Oculus Rift Control of a Mobile Robot

Providing a 3D Virtual Reality Visualization for Teleoperation
or
How to Enter a Robots Mind

DANIEL BUG

Master’s Thesis at CSC
Supervisor: John Folkesson
Examiner: Danica Kragic

TRITA xxx yyyy-nn
Abstract

Robots are about to make their way into society. Whether one speaks about robots as co-workers in industry, as support in hospitals, in elderly care, selfdriving cars, or smart toys, the number of robots is growing continuously. Scaled somewhere between remote control and full-autonomy, all robots require supervision in some form. This thesis connects the Oculus Rift virtual reality goggles to a mobile robot, aiming at a powerful visualization and teleoperation tool for supervision or teleassistance, with an immersive virtual reality experience. The system is tested in a user study to evaluate the human-robot interaction and obtain an intuition about the situation awareness of the participants.
Contents

1 Introduction ................................. 1
   1.1 Problem statement ...................... 5
   1.2 Contributions of the thesis .......... 5
   1.3 Visions ................................ 6
   1.4 Outline ................................ 6

2 Related Work ............................... 9
   2.1 Systems designed for teleoperation .... 9
   2.2 Human-Robot Interaction ............... 10
   2.3 Visualization ........................... 10
   2.4 Looking into the Crystal Ball ......... 11

3 System Components and Background .... 13
   3.1 Oculus Rift ............................. 13
      3.1.1 Stereo Vision ..................... 14
      3.1.2 Candy for the Eyes .............. 16
   3.2 Mobile Robot: Scitos G5 ............... 18
   3.3 Robot Operating System ............... 18
   3.4 Ogre3D Rendering Engine .............. 20
      3.4.1 Main System Components .......... 20
      3.4.2 Meshes and Textures ............. 22
      3.4.3 Rendering Pipeline .............. 22
      3.4.4 Geometry, Vertex and Fragment Shaders .... 24
   3.5 Object Oriented Input System ......... 25

4 Implementation ........................... 27
   4.1 The Fundament ......................... 27
   4.2 Building the Environment ............. 28
      4.2.1 Bandwidth considerations .......... 28
      4.2.2 Snapshot Reconstruction from Textures and Shaders .... 29
      4.2.3 Connection to ROS ................ 32
   4.3 Taking Action ........................... 34
   4.4 Overview ................................ 36
# Experimental evaluation

5.1 Experimental setup .............................................. 42
5.2 User Study ....................................................... 43
5.3 Results ............................................................. 44
  5.3.1 Measures ..................................................... 44
  5.3.2 Subjective Feedback I ....................................... 45
  5.3.3 Subjective Feedback II ...................................... 47
  5.3.4 Discussion .................................................... 47

# Summary and Conclusion 51

# Future Work 53

Bibliography 57

A 61
Acknowledgments

Credits go to the CVAP and CAS institute at KTH. In particular, I want to thank my supervisor John Folkesson for advice and guidance, as well as Mario Romero, who inspired the vision-part of this work and kept an eye on some formalities in the beginning, and Patric Jensfelt, who contributed in many discussions and progress meetings.

From the team of teaching assistants I wish to thank Rasmus Göransson for his help with my first steps with the rendering engine and shader programs, Rares A. Ambrus for his help during the user-study and, together with Nils Bore, for the help with the robot hardware and advice on ROS.

Final words of thanks go to the participants of the study for their time and constructive feedback.
Chapter 1

Introduction

This thesis project "Oculus Rift Control of a Mobile Robot" connects the Oculus Rift virtual reality goggles to a Metralabs Scitos A5 mobile robot. The project implements the initial platform to render the 3D environment in stereo and enable the user to control the robot in the real world based on its virtual representation. As an introduction, this section will familiarize the reader with the necessary terms, definitions and ideas to understand the motivation behind the project and possible development in the future.

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

- "Three Laws of Robotics", Isaac Asimov

Starting this thesis with the "Three Laws of Robotics" and thus, a link to a 1942 science-fiction novel by Asimov, the huge step is indicated that robots have taken from pure elements of fiction, to elements of modern industry environments, laboratories, hospitals and even modern homes. In that sense, robots are becoming increasingly common. As nearly every technological object tends to become smart and connected, it is necessary to differentiate between different types or ideas of robots. The definition of a robot as a smart object is rather weak, since it includes every modern cellphone or TV. Today's vehicles on the other hand, with their multiple assistance systems for driving, navigation and safety possess a certain level of autonomy, since some devices actively change parameters or decide about the application of the drivers control decision. Defining a robot as a smart autonomous
CHAPTER 1. INTRODUCTION

*machine* is sufficient to filter out most relevant devices and puts emphasis on the idea that a robot should have autonomy and mobility. Aiming at the science-fiction origin of the idea, a robot can also be defined as an artificial human being. Though there is a lot of research in this area, this definition is clearly too strict in the sense that it puts constraints on the appearance and design, which contradicts the common use of the term for machines - robots - in industry. For most robots the second definition will be the most suitable of these three, though the term remains fuzzy and depending on its context. With the ongoing advances in computational power and thus, capabilities in artificial intelligence and smart technology, the lines between definition will blur even more, as smart-phones turn to sensors and databases for connected devices, or make it possible to remotely operate arbitrary machines via app-interfaces.

**Robot Applications**

Robots are applied for various tasks in an increasing number of fields. One of the historically earliest examples is the industry sector. Production relies on large machine lines, which along with the rising availability of computational power, have become smart enough to count as robots. Even the last definition as artificial human being might be fulfilled, if a production line robot is seen as a strong automatized replacement for a worker, or just the task related body part, e.g. an arm. Usually, these robots are applied where a task is heavy, discomforting, or dangerous for a human. While they can provide high forces and moments, the level of autonomy and mobility is low, due to specialization on a single task.

In medicine, robots are used or will be used in the near future in surgery, as an interface to improve the accuracy of the surgeon, ease the control of other technological equipment, e.g. medical lasers, and eventually one day to enable specialists from the other end of the world to help patients in far away places, without the necessity of time consuming travel [2]. Related to this thesis is the field of service robots, expected to take an important role in health care, elderly care and home assistance [1]. Due to the increased need for employees in these sectors the assistance through robots will allow the personnel to focus on the care, instead of minor, yet time consuming tasks.

Transportation might change in the future to become a robotics field, where autonomous cooperative vehicles, or e.g., railways in [3], ensure an optimal traffic flow and resolve problems like traffic-jams or the security-risk arising from drivers, who follow their schedule until exhaustion.

Military is another stakeholder in robotics. In recent conflicts, the first missions with unmanned drones have been executed, for territorial investigation and as 'unmanned combat air vehicle'. Especially in warfare, the idea of sending machines instead of soldiers into hostile territory is a way to minimize the own risk of combat losses, but at the same time raises difficult ethical concerns. Well known psychological studies,
like the Milgrim experiment [4], show the willing obedience of subjects to orders given by an authority, which takes the responsibility from them. Likewise, individual bounds tend to vanish, if the contact between two subjects is depersonalized or entirely anonymized through technology. A common phenomenon that can be observed everyday in the amount of bullying, insulting, or harassing commentary in the world-wide web. Note that remotely controlled drones combine both, an operator who can shift his responsibility to the higher rank person in charge and the anonymity of the ‘unmanned air combat vehicle’ - a mixture that, in the authors opinion, in a very dangerous way, reduces the risks a technologized participant has to expect from a fight. Another area of military related research focuses on ground vehicles for exploration and transport. Such technologies can as well be applied in civil tasks, e.g. the surveillance of company property, or in public security, e.g. to scout collapsed buildings and find trapped victims. A pure example of a rescue robot is given in [5].

Entertainment is the final example mentioned here. In this category, a uniform classification of properties is impossible, since robots of almost all types are present: from robot pets [6], over remotely controlled quadrucopters, the "LEGO Mindstorms" [7] products, to instrument-playing human-like robots. It is estimated that the home-consumer market will in the future be a strong growing sector within robotics [8]. Which of the above examples of robot applications suit the "Three Laws of Robotics" is left for the reader to decide.

**Autonomy and Visualization**

Apart from the tasks they perform, robots can be classified in terms of their level of autonomy. The scale reaches from remotely controlled robots, to full autonomy up to the task level. Commonly, the level of autonomy is directly related to the application layer on which the user interacts with the machine and complexity is growing with the level of autonomy. The use cases that are placed between the named extremes will be called mixed-initiative control, i.e. the robot keeps a certain autonomy, but the user can influence decisions on several actions.

In the case of full autonomy, the robot has to be able to handle itself in all (un)foreseen situations. This implies that, technically speaking, all control loops on all system levels up to the task have to be closed by the software running the robot. For all lower levels of autonomy, the user is able to interfere with the program and hence, takes an active role in at least one system loop. A meaningful user input therefore requires knowledge about the machine state on the user side.

Vision is probably the most advanced sense of human beings and therefore a well-suited interface to the machine, if it is addressed in the right way. Vision plays a large role in human perception, since it combines spatial, brightness and color information simultaneously and has a higher reach than the sense of touch, taste or (in many cases) even hearing - things that should be utilized by a good visualiza-
CHAPTER 1. INTRODUCTION

Figure 1.1: Scale of robot autonomy levels linked to different robot-application layers (inspired from [9]). The robot posses no autonomy, if the user controls the hardware inputs. With each layer of control that is closed on the robot, the machine gains autonomy. If the user interaction is only on the level of specifying a task, almost full autonomy is reached.

...motion. While in the past visualization usually meant viewing large sets of parameters displayed on a screen and updated occasionally, the trend seems to go towards (live-)feedback from cameras and sensors mounted on the robot - [10, 11, 12, 13, 14, 15] for instance, use visual sensor feedback. Seeing the robot and its surrounding, or a virtual representation of it, is immensely improving what is called the situation awareness of the user. A number of projects research different methods to optimize the user interaction with robots. Optical feedback itself can be subdivided further by the users influence on the view point. While a mounted camera may provide a good quality video stream, its fix position can lead to occlusions and creates the need for the user to rearrange the robot position, where in many cases an adaption of the viewpoint would be sufficient. For example, the distance to an obstacle ahead is easily seen in a side-view, but can be hard to estimate from a 2D video. Recent methods take 3D measurements with lasers of kinect-like devices into account and generate 3D scenarios, which allows a free viewpoint selection.

Visualization is a powerful tool and provides, if applied appropriately, a natural interface from the machine to the user. In this context, the term 'natural' aims at the intuitive use of the interface without the necessity for extensive explanation or training. Looking at common human-to-machine interface, e.g. keyboard, mouse, joystick etc. the term natural interface extends to the avoidance of physiologically discomforting, stressing or exhausting poses. Although it is most common, the keyboard does in this definition not count as a natural interface.

On the most recent gaming conventions, known vendors as well as start-ups have announced their efforts to develop hard- and software for highly convincing virtual reality experiences. The most famous examples are the Oculus Rift Virtual Reality goggles [16] and Sonys 'Project Morpheus' [17]. Both are expected to be supported by common gaming platforms within a few years. Virtual Reality (VR) seeks to realize the above concepts of situation awareness through visual and audio stimulus and natural interfaces in order to let the user dive deep into the fictive game...
1.1. PROBLEM STATEMENT

Environments and actually embody him in the game rather than guiding an avatar in first-person, or third person view. Enthusiastic reactions by the gaming community promise large commercial interest. The acquisition of the Oculus VR start-up by Facebook in March 2014 for 2 Mrd. US Dollar indicates the high expectations on virtual reality. Google’s project ‘Tango’, which develops a smart phone able to collect and analyze 3D data, may as well illustrate the rising interest in the third spatial dimension.

1.1 Problem statement

The prognosis of an increasing number of robots in all-day situations, see [1], implies the demand for natural interfaces and intuitive ways to understand and to teach how these complex machines work. This should include an understanding for the limitations and their perception as well. Modern visualization techniques are the key to make the challenges in robotics perceivable for everyone, independent of their experience. The CVAP/CAS Institute at KTH is researching in the area of autonomous robotics as part of the STRANDS project [18]. With the Metralabs Scitos A5 robot at the institute the goal is to build a system capable to perform enduring tasks, self-organized and with a high level of autonomy. At its current state, the machine still needs supervision and eventually requests help while navigating through the institute and solving predefined tasks, usually exploration.

A simple visualization is given with the ROS program ‘rviz’. It allows to display various state variables on the robots built-in screen and accepts even a few user inputs. This thesis project aims to extend the accessibility of the robot by a virtual reality interface, with similar functionality.

1.2 Contributions of the thesis

The main idea of this work, as illustrated in Figure 1.2 is to connect Oculus Rift VR goggles to the mobile robot in order to visualize the robot’s 3D world-model and to control the robot on a task level from inside the virtual reality. The user-to-machine interface will be a standard gamepad, allowing a free navigation in the VR plus a small set of commands to interact with the robot. It will enable the user to steer the robot to points-of-interest in the VR, e.g. to guide the robot to unexplored positions, improve the visualization details, remove occlusions, or similar.

The integration of the platforms: Ogre3D engine, Oculus Rift SDK and the Robot Operating System (ROS) will form the application fundament for the visualization. Expected challenges are the decoupling between 3D data messaging and processing and the rendering pipeline. The decoupling is essential to obtain a reasonably high frame rate and hence, smooth visualization and player movement in the 3D VR on the one side and an efficient handling of the large amount of data on the other side.

The system should be modular and its components reusable, so that the development can be continued in future projects easily.
1.3 Visions

Although the ideas mentioned in this section will not be part of the actual implementation, they give an impression on possible future development, but in the same way they are an important part of the thesis’ motivation. In its basic form, the system represents a teleassistance interface for the robot. The assistance is purely in terms of navigation and mostly interesting for exploration tasks, but could be extended to other areas, too. For example, an interaction of the user and a robot avatar in the VR environment could be the interface for a virtual machine training platform.

The 3D visualization is of course not limited to the robot state and 3D snapshots of the scenery either. Possible program extensions could merge other information into the VR and enable the user to evaluate and double-check the data, which the robot uses for its algorithms. Since this option is eventually prevented by costly bandwidth requirements (considering the desirable wireless connection between robot and visualization PC), a subset of the robot data might be more interesting.

In different research projects, e.g. [19], the robot is trained to recognize and track people during his exploration. Such activities in the surrounding might be data, which is worth displaying in the VR. If activity is visualized in a smart way, the time of occurrence can be included, too, enabling the user to not only see the virtual environment as static, but as a dynamic and alive world. Considering that, despite the gathering of 3D data, the robot is moving on a 2D plane, the third dimension could be used to visualize activity and the time dimension included in it.

1.4 Outline

This report presents the project in a straight forward, standard way to the reader. Chapter 1 introduced a number of terms and definitions and sketched the idea of the work. In the following chapter, Chapter 2, the thesis will be placed in its scientific context, by looking at the related research in the fields of teleoperation, human-machine interaction (human-robot interaction) and visualization. Similar projects are referenced, briefly summarized and finally, some outlook is given, on what might be part of the future of the project.
1.4. OUTLINE

Afterwards, Chapter 3 and Chapter 4 cover the background information, introduce the system components and show how they are integrated into the implementation. Especially the fourth chapter reveals several tricks in the implementation, which are crucial in order to obtain satisfying results.

Chapter 5 contains a description of the experiment, which was performed to evaluate the tool. Detailed information is given on the setup, the user study, the results and their discussion.

The report is concluded with a summary and conclusion in Chapter 6 and concrete suggestions for future projects are given in Chapter 7.
Chapter 2

Related Work

This chapter will give the reader the context to understand and evaluate the connection and relevance of the different aspects of this thesis. Since robots are becoming more and more common, the background knowledge of operators changes. The robot leaves the computer scientific laboratories and is integrated into the regular work environment of people who are non-specialists in the field of robotics. It is therefore interesting to look at several public-scientific sources, as well.

2.1 Systems designed for teleoperation

In [15], Okura et al. propose a system for the teleoperation of a mobile robot via augmented reality, which is very close to the idea of this thesis’ work. It features a head-mounted display for visualization, a gamepad for control and a mobile robot, which is equipped with depths cameras and one omnidirectional camera. The paper differs in the degree of autonomy of the system. The robot is strictly user controlled and designed for teleoperation, while for this thesis, the (long-term) goal is to keep a high autonomy in the navigation and planning of the robot. Martins and Ventura use a similar system in [14], where the head movement is used as control input to the robot. Ferland et al. apply a 3D laser scanner in [20] and merge scene snapshots into a 3D projection on screen, to enhance the operators situation awareness. The paper includes a short study with 13 participants for evaluation.

A historical perspective on the development of exploration robots, such as the mars rovers, is given by Nguyen et al. in [21]. A focus is kept on the benefits of visualization tools and the advantage of user interaction on a task level rather than direct control of the robot. The authors point to the constraints on the achievable performance in a directly controlled application due to communication delays. For extraterrestrial applications, a direct control-link is usually even impossible.

Further material on the teleoperation of vehicles was published by Fournier et al. in [13], Zhang et al. in [12] and Alberts in [11]. All three emphasize the importance of visualization as the main factor for situation awareness. Additionally, [13] and
CHAPTER 2. RELATED WORK

[11] give examples of how a good visualization can be achieved. While in [13] an immersive augmented reality chamber (CAVE) is used, different rendering methods, i.e. point-clouds, triangulation and quad visualization, are compared in [11]. In [22] by Huber et al., the visualization for remote control of a vehicle is evaluated in terms of its real-time capability.

In the paper [10] by Lera et. al., a system for teleoperation is presented, which uses augmented reality, i.e., including additional helpful information into the VR, to improve the orientation and user-comfort. Their evaluation yields that the supportive landmarks and hint indeed easen the process of finding through a test-parkour.

2.2 Human-Robot Interaction

Another important sector in robotics is human health care, especially for older people and people suffering from physical disorders. Although functionality for teleoperation is still very interesting, e.g. to guide the robot through a difficult situation, improve the clients security, or simply as an interface to establish a telepresence contact to a doctor or nurse, as suggested by Hashimoto et al. in [23], the focus changes towards the question how to provide comfortable and safe human-robot interfaces.

As pointed out earlier, robots are on their way into society. For applications in surveillance, military, or research, it might be feasible to assign the operation of the robots to trained employees, but the more common robots become in civil applications, for example rescue, health-care, medicine, or even entertainment, the stronger the need for an intuitive understanding of the machine’s functionality and limitation. In [24], Anderson et al. propose an intelligent co-pilot as an architecture for embedded security functionality in cars, which keeps the restrictions to the user-comfort to a minimum. Their aim is to close the gap between fully human-operated vehicles, in which the security systems are merely visual or acoustic warnings, and fully autonomous vehicles, which take away the freedom of the driver. The main idea is to let the security system define safe corridors, in which the user may act freely. In [25] by Valli, an appeal is made to scientists, programmers and developers to think more about the way people actually interact (with each other) and let their observations guide the design of human-machine interfaces. The study [26] by Goodrich on human-robot interaction, gives a broad overview on robot application, interfaces, requirements and current challenges in the field of HRI.

2.3 Visualization

One main result of the research in the field of HRI is that visualization significantly improves the comfort during the operation and may influence even safety and reliability. In [27] by Tvaryanas and Thompson, an evaluation for the case of remotely piloted aircrafts yielded that 57% of all mishaps (out of 95 recorded events), could be tracked back to a lack in situation awareness and in multiple cases a misinterpretation of the interface lead caused the trouble. Thus, a large part of teleoperation is
2.4. LOOKING INTO THE CRYSTAL BALL

about the visualization of the environment. However, a high-quality visualization in real-time implies an enormous bandwidth that is occupied by the data stream. Different concepts for the compression of the information can be imagined, e.g. the use of video-codecs, exclusion of additional data, etc. Usually in the literature, the 3D data is represented as a point-cloud, a collection of 3D coordinates, associated with different parameters, e.g. color values and labels. A remaining degree-of-freedom is, if pre- or post-processed data is streamed from the robot to the rendering PC, which will at the same time influence the distribution of the calculative load. In [28] by Schnabel et al., an efficient RANSAC implementation to compress point-cloud data into shapes is proposed and evaluated. Based on local variation estimation, Pauly et al. describe methods in [29] to reduce the number of points along with minimal changes to the model quality. Applying similar methods for preprocessing means to sort out irrelevant points and transmit only the important information through the channel. Especially for flat, rectangular surfaces, this method bears an enormous potential to remove redundant traffic.

2.4 Looking into the Crystal Ball

The work presented in this section has a less obvious relation to the application, but it contains interesting knowledge which can be seen as a vision for the system.

Introducing a time dimension. Ivanov et al. introduce a security system that combines simple motion sensors and few cameras for activity measurement in [30]. An advantage of decreasing the number of cameras, is a lower level of intrusion into peoples privacy. It is pointed out how this more anonymous form of surveillance provides a capable tool for human behavior analysis. In [31] by Romero et al., a system for video analysis through low-level computer vision techniques is presented. The activity measurement by motion sensors is replaced by computer vision tools on the video signals. Through generating heat maps from the motion in the scenery, the data is anonymized, while all necessary information to observe behavior remains visible. This allows a supervising person to identify extraordinary and relevant activities very easily, without actively watching the whole video stream. Romero et al. discuss the system in action in [32], in a study on human behavior visualization. A very good impression on the power of these visualization and analysis techniques is presented in the TED talk by Roy [33], in which similar methods are used to study the development of speech. In the project’s course, over 200 TB of video data were recorded. It is emphasized that this amount of data could not be analyzed appropriately without the application of visualization. Furthermore, Roy demonstrates how to use the method to correlate discussions in social media to topics presented in television. The analysis leads to the identification of social role-prototypes and information feedback loops. Related scientific work has been published by Roy, De Camp and Kubat in [34, 35, 36].
How is this related to the topic of robotics? All three authors confirm the power of visualization techniques. Behind each case is the idea to introduce a representation of time and changes over time, resulting in the benefit to see the history of the scenario and being able to reflect on it. Measuring the activity, the changes, in a mobile robot’s world provides new interesting options as well. The direct transfer of the named articles to the case of a mobile robot is to be able to analyze the behavior of the robot in the environment. In this case, the robot is mostly just a mobile camera system, but note that identifying other activities carries the possibility to display human-robot interaction, too. For an operator, an intuitive visualization of the history and surrounding activity could improve the understanding of the decision making and navigation of the machine and thus, simplify the supervision. For the navigation task, measuring activities may help to identify moving obstacles and decrease the need for supervision, if a measured activity is mapped to proper strategies. Finally, a visualization of which data is currently evaluated can help to build an intuition on how a robot navigates through its present, or rather, what amount of data actually incorporates the term ‘present’ for a robot. This, of course, already is an outlook into the possible future of this project.
Chapter 3

System Components and Background

It is time to introduce the essential system components in details and explain the relevant concepts related to it. After this chapter the reader should be able to put many pieces of the puzzle together and figure out how the program works. Starting off with the Oculus Rift itself, the topic of stereo vision is discussed in a more general way, introducing the concept of (stereo)projection and related phenomena like occlusion. In addition to that, multiple methods to generate 3D experiences are explained. Another focus is on the question what factors influence the quality of the 3D experience for the user. Afterwards, the mobile robot is introduced and the Robot Operating System (ROS) is explained. The final sections deal with the Ogre3D engine, explaining the concepts of its most important components (in relation to the project), the rendering pipeline and 3D shaders. The details of the implementation, tricks and a few links to the program code will be shown in the next chapter.

3.1 Oculus Rift

The 'Oculus VR' company started in 2012 as a crowd-funding project with the vision to build "immersive virtual reality technology that’s wearable and affordable" [16]. Former virtual reality goggles have not been able to achieve satisfying performance for gaming purposes due to technical limitations, such as low refresh rates for displays, small fields of vision and high latency, which prevents the user from diving into the 3D world and furthermore, a quite expensive equipment (see [37] for a summary of these issues). The experience is often described as looking on a far distant screen, with clearly visible black borders and lacks between actual movement and rendering reaction. The Oculus Rift solves some of these problems simply through the technological advances of the past years, e.g., the increased refresh rate and resolution of modern light-weight mobile displays, but as well applies several tricks to overcome the odds. To fill out the users entire field of vision, the goggles feature a set of lenses to map the viewport to each eye and bring the user close to the screen and achieve a vision angle between $105^\circ$ – $115^\circ$. A fragment shader program is used to pre-distort the views, in order to cancel out the lens distortion...
CHAPTER 3. SYSTEM COMPONENTS AND BACKGROUND

(for details/illustration see [38]). The problem of latency relates to the delays between the measurements of the motion sensors which track the head position and the rendering update, which can be perceived by the user in the order of a few hundred milliseconds. Predicting the head movement from the past measurements with polynomial extrapolation, helps to decrease the influence of these delays.

Overall, the Oculus Rift manages to provide an immersive virtual reality experience, with a large field of view and sufficient image and motion-tracking quality at a reasonable price. In this project, the first development kit is applied and hence, for the future, several issues in terms of latency and image resolution are expected to be improved along with the development of the device. The enthusiastic reaction in media and the gaming community are an important reason, why the Oculus Rift was chosen as the virtual reality interface.

This section focuses on the concepts related to stereo vision and projection as well as it explains the lens distortions and movement prediction, which are the main concepts for the immersive experience. For algorithmic details and concrete examples see the Oculus Rift SDK Documentation [39]. Information on the implementation in this project is given in chapter 4.

3.1.1 Stereo Vision

Vision is maybe the most advanced sense of human beings. At around 24 frames per second (fps) the brain fuses the information of two highly developed camera-like sensors into one 3D model of the world. A single eye alone cannot provide the 3D information with the same accuracy, which can easily be explained by looking at a simple camera model, illustrated in Figure 3.1.

From the geometry, the equation

\[
\begin{bmatrix}
    x \\
    y
\end{bmatrix}_{2D} = \begin{bmatrix}
    s & 0 & 0 \\
    0 & s & 0
\end{bmatrix} \cdot \begin{bmatrix}
    X \\
    Y \\
    Z
\end{bmatrix}_{3D}, \text{ with } s = \frac{f}{Z},
\]

follows. Where \((x, y)\) are the coordinates in the image, \((X, Y, Z)\) are the real-world coordinates and \(f\) and \(Z\) are the camera’s focal length and the distance to the object. It is quite easy to see that for a single camera system the scaling factor is not unique and each pair \((x, y)\) of image coordinates could result from a line of 3D coordinates. Thus, with a single camera, the depth information is lost and a 3D reconstruction of the scenery is impossible. In order to get the information about the depth, the vision system has to be equipped with at least one more camera. Given the relative orientation and position of the two cameras, the 3D coordinate can be computed as the intersection of the back-projected lines, for all points that appear in both images. It is even possible to estimate the homography, i.e. the relation between the two cameras by finding and comparing image features in both views to each other. This process is not even bound to stereo vision. For the sake of completeness, it is mentioned that the method can be extended to multi-camera systems, which is nowadays a common method for motion capturing in movies, games and sports.
3.1. OCULUS RIFT

Figure 3.1: Simple pin-hole camera. Due to the projection from a single point the object size is ambivalent.

Systems with a low number of cameras suffer from occlusion effects, as shown in Figure 3.2.

Figure 3.2: Occlusion in the case of stereo cameras.

Here, two cameras are looking at the same object. Behind the object is a zone, which appears in none of the two camera images and two larger zones that are seen by just one of the cameras. The first case is the case of occlusion and no visual information can be gathered unless the setup is changed, while for the second case only 2D data is recorded. Thus, in the 3D reconstructed view these areas can not be filled with
meaningful data either, since the depth remains unknown. A simple rule of thumb for an eye-like stereo camera setup would phrase this observation as "Close objects cast huge shadows (in the 3D projection)". These 'shadows', or blank spaces, are not seen from the angle of the cameras, but appear from all other viewing angles. Stereo cameras are not the only possibility to obtain 3D data. Other methods include laser scanners and infrared measurements, as in kinect-like motion sensors. For both types, the depth is measured directly from the laser or infrared based on a change of characteristics between a sent signal pulse and its reflection. Related to the different method, the characteristics in terms of occlusion will be slightly different.

3.1.2 Candy for the Eyes

In the following, the focus is on the question of how to get the visual information to the eyes. Many decisions and actions in everyday life are in fact influenced by visual excitement, from traffic lights to advertisement and from the bestseller book to the daily news. Modern computers are used as powerful and capable visualization devices and display content on all kinds of two-dimensional screens. Classical computer games, CAD tools and other 3D applications simulate the 3D view using a single projection, i.e., rendering the view for a single camera. Due to the projective view, lighting effects, shading, etc., the user still gets a good intuition about the spatial relations between objects, but the plasticity of the 3D world is lost. If a strong and realistic impression of the 3D environment is desired, the scene has to be projected for both eyes individually. The key question is how the two views will reach their respective eye, without being perceived by the other one. There are different approaches to handle this task, e.g.,

- color or polarization filters
- shutter glasses
- head mounted displays

If color or polarization filters are applied, the two views are merged on the same screen, but are coded differently, see Figure 3.3. Color filters usually use a red and blue filter to distinguish between the eyes, while the second method relies on orthogonal waveform polarization. The user has to wear goggles with the appropriate filters in order to experience the effect. Color filters are extremely cheap, but will alter the original scene colors as well, which makes them less interesting for many commercial purposes. Polarization filters are today’s standard approach for 3D movies in cinema, due to relatively low cost and better color preservation.

Shutter glasses use a different method. Instead of merging and rendering the two views in a single frame, i.e., separating the views spatially and 'coding/coloring' them, the views are rendered and shown alternatingly. The shutter glasses have to block and clear the path to the respective eye synchronously. Besides the synchronization between rendering PC and the shutters, this method requires a very high
3.1. OCULUS RIFT

Figure 3.3: Comparison of stereo projection methods. a) Red/blue color filtering, b) head-mounted displays.

display refresh rate to work satisfyingly.

Finally, the approach of head mounted displays (HMD) is to move the screen directly in front of the eyes and render the views in complete spatial separation, like in Figure 3.3b. The figure shows the views including the pre-distortion for the lenses. The quality of the 3D experience in the HMD method is influenced by a large number of factors.

- Since each eye has its own viewport, the eye-to-eye distance needs to be adjusted for the user. Even small deviations can make the scene look wrong especially in combination with other misalignments.

- It is recommended to use a head-model, which takes care of the fact, that the human head moves around the neck and not its center. Yaw, pitch and roll angles need to be interpreted in terms of this off-centric movement, in order to create a realistic illusion.

- The lens distortion observed by the user is dependent on the eye-to-screen distance. If the distortion is not canceled out correctly, the result will be a tunnel-vision with a relatively focused center, but weirdly bent lines at the outer areas of the view. Whether this is to describe as ‘drunk’ vision or not, it can easily make the user feel sick.

- A factor that influences the latency is the order of the polynomial prediction for the head movement. Although this is typically not a parameter selected for or by the user, it should be mentioned that an observable latency can completely negate the experience and, together with remaining lens distortions, is likely to be the main reason for nausea.

The first two points, the eye-to-eye distance and the head-model, name parameters that have to match physical measures of the user. The required accuracy depends on the sensibility of the user towards the respective disturbances, well adjusted
parameters usually extend the time that can be spent in the VR before feeling dizzy.

In order to cancel out the lens effects, each viewport has to be pre-distorted in a so-called barrel-distortion, which essentially is a radial scaling, where the scaling factor is determined by a 6th order polynomial in $r$. According to [38], the formulation in polar coordinates is

$$(r, \phi) \to (f(r)r, \phi), \text{ with } f(r) = k_0 + k_1r + k_2r^3 + k_3r^6,$$

(3.1)

where the $k_i$ parametrize the distortion. An adjustment can be necessary, since the eye-to-screen distance and the eye itself vary for each user.

The effect of motion sickness caused by virtual reality is assumed to stem from mismatches between the visual information shown in the HMD and other perceptions, e.g., the sense of balance in the inner ear. This example in particular is the result of latency, i.e., delays from the motion-measurement to the end rendering chain. While the sense of balance reports a head movement, the visual input reacts delayed. One assumption is that the body interprets these delays as a result of poisoning (e.g., due to the consumption of bad food), which leads to the feeling of sickness and eventually even the reaction to throw up. The delays are a technical limitation and even if the computational power and bandwidth increases, delays will always be present. The limitation can however be reduced mathematically by predicting the head-movement into the future to cancel out the delay. For the sensor fusion and angle estimation, a 5th order polynomial-prediction is applied.

3.2 Mobile Robot: Scitos G5

The mobile robot is the second important technical component in the project. As explained, it is supposed to run in an exploration mode to scout its environment and provide scene snapshots to fill the VR. The robot is therefore equipped with Asus Xtion kinect-like camera sensors. The heart of the robot is an embedded PC with an Intel Core 2 Duo Mobile processor. The machine runs the Robot Operating System (ROS) [40], which is a modular operating platform for robotic applications and includes a huge variety of packages that handle sensor data, task management and communication. For the communication a wireless router is set up locally in the lab, which connects remotely (via wifi) to the robot.

3.3 Robot Operating System

ROS is a large open source tool-set and collection of general purpose libraries to operate robots. Since the tasks and possible technical equipment of a robot is quite unpredictable the system aims at being highly adaptable. Thus, ROS is organized very modular in packages that can be loaded (even during run-time) and are organized by a central root element, the roscore. Every program is connected as a node or nodelet to the core. All communication between nodes, even the information flow
3.3. ROBOT OPERATING SYSTEM

from sensor input, over different processing steps, to some output is registered in the
roscore in terms of so-called ‘topics’ and can be modified by a publisher (providing
topic content), or a subscriber (processing topic content). The system works ‘on
demand’, meaning that topics will be advertised all time, but only processed and
published, if a subscriber exists. It is in the nature of this setup that the nodes work
as asynchronous tasks in individual threads and make indeed good use of modern
multicore technology. ROS provides library functions for message synchronization
and buffering as well as many utilities to access important information, e.g., trans-
formations between local coordinate systems. Furthermore, many utility functions
allow easy integration of other libraries, e.g., the point-cloud library [41] or OpenCV
[42].

Figure 3.4: Incomplete sketch of ROS elements. All program nodes are registered at
the roscore and may utilize nodelets, running e.g. conversion algorithms to support
their task. Network connections are managed by the system automatically.

Another advantage, particularly for this project, is that ROS automatically wraps
network connections. All it requires, is to tell the nodes their own host-name and
the address of the roscore by setting the respective environmental variables and the
connection will automatically be set up, whenever a subscription requires data over
the network.

Take for instance the software setup in Figure 3.4 where four nodes are loaded and
connected to the roscore. Out of these nodes, three are started on the robot itself,
organizing the sensing, navigation and action (motor control). For the purpose
of supervision a visualization node is started on a external PC and connects to the roscore via a wireless link. In order to obtain the camera stream from the robot sensor, the visualization requests the images in a compressed form, which requires the sensor-node to fire up the respective compression nodelets. It shall be mentioned that for each machine, the configuration can be stated automatically by xml-structured launch files, which makes it a lot easier to manage and test different configurations. Even nodelets can be run from the launch files, yet many extensions like the image_transport package, provide subscriber types, which support utilities like compression automatically.

A good place to get started with ROS is the online documentation on [40], which contains many tutorials, descriptions and details to the various packages. Sometimes however, these tutorials can be a bit of a puzzle, since the pieces of information are spread on different pages or paragraphs. So be prepared to research the package description, together with the tutorial page and eventually the code documentation to fully understand the capability of the system.

3.4 Ogre3D Rendering Engine

Along with the Oculus Rift a software development kit (OculusSDK, [38]) is provided, which takes care of the handling of the sensors and the motion-prediction. However, the rendering process is intentionally not touched by the OculusSDK. The choice of a rendering engine is left to the developer and the SDK is designed to be easily integrated into any environment. For this project the Ogre3D engine was chosen, mostly because its community already provides libraries supporting the Oculus Rift and wrapping the SDK in a very handy way.

Ogre3D is an open-source, cross-platform rendering engine. It aims a productivity and the independence from 3D implementations, such as OpenGL of Direct3D, and provides numerous features for scene creation, material and texture management, animations and special effects. Multiple times the Ogre3D project has been supported by the 'Google Summer of Code’ and benefits from a strong community.

3.4.1 Main System Components

Generally, rendering engines are wrappers to provide a high level access to the graphic card acceleration for 3D applications. Ogre addresses this task in a strictly object oriented fashion, i.e. providing interfaces, abstract classes and parent classes for the high-level concepts and leaving the hardware-specific details to the instantiated dynamic class types, which are chosen for the executing machine. Take for example the Ogre::Texture class, which allows you to set up all relevant properties of a texture, but delegates the 3D implementation specific work to Ogre::D3D11Texture, Ogre::D3D9Texture, Ogre::GL3PlusTexture, or Ogre::GL3Texture. The core elements of Ogre are shown in Figure 3.5.

Root is by name the initializing instance of the rendering engine. During the initialization the root will detect and set the 3D implementation and prepare the
3.4. OGRE3D RENDERING ENGINE

Figure 3.5: Ogre UML overview, taken from the Ogre3D manual.

corresponding hardware buffers and instantiate the window system. Apart from a few parameters, there is not much to set here. From the programmers point of view, the Resource and Scene Managers are more interesting. The resource system will handle every component the scenery will be constructed with. Metaphorically, it defines what a brick looks like, whereas the Scene Manager builds the house. The resource system instantiates each one unique manager for textures, materials, shader programs, etc., and makes it available as static class members according to the singleton pattern. Once the system is initialized, the singletons can be accessed easily from anywhere in the program. In contrast to that, the SceneManager is not unique, since there might be different scenes available, e.g., different game levels or areas. This manager holds the information on what actually is displayed. It can instantiate entities, which are the basic movable objects in Ogre and other more specialized objects, like lights and cameras.

Ogre utilizes a variety of scripts to load plugins, e.g., support for different shader languages, and media (i.e., the resources), which allows to specify and change components quite comfortably outside the program and more importantly, without the necessity to recompile after each change.
3.4.2 Meshes and Textures

In this section the use of meshes and textures is explained in more detail and a few common terms in rendering are defined, before the rendering pipeline is treated in the next section.

Rendering is about forming a 2D picture, displayed on a screen, from the 3D model data in the scene and as mentioned, the 3D scene can be subdivided into object prototypes. Even from simple image processing tool the term *pixel* is well-known. It is an abbreviation for picture-element and holds the color information for a certain position in an image.

The equivalent for 3D is a *vertex* (pl. *vertices*). A vertex holds rendering information for a point in the 3D space. This rendering information can be a color, interconnections with other vertices, or coordinates of an image that is used for coloring.

The setup of vertices and interconnections is called *geometry* and images which are linked to a geometry are called *textures*. If the setup consists of vertices without any interconnection, the rendering result is a point-cloud. In a distant view, a dense point-cloud can appear as a solid object, but in a close view, the single points will separate. By telling the engine, how to interconnect the vertices, it is able to fill the space in between, either by interpolating the specified color values of the neighboring vertices, or by interpolation from the linked texture coordinates and the rendering results in a surface instead - even from a close view. Although today some engines support other geometric primitives, the most common 3D geometry is a triangular mesh and all other polygon structures can be decomposed into a triangular mesh.

Basically, the system works with nothing else than a set of 3D points and a strategy, what to do with the space in between. In case of color-specification, a nearest neighbor interpolation based on the vertices is used, while in case of the textures, the interpolation works with the nearest texture-element (*texel*). This way of linking image coordinates to vertices is called *uv-texturing*, where the pair \((u,v)\) indicates that 2D image coordinate are used. Figure 3.6 illustrates the coloring. In the upper row, the three vertices store color values and the engine interpolates smoothly in the area specified by their connections. In the lower row, the texturing for the robot avatar is shown. The grid is formed by rectangles (which can easily be decomposed into two triangles each) and every vertex has an according texel in the image in the middle. The texturing result is shown in the picture at the right.

To recap, 3D meshes and textures are important resources for the rendering and form the base for the scene content. In Ogre3D, the resources are organized by a management system. In the scene manager, the scene graph is formed and connects various instances of the resources in a hierarchical way. The scene graph is then the input to the rendering pipeline.

3.4.3 Rendering Pipeline

The rendering pipeline is an abstract concept of how to process the 3D data in the scene in order to display it on screen. Its implementation is the heart of every 3D
3.4. OGRE3D RENDERING ENGINE

Figure 3.6: 3D surface representation. Upper row: color interpolation from the corner-vertices. Lower row: uv-texturing example to form a robot avatar with a 3D mesh and texture image.

engine. From the bird’s-eye view, the pipeline has three main stages: application, geometry and rasterization. Figure 3.7 illustrates the stages of the pipeline. The terminology and explanations are a summary of [43], Chapter 2.

The *application* stage is where the programmer has full control over the engine. In a computer game for example, this stage would contain the game AI, processing of the user input and according adaptations of scene content or properties, like detail levels or textures. A general sub-structure of this step cannot be given, since it is entirely up to the developer to design the application, which could include e.g., multi-threaded tasks for speedup, or simply single-threaded sub-pipelines. The outcome of the application step specifies whatever has to be displayed and in what way and is passed on to the geometry stage.

In the geometry stage, the settings from the application stage are applied to the scene elements. This includes vertex and fragment operations which are processed in the sub-pipeline: scene transform, lighting, projection, clipping and screen mapping. In
the first step, the scene elements are transformed from their local coordinates into world coordinates and then into the camera coordinates, which already equals the user’s viewpoint. The transformation is calculated along the scene graph, from the objects to the root and then to the camera nodes. Within this view, lighting and shadow effects are calculated, usually based on the vertex normals of an object.

The projection step maps the camera view into a unit-cube according to the chosen projection type, i.e. parallel or perspective projection. In the normalized space, the clipping step discards everything outside of the user’s view. If objects at the sides are only partially inside the view, this step replaces the unseen vertices with characteristic vertices at the border. In the last step of the geometry stage, the unit-cube is mapped into the screen coordinates, e.g., to fit the screen size (in full-screen mode) or the window size.

Finally, the rasterization stage is generating the 2D image, which is displayed on the screen, i.e., finding the correct color for each pixel in the screen raster. In addition to the calculated object colors, lighting and shading effects, this step performs the look-up of texture elements and eventually interpolates missing values. Based on a Z-buffer (Depth-buffer) algorithm, like for instance \[11\], the rasterizer as well considers occlusion of scene elements. The screen image is double-buffered to hide the manipulations from the user and perform them in the background buffer.

Geometry and rasterizer stage make both use of the accelerated processing on the GPU. In contrast to the CPU, which has a low number of general purpose processing cores, the GPU has a huge number of very specialized cores. Transformations, projections and clipping tests are per-vertex operations, while texture look-ups and interpolation are per-fragment operations, which can usually be parallelized without further constraints and are therefore predestined to make use of this acceleration.

### 3.4.4 Geometry, Vertex and Fragment Shaders

It has been mentioned that Ogre utilizes a number of scripts to outsource graphical effects from the application. While some of these scripts (i.e., material scripts and compositors) are engine specific, the class of geometry scripts, vertex shaders and fragment shaders are a more general matter.

Shader scripts contain a collection of functions that operate on geometry, vertex, or fragment data respectively and can be compiled during start-up and executed on the GPU. They are commonly applied to generate post-rendering effects, i.e. 'last-minute' application of additional highlighting/shading textures, vertex bending effects (object deformations) or transparency effects. Modern shaders allow data crossovers, in the sense that a fragment shader can make use of vertex data, for instance. The bridge to the application are the textures and material parameters, which can be set from within the 3D engine. OpenGL, Direct3D and global vendors like nvidia have created several languages to define shader programs. In this project, nvidia's shader language 'cg' is used to speed up the reconstruction of 3D content from image data. The code is written in a C like fashion and with \[15\] a good online resource exists to step into the details.
3.5 Object Oriented Input System

The Object Oriented Input System (OIS) is an open-source cross-platform library to access mouse, keyboard, joystick and gamepad devices for gaming purposes. It provides an easy to use interface, featuring event-callbacks and status-requests and wraps the driver handling for the software developer. Most popular input devices from all commonly known manufacturers are supported. The role of OIS will be a minor one, though, since the gamepad can, for the course of this project, be more efficiently accessed via ROS. In fact, to reuse an existing teleoperation node for the robot, the ROS message system has to be used, leaving keyboard and mouse input for the OIS library.
Chapter 4

Implementation

This project’s purpose is to create a platform for teleassistance and visualization, which makes the implementation part a fundamental part. This chapter explains the methods and tricks to integrate the different systems and libraries into one application. It answers the question how the empty 3D space is filled with content, what techniques ensure a feasible data transmission and save the frame rate from dropping. Furthermore, the user-interface will be specified in detail, by defining and discussing the possible actions. The chapter is concluded by a summarizing system overview.

4.1 The Fundament

... is the Ogre3D tutorial application. It is published on the website [46] and contains a basic structure to run the engine and already handles keyboard and mouse inputs. The structure separates the scene creation from the rest of the application, aiming at an easy setup for large static environments. Although the environment is rather dynamic in this application, the structure was preserved and the scene creation is used for the initialization of the graphic objects instead. Every entry point to the application layer of the rendering engine is given in the form of callbacks and hence, for every event the application needs to react on, the base application implements the according listener pattern, e.g., a frame listener to get access to the start, the event queue and the end of each frame. The input structures from OIS and ROS fit quite well into this setup, since they both work based on callbacks, or at least have the option to do so. A part that has to be adapted is the handling of the player motion, for which the tutorial application applies a utility class. The stereo rendering for the Oculus Rift however requires a certain setup of camera and scene nodes in order to model the user’s head and the according distances described previously. For the handling of the Oculus Rift, a free package from the Ogre3D community member Kojack was used. It takes care of the necessary scene node and camera setups and includes material, composer and shader scripts for the lens distortion and color adaptation. The interface is a single \texttt{oculus} object which has to be instantiated...
before the rendering is started. It initializes and registers the motion sensors, sets
up the camera nodes for left and right eye and connects them to viewports on the
left and right half on the application window, accordingly. In the frame events of
the rendering loop, the view can then be updated, simply by calling the `update()`
method of this instance. To comfortably organize the movement of the player body
(independent from the head), the package was slightly modified to fit to a utility
class, which was programmed to wrap the handling of the user input and convert it
into player movements. Since the Oculus Rift blocks the user’s vision, a user input
relying on keyboard and mouse seemed inappropriate and the options were therefore
changed to a gamepad as main input device.

With these elements in place, the application is able to render any scene in stereo
and take various user inputs from keyboard, mouse, gamepad and the Oculus Rift
into account.

4.2 Building the Environment

The central question now becomes how to fill this empty space with the 3D data
the robot collects. At every time instance, the user will see a low-fps (2 fps) video
stream from the camera sensor and further options are given to save 3D information
from this video stream. The kinect-like camera publishes a rgb image and depth
image and the low frame rate is a compromise due to the bandwidth as discussed in
the following subsection 4.2.1. Afterwards, the process of reconstructing a 3D view
from the two images is explained in subsection 4.2.2. Furthermore, the message
handling and synchronization is discussed and the user interface is introduced.

4.2.1 Bandwidth considerations

On the robot itself the information is mostly used in the form of point clouds,
where each point consists of the 3D coordinates, color information (rgba) and other
associated data, e.g., a label to denote a cluster this point belongs to. Because
the point-cloud data is comprehensive and uncompressed, it yields some problems.
Firstly, the data has to be transmitted from the robot to the rendering PC and sec-
ondly, the data has to be processed and inserted into the scenery without decreasing
the frame rate too much. The Asus Xtion camera on the robot provides a resolution
of \(640 \times 480\) pixels resulting in \(307200\) points for each image. The required minimal
bandwidth \(B_{pc}\) for the transmission of an unlabeled point-cloud could, for instance, be

\[
B_{pc} = (640 \cdot 480\text{points} \cdot 4\text{bytes/float} \cdot 4\text{floats} + s_h) \cdot f_s \approx 49.15\text{MB/s} \quad (4.1)
\]

if the sampling frequency \(f_s\) is set to 10Hz and all values are coded as `float32`,
i.e., one float for \(x, y\) and \(z\) and one extra to code color as rgba and assuming
additional header information in the size of \(s_h = 16\) bytes. Note that such bandwidth
values are far too high for the available wireless connection in this project (wifi,
4.2. BUILDING THE ENVIRONMENT

rated at 54Mbit/s, as well as the memory consumption and processing load is very expensive for rendering purposes. During the development of the system, a version with a point-cloud stream was tested on a local machine (i.e., without the wireless connection) and resulted in a frame rate between 0.5 and 2 fps. One way to decrease the data volume would be to compromise on the data rate, or the number of points, but still sub-sampling the point-clouds does not tackle the real problem.

The point-cloud contains all information in high-detail, but lacks compression. For the camera images, which were used to generate the point-cloud in the first place, efficient image compression exists in the form of the PNG and JPEG formats. With the PNG format a loss-less compression, suitable for the depth images, is possible, while the rgb images can even be compressed with the slightly lossy JPEG format.

ROS, which is used for the sensor management and the image acquisition, provides a tool to measure the bandwidth of a topic. Table 4.1 compares the measurements for the different possible topics.

<table>
<thead>
<tr>
<th>values in MB/s</th>
<th>rgb</th>
<th>depth</th>
<th>both</th>
</tr>
</thead>
<tbody>
<tr>
<td>uncompressed</td>
<td>27.9</td>
<td>37.2</td>
<td>65.1</td>
</tr>
<tr>
<td>compressed</td>
<td>1.49</td>
<td>3.53</td>
<td>5.02</td>
</tr>
<tr>
<td>point-cloud</td>
<td></td>
<td></td>
<td>121.5</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of data representations for transmission (Peak values at a frequency of 20 pictures per seconds). The point-cloud was published at 10 – 11Hz, but uses a different format as in the derivation example above.

Compressing the images yields an acceptable bandwidth and makes the wireless transmission feasible. On the applications side, the remaining task is to process the compressed images into something that can be rendered in 3D. The obvious choice, of generating a point-cloud on the rendering PC as well, has been tested, but was discarded for performance reasons: Ogre3D allows the creation of so-called ManualObjects, which are containers for user-defined geometry that can even be changed during runtime. However, each manipulation of an object (creation or update) requires a recompilation of the geometry, leading to a significant pause in the rendering loop and destroying the fluency of the application. Considering the fact that the point-cloud is constructed from the rgb and depth image, given the camera position and orientation, it can be stated that the images themselves must contain all information needed for the 3D reconstruction. Thus, the method of choice is to rebuild the view directly, instead of converting the data into a point-cloud. These reconstructed views will in the following be called (3D) snapshots.

4.2.2 Snapshot Reconstruction from Textures and Shaders

In subsection 3.4.4, the shaders were mentioned as mini-programs to operate on vertex and fragment data and to post-process rendering objects, e.g., by applying textures. The method described in the following makes use of the fact that the
geometry of the images, i.e., the number of vertices and their interconnection as a triangular mesh, is the same for all images. Rather than redefining and recompiling the whole geometry for every incoming snapshot, a single standard geometry is defined, compiled to a mesh and the snapshot each time instantiated as a copy of the default mesh (saving the time for the compilation). Figure 4.1 shows the geometry of the snapshots.

![Figure 4.1: Left: standard geometry for the snapshot objects. Right: back-projection from the standard geometry, considering pixel neighborhood for color-interpolation and the taking z-axis as focal length.](image)

Each vertex is connected to 6 out of 8 neighbors in a dense triangular mesh. In the standard object and its local coordinate system, the vertices just need to have a horizontal and vertical coordinate, the depth can be set to a default value \((0.0f)\) and will be replaced from the depth image by the vertex shader. Note that in Ogre the default depth value might still be used for collision checks and clipping, which can make it necessary to adjust the value. In addition to that, the texture coordinates for the rgb image will be set along with the geometry.

A depth image contains the \(Z\) coordinate of a 3D point, which is the distance between the projection origin of the camera and the point along the camera plane normal. In order to perform the back-projection, the following formulas hold

\[
\begin{align*}
X &= \frac{x}{f} \cdot d \\
Y &= \frac{y}{f} \cdot d \\
Z &= d
\end{align*}
\]  

(4.2)

Where \(d\) is the depth value, \((x, y)\) are the image coordinates as shown in Figure 4.2 and \((X, Y, Z)\) are the 3D coordinates. The fractions \(x/f\) and \(y/f\) are the same in each image and since the coordinates have to be rearranged anyway, the geometry positions can be used to store the fractions directly, instead of \((x, y)\), which will save some computations. Finally, the rgb and depth images are loaded as textures and
4.2. BUILDING THE ENVIRONMENT

registered in the material of their respective mesh-instance in order to have them accessible for the post-processing by a vertex and fragment shader.

![Diagram of shader programs](image)

Figure 4.2: Back-projection of a snapshot. The depth and rgb values are read from textures and applied in shader programs. A vertex shader takes care of the correct placement, while the fragment shader sets the final color value and eventually discards pixels, if they are in an area of a jump in the depth.

As shown in Figure 4.2, first, the vertex shader registers the specified depth texture and performs the transformation from Equation 4.2 in the local coordinate system and afterwards transforms the snapshot into the world coordinate frame. The transformation into world coordinates is always accessible and calculated in the 3D engine by traversing the scene graph and applying all transformations from the current scene node to the root. As a result the (basically) 2D geometry is now converted to a 3D geometry and the mesh makes it possible to construct a surface. Second, the fragment shader takes care of the coloring. It registers both depth and rgb texture. The depth texture is needed to estimate, if the pixel is lying in a flat surface, or in an area where the depth value makes a sudden jump. In the latter case, the pixel is discarded and not displayed, since a sudden jump usually means a transition from fore to background, which should not be connected by color interpolation. If instead the surface is smooth, the fragment shader looks up the according value from the rgb texture and outputs it. In the implemented version, the estimate is done by looking up the depth values in the neighborhood of the current vertex. An alternative would be to use a third texture as a mask, i.e. a Boolean map saying whether to display the pixel or not, with a precomputed highlighting of the edges, e.g., from an image gradient. The advantage would be to have one additional texture look-up during rendering instead of six (due to the 6 out of 8 neighborhood), on the price of a longer image processing step and an additional texture registration.
4.2.3 Connection to ROS

From the past sections it is known which image types can be transmitted and how to use them in order to reconstruct the snapshot efficiently. Still, even the compressed images have to be processed and the snapshot orientation and translation has to be obtained from the robot, too. This section focuses on the missing aspects needed to complete the connection from the ROS system on the robot to the rendering application. The compression of the images is taken as given. As explained in the example in section 3.3 the image_transport package takes care of all conversions.

Multithreaded Message Handling

Sending compressed images to the application is not enough. The images are captured and transmitted by the ROS in system-specific messages with according time-stamps and other ROS header information. In the visualization, the messages have to be converted into a data format the rendering engine can handle and a synchronization between the rgb and depth channel is desirable, to ensure the best possible alignment between the images for a good 3D reconstruction.

One of the main criteria for a immersive VR experience is a smooth rendering. This implies that the effect of the image processing steps, i.e., the conversion from a sensor_msgs::CompressedImage to an Ogre::Image, which can be loaded as texture, must be minimized. It has already been mentioned that ROS supports multithreading very well, due to its modularity, but even for a single node, ROS provides the option to launch the callback cycles in separate threads. Thus, the easiest way to avoid frame rate drops is to perform the message handling as an asynchronous parallel task. For the conversion, the binary data blob of the ROS messages is loaded into an Ogre::DataStream object, which can then again be decoded into an image. Unfortunately, the message data block contains some overhead in addition to the PNG or JPEG image. Hence, streaming the data directly into the engine will not work. Instead, the stream is searched for the flags which denote the start of a PNG or JPEG image and afterwards cropped to the image content, by cutting off all previous bytes. Once all images are converted and the according camera transformations have been fetched, the callback informs the engine that there is an update and the engine will handle the update at the next frame event. To avoid crashes due to data access from multiple threads, the message handling has to be paused until the engine has placed the update, which can - on low frame rates - cause message losses.

Although the processing step is an asynchronous task, the rgb and depth messages have to be synchronized. This is again supported by ROS in the form of message_filters and TimeSynchronizers with different policies for synchronization. Figure 4.3 illustrates the process of synchronization and conversion. Basically, the message_filters are buffers, subscribing to a topic and collecting a number of messages sorted after a certain criterion. Together with the TimeSynchronizer these buffers are used to find pairs of messages, based on their time stamp and can
4.2. BUILDING THE ENVIRONMENT

Figure 4.3: Message filtering and approximate time synchronization by ROS. The filter sorts the messages according to their time stamp and the synchronizer pairs up two approximately matching images into one callback.

be bound to a single callback to do the processing. The selected synchronization policy `EXACT_TIME` or `APPROXIMATE_TIME` decides how strict the pairing is done. For this project, it turns out that the exact policy does usually not provide rgb-depth pairs at all and hence, the approximate time policy is applied.

A synchronized image pair allows a good 3D reconstruction, but two images alone merely bear the information of the local coordinates. In order to place the snapshot in the world coordinates, the camera translation and orientation at the time instant the pictures were taken have to be known, too.

**Listening to Transforms**

The mechanism to access information on transforms is different from the usual message handling. On a robot, there are many potentially interesting local coordinate systems required by various elements of the overall system. In ROS, these local coordinate systems are being referred to as *coordinate frames*. In order to keep track of the various frames, ROS stores all transformations between connected coordinate systems in a transformation-tree (tf-tree), with the world coordinates as root element. To obtain the transformation between two different frames, the connection of the frames in the tree are searched and the according transformation matrices are applied along the path. Delays from processing or communication make it necessary to track the time dimension as well and store the past of the transforms up to a certain duration. Keeping the tf-tree up to date, creates a constant message traffic, of course, but for convenience, the handling of the individual updates are hidden from the developer within the `tf::tfListener` class. With a simple function call, the transform between all registered coordinate frames can be requested within the stored time frame. If the time instant requested is not present exactly, the returned
result will be an interpolation between the two closest states. With reference to the project, this means the callback which handles the synchronized image messages can simply fetch the appropriate transform from the tf-tree. For an optimal re-projection the transform has to match the time stamp of the depth image. Another problem in connection with the transforms, are the differing coordinate systems in ROS and Ogre3D.

### Coordinate Systems

Ogre3D applies the $xyz$ coordinate system as right, up and backward direction, which is very commonly used in 3D engines. ROS actually has specifications for two sets of base vectors. The commonly used set applies $xyz$ as forward, left and up direction. For cameras and other optical sensors, a second frame is generated, which codes $xyz$ as right, down and forward. Figure 4.4 shows all three variants in comparison.

![Coordinate Systems](attachment://coordinate_systems.png)

(a) Ogre coordinate axes. (b) ROS coordinate axes for geometry. (c) ROS coordinate axes for optical frames.

Figure 4.4: Comparing the coordinate systems in Ogre3D and ROS. $x$ is colored as red, $y$ is green and $z$ is blue. Note that all coordinate systems are right-handed.

There are different methods to transform the coordinate systems into each other. The way chosen for the application is to fetch the transformation from the optical frame to the world coordinates and rotate it to the Ogre3D orientation. This adaptation is done by decomposing the rotation into yaw, pitch and roll and resorting them to the target coordinate system.

### 4.3 Taking Action

In this section, the user interface is described in detail, naming the various input options and discussing the possible actions in the virtual environment and for the teleoperation, via a gamepad. Since the Oculus Rift blocks the user’s vision it is important to use an intuitive set of commands, which can be operated from a single device. The choice for the gamepad was taken, because it is a comparatively
4.3. TAKING ACTION

ergonomic device with few input keys and joysticks, which can easily be operated without looking. In modern computer games, gamepads are commonly applied to take a vast amount of input commands via the buttons and various combinations of buttons, triggers and joystick movements - a complexity that exceeds the needs in this project by far.

Within the virtual environment the user is able to move around freely, in order to explore the world from arbitrary angles and viewpoints. At any time instance the robot position is updated and visualized by an avatar. By holding one button pressed the user can switch modes to robot-control and the gamepad input is redirected to steer around the robot, while the last viewpoint of the user is preserved. The advantage of this setup is the ability to choose the best viewpoint for the control situation, e.g. a side-view to approach an obstacle, while the draw back is that for enduring control situations the viewpoint has to be adapted and hence, the user has to switch between robot- and player-control mode.

Switching the control modes is realized via the ROS system again, in a standalone node. The gamepad is handled as an input sensor and checked for the mode button. If pressed, the joystick state message is republished on a topic for a previously implemented control node for the robot, otherwise, it is routed to the topic for the visualization tool (see Figure 4.6). Figure 4.5 shows all input options granted to the user.

Figure 4.5: User interface for player navigation and robot control.

In "robot control mode" the signals are routed to the Scitos G5 and the user can remotely steer the robot in forward/backward direction or turn it left/right (unicycle
machine). For the "player mode" there are much more options:

- **moving the player** is done with the joysticks and allows forward/backward, left/right, up/down and turn left/right movements. Looking up/down or rolling the head is only possible from the Oculus Rift input. The idea is that the virtual player - controlled from the gamepad - represents the user’s body, which although the user is allowed to fly around, should not be pitched or rolled.

- **handling snapshots** with the gamepad buttons, the user can
  - take snapshots
  - change their visibility (on/off)
  - trigger an automatic panorama (360°)
  - flip the visibility of the panorama

- **use utilities** such as the 2D map of the surrounding, switching between first-person and free-viewpoint, or resetting the Oculus Rift orientation.

### 4.4 Overview

This final section summarizes the design from the different perspectives of ROS and Ogre3D. The overall system in Figure 4.6 displays the same components as in Figure 1.2 but adds the integrated software components and details of the communication channels. It is shown that the application integrates the Ogre3D engine (handling the rendering for the Oculus display), the OculusSDK (handling the motion sensors) and ROS (routing the joystick signal and handling the sensor data from the robot) to a 3D visualization tool. Furthermore, the graphic shows the multiple channels from the robot to the application.

There are in total four channels for image transmission: rgb and depth channel for the user-triggered snapshots and rgb and depth channel for the automatic room panorama. Finally, there is a channel to transmit the robots transformation, i.e., the position and orientation of all movable robot parts.

In the Ogre3D engine it is worth looking at the scene graph, which is built up during runtime and collects all elements that are rendered. Figure 4.7 shows the complete scene graph.

As mentioned in section 4.3 the virtual player, PlayerBody, represents the body of the user and is restricted in the possible orientations. The head movement, tracked from the Oculus Rift, is then super-positioned in the CameraNode. The RobotNode holds the robot avatar, a pre-loaded and textured mesh, and is updated each frame in its orientation and translation. While the child nodes in SnapshotLibrary and RoomSweep store the recorded textures for static rendering, the VideoStream node stores textures optimized for dynamic access in order to be updated at 2Hz. For the
4.4. OVERVIEW

Figure 4.6: System overview displaying the application components (Ogre3D, OculusSDK and ROS), the parts they handle (rendering, motion sensing, image and control message handling) and the channels used in the system.

Figure 4.7: Scene graph in Ogre3D. Oculus Rift orientation and player movement are decoupled even in the setup of the scene nodes holding the PlayerBody and the cameras for left and right-eye. There are extra nodes for the robot and the video stream, as well as two nodes collecting the 3D Snapshots and the room panorama(s).

library nodes, a wrapper-class takes care of the memory management and preallocates memory chunks in order to keep the influence on the rendering framerate to a minimum.

Flipping the visibility of the snapshots (one of the player actions) means flipping the visibility of the library node and iteratively forwarding the setting to all children. If further snapshots are taken, they are attached to the library as visible, which as a result means that two sets of snapshots/panoramas can be recorded.

Finally, an overview about the application flow is given in Figure 4.8. On startup, the Ogre3D Engine has to be initialized first, setting up the rendering and resource
CHAPTER 4. IMPLEMENTATION

Figure 4.8: Application flow. After initializing the different systems, the rendering loop is started and receives (most) updates from the asynchronously running ROS thread. All updates are introduced in the beginning of a frame event. On termination, the 'shutdown' command is broadcasted through the system and all components are cleaned up.

system, and loading plugins and needed graphical resources. The Oculus wrapper can be initialized afterwards, to configure the camera and screen setup for the stereo rendering. The final step before entering the rendering loop, is to initialize the ROS, which means to configure all topic subscriptions and to start an asynchronous thread for message handling.

During the runtime, the updates from the messages are processed (prepared as rendering update) in the separate thread as good as possible. Once an update is ready, it is inserted at the beginning of the next frame. There are two things to note about the ROS messages. Firstly, the depicted functions - with the exception of "transform" - represent callbacks, which is, why there are now connections indicating a 'flow'. Secondly, the transformations, as pointed out earlier, are handled hidden from the programmer in a Listener object and although it updates asynchronously as well, it is actually not a callback.

Stopping the program works in reverse order of the initialization. First the separate
4.4. OVERVIEW

ROS thread is stopped and the ROS components are cleaned up. After that, the Oculus wrapper and finally the 3D engine may follow. With these overviews the implementation is concluded and the reader has hopefully gained some understanding of the concepts behind the application. The next chapter introduces the experiment and the user study which were used to evaluate the system.
Chapter 5

Experimental evaluation

The question addressed in this chapter is "How do I know if and how good it works?". The emphasis in the introduction has pointed much in the direction of human-machine interaction, which will hence be reflected in the user study that forms the experiment. The goal is to get a proof-of-concept that the combination of the Oculus Rift with a mobile robot provides a useful teleoperation platform. In the linked literature some examples are included for this type of evaluation in [15, 11], in order to make the results more comparable.

The papers have in common that the users had to solve one or multiple tasks in a parkour. In [15] the research focus was on the comparison between different viewpoint options, i.e., first-person, third-person and free-viewpoint, with the result that the free-viewpoints allow the most accurate steering in terms of approaching an obstacle. The measures taken in [15] were the time for the task execution and the distance to the obstacle, which had to be approached as close as possible, but without collision. The strength in the method is that it provides both objective measures and subjective feedback to evaluate the system. In [11] the tasks included teleinteraction with objects in the environment and the number of unintended object collisions was recorded in addition to the time until task completion. Since the type of robot has a different task and equipment from the Scitos G5, it will be hard to compare the results to this study. In addition to these two user studies, the paper [47] lists common metrics for human-robot interaction and discusses their application and related factors. It identifies different classes of measures in an attempt to group the metrics in these categories, e.g., the time-until-completion is named as an efficiency measure for navigation tasks and distances relate to spatial perception (for teleoperators).

With these sources in mind, the experiment is designed to take similar measures into account. The idea of a parkour with different navigation and exploration tasks is easily feasible. In the evaluation, the focus is on the user experience and the situation awareness due to the VR. Apart from the measurement, a set of questions will allow the user to grade his experience subjectively on a scale from one to five. Before going into the details of the evaluation process the experimental setup is introduced.

\footnote{In this context, a testing route in the laboratory with tasks and obstacles.}
CHAPTER 5. EXPERIMENTAL EVALUATION

5.1 Experimental setup

Following the description in this section, it should be possible to get an impression on the environment the user has to explore and the difficulty of the task. Figure 5.1 shows a map of the experiment. Along with the necessary equipment for the control and visualization, the user is placed in a room on the corridor. The robot itself will be placed in the corridor, right in front of the room, so that the initial position is more or less unknown. The user is then told to steer the robot back into the room, where two other tasks await. The location of the room will be known at least in some simple linguistic form, e.g., "Down the corridor, on the right side". In the room, the user will then approach the whiteboard in order to read out a text that is written on it (during task 1, while the user cannot see it). Indeed, the text formulates the third and final task, which is to find a marked area on the ground and steer the robot into this 'parking position'. More precisely, the user is asked to park the robot on top of a shape, which is taped on the ground, and in the direction which is denoted by an arrow inside the shape. An example formulation of the third task could be "Find the black shape and park the robot."

Figure 5.1: Experimental setup.
5.2. USER STUDY

This aims to test the following qualities:

- **Task1**: Navigating to the room will give a hint on how comfortable the user is with the control of the robot. How well is the orientation in the environment and how easy can the robot be lead to a basically known location? An appropriate measure is the time between start and arrival in the room.

- **Task2**: Reading out a text from the whiteboard motivates the user to explore the room, experiment with camera angles and take snapshots. The evaluation will be either yes or no, stating if the user succeeded in finding and reading out the text or not.

- **Task3**: The final task of steering the robot to a denoted parking position of unknown exact location challenges the user to explore the environment to a higher extent. The final measure will be the distance from denoted shape to the robot, if the shape is not covered completely by the robot. Additionally, the angle between the denoted position and the final robot orientation is recorded. Both measures are taken, when the user is satisfied with the position, which means, it is the user ending the experiment.

From these tests a general intuition about the quality can be obtained and the provided measures will allow a basic comparability to the named sources. Note that especially the time intervals will not be comparable, because the length and speed of the robots in the references are unknown. Still, variance or range can be used to judge, if the difficulty is comparable for all participants, or if the operation time depends on the user’s experience in robotics. It should be mentioned that the approach to solve the tasks was up to the user. For instance, in order to solve task one, some did well with just the low-fps video stream, others relied on snapshots of the door.

5.2 User Study

Besides the performance measures, the user is given the opportunity to provide feedback in a questionnaire. Five questions allow grading the experience from (1) to (5), a sixth question on nausea has several options and two open questions allow for any kind of free feedback. Furthermore, the user is asked to comment on his background experience with technology and robotics in particular. The questionnaire is shown in the appendix in Figure A.1 and it is recommended to take a look in order to interpret the evaluation results.

The user study was conducted under one week and in total \( N = 12 \) participants tested the system, who were invited in small groups of 2-3 persons. Four people were researchers at the CAS/CVAP institute, four people were students with technical majors and four participants had a non-technical background.
5.3 Results

In the following, the results are listed and discussed. First, the measures from the experiment are considered, followed by the subjective feedback. The section is concluded by a discussion of the most important comments from the free feedback and reflecting on the results.

5.3.1 Measures

As explained in the experimental setup, the measures taken were the time to complete the first task, the yes/no answer, if the participants were able to read out the text on the whiteboard or not and distance plus angle for the denoted parking position. Table 5.1 shows the mean value and the range of the measures.

<table>
<thead>
<tr>
<th>measure</th>
<th>mean</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 [min]</td>
<td>7.08</td>
<td>[2:15, 14:00]</td>
</tr>
<tr>
<td>M2 [y/n]</td>
<td>1.0</td>
<td>–</td>
</tr>
<tr>
<td>M3 [cm ]</td>
<td>7.6</td>
<td>[3, 20]</td>
</tr>
<tr>
<td>M4 [deg]</td>
<td>13.6</td>
<td>[0, 60]</td>
</tr>
</tbody>
</table>

Table 5.1: Measured results of the user study. M1: time to finish task 1. M2: reading from the whiteboard (yes = 1, no = 0). M3: distance from the outer part of the shape to the robot. M4: angle between denoted and achieved parking orientation.

It usually took the participants 7:08 min to steer the robot back into the room. The major difficulty was to get the robot through the narrow door, with roughly 5-10 cm at each side at the base of the robot. One participant had problems finding the door, but managed to locate it after changing the approach from video-based navigation to snapshots. The fastest time of 2:15 min was achieved by a research engineer, who had experience in controlling the robot from before and solved the task with the video-stream, exclusively. Typically, the group without robot experience would steer the robot very carefully (with some fear of damaging the system) resulting in a longer time for Task1. The exception was one participant, who asked, if the robot would take damage if driven against a wall. After clarification of the question (a bumper on the robot base would most likely prevent such damage), the participant acted with more confidence and solved the task faster than average and without collisions. It can be concluded that the time of operation is indeed depended on the user’s previous experiences with robots, but as well on the attitude of the user, in the sense that a self-confident, unexperienced user can be expected to reach a similar time-to-completion compared to a cautious user with training.

Every participant managed to read out the whiteboard text. Only two participants needed to re-position the robot and had to flip the visibility in order to record a new snapshot of the board, before the task was solved.

No one managed to park the robot exactly on top of the shapes, covering them...
5.3. RESULTS

completely. This turned out to be a flaw in the experiment. The robot avatar in the virtual environment is slightly bigger at the base and hence, the shape appeared to be hidden in the VR, while it was still partially visible in the real world. Most people did quite well, though, with an average of 7.6 cm of visible shape. Finally, the angle between the denoted and the achieved parking orientation was 13.6°. This average is slightly biased by one large outlier of 60°, where the participant simply ignored the denoted arrow. All other values are in the range \([0°, 15°]\), with an average of 9°. Both measures show that despite the flaw with the avatar size an accurate teleoperation is possible with the system.

5.3.2 Subjective Feedback I

The questions the users could grade on the scale from (1) to (5) were:

1. "How real did the virtual environment feel?", where (1) states very artificial and (5) states absolutely real.
2. "Seeing the scene in 3D helped my orientation!", where (1) expresses disagreement and (5) full agreement to the statement.
3. "I had a good sense of what happened in the real world!", with the same scale as above.
4. "Moving in the virtual world was:" (1): difficult, up to (5): easy.
5. "Controlling the robot was:" (1): difficult, up to (5): easy.

The evaluation is illustrated in Figure 5.2 by the histograms of the participants’ answers. For all participants, there was no doubt that the virtual world was an artificial one. Still, the majority graded the experience as convincing in question Q1. The lower gradings could be a result from the observation of sensor noise, which makes surfaces appear wrinkled instead of smooth, or the lack of scene elements, if no 3D snapshots were taken.

In question Q2, the majority agreed strongly that the 3D perception of the VR through the Oculus Rift helped to navigate in the VR and to operate the robot. Some participants explicitly mentioned the benefit from the perception of the distance to obstacles, like chairs or tables, or the option to look from different angles. Q3 was answered in the higher part of the scale, but quite mixed. One fact discussed with the subjects was that the video-stream is in some cases too slow to capture people passing by. Persons appear suddenly, without warning in front of the robot and even with a slight delay due to network delays. While for some participants this was tolerable, others felt insecure about when to operate the robot and when to wait.

Q4 and Q5 have been evaluated quite similar to each other. Since both aim at the human-machine aspect of the project the result is below the expectation. Most
participants were overwhelmed by the amount of available functions and used only few of them during the experiment. One assumption is that less options might have resulted in easier handling for the participants, since the level of complexity already requires long explanation and some practicing in advance.

Figure 5.2: Subjective evaluation of the participants. Histograms showing the votes for the questions Q1 to Q5 of the questionnaire.
5.3. RESULTS

The question Q6 of the feedback (not in the figure) was about the occurrence of nausea, "I had a feeling of motion sickness/nausea:", with the options to answer 'never', 'after some time' or 'the whole time'. Out of the 12 participants only one reported to have felt a slight motion sickness the whole time. Four people reported to have felt motion sickness after a while and 7 participants reported no problems with motion sickness.

5.3.3 Subjective Feedback II

Further subjective feedback was collected in the form of comments on the questions "What was good?" (⊕), "What should be changed/improved?" (⊖) and any free commentary (⊙) on the questionnaire. In the following, the given answers are listed as keywords, based on the number of participants who named similar features/problems, starting with the frequently named ones.

⊕ 3D snapshots, easy control, free-viewpoint, first-person, room sweep, number of functions

⊖ robot control, lack of camera-angle visualization, jagged video stream, player control, easier snapshot access, instructions too fast

⊙ couple the human-head motion to the camera-angle

Some of these namings sound contradicting, e.g., the positive naming of "easy control" clearly contradicts the recommendation to improve the aspect of "robot control". The interpretation of the subjective feedback is subjective by nature, too. Still, there are several aspects that were pointed out in the commentary and in dialogue with the participants that should be part of the discussion.

5.3.4 Discussion

Reviewing the experimental results, a very positive observation is that all participants managed to operate the robot through all tasks successfully and without major difficulties. Even inexperienced participants with no prior experience in control, robotics, or gaming were able to handle the system after less than ten minutes of explanation/training and handled the jump into the cold water well. Looking at the measures M1-M4 together with the subjective feedback on the questions Q1-Q3 confirms that the concept of robot teleoperation (or teleassistance) with the help of immersive 3D virtual environments provides a capable interface with potentially high precision, good orientation and situation awareness of the operator. However, the system is in a prototype state. The users gave positive feedback about the available options and concepts for the human-robot interaction, while the criticism relates to details of the current implementation, of which some are easily corrected. For instance, one of the most frequent confusions in the interface was that, due to the signal routing to either robot or visualization, 3D snapshots could only be triggered
in player-control mode. This restriction is simply created from the implementation, not out of necessity and removing the restriction will already drastically reduce complexity. The 360\degree panoramas where mentioned as another positive feature, but were only used by a few participants. Prior to implementation, alternative concepts of navigation were discussed, e.g., the placement of points-of-interest by the user and autonomous navigation to these points by the robot, or an automatic acquisition of snapshots, if the robot moved or turned sufficiently far. Although these methods reduce the complexity of the teleoperation in the same way the room panorama does, they take away the active element from the user and make the participant merely an observer with occasional user-actions. Still, some of these alternatives could be tested in the future and the results compared in similar user studies.

In the often criticized aspect of "robot control" multiple factors accumulate. First of all, the criticism points at the sensitive response to joystick commands in forward/backward direction and pending movement commands stored and triggered along with the switching between player and robot-control mode: if the control button is released while the joystick is still pointing forward/backward, the movement speed will be kept somewhere in the system and the robot performs 'ghost movements' as soon as the control mode is switched back to the robot. The problem becomes crucial close to obstacles, in narrow environments, or, in general, wherever a precise control is required. Exorcising these 'ghost movements', should result in an improvement in the measures M1, M3 and M4.

A simple visualization of the camera angles could be done by adding a coordinate system, or just an arrow indicating the direction. Along with the visualization, the jagged video signal was mentioned, too. It is assumed that the torn/ripped/jagged appearance is a result from the internal handling of the textures in the Ogre3D engine and is therefore not easy to adjust. Different options in the engine could be tested to further investigate this problem, since it reduces the user’s trust in the video stream and in several cases (Task2) enforces the use of Snapshots although the video should basically be sufficient.

Most interestingly, the positive feedback "number of functions" and the negative point "instructions too fast" were given by the same participant. In general, participants focused on a subset of functions to solve the tasks, while this user tried to apply them all and perhaps would have needed more time to remember the location of the triggering button by him/herself. This can already be seen as an indication that the number of options maybe was too high (meaning "less-is-more" in terms of user comfort). A way around this conflict would possibly be to include a help-functionality, i.e., to provide the option to visualize the interface while wearing the Oculus Rift.

Finally, regarding the listed free user comment, suggesting to "couple human-head motion to the camera angle", it has to be mentioned that one major goal of the project was to decouple these two. The reason for this is the slow movement of the pan-tilt camera. Setting the Oculus Rift angles as reference angles for the camera would most likely be more confusing, due to the large lag between initiation and response of the movement. It is assumed that the suggestion came along with the
5.3. RESULTS

criticism about the interface to the camera angles. One option to realize a very similar approach would be, to send the instantaneous head angles on user demand (via another button), instead of linking the angles continuously. As a result, the user could re-position the camera automatically, without the need of steering it manually. Together with the visualization of the camera orientation, this should remove the source of confusion.
Chapter 6

Summary and Conclusion

This project implements a robot teleoperation interface, immersing the user in a 3D virtual environment, by integrating the OculusSDK, Ogre3D Engine and the Robot Operating System (ROS). It has been emphasized that even in highly autonomous system the risk for failure, due to unforeseen situations, is present and creates the need for assistive systems to overcome these situations. Furthermore, teleoperation interfaces are interesting during development and for educational purposes. From a research point of view, the project focused on questions regarding the human-machine interaction and natural interfaces, by applying and evaluating the use of state-of-the-art virtual reality devices (Oculus Rift) and easy to use inputs (gamepad).

A major part of this project was the implementation. Herein, a crucial step was to decouple the message handling from ROS to Ogre3D from the rendering pipeline, in order to provide a fluent visualization. This was achieved by multithreading and efficient data representation in the form on compressed (image) data and pre-compiled geometry objects, combined with vertex and fragment shading. On modern computers, this approach yields a rendering frame rate of more than 60fps, which is the recommended frame rate for the Oculus Rift. The application features a visualization of the robot position and orientation on a 2D map of the environment, two possible perspectives for the user (first-person and free-viewpoint), a video stream updating at a low frame rate (currently 2 fps; this is a trade-off due to bandwidth limitations) and the possibility to fill the environment by taking 3D snapshots from the video stream, or recording 360° room panoramas. In addition to that, the user can control the visibility of the recorded snapshot-sets, in case the scene appears crowded, or the snapshot acquisition was imperfect.

The system was tested in a study with 12 participants, who had to solve several tasks, encouraging them to explore the environment with the help of the various functions. As an evaluation, measures were taken from the different tasks in the experiment and the users were asked to give subjective feedback in a questionnaire. The result is a proof-of-concept for the immersive 3D visualization for teleoperation, since all users succeeded in the experiments, but uncovered a number of issues, mostly re-
lated to the control interface. Regarding the measures, even with the limitations in the interface, the users were able to perform a quite accurate control and familiarized themselves with the interface very quickly. It seems reasonable to conclude that the Oculus Rift is an interesting device to be used as a visualization interface in teleoperation/teleassistance - after all, the majority of the users gave feedback indicating their satisfaction about the level of immersion into the 3D world. Some of the minor flaws, named in the subjective feedback, will be corrected before the project handover, other more extensive topics require further investigation and can motivate future projects. It is expected that for the future virtual environments and HMDs will play a large role as teleoperation interface.
Chapter 7

Future Work

This thesis is concluded by introducing improvements, alternative concepts and additional features which can motivate future projects.

Improvements

Most of the human-machine interaction aspects which were criticized in the user study can be adapted easily. The command routing, for example, can be circumvented by an improved message handling in the visualization tool, removing the restrictions from the player and robot-control mode. Limiting, or rescaling, the joystick signal for forward/backward control of the robot reduces the sensitivity and makes it easier to control the robot precisely. Regarding the perspectives, it is quite easy to add a third-person view into the set of configurations, which completes the tool with all commonly known gaming perspectives. Furthermore, a method to comfortably reset the orientation of the robot camera would improve the usability. To avoid confusion during testing, supervisory functions to trigger snapshots or panoramas from the keyboard could be added.

Advances in Control

Building the environment is one of the main tasks left for the operator, in order to accomplish a good orientation and situation awareness - but it requires active control and pressing buttons. It could be tested, if an automatic acquisition of the snapshots heightens the comfort during operation, or if it degrades the user to a mere spectator. A first idea for the automatization, is to request a snapshot as soon as a sufficiently large change in position and/or orientation takes place.

If further autonomy shall be transferred from the user to the robot, the teleoperation via differential steering signals can be replaced by an autonomous navigation of the robot to user-set points-of-interest (POI). This is a first approach to merge existing navigation and obstacle avoidance into the application. A second way to combine
the steering with the obstacle avoidance, with the purpose of a higher safety during teleoperation, is motivated from \[24\]. The laser scanner and chest-camera could be used to identify obstacles and define corridors, in which the user may operate the robot freely. In a more sophisticated version, the method could assign collision risks to different zones/corridors and maybe adapt the velocity for operation based on the risk. These risks could, for instance, stem from activity measures (identifying moving objects and people) and reduce the velocity to lower the likelihood of a collision along with harm caused by it, should it occur anyways. Another risk factor could be the number of objects in an area to allow a careful operation in areas with numerous objects and providing more speed in areas which are mostly empty. Furthermore, this driving assistance could be used to stabilize the movement in narrow areas, such as the doorways from the experiment and help users to pass through similar zones.

**Advances in Interaction**

The Oculus Rift blocks the users vision. In this project, the odds were overcome by the application of a gamepad, which can be used quite easily, even without vision. Still, the number of commands is limited and with increasing numbers of features, the enforced button and joystick combinations become complex again. In the question of natural interfaces, language would probably be more intuitive than a gamepad for many functions, especially all "triggered" events, such as snapshots, panoramas and view-resets, for instance. In these cases, the user may just verbally formulate the command, instead of pressing a button that is hidden to him/her, or even remembering a combination of buttons. It can as well be imagined that the system responds acoustically, allowing an even deeper immersion into the virtual world and the interaction itself:

**Operator**: "Rosie\[a\] take a snapshot."
**Operator**: "Rosie, explore this location." ...
**System**: "I reached the position and will now record the panorama." ...
**System**: "The point-of-interest is now explored."

\[a\]The institutes robot is named 'Rosie.'

**Advances in Data Compression**

For an immersive 3D experience, the virtual world needs to be displayed at a high rendering frame rate. Low rates destroy the fluency of the movement, provoke latency and disorientation. Rather than transmitting the image information and reconstructing the 3D Snapshot at full resolution, it can be utilized that most of our world consists of rather simple geometric objects and surfaced, such as planes and cylinders. Algorithms like RANSAC aim at the efficient decomposition of point-cloud information into geometric primitives and thereby achieve impressive compres-
sion rates. Extending the system with such advanced analysis of the 3D information, can provide a more efficient bandwidth usage, a better representation of the environment, ensures an efficient rendering and could be the foundation for virtual machine learning applications.

**Advances in Visualization**

Scenario generation from independent 3D snapshots has some drawback in terms of the alignment in spacial and color dimensions. Depending on the reliability of the robot positioning and the lighting angles, the system has several imperfections. Especially at narrow doors, window frames or tables it was sometimes observed that two sequentially taken snapshots were misaligned and hence, provided (slightly) contradicting information on the environment. In the color space, these misalignment result from varying lighting conditions, seen by the camera, at different angles or time instances. Aligning these spaces, maybe in combination with an automatic snapshot acquisition, would improve the visual quality of the VR as well as the accuracy of the 3D information.

In the beginning of the thesis, the introduction of a time axis was mentioned as a way to measure and visualize activity in the surrounding. It relates to both, control and situation awareness. In control, the motion detection through activity measurement allows to react to moving obstacles (usually people) and its visualization allows the operator to judge the activeness in alive environments.
Bibliography


Appendix A
Questionnaire: Oculus Rift Control of a Mobile Robot

What is your job/background/studies?

Do you have experience with robots? (Y)es (N)o

Evaluation

Please answer the following questions on your experience in the parkour. For question 1) - 5), there are two ‘extreme’ statements and you can select a value on a scale from 1-5 to grade your experience between them, by making a clear cross (X). Question 6) has some extra options and 7) and 8) are open questions. If you feel like specifying your answer, please do so! There is plenty of space on the back of this paper.

1. How real did the virtual environment feel?
   - very artificial (1) (2) (3) (4) (5) absolutely real

2. Seeing the scene in 3D helped my orientation!
   - not at all (1) (2) (3) (4) (5) yes, totally

3. I had a good sense of what happened in the real world!
   - not at all (1) (2) (3) (4) (5) yes, totally

4. Moving in the virtual world was:
   - difficult (1) (2) (3) (4) (5) easy

5. Controlling the robot was:
   - difficult (1) (2) (3) (4) (5) easy

6. I had a feeling of motion sickness/nausea:
   - never
   - after some time (roughly, when _________)
   - the whole time:
     - only little (1) (2) (3) (4) (5) strong

7. What was good?

8. What should be changed/improved?

You are invited to leave any further thoughts, feedback, or commentary on the back of this paper!

Thanks for participating.

Figure A.1: Questionnaire of the user study.