Institute for Computer Science VII
Robotics and Telematics

Bachelor's thesis

Realtime point cloud streaming and visualization in Virtual Reality for remote operation of a mobile robot

Sven Kunze

June 2022

First reviewer: Prof. Dr. Andreas Nüchter
Advisor: Michael Bleier

Summary

This bachelor thesis describes a system for visualizing real-time point clouds in Virtual Reality for the remote control of mobile robots. It builds upon the Robot Operating System (ROS) for the handling of point clouds and communication and control of a mobile robot. Different methods for displaying point clouds in Virtual Reality are implemented and tested. For constructing and running this Virtual Reality system the video game engine Unity [14] is used. Additional point clouds from laser scanning and Structure from Motion (SfM) are included to visualize the static environment. These are either in vertex or meshed form. Remote control of the robot is achieved utilizing a VR controller or a glove with finger tracking and haptic feedback. This work evaluates certain performance aspects of such a system. Special focus is put on the measurements of latencies, such as the latency for point cloud transmission and displaying, as this might hamper adoption of a similar system for real-time use cases.

Contents

List of Figures

Chapter 1

Introduction

Remote operation have recently become more important because more people started to work from home due to the pandemic. But even before Covid-19 especially spacecraft have mostly been remotely controlled. Other applications involve dangerous tasks, e.g., dealing with the Fukushima nuclear disaster. But as the hardware gets cheaper more use cases will become financially viable. Notably repetitive tasks or those that only intermittently require human input may profit from using off-site control. In mining and underwater applications operators usually have to rely on cameras and 3D sensors capturing point clouds for information about the environment as there is no line of sight. This data is hard to interpret due to the lack of spatial context and depth perception. For better situational awareness a virtual reality (VR) system is used. In combination with a glove that tracks finger positions this allows quick and intuitive operation of a robot. Haptic feedback gives the user additional information about the forces involved. This work is part of the Tastsinn-VR project [12].

1.1 Contribution

In this work a system for remote control of a mobile robot in Virtual Reality is developed and tested (Fig. 1.1). This includes demonstrating a tool for creating the VR environment where the realtime sensor data is visualized. This work focuses on point clouds as the main method for communicating environmental 3D realtime information. Characteristics of the point cloud streaming and visualization system, such as latency, throughput and frame rate for different point clouds and packet sizes, are measured. The performance of the VR visualization is also compared to widely used tools such as RViz [10]. Additional visual elements, such as 3D models, fog and photogrammetric scans, are also integrated and tested. Different forms of presenting the point cloud, such as coloring based on reflectivity or height, are demonstrated.

1.2 Structure

This thesis is structured as follows: Firstly, the state of the art of all the necessary technology for this work is presented. This includes the sensors, the Virtual Reality system, the communication aspects, the robot and use cases for this work. In the third chapter the technical details of

Figure 1.1: Photo of the Tastsinn-VR project including a VR headset, a haptic finger-tracking glove, a mobile robot and a laser scanning backpack.

this work are described. It is divided into a visualization and a communication part. The visualization includes all the visual elements that make up the VR scene, such as static objects, the live point clouds and the remote control interface. In the communication part the employed protocols and the overall architecture is described. The fourth chapter portrays the subsystem and full-scale real world tests with a mobile robot. This includes performance measurements using prerecorded live point cloud data on a local computer and over a network connection. The last chapter highlights the conclusions drawn from this work and gives an outlook of how to improve and build on this work.

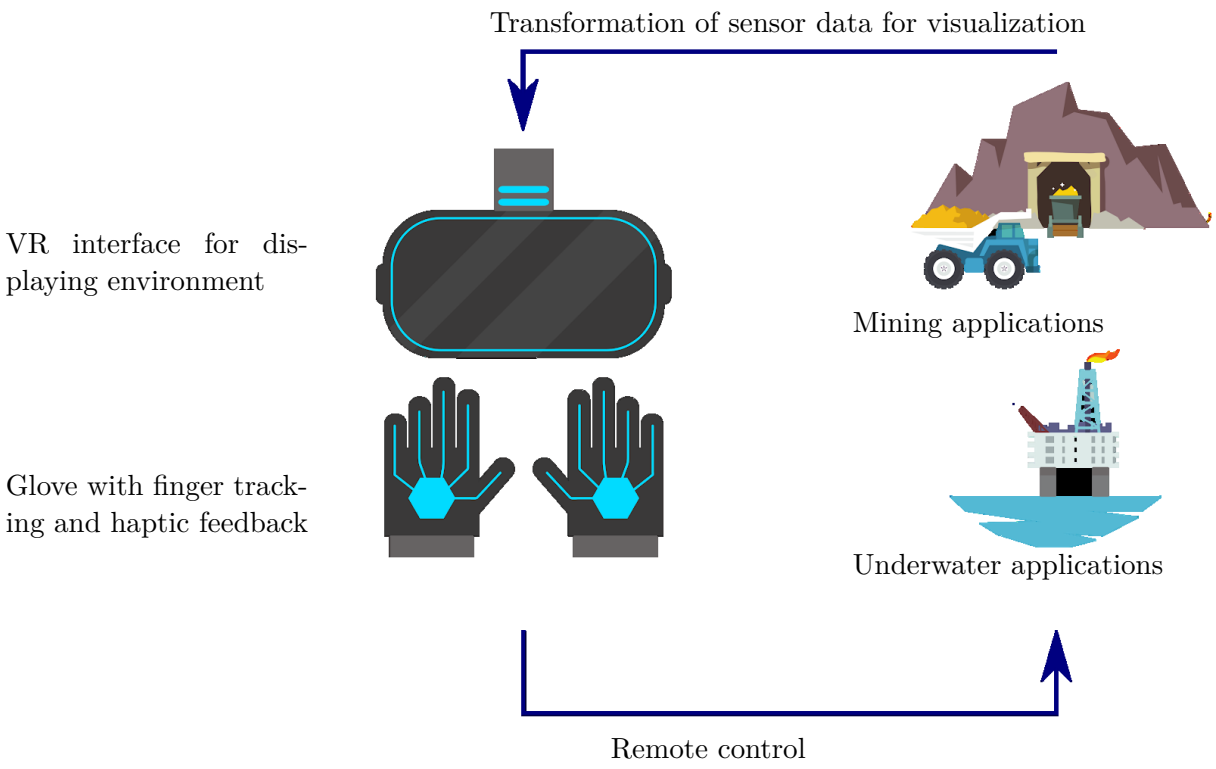

Figure 1.2: Overview of the system.

Chapter 2

State of the Art

2.1 Sensors

There are many different types of sensors which provide useful information for remote operation of robots. These include cameras, lidars, radars, ultrasonic sensors, odometers and inertial measurement units (IMUs). This work focuses on the use of lidar. Lidar is an acronym of "light detection and ranging". Most lidars today work by sending out laser pulses and measuring the return signal. The reason for using lasers instead of other sources of light is that they are focused to a small beam and have a very narrow frequency band. This makes it easier to filter out the the light by other sources from the return signal. There are many different methods to analyze the return signal. Among the simplest ones is to measure the time it takes for the first light to come back and determining its intensity. More complex methods, such as full wave analysis, try to gather as much data as possible from each signal. For transparent objects or those that have many holes (e.g. treetops) there are multiple returns for multiple surfaces at different distances. The lasers in lidars are often steered by rotating, swiveling and/or tilting mirrors to measure in a wide field of view. The components of a typical two-axis Lidar sensor are shown in Fig. 2.1. The direct output of such sensors are range and intensity images. These are then often converted into 3D point clouds. Because most lidars do not provide color information cameras are used to supplement that data.

2.2 Virtual Reality

Virtual Reality encompasses a lot of different technologies but for this work we will focus on head-mounted displays (HMD) (see Figure 2.2). This kind of VR has become increasingly popular in recent years mostly due to its use in computer games and VR video. But as the hardware has improved it is now also used in research and for training in various industries. Most VR headsets use one display for each eye for which images are separately rendered. Typical screens, such as the one in "Oculus Quest 2", have resolutions of around 1,832 by 1,920 pixels per eye and refresh rate of 120Hz [6]. They either have three or six degrees of freedom (3DOF vs 6DOF). Three degrees of freedom means that the headset only tracks rotation while 6DOF also includes translation. To achieve this gyroscopes, accelerometers and cameras that track

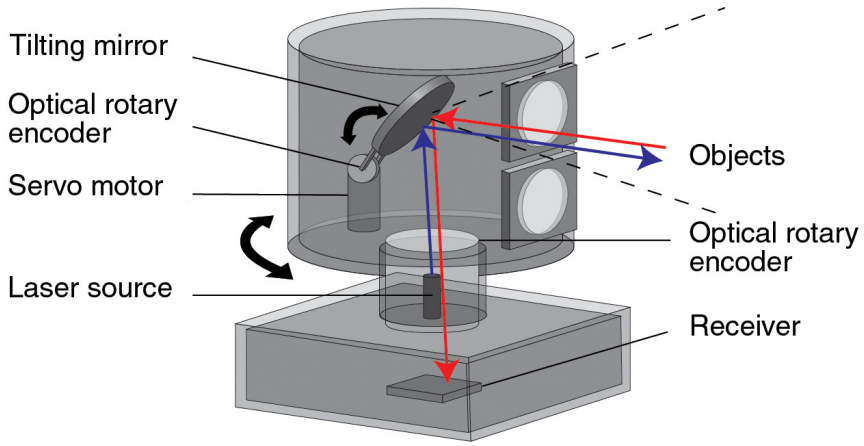

Figure 2.1: Components of a typical two-axis Lidar sensor [7].

features of the surroundings are used. Determining the position of hands and controllers is often done by the same cameras. In other systems the VR headset and controllers ascertain their position with stationary light-emitting beacons. Early headsets required a computer to render the images but now there are several standalone devices containing smartphone derived processors. Although these are less powerful in terms of processing and graphics capabilities than their PC counterparts they still serve many applications. For more demanding workloads many standalone headsets are also connected to a computer either via a USB cable or wirelessly. Because of bandwidth constrains the video stream is rendered and compressed on the computer and decompressed on the VR headset. This adds extra latency compared to normal PC virtual reality which is using a typical display signal, such as HDMI or DisplayPort.

Previous projects, such as the paper "Immersive Point Cloud Virtual Environments" [18], have displayed point clouds in VR using the 3DTK software library. It used the Oculus Rift Developer Edition headset with 6DOF tracking. The point cloud was acquired by a mobile robot with a laser scanner beforehand and included 10-20 millions points.

2.3 Communication

Communication includes several aspects: Firstly there is the connection between the VR system and the robot. For this the data is usually sent over Ethernet or Wifi using TCP/IP. Many middle-ware protocols work on top of this connection. For robots especially in the context of research the Robot Operating System (ROS) is used. It provides a server that nodes connect to. ROS nodes publish and subscribe to topics that are organised in a simple hierarchy. ROS also provides standard message formats for commonly used types of data, such as point clouds, transforms and images. Another communication protocol is WebSocket which is widely used for websites and similar applications. It offers a full duplex tunnel over TCP between a server and

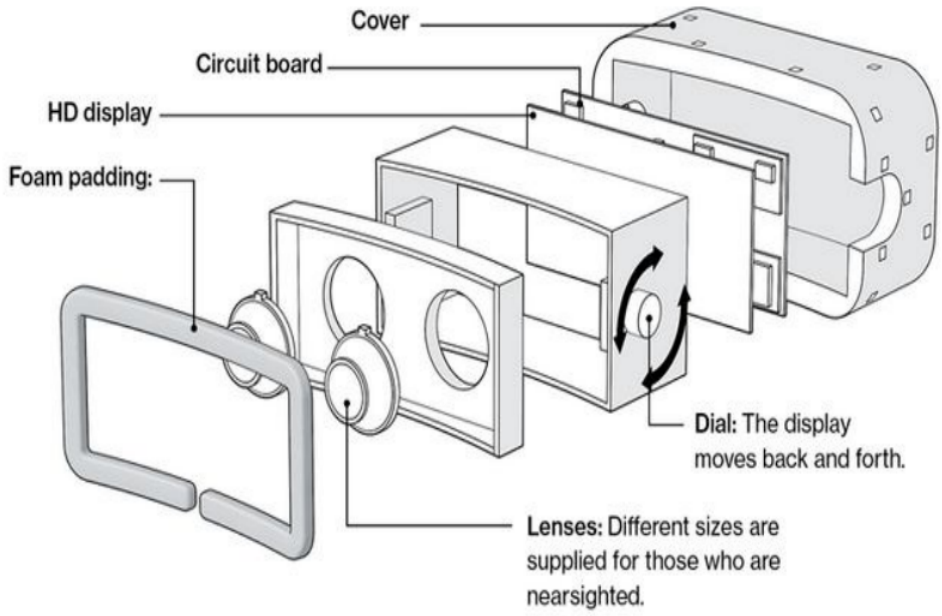

Figure 2.2: Components of a VR Headset [19].

a client [20]. In case the VR headset is not standalone there is also the important connection between the VR headset and the computer that renders the images.

2.4 Robots

A robot is a machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer [1]. Robots have sensors and actuators that allow them to sense and interact with their environment. There are many different types of robots: The two main categories are stationary and mobile robots. In many cases the actuators are driven by electric motors. Other ones are pneumatic and hydraulic powered especially when high amounts of force are needed. There are many different types of sensors: These include cameras, radars, lidars, microphones, gyroscopes, accelerometers, speed sensors, thermometers, force and pressure sensors. Robots can be remote-controlled or have different levels of autonomy.

2.5 Use cases

As discussed above this work has many applications, especially in contexts where it is easier, safer and/or faster to remotely operate a robot. Specific use cases include mining and underwater robots where the environment is dangerous. But as this technology gets cheaper and simpler to develop and use, more possibilities will be opened up. For example, in a production environment where there are already robots of many types, a virtual reality system helps monitoring and allows quick intervention in case of a problem.

In the "¡VAMOS! PROJECT" an underwater robot was controlled using a Virtual Reality system [17]. The visualization for the operator was done using a screen showing different perspectives of the digital replica including a 3D model of the robot and the underwater terrain, as well as live point cloud scans.

Chapter 3

Virtual Reality for Remote Control

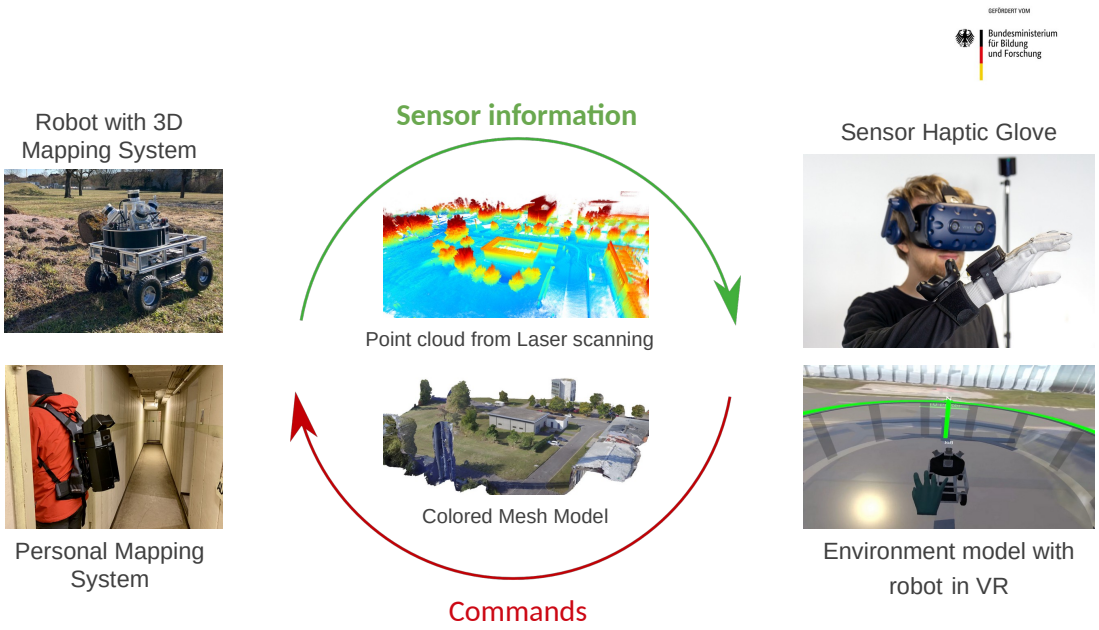

Figure 3.1: Components of the system.

3.1 Volksbot Ackermann

In this work a "Volksbot Ackermann" (Figure 3.2) by the Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS) is used as a mobile robot. It has two axis and four wheels. One axis is used for Ackermann steering. Mounted on top are four Lidar Scanners as can be seen in Figure 1.1 which are connected to the onboard computer that is hosting the ROS system. The robot also contains a wireless router for communicating to the VR system.

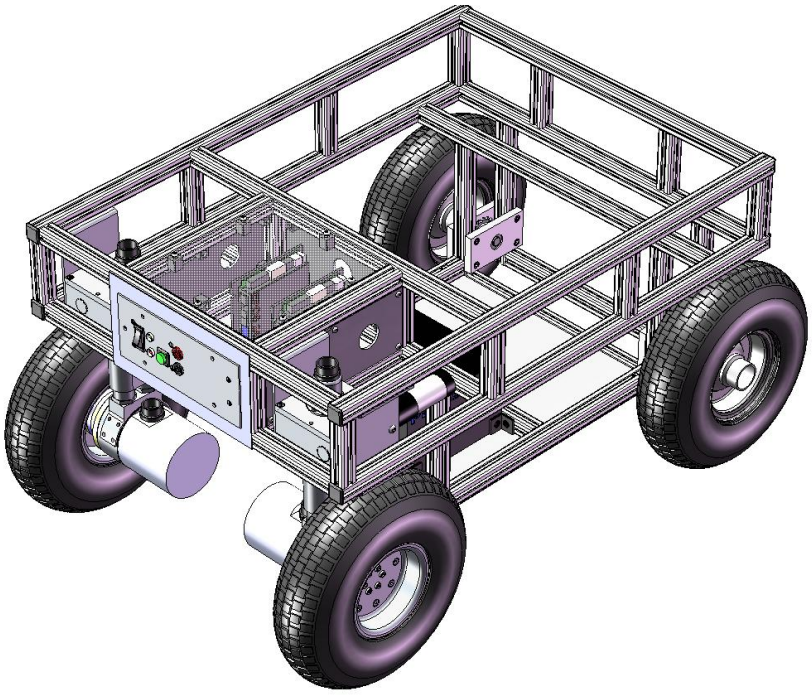

Figure 3.2: CAD Model of a Volksbot Ackermann [2].

3.2 Visualization

For visualization the video game engine "Unity" is used. It offers a number of features that are needed for this project: Firstly, it has good support for developing VR interfaces by having plugins for the major VR libraries, such as "SteamVR" [11], which is used in this work. Unity also provides many useful features for 3D rendering, such as custom shaders used for point cloud rendering. Because Unity allows for general purpose 3D visualizations, additional elements, such as 3D models, are easily added.

3.2.1 glTF

The Unity editor allows creating and editing scenes where visual elements and supporting code are imported and positioned. To run the program outside of the Unity editor it needs to be exported. This also contains the exact description of the relevant scenes. A disadvantage of this process is that the scenes can not be edited after exporting. One solution to this problem is to separate the scene description from the program rendering and executing the scene. In this work the "glTF" format is used to allow changing the VR scene after compilation of the code. This enables faster iteration and more flexibility. glTF (derivative short form of Graphics Language Transmission Format) is a standard file format for three-dimensional scenes and models [4]. For exporting and importing glTF files the projects "UnityGLTF" [15] and "GLTFUtility" [3] are used. glTF natively only supports 3D meshes, textures and a scene hierarchy with translations, rotations and scale. For the variables of custom Unity scripts the "extras" property of the glTF

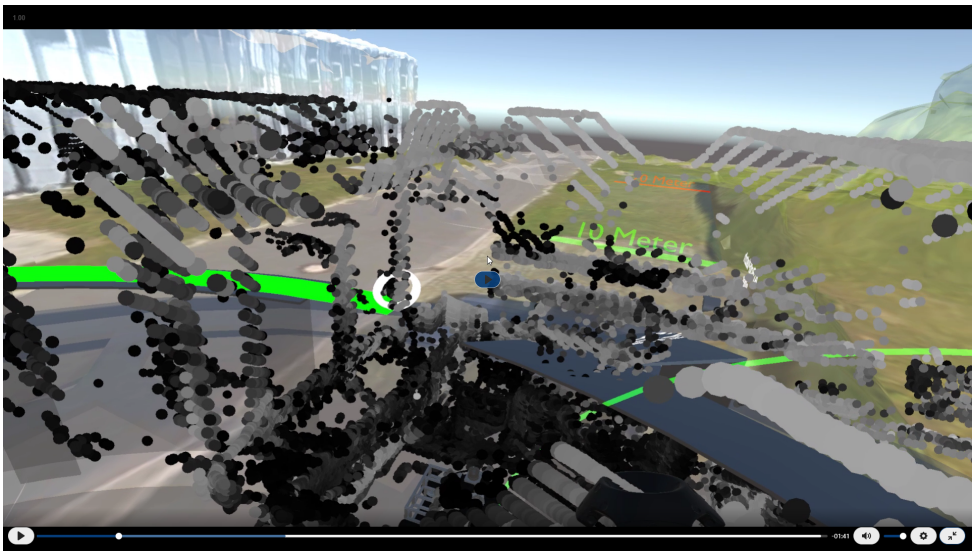

Figure 3.3: Point cloud with photogrammetric scan in background.

specification is used. As the glTF format is based on JSON the variables in the Unity scripts are also serialized as JSON. JSON (JavaScript Object Notation) is a lightweight human-readable data-interchange format [5]. The code and therefore the logic is not stored in the glTF document but is part of the program that is exported from the Unity editor. Therefore only certain scene information, such as translation, rotation, scaling and the public variables of Unity scripts, can be changed in the glTF file.

3.2.2 Static objects

Static objects are all objects that do not use live data. This includes photogrammetric scans of the environment which are captured beforehand (Fig. 3.3.) There is also the Navigation system which facilitates movement in VR. The VR system provides 6DOF tracking but this is limited by the size of the room and the capabilities of the tracking system. In this case using 4 tracking base stations one gets a maximum area of 10m by 10m [16]. As the visualized point clouds have dimensions much larger than that another system for moving within the scene is needed. This is accomplished with a kind of drag gesture: When the user holds down the grip button on the VR controller the entire scene is moved according to the movement of that controller. In this case 1m of movement translates to 1m of movement in the VR Scene. For larger distances this motion of the hand might be tiring. That is why one can also adjust the scale of the scene by holding down the grip button on both controller and moving the hands apart or closer together.

3.2.3 Point clouds

Point clouds are captured by lidar sensors or similar ones and sent over the network using the ROS message format ”sensors_msgs/PointCloud2”. But the shader that ultimately renders the point cloud in Unity uses a different format. Therefore the PointCloud2 needs to be converted.

To render point clouds in Unity the Unity Package "Pcx" [8] is used. It only support static point clouds out of the box but it was modified to load point clouds from network sources in the form of byte arrays at runtime. These byte arrays need to have the following format: A single point consists of 3 * 4 bytes for 32 bit floating numbers representing the x, y and z coordinates followed by 4 bytes containing 8 bit unsigned integers representing rgba values. Therefore one point makes up 16 bytes. For the full point cloud these individual points are just laid out sequentially in the byte array without any padding. Point clouds coming from ROS as PointCloud2 messages also have a byte array containing the point data. But it is in a different format than is needed for the Unity shader. Therefore the point cloud needs to be processed before it is sent to Unity. This is done by a python program using the "numpy" and "ros_numpy" modules. "Numpy" is a Python library that is capable of handling large multidimensional arrays. It is written in C and is therefore very performant. The floating point numbers containing the position need to be filtered and placed at the correct positions in the byte array for Unity. For the color values it is a little more complicated: Many sensors don't capture color information but only intensity or reflectivity, which is the absolute intensity of light returned from a point scaled to compensate for different distances. The simplest way to calculate the color value from the reflectivity is gray scale where the red, green and blue channel all have the same value. This is all done with numpy functions which makes it very fast and efficient. Then this binary data is sent to Unity and received. As it is already processed into the correct format for Pcx no more filtering or transforming is done in Unity. The point clouds is then loaded and held in GPU memory. Once the data is received a new GPU buffer of the correct length is created. The size in bytes of it is a multiple of 16 because that is the size of an individual point as decribed above. Then this buffer is filled with the received byte array. The point cloud is rendered using a C# script with a custom point cloud shader.

3.2.4 Remote control

Remote control of the mobile robot is achieved with the use of goal poses: The operator in the VR environment can either use a button on 6DOF-tracked VR controller or using the finger-tracking glove (Fig. 3.4). In the case of the glove a 6DOF Vive tracker is utilized to determine the position of the goal and the goal is placed when the user forms a fist. Once a goal pose is put down, as can be seen in Figure 4.16, it is published through a ROS topic that the mobile robot is subscribed to. The robot then uses its steering algorithm to drive to that position. It determines its position relative to the goal pose via its odometry.

3.3 Communication

Efficient and low latency communication is an important aspect of this work as the sensors are handled by different computers than the visualization and control components. Figure 3.5 shows a general overview of the communications architecture. Only the components highlighted in green are relevant for this work. But to allow a similar system to be used for more applications additional inputs and outputs are possible and are already utilised in other contexts.

Figure 3.4: Finger-tracking glove with Vive tracker and VR Headset.

3.3.1 ROS

ROS has a variety of clients for different programming languages. As the WebSocket server is written in Python the rospy and roslibpy libraries are used. Rospy directly receives and sends ROS messages. Roslibpy requires rosbridge which converts native ROS messages to a JSON format. This is useful for Unity as there are no .NET/C# native ROS libraries. Therefore the default way for Unity to receive ROS messages is as JSON using the rosbridge. This works well for small messages but not for large ones like point clouds. As binary data, such as point clouds, cannot be directly contained in JSON it is converted to base64 and needs to be converted back before visualization in Unity. This adds latency and additional computation. To avoid this the WebSocket server handles point clouds differently than the default way described above. It receives the native ROS messages using the python module ”rospy” instead of ”roslibpy”. These message are converted as described in Section 3.2.3. The created byte array is then sent over WebSocket as binary data. ROS also has a convenient way of recording and storing ROS messages called ”rosbag” [9]. This used to test the realtime streaming and visualization of point clouds in VR.

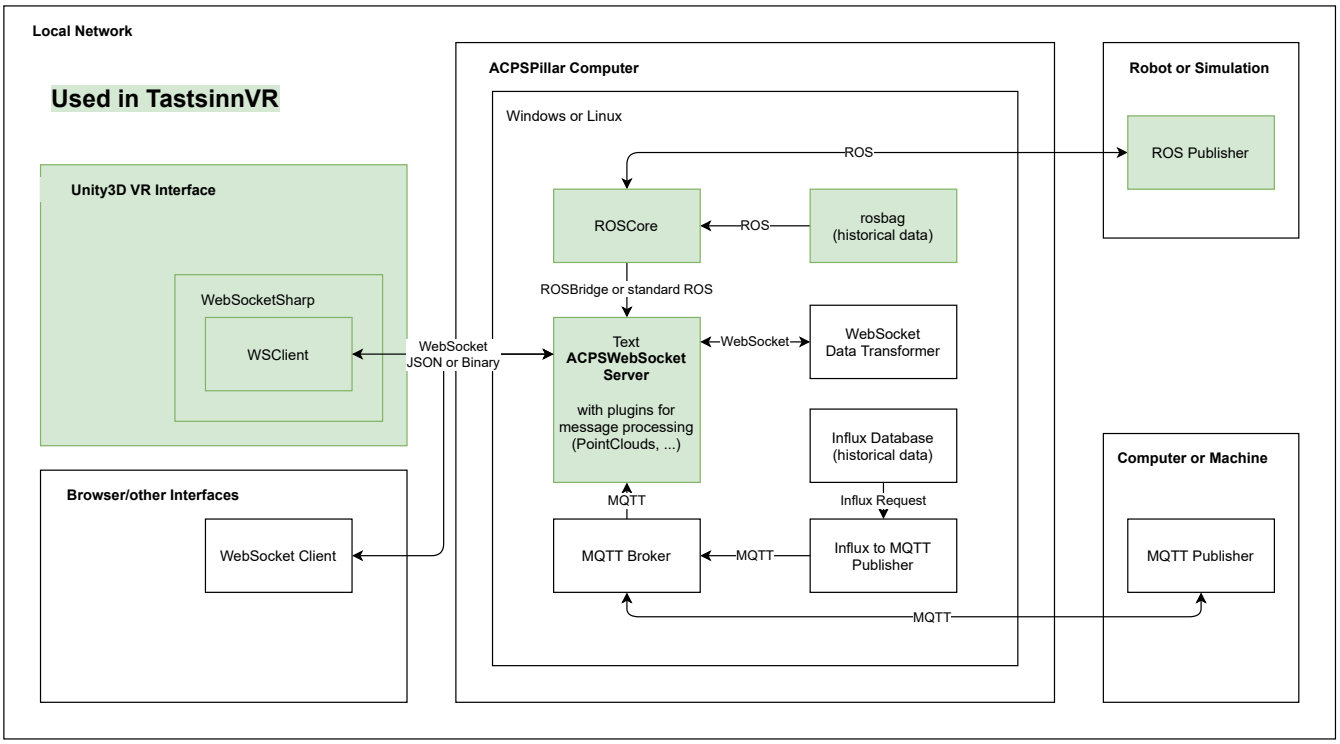

Figure 3.5: Communications Architecture.

3.3.2 Scene server

As mentioned in Section 3.2.1 the scene description of all visual elements is stored in an external document using the human-readable format glTF. This file is either stored locally on the device that is running the exported Unity program or downloaded at runtime from an external server. This is the purpose of the scene server which provides the scene description over the network. It is an HTTP server which delivers the requested glTF files specified by a certain path.

Chapter 4

Experiments

4.1 Local tests

Firstly, the performance of the VR point cloud visualization is tested locally which means only using one computer. For this purpose a rosbag containing recorded point clouds is used. It was created by driving a mobile robot equipped with lidars on the campus of the University Würzburg. It has a compressed size of approximately 4.51GB and a length of 122 seconds. It contains the 6 topics listed in Table 4.1. The messages in the "/map/cloud" topic have point clouds containing between 27033 and 140324 points. This topic is an aggregate of all the lidar scans and therefore increases in point number over time. The local tests were done on Windows 10 PC running on an Intel i7-4770K @ 3.5GHz with 16 GB of RAM and a NVIDIA GeForce GTX 960 with 2GB of VRAM.

Figures 4.1 to 4.5 display several screenshots in the Unity Editor of the visualized point clouds contained in the above mentioned rosbag. In these screenshots different colormaps, such as grayscale and "Turbo" [13], are used. There are also two types of underlying color information: One is just based on the height coordinate; the other expresses the captured reflectivity values. In Figure 4.5 an additional fog is utilized to enhance depth perception.

ROS Topic	No. msgs	ROS Message Type	Description
/map/cloud	169	sensor_msgs/PointCloud2	Over Time Aggregate of the lidar scans
/map/pose	169	geometry_msgs/PoseStamped	Pose for /map/cloud
/robot/cloud	1197	sensor_msgs/PointCloud2	Individual lidar scans
/robot/pose	1197	geometry_msgs/PoseStamped	Pose for /robot/cloud
/tf	25598	tf2_msgs/TFMessage	Coordinate Frames
/tf_static	6	tf2_msgs/TFMessage	Static Coordinate Frames

Table 4.1: Topics of the rosbag.

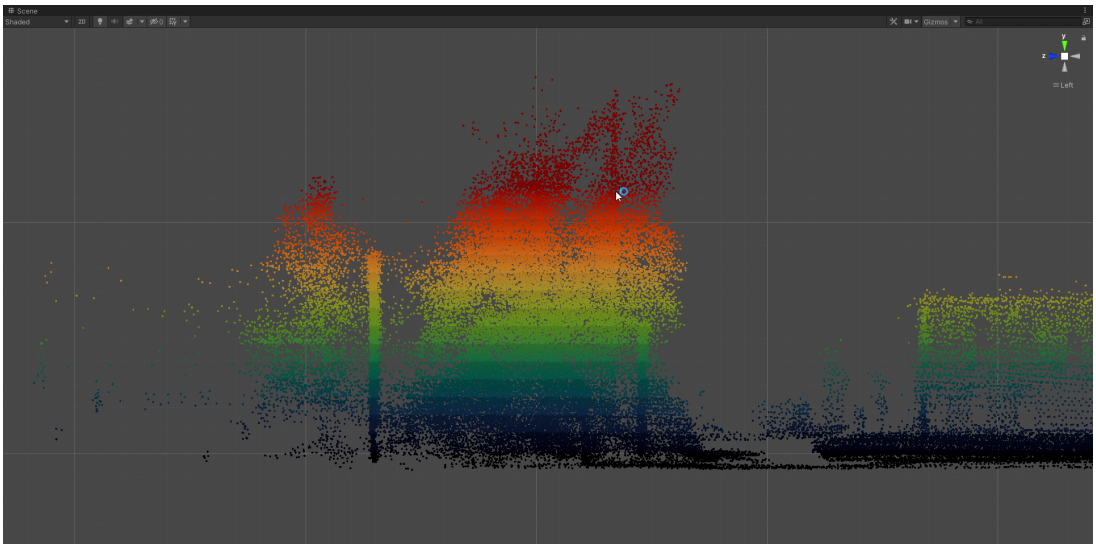

Figure 4.1: Screenshot of the Unity Editor displaying a point cloud colored using the height coordinate and the ”Turbo” colormap - Side view.

4.1.1 Comparison with RViz

As a first measurement of the performance of the VR point cloud visualization system it is compared against RViz. RViz is a 3D visualizer for the Robot Operating System (ROS) framework [10]. For this comparison the rosbag mentioned in Section 4.1 is played and the completion timestamps of the individual steps in the transmission and visualization pipeline are recorded. The four measured steps are:

1. The point cloud is received by the WebSocket server
2. The WebSocket server starts sending the point cloud to the VR visualization in Unity.
3. The VR visualization in Unity starts receiving the point cloud
4. The VR visualization in Unity finishes receiving the point cloud

These timestamps are then compared to the respective times when the loading of the point cloud in RViz is completed. This is done by starting RViz with the ”–log-level-debug” argument which prints out the timestamps when ROS messages are received. This process is repeated for different packet sizes containing numbers of points ranging from 1000 to 150000. RViz and Unity both use the system clock; therefore the timestamps can be compared against each other. The delta times between the steps for Unity and RViz and corresponding point cloud sizes are graphed in Figures 4.6 to 4.13. The right vertical axis applies to the number of points while all other series are delta times and pertain to the left vertical axis.

One result from this comparison is that the additional latency compared to RViz is comparable between the test where individual packets contain 1000 to 4000 points. For packets sizes 8000 and 16000 especially the completion time for Unity goes up. For even larger packet sizes also the starting times for loading the point clouds in Unity also increases.

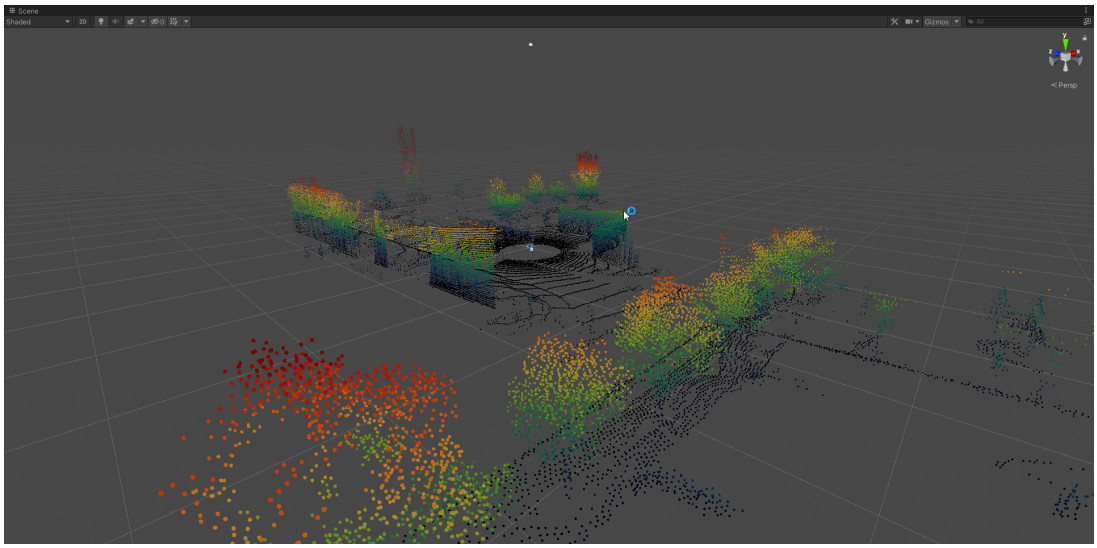

Figure 4.2: Screenshot of the Unity Editor displaying a point cloud colored using the height coordinate and the "Turbo" colormap.

4.2 Test over network connection

The same rosbag is used to test the streaming of point clouds over a wireless network connection. In this case the roscore, rosbridge and playback are running on a laptop while the Unity instance and the WebSocket server run on the same computer from the previous tests. The laptop is connected to the router via the 5GHz WiFi band while the other computer is connected via Ethernet. As one can see in Figure 4.14 the latency is higher than in the local test but remain below 0.8 seconds for point clouds smaller than 100000 points. For the bigger point clouds the bandwith of about 100 Mbit/s is not enough to transmit those point clouds, that is why the latency increases sharply.

4.3 Test with a mobile robot

One of the goals of this work is to visualize live point cloud data coming from lidar scanners mounted to a mobile robot. The results of this test is described in this section. The architecture as described before and shown in Figure 3.5 is set up with the computer hosting the VR system connected to the WiFi network of the mobile robot. The robot publishes a combined point cloud scan over one ROS topic that the WebSocket server and by extension the Unity VR program subscribe to. In Figure 4.15 the operator controls the robot in the bottom right corner by setting a goal pose for the robot.

In Figure 4.16 one can see a screenshot of the view in the VR Headset. The scene contains a 3D model of the mobile robot in the correct live pose as well as the realtime point cloud in white. The green object is a goal pose that the user has set which the robot tries to navigate to. The grey curve is part of the VR Navigation system. It helps to provide the user with a solid reference frame in VR to avoid motion sickness; especially when there is no live data yet. After

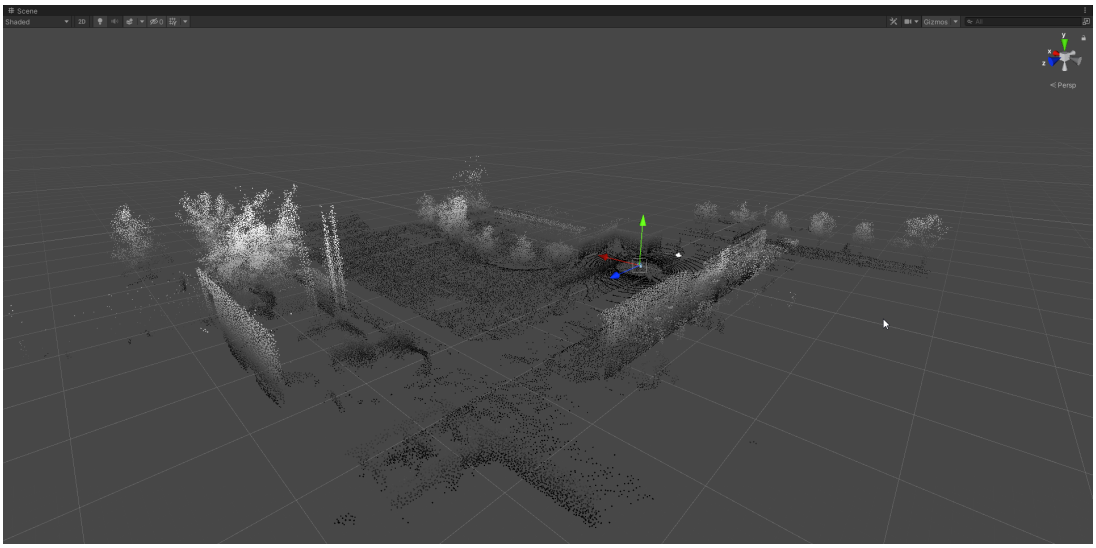

Figure 4.3: Screenshot of the Unity Editor displaying a point cloud colored using the height coordinate and a grayscale colormap.

the goal pose is set, it takes about 5 seconds before one can see the movement of the mobile robot starts towards the goal pose in the VR scene (measured by analysing a video recording). The robot reaches the goal pose after 6 seconds as shown in Figure 4.17.

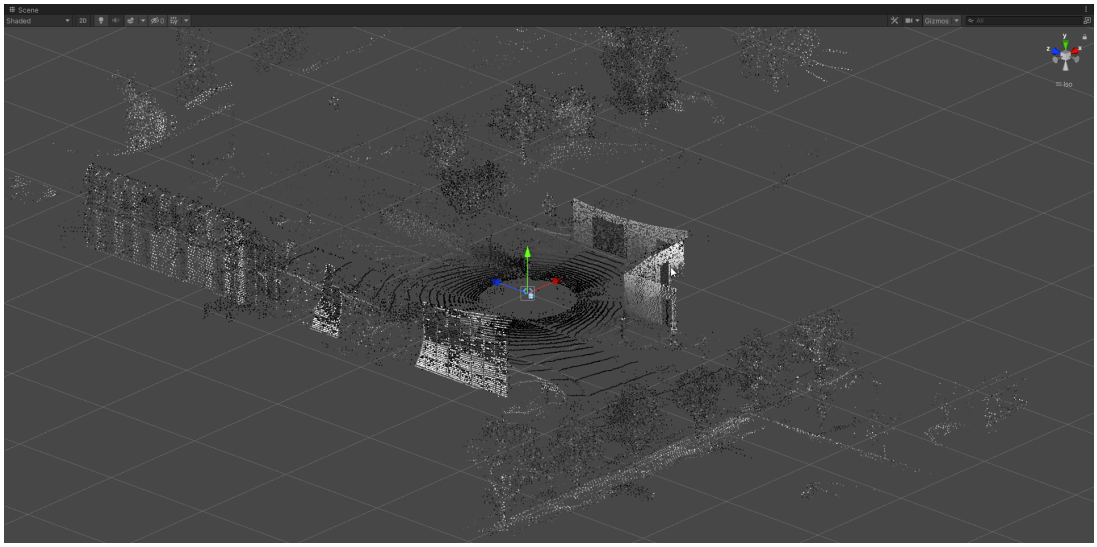

Figure 4.4: Screenshot of the Unity Editor displaying a point cloud colored using the reflectivity values and a grayscale colormap (isometric projection).

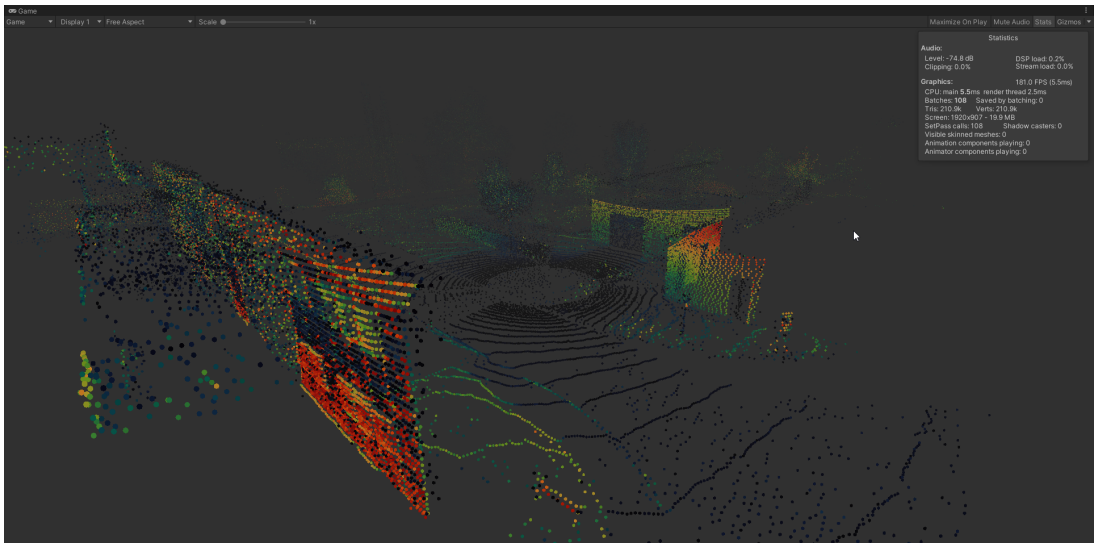

Figure 4.5: Screenshot of the Unity Editor displaying a point cloud colored using the reflectivity values, the "Turbo" colormap and fog.

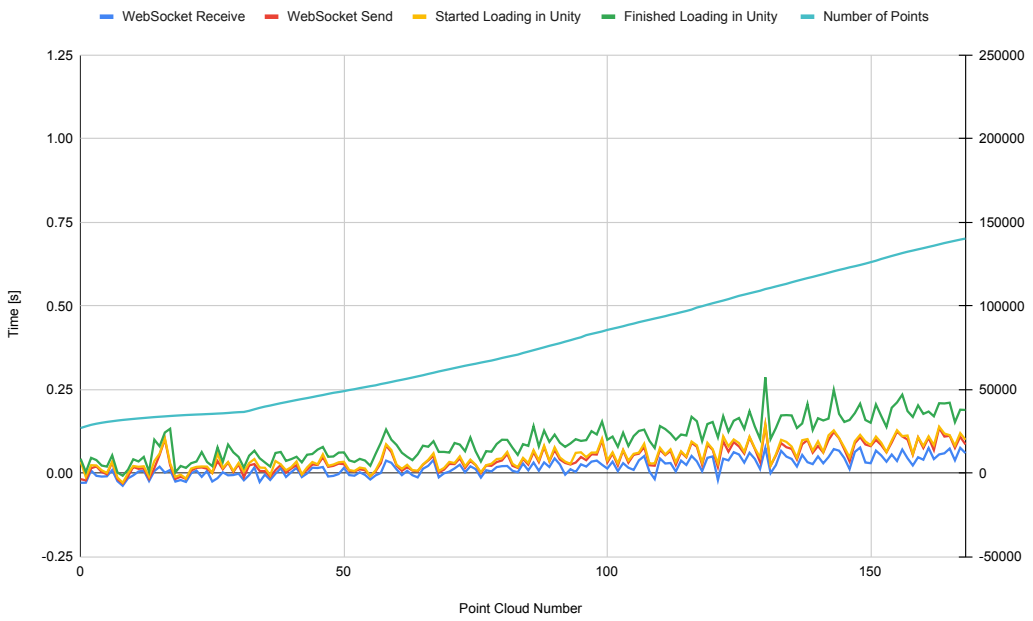

Figure 4.6: Delta time compared to RViz using 1000 points per packet.

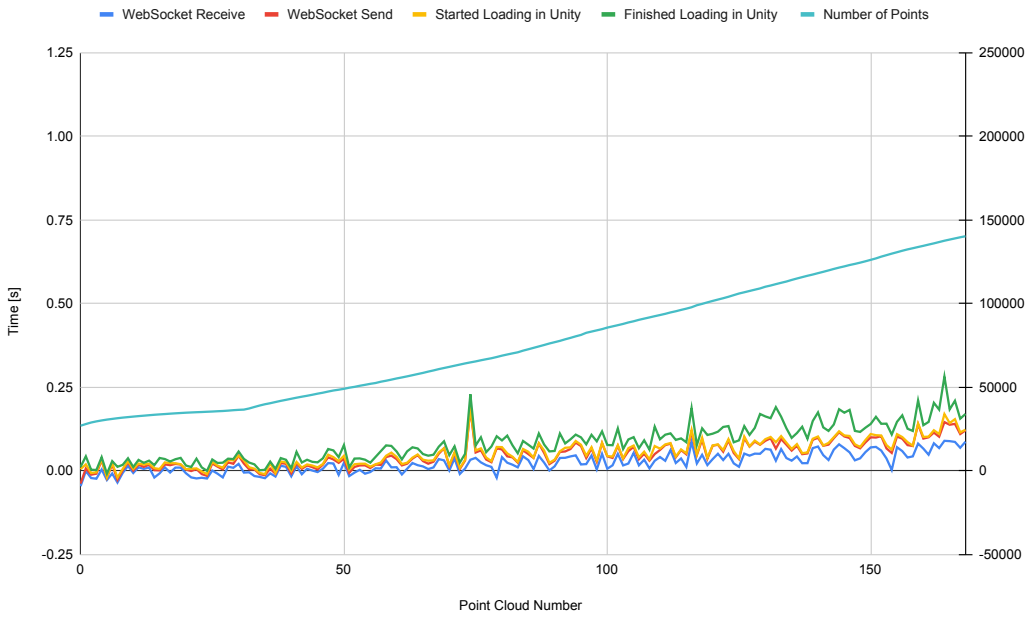

Figure 4.7: Delta time compared to RViz using 2000 points per packet.

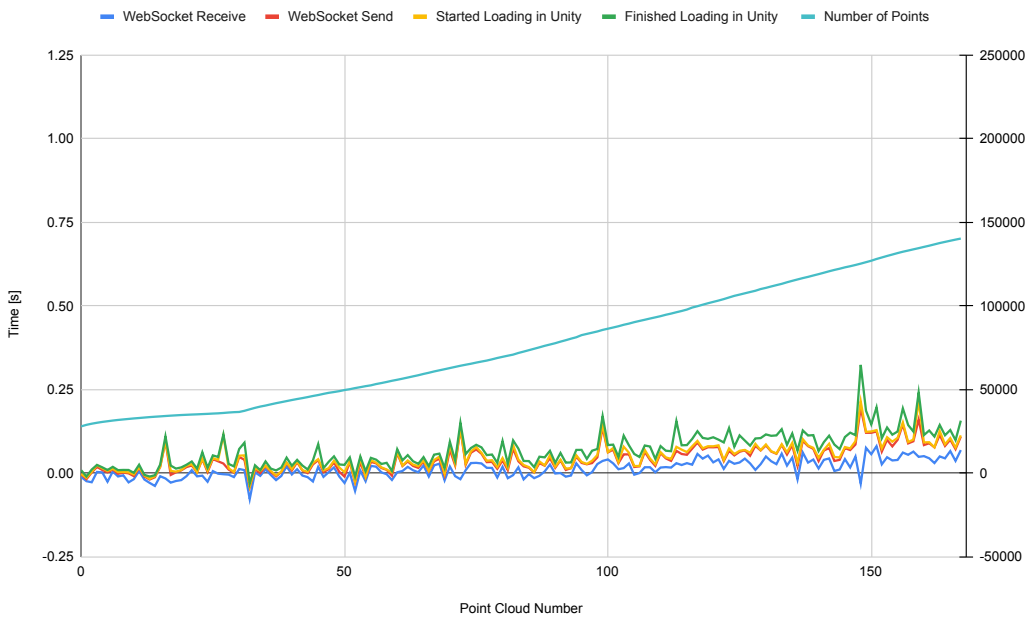

Figure 4.8: Delta time compared to RViz using 4000 points per packet.

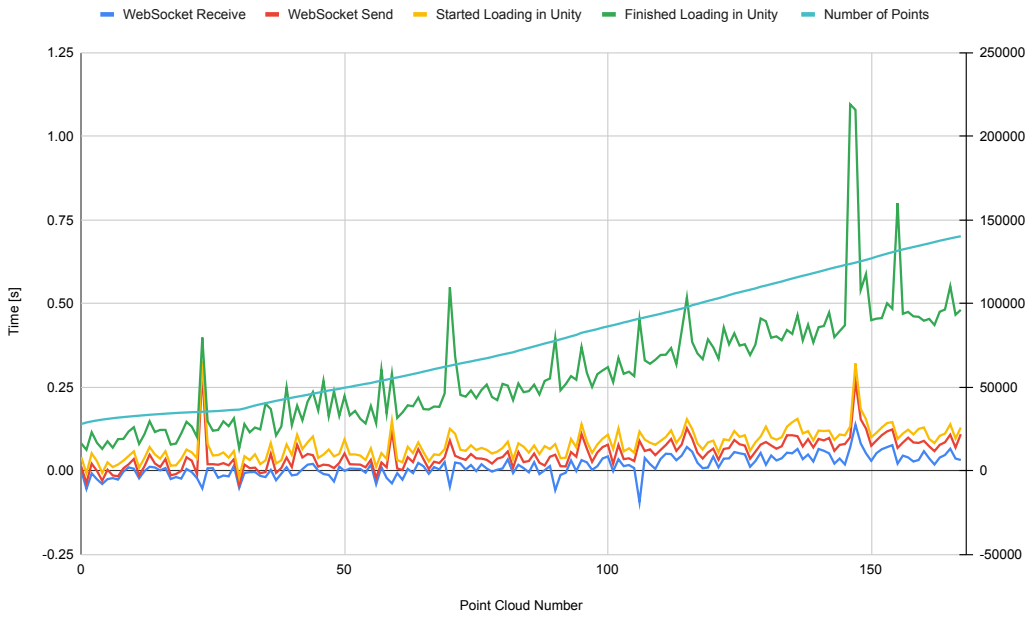

Figure 4.9: Delta time compared to RViz using 8000 points per packet.

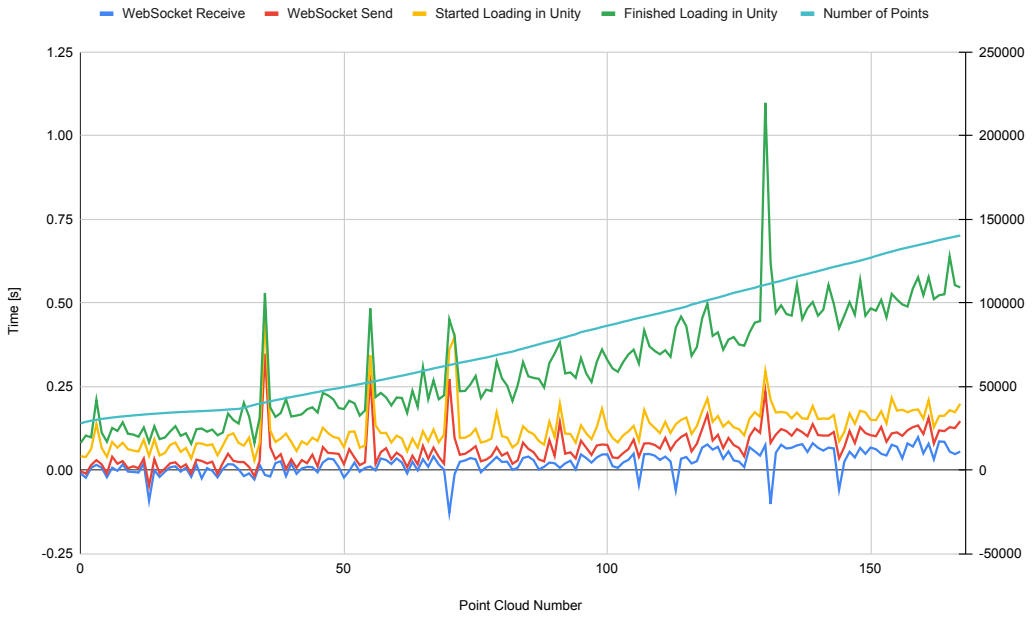

Figure 4.10: Delta time compared to RViz using 16000 points per packet.

Figure 4.11: Delta time compared to RViz using 32000 points per packet.

Figure 4.12: Delta time compared to RViz using 64000 points per packet.

Figure 4.13: Delta time compared to RViz using 150000 points per packet.

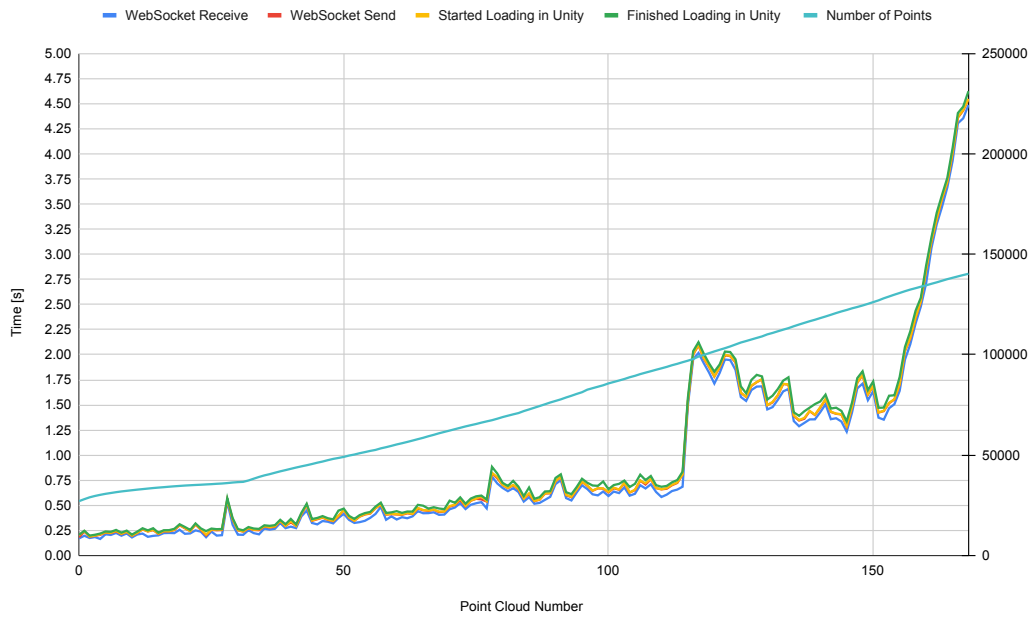

Figure 4.14: Delta time compared to RViz using 2000 points per packet - Test over WiFi.

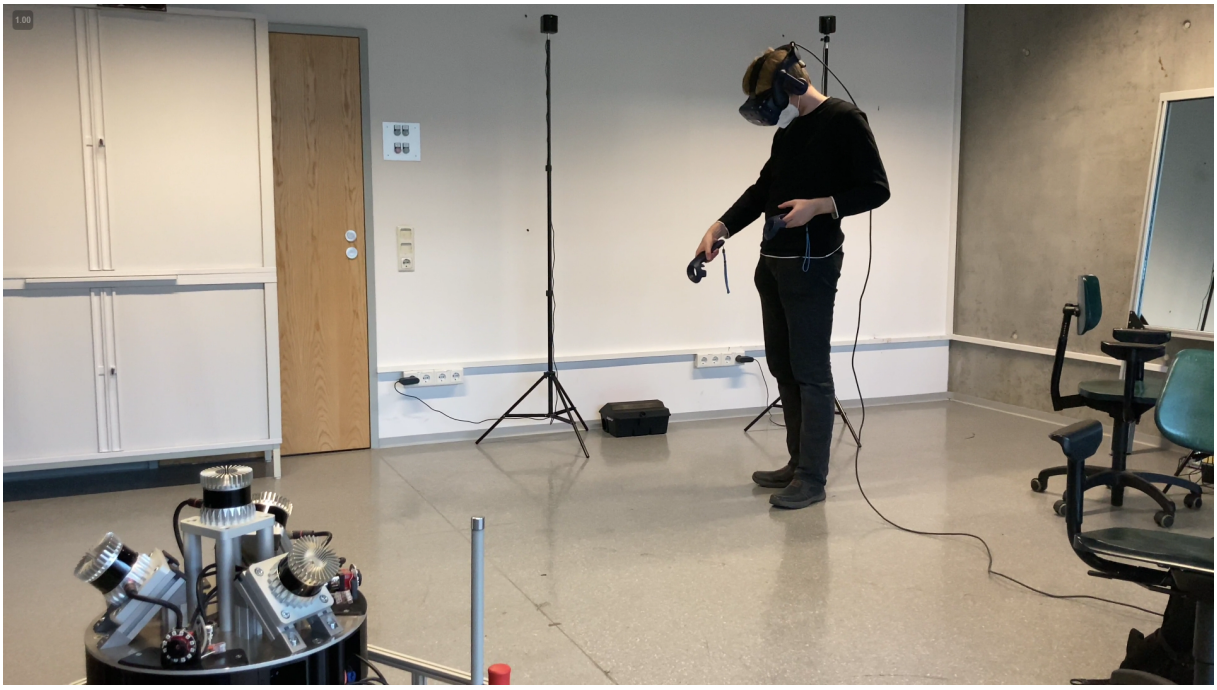

Figure 4.15: VR system and remote controlled robot.

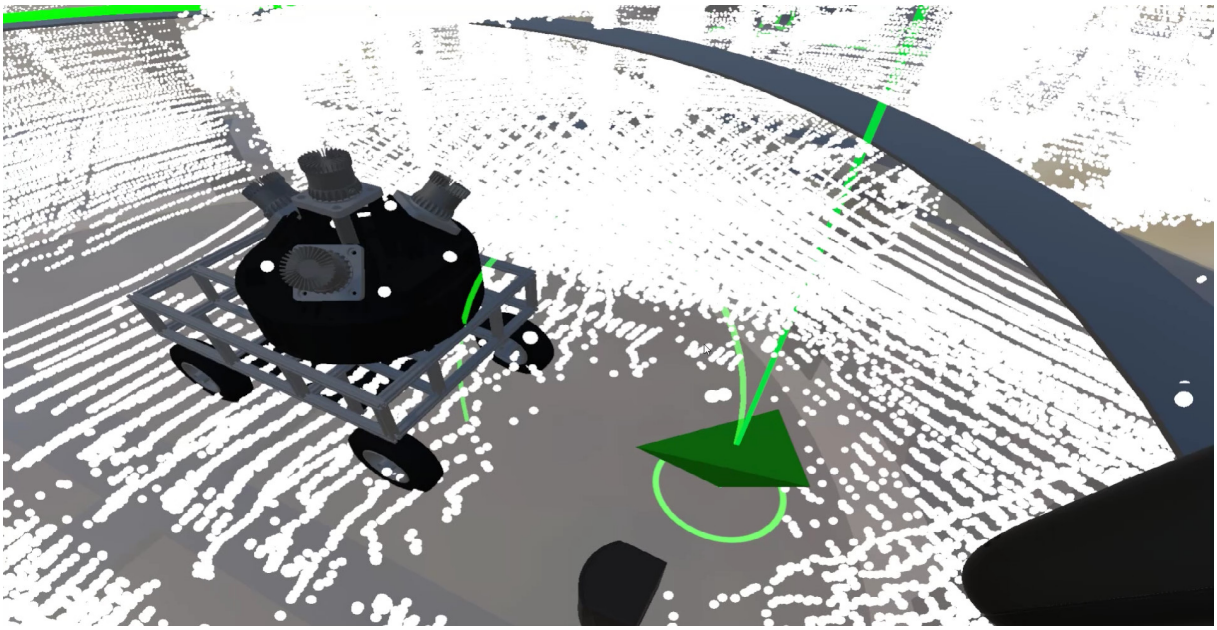

Figure 4.16: Robot model, goal pose and live point cloud in VR scene.

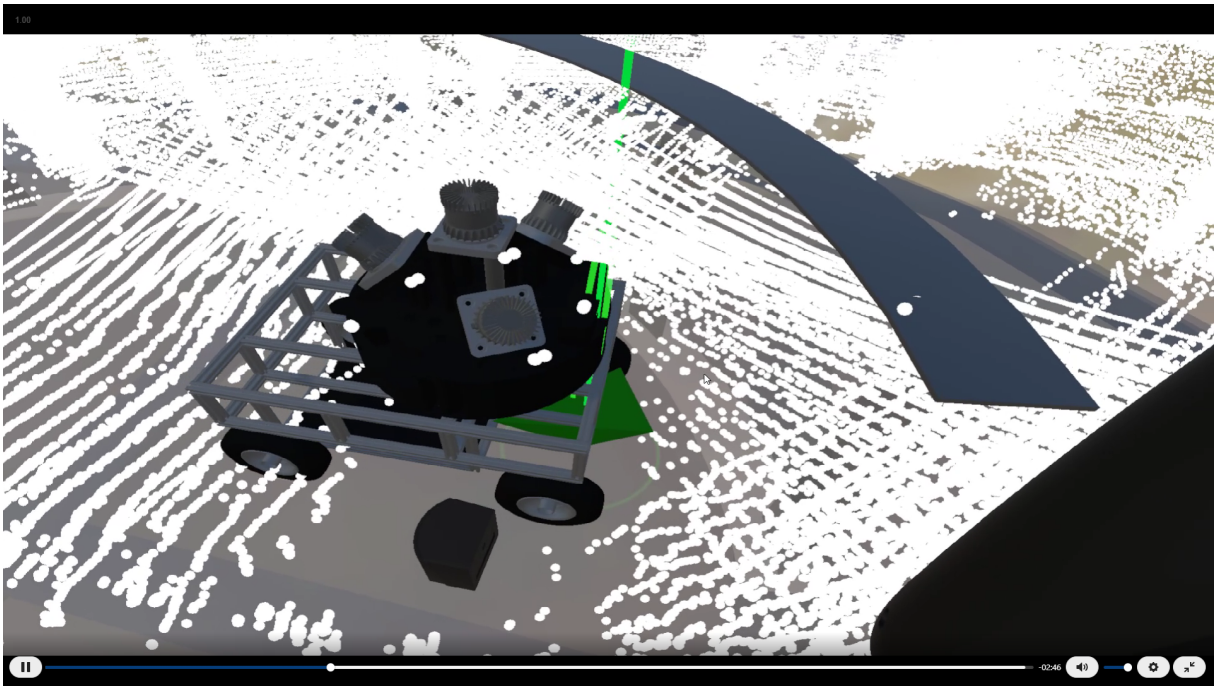

Figure 4.17: Robot reaches goal pose in VR scene.

Chapter 5

Conclusion

The main goals of this work have been achieved. The state of the art of the relevant technology and methods has been described. Point clouds are visualized in Virtual Reality using the Unity game engine. The two different sources of point cloud data have been successfully tested: Recordings using rosbags as well as live point cloud data coming from lidar scanners mounted on a mobile robot streamed over WiFi. Different forms of presenting the point cloud, such as coloring based on reflectivity or height, have been demonstrated. Additional visual elements, such as a model of the robot at the tracked position, fog and photogrammetric scans, have also been integrated and tested. The remote control in Virtual Reality of the mobile robot through the use of goal poses has been achieved. The performance of the system, especially as the latency for the point cloud streaming, has been measured and compared to the standard tool RViz.

5.1 Future work

Several aspects of this work can be improved: Especially lowering the latency for point clouds of the size in this work and larger ones makes it more suitable for real-time applications. To achieve this point clouds could be compressed with different fast algorithms. These either operate lossless or lose some accuracy. Point clouds also have the disadvantage that they are see-through. With sufficiently fast algorithms these could be converted into mesh form which does not have that problem. Other methods for remotely controlling a mobile robot could also be evaluated.

Bibliography

[1] Definition of 'robot'. Oxford English Dictionary.

[2] Fraunhofer Volksbot Ackermann. https://www.volksbot.de/ackermann.php.

[3] GLTFUtility: Simple GLTF importer for Unity. https://github.com/Siccity/GLTFUtility.

[4] Graphics Language Transmission Format. https://www.khronos.org/gltf/.

[5] JavaScript Object Notation. https://www.json.org/json-en.html.

[6] Oculus Quest 2 Specs. https://business.oculus.com/products/specs/.

[7] Optical encoders and LiDAR scanning. https://www.renishaw.de/de/optical-encoders-and-lidar-scanning--39244.

[8] Pcx - Point Cloud Importer/Renderer for Unity. https://github.com/keijiro/Pcx.

[9] rosbag: A file format in ROS for storing ROS message data. http://wiki.ros.org/rosbag.

[10] RViz: ROS 3D Robot Visualizer. http://wiki.ros.org/rviz.

[11] SteamVR unity plugin. https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647.

[12] Tastsinn-VR. https://www.interaktive-technologien.de/projekte/tastsinn-vr.

[13] Turbo colormap. https://ai.googleblog.com/2019/08/turbo-improved-rainbow-colormap-for.html.

[14] Unity (game engine). https://unity.com/.

[15] UnityGLTF: Runtime GLTF Loader for Unity3D. https://github.com/KhronosGroup/UnityGLTF.

[16] Vive Pro VR System. https://www.vive.com/eu/product/vive-pro-full-kit/.

[17] Michael Bleier, C. Almeida, António Ferreira, R. Pereira, Benjamin Matias, J.M. Almeida, John Pidgeon, Joschka van der Lucht, Klaus Schilling, Alfredo Martins, E. Silva, and Andreas Nuchter. 3d underwater mine modelling in the ¡vamos! project. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W10:39–44, 04 2019.

[18] G. Bruder, F. Steinicke, and A. Nüchter. Immersive point cloud virtual environments (poster). In *Proceedings of IEEE Symposium on 3D User Interfaces 3DUI Proceedings of IEEE Symposium on 3D User Interfaces (3DUI '14)*, pages 161–162, March 2014.

[19] Parth Rajesh Desai, Pooja Nikhil Desai, Komal Deepak Ajmera, and Khushbu Mehta. A review paper on oculus rift-a virtual reality headset, 2014.

[20] A. Melnikov I. Fette. The WebSocket Protocol. `https://datatracker.ietf.org/doc/html/rfc6455`, 2011.

All links have been checked at the time of writing on 26.06.2022.

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Würzburg, June 2022