to connect the devices using extremely low latency and high bandwidth.

3.6. Prototype Limitations

Despite the successful integration of various technologies, the current prototype has some limitations. Precision tasks remain a challenge due to the latency and robot's motor precision. Movement errors are small but can cause problems when they build up over time. The system's ability to synchronize after errors happen can improve using additional sensors and techniques. Additionally, the learning curve for users that are not familiar or do not use VR often can impact both task completion speed and success rate. Network slicing is not used in this system but can also be significant, enabling developers to reduce latency by providing dedicated and customized network resources [37]. This can be very useful in transmission of large volumes of data, such as high-quality video streams, even when combined with a lot of sensor information, all without degradation in performance.

The use of 5G not only reduces latency but increases security and reliability that can impact latency in the long run, enabling teleoperation without interruptions.

These limitations provide important insights into the system and help create better future iterations. A future version of the VRobo prototype should be focused on improving accuracy, reducing latency, and evolving the VR interface.

4. User Study

In the scope of this thesis, a user study was conducted to evaluate latency, movement smoothness, and QoE for the developed prototype. By targeting different users and tasks, the study aims to evaluate intuitivity of the VR interface and how users adapt to the teleoperation system over time. Users tried out different control methods during the study. These control methods are: controlling the robot using only the digital robot's position, controlling the robot using both the virtual position and the camera view, controlling the robot using only the camera view, and controlling the robot by pointing to the place on the ground where it should move. Camera view is the term that relates to the camera stream coming from the robot and is shown inside the VR. By randomizing control methods, it also examines how quickly users learn to perform tasks in VR. To address these goals, the study addresses the following research questions (RQs):

RQ1: *Which method of robot control is the easiest and most intuitive by the users?*
RQ2: *Which method of robot control and camera view enables users to complete each task in the shortest amount of time?*

RQ3: *How does the presence of a camera view impact users' precision and speed when controlling the robot?*

RQ4: *Which method of robot control and camera view provides the highest level of accuracy for task completion (with tasks including driving towards a goal, driving around objects and parking)?*

RQ5*: How do latency and stream quality (480p and low latency vs. 720p and high latency) affect user performance, task completion speed and preferences during teleoperation?*

4.1. Methodology

4.1.1. Laboratory Setup

The Study was conducted in a dedicated laboratory room and its corresponding DT (virtual replica). The virtual room was equipped with three cubes at different parts of the room that the user had to touch with the virtual robot. Each of the cubes represented a task, to touch the first cube, users had to navigate around the chair. For the second cube, the user had to make a 180° turn and navigate back around the chair. For the last and third task, the user had to drive backwards and 'park' the robot in the position it started from. All three tasks were performed across all 5 test scenarios (explained in the following section). Users could choose how they want to do the task and where they want to put most of their focus into. This included driving forward with rotating towards the point or driving both forward and backward depending on the situation. It also included focusing on only one, both, or neither camera. The only thing that remained the same was touching the cubes and checking if they turned green, which was the sign that the task has been successfully completed. Regardless of the user's VR experience or speed in completing tasks, all users went through the same tasks and scenarios. The DT of the space can be seen in **Figure 4.1**, while the real life space used for the study can be seen in **Figure 4.2**.

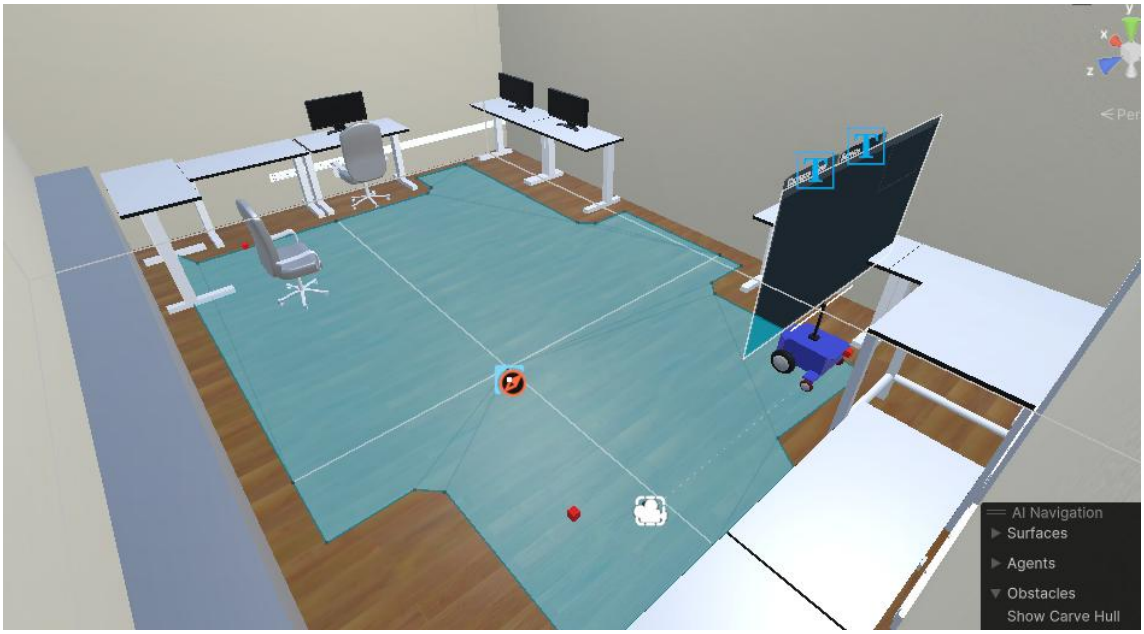

Figure 4.1: Virtual representation of the DT of the space used for the study.

Figure 4.2: Real life space used for the study.

Assessing QoE in teleoperation has become increasingly important, particularly with immersive technologies such as VR .Previous work [43] explored interactive QoE assessment in the context of teleoperating robots, highlighting the complexities involved in user interactions with remote environments. The study uses a within-subjects design in which all users are playing all scenarios but in different order [42].

4.1.2. Test scenarios and procedure

Participants went through the test that took approximately 30 minutes to complete. Users that have more experience with VR took less time to complete the study. Participants were first asked to sign a consent form and were then directed to the VR HMD where they started the study process. Participants then received instructions on how to use the given hardware and were explained what the tasks are and how to complete them. There were five test scenarios in total, four of them being done at random order. These scenarios can be seen in **Figure 4.3** and include:

- robot driving with camera on →**Robot Camera** scenario,
- robot driving with camera off →**Robot No Camera** scenario,

- robot driving in FPV mode, high latency and high video quality → **FPV High** scenario,
- robot driving by pointing → **Point And Click** scenario.
- robot driving in FPV mode, low latency and low video quality → **FPV Low** scenario

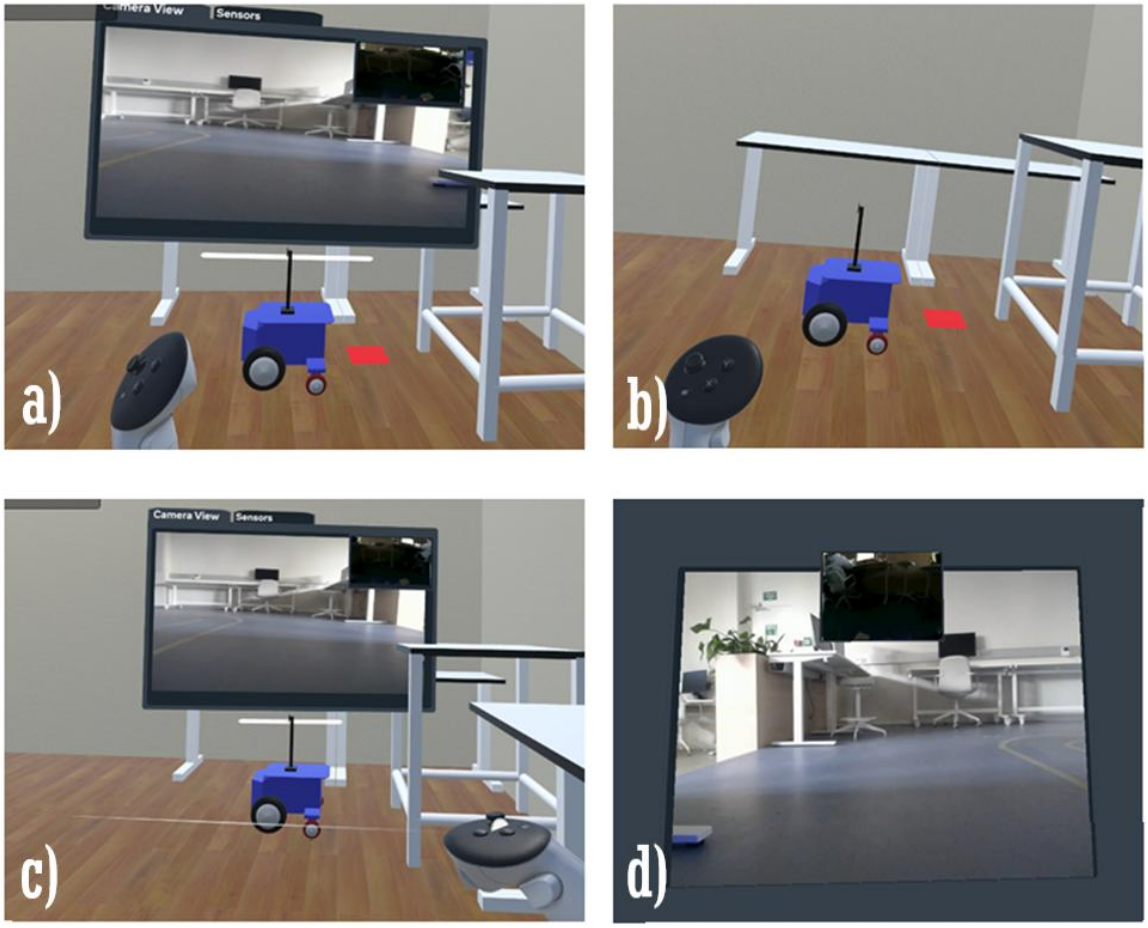

Figure 4.3: Screenshots of four different control methods inside VR: Robot Camera (a), Robot No Camera (b), Point And Click (c), and FPV (d).

Each participant first tested four out of five above-mentioned scenarios in a randomized order, excluding one of the FPV scenarios. The participant then tested the remaining FPV scenario. The reason for this was that we did not want the users to play the FPV scenario twice and other scenarios once before the first section of the questionnaire. Which of the FPV scenarios was done in the first group and which was done fifth, was also randomized across participants. Robot driving with camera on is a scenario in which users drove the robot using the VR controllers and had the camera above the robot for extra

information about the environment. The resolution of the camera in all scenarios besides the FPV High scenario was 480p. Robot driving with camera off is the same as the previous scenario but without the camera. This is the only scenario where the users did not use the camera information and only used the virtual robot's position when moving around the room.

Robot driving by pointing is the only scenario in which the users did not have full control of the robot. They pointed to a spot in the VR environment where they wanted the robot to go. The robot would calculate the rotation and distance needed to reach the chosen point and would drive there automatically. In case there was an obstacle in the way of going from the current position to the newly set position, the robot would create the shortest path using waypoints and would make straight lines with stops to rotate. FPV scenarios had a panel showing both camera views on it. The panel was always in front of the user and took a big part of their screen. Users were still able to see parts of the DT around the panel, and could turn it off to fully see the VE. Finally, the difference between the "High" and "Low" FPV scenarios was that the higher resolution scenario used a video stream in 720p quality while the other scenario used a 480p quality. This resulted in three times more pixels being generated for the higher quality stream, thus resulting in higher latency. The latency when streaming 720p was approximately two times greater than the latency observed in the other scenarios. Estimated latency in the faster scenario (480p quality) was around 110ms while it was around 220ms for the slower scenario (720p quality) round. Latency was approximated using time stamps in Unity and it calculated round trip time latency. One timestamp when data to move the wheels was sent, and one when the data that the wheels were moved was received. Calculated latency varied depending on robot moving or standing still and camera stream quality.

4.1.3. Collection of subjective and objective metrics

To facilitate this study, a questionnaire was developed, consisting of 39 questions across four sections. Participants answered the questionnaire on a laptop by removing the VR HMD after doing each of the scenarios. The questionnaire is given in Appendix A. It consists of various question types:

- multiple-choice (single select option),
- 5 point absolute category rating (ACR) scale,

- ranking scenarios, and
- open-ended question (optional, to further elaborate on their response).

Each participant completed the first section of the questionnaire before completing any of the scenarios. This section collected demographics data and data regarding previous experience in using VR and radio-controlled vehicle technologies. After completing each of the scenarios, the part of the questionnaire regarding that scenario was filled. After completing all the scenarios, questions about ranking and comparing the scenarios were answered. Additionally, objective metrics that were measured were the number of errors per scenario and time to complete each of the tasks during the scenario. Any kind of impact between the real robot and the environment is considered an error and was collected manually by watching for collisions. Task completion time was collected by measuring the time it took the participants to complete each of the tasks. Each of the participants was asked to subjectively rate their performance per scenario in terms of speed and task completion rate. This was done in order to compare their subjectively perceived performance with the actual objective metrics of their performance.

4.1.4. Participants

The research study was done with 13 voluntary participants, with 8 identifying as *male* and 5 identifying as *female*. The age range spanned from 22 to 55, with a mean age of 25. Out of 13 participants, 10 of them had previously operated a radio-controlled robot vehicle, while none of them reported doing so on a weekly basis. The frequency of using a VR device can be seen in **Table 4.1**. Four of the participants use VR devices on a weekly basis, only one participant used a VR device for the first time during the study.

VR Use Experience	User Count
More times a month	4
Occasionally, but less than once a month	6
1 to 3 times in life	2
Never used a VR device	1

Table 4.1: Experience using VR devices prior to the study.

Participants were also asked to describe their attitude towards virtual reality technology. Almost all the users gave a maximum positive rating, with only one user describing their attitude as negative. Average attitude was 4.46 on a range from 1 to 5 (1 meaning mostly negative and 5 meaning mostly positive). Participant attitude towards VR technology can be seen in **Table 4.2**.

Attitude Towards VR	User Count
Mostly positive	9
Positive	2
Average	1
Negative	1
Mostly negative	0

Table 4.2: Participant attitude towards VR technology

4.2. Results and Discussion

This section presents results from both subjective user-reported and objective performance-based metrics collected during the study. The discussion covers connections and discrepancies between subjective user feelings about how they did in the scenarios with measured metrics of how they actually did in them.

4.2.1. Subjective metrics data

Participants' perceptions of each scenario were assessed during the study. The study focused on users giving ratings about: video quality, ease of navigation, ability to complete the task, overall QoE, and level of frustration. Users' ratings of video quality can be seen in **Figure 4.4**., portraying average scores (Mean Opinion Scores, MOS) with 95% confidence intervals (CI) shown.

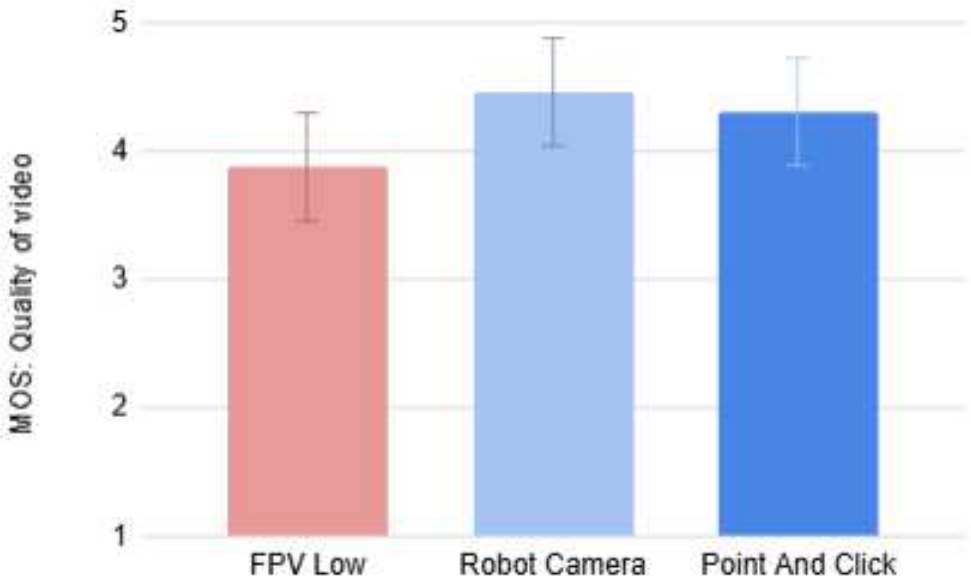

Figure 4.4: Average participants ratings on quality of video (with 95% CI).

Portrayed results only contain one FPV scenario and it is FPV Low scenario. Video settings in this scenario were the same as for Robot Camera and Point and Click, thus enabling us to compare perceived video quality only in terms of different view perspectives. All of the scenarios using a camera had a similar average rating with overlapping CIs. The ratings were given on a 5-tp ACR scale (1 being poor and 5 being excellent). The scenario with the highest average rating was 'Robot Camera' with 4.46 while the lowest average rating was given to the FPV Low scenario with 4.08.

Questions portraying QoE in the second section do not consider FPV High scenario as it used a different latency. Its purpose was to compare latency and camera resolution during the FPV scenarios. Participants' ratings on ability to complete the task that can be seen in **Figure 4.5** gave data that aligned closely with ratings on ease of navigation that can be seen in **Figure 4.6** and **Figure 4.7**.

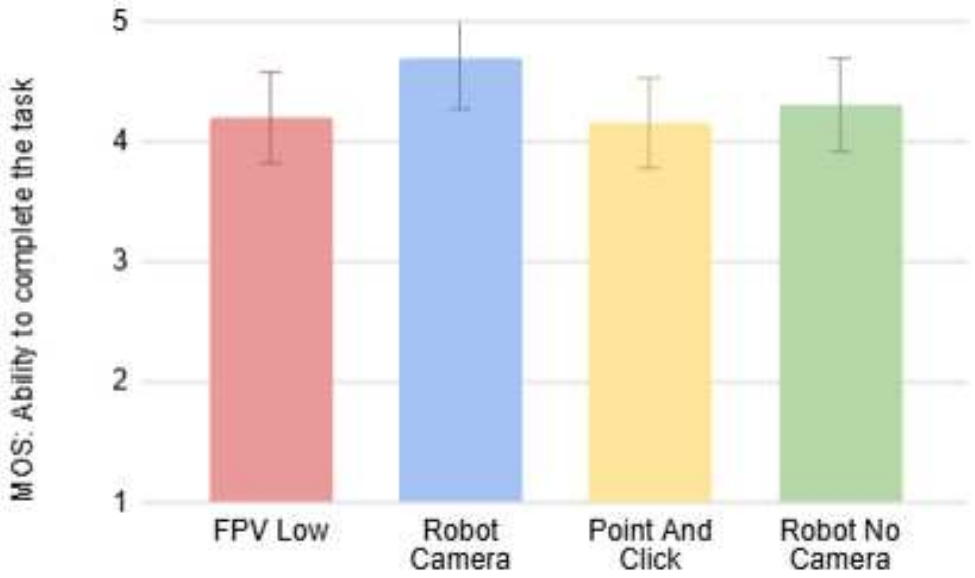

Figure 4.5: Participants' average ratings on ability to complete the task (with 95% CI).

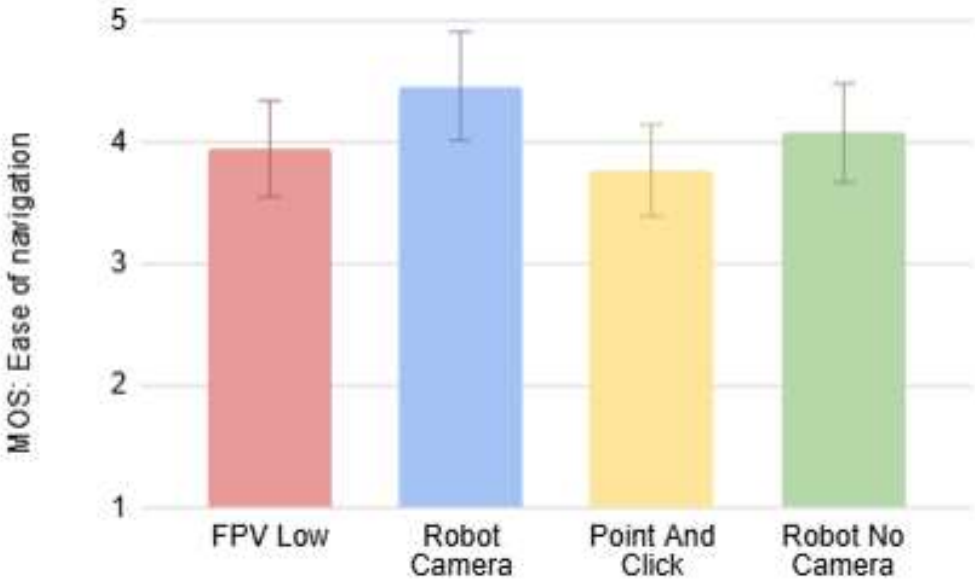

Figure 4.6: Participants' ratings on ease of navigation (with 95% CI).

.

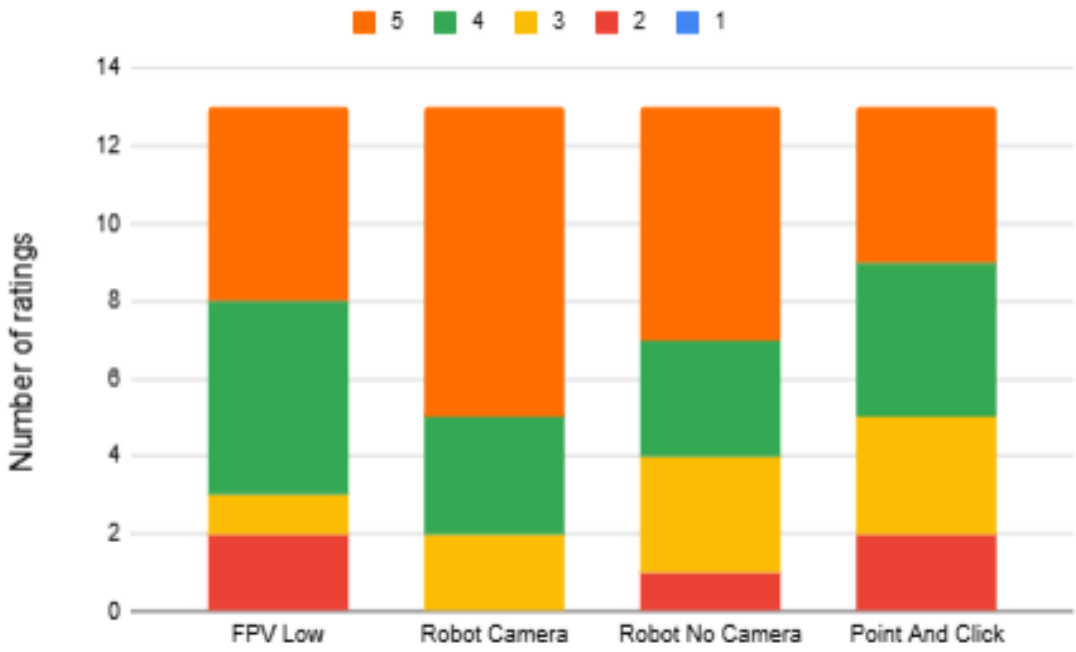

Figure 4.7: Distribution of participant ratings per scenario when asked about ease of navigation.

With respect to ease of navigation, participants rated the 'Robot Camera' scenario as the best one while the same scenario just without the camera was rated as second best in both categories. 'Robot Camera' did not receive any rating lower than a 3. 'Point And Click' scenario ratings differ from FPV Low scenario ratings by just one rating overall (rating 3 instead of rating 4). Most of the users felt that having the camera helped them in completing tasks faster and with higher precision. Many of the users also felt that the FPV Low scenario had a camera video feed screen that got in the way of their field of vision and interrupted them doing tasks. The 'Point And Click' scenario received poor ratings as it was the one in which many of the users felt they had the lowest amount of control as the robot would calculate its own way to move and do so. There was no way for the participants to stop the robot once it started going. Only option users had was to change the target position to a new one. Three participants felt that 'Point And Click' was the best scenario as they didn't have to think much and could just click and relax.

Overall QoE of participants can be seen in **Figure 4.8**.

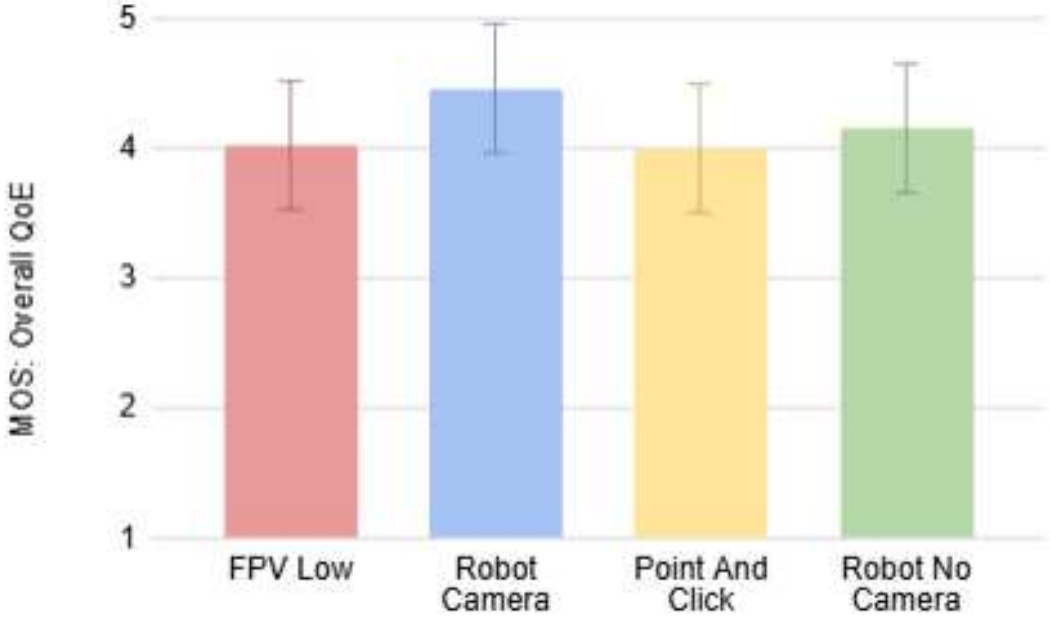

Figure 4.8: Participants' average ratings on overall QoE (with 95% CI).

Overall QoE follows the same pattern and exhibits similar trends to those seen in **Figure 4.5** and **Figure 4.6**.

The last subjective question in this section was about the level of frustration participants felt in the scenario. Ratings were given on a scale from 1 to 5 (1 being very low and 5 being very high) and they can be seen in **Figure 4.9**.

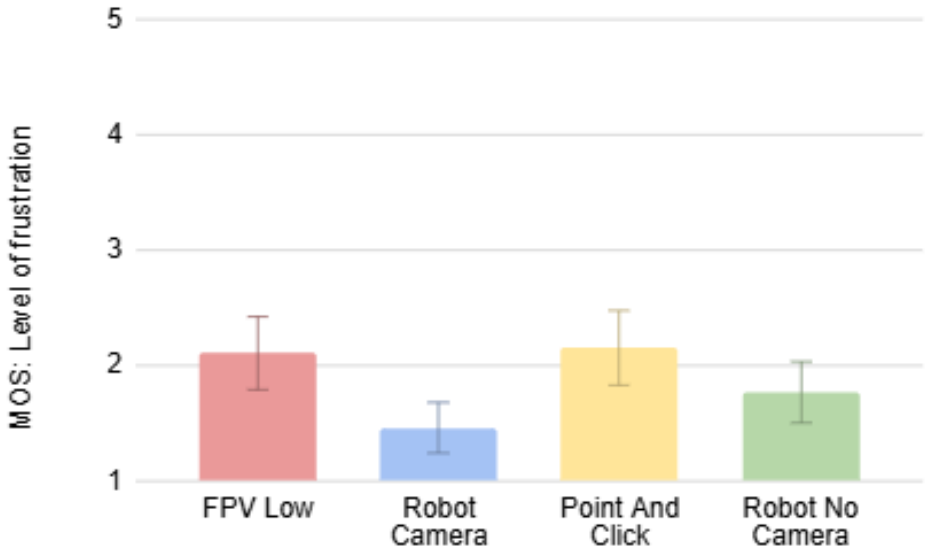

Figure 4.9: Participants' average ratings on level of frustration (with 95% CI).

Data in this graph is consistent with the previous graphs giving lowest frustration scores to the 'Robot Camera' and 'Robot No Camera' scenarios. Main reasons that raised participants' level of frustration were (received from their feedback):

- 'Robot No Camera' not having any camera information that would help them know where the robot is in the real world
- 'Point And Click' not giving them enough control over the robot's movements
- 'FPV Low' screen being in the way of seeing where the robot is in the virtual space

Furthermore, the distribution of ratings for each of the scenarios regarding frustration can be seen in **Figure 4.10**.

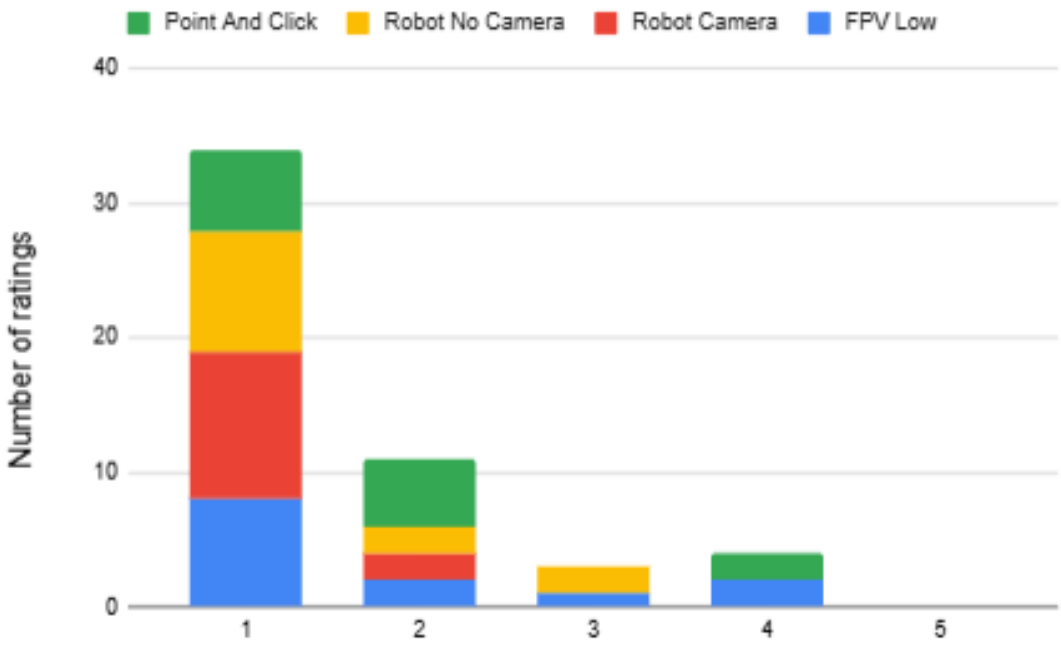

Figure 4.10: Number of ratings given to each scenario based on frustration.

All scenarios got the biggest number of ratings in the form of 1 meaning very low frustration. We can see that 'Robot Camera' did not receive a rating higher than 2. This matches the previous data as that was the easiest scenario in terms of navigation and ease of use for the participants. However, 'Point And Click' and FPV Low scenario were given a rating of 4 by two participants, meaning that they highly frustrated some of the participants.

After completing all the scenarios, participants were asked to rank each of the four scenarios (not distinguishing between FPV high and low) from best to worst in terms of ease of use (1 being best, 4 being worst). The graph data mostly follows the previous

answers. Main difference is that after completing all the scenarios, average ratings indicate that 'Robot Camera' is by far the favorite scenario with 0 participants choosing it as worst in terms of ease of use. The interesting thing is that 3 out of the 13 participants chose Point And Click' as their best scenario, while everyone else ranked it as worst or second to worst scenario. Results can be seen in **Figure 4.11**.

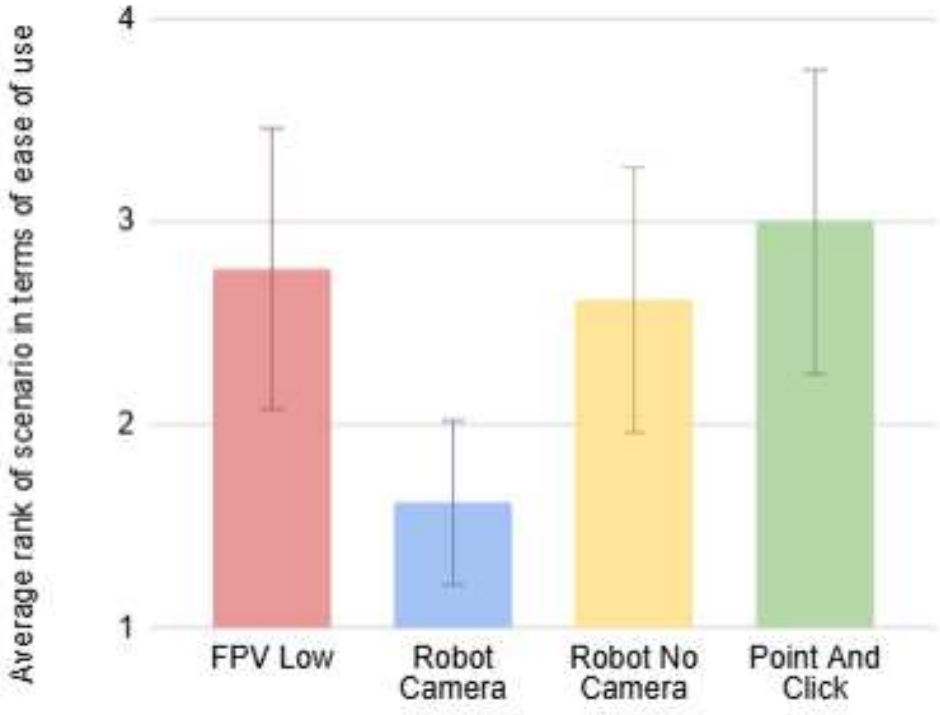

Figure 4.11: Average rank of scenarios in terms of ease of use (with 95% CI).

Participants were further asked to choose whether they preferred a lower latency lower resolution camera or higher latency higher resolution camera (tested in the FPV scenarios). Results can be seen in **Figure 4.12**.

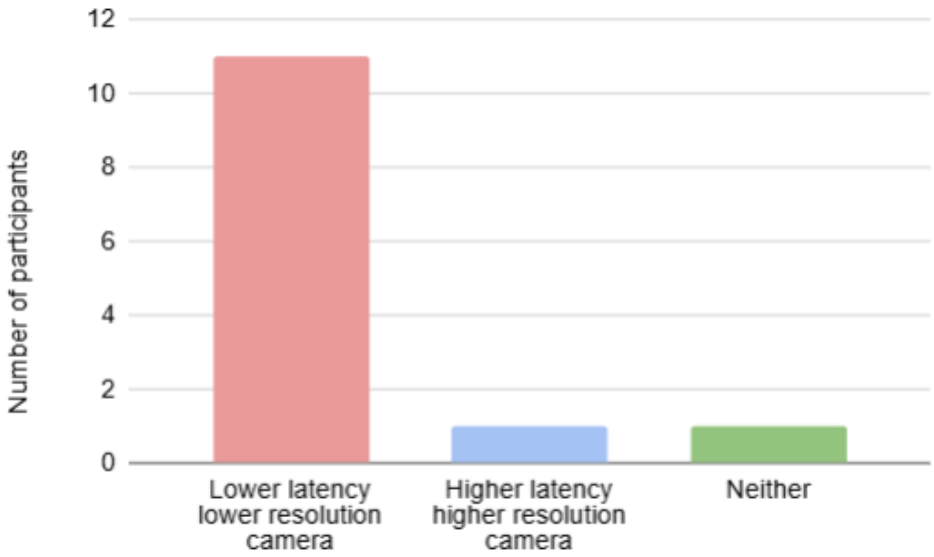

Figure 4.12: Participants' latency and video quality preference.

The results show that most of the participants prefer low latency at the cost of lower video quality. The participants' reasons for this choice were that the camera view wasn't as important as the responsiveness of the robot (collected from the open questions from the users). The only person that chose higher resolution as more important did not give any reason for his/her choice. Most of the participants felt much more confident when the delay between the controls and the movements was lower. This made them make fewer mistakes and be faster overall. Camera view was only used to check whether the robot's position is synchronized well with the virtual one and in order not to hit objects in the real world.

Furthermore, participants were asked how much they felt the camera helped them when completing tasks. Results can be seen in **Figure 4.13**.

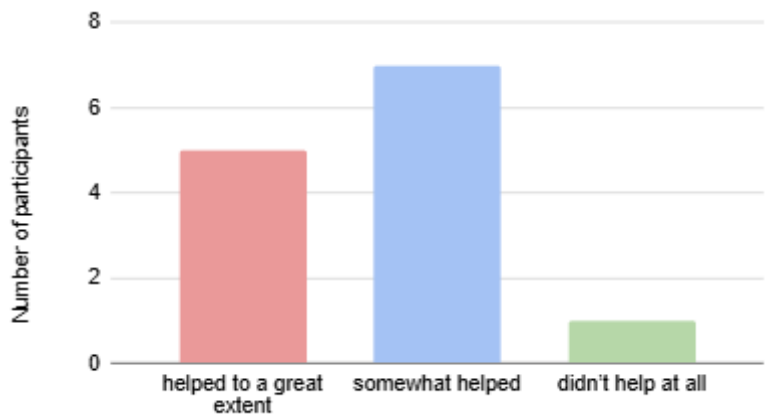

Figure 4.13: Participants feelings about camera help when doing tasks.

Most of the participants felt the camera somewhat helped them while some think it helped to a great extent. Only one participant said it didn't help at all. The main reason given for this was complete trust in the virtual robot's position and feeling the camera was only in the way, taking a part of the screen. Others gave reasons that the camera was not in the way and it was nice to have it in case they wanted to check where the robot is in the real world.

4.2.2. Objective metrics data

Objective metrics focused on the number of collisions and the task completion times. The main reason that led to collisions were desynchronization due to wheel slippage and participants not paying attention to virtual or real-life objects when they were close to them. The number of collisions per scenario can be seen in **Figure 4.14.**

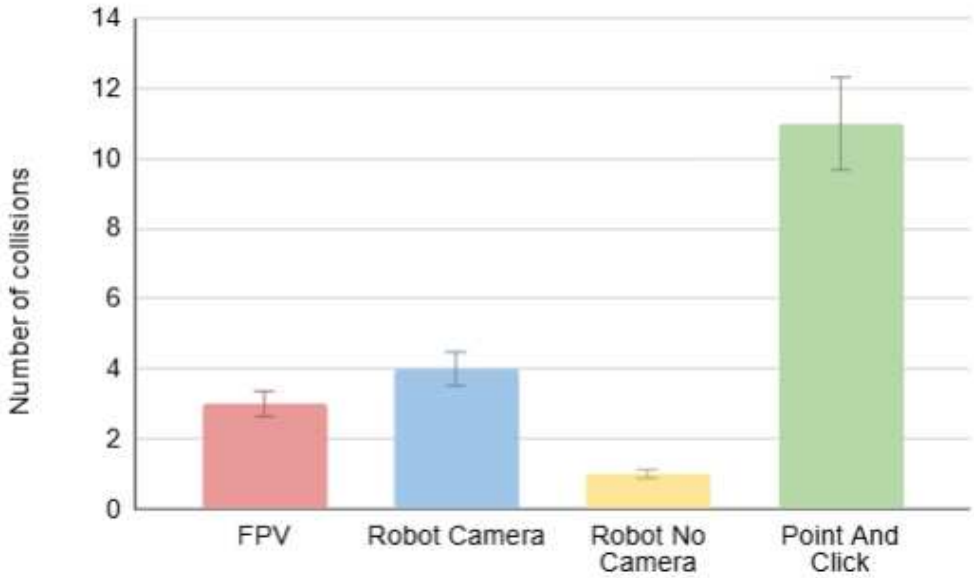

Figure 4.14: Number of collisions per scenario (with 95% CI).

There were not a lot of collisions that happened across the study. Participants averaged 0.29 collisions per scenario. Five participants didn't make a single collision across all scenarios while the highest number of collisions per participant was 5 and it was done by only one participant. 'Point And Click' had by far the highest number of collisions. This happened because the robot was rotating a lot and users did not have full control after telling the robot where to move. In case the collision happened, the study administrator would move the robot back to the correct position. After correcting the position, the user

would continue the study. There was also a bug in the system that led to desynchronization if participants pressed wrong buttons while the robot was calculating its moving route. Fixing this bug could result in a lower number of collisions in future studies. Interestingly, the lowest number of collisions happened in the 'Robot No Camera' scenario which was the only scenario without the camera. Participants only focused on finishing tasks and didn't take much care about anything else. There also wasn't a camera feed panel that would take a part of their VR screen, resulting in their field of view being very clear.

Average task completion times can be seen in **Figure 4.15**.

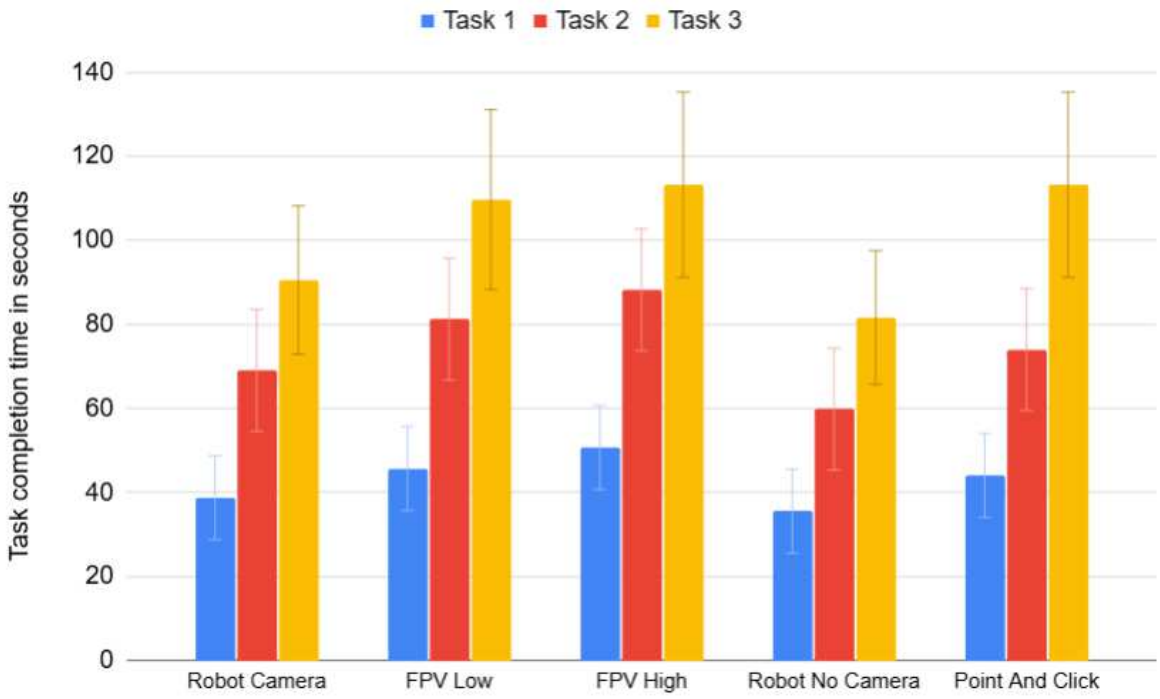

Figure 4.15: Average task completion time in seconds for each of the tasks across all the scenarios (with 95% CI).

Average task completion times show us that 'Robot No Camera' was the fastest scenario across all three tasks. Its average time to finish all the tasks was more than 20 seconds faster than the average finish time across all 5 scenarios. 'Robot Camera' was second in terms of speed. Users lost most of their time navigating around the chair and were otherwise very fast compared to other scenarios. Furthermore, 'FPV Low' shows task completion times for FPV scenario with low latency low resolution camera are generally a bit faster than the same scenario with high latency high resolution called 'FPV High. The reason for the times not having a bigger gap as can be expected from the subjective data is that 'FPV High' was often the last scenario. The participants were already very familiar

with the environment and the tasks which made it easier for them. Lastly, data shows that parking the robot in 'Point And Click' was the slowest out of all the scenarios. Time between the second and the third task shows how difficult and slow this task was for the users in the scenario. However, first and second task in these scenarios were done in times almost exact to the 'Robot Camera' scenario. Task completion time in seconds for just task two can be seen in **Figure 4.16**.

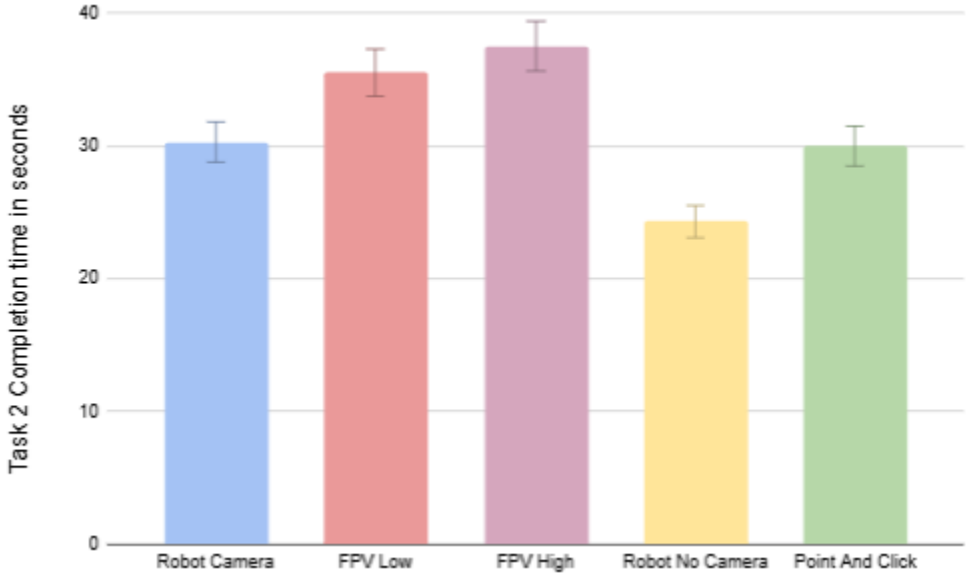

Figure 4.16: Average participant time to finish the second task in seconds (with 95% CI).

The second task was the hardest one in terms of complexity. Participants had to navigate around the chair, either going backwards or turning around. After that, they had to drive to the other side of the room in order to collect the virtual cube. The graph in **Figure 4.16** shows that using the camera in this scenario slowed the users by over 20%. This lost time is very similar to how much 'Point And Click' was slower due to it rotating towards the point and then driving forward. FPV scenarios were both much slower than the other 3. The reason for this was that most of the users kept turning the FPV view on and off in order to see where the virtual robot is inside the VE. Doing this made them much slower in general when completing any task, but specifically during this task that needed an extra dose of maneuvering.

Comparing objective and subjective data we can conclude that participants feel safer and better when having a camera feed in the scene. However, having the camera makes them objectively slower as they have additional information to which they pay a lot of their attention that isn't as important in completing the tasks. More experienced users

have a much smaller gap between times in their scenarios. This is probably due to the fact they are used to the controls and VE. Another important thing is they are not wasting much time thinking about the camera and the environment but rather spend most of their focus on completing the task.

The study did not give a clear result in terms of which scenario is the most intuitive or easiest for the users but gave us important information about each of the scenarios. 'Robot Camera' scenario seems to lead the way as the best out of the five. It has the fastest to learn and easiest to use controls on average with a camera stream that gives a feeling of safety to the users. 'Robot No Camera' is the fastest scenario that was great for more experienced participants and works great when synchronization errors do not exist or are minimal. FPV scenarios were good in situations where users had to use the camera for their tasks. There wasn't much need for this in the study so most of the users felt it only got in the way of seeing the VE and doing the tasks. In different circumstances where camera is much more important such as tasks that require very high levels of precision or tasks that are unable to be completed using only the virtual representation of a robot. FPV scenarios could potentially be higher ranked. Lastly, the 'Point And Click' scenario has potential as the easiest method to drive the robot around. Users found it unsafe and unintuitive due to the lack of control after selecting the point to which the robot should move. Additional programing and sensors could make this the easiest and most intuitive control method. A bigger study that uses additional scenarios, additional tasks, and additional participants should be conducted to identify and confirm these results.

4.2.3. Limitations

This study aims to evaluate the effectiveness and intuitiveness of various remote mobile robot control methods using a VR-based interface. An important limitation to mention is the relatively small sample size of participants. This can impact statistical significance of the results and can make generalizing that depends on participant type incorrect. Additionally, the study incorporates 5 different scenarios in different orders. Doing the same task with different control methods for the first or fifth time can significantly impact user satisfaction and performance. Especially for users that do not use VR devices often as they need some time to get used to the new interface and now controls.

Future research would benefit from using larger and more diverse participant groups to obtain more reliable data. Subjective questions present another limitation considering a low number of participants. Participant feedback can vary depending on their view on VR and how they interpret questions. Most of the questions are multiple choice and close-ended in order to minimize different interpretations. Objective metrics such as task completion speed and number of errors were added to compare participants feeling with objective scores they had. Lastly, bugs in the system as well as collisions with the environment that made the robot's virtual position different to the real robot's position caused instances where the robot had to be manually set to a correct position. Additionally, a tutorial covering all the controls and tasks over the five different scenarios can be added to help users understand and adapt to the new environment. This can help minimize differences that occur between first and last scenario during the study.

Conclusion

This thesis covers the research, design, development, and evaluation of VRobo, a virtual reality-based application that uses a DT for real-time synchronized teleoperation of a physical mobile robot. The prototype enables remote operators to control a physical robot by moving the virtual robot within a virtual scene, providing real-time camera visual feedback and sensor data from the physical robot.

The development includes setup and use of bidirectional communication between the physical and the virtual robot, use of a private 5G network and mapping of VR controller inputs into robot actions. Methodologically, a conducted user study provided data about the most intuitive interaction modes and synchronization errors.

The system has shown promising results in robot control and physical environment understanding inside VR. The camera stream from the robot that is seen inside the VE did not help participants in completing tasks faster. However, it did give participants better understanding of the environment and made them feel safer when doing tasks around the room. High levels of synchronization enabled users to confidently teleoperate a robot while performing different tasks. The study also helped identify the importance of camera, latency, and intuitiveness of a robot's controls when teleoperating a mobile robot.

In the future, improvements can be made by using more advanced sensors and by additionally optimizing latency using a different 5G network setup. This can enable users to better explore and understand dynamic environments inside VR and will offer users immersive experience for robot teleoperation. Making sure no errors in synchronization happen between physical and virtual worlds is another very important task as those errors can often stop the whole system from working correctly.

References

[1] Thamrongaphichartkul, K., Worrasittichai, N., Prayongrak, T., & Vongbunyong, S "A framework of IoT platform for autonomous mobile robot in hospital logistics applications." *2020 15th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP)*. IEEE, 2020.

[2] Beltrán-González, Carlos, et al. "Methods and techniques for intelligent navigation and manipulation for bomb disposal and rescue operations." *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*. IEEE, 2007.

[3] Byrn, John C., et al. "Three-dimensional imaging improves surgical performance for both novice and experienced operators using the da Vinci Robot System." *The American Journal of Surgery* 193.4, 2007: 519-522.

[4] Fong, Terrence, Illah Nourbakhsh, and Kerstin Dautenhahn. "A survey of socially interactive robots." *Robotics and autonomous systems* 42.3-4, 2003: 143-166.

[5] Dellin, Christopher M., et al. "Guided manipulation planning at the darpa robotics challenge trials." *Experimental Robotics: The 14th International Symposium on Experimental Robotics*. Springer International Publishing, 2016.

[6] Moniruzzaman, M. D., et al. "Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey." *Robotics and Autonomous Systems* 150, 2022: 103973.

[7] Gallipoli, Marco, et al. "A virtual reality-based dual-mode robot teleoperation architecture." *Robotica*, 2024: 1-24.

[8] Galarza, Bryan R., et al. "Virtual reality teleoperation system for mobile robot manipulation." *Robotics* 12.6, 2023: 163.

[9] Kaarlela, Tero, et al. "Towards metaverse: Utilizing extended reality and digital twins to control robotic systems." *Actuators*. Vol. 12. No. 6. MDPI, 2023.

[10] Fan, Wen, et al. "Digital twin-driven mixed reality framework for immersive teleoperation with haptic rendering." *IEEE Robotics and Automation Letters*, 2023.

[11] Koubaa, Anis, ed. *Robot Operating System (ROS)*. Vol. 1. Cham, Switzerland: Springer, 2017.

[12] Basañez, Luis, and Raúl Suárez. "Teleoperation." *Springer Handbook of Automation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. 449-468.

[13] Borenstein, Johann, et al. "Mobile robot positioning: Sensors and techniques." *Journal of robotic systems* 14.4, 1997: 231-249.

[14] González, Claudia, et al. "Advanced teleoperation and control system for industrial robots based on augmented virtuality and haptic feedback." *Journal of Manufacturing Systems* 59, 2021: 283-298.

[15] Tao, Fei, et al. "Digital twin modeling." *Journal of Manufacturing Systems* 64, 2022: 372-389.

[16] Jones, David, et al. "Characterising the Digital Twin: A systematic literature review." *CIRP journal of manufacturing science and technology* 29, 2020: 36-52.

[17] Kaur, Maninder Jeet, Ved P. Mishra, and Piyush Maheshwari. "The convergence of digital twin, IoT, and machine learning: transforming data into action." *Digital twin technologies and smart cities,* 2020: 3-17.

[18] LaValle, Steven M. *Virtual reality*. Cambridge university press, 2023.

[19] Virtual Reality Society. What is Virtual Reality? Last accessed on 17/12/2024. [Online]. Available : https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html

[20] Rubio-Tamayo, Jose Luis, Manuel Gertrudix Barrio, and Francisco García García. "Immersive environments and virtual reality: Systematic review and advances in communication, interaction and simulation." *Multimodal technologies and interaction* 1.4, 2017: 21.

[21] Philipp A Rauschnabel, Reto Felix, Chris Hinsch, Hamza Shahab, and Florian Alt. What is xr? towards a framework for augmented and virtual reality. Computers in human behavior, 133:107289, 2022.

[22] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." *ICRA workshop on open source software*. Vol. 3. No. 3.2. 2009.

[23] Rosen, Eric, et al. "Testing robot teleoperation using a virtual reality interface with ROS reality." *Proceedings of the 1st International Workshop on Virtual, Augmented, and Mixed Reality for HRI (VAM-HRI)*. 2018.

[24] Hussein, Ahmed, Fernando García, and Cristina Olaverri-Monreal. "Ros and unity based framework for intelligent vehicles control and simulation." *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE, 2018.

[25] Geometry message definitions: Twist. Last accessed on 18/12/2024. Available : https://docs.ros.org/en/jazzy/p/geometry_msgs/interfaces/msg/Twist.html

[26] Unity Colliders documentation. Last accessed on 18/12/2024. Available : https://docs.unity3d.com/540/Documentation/Manual/CollidersOverview.html

[27] Slater, Mel, et al. "How we experience immersive virtual environments: the concept of presence and its measurement." *Anuario de psicología* 40.2, 2009: 193-210.

[28] Wikipedia. Unity (game engine). Last accessed on 19/12/2024. [Online]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine)

[29] Linowes, Jonathan. *Unity 2020 virtual reality projects: Learn VR development by building immersive applications and games with Unity 2019.4 and later versions*. Packt Publishing Ltd, 2020.

[30] C Sharp (programming language). Last accessed on 19/12/2024. [Online]. Available: https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[31] ROS System. Last accessed on 19/12/2024. Available: https://www.ros.org/

[32] Meta Quest. Import Meta XR SDKs in Unity Package Manager. Last accessed on 19/12/2024. [Online]. Available : https://developers.meta.com/horizon/documentation/unity/unity-package-manager/

[33] Meta Quest 3. Last accessed on 22/12/2024. [Online]. Available : https://en.wikipedia.org/wiki/Meta_Quest_3

[34] Cañellas, Ferran, et al. "5G NR, Wi-Fi and LiFi multi-connectivity for Industry 4.0." *IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2023.

[35] Wen, Miaowen, et al. "Private 5G networks: Concepts, architectures, and research landscape." *IEEE Journal of Selected Topics in Signal Processing* 16.1, 2021: 7-25.

[36] Chen, Lei, Narasimha Shashidhar, and Qingzhong Liu. "Scalable secure MJPEG video streaming." *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2012.

[37] Zhang, Shunliang. "An overview of network slicing for 5G." *IEEE Wireless Communications* 26.3, 2019: 111-117.

[38] Torres, Edison Orlando Cobos, Shyamprasad Konduri, and Prabhakar R. Pagilla. "Study of wheel slip and traction forces in differential drive robots and slip avoidance control strategy." *2014 American Control Conference*. IEEE, 2014.

[39] Rosen, Eric, et al. "Testing robot teleoperation using a virtual reality interface with ROS reality." *Proceedings of the 1st International Workshop on Virtual, Augmented, and Mixed Reality for HRI (VAM-HRI)*. 2018.

[40] Naceri, Abdeldjallil, et al. "Towards a virtual reality interface for remote robotic teleoperation." *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, 2019.

[41] Crespo Allueva, Desirée. "Performance comparison of video compression algorithms for digital cinema." Master thesis, Universitat Politècnica de Catalunya, Barcelona Tech - UPC, 2014.

[42] Gary R VandenBos. APA Dictionary of Psychology. American Psychological Association, 2007.

[43] Jahromi, Hamed Z., et al. "You drive me crazy! interactive QoE assessment for telepresence robot control." *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, 2020.

[44] Definition of a Digital twin. Last accessed on 11/2/2025. Available : https://www.digitaltwinconsortium.org/initiatives/the-definition-of-a-digital-twin/

List of Figures

Figure 2.1: Real-world and VR application communication over a 5G network. 8

Figure 2.2: ROS - Unity Communication link overview, adapted from [24]. 9

Figure 2.3: Virtual scene of the room. 10

Figure 2.4: Picture of the physical room. 11

Figure 3.1: Private 5G network architecture example for this prototype. 17

Figure 3.2: Front and back camera view on the panel above the robot. 19

Figure 3.3: Low-poly robot and other objects in the scene. 20

Figure 3.4: Mapping of VR controller buttons to robot control actions in the VRobo prototype. 21

Figure 3.5: Code responsible for mapping sensitivity of thumbstick to motor actions. 21

Figure 3.6: First Person view panel inside the DT with both cameras turned on. 22

Figure 3.7: Reading input from Meta Quest 3 VR controllers. 23

Figure 3.8: Transforming controller inputs into twist messages. 24

Figure 3.9: Panel above the robot showing battery percentage and ultrasonic sensor readings. 25

Figure 3.10: Code used for subscribing to and parsing battery data in Unity. 26

Figure 3.11: Unity window with settings used for connecting to the ROS. 28

Figure 3.12: ROS code that publishes battery percentage data. 29

Figure 3.13: Unity code that published to the /cmd_vel topic. 30

Figure 3.14: Parts of python script used to start and maintain video stream from the cameras. 31

Figure 3.15: Unity script that reads the video stream from a URL. 32

Figure 4.1: Virtual representation of the DT of the space used for the study. 37

Figure 4.2: Real life space used for the study. 38

Figure 4.3: Screenshots of four different control methods inside VR: Robot Camera (a), Robot No Camera (b), Point And Click (c), and FPV (d). 39

Figure 4.4: Average participants ratings on quality of video (with 95% CI). 43

Figure 4.5: Participants' average ratings on ability to complete the task (with 95% CI). .. 44

Figure 4.6: Participants' ratings on ease of navigation (with 95% CI). 44

Figure 4.7: Distribution of participant ratings per scenario when asked about ease of navigation. 45

Figure 4.8: Participants' average ratings on overall QoE (with 95% CI). 46

Figure 4.9: Participants' average ratings on level of frustration (with 95% CI). 46

Figure 4.10: Number of ratings given to each scenario based on frustration. 47

Figure 4.11: Average rank of scenarios in terms of ease of use (with 95% CI). 48

Figure 4.12: Participants' latency and video quality preference. 49

Figure 4.13: Participants feelings about camera help when doing tasks. 49

Figure 4.14: Number of collisions per scenario (with 95% CI). 50

Figure 4.15: Average task completion time in seconds for each of the tasks across all the scenarios (with 95% CI). 51

Figure 4.16: Average participant time to finish the second task in seconds (with 95% CI). 52

List of Tables

Table 3.1: Performance comparison of video compression standards: MJPEG2000, HEVC HM, H.264, and x265 taken from [41]. 34

Table 4.1: Experience using VR devices prior to the study. 42

Table 4.2: Participant attitude towards VR technology 42

Abbreviations

2D	*Two-Dimensional*
3D	*Three-Dimensional*
AMCL	*Adaptive Monte Carlo Localization*
API	*Application Programming Interface*
AR	*Augment Reality*
DT	*Digital Twin*
FPV	*First Person View*
HMD	*Head Mount Display*
IoT	*Internet of Things*
JPEG	*Joint Photographic Experts Group*
JSON	*JavaScript Object Notation*
MR	*Mixed Reality*
QoE	*Quality of Experience*
ROS	*Robot Operating System*
RQ	*Research question*
SDK	*Software Development Kit*
URL	*Uniform Resource Locator*
VE	*Virtual Environment*
VR	*Virtual Reality*
XR	*Extended Reality*

Appendix A. User Study Form

User study questionnaire

User study for Master's Thesis

"Mobile Robot Teleoperation using a Virtual Reality-based Representation of a Digital Twin"

The information and responses gathered by this user data will be used solely for research as part of the Master's Thesis "Mobile Robot Teleoperation using a Virtual Reality-based Representation of a Digital Twin ". All data will be evaluated collectively, and your personal information will be kept anonymous.

General information

1. **Age:**

   _____

2. **Gender:**

- Male
- Female
- Don't want to say
- Other...

3. **Choose the statement that best describes your experience with Virtual Reality (VR) devices**

- I have never used a VR device
- I have tried a VR device 1 to 3 times in my life
- I occasionally use a VR device, but less than once a month (on average)
- I use a VR device once a month or more

4. **Choose the statement that best describes your experience with using robot vehicles (e.g. RC cars)**

- I have never used a robot vehicle
- I have tried using a robot vehicle
- I use a robot vehicle a couple of times per year
- I use a robot vehicle every week

5. **How would you describe your attitude towards virtual reality technology**

Mostly negative

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Mostly positive

*Questions regarding each of the scenarios **separately***

6. **FPV Rate the quality of the video**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

7. **FPV Rate the ease of navigation**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

8. FPV Rate ability to complete the task

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

9. FPV Rate overall QoE

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

10. FPV Rate level of frustration

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

11. 🖭 ROBOT CAMERA Rate the quality of the video

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

12. 🖭 ROBOT **CAMERA Rate the ease of navigation**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

13. 🖭 ROBOT CAMERA Rate ability to complete the task

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

14. 📷 ROBOT CAMERA Rate overall QoE

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

15. 📷 ROBOT CAMERA Rate level of frustration

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

16. 🚫 ROBOT **NO CAMERA** Rate **ease of navigation**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

17. ⊘ ROBOT **NO CAMERA Rate ability to complete the task**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

18. ⊘ **ROBOT NO CAMERA Rate overall QoE**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

19. ⊘ **ROBOT NO CAMERA Rate level of frustration**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

20. 👇 POINT **& CLICK Rate the quality of the video**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

21. 👇 POINT **& CLICK** Rate **ease of navigation**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

22. 👇 POINT **& CLICK Rate ability to complete the task**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

23. 👇 POINT **& CLICK Rate overall QoE**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

24. 👇 POINT **& CLICK Rate level of frustration**

Poor

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Excellent

Control Methods: *Users' preferences across the four control methods*

25. Rank the following robot control and camera view methods in terms of ease of use (1 - Best, 4 – Worst, cannot choose the same number twice)

	1 - Best	2	3	4 - Worst
FPV screen	○	○	○	○
Robot camera view	○	○	○	○
Robot without camera view	○	○	○	○
Point and click driving	○	○	○	○

26. Which control method did you feel was the most intuitive? Why?

_____

27. Which control method did you find most frustrating or difficult? Why?

_____

28. How confident were you in controlling the robot using each method?
(1 = Not confident, 5 = Very confident)

	1	2	3	4	5
FPV screen	○	○	○	○	○
Robot camera view	○	○	○	○	○
Robot without camera view	○	○	○	○	○
Point and click driving	○	○	○	○	○

Efficiency and camera modes

29. IN FPV - Which video quality setting did you prefer when performing tasks

- ○ Higher latency higher resolution camera
- ○ Lower latency lower resolution camera
- ○ Neither

30. To what extent do you feel that including a camera view impacted your ability to accurately complete tasks?

- ○ didn't help at all
- ○ helped very little
- ○ somewhat helped
- ○ helped to a great extent

31. Did you feel that having the camera view helped you complete tasks faster as opposed to scenarios without a camera view?

- ○ didn't help at all
- ○ helped very little
- ○ somewhat helped
- ○ helped to a great extent

32. In which situations would you prioritize low latency over video quality? (more than 1 choice enabled)

- ✓ Navigation (Figuring out where to move next)
- ✓ Precision tasks (Parking)
- ✓ General driving (Moving around)
- ✓ I prefer high latency and high resolution in all

33. What improvements would you suggest for the control methods or video quality settings?

_____

34. Do you have any additional comments or feedback on your experience with controlling the robot?

_____

Objective metrics

35. FPV mode used first

- ○ High Quality
- ○ Low Quality

36. Number of collisions

_____

37. Time to complete task 1 (Driving around objects)

_____

38. Time to complete task 2 (Driving towards point)

_____

39. Time to complete task 3 (Parking)

_____

End of the Questionnaire!

Abstract

Mobile Robot Teleoperation using a Virtual Reality-based Representation of a Digital Twin

This Master's thesis presents the design and implementation of the VRobo prototype that integrates a physical robot with a virtual reality-based digital twin, allowing real-time teleoperation of the robot using a VR interface. It uses ROS to control the robot and its sensors and uses the Unity game engine to create an immersive virtual reality application for Meta Quest 3 with an interface for controlling the robot. The prototype communication is optimized using ROS Bridge and a 5G network to enable low-latency communication and achieve elevated levels of synchronization between the robots and technologies used. Quality of Experience testing was conducted in order to evaluate system performance, focusing on user satisfaction, intuitive interface for controlling the robot, and levels of operator immersion. This work advances the integration of robotics and virtual reality, using digital twins and 5G networks. It bridges the gap between virtual environments and real-world robotic control but also helps in creation and integration of virtual reality applications for operating dynamic real-world environments.

Keywords: Mobile Robot, Teleoperation, VR, Virtual Reality, Digital Twin, ROS, Unity, Meta Quest 3

Sažetak

Teleoperacija mobilnog robota korištenjem reprezentacije digitalnog blizanca temeljenoj na virtualnoj stvarnosti

Ovaj diplomski rad predstavlja dizajn i implementaciju VRobo sustava koji upravlja fizičkim robotom unutar digitalnog blizanca temeljenog na virtualnoj stvarnosti. Omogućuje daljinsko upravljanje robotom u stvarnom vremenu s pomoću sučelja virtualne stvarnosti. Koristi ROS za upravljanje robotom i njegovim senzorima te Unity game engine za stvaranje imerzivne aplikacije virtualne stvarnosti sa sučeljem za upravljanje robotom unutar *Meta Quest 3* virtualnih naočala. Komunikacija sustava optimizirana je s pomoću *ROS Bridge-a* i 5G mreže koja omogućuje komunikaciju niskog kašnjenja i koja omogućava postizanje visoke razine sinkronizacije između dvaju robota odnosno korištenih tehnologija. Rad sadrži i testiranje kvalitete iskustva kako bi se dobili dodani podaci o svojstvima sustava, fokusirajući se na zadovoljstvo korisnika, intuitivnost sučelja za upravljanje robotom te razinu imerzije upravljača. Ovaj rad pridonosi području robotike i virtualne stvarnosti, korištenju digitalnih blizanaca i 5G mreži. Stvara vezu između virtualnih okruženja i mobilnih robota u stvarnom svijetu, ali također pomaže u stvaranju i integraciji aplikacija virtualne stvarnosti za upravljanje dinamičkim okruženjima stvarnog svijeta.

Ključne riječi: Mobilni Robot, Teleoperacija, VR, Virtualna Stvarnost, Digitalni Blizanac, ROS, Unity, Meta Quest 3