Universität des Saarlandes
Naturwissenschaftlich-Technische Fakultät I
Fachrichtung Informatik

Long-Distance Teleinteraction Using 3D Environment Reconstruction in Virtual Reality and Video Streaming

Masterthesis

vorgelegt von

Marvin Barth

am 17.01.2018

Angefertigt und betreut unter der Leitung von
Prof. Dr. Tetsunari Inamura
Christian Felix Bürckert, M.Sc.

Begutachtet von
Prof. Dr. Dr. Wolfgang Wahlster
Prof. Dr. Tetsunari Inamura

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, ...

(Datum / Date) (Unterschrift/ Signature)

Abstract

With the introduction of *Industrie 4.0*, there is an increasing need for human agents to be physically present at possibly distant locations with the purpose of remote interaction. As a result, the use of mobile telepresence robots recently has become a prominent research topic. To provide the human agent with a better perception and understanding of the robot's situation, head mounted displays are often used to visualize the robot's environment. Delay of robot movements and streamed video footage as a potential cause of motion-sickness have emerged as some of the main challenges for this technology. Especially in long-distance scenarios, where the induced network latency cannot be avoided due to the given infrastructure, there is a high risk of motion-sickness for the human agent. To work around this latency, this thesis proposes an approach utilizing 3D reconstruction to create a local copy of the robot's environment. Using this locally created copy, the environment can be visualized with a low latency from there on, thus reducing the risk of motion-sickness for the human agent.

In the context of this thesis, a concept for the proposed approach is designed. The concept describes the architecture of the system including the different entities making up the system, their purpose and the interfaces between the different involved entities. The concept is then implemented prototypically for evaluation. During the development, good practices and potential issues are located to produce reference material for future research. The involved 3D reconstruction is implemented using three different approaches utilizing different technologies. Two of the three approaches, namely the visualization with meshes using the Project Tango and the visualization with point clouds using the ZED Camera, do not result in a successful implementation due to practical issues. The third approach, which utilizes compressed point clouds from the second approach to visualize the robot's environment succeeds and is used for evaluation.

The evaluation consists of a technical evaluation of the three approaches to 3D reconstruction and a performance evaluation of the implemented prototype.

Also, a user study for an evaluation of the user experience provided by the implemented teleinteraction prototype is prepared.

Contents

1 Introduction

Telepresence describes the concept of a human agent being able to perceive, communicate or even interact in an environment in which the agent is not actually physically present, but instead is embodied from a possibly distant and different space (see 1.1), usually through the use of different technologies [15].

A core requirement for telepresence often is the application of various stimuli recorded in the remote environment to give the agent some sort of feedback, as well as the experience of being on-scene, while engaging in telepresence [52]. Very basic but common examples like *Telephony* and *Video Telephony* include the transmission of sound and or video footage in both directions to give each of the participants the impression of being present with the partner with whom they are communicating.

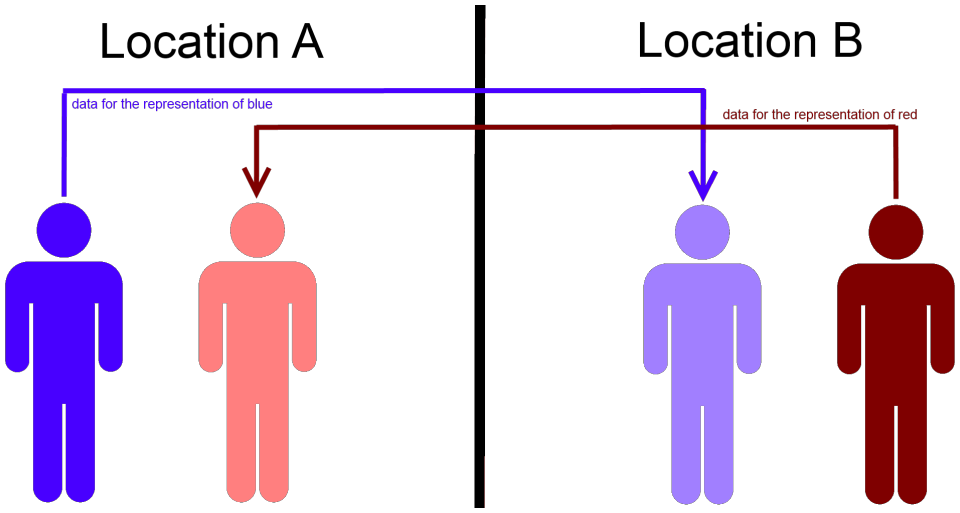


Figure 1.1: Sketch of the basic concept of telepresence.

Teleoperation describes the concept of operating a system or machine remotely, possibly over a long distance (see 1.2). In teleoperation, two major parties are involved: The operator, who is usually one or multiple human agents and the machine or system, which is operated to serve a certain purpose. Tools like remote controls or computers are usually involved, when sending commands to operate the machine or system via some kind of wired or wireless network.

The two concepts of telepresence and teleoperation come together with the term *Telerobotics*. Telerobotics refers to an area of technology, or more specifically robotics, which focuses on the operation of robots over distance, using some sort of wireless network for communication [44].

So called *teleoperated telepresence robots* are a popular approach to embody a human

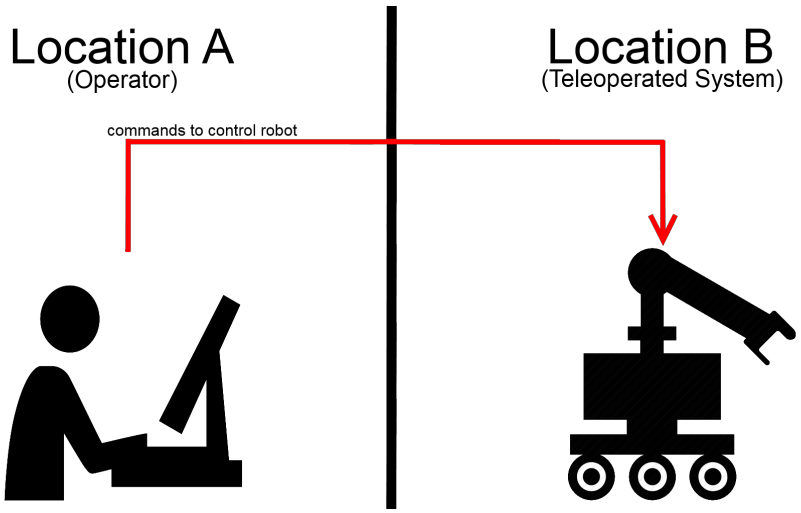


Figure 1.2: Sketch of the basic concept of teleoperation.

agent in an environment to grant him physical presence[1]. Existing telepresence robots can usually be remotely operated by the human agent to navigate around, while often playing back image and possibly sound of its operator to indicate the presence of the operating human. Live video and sound can be recorded by the robot to mimic the senses of seeing and hearing. Exposing the operating human agent to these stimuli, will let him perceive the robot's environment with a certain degree of realism and allow for informed navigation. Some teleoperated telepresence robots are even able to physically interact with their environment, leading to the concept of *Teleinteraction*, which is the topic of this thesis.

Teleinteraction is what we will call the enhanced concept of telepresence that allows for not only embodiment in an environment, but also the interaction with said environment via a teleoperated robot (see 1.3). This means that using teleinteraction, a human agent can physically interact with a possibly remote environment, in which he is not physically present, by teleoperating a telepresence robot to act in his stead. An early example for this approach is telesurgery, the idea of enabling remote surgery [6]. In the context of this thesis, we will often refer to the human agent, who is teleoperating the robot, as *operator* and to the teleoperated robot as *surrogate*.

Based on its key features, the concept of teleinteraction can have a wide variety of applications. Generally speaking, tasks that are located remotely could be executed with teleinteraction instead of sending an actual human agent to the site, given that a fitting surrogate is present to perform the task, while an operator has sufficient information about the surrogate's environment and control over the surrogate to perform the task [36]. While this is a strong condition for the usability of teleinteraction, it already reflects some of the main requirements that have to be met in order for teleinteraction to be viable (see 1.4).

One of these requirements would be that the used telepresence robot is both technically and physically capable of executing the task at hand. This means that the robot

[1]VGO Communications Inc, a company specializing in teleoperated telepresence robots. Retrieved on 17/10/2017 from http://www.vgocom.com/

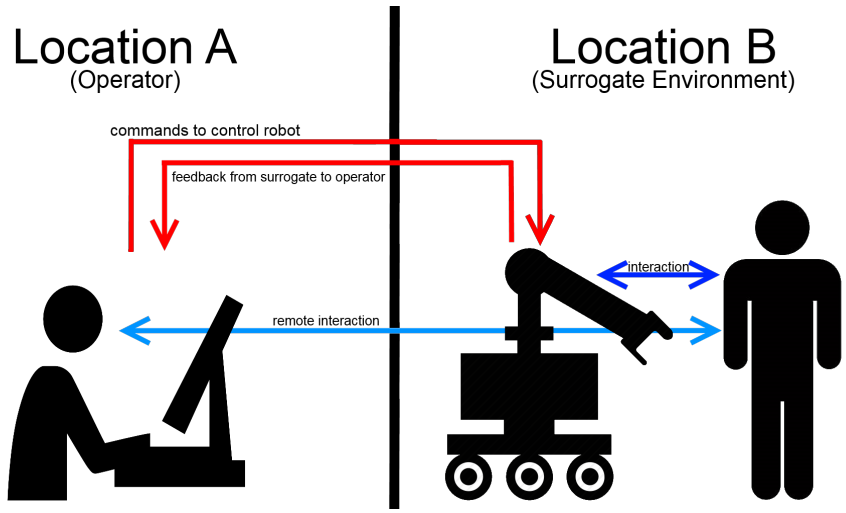


Figure 1.3: Sketch of the basic concept of teleinteraction.

must be able to operate in the target environment in general, as well as to be able to execute the task at hand, given its physique and equipment. If a robot was for example, neither heat-resistant nor able to pick up objects, it would be unlikely to be usable to reliably do repairs using a screwdriver in a smelting plant.

Another crucial requirement would be a proper way of controlling the telepresence robot. To execute a task, all relevant parts of the robot would have to be controllable without the controls interfering with the execution of the task. Thus, assuming that there is a robot that is capable of executing a certain task, there would also have to be some way of controlling the robot to do so, which the operator can handle.

The last requirement I am going to discuss at this point is sufficient feedback to the operator to control the surrogate. The operator may be in a different space or too far apart from the surrogate, so he might not be able to perceive the surrogate's environment in enough detail or at all. In these cases, it is important to implement some sort of feedback for the operator, by recording stimuli to which the operator can be exposed to perceive changes in the environment and react accordingly. A simple example would be the streaming of live video recorded by some teleoperated robot in a navigation task to aid the operator's orientation.

These are only a subset of the requirements that have to be met to successfully apply teleinteraction to various scenarios. However, these three requirements are some of the core requirements and are shared by most of the possible applications of teleinteraction.

An example of existing forms of teleinteraction is the disposal or defusing of bombs (see 1.5). As a human agent would be highly endangered, while approaching the bomb to defuse it, so called bomb disposal robots are used to take this risk in a humans stead. This type of robot is often equipped with the equipment to cut wires in order to defuse bombs, as well as some sort of video recording device to provide live image of the bomb and environment. A bomb expert will act as the operator for the bomb disposal robot, navigate it to the bomb and have it defuse the bomb, using the robot's equipped tools.

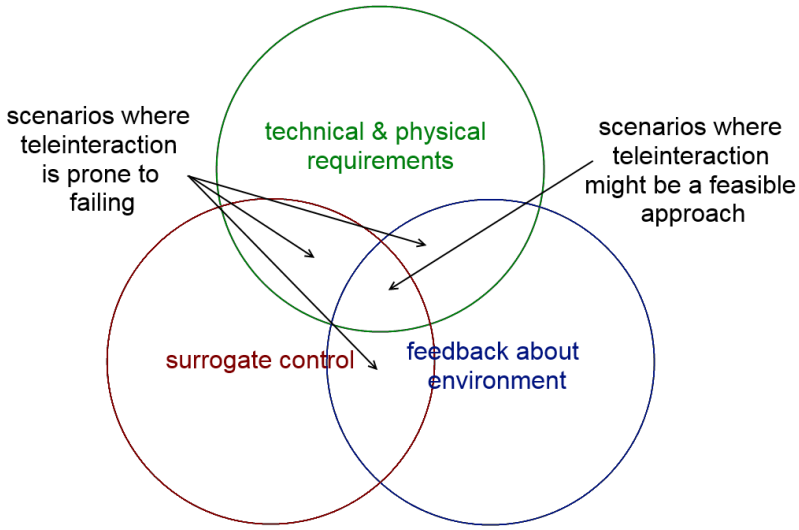


Figure 1.4: Sketch of the basic applicability of teleinteraction.

1.1 Motivation

With the *Internet of Things* becoming a prominent topic in economy, the world is in a state of digital transformation[2]. The *Internet of Things* describes a collapse of the boundaries between the real and the digital world. With technology evolving, devices and objects in the real world become able to communicate with each other, making machine-to-machine, machine-to-infrastructure and even machine-to-environment communication possible if paired with the internet. With the advent of the IPv6 internet protocol, it would be theoretically possible to assign each and every physical object a unique IP within the internet. Combining these two thoughts lets us foreshadow that in the near future there might be a drastic increase in objects and devices, connected via the internet, making completely new concepts and approaches feasible.

In this regard, a press release was issued by the *Bundesministerium für Wirtschaft und Energie (BMWi)*[3] in Germany, on the 27th September 2014. The contents of the press release focus on *Industrie 4.0* and its successor, the *Smart Service World* (see 1.6).

Industrie 4.0 describes the aim of commencing a fourth industrial revolution in the history of mankind, making use of the Internet of Things and the modern networking capabilities of devices to drastically improve productivity and efficiency in existing economy[4]. The fundamental concept of Industrie 4.0 is the creation of a digital network using modern information and communication technology to connect human agents, products, machines, logistic and facility of an economy. There are four principles making up the concept [21]:

- *Interoperability*: connectivity of machines, devices, sensors and people to allow

[2]Internet of Things. Retrieved on 18/10/2017 from https://goo.gl/LP4oph
[3]Contents of the press release. Retrieved on 18/10/2017 from https://goo.gl/oeiYfE
[4]Industrie 4.0. Retrieved on 18/10/2017 from https://goo.gl/eFzQij

Figure 1.5: Image of a bomb disposal robot. Source: https://goo.gl/qCdV9K

for communication with each other via the Internet of Things

- *Information transparency*: ability to build a virtual copy of the physical world by enriching digital plant models with sensor data

- *Technical assistance*: support through aggregation and visualization of information to the human and assistance by easing or covering a range of tasks for human workers

- *Decentralized decisions*: automation of the system to make decisions and perform tasks autonomously, unless exceptions require a delegation of the situation to a higher instance

Based on these principles, production and production steps can be encapsulated into highly automated modules. Identical modules can then be deployed to any location, where production should take place (see 1.7). This allows for a highly flexible mode of operation, yielding increased productivity and more efficient operation.

Based on the concept of Industrie 4.0, the concept of a *Smart Service World* was formed. The basic idea behind the *Smart Service World* are internet-based services for not only economy, but also in other areas[5]. Various disciplines from science to everyday life have already shown that there is an increasing need for these so called *smart services*.

Prominent examples are remote medical treatment or telesurgery in medicine [6, 33], remote mentoring and tutoring in education [30, 29], remote office in work environment[6]

[5]Smart Service World. Retrieved on 19/10/2017 via https://goo.gl/47oaWp

[6]Suitable Technologies, "Beam Telepresence Robot". Retrieved on 11/10/2016 via https://www.suitabletech.com/beampro/

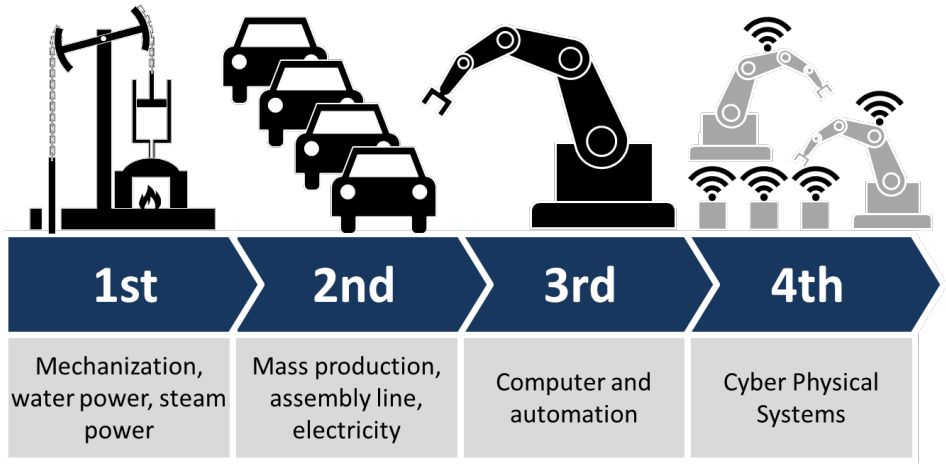


Figure 1.6: The four industrial revolutions. Source: https://goo.gl/3t4chJ

and accessibility to certain services for disabled people in everyday life [35][7]. The named kind of tasks usually require a human agent to go to the site of the task in person to execute it. However, with the use of appropriate technology, it might be possible to solve these tasks virtually and remotely. Smart services and the underlying infrastructure might just be the appropriate way to attempt solving the issues at hand.

By its design principles, smart services operate with a high degree of automation. Actual human involvement is often only required in exceptional situations, for example in emergency or error cases. As it might not be optimal to employ a human agent to stay on stand-by waiting for such situations, it would be more efficient to have the human agent only deployed in case he is needed. For example, a single human agent could be assigned to different instances of a smart service and react to them, whenever an exceptional situation arises. Depending on the nature and location of the task, the agent might have to travel to the site in the context of the task. A remote resolution could be favourable. Assuming there was a remote solution, this would allow to have one centrally located human agent to be assigned to various instances of a smart service in different locations, watching over them and being deployed to single services if required. Teleinteraction and telepresence with the use of teleoperated telepresence robots could serve as one possible solution to implement this scenario of smart services, where physical presence of the human agent is required to act or interact with the smart service.

There already are areas, in which teleinteraction and telepresense have emerged as a popular approach to remote execution of tasks. For example, in office work environment, telepresence robots are already used for home office or working in remotely located facilities [8,9]. The used telepresence robots are able to navigate through office space, play

[7]Suitable Technologies, "Museums Adopt Technology to Give Guests with Physical Disabilities the Opportunity to Tour Art Exhibits with BeamPro", Retrieved on 10/10/2016 from https://blog.suitabletech.com/2015/03/12/museums-adopt-technology-to-give-guests-with-physical-disabilities-the-opportunity-to-tour-art-exhibits-with-beampro/

[8]Suitable Technologies, "Beam Telepresence Robot". Retrieved on 11/10/2016 via https://www.suitabletech.com/beampro/

[9]Double Robotics Inc, "Double Telepresence Robot," Retrieved on 11/10/2016 from

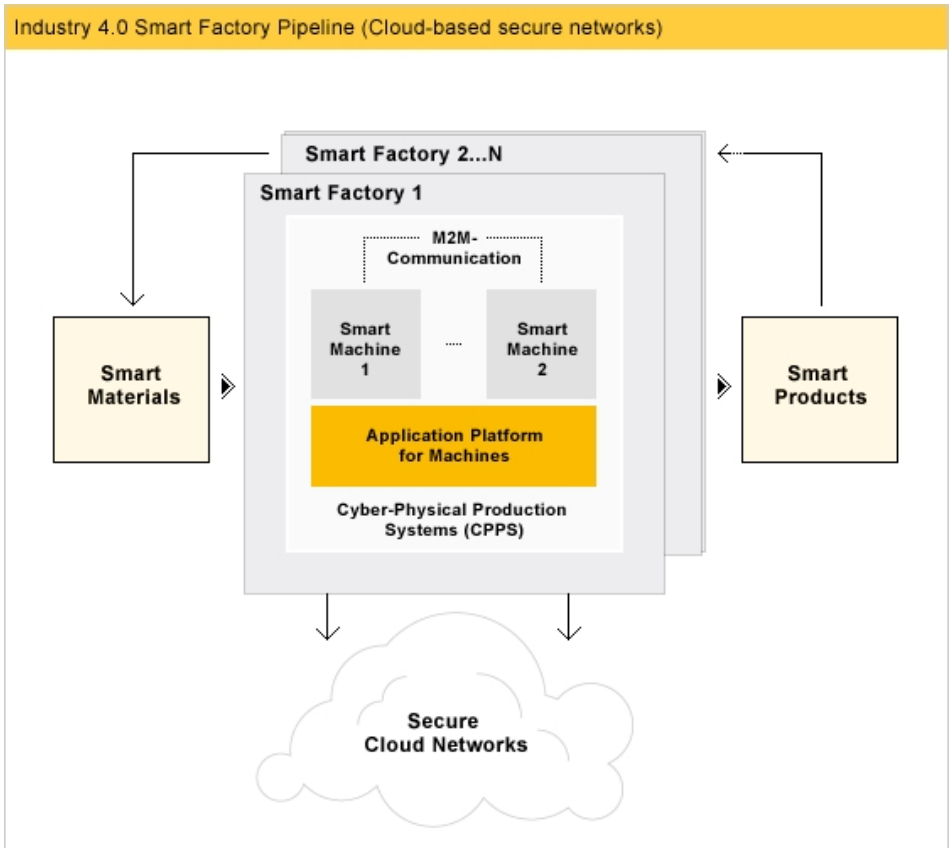


Figure 1.7: Production modules in Industrie 4.0 in the context of smart factories. Source: https://goo.gl/E6d1o4

back live sound and video of the operator and record live feedback using microphones and video cameras. To play back recorded feedback from the surrogate, conventional displays and speakers are used at the operators location. In this scenario, the surrogate functions as the operators substitute for interacting with co-workers and attending of meetings, allowing the operator to perform his duties in cases, where it would not be possible or simply too inefficient to attend the office in person (see 1.8).

Depending on the scenario and task, it can be very helpful or even important to create a high level of immersion to help the operator's orientation when controlling the surrogate to work on the task [42]. Conventional telepresence approaches often involved playing back video footage on simple computer monitors as feedback for telepresence and teleinteraction [43]. However, displaying video footage on monitors is lacking immersion for the operator. Recent approaches have emerged utilizing *Head Mounted Displays (HMD)*, which are used in *Virtual Reality (VR)* to reach high level immersion, for visualization and sometimes even controlling of the surrogate.

With the introduction of HMDs to telepresence and teleinteraction, while creating new opportunities and capabilities, new challenges and potential issues are introduced as well. HMDs are known to cause motion or simulator sickness, if not utilized properly. A too high delay in what is visualized by the HMD's monitor is one of the main reasons

http://www.doublerobotics.com/

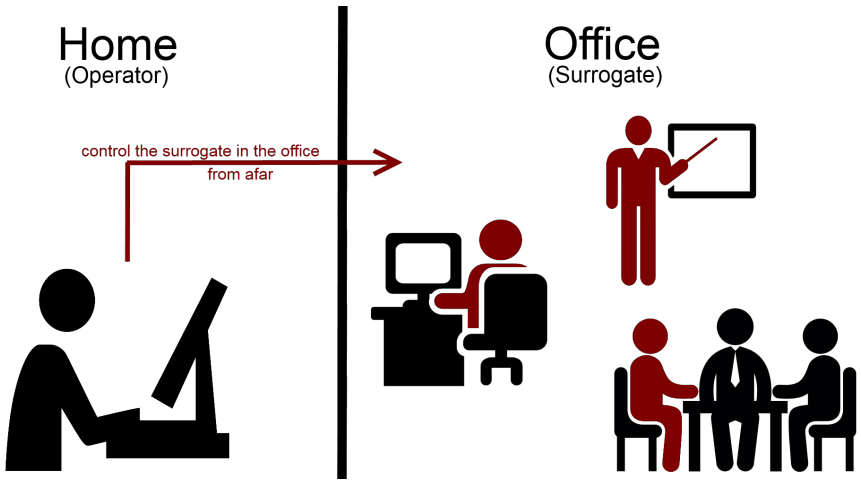


Figure 1.8: Sketch of the basic concept of remote office work using telepresence.

for that [38]. Therefore, the transmission of live video can be critical if there is too much delay between the recording and the playback of said video. Recent research has found promising ways to work reduce latency to a sufficiently low delay to avoid this simulator sickness for short and even medium ranges [1][10].

In the context of smart services, it is also possible that teleinteraction over long distances might be required. Teleinteraction over longer distances is bound to induce inevitable, possibly fluctuating end-to-end latencies due to the underlying network structure. These network latencies will contribute to motion sickness, especially over long range, making it hard for the operator to control his surrogate properly. While a stable latency is less likely to cause motion sickness [16], too high latencies, as induced in long range scenarios, generally interfere with teleinteraction and make proper operation unlikely.

For reference, the optimal achievable end-to-end latency would be the time that light takes to travel between the two locations using a straight connection without any detours. While in theory, using quantum physics, a direct communication over arbitrary distances could be established, this is practically not possible given the current state of technology[11]. The speed of light based end-to-end latency will therefore serve as a practical lower bound for end-to-end latency. With regard to this thesis, which is a cooperation between the *National Institute of Informatics* in Tokyo and the *Deutsches Forschungszentrum für Künstliche Intelligenz*, the lower bound of said latency would be around 30ms for the distance between Germany and Japan. Even this theoretical lower bound would be too high for optimal operation, as only a latency of 20ms or less would be non-perceivable to the operator [1]. While a latency of up to 60ms is still considered to be good enough for the use of HMDs without a high risk of motion sickness, there is not much latitude for latency straying from the theoretical value.

For a practical value, propagation delay of the used transmission media and processing times for each network hop have to be added up. For instance, given the current

[10] doraplatform.com, "DORA Platform," Retrieved on 11/10/2016 from http://doraplatform.com/

[11] *Far Apart, 2 Particles Respond Faster Than Light*, Retrieved via https://goo.gl/idhDVp on 05/12/2017

network infrastructure, pinging the NII servers in Tokyo from Germany provides a latency of roughly 272ms. This value, of course, is way too high of an end-to-end latency to allow for teleinteraction without the operator feeling motion sick [1][12].

To counteract the negative impact of the inevitable end-to-end latency on a HMD's visualization, techniques and approaches that are not or only slightly influenced by latency have to be researched. By reducing the latency of e.g. a video that is played back or working around the too high latency with which an environment or such is visualized, it should become possible to perform long-range teleinteraction without risking motion sickness of the operator in action.

The idea of this thesis is to realize an alternative approach to long-distance teleinteraction by combining 3D reconstruction in virtual reality with conventional video streaming (see 1.9). The operator will immerse into a locally managed virtual environment, which resembles the surrogate's surroundings. For this purpose, the surrogate will record its environment and send the necessary data to the operator, such that a virtual clone of the environment can be constructed. Even though delayed, once the data has been received and reconstructed locally, the environment can be navigated without any involved network transmission, thus circumventing the troublesome end-to-end latency interfering with live video based approaches. While, the surrogate's environment might change and updates might have to be applied to the virtual copy via the network, many scenarios will likely not change so drastically as that some delay to said updates would be critical or not tolerable. This way, having worked around the core issue of network latency interfering with live image, the operator should no longer be exposed to the high risk of motion sickness due to latency.

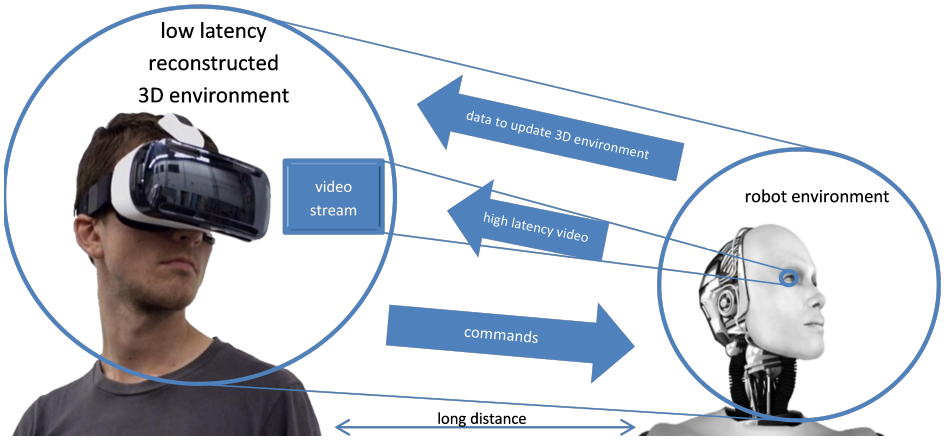


Figure 1.9: Concept of long-distance teleinteraction utilizing 3D reconstruction and video streaming.

While one might suggest that a 3D reconstructed environment alone might be lacking to fullfill certain tasks, which require a very detailed visualization of the environment, it could suffice to add the delayed conventional video streaming to supplement these details in addition to the 3D reconstruction. As for how to add video to the VR

[12]Oculus VR, "Oculus Best Practices," Retrieved on 11/10/2016 from https://develope6.oculus.com/documentation/intro-vr/latest/concepts/bp_app_simulator_sickness/

experience, using *Augmented Reality (AR)* based fashion of adding video to one's field of vision might be a feasible idea.

To give an example, in a remote inspection of an assembly line, 3D reconstruction might well be sufficient to navigate around the assembly line to points of interest, but possibly not enough to inspect fine detailed parts. However, after having located a part of interest, video streaming could be used to transfer highly detailed image of the part for inspection. As much less movement of the surrogate and surrounding objects is involved in such a situation, the delay in video transmission due to the end-to-end latency will likely not pose a threat to the operators well being.

Video streaming is so to say used as an additional extending medium to 3D reconstruction supplementing for possible drawbacks and improving the overall experience of the approach, while not conflicting with the goal of avoiding motion sickness for the operator in action.

As already mentioned, there are two other core requirements for the successful application of teleinteraction: physical and technical capabilities of the robot, as well as a proper control interface for the surrogate. In this thesis, I am not going to focus on either of these requirements. The main objective of this thesis is to find a proper way to provide sufficient feedback to the operator about the surrogate's environment. A simple robot capable of navigating that environment is thus sufficient. Several companies have already designed robots tweaked for the execution of various tasks[13], so by picking an appropriate robot for the task it is possible to meet the requirement of physical and technical capability. Of course, the reward of applying teleinteraction to a scenario should be weighed against the costs, since customized robots for the solution of a task might be expensive. As a proper control interface for a robot heavily depends on the type of robot and task at hand, it does make little sense to focus on the interface at this point. In the context of this thesis we will design a control interface for our general teleinteraction scenario. This control interface is likely to not be sufficient for more specialized robots or certain tasks. In these cases, more powerful control interfaces [39, 53][14,15] could be used, depending on the task and robot to be used in the teleinteraction scenario.

1.2 Goals

For the context of this thesis, there are several goals related to the design and development of a long-distance teleinteraction approach utilizing Head Mounted Displays. In the following, a list of these goals will be given and the details of each of the goals will be elaborated:

- **Finding a way to counter simulator sickness in long-distance teleinteraction and telepresence:**
  How can simulator sickness induced by the inevitable end-to-end latency in long-

[13] automatica exhibition for Smart Automation and Robotics, Retrieved on 06/11/2017 from http://automatica-munich.com/about-the-fair/exhibition-sectors/service-robotics/index.html

[14] neurable brain interface, Retrieved on 04/11/2017 from http://www.neurable.com/developers

[15] Capio Dual-Arm-Exoskeleton Control Interface, Retrieved on 06/11/2017 from http://robotik.dfki-bremen.de/de/forschung/projekte/capio.html

distance teleinteraction and telepresence scenarios be avoided, while preserving the advantages of the use of HMDs?

- **Conceptual design of a long-distance teleinteraction approach:**
  Design a concept of the system that resembles the described approach to long-distance teleinteraction and telepresence, including 3D reconstruction in virtual reality and video streaming.
- **Prototypical implementation of the designed concept:**
  Implement a prototype of the designed long-distance teleinteraction system with appropriate hardware and software architecture for the purpose of locating practical implementation issues and documenting them, as well as creating an implementation for live experience.
- **Evaluation of the prototypical implementation:**
  Evaluation of the implementation of the long-distance teleinteraction approach to locate technical errors and weaknesses of the approach and possible potential for improvement.
- **Evaluation of different 3D reconstruction approaches:**
  Evaluation of different implemented 3D reconstruction approaches and evaluation of the viability of utilized hardware for teleinteraction purposes.

1.3 Outline

Chapter 2 will introduce related work covering research on alternative approaches to teleinteraction and the used technologies and methods of these approaches. Also, virtual reality and 3D reconstruction of environments will be discussed, as they serve as the foundation to my approach. Chapter 3 will elaborate on the design of the proposed teleinteraction system. The required components and their contribution to the system are discussed. Chapter 4 will present the implementation of the described concept, including technical and practical details, decisions and errors that were encountered or made during the process. Chapter 5 will cover the evaluation process of the teleinteraction system. A technical evaluation of the used technology is included in the chapter. Chapter 6 will draw a conclusion about the research conducted in the context of this thesis. Chapter 7 will then give an outlook on possible extensions and improvements of the proposed system. Also, a user study that could not be conducted due to a lack of time is presented.

2 Related Work

In this chapter, related work on the field of telepresence and teleinteraction will be introduced. Starting with early telepresence, existing telepresence approaches will be presented. A special focus will be on the realism and immersion of the approaches. The influence on latency on telepresence and teleinteraction experience in general and existing approaches to counteract latency in different scenarios will be discussed. To summarize the previously presented related work, already existing telepresence and teleinteraction systems will be elaborated. Lastly, 3D reconstruction with different approaches and technologies will be introduced.

2.1 Early Telepresence Approaches

The idea of telepresence dates back to before the 1980s [36]. In his paper, Minsky describes the fundamental idea of telepresence as wearing pieces of clothing that are equipped with sensors and muscle-like motors. Each motion of the body will be picked up and reproduced in another place by a mechanical body, which can in turn feel what is happening to it. Even back then the idea was to work in another location, possibly another planet, while not being bound by the limits of the physique and capabilities of our own bodies. Controlling and manipulating the sensation of the mechanical body would allow transformation of information making sensations like pain avoidable or in general your work comfortable and safe. Since then, many ideas and approaches to telepresence have been discovered, implemented and researched to achieve what was described by Minsky already back then.

One of the very first ways of telepresence is telephony [52]. Telephony enables someone to talk to a person, as if he was with that person, even though they might possibly be a long distance apart. While in this scenario, there is no mechanical body copying the motion of its operator, there is a mechanical mouth copying what its operator is saying and thus delivering voice to the operator's partner on the phone. While telephony allows for basic interaction on a communicative level, there is a serious lack of embodiment with this approach.

A more modern version of telephony is Voice over IP or Video over IP. Instead of using phones to convey sound, computers with microphones are used to convey sound and image. While Voice over IP basically is the same as telephony, except for the use of different technologies to convey sound, Video over IP is a slightly extended approach. By displaying live image of one's opponent, the opponent is very weakly embodied in your presence, by indicating the physical appearance and visual activities.

While both approaches allow for long ranged communication between human agents, there is a lack of sufficient embodiment in general.

To increase the level of embodiment and ease of communication, a new communication prototype was recently proposed (see 2.1). Even though not physically, the

Figure 2.1: Room2Room: Augmented Reality based communication [41].

approach embodies the actual communication partners in each other's environment using augmented reality [41][1]. By performing 3D capture using color and depth cameras, each communication participant can be recorded in his entirety. The approach then uses holographs to project each participant into the partner's space, while preserving perspective. Communicating with a life-sized virtual copy of the partner increases the shared understanding of verbal and non-verbal cues, contributing to the experience of an actual face-to-face conversation. However, this approach is still limited only to communication or similar scenarios requiring no actual physical interaction.

Not only does a lack of physical embodiment impede with tasks that require physical presence, but there is also evidence that physical embodiment improves interpersonal trust between agents communicating or collaborating through telepresence [43].

A simple approach to overcome the lack of physical embodiment with the aim of solving tasks that require physical interaction, is to provide remotely controllable tools. An operator can use these tools to solve given tasks at hand, with or without being present. The tools can also enhance the operator's capabilities and allow for completion of tasks that would not be solvable by the operator otherwise. A very prominent example is *telesurgery*, a form of remote surgery that makes use of highly precise robots to perform surgery that human surgeons could possibly not perform manually, sometimes even over distances (see 2.2). In telesurgery, the surgeon usually controls or rather guides mechanical arms that perform the surgery on the patient in his stead, while he gets live image of the site [6]. To control the surgery robots, controllers that include haptic devices can be used to give the surgeon not only visual but also haptic feedback [11]. This way the surgery robots performing the actual surgery on the patient will be guided on how to move and operate by the surgeons know-how. Partial routines and movements can also be automated to improve performance or ease the workload for the surgeon. While approaches like telesurgery can, in fact, allow for remote execution of certain tasks while granting some form of physical embodiment, this approach is limited to only a certain subset of possible scenarios and highly dependent on the tools that

[1]Room2Room, Retrieved via https://goo.gl/TAkdxx on 10/11/2017.

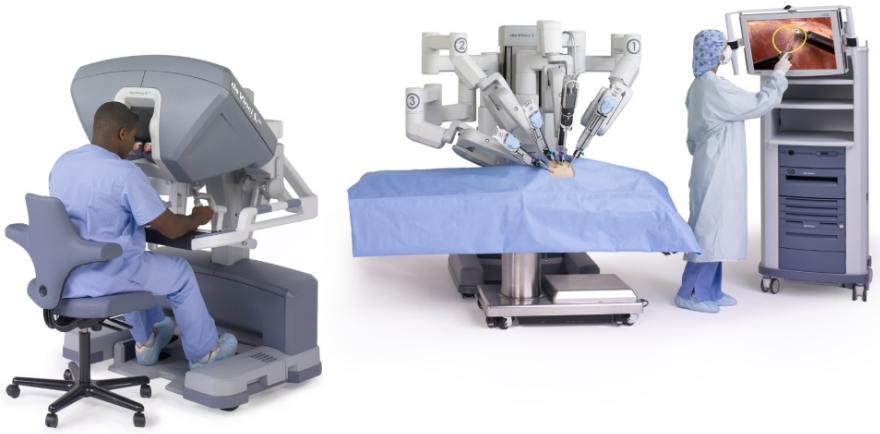


Figure 2.2: Telesurgery, remotely perform high precision surgery with the use of remote robotics. Source: https://goo.gl/y7Ju2G

are provided and their capabilities. For more general scenarios, a way of embodying a human agent while conserving and possibly enhancing his capabilities is of interest.

Thus, to avoid the general lack of physical presence and interaction, mobile telepresence robots have been designed in order to provide agents with an actual physical body to represent them. With a mechanical physical surrogate body, the operator will have a second physical body of himself in the environment of interest to act, interact and solve tasks he is presented with. Therefore, it is import to conserve as much of the agent's capabilities, tweak or even enhance them to meet all the requirements for a purpose or easy remote execution of tasks.

Not only the capabilities of the mechanical surrogate body influence how well the operator can act or interact in an environment. For an agent to be able to not only act or interact randomly, he has to perceive his surroundings or environment in some way. While different media like sound or image can be used to support the agent's perception, not all ways of conveying information are capable of encoding the full range of ambient and spatial information that is available and required to create a fully realistic perception of the environment [50]. In scenarios, where the agent requires certain in-depth information, it might be possible that a lack of said information might interfere with the execution for a task or at least impede it. For this reason, it is often desired to raise the realism of telepresence and teleinteraction applications as much as possible. The ultimate level of realism would be to trick the operator into believing that he is actually physically present in the surrogate's environment by applying stimuli to provide an engrossing total environment. This concept is often referred to as *immersion* [7]. Immersion is often said to play an important role in telepresence and teleinteraction, as deep immersion implies an increased understanding of the environment and the situation of the surrogate.

2.2 Head Mounted Displays and Virtual Reality Based Approaches

With the advent of Virtual Reality during recent years, telepresence was highly influenced. As immersion usually plays a central role in Virtual Reality related applications, VR technology has been introduced and used frequently in telepresence and teleinteraction as well.

Figure 2.3: Examplary image of the HTC Vive Head Mounted Display. Source: https://goo.gl/qRDnxr

The central idea that was adapted from Virtual Reality, is the use of so called *Head Mounted Displays (HMD)*. As the name suggests, these displays are devices that can be worn on the head of an agent. Figure 2.3 shows the *HTC Vive* head mounted display. Head mounted displays use small display optics in front of one (monocular HMDs) or both eyes (binocular HMDs) to influence the vision of the agent or to display certain image to him. In Virtual Reality, head mounted displays usually utilize two such displays to completely overwrite the vision of the real environment by a virtual one, creating the feeling of seeing a virtual reality that does not actually exist in the physical world (see 2.4).

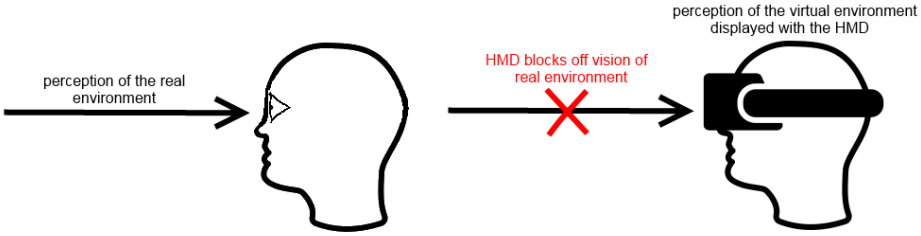


Figure 2.4: Perception of the real world vs. perception of a virtual world.

There are different features that can extend the use of head mounted displays. The most rudimentary HMD simply projects an image for the agent to see. In this most basic version the image is not based on the agent's head position or angle, so the image cannot adapt to the agent's perspective. To improve on this, more sophisticated HMDs incorporate a positioning system to track the agent's head pose. Using the available head pose data, the imagery can be slaved to the perspective of the agent and appropriately adapt the image to the agent's perspective, when he changes his pose. This allows for simple exploration of the virtual environment simply by head movements, instead of requiring the use of separate controllers. Currently, many head mounted displays still use wires to transfer data to and from the headset, thus limiting the freedom of movement of the agent wearing the HMD. By using wireless systems, this freedom of movement can be further improved. Another extension of head mounted displays is the incorporation of limb and eye tracking. These techniques represent powerful input control interfaces that can be used to allow for convenient and natural interaction with the virtual environment.

As head movements in the real world induce inevitable changes in vision and movement of the body is usually required for any interaction with real world environment, it is quite obvious that mapping these actions to virtual reality applications and technology is crucial. Therefore, to create a realistic experience, when diving into a virtual environment, techniques like limb tracking and tracking of an agent's head pose will increase the level of realism and thus the immersion into the virtual environment [14]. Using this idea, a lot of research has tried to immensely increase the level of immersion for telepresence or teleinteraction [39].

While the idea of using head mounted displays in telepresence and teleinteraction is not new, research on the topic only started picking up in recent years. Head mounted displays have been researched as both a remote control interface for robots, as well as a means of visualization of complex environment information. Even though there are several potential technical issues, there is evidence that the use of head mounted displays to support an agent's control over a remotely located robot is a viable option for telepresence and teleinteraction [42]. Mapping head movements to actual robot movements, while keeping the agent's head pose and sense of direction in mind, as to not confuse the agent, is one of the main challenges. A combination of traditional controllers and head mounted displays could be a viable approach, as it allows for separation of the agent's sense of where the robot is facing and where he is actually looking.

Based on this idea, mobile telepresence robots have been equipped with special cameras that can move relative to the robots facing direction or cover a wider range. This resembles an imitation of the human body, where the head is freely rotatable to a certain degree dividing the main body's orientation from the head's or rather the field of view's orientation.

One way to implement this is the use of pan-tilt cameras [43] (see 2.5). These cameras can usually be turned and rotated almost freely around their center, allowing for imitation of head rotations. By linking the camera to the head tracking system of the HMD the pose data from tracking the operator's head orientation can be used to synch the camera pose. The image recorded by the camera will thus match the perspective of the operator.

Another possible approach is the use of panoramic cameras, combined with head

Figure 2.5: Pan-tilt camera (https://goo.gl/AYPTE6) and panoramic camera [18].

mounted displays [18]. Using panoramic cameras a 360 degree image can be recorded at a time, without having to rotate, turn or move the camera. While the HMD cannot display the whole image, it is possible to process the tracking data into a virtual head rotation, which can then be used to display the part of the 360 degree image that matches with the right perspective of the operator.

Since video image recorded by a conventional camera does not allow for very good perception of depth, it might be favourable for the level of immersion to try and improve the depth perception and with it the spatial awareness of the displayed environment [4]. For the human, spatial vision is based on the perception of the environment through the human's eyes that are slightly offset from each other, resulting in better spatial awareness due to additional depth perception. As already explained, head mounted displays usually incorporate two separate displays, one for each eye. The concept of spatial vision can be applied to head mounted displays, by recording image with stereo cameras. Stereo cameras consist of two single cameras, arranged slightly offset to each other to mimic a pair of human eyes. By displaying the image recorded by such stereo cameras and displaying each single camera's image to a respective eye through the HMD, human vision, including spatial vision, can be imitated [27] extending the information of the displayed video by additional depth information.

There is evidence that the use of head mounted displays in telepresence and teleinteraction is a viable approach to increase the immersion level for the operator and thus his performance on a presented task. This was found in a user study consisting of a remote collaboration between a teleoperated surrogate and a human agent [28]. The task for the operator was to guide a human agent through the assembly process of a piece of IKEA furniture. IKEA is a furniture retailer that is known for their easy-to-assemble furniture[2]. The surrogate would provide the operator with live image from the scene during the execution of the task. Recording and displaying of the live video was implemented in three different ways that were compared in the study:

- recording using a conventional monocular camera and displaying via conventional display
- recording using a conventional monocular camera and displaying via HMD
- recording using a stereo camera and displaying via HMD

[2]IKEA, Retrieved via http://www.ikea.com/ on 10/01/2018.

The implementations with head mounted displays used head tracking information to control a pan-tilt platform with the camera mimicking the operator's head movements and properly display the video from the operator's perspective.

The purpose of the study was to confirm the positive influence of head mounted displays on telepresence and teleinteraction, as well as researching possible differences between the use of conventional monocular cameras and the use of stereo cameras.

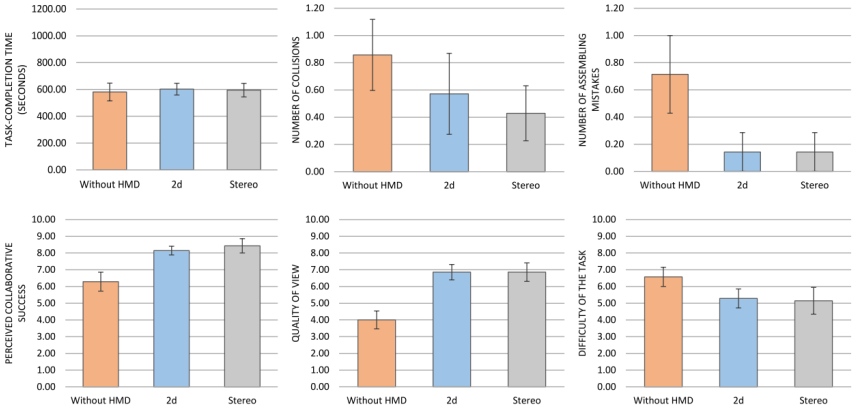


Figure 2.6: Results of the user study about the influence of head mounted displays on telepresence and teleinteraction [28].

The results of the study (see 2.6) show that the use of head mounted displays increases the task performance, while easing the workload that is perceived by the operator, compared to not using the head mounted display. Looking at the use of conventional monocular camera and stereo cameras, there is little to no difference, besides a slight decrease in collision of the surrogate with the environment, when using stereo cameras. This may suggest that either there is no real benefit in using stereo cameras or that the task simply did not require a lot of spatial vision resulting in about equal performance. However, the slight reduction in collisions indicates that the use of stereo cameras paired with head mounted displays should at least perform about equal, possibly with a slight tendency to improve situational awareness.

While head mounted displays can have great benefits for immersion and performance of telepresence and teleinteraction, there is also the critical issue of reaching true or sufficient immersion, while avoiding to break the immersion and negatively affecting the operator. The human body is adept at spotting things that are off or feel unrealistic or wrong, when being presented with a virtual or fake environment. Therefore, the use of head mounted displays is bound to the risk of inducing motion-sickness or in general simulator-sickness to the operator wearing the head mounted display [1, 10, 38]. As motion-sickness can easily interfere with the execution of a task and negatively influence the well-being of the operator, it should be avoided if possible.

There has been research on this aspect of realism and simulator sickness to find out which factors influence the perception of the virtual reality. The basic problem appears to be a feeling of stability that is easily interrupted, while immersing into a virtual environment [2, 9][3]. There is a set of influencing factors that contribute to this feeling

[3]Oculus VR, "Oculus Best Practice„" Retrieved on 11/10/2016 from

of stability in VR. These include image displaying latency, the size of the field of vision, the complexity of the virtual environment, as well as stereoscopic vision [10, 5][4].

A bigger, so called *display field of vision* or display wideness will contribute to discomfort. The *camera field of vision*, which is the vision of the virtual environment that is rendered to the display, can increase discomfort for smaller sizes, because the agent has to perform more head movements to fully perceive his environment and gain situational awareness. Stereoscopic vision and spatial depth was also found to be contributing towards simulator sickness. An appropriate use of spatial depth and appropriate placement of objects inside the virtual environment can minimize this risk though.

While the field of vision, complexity of the virtual environment and stereoscopic vision contain a risk of inducing simulator sickness-simulator sickness, careful handling of these factors can minimize this risk. The latency of displaying image to the operator via head mounted display however, is hard to control or minimize depending on the scenario. Therefore, latency can be one of the major challenges when working with head mounted displays.

2.3 Latency as a Major Influencing Factor

Virtual reality has a risk to induce motion-sickness or simulator-sickness in general, depending on the use and circumstance. Research has shown that delayed displaying of imagery via a head mounted display contributes to so called *visually induced motion-sickness* [9]. As a result, latency becomes a central factor for virtual reality induced motion-sickness [5][5]. This makes it important to pay special attention to latency, when using head mounted displays and virtual reality based approaches for teleinteraction. As already mentioned, the human body is adept at spotting differences and gaps between a fake or virtual environment that is presented with via an HMD and reality. A delayed change of the field of view for example that does not fit an operator's head movements will contribute to motion-sickness.

In general, a higher latency will have a bigger effect on the well-being of the operator. However, there have been findings that a stable latency will be more unlikely to induce motion-sickness, as human agents can get used to the latency to a certain degree if it still is within a reasonable bound [25]. Different telepresence prototypes suggest that for a non-constant latency, 60ms is the upper limit to grant an acceptable VR experience. For the operator to no longer be able to distinguish between reality and virtual reality, a latency of 20ms or lower would be required [28]. When dealing with constant latency, it is suggested that there is no significant difference between lower and higher latencies up to a certain point. The developers of the Occulus Rift, for example, suggest that a constant latency in the range between 48ms and 300ms will have no negative impact

https://develope6.oculus.com/documentation/intro-vr/latest/concepts/bp_app_simulator_sickness/, 2016

[4]Oculus VR, "Oculus Best Practices", Retrieved on 11/10/2016 from https://develope6.oculus.com/documentation/introvr/latest/concepts/bp_app_simulator_sickness/, 2016

[5]Oculus VR, "Oculus Best Practices", Retrieved on 11/10/2016 from https://develope6.oculus.com/documentation/introvr/latest/concepts/bp_app_simulator_sickness/, 2016

on the operator and his well-being [16][6].

Most recent telepresence and teleinteraction prototypes utilize live video and video streaming, to give the operator visual feedback [53, 1, 40]. Video streaming has shown to be prone to high and fluctuating latency due to underlying network infrastructure, network congestion and transmission distance [33, 31]. While techniques and streaming methods have been invented to counter this risk of latency [49, 3, 31, 24, 19], it often cannot be assured that the video's playout delay stays within the latency bounds for acceptable VR experience. Thus, especially for longer distances, using video streaming would imply a risk for the operator and might be inappropriate.

Network latency in video streaming is not the only kind of latency that is involved in telepresence and teleinteraction. The laws of physics imply different constraints that result in various types of latency, which mainly consist of motoric latency, network or transmission latency and processing latency. The different kinds of latency add up to the end-to-end latency between sending a command to the robot and its physical execution or the recording of a video frame and its actual playout.

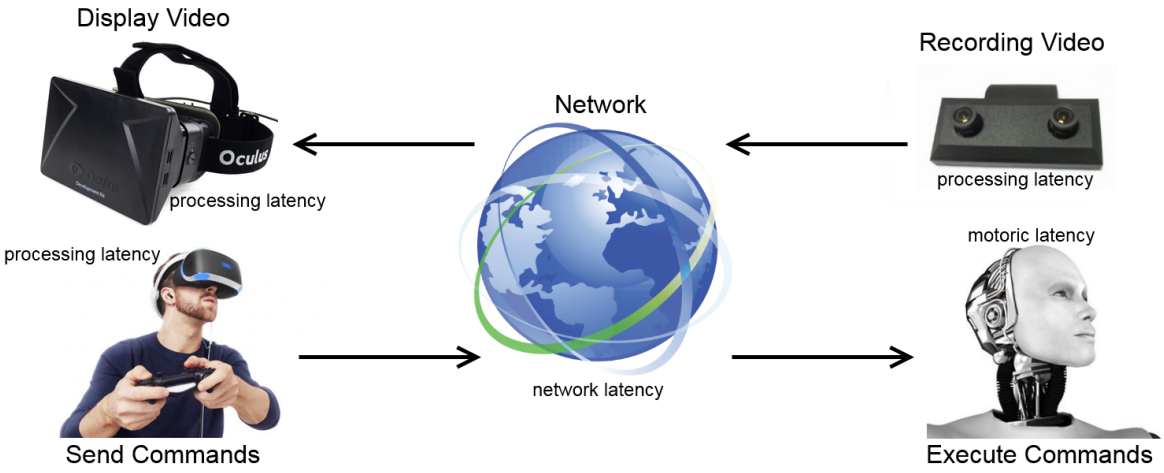


Figure 2.7: Rough sketch of the latencies involved in teleinteraction.

Because of the high potential for various kinds of latency (see 2.7), it is important to understand why and how each kind of latency is induced, how it affects telepresence and teleinteraction and how it can be reduced or countered. The different categories of latency will each be discussed in table 2.8.

Due to physics, even the best-case scenario includes latency. Not taking any robots, computers or processing into account, all that is left is the transmission of the data, thus the transmission latency. For reference, assuming that we have a direct connection between two locations and are able to transfer data at maximum physically possible speed (speed of light), there still will be a non-trivial delay of 1ms for each 300km of the connection. Actual connections using wires are of course neither directly connecting the two locations nor are they that fast. Actual wires will induce *transmission delay*,

[6]Oculus VR, "Oculus Best Practices", Retrieved on 11/10/2016 from https://develope6.oculus.com/documentation/intro-vr/latest/concepts/bp_app_simulator_sickness/, 2016

Category of Latency	Cause	Affected Entities
motoric	inertia and output of motors	surrogate's motion
transmission	physical constraints of wires and wireless on data transmission	network, display, controller
network	transmission latency, network hops, packet processing, congestion, transmission errors	underlying network
processing	limited processing power of involved machines	operator's hardware, surrogate, unit processing camera

Figure 2.8: Table of the categories of latencies involved in teleinteraction.

depending on the material they consist of and thus how fast they can transfer data. This makes the practical latency way worse than its theoretically possible bound.

Furthermore, wires are not the only devices that make up a network. Actual networks consist of many switches, routers and sometimes other sub networks that are connected with wires to ensure that network communication can be transmitted anywhere throughout the network. When sending data from one location to another, that data has to pass through all the devices and wires on its way, while being processed at each of the hops. The path taken might not be a direct one connecting the two locations, possibly further increasing latency. While a single hop in the network can be done really fast, especially for longer distances and higher hop counts, the tiny delays keep adding up until they can become a major delay between sending and receiving of information.

Commands, feedback and video frames consist of a high amount of data. Their transmission over a network is thus bound to be affected by network latency, delaying any communication between the surrogate and its operator up to a point that is possible to interfere with proper operation [39]. In common best-effort-networks like the internet, packets or data might additionally be lost and might have to be retransmitted, thus further increasing network latency.

In general it is hard to reduce network latency, as the network infrastructure is often not accessible or changeable. Working with the internet, there is usually no direct way to influence the network infrastructure, thus there is a fixed latency to work with. How much latency exactly is induced by the network can differ. Networks utilizing different infrastructures can improve or worsen this delay relatively to each other, making certain types of network more or less appropriate for certain scenarios [33].

One common way to work against network latency is to make the application more resistant to high latency or working around it in a way that does not negatively affect the application. Video streaming is a prominent example for this, because of the involved *playout latency* [19].

When streaming a video, the frames of that video have to be transmitted from a server to the client where they are played out. This already induces a basic network latency due to the transmission over a network. To reduce the amount of data, usually only

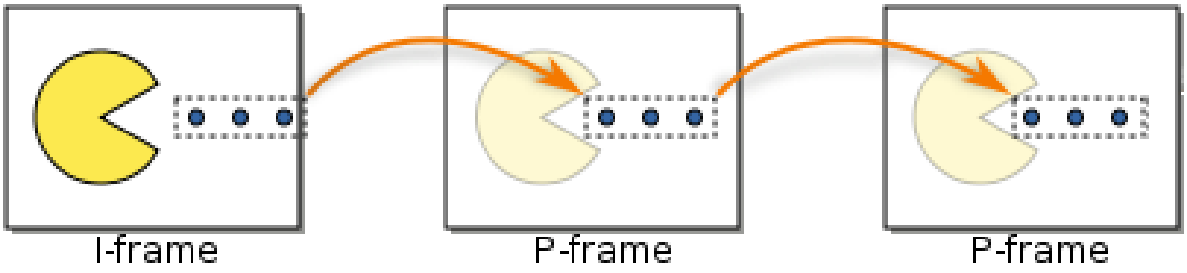


Figure 2.9: Video streaming example and frame dependency.

some frames, called I-frames, are sent completely. The other frames, called P-frames, reference an I-frame and describe the difference from that I-frame (see 2.9). The client can then construct the actual frame from this highly compressed data locally. However, if an I-frame is lost, all the P-frames referencing it would become useless, so reception of the I-frame has to be ensured by resending them, inducing a higher latency until the next frames can be displayed. For cases where frames are consumed faster by the client than they arrive, buffering and pre-loading are usually used. Frames are collected until the time to playout the available frames is enough to request and receive the missing frames of a video. This induces a delayed playout leading to playout latency. While avoiding impact on the playout of the video itself, a delayed start of the video playout is introduced. This generally conflicts with the goal of lowering latency and transmitting video live [32]. Therefore, different approaches to minimize the additional latency in video streaming [47, 32] like for example channel-adaptive media playout [49, 24, 19, 31] have been proposed. The basic idea of these approaches is to make data transmission more resistant by reducing package loss. In case of channel-adaptive playout, available channel feedback can for example be used to select and send I-frames only during times the risk of packet-loss is expected to be low. By reducing the number of required retransmissions, additional latency can be avoided, thus reducing the overall delay of video playout. It is important to note that this kind of approach cannot fight the basic network latency that is induced by sending packets over the internet, but only the additional latency introduced by the video streaming scenario. Another very central approach to working around latency is peer-to-peer networks or in the context of video streaming, peer-to-peer streaming [3]. Different machines are linked together in an additional network laid over another network like the internet to provide each other with services like providing media or video while being able to use other machines' services like receiving video. While not affecting the infrastructure of the underlying network itself, this approach can shorten the distance and hops the video has to travel, as the best video provider can be chosen from among the network based on his distance. This might then result in reduction of the actual network latency, as the network path from server to client can be reduced. Combined with reduction of the video streaming scenario's additional latency, peer-to-peer networks can drastically improve the overall latency for video streaming. Since this benefit is based on the availability of the streamed video at different nodes in the peer-to-peer network, this approach cannot be used for live video. As the video will be recorded at one location in the network, it has to be requested from a specific node setting the path that has to be used for the video transmission. Thus, the network latency will, again, be solely dependent on the distance and network

between the provider and the client.

Processing of complex data or media can sometimes be time-consuming. Therefore, recording or processing video or environment data of the surrogate's environment, processing commands to the surrogate on both client and surrogate side, as well, as displaying image to the operator or handling the surrogate's feedback can become a source of so called *processing latency*, if the workload becomes too much or operation is too inefficient. An increase in required processing time of visual feedback could increase the overall latency enough to make the operator feel motion-sick, when using a head mounted display. Furthermore, delayed execution of commands due to high command processing time could decrease the operator's controls over his surrogate interfering with the usability and possibly the ability to execute a task. Therefore, when developing or working with telepresence and teleinteraction systems, efficiency and appropriate workload should be paid attention. Also, appropriate hardware should be used to provide fundamental processing capabilities for the purpose. If a system was using virtual reality and head mounted displays for example, not having appropriate processing power could make the whole system unusable due to lagging or freezing visuals making operation impossible.

Robots or surrogates use motors to move their bodies or parts of their bodies. These motors usually cannot instantly apply the required acceleration to move a body part with the required speed. Together with inertia of the surrogate, this results in *motoric latency*, delaying a robot's intended motion after receiving or deciding to perform that very motion. While it might be possible to ignore such constraints in virtual reality, the physical surrogate cannot, leading to a discrepancy between virtual representation and the physical world [18]. This discrepancy can then in turn lead to wrong decisions about how to interact with the environment as the discrepancy increases. There are some ways on how to cope with motoric latency by either reducing it, i.e. improving the hardware to be more responsive and less prone to motoric latency, or by taking it into account in the virtual environment. Latter one can be done by simulating motoric latency or by collecting feedback data about the real world and adjusting the virtual off drifting world to reality[7].

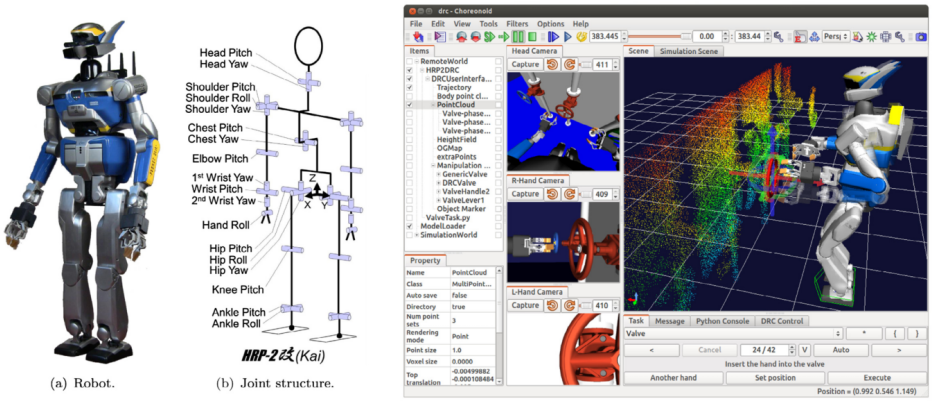
The previously named categories of latency will affect telepresence and teleinteraction scenarios. In a context of a head mounted display based scenario, all of these latencies might contribute to motion-sickness. Thus, especially in these scenarios, it is important to take all the different kinds of latency into consideration to ensure an acceptable experience. In the following section, some existing implementations are presented.

2.4 Existing Telepresence Prototypes

Telepresence and teleinteraction have advanced steadily thanks to the frequent research on the topic in recent years. Several systems and approaches have been proposed over the years making way for some really advanced telepresence and teleinteraction systems, some of which are even are sold commercially. In the following, three of those advanced systems will be presented.

When facing disaster or emergency situations as for example the Fukushima incident in 2011, it can be advantageous to deploy teleoperated robots to handle the situation

[7]Google Tango Position Tracking, Retrieved via https://goo.gl/L67sLw on 29/11/2017



(a) Robot. (b) Joint structure.

Figure 2.10: Humanoid robot and teleinteraction control interface of [12].

instead of sending human agents risking their safety. Conventional robots often do not have the capabilities to sufficiently interact in scenarios with environments meant to be accessed by humans. Humanoid robots could overcome this deficit. To propel the development of humanoid robot teleoperation, the DARPA Robotics Challenge was created. The *National Institute of Advanced Industrial Science and Technology* participated in this challenge with a very advanced teleinteraction prototype [12] (see 2.10). Different tasks like opening doors, unplugging or inserting power plugs and closing valves are given in the challenge, while working under heavily unreliable network conditions. For the development of their approach, the environment was assumed to be unstructured and partially unknown. To provide visual feedback, 3D scanner systems are attached to the robot to provide a virtual point cloud representation of the environment together with a 3D model of the robot itself. The point cloud representation serves the purpose of giving the operator a good overview of the situation, since the network is too unreliable to work with simple videos. In addition, RGB cameras are used to provide additional visuals about details in the environment. To display the imagery of the environment a control interface on a computer is used. Controls of the robot were partially automated, also due to the network condition. The operator can give commands on where to move or how to change the surrogate's arm pose, while the surrogate will determine how to execute the command. With this prototype, the team managed to successfully perform the task of unplugging a cable from a socket, whose position was initially unknown and plugging it into another socket of unknown whereabouts. Furthermore, tasks like opening doors or closing valves can be reliably performed using the prototype.

There are basic telepresence approaches that are already commercialized and used in everyday life and for remote office in work environment[8] (see 2.11). *Beam* telepresence robots are simplistic mobile telepresence robots, whose purpose is the embodiment their operator at a location to allow for communicative interaction with the environment and agents that are present there. For proper embodiment, live image and sound of the operator are recorded and played back via a display and speaker on the robot. The surrogate can be easily navigated through home, office and similar environments to make the operator mobile. Live sound and image of the environment are recorded by

[8] *Beam* Telepresence Robot, Retrieved via https://goo.gl/U8ePz2 on 29/11/2017

Figure 2.11: Beam Standard mobile telepresence robot. Retrieved via https://goo.gl/tppc1E on 29/11/2017.

the surrogate and then played back for the operator. Controlling and visualization of the surrogate environment are handled via either computers or smart phones and their connected displays. While the *Beam* telepresence robots allow for remote interaction with agents and environment on the communication level, there is no actual physical interaction. Furthermore, due to the use of conventional displays there is only a low level of immersion for the operator. On the upside, this system does not risk the induction of motion-sickness, as there is no head mounted display involved. The *Beam* system is thus a simplistic telepresence system suited for remote work that requires only communicative skills without risking any implications for the operator's well being.

Figure 2.12: *DORA Platform* mobile telepresence robot. Retrieved via https://goo.gl/9FKVPc on 29/11/2017.

The *DORA Platform* takes immersion to the next level[9] (see 2.12). DORA is made for basic navigation, not implementing any interaction capabilities at this point. While still depending on live video for visualization, the DORA system uses a head mounted display and head tracking to track any movement of the operator's head and has the surrogate mimic the movement precisely with low latency[10]. DORA uses a stereo camera to capture stereoscopic image, transmits it via network and displays the image on the head mounted display. This synchronization of the operator's head movement and vision with the surrogate's camera pose and recorded image allows for a high level of immersion, which is desired for better situational awareness in telepresence and teleinteraction [1]. The system uses the radio link for short range telepresence and WiFi or the 4G network for longer ranges, when communicating between operator and surrogate. While a delay of 70ms is slightly too high for an optimal VR experience, the developers are further optimizing DORA to reach the bounds for optimal immersion without risk of motion-sickness.

2.5 3D Reconstruction and Realtime Reconstruction

During recent years, low-cost 3D sensing hardware for scanning of real world environment has become available. Very prominent examples are the Kinect Fusion[11] and Google's Project Tango[12]. This hardware allows for live scanning of the real world environment and translation of the scanned information into a virtual model of the environment. There are various kinds of 3D sensing devices, such as for example stereo cameras or time-of-flight cameras. A stereo camera can be seen in 2.13.

Figure 2.13: Stereo camera. Retrieved via https://goo.gl/5wj6iX on 02/12/2017.

Common 3D sensing devices find representative points in the environment using triangulation to create point clouds. There are different libraries available for creation and processing of point clouds [45]. Depending on the environment, point clouds may contain partially insufficient information resulting in noise or even gaps in the resulting

[9] *DORA Platform*, Retrieved via https://goo.gl/9FKVPc on 29/11/2017.
[10] Example video of DORA, Retrieved via https://goo.gl/C7kKvK on 01/12/2017
[11] Microsoft, "Kinect Fusion", Retrieved on 02/12/2017 from https://goo.gl/x85U9Y
[12] Google *Project Tango*, Retrieved on 02/12/2017 from https://goo.gl/Sa13eu

representation. To be able to provide a proper representation and fairly realistic virtual environment for immersion, these gaps and noise are preferably avoided. More dense point clouds can be helpful in yielding better results, as they contain more detailed information or more information in general. Also, techniques for correction [17] and better exploitation of the available information [26] have been proposed to achieve a virtual space that is closer to reality.

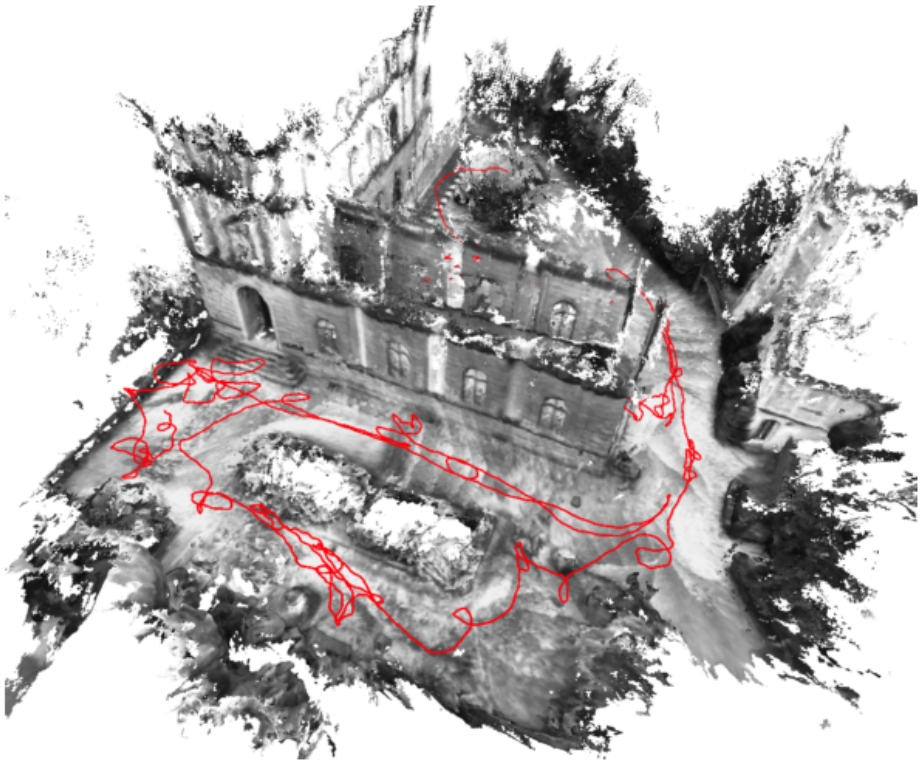


Figure 2.14: 3D reconstructed environment [46].

For the purpose of telepresence and teleinteraction, creation of a virtual environment from the surroundings of the robot can be an important aim [12]. As the robot is mobile and should be able to navigate freely through a possibly unknown environment, it is not possible to reconstruct the environment beforehand and just use the resulting model. The use of pre-scanned virtual environment models would require a limitation of the area, where the robot can move. Therefore, high-performance real time solutions will be needed for creation of the virtual environment in unknown environments [17].

Moving a 3D sensing device, while scanning its environment requires tracking of its position and orientation. The term *Simultaneous Localization and Mapping (SLAM)* describes the task of keeping track of the pose of an agent, while he moves through and maps an environment. A central problem of this task is the detection and correction of *drift*. While sensors will keep track of how the agent will move relative to its origin, inaccuracies and measuring errors will offset the tracked pose from the actual pose of the agent in the environment.

Since the robot carrying the 3D sensing device is moving, there is a continuous stream of new input on the surroundings available. This input can be used to extend the

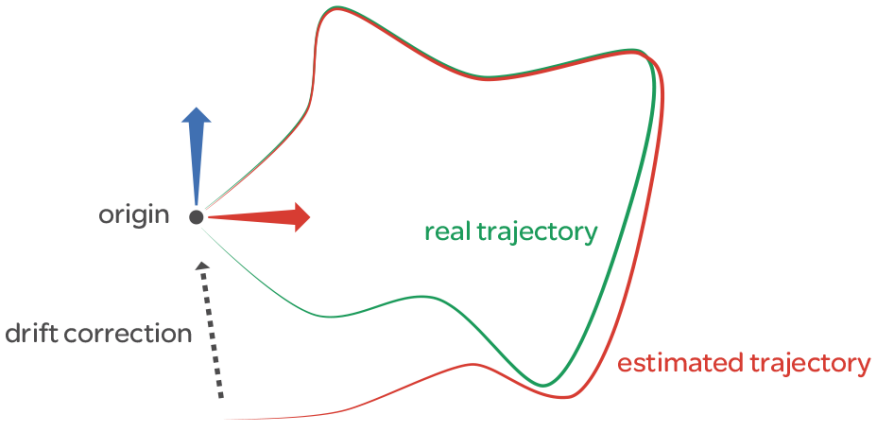


Figure 2.15: SLAM and drift correction. Retrieved via https://goo.gl/CHomsX on 02/12/2017.

environment, but also to correct errors and drift in the already scanned environment [23, 54] (see 2.15). Different research groups have tackled this issue of combining SLAM with 3D reconstruction. A continuous correction of the environment, however, has shown to sometimes be too time-consuming to give a true real time experience [13]. It was shown that real time correction of errors is indeed feasible. Prototypes for real time error correction have been proposed using the Kinect Fusion for medium sized indoor scenes [8] and using Google's Project Tango for large-scale outdoor scenes [46].

3 Concept

The goal of this thesis is to come up with a way of countering the high network induced latency in long range teleinteraction and the joint risk of simulator sickness, when utilizing head mounted displays. We suggest an approach to teleinteraction that utilizes live 3D reconstruction to visualize a surrogate's environment instead of commonly used video streaming to do so. To support the visuals, live video is additionally recorded and streamed to the operator in an augmented reality fashion. Given that the environment data of a virtual reconstruction still has to be transmitted over the network, the actual latency cannot be avoided. By using a persistent 3D reconstructed model of the environment for visualization, it is possible to explore this virtual environment without any delay once it has been received. This way, the network latency's contribution to the operator potentially experiencing simulator sickness can be avoided.

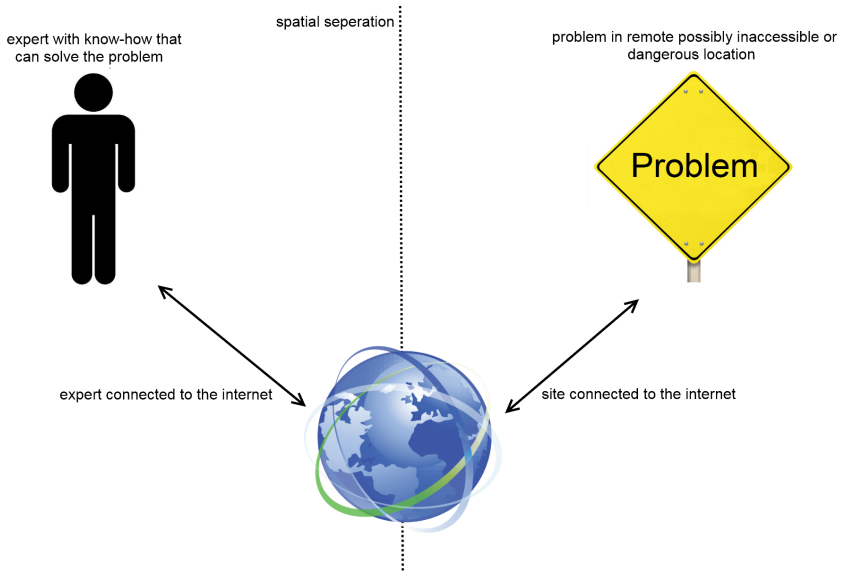


Figure 3.1: General problem of a scenario suited for application of teleinteraction.

The basic structure of scenarios that are suitable for the application of teleinteraction usually consist of a problem located in one space and an agent that can and should solve the problem located in a different space. While the agent could travel to the problem's space to solve it in person, there are sometimes reasons, why this cannot be done or would not be optimal to do. Such reasons might be inaccessibility of the problem space, immediate dangers for human agents in that space or simply the inefficiency of having to send a human agent to the site to solve the problem. If both locations are connected via a network (usually the internet), it is possible to attempt to solve the problem remotely (see 3.1). Given a problem that requires physical presence of an agent, some

sort of surrogate robot could be deployed to solve the problem under instruction of its operator (see 3.2).

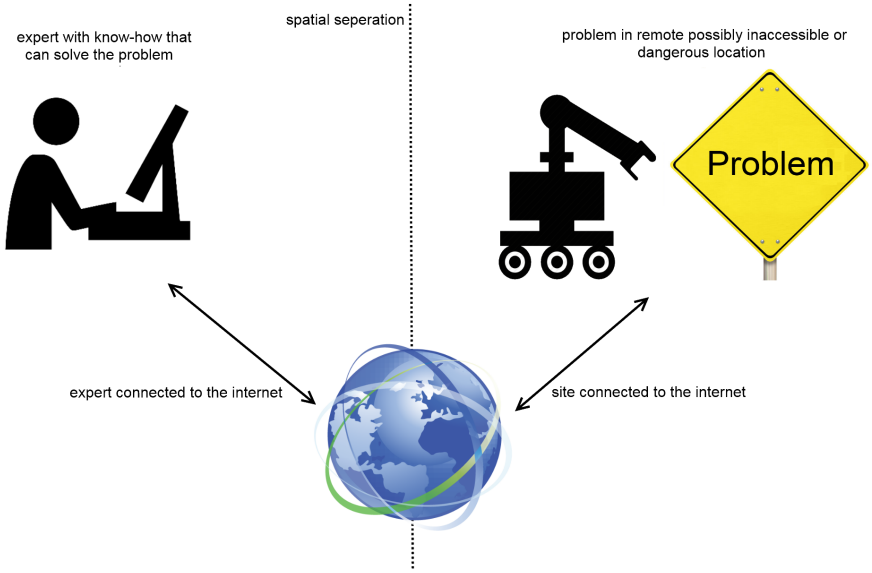


Figure 3.2: Teleinteraction scenario with involvement of a surrogate.

There are three main parties involved in teleinteraction: the human agent or operator, located in a space that is separated from the problem site; the robot surrogate, located on site of the problem that is to be solved in the context of the teleinteraction; and the network, connecting the spaces of operator and surrogate to allow for communication between the other two parties.

The core idea of teleinteraction is to to physically embody the operator in the target environment [15]. Thus, a surrogate with the necessary capabilities has to be available in the environment, where he should interact. The surrogate will capture data about the environment that can be visualized to the operator. Given a command, the robot will act accordingly to follow the instructions of its operator. For communication the surrogate is connected to a network.

The network connects the surrogate to its operator to enable transmission of data in both directions. The network connection should preferably be reliable, while optimally inducing minimal network latency.

The operator controls the surrogate while perceiving the surrogate's environment from its perspective. Interfaces for visualization and control are provided to the operator for this purpose.

There have been many simulations [22, 51, 11, 34] recently that try to reproduce situations and scenes, where a human agent has to perform a task or solve a problem. In case of a simulation this task is not remotely located, but located in a virtual environment. To give the human agent a better understanding of the problem and the situation, the use of virtual reality and head mounted displays has also become a popular approach to enhancing the realism of such simulations. Based on the similar purpose of both teleinteraction and simulation systems, parallels between the requirements and design of such systems can be drawn. The main difference from teleinteraction is that in simulations, while being inaccessible, the target environment is usually not remote. The

environment is often only virtually created and managed, without any real environment involved. While this introduces problems like the creation of a realistic experience or the creation of coherent physics in the context of the simulation, there are also major advantages to completely virtual environments. A virtual environment can be stored and communicated with locally, without or without major involvement of networks. Thus, the central issue of network latency over long range is omitted.

Even though there is a big difference in what the actual challenge for the design and implementation of each system is, the required components and necessary communication between them is very much alike (see 3.3). Knowing this, architecture of existing simulation systems can be reused during the design of a teleinteraction system.

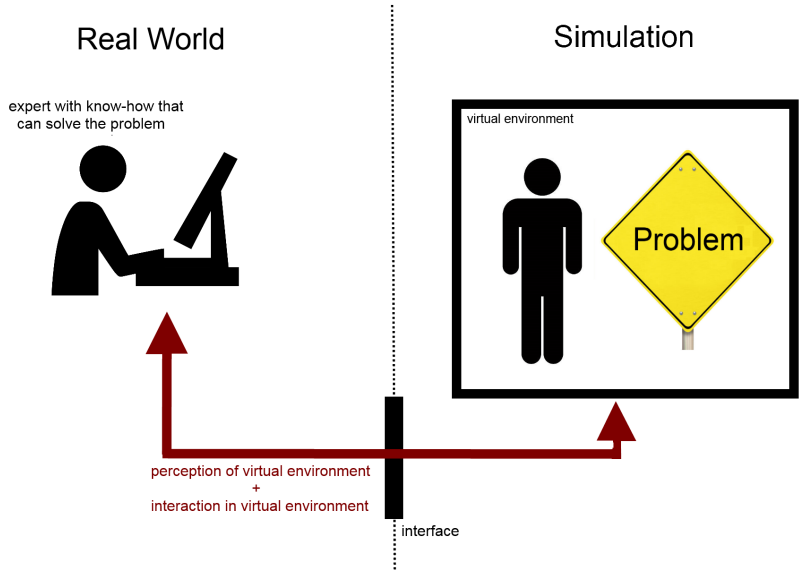


Figure 3.3: Simplified architecture of a simulation system based on the architecture of SIGVerse [51].

3.1 Architecture of the Teleinteraction Prototype

The requirements and features of the described teleinteraction system result in the conceptual architecture shown in 3.4. In the following, each component of the system, as well as the interfaces between them will be discussed. This discussion will focus on the general structure of each component, tasks and responsibilities of the component for the system and expected challenges when integrating the component into the system. For the challenges, strategies to avoid arising problems will be suggested.

Human Agent

The human agent is the operator of the teleinteraction. His purpose is to use his knowledge and know-how to guide and command the robot surrogate in the remote location during the completion of the task of problem at hand. To do so, the operator of course requires detailed information about the problem or task and situation that the

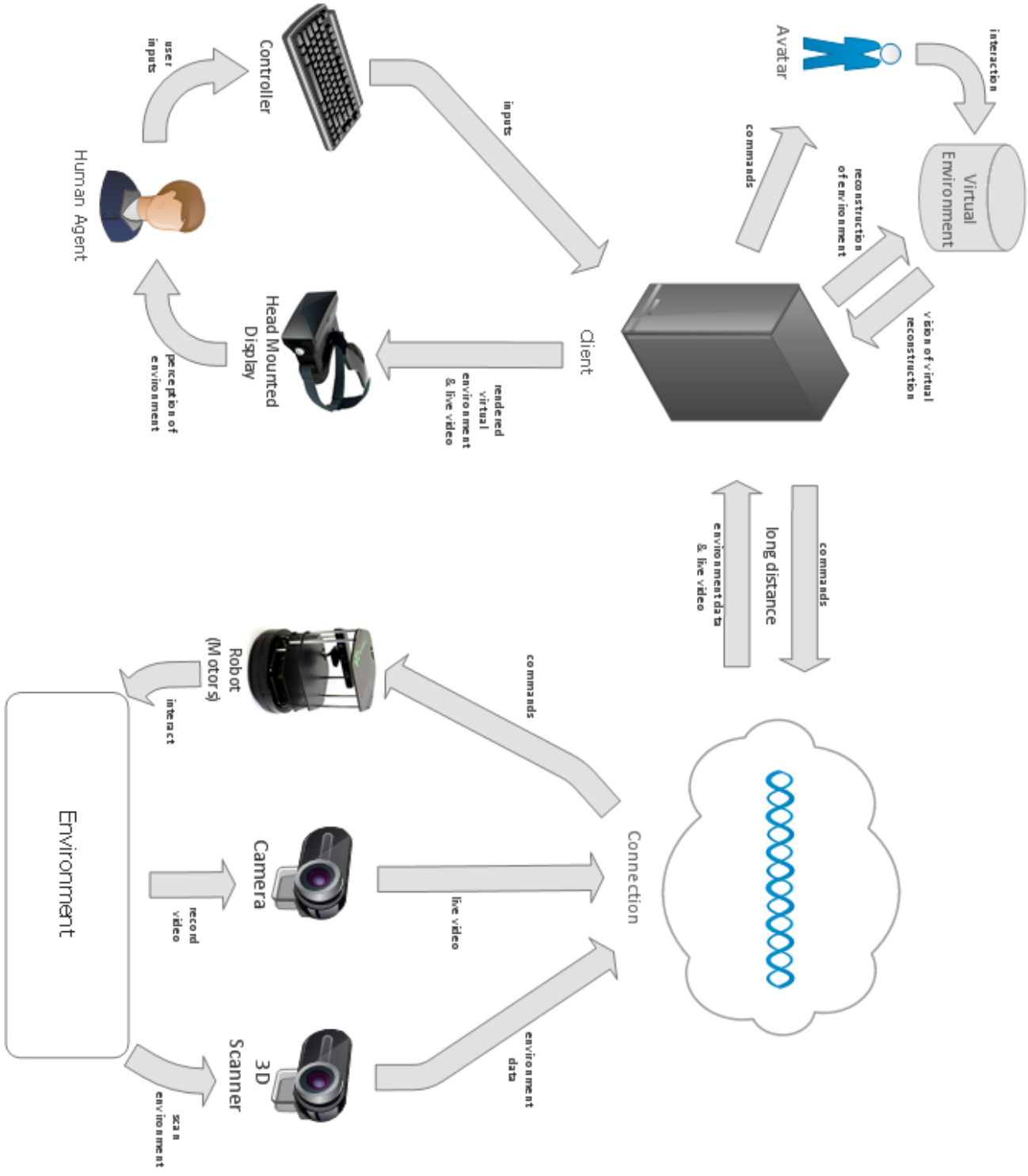


Figure 3.4: Flow diagram of the general teleinteraction system architecture.

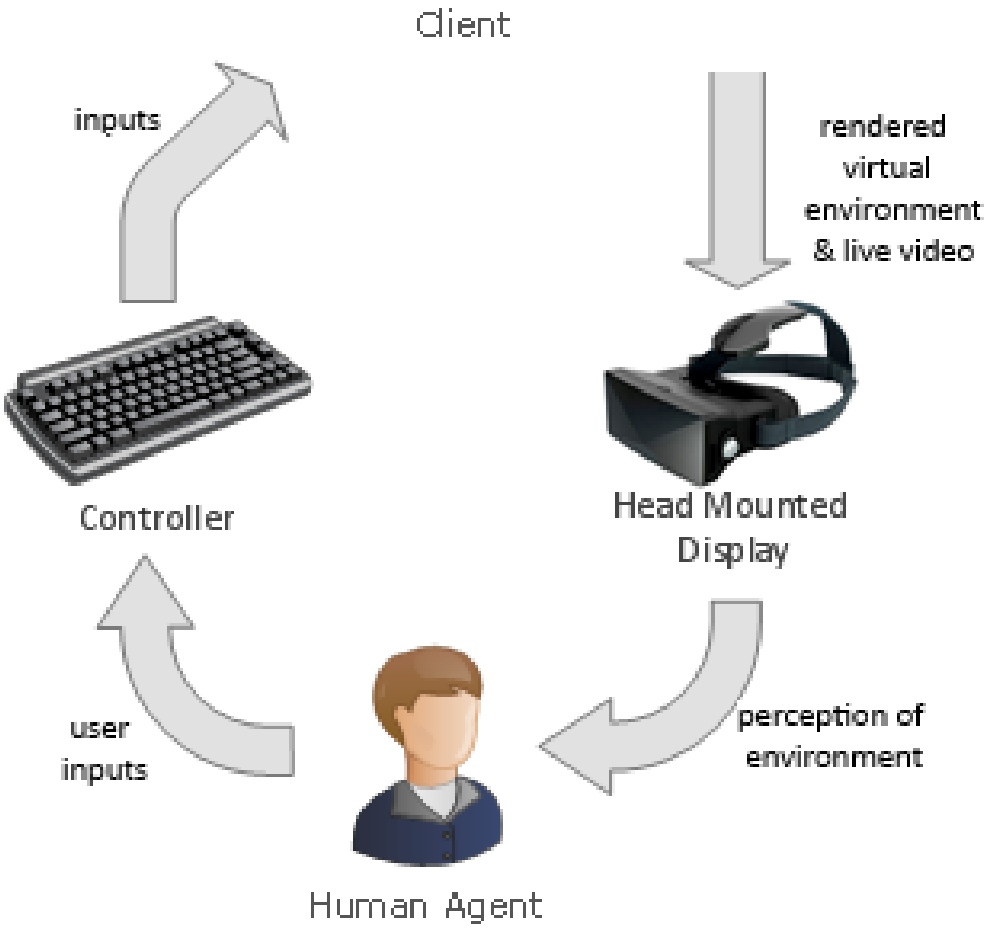


Figure 3.5: Flow diagram of the human agent and his equipment.

surrogate is in. Also, the operator has to be able to sufficiently control the surrogate to be able to have him carry out actions as required.

For visualization, a head mounted display is used. To receive imagery, the head mounted display has to be connected to a local machine running the teleinteraction client, who provides visuals of the virtual environment. These visuals incorporate the live video that can be recorded by the surrogate, if desired.

To issue commands to the surrogate, the operator is equipped with a controller. The type of controller is highly dependent on the capabilities of the surrogate, as all parts of the surrogate should be sufficiently maneuverable. Head mounted displays often provide head tracking data. This data can be incorporated when issuing commands to the robot. If for example the robot's head is rotatable, the operator's head pose might be used to control the robot's head pose. The operator's inputs are captured by the client, translated to commands and forwarded to the operator's avatar and the surrogate. While this concept directly integrates the controllers into the client, it might be well thinkable to separate controller and client. In this case, the controller could communicate directly with the robot via network.

There are no direct challenges involved, when integrating the human agent into the

system. However, it is the human agent who will become motion sick if visual latencies become too high, making him the indicator of a deficient long-range teleinteraction system.

Client and Virtual Environment

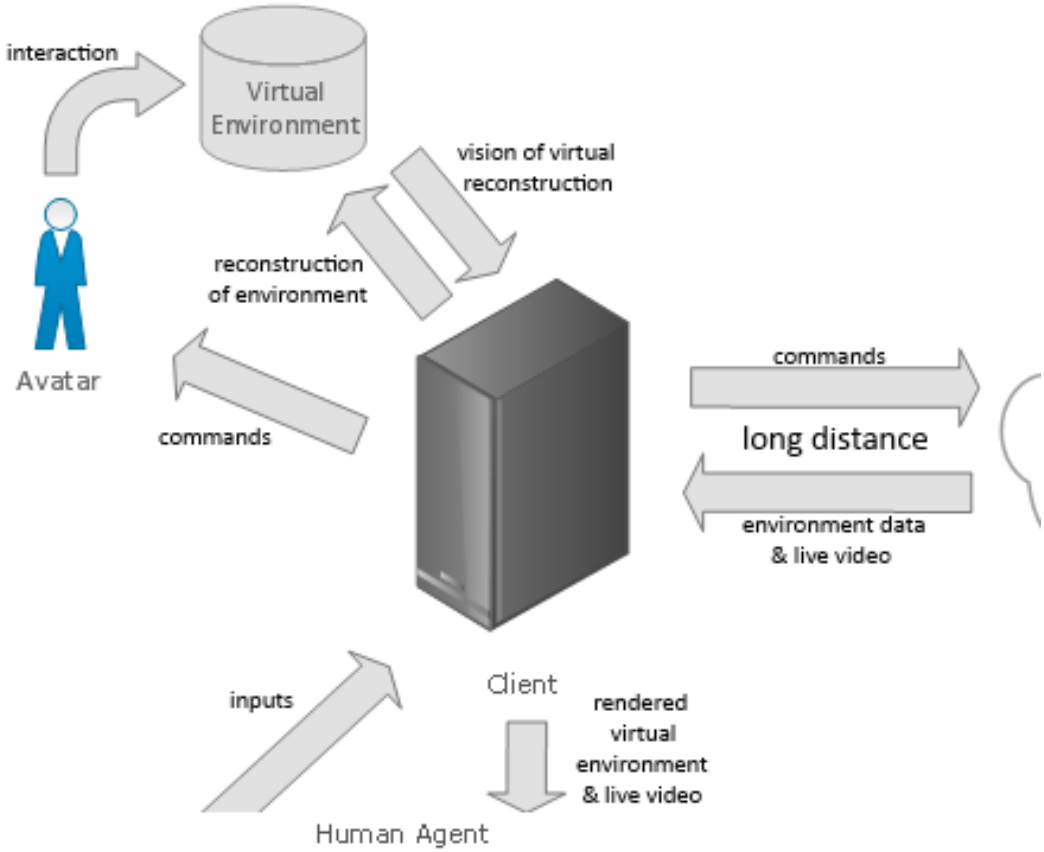


Figure 3.6: Flow diagram of the teleinteraction system's client.

The client is the interface for the operator to connect to the rest of the teleinteraction system.

To be able to provide a low-latency environment for the operator to explore, the client stores a virtual reconstruction of the surrogate's environment. Environment data that is gathered by the surrogate is received and processed to create the virtual reconstruction. Different reconstruction schemes can be used to convert the environment data to objects in virtual space that represent the real world. The resulting virtual environment can then be used to render imagery to the operator's head mounted display. Streamed live video is added to the operator's field of view if required. It is important to render said imagery coherently with the surrogate's real position and the operator's head pose to avoid confusion. An avatar mimicking the surrogate's actions is placed in the virtual environment for this purpose.

The client's contribution to a functioning system is the processing and transmission of inputs and the displaying of the surrogate's environment. The challenge resulting from

this role is the synchronization of the real world and the virtual environment. An offset between those, e.g. the surrogate being in a slightly different position than the avatar relative to the environment, can have negative impact on the success of interaction with the environment. This offset can be caused by insufficient precision in the virtual environment or missing information (see 3.7), but also due to the many kinds of latency that are involved (see3.8).

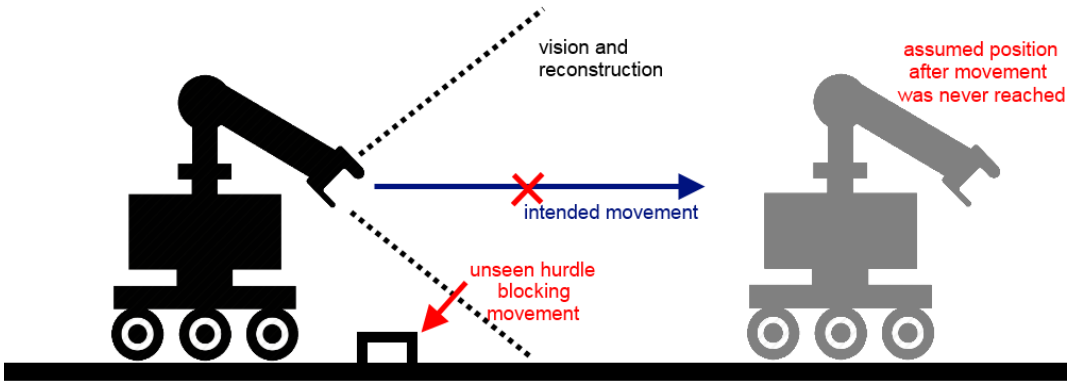


Figure 3.7: Offset between virtual and real environment due to missing information.

To give an example of how offset can be introduced due to missing precision or missing information of the virtual environment, imagine a robot that should perform a forward movement. Assume that a small hurdle that the robot cannot pass is placed in front of the robot's initial position. The hurdle is too close to the robot to be in the field of vision of cameras and 3D scanners. Even though the environment is reconstructed and can be taken into account when predicting a forward movement, the existence of the hurdle is unknown. While the system will assume that the robot can just move forward, the hurdle will block the robots path, keeping him in place. Thus, the robot will not have moved to the predicted point, but be stuck in front of the hurdle, creating an offset in the position of the avatar and the real position of the surrogate.

Motorical latency is a good example to show how latencies can introduce an offset between the virtual and real environment. Assume that the operator gives the surrogate the command to move forward with a constant speed v for some time t. The avatar inside the virtual environment is set to the respective velocity and travels at constant speed for the specified time. The actual robot might have to accelerate first with an acceleration a for some time t_{accel} to reach the specified speed, resulting in a part of the movement being executed with slower speed. This will result in an offset s_{off} of the travelled distances:

$$
\begin{aligned}
s_{avatar} &= v * t \\
s_{robot} &= \frac{1}{2} a * t_{accel}^2 + v * (t - t_{accel}) \\
s_{off} &= s_{avatar} - s_{robot} \\
&= v * t_{accel} - \frac{1}{2} a * t_{accel}^2
\end{aligned}
$$

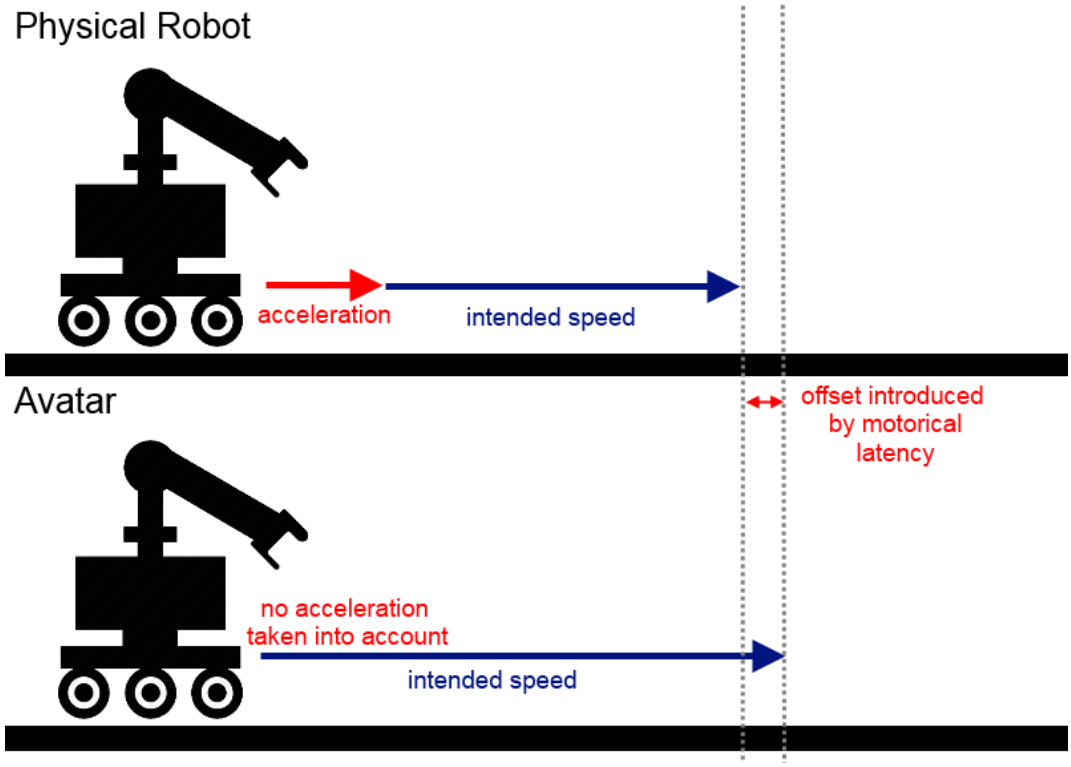


Figure 3.8: Offset between virtual and real environment due to involved latency.

A similar effect can emerge due to inertia, when stopping a movement. Of course, inertia and motorical latency can be simulated and incorporated into the avatar's movements. However, it can be hard to impossible to exactly recreate real conditions, causing small offsets that keep adding up.

As shown, different factors introduce an offset between the avatar in the virtual environment and the surrogate in the real world. During a teleinteraction scenario, any offset will affect all future actions and interactions that are to be carried out. Each action could possibly add more offset, bringing avatar and surrogate completely out of sync. The operator's perception directly influences his reactions to the environment and interaction with it. If this perception does not match up with reality, it is hard or even impossible for the operator to act appropriately. It is therefore important to make sure that avatar and surrogate are synced. One way of doing this is to track the robot's actual position in its space and incorporate a way of reporting position feedback. This feedback can then be used to adjust the avatar's position to reality to reduce or remove offsets that were introduced over time. To avoid offsets at all times, feedback should be constantly available. This problem is a prominent core problem from the area of SLAM and will be further elaborated later on.

Connection

The connection is not a component of the teleinteraction system per se, as its only purpose is transmission of data in both directions. Depending on how operator and surrogate are connected, the connection might only consist of the network. In cases,

where data cannot just be directly transmitted from one side to the other or if it needs to be processed along the way, additional components could be added to the connection.

The core responsibility of the connection is the reliable deliverance of data between client and surrogate. As there is no completely reliable transmission of data in networks like the internet, it is crucial to use reliable transport protocols to guarantee the reception of important data. This becomes especially important over long distances, where the risk to lose data increases. While it might not be a major problem to lose some of the environment or video data that is sent from the robot to the client, dropping commands and inputs could lead to dangers in the robot's environment for other agents or the robot itself. For example, the command to stop the robot before he injures a human agent in his vicinity is a command that should under every circumstance be delivered to the robot reliably.

The connection is the component of the teleinteraction system that introduces the high network induced latency to any communication between operator and surrogate. As such, it is of critical importance to consider the effect of latency on any data that is sent via the connection.

Robot

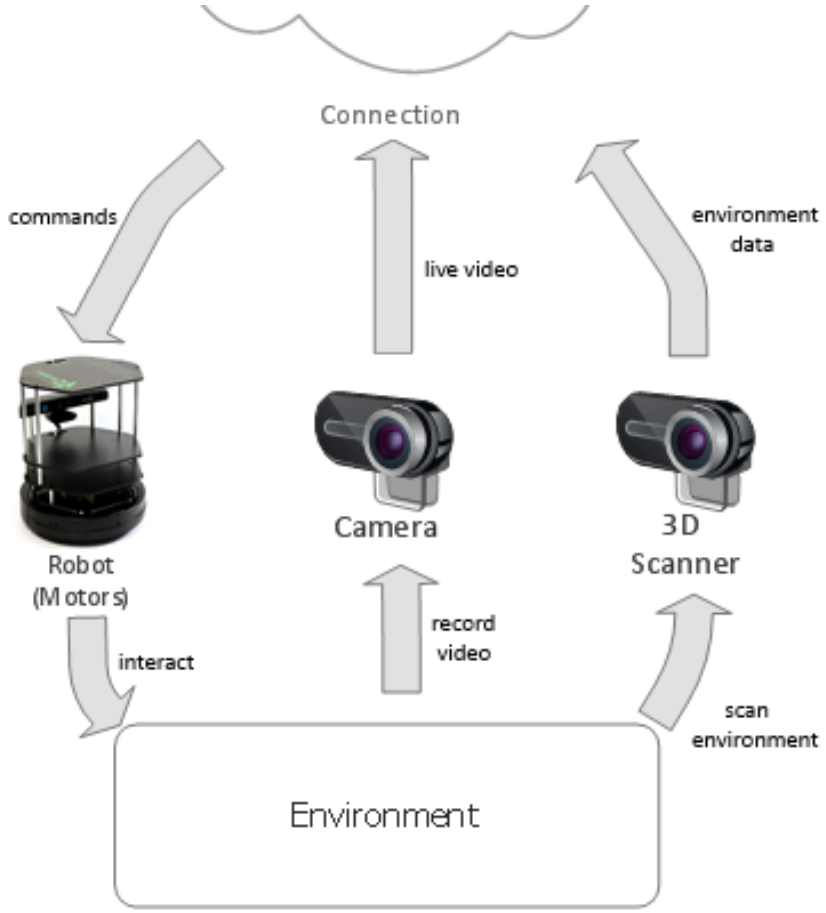


Figure 3.9: Flow diagram of the robot and its equipment.

The robot is the surrogate that embodies the operator in the remote environment. He grants the operator a physical body to act and interact in the environment. Depending on the purpose of the teleinteraction, the robot might have different capabilities, physique or equipment that allows him to perform the task that is at hand in a given scenario.

The robot has central components that are shared in any scenario, as they make up the underlying functionality. As the robot needs to be able to move to act in the operator's stead, robots contain motors to move their parts to the full extent of their physical capabilities. When a command from the operator is received, this command is translated into movements by actuating the respective motors.

To provide the operator with visuals, the robot is equipped with cameras and 3D scanners. While the teleinteraction system is running, these tools record information about the surrogate's environment, so the operator can grasp the situation and react appropriately. An elaborate explanation of the role of these will be given in the next section.

A challenge that can arise when integrating a robot into the teleinteraction system is the translation of commands into physical actions to carry out. An appropriate control scheme needs to be available, covering eventualities that are implied by the delay in the long-distance teleinteraction setting. For example emergency stops that precede received commands might be favourable to avoid harming other agents or the surrogate himself. Partial automation of surrogate actions could support successful interaction, even with delayed execution of commands.

Recording of Visuals and Displaying

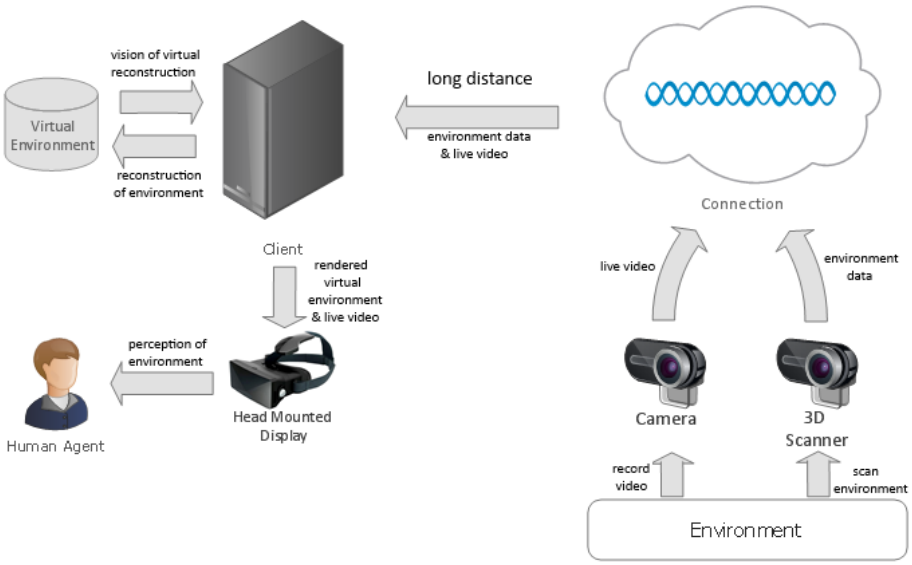


Figure 3.10: Flow diagram of video recording and 3D reconstruction devices.

The operator has to receive some kind of feedback about the environment to perceive the surrogate's environment and properly react to it. The components for recording

visual data together with the components for displaying recorded visual data are used to provide such feedback in form of imagery.

The different involved components are distributed over the whole teleinteraction system. Cameras and 3D scanners need to be affixed to the robot, as they are required to be in the remote environment, where data for visualization should be recorded. Once environment data is recorded, it can be processed if necessary and sent via the connection to the client.

After reception of the environment data at the client, the real world environment can be reconstructed virtually. The reconstructed environment can then be displayed to the operator at any point, as it is stored persistently. For the use of live video as a supportive media, an augmented reality approach can be used rendering the transmitted video as a small window in the operator's field of vision.

There are two challenges that are likely to arise for the visualization of the surrogate's environment. As a head mounted display is used, there is a risk to induce motion-sickness to the operator. While the approach of reconstructing the environment locally avoids major network latencies that would interfere with the use of for example video streaming, there are still other types of latency involved, such as processing latency. It has to be ensured that the performance of the client is sufficient to render the the virtual environment appropriately. Furthermore, to get a good understanding of the situation and the environment, the reconstructed environment should meet certain levels of quality. Camera and 3D scanner hardware, as well as the reconstruction scheme that is used, have to be chosen accordingly.

Regarding the avoidance of simulator-sickness, there are additional practices that have shown to positively influence the VR experience. As explained, simulator-sickness is caused as an effect of the human agent's sense of stability being disrupted by discrepancies in the visual perception of the agent's environment.

By creating fixed reference points in the operator's field of vision, the stability in VR can be improved. Overlaying a fixed HUD over the virtual environment can provide such fixed reference points. Another option is to have the avatar in the virtual environment look at the virtual environment through a window or frame from within some sort of vehicle. The VR movie *Sonar* used this technique to allow for rapid movements of the avatar through the virtual environment[1]. The illusion of the operator himself staying still, as he does in reality, is created while the vehicle is being moved.

To avoid discomfort and simulator-sickness, movements of the avatar should be matched with the movements of the agent wearing the HMD to avoid discomfort. Given that the operator usually has limited space to act in (due to the HMD cables or simply due to lack of physical space) while the virtual environment might cover big areas, mapping the operator's movements to surrogate movements in virtual reality can become difficult. *Valve*'s game *The Lab* sometimes provides the player with a big virtual world to explore, while the player possibly only has some square meters to move. To still allow for exploration of the virtual world, the player can teleport to any accessible point on the ground. Since teleporting is obviously not an action that the player can perform in the real world, this gap has to be bridged. Valve has implemented a blinking effect that is used, whenever the player teleports[2]. Using this effect, the illusion of the player

[1]View through a window (at 3:09), Retrieved via https://goo.gl/U3Bu9c on 12/12/2017.

[2]Blinking effect when teleporting the avatar in VR (at 2:22), Retrieved via https://goo.gl/4DzTJi

closing his eyes in one location and awakening in another location is created, weakening the discrepancy between real world and the virtual world, as the player is standing still in both.

3.2 3D Reconstruction

The use of 3D reconstruction to create a local and persistent representation of the surrogate's environment is the core idea of the teleinteraction approach presented in this thesis. Using a locally reconstructed virtual environment will allow to display this environment without any involvement of network transmission. This way, the high end-to-end latency in long-range teleinteraction scenarios can be avoided that might otherwise induce simulator-sickness.

There are different techniques and devices that can be used to reconstruct an environment virtually. As teleinteraction requires real-time reconstruction of the surrogate's environment, devices like *FARO FOCUS* laser scanners[3] are not fit for this application. While these devices provide a highly detailed scan of up to large areas, devices that scan the environment and return data on-the-fly are required to reconstruct parts of the environment as early as possible. Providing the operator with visual feedback as quick as possible is crucial, when facing a yet unknown part of the surrogate's environment.

As already mentioned, the problem of construction of a virtual representation of an environment, while navigating that environment is called *Simultaneous Localization and Mapping (SLAM)*. While navigating the target environment with the surrogate, there is a constant stream of new environment data to be recorded and incorporated into the already reconstructed virtual representation of the environment. The on-the-fly reconstruction of the surrogate's environment is therefore a SLAM problem.

The two main purposes of the SLAM problem at hand are the creation of a map of the surrogate's environment and the estimation of the the surrogate's pose at all times. In the context of teleinteraction, the map creation is equivalent to the reconstruction of the environment. Instead of estimating the pose for the surrogate itself, the pose of the 3D sensing device that is used by the SLAM algorithm is tracked. Given the pose of the 3D sensing device, which is attached to the robot, the surrogate's pose can be deduced and used as well.

Since the reconstructed environment as the main component for visualization is the basis for the operator to decide how to react to the environment, it is sensible to keep it free from errors and as precise and close to reality as possible. Therefore, any errors or drift that are introduced due to inaccuracies or measuring errors of the 3D sensing device should be fixed, if possible. Drift correction and loop closure should therefore be implemented by the applied SLAM algorithm to minimize mistakes and improve the quality of the 3D reconstruction.

Once a SLAM algorithm has been implemented and the camera's pose is available at any point, depth information can be translated to points in space. Each pixel of a recorded depth image can be used to calculate the position of a point based on the depth data and camera pose. By calculating points in space for all the valid depth

on 12/12/2017.

[3]FARO FOCUS laser scanners, Retrieved via https://goo.gl/NkRgES on 12/12/2017

information and storing them together, point clouds are created that can be used to reconstruct the recorded environment.

There are different ways to recreate an environment from points inside of it, some of which are point clouds themselves or meshes created from these point clouds.

Point Clouds

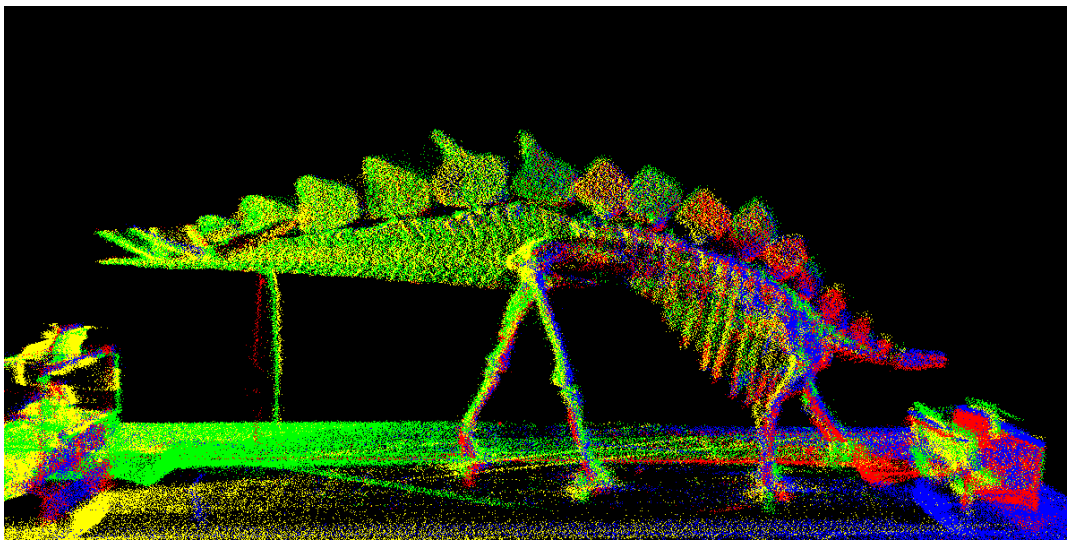


Figure 3.11: Visualization using a point cloud, Retrieved via https://goo.gl/kFrJDt on 12/12/2017.

The simplest way to visualize an environment after scanning is to visualize each point of the recorded point cloud. By visualizing the points in the same space, while keeping their position relative to each other, the environment will eventually become recognizable, if there is enough data, namely points available (see 3.11).

As there is a continuous stream of new environment data during a teleinteraction session, the point cloud will continue to grow and eventually become quite big. Also a 3D sensing device might be really precise and thus create a lot of points in a short time. In such cases, transmission of a large amount of points over network might be inefficient, since a full transmission would take too long. Therefore it might be beneficial to think about a way to decrease the network load and thus shorten the transmission time. Possible approaches are segmentation or compression of point clouds.

While point clouds are a valid option for visualization, reconstructed environments can become hardly recognizable if there is too little points available (see 3.12). As the graphic shows, the sparse point cloud is somewhat see-through and might not seem like an actual object. A possible way to circumvent this problem is the use of meshes.

Meshing

A *mesh* or *polygon mesh* is a collection of vertices, edges and faces that connect points of an object to render it visible to the eye. Figure 3.13 shows the mesh of a dolphin. The surface of the object is represented by polygons, making the object's surface clearly

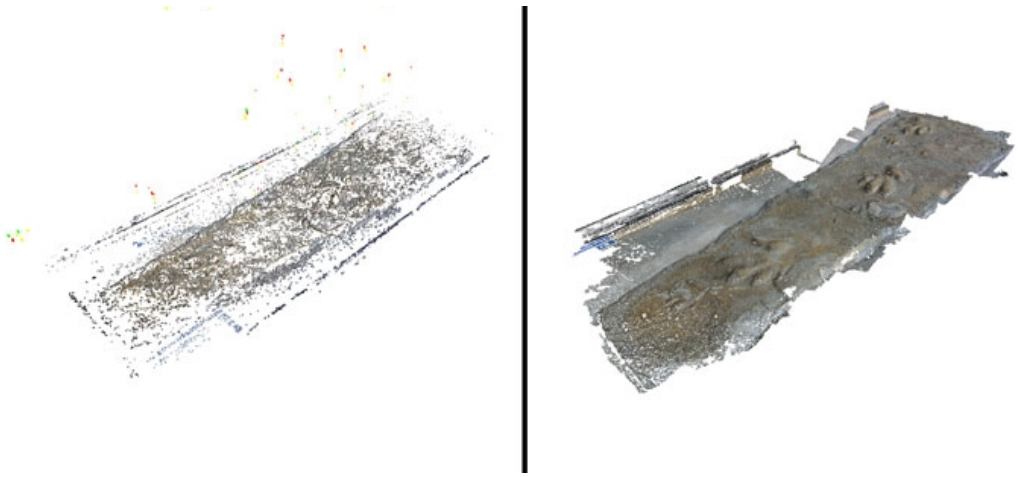


Figure 3.12: Sparse vs. dense point cloud, Retrieved via https://goo.gl/nLa1vg on 12/12/2017.

visible and the shape easily recognizable. This approach can be used to reconstruct the surrogate's environment virtually.

The creation of a mesh from environment data can be performed in different ways. It is possible to create a mesh from point clouds, however, there are alternatives. Devices like Google's *Tango Project* or Microsoft's *Hololense* are capable of scanning the environment and directly providing a mesh of the environment.

Just as with point clouds, there is a continuous stream of environment data that can be used to extend and improve the mesh. As the mesh will have to change over time due to correction and extension of existing data, this change has to be propagated to the client side, where the virtual reconstruction is built. Again, compression of this data might be favourable to avoid too high transmission times for the data. The *Tango Project* tackles this performance issue by dividing any created mesh into segments and providing tracking of changes in these segments to avoid redundancy.

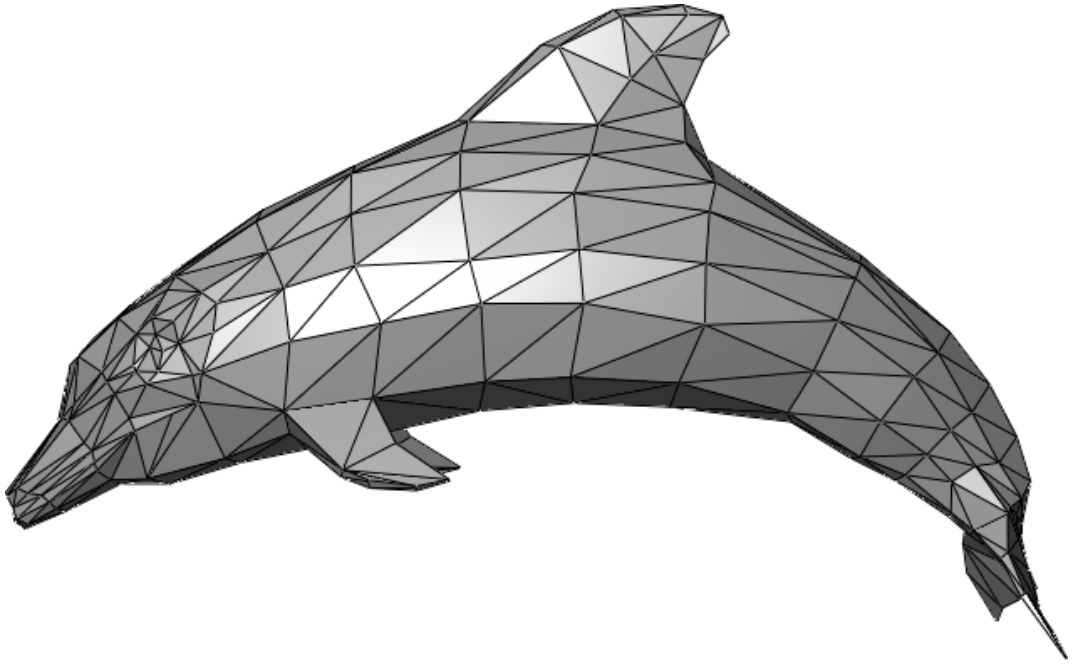


Figure 3.13: Visualization of a mesh, Retrieved via https://goo.gl/nAqW6K on 12/12/2017.

4 Implementation

This chapter describes the implementation of the conceptually designed prototype from the previous chapter. First, the components that were used during the development are introduced. Afterwards, the implementation process is presented step-by-step. Issues that were faced and design choices that were made in the process are elaborated on. Three approaches to scanning and reconstructing the surrogate's environment have been implemented to build a working prototype, as there were issues with some of the approaches.

4.1 Utilized Software

The prototype for the teleinteraction system will be implemented using a variety of provided frameworks, libraries and devices. Each of them will be introduced in this chapter, along with their capability, limitations and reasons why they were chosen for this implementation. The following is a list of the used components, to give a short overview:

- *The Thrift Eventbased Communication System TECS*: TECS is a networking toolkit enabling communication between software developed in different programming languages across different networks, platforms and operating systems.
- *Robot Operating System ROS*: ROS is an open-source meta-operating system made to be run on robots allowing for easy development of robot behaviour while abstracting from specific robot hardware.
- *Unity*: The Unity framework is a game engine that is commonly used in game developing and development of virtual reality applications.
- *SIGVerse/UnityROS*: SIGVerse is a client-server based simulation environment for Human Robot Interaction that serves as the environment for the RoboCup@Home competition. The included UnityROS library allows for integration of ROS into the Unity framework.
- *Several 3D Reconstruction Devices*: Different approaches using different devices have been used during the implementation. The respective devices will be introduced during the description of each approach.

TECS

The *Thrift Eventbased Communication Service (TECS)* is a networking toolkit developed by the researchers at DFKI Saarbrücken to bring together applications on different

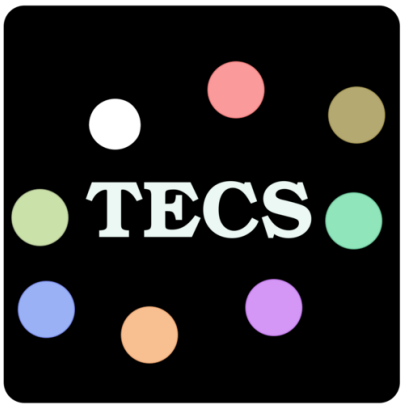


Figure 4.1: Thrift Eventbased Communication Service TECS, Retrieved via https://goo.gl/BWw9cJ on 13/12/2017.

systems using different architectures[1] (see 4.1).

The basic idea behind TECS is to enable easy and fast communication between different applications by providing core interface functionalities, such as *remote-procedure-calls (RPC)*, *publish-subscribe* and *message-passing*. TECS provides support for most common programming languages, such as C++, C#, Ruby and Python on Windows, Mac OS and different Linux distributions.

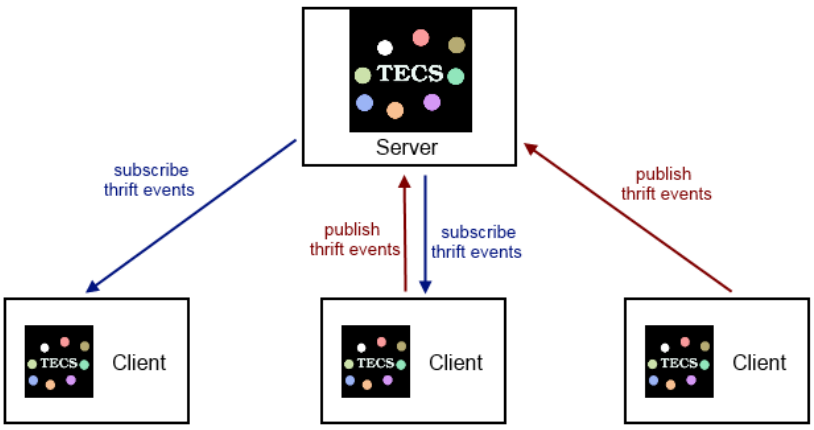


Figure 4.2: Thrift Eventbased Communication Service TECS, Retrieved via https://goo.gl/BWw9cJ on 13/12/2017.

As the name suggests, TECS is based on *Apache Thrift*, a framework that supports easy implementation of RPC-clients and servers by supporting the creation of the required data types, including their serialization and transportation, as well as invocation of remote methods. Thrift can automatically generate these functionalities based on simple type definitions. TECS expands this functionality by providing methods that implement message-passing and publish-subscribe (see 4.2). Networking with TECS

[1]TECS, Retrieved via http://tecs.dfki.de/tecs/ on 13/12/2017

can be done within a single line of code by connecting to TECS servers that will handle distribution of data between the connected clients.

For the proposed teleinteraction system, software running on many different platforms that are distributed over several locations, need to be integrated and be able to communicate and exchange data with each other. TECS is meant to do just that. The underlying Apache Thrift originates within *Facebook*'s backend and was meant to ensure highly scalable backend-service [48]. Working efficiently in a complex system such as Facebook's backend, Thrift has demonstrated its efficiency and applicability. Together with the fast extensibility and easy integrability of the framework, Thrift can be considered a suitable tool for handling the communication between the components of the teleinteraction system. TECS, which is based on Thrift to implement additional interface functionality, is thus also a valid option to implement communication in the teleinteraction system.

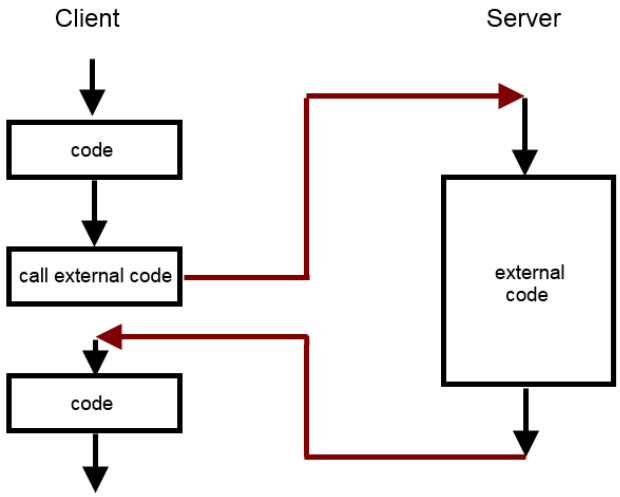


Figure 4.3: Concept of remote procedure calls.

Remote procedure call (RPC) describes the concept of a program calling a method in a different address space, but in the same manner as if the call was done and processed locally (see 4.3). Usually, clients perform such calls to get certain functionality executed by a server.

As the client in the teleinteraction system requires the robot and its equipment to gather the necessary data for the 3D reconstruction, the client has to delegate this responsibility to the robot. Since the gathering of data is a continuous process that keeps producing data, RPC is not a fitting concept to trigger that process. Since scanning of the environment should not be interrupted at any time to not lose data, the client would have to call the environment scanning method in the beginning and wait till the end of the teleinteraction session for the method to end. This makes RPC a suboptimal option for the implementation of teleinteraction communication.

Message passing describes the concept of a process passing data to another process that should handle sent data without invoking the actual code that is to be executed. The receiving process, which is usually running independently, decides on how to handle

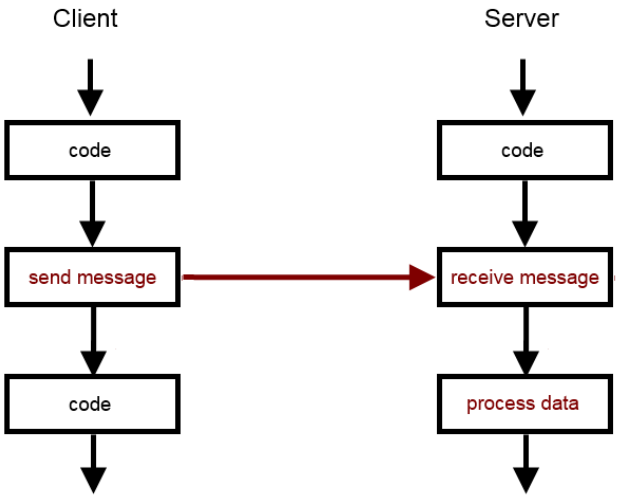


Figure 4.4: Concept of message passing.

the received data depending on its state and purpose (see 4.4).

As already mentioned, the teleinteraction client delegates the responsibility of gathering environment data to the robot and its equipment, while expecting the data as feedback. This data could be provided by the robot via message passing, as the robot can just send data and keep generating new data, while not concerning with what the client actually reconstructs from the provided information. This would make message passing a fitting concept for reconstruction purposes. However, for teleinteraction controls, where multicasting of commands to both the avatar and the robot is required, message passing will not suffice, as there are multiple components requiring the same data to be provided to them. The same is true for potential future scalability of the system to a multi agent system.

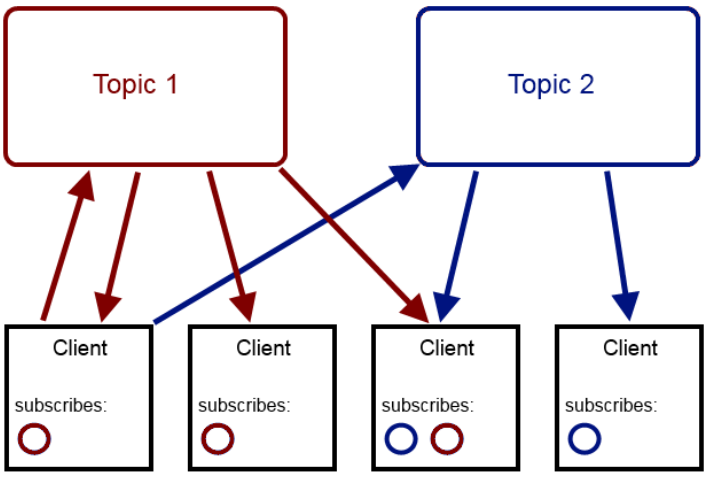


Figure 4.5: Concept of the publish-subscribe pattern.

Publish-subscribe describes a messaging pattern, where senders or *publishers* of messages transmit their messages not directly to a receiver, but to categorized groups of receivers or *subscribers* that are interested a category of message (see 4.5). Messages can be categorized topic or content based. If a message of a certain topic or with certain content is sent, all subscribers of the topic or content will receive that message.

Publish-subscribe is, in a sense, an extended form of message passing that involves multicasting of messages to whole groups and thus introduces better scalability. For the teleinteraction controls and a possible multi agent system, publish-subscribe solves the issues with message passing, as agent and components requiring the same data can simply be summarized as a group and be provided with the necessary data at the same time.

For this reason, TECS is used for the implementation of the teleinteraction system, as the provided publish-subscribe functionality is highly suitable for the required communication between the different components of the teleinteraction.

Robot Operating System

The *Robot Operating System (ROS)* is a framework for developing robot software. The framework allows for simplified creation of robust robot behaviour, even over many different robotic platforms, by providing helpful tools and libraries. Core features included in ROS are for example hardware abstraction, low-level device control, message-passing between processes and package management.

To perform teleinteraction, a surrogate robot is required to act and interaction with the remote environment in the operator's stead. For this purpose, the robot needs to be controllable via software, so it can be integrated into the teleinteraction system. Using the ROS meta-operating system, the robot will be provided with a software platform that will allow for high level communication with the robot and control on hardware abstracted level.

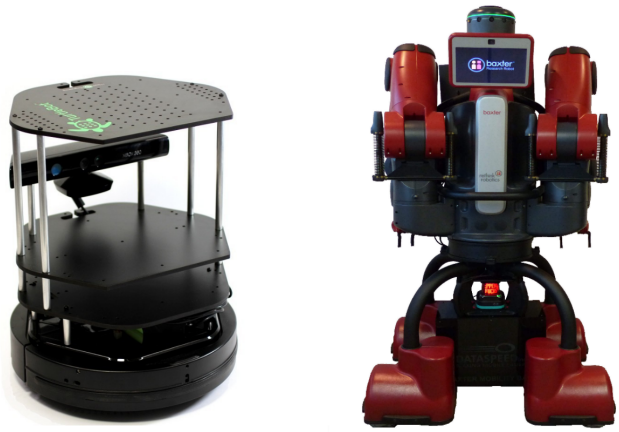


Figure 4.6: *Turtlebot* and *Baxter Mobility Base*, Retrieved via https://goo.gl/ib3UH6 and https://goo.gl/n6nZKW on 13/12/2017.

As ROS is the common standard framework for programming or working with robots,

most commercial robotic platforms support the ROS framework. So do the *Turtlebot*[2] and the *Baxter Mobility Base*, which were the available robots for the context of this thesis (see 4.6). Therefore, using the ROS framework for communication with the surrogate is the best option.

Figure 4.7: ROS Indigo Logo, Retrieved via http://wiki.ros.org/indigo on 13/12/2017.

There are several versions of ROS available. The different versions of ROS are divided into distributions, each representing a versioned set of packages. For this implementation, the *ROS Indigo Igloo* (see 4.7) distribution is used, as the current version of SIGVerse's UnityROS that will also be used in this thesis works with the ROS Indigo release. To avoid potential incompatibility, two more recent versions *Kinetic Kame* and *Jade Turtle* will not be used.

ROS Concepts

The idea of ROS is to provide an operating system for robots, where little programs called *nodes* can be executed. A robot can run multiple nodes, each possibly having a different purpose or providing a different functionality. Robot platforms usually come provided with nodes implementing their basic behaviour, like translating standard movement commands to movements for example. These nodes may be reimplemented by developers to adjust or change the basic behaviour of the robot. Additional nodes can be implemented and executed to add functionality to the robot's behaviour. Nodes that work on the same or similar tasks often are bundled to *packages* to organize the structure of the robot system better.

Communication between nodes in a ROS system works based on the publish-subscribe pattern. ROS utilizes topic based publish-subscribe messaging pattern to send messages containing data to other nodes in the system to hand over data or send commands. For

[2]Turtlebot, Retrieved via http://www.turtlebot.com/ on 13/12/2017.

actual use, message types have to be defined first and can then be sent throughout the system for a topic.

Additionally, ROS implements basic remote procedure calls in the form of *services*. These services are methods that are meant to execute code and return a result. By calling these services during the execution of a node, the node will do a remote procedure call to have the provider of the service calculate and provide a certain result for him. This result can then be used by the calling node for its own execution.

A way of communicating with sensors and motors has to be provided by the system to connect the ROS operating system with the actual physical robot. Therefore, sensors and motors that are connected to a ROS system provide their own topics that are used for communication and controlling of the hardware. In the case of sensors, data that is recorded, will be published in form of appropriate events by the sensors in a topic exclusively for reporting of the sensor data. Analogously for motors, commands like setting velocity are to be sent to a motor provided topic used exclusively for the controlling of the motor. As both sensors and motors might not only either produce or consume data, there might be other topics provided as well to control sensor settings or check motor feedback for example.

Unity

Figure 4.8: Unity Game Engine, Retrieved via https://goo.gl/qNk7YE on 14/12/2017.

Unity (see 4.8) is a cross-platform game engine that was developed by *Unity Technologies*[3]. Its primary purpose is to serve as a development environment for the development of 3D and 2D video games and simulations. Supported platforms include computers, consoles and mobile devices.

Due to Unity's open software architecture, it has become target for development of advanced media, as for example virtual reality and augmented reality. As those applications profit from efficient and powerful 3D graphics, many VR and AR platforms provide SDK's and interfaces for the direct integration of the hardware into Unity. Prominent

[3]Unity Game Engine, Retrieved via https://unity3d.com/de on 14/12/2017

examples are the *HTC Vive*[4], *Oculus Rift*[5], *Vuforia*[6] and *Microsoft HoloLens*[7]. As the HTC Vive is used in this thesis, it makes sense to use one of the prominent development platforms for VR applications, as for example Unity or its rival product *Unreal Engine*[8].

Unity supports the programming language C# , which provides a big collection of code, libraries and frameworks that can easily be integrated into the system. TECS as an essential component is also easily integrable into Unity. UnityROS, which is available as an interface between ROS and Unity, will allow flawless communication between the game engine framework and the robot. Therefore, Unity will be used for the implementation of the teleinteraction system. Additionally, Unity provides the so called *Asset Store*, where various kinds of existing code and tools are provided to support development, in case this is required. The Unity version that is used is *5.4.0f3*.

Basic Functionality

To develop an application in Unity, a *project* has to be created first. Projects are bundling settings, game worlds, code and other resources in a project structure to enable proper management of the applications components. Unity provides various build, performance and project management settings that can tweak the application that is to be created to the needs of the developers.

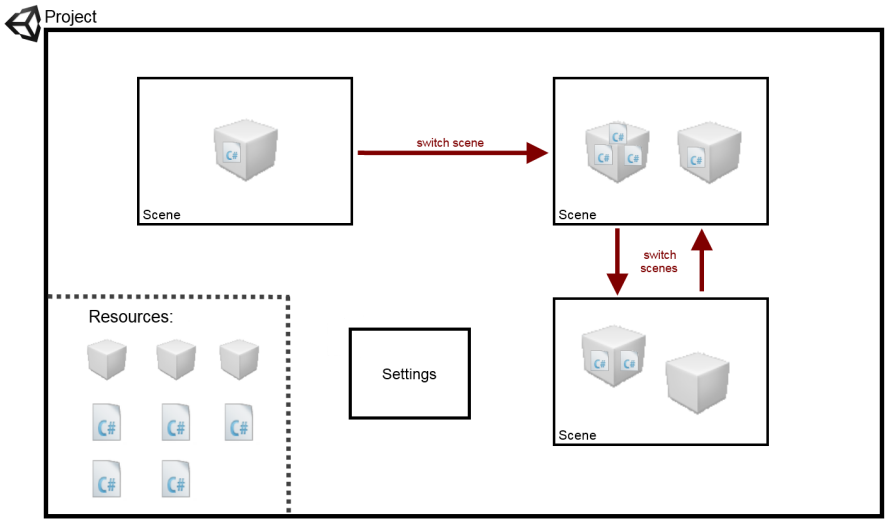


Figure 4.9: Basic Functionality of the Unity Game Engine.

The application itself is generally developed by creating *scenes*, game worlds in which game objects are created and managed and by attaching *scripts* to said objects that define object behaviour and game functionality (see 4.10). An application can have multiple scenes to switch between or even at the same time. Unity provides basic game

[4]HTC Vive Unity Support, Retrieved via https://goo.gl/68Xmvy on 14/12/2017
[5]Oculus Rift Unity Support, Retrieved via https://goo.gl/HHGw63 on 14/12/2017
[6]Vuforia Unity Support, Retrieved via https://developer.vuforia.com/ on 14/12/2017
[7]HoloLens Unity Support, Retrieved via https://goo.gl/mc2jy7 on 14/12/2017
[8]Unreal Engine, Retrieved via https://goo.gl/z7YBdh on 14/12/2017

objects as for example geometric objects or sprites that make up a lot of the game objects used in many projects. Advanced objects, such as virtual cameras, can be used to easily render imagery or play back sound. Game objects that exist inside scenes have attributes based on the their type that can be used to tweak their physics. Additionally, each game object can be assigned an arbitrary number of so called *MonoBehaviour* scripts to define the game object's behaviour. Each MonoBehaviour provides different method handlers to synchronize the execution of code with for example the creation of a scene or each new frame. Scripts can introduce additional properties to a game object, while having access to the game object's properties originating from either the object itself or other attached scripts, if publicly accessible. Using the modular design of simple behaviours and combination of different MonoBehaviours, complex functionality can be easily created for the different game objects.

Furthermore, it is possible to write classic code that is not and cannot be attached to any game objects. As Unity's main thread only executes code that is present in active scenes, this code is not directly connected to the game engine. It will in turn only be executed, if somehow referenced by MonoBehaviours that are present in active scenes. By calling respective methods or referencing required classes the code can be easily included. Analogously, external code in form of libraries or frameworks can be included into Unity, by adding them to the project structure.

MonoBehaviour Method Handlers

Unity provides a set of different method handlers[9] that allow for executing code at certain points in the program flow or for synchronization of code execution with the frame rate. Here is a non-exhaustive list of available method handlers and their triggers:

- *Awake()*: This handler is called, when the script instance is loaded. It is executed only once during its lifetime and timed before the *Start()* handler. Best practice suggests to assign initialization of variables or game state to this handler.
- *Start()*: This handler is called on the frame, when a script instance is enabled for the first time directly before any *Update()* calls. It is executed only once during its lifetime. Best practice suggests to assign initialization processes to this handler that depend on the finished initialization of other objects' *Awake()* handler.
- *Update()*: This handler is called once per frame.
- *FixedUpdate()*: This handler is called at a fixed rate matching the optimal frame rate. If the actual frame rate drops, the *FixedUpdate()* handler will be called at a higher rate than the actual frame rate approximating the optimal frame rate. When running a program without lag, *Update()* and *FixedUpdate()* handlers are called at the same rate. Best practice suggests to perform any physics related changes in the scene in this handler to avoid distortion or errors in the applications physics due to lags or dropped frames.
- *LateUpdate()*: This handler is called after all *Update()* handlers have been called for a frame. Best practice suggests to assign code, whose execution depends on changes made in *Update()* handlers to this handler.

[9]MonoBehaviour documentation, Retrieved via https://goo.gl/7ehLXw on 19/12/2017.

SIGVerse Simulation Environment and UnityROS

Figure 4.10: Concept sketch of SIGVerse, Retrieved via https://goo.gl/t2fgHK on 14/12/2017.

SIGVerse is a simulation environment developed for the simulation of Human Robot Interaction[10]. As the mechanisms of real-life intelligent systems are very complex, interaction of agents with their environment or with other agents requires knowledge from many peripheral fields. To get these different fields together, it is important to include an interdisciplinary aspect into research on this topic. This is time consuming and labour intensive due to the different natures of experiments in the respective disciplines involved. SIGVerse aims to serve as a simulation environment that combines dynamics, perception and communication simulation to ease and support future research. SIGVerse is designed with the concept of multi agent systems in mind, thus requiring an appropriate system architecture to scale the simulation environment with increasing number of agents.

Client-Server System

SIGVerse utilizes a basic client-server layout for the system architecture (see 4.11). Client applications can connect to a central server performing the simulations, connecting the different clients and verifying the integrity of the simulated interaction. The clients will only serve as an interface for the agent to interact with or within the system.

As seen in figure 4.12, there are several interfaces between the different components of the SIGVerse system [37].

To connect the human agent to the SIGVerse system, he is provided with a typical VR interface, including a head mounted display, an audio headset and respective motion capture devices. Wearing these, it is possible to capture the human agents actions, while playing back video and audio to him to enable perception of the virtual environment.

[10]SIGVerse, Retrieved via http://www.sigverse.org/ on 18/12/2017

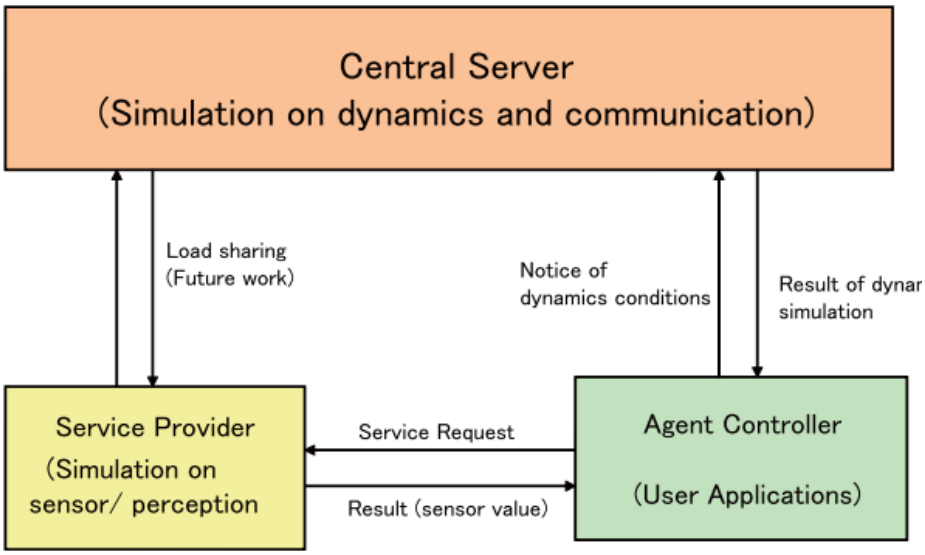


Figure 4.11: Client-Server Architecture of SIGVerse [22].

Input and output of media and data via the VR interface is handled by a locally running Unity client. Scripts are used to handle the different interface devices and join together in a local virtual environment that acts as the gateway into the SIGVerse simulation space for the human agent's avatar.

The client is connected to a Unity server that synchronizes and coordinates the different clients that join the system. The server creates the joint simulation platform that robots and agents connect to for their interaction purposes, handling and verifying the integrity of performed actions and their effects on the environment.

Beside clients that represent human agent avatars, robots can connect to the system as well, using their own Unity client. These Unity clients are similar, except that they do not have a VR interface for a human agent, but instead a ROS interface connecting them to a real or simulated robot. This interface is the *UnityROS* framework, allowing for integration of the ROS system into the Unity framework via network.

UnityROS

UnityROS is a library that was originally developed by Michael Jenkin for communication between the Unity framework and ROS[11]. The SIGVerse team adapted and extended the functionality of the original library to integrate into Unity easier and improve the performance.

UnityROS enables communication between a Unity program and the common ROS package called *ROSBridge*, more specifically a ROS node implementing websocket connections. The node turns incoming data into ROS messages and vice versa, when wanting to send ROS messages to the Unity program. Messages are sent and received together with the corresponding topic, so the message can be distributed on each side to only the subscribers of the topic.

[11]Unity ROS Project, Retrieved via https://goo.gl/bLc6Wy on 14/12/2017.

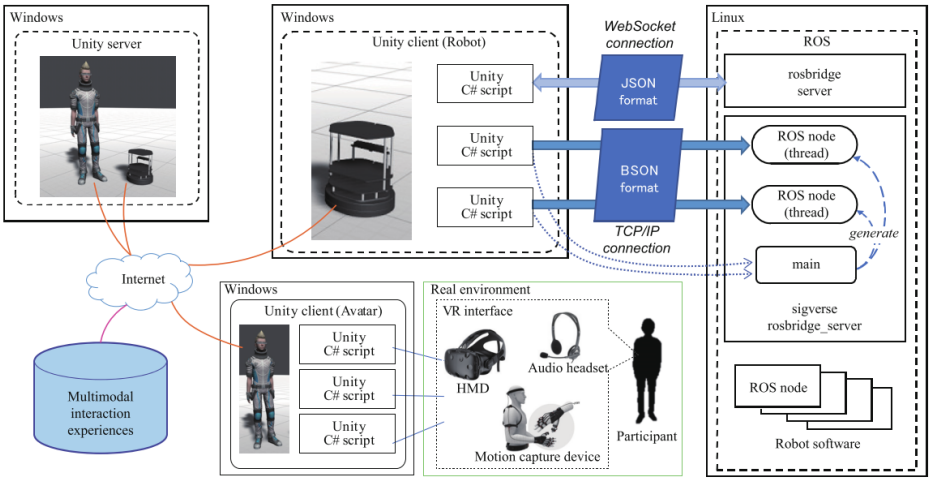


Figure 4.12: Cloud-Based Multi Agent System [37].

To be able to serialize and deserialize messages on the Unity side, each message type requires a class that represents its respective type in Unity. A generator has been added to SIGVerse to allow for automatic generation of C# classes from the ROS definition to avoid having to manually implement each of these almost identical classes. The generator uses a template and fills it with properties, serialization and deserialization code depending on what attributes are added to the message type by definition.

After all necessary message classes are generated, they can be added to the Unity project together with the library, if not yet available. UnityROS provides publisher and subscriber classes that can be used to create instances publishing and subscribing to any specified topic.

4.2 Building the Prototype in Three Steps

For a reasonable implementation process, it is sensible to implement the system modularly component by component. Sufficient testing of partial functionalities of the already implemented system, before proceeding with the extension of additional functionality eases prototyping and ensuring of proper functionality of the system. Therefore, this prototype was implemented in three steps.

The very first step is the implementation of the client side of the teleinteraction system. As it requires only little effort to simulate a simple robot in Unity's virtual space, the client side of the teleinteraction system can be fully implemented and locally tested without any involvement of robot hardware. The simulation that results from the first implementation step, will be an almost fully functional teleinteraction client, which only misses the local 3D reconstruction of an actual environment, since there is no physical robot to record data yet.

In the second step, the client side is extended by adding a physical robot to the system. This robot is only supposed to be controllable by the client at this point without any 3D reconstruction or visual feedback being performed. Thus, communication between client and robot via Unity and ROS are the main focus of this step, resulting in a

system that misses only the visual feedback to its operator. Given that an environment is known and does not need to be recorded and reconstructed any more, this prototype should already be applicable.

In the third and last step, the prototype will be extended by 3D reconstruction and video feedback for visualization on both client and robot side. This step finalizes the prototype making the implementation complete.

An overview of the partial prototype that is to be implemented is given during each of the three steps. The architecture, as well as the functionality of each component are elaborated on, including design choices that were made during the implementation. At the end of each step, an overview of already implemented and working functionality is given.

4.3 Teleinteraction with a Simulated Robot

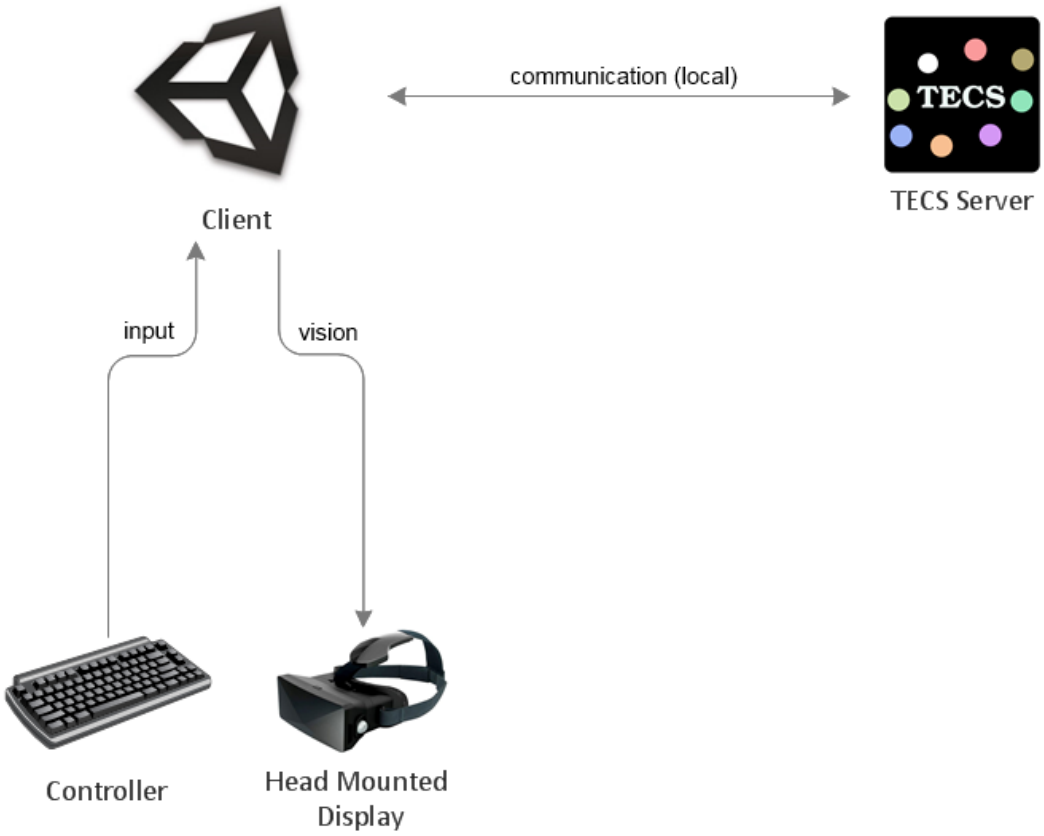


Figure 4.13: Prototype after completion of the first step.

The first step is to implement a client and get the client side of the teleinteraction system working. The surrogate robot is simulated in the virtual environment for testing purposes.

Figure 4.13 shows the architecture of the prototype that is implemented during the first step. The human agent is provided with an *HTC Vive* head mounted display and

a keyboard as controller. The Unity client is running on the operator's machine storing a virtual environment that should later represent the surrogate's environment. Inside the virtual environment, the operator's avatar, as well as the simulated robot act and interact as instructed by the operator's keyboard inputs. Commands to the simulated robot are sent via a local TECS server connection. Visuals for the operator are provided by rendering the virtual environment from the avatar's point of view and are augmented by interpreting the simulated robot's vision as live video.

To develop the client application, a new empty Unity project is created. The project is created with the 3D option, as virtual reality based applications require the 3D physics engine. Unity provides the option to create 2D games providing 2D physics by default. However, this is not desirable in this case.

Virtual Environment

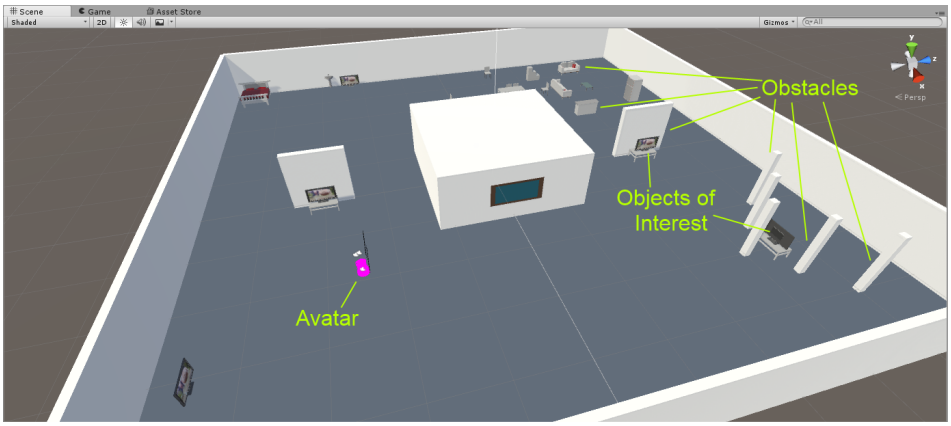


Figure 4.14: Virtual environment created for testing purpose during the first step.

Before starting with the implementation of any code, a scene has to be created that will represent the virtual environment. For testing purposes, a housing inspired environment was created (see 4.14), including walls and furniture as obstacles and televisions displaying some image as objects of interest to inspect. The operator's avatar and a surrogate robot model are placed in the scene, as well.

To simulate missing detail in the reconstructed environment that is supplied by the live video, the displayed image is added to a special layer that can be rendered separately. The special layer is then set to only be rendered by the virtual camera attached to the surrogate, to simulate additional details being perceivable by using the video streaming augmentation.

Visual Feedback

Visual feedback is to be provided to the operator by displaying imagery of the virtual environment via a head mounted display. *Valve*'s *HTC Vive*, which is one of the most prominent and easy-to-integrate VR platforms featuring native Unity support, is used in this implementation.

Installing and running applications with the HTC Vive requires a copy of the digital distribution platform *Steam*. By downloading the steam application *SteamVR* via Steam and setting up the device with the downloaded application, the HTC Vive is ready to be used.

To integrate the HMD into a Unity project, the *SteamVR Plugin* asset has to be downloaded from the Unity asset store and imported into the project[12]. If the HTC Vive is already installed on the system, it can then directly be used within the Unity project.

The asset provides objects and scripts that are required to connect the Unity application to the HTC Vive. One of these objects is an invisible object called *[SteamVR]* that has to be placed in the scene.

Unity provides camera objects that can be used to render vision from the camera object's point of view. Additionally, the asset provides the *Steam VR Camera* script that needs to be attached to the camera object within the scene, whose vision should be rendered to the HMD. Both avatar and surrogate are equipped with a camera object, to record their vision of the virtual environment. As the operator should perceive the environment from the avatar's point of view, the script is appended to the avatar's camera object.

When starting the application, Unity will check for an available HTC Vive by executing the SteamVR application and connect the camera with the Steam VR Camera script to the HMD's display. With this setup, any head movement performed by the operator will be translated to a camera movement in the virtual space and each frame, a new image will be rendered to the HTC Vive for the operator to perceive.

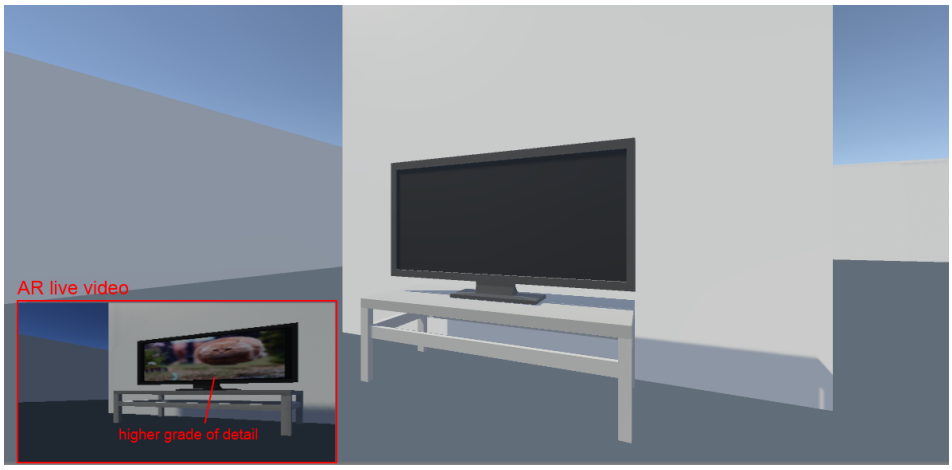


Figure 4.15: Sample view of the environment including AR inspired video stream.

Figure 4.15 shows a sample view that is presented to the operator. The image was not captured on the HTC Vive, but instead as a screenshot in the Unity window. The image therefore slightly differs from what the operator perceives via the HMD, but it gives a general idea of how the virtual environment is presented.

On the lower left, a small frame is added to the operator's field of view, displaying the live image that is recorded by the surrogate. The surrogate records video by rendering

[12]Steam VR Plugin, Retrieved via https://goo.gl/wwx2fG on 19/12/2017.

the field of vision of its camera object to a texture once per frame, by using the provided *LateUpdate()* handler to ensure that all other changes in the environment for this frame have finished. The texture is then sent to the client via a simulated connection. The client contains a script that will try to display one of the received textures each frame using the *Update()* handler if available, creating the impression of a video being played in the AR window. As the AR window is a game object that is placed in the avatar's field of vision, it has to be adjusted to potential head movements of the operator. Therefore, its position is adjusted relatively to the avatar's camera object. As the head tracking will be applied to the camera object once each frame, it is sufficient to adjust the AR window's position once each frame, using the *Update()* handler. Furthermore, video streaming can be toggled, commanding the surrogate to start or stop recording and transmission of live image. The AR window will accordingly only be displayed, when video is streamed.

Human Agent Controller

According to the concept, the human agent acting as the operator of the teleinteraction system needs to be able to control the surrogate with some sort of controller. For this implementation a simple keyboard is used to control the surrogate. A limitation to navigation of a robot should be sufficient to prototypically implement and evaluate the 3D reconstruction approach to teleinteraction. Therefore, a simple keyboard can cover all required controls like moving or turning to command the robot. Also, the keyboard can be used while being seated. A seated teleinteraction experience will support the operator's sense of stability during immersion into VR.

```
PSClient tecsSocket = PSFactory.CreatePSClient(PSFactory.
    CreateURI("<ClientName>", Settings.ServerURI, Settings.
    ServerPort));
tecsSocket.Connect();
```

Figure 4.16: Code to set up a TECS connection with the TECS server.

```
tecsSocket.Send(".*", "<Topic>", event_object);
```

Figure 4.17: Code to send a TECS event via a TECS connection.

For recording and interpretation of controller inputs, a script called *HumanAgentController* is written and attached to the avatar.

Using the *Start()* method handler, a connection with the TECS server is initialized (see 4.16) at the start of the client application. As the controller only reacts to keyboard inputs, there are no TECS topics that need to be subscribed to.

In the *FixedUpdate()* handler, keyboard inputs are interpreted and translated to commands for the robot. It is common practice to capture keyboard inputs in the *Update()* handler. However, to make sure that commands are consistently sent to the robot without interruption due to dropped frames or lag, the controller script was implemented using the *FixedUpdate()* handler. W, A, S and D keys are converted to

movement directions in form of a *TwistEvent* and are sent to the avatar locally and to the surrogate via TECS connection (see 4.17). *TwistEvent* is a message type made up of two vectors representing linear and angular velocity for robots. It is inspired by the *Twist* message in ROS that is used to assign a velocity to motors of a robot. Furthermore, strokes of the space bar are turned into *RequestVideoEvents* and sent to the AR window locally and to the surrogate via TECS. *RequestVideoEvents* contain a boolean flag to turn on or off the video streaming depending on the value.

Avatar

The avatar is the operator's gateway to immersing into the virtual environment. The avatar simulates actions as for example movements that are carried out by the robot and perceives the virtual environment, so the this perception can be shared with the operator (see 4.18).

Commands in form of *TwistEvents* are sent locally to the avatar. Communication between avatar and robot requires a TECS connection that is set up in the *Start()* handler. As already explained, the incorporation of pose feedback can help in reducing the drift between reality and virtual environment. Therefore, the avatar subscribes to the topic of *PositionEvents* that contains exactly this pose feedback.

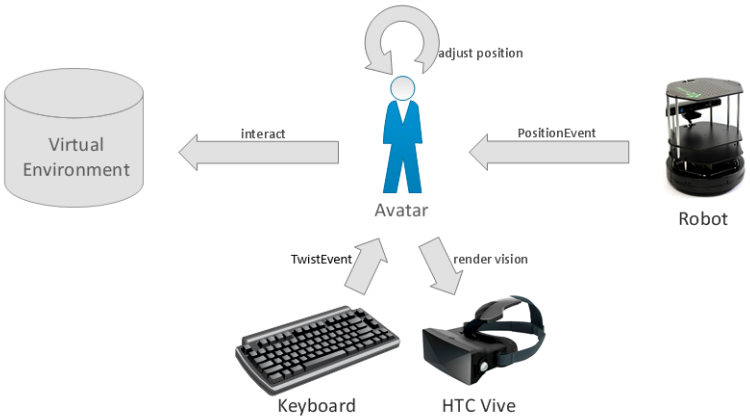


Figure 4.18: Flow diagram of the virtual avatar.

Different approaches to incorporate the pose feedback contained in the *PositionEvents* were implemented and are explained at a later point.

Even though they are sent locally, commands to make the avatar simulate surrogate movements are sent in the form of TECS *TwistEvents* triggering a callback in the script. The callback creates and stores the currently desired velocity in form of three dimensional vectors for both linear and angular velocity from the received *TwistEvent*. In the *FixedUpdate()* handler, the avatar is moved based on this velocity.

As explained in the concept, a drift between reality and virtual space can be introduced due to motorical latency and the virtual avatar not simulating this latency. Therefore, instead of directly setting the avatar to the desired velocity, the avatar simulates the acceleration up to the desired velocity.

Simulated Robot

The simulated robot is supposed to represent the physical surrogate robot that is not yet integrated into the system during the first step. Thus, he needs to move according to the operator's commands and provide both visual and pose feedback.

Analogously to the avatar, the simulated robot establishes a TECS connection on startup and then subscribes to the topics *TwistEvent* for instructions on how to move and *RequestVideoEvent* for signalling video streaming requests. Received events via this connection are picked up similarly to the avatar script, but processed immediately, as they only set flags or current desired velocity.

Reporting its current position is one of the core responsibilities of the robot, as the feedback is required to reduce the drift between reality and the virtual representation of reality. A consistent stream of pose feedback can help to adjust the avatar's position contemporarily fixing existing drift and reducing future drift. Therefore, the simulated robot sends a *PositionEvent* in regular time intervals using the *FixedUpdate()* handler via the local connection.

Incorporating Pose Feedback

In the concept, the problem of synchronization of the avatar with the actual surrogate robot was described. As involved latencies and inaccuracies in the avatar's simulation of robot actions keep adding up, reality and virtual reality keep drifting apart. In this teleinteraction prototype which is limited to simple navigation of the surrogate's environment, this induces a discrepancy in the avatar's and the surrogate's position. To synchronize avatar and surrogate and to reduce this discrepancy, the incorporation of pose feedback provided by the surrogate in form of *PositionEvents* is implemented in the avatar's script.

As mentioned, the avatar subscribes to the *PositionEvent* topic to receive the pose feedback via the TECS connection. Different approaches to adjust the avatar's position to the reported pose feedback were implemented. The reception of the *PositionEvent*, as well as the *FixedUpdate()* handler can serve as triggers for adjustment of the avatar pose.

Implemented approaches include simple resetting of the avatar pose to the reported pose feedback, whenever the avatar is idle or constant step-by-step adjustment to the most recently reported pose to avoid major jumps of the avatar.

As approaches like resetting of the avatar pose might cause the avatar to teleport over uncomfortably long distances based on the degree of present drift, a blinking effect similar to the one implemented in the game *The Lab*[13] was added. Whenever the avatar's position is reset to the reported pose, the blinking effect will be triggered dampening the impact of the avatar's sudden movement.

Implemented Functionality

The first step of the implementation was mainly focusing on the development of the client application and ensuring its functionality in the overall system by adding a sim-

[13]Blinking effect when teleporting the avatar in VR (at 2:22), Retrieved via https://goo.gl/4DzTJi on 12/12/2017.

ulation of the missing components. The integration of the HTC Vive and the keyboard controller into the Unity client application was finished. Inputs will be translated to command events and sent to the surrogate and locally handed to the avatar for execution. The avatar executes movement commands, while simulating basic motorical latency. Rendering of the avatar's vision to the HMD, as well as augmentation of this vision with live video is already included. Adjustment of the avatar's position based on pose feedback from the robot to reduce drift is also implemented.

Even though a simulated robot was used to test the functionality, the client will keep this functionality for the keep steps due to the modular design. This allows us to remove components and replace them with alternative components with the same interface. By removing the simulated robot and adding a real robot that uses the same event and message types to the system, a physical robot can easily be added to the teleinteraction system.

4.4 Teleinteraction without 3D Reconstruction

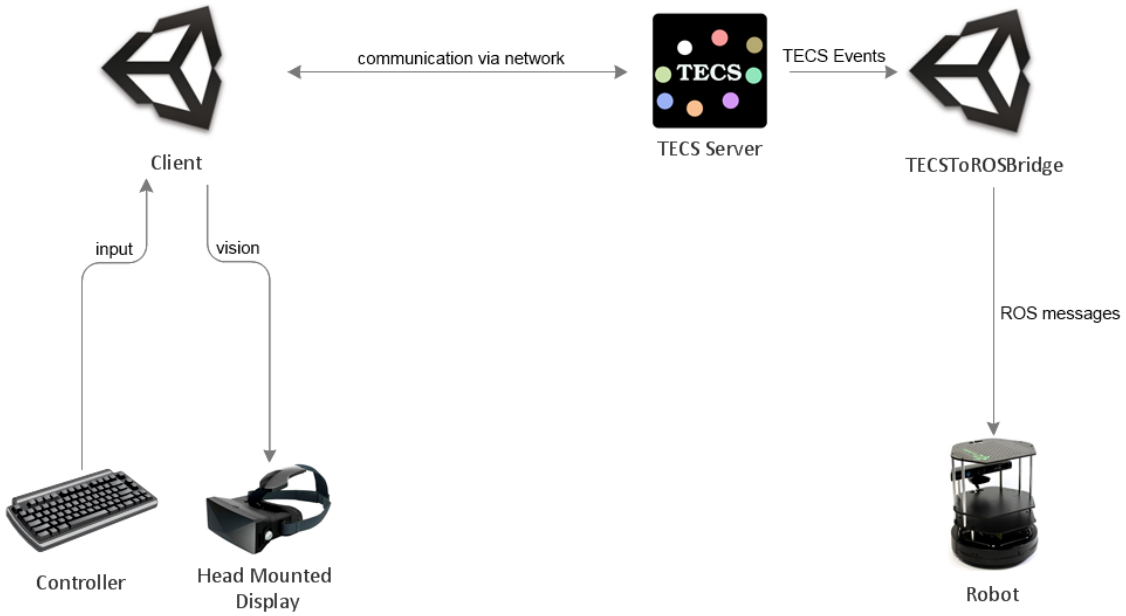


Figure 4.19: Prototype after completion of the second step.

The second step is to integrate a real robot surrogate into the teleinteraction system. In scenarios, where a virtual representation of the surrogate's environment is already available, this prototype should already suffice to perform basic teleinteraction with some limitations.

The architecture of the prototype implemented in this second step is based on the one from the previous step. The main difference is the removal of the simulated robot from the client and extension of the system by a real robot (see 4.19). Commands to the robot are thus now sent to a TECS server that is not bound to be local anymore, involving

actual potentially high network latency for the first time. Since TECS messages can currently not be directly transmitted to the robot, a *TECSToROSBridge*, translating TECS events to ROS messages, is part of the connection. As the websocket connection used by ROS can be rather slow, it makes sense to locate this TECSToROSBridge close to the robot surrogate if possible. Converted commands are sent to the robot from the bridge, where they are executed.

The support of live video augmentation was deprecated in this version, as the client side functionality was confirmed during the first step and recording of live video by the robot would require a camera. As cameras and environment scanners are part of the third and final step, there is no hardware available to provide video for this step. By adding a simple camera device sending frames via the connection, the video streaming functionality could easily be restored.

Robot Surrogate

Instead of a simulated robot, a real physical surrogate robot is integrated into the teleinteraction system in this step. Due to the nature of the navigation task that is used as the underlying teleinteraction scenario, robots that can properly move around, have sufficient capabilities to be used in the teleinteraction prototype.

The robots that were available and meeting the required capability were the *Turtlebot*[14] and the *Baxter Mobility Base*. The Turtlebot provides a simple interface, good documentation and many usage samples. However, its mobility and dimensions are inferior to the Baxter Mobility Base for our purpose. The Mobility Base incorporates a mecanum wheel system enabling the robot to move on most surfaces without any issues. Also, the Mobility Base is taller than the Turtlebot, allowing for higher placement of cameras and 3D scanning devices. This can be advantageous for 3D reconstruction as higher levels of the environment can be scanned more easily and likely better. Therefore, the Baxter Mobility Base is used for the implementation of the teleinteraction system.

The Baxter Mobility Base is running with the ROS operating system. To connect to a robot running ROS, there are provided ROS nodes that handle incoming connections. One of these nodes is the *rosbridge_websocket* node from the *rosbridge_server* package. By running this node, it is possible to connect to the Baxter Mobility Base's ROS operating system via websocket connection and transmit ROS messages to the robot.

The interface for navigation of the Baxter Mobility Base is the */mobility_base/cmd_vel* topic[15]. By sending *Twist* messages to this topic containing the desired linear and angular velocity, the robot will perform movements according to the operator's intention. The Baxter Mobility Base implements a safety mechanism, stopping the motors if the Twist message is not repeated in regular intervals. Therefore, messages have to be sent repetitively to keep the robot moving with constant speed.

Client-Robot Connection

As in the first step, commands to the robot are sent to a TECS server. Depending on the location of the robot, this TECS server might not be local any longer, but located

[14]Turtlebot, Retrieved via http://www.turtlebot.com/ on 13/12/2017.
[15]Baxter Mobility Base Documentation, Retrieved via https://goo.gl/BS7TQF on 20/12/2017.

somewhere between the operator's location and the robot's environment. The actual robot surrogate runs with the ROS operating system and requires commands in form of ROS messages. Thus, the TECS events sent by the client need to be converted. The *TECSToROSBridge* was implemented to translate both TECS events to ROS messages and ROS messages to TECS events as necessary, connecting the TECS based communication on client side with the ROS based communication on robot side (see 4.20).

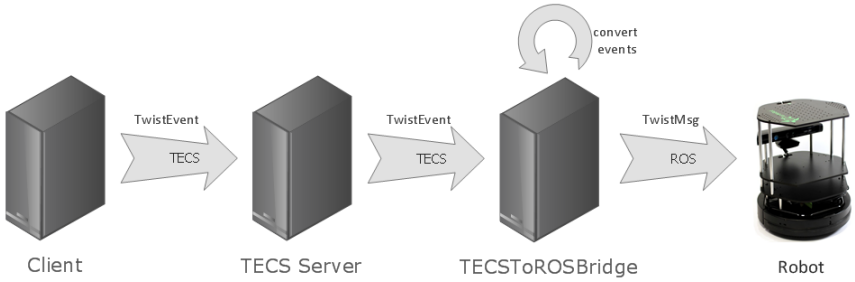


Figure 4.20: Communication between client and robot.

UnityROS allows to inject ROS messages into the ROS operating system from Unity. The TECSToROSBridge was therefore developed in Unity. It is a simple program without graphical user interface that consists of a single thread continuously checking for incoming messages from both sides of the system and translates them for the other side.

The connection to both TECS server and robot's ROS operating system are set up in the *Start()* handler at the start of the application. The bridge application needs to subscribe to the *TwistEvent* topic to receive the operator's commands. UnityROS requires the publishers to advertise their topics in advance. Therefore, the *Twist* message's topic must be advertised in addition to the establishment of a connection to the robot's operating system.

Implemented Functionality

The second step of the implementation was focusing on the integration of an actual physical robot surrogate into the teleinteraction system. A TECSToROSBridge was developed allowing the translation of commands in form of TECS events to ROS messages. These messages can then directly be injected into the robot's ROS operating system and executed.

The current prototype implements all the functionality to control the surrogate robot. It is missing the visual feedback for the operator to perceive the robot's environment and grasp the robot's situation. In the third and final step the visual feedback in form of a 3D reconstructed virtual representation of the environment augmented by live video streaming will be implemented to add the missing functionality to the teleinteraction system prototype.

4.5 Teleinteraction with 3D Reconstruction

The third and final step of this implementation is to add the missing visual feedback of the surrogate's environment to the teleinteraction system. By 3D reconstructing a virtual representation of the real world environment and augmenting it with recorded live video the operator will be able to perceive the real environment from the surrogate's point of view, helping him immerse into the surrogate's situation.

During the third step, different approaches to 3D reconstruction were implemented. Some of the approaches were not successful due to technical issues with the 3D sensing hardware, while other approaches had practical issues that made a proper integration into the system impossible. Therefore, alternatives were researched and implemented to create a working prototype for teleinteraction. Each of the approaches will be explained in the following, including their implementation and the issues that were encountered.

As the third implementation step introduces functionality that is shared by the different 3D reconstruction approaches, changes to the prototype that are related to this functionality will be elaborated on in advance.

Adjusting the Scene

To add 3D reconstruction to the teleinteraction system, the client side's Unity scene needs to be adjusted. Currently a sample environment is contained in the scene, which would be in the way of any reconstructed objects from the actual real environment.

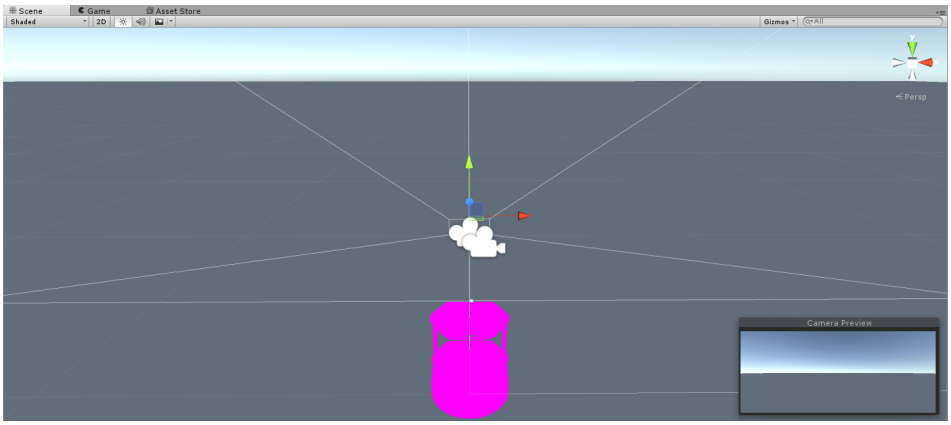


Figure 4.21: Emptied scene that provides the required space for 3D reconstruction.

Therefore, a new scene is created that is completely empty, except for the the operator's avatar4.21. Only a plain object is added to resemble the ground floor.

To perform the actual reconstruction of the provided environment data, a script is needed on the client side. The exact functionality depends on the implemented approach and will be explained in the respective chapter. For the script to be executed, it is added to the scene, appended to an *EnvironmentManager* object.

Live Video Augmentation on Client Side

During the first step, the augmenting video streaming was implemented by having the simulated robot render its vision into a texture and sending it locally to the client, which in turn displays the texture in the operator's field of vision.

As the third step involves actual cameras recording video image in a possibly remote location, video frames have to be transmitted via the network, be received and then displayed accordingly. The already existing script for receiving and displaying textures from the simulated robot, thus has to be altered to subscribe to the video frame topic *TextureFrameEvent* and render the frame to the AR window in the operator's field of vision.

While different cameras can encode frame data in different ways, by serializing the data in a byte buffer representation and interpreting it accordingly on the client side, a common message type can be used to to send frames via the network connection. Figure 4.22 shows the underlying Thrift definition for the TECS event that is used to send frames from the robot side to the client.

```
struct TextureFrameEvent {
  1: required binary frame;
  2: required i64 timestamp;
}
```

Figure 4.22: Thrift definition of the *TextureFrameEvent* message type for sending video frames.

SLAM and Pose Tracking

With the integration of 3D reconstruction and required 3D environment scanners, the robot acquires the required hardware to perform SLAM while navigating through its environment. This means that the robot incorporates pose tracking, providing the position of the robot.

In the first step, different approaches to incorporate pose feedback from the robot have been implemented to reduce drift between the robot's and the avatar's position. By using the pose feedback that is now provided by the SLAM algorithm, the implemented approaches can be fed this feedback to adjust the avatar's position in the virtual environment to the actual position reported by the robot.

To incorporate the pose feedback on the client side, the robot or his equipment have to send their data via the network connection. The *PositionEvent* topic is used to distribute the pose feedback via TECS. The respective event definition can be seen in figure 4.23. *PositionEvents* consist of the 3D vector representing the robots position in space and the quaternion representation of its rotation.

Both position and orientation of an object depend on the origin and scale of the underlying coordinate system. As the Unity coordinate system and reality's coordinate system, which is represented by the SLAM algorithms internal coordinate system, use different scales and possibly orientation, feedback from the robot cannot just be used on the client side. A different base unit of pose values might distort the scaling on the

```
struct PositionEvent {
  1: required double pos_x;
    2: required double pos_y;
    3: required double pos_z;
    4: required double rot_x;
    5: required double rot_y;
    6: required double rot_z;
    7: required double rot_w;
    8: required i64 timestamp;
}
```

Figure 4.23: Thrift definition of the *PositionEvent* message type for reporting pose feedback.

Unity side. Also, a differing orientation of both coordinate system will introduce an unwanted rotation in the transmitted feedback. Thus, offsets in position, rotation and scaling are factored in, whenever a conversion of data from one coordinate system to the other is necessary.

Meshing using the Google Tango Project

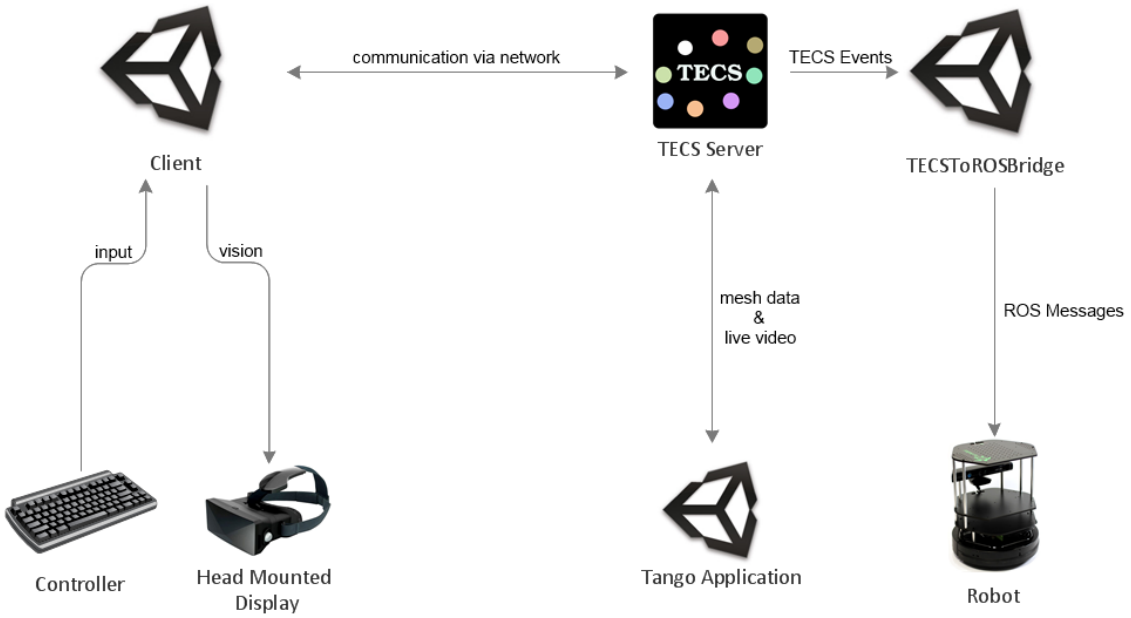


Figure 4.24: Prototype after completion of the third step with use of the Tango tablet.

The very first approach to 3D reconstruction to be implemented in the context of this thesis was the creation of meshes using Google's *Project Tango*. The device is supposed to provide the capability to scan real world environments freely and live reconstruct them as meshes that can directly be integrated into other applications. The

Tango features good Unity support, thus making it a great approach for the suggested teleinteraction system.

Figure 4.24 shows the architecture of the Tango based prototype implementation. The Tango device is a tablet that uses integrated cameras on its back to scan the environment. Therefore, the robot is equipped with the Tango such that the cameras can properly scan the environment. The scanned environment data will be processed by the a separate application running on the Tango device and then sent to the client. As the application is not running on the robot, there is no involvement with ROS. Data will therefore be directly sent via TECS without going through the TECSToROSBridge.

Creation of the Tango Application

The application that is responsible for recording and reconstructing the environment as a mesh was developed as a separate Unity project. The Unity project can be built for the Android OS running on the Tango device and then installed on the device. By opening the application on the tablet, the application should connect to the teleinteraction system and start with the reconstruction of the surrogate's environment.

To develop with the Tango functionality, Google provides an SDK in form of a Unity asset that can be imported to the project. The asset includes several examples projects on the different features the Tango tablet incorporates. The documention of the Tango Project can be found here[16]

After importing the Unity asset, a scene is created, where essential objects for the 3D reconstruction features provided by the Tango SDK are placed. The essential provided game objects are the *Tango Delta Camera* and the *Tango Manager*. Their attached scripts integrate the hardware features with Unity and handle basic functionalities like visualization of already reconstructed environment or translation of Tango device movements in the real world to camera movements in the Unity scene. As instructed in the Tango documentation, a *LifecycleManager* is developed that registers the application to the Tango system, so Tango related handlers are triggered and respective data is provided to the application.

The Tango device provides a variety of data that needs to be transmitted to the client. This data includes the mesh reconstructed of the environment, pose feedback and video image for the stream. To transmit the data, a TECS connection is required, which is established at the start of the application.

Extraction and Transmission of Mesh Data

As long as one or more Tango based applications are running and registered to the internal Tango service running on the tablet, the device will scan the environment, reconstruct and store it internally. The stored mesh is divided in segments and organized in a three dimensional grid. Functionality to extract either the whole mesh or single segments is provided by the SDK.

To avoid transmission of redundant data, it is sensible to only send parts of the mesh that experienced change. Therefore, the *OnTango3DReconstructionGridIndicesDirty* handler, is used to trigger the extraction and transmission of the mesh data. Whenever a segment of the mesh changes, the handler is called and provides a list of the segments

[16]Tango Project Documentation, Retrieved via https://goo.gl/JmXkQu on 21/12/2017.

that changed, since the last call. By memorizing the changed segments, they can be transmitted in regular intervals.

Meshes are represented by lists of vertices, normals and triangles, but also colors if available. A respective event for sending the mesh data is created, which can be seen in 4.25. After a mesh event is created, it is published via the established TECS connection on the *MeshEvent* topic.

```
struct MeshEvent {
  1: required list<UnityVector3> vertices;
  2: required list<UnityVector3> normals;
  3: required list<UnityColor32> colors;
  4: required list<i32> triangles;
  5: required i32 numVertices;
  6: required i32 numTriangles;
  7: required i32 x;
  8: required i32 y;
  9: required i32 z;
  10: required double resolution;
}
```

Figure 4.25: Thrift definition of the *MeshEvent* message type for transmitting mesh data.

3D Reconstruction in Unity

For visualization of the surrogate's environment, the meshes that are created by the Tango device need to be displayed in the client scene, such that the avatar and thus the operator can perceive them. To implement the construction and placement of the meshes, the *EnvironmentManager* object is added to the client's Unity scene. The attached script instantiates mesh objects that are placed in the environment, such that it resembles the surrogate's real world environment.

A connection to the TECS server is required to receive the mesh data required to construct the actual meshes on the client side. The EnvironmentManager subscribes to the *MeshEvent* topic, where the mesh data is distributed. During the teleinteraction session, the EnvironmentManager keeps waiting for *MeshEvents* to reconstruct mesh segments and display them, as soon as they arrive.

Generic meshes can be constructed in Unity, by creating game objects with a *MeshFilter* and a *MeshRenderer* component to create and render the mesh. Once a mesh segment is created, it has to be placed at the correct position so the different segments will jointly create a representation of the surrogate environment. The *MeshEvent* contains the segments Tango coordinates, which can be used to calculate the correct position for the segment.

Position Tracking

The Tango device runs a SLAM algorithm internally during reconstruction of the environment to keep track of its position in space relative to the position at the start

of the session. The Tango service provides the *OnTangoPoseAvailable* handler, which is triggered, whenever a new pose is available. By using the handler the current pose data can be retrieved. By creating a *PositionEvent* and publishing it via the respective channel, the Tango device will provide the client with the desired pose feedback.

Adding Real Video Streaming

Video streaming is one of the core features of the teleinteraction system to provide a more detailed perception of the surrogate's environment. The Tango tablet includes multiple cameras that are among others used for the reconstruction of said environment. However, the cameras can also be used to capture video image. The Tango service provides the *OnTangoImageAvailableEventHandler* to extract live image. The handler is called, whenever a new video frame was recorded and provides the image data of the newly available frame.

Video is not supposed to be permanently streamed to the operator. Therefore, it makes sense to reduce the network load by not transmitting video frames, if they are not requested. The app subscribes to the *RequestVideoEvent* topic to get informed about the current state of the video stream.

The video frame is sent to the client via the *TextureFrameEvent* topic in form of the matching event type. The client will then be able to display the image after reception.

Implemented Functionality

After implementing the Tango application, it was installed on the Tango device to run the 3D reconstruction of the surrogate's environment. The application was supposed to create a mesh representation of its environment, while running a SLAM algorithm, keeping track of its position in space and streaming live video.

The Tango application and device did not work as intended. Either the application would crash right at the start or freeze without showing anything and crash after a few seconds. None of the desired data was actually provided by the device. After making sure that a TECS connection was established successfully, the Tango application was debugged and its logs were analyzed to locate the problem. A lot of time was spent on the identification of the problem that caused the crashes, but the application would either not even launch successfully or not produce any data before eventually crashing. Known issues were that rapid movements of the device, as well, moving the device to close to objects that should be scanned would result in crashes. However, even avoiding these issues, the application would still not work. Since the provided example applications showed a similar behaviour, it was concluded that the problem might lie within the Tango technology. This idea is supported by the fact that *Project Tango* was recently cancelled by Google, stopping the production and distribution of the device[17].

After spending high amounts of time on fixing the application to get the Tango device to properly work, the approach to use the Tango device was discarded. An alternative approach using a different technology was found and implemented.

[17]Project Tango cancelled, Retrieved via https://goo.gl/cgkMfo on 21/12/2017.

Visualization using Point Clouds

After deciding to discard the use of the *Project Tango*, an alternative 3D reconstruction technology had to be found. The two alternatives that seemed promising were Microsoft's *HoloLens* and Stereolabs' *ZED Camera*. The HoloLens[18] as an augmented reality headset includes functionality to scan the environment and create a mesh from it, similar to Tango Devices. The ZED Camera[19] is a 3D camera that is made for depth perception and recording of point clouds in any environment.

A master thesis comes with a timely limitation for its making. After spending a lot of time on the Tango, there was not enough time left to try out both devices. Therefore, while both are promising alternatives, only one of both HoloLens or ZED Camera could be used to implement alternative approaches to 3D reconstruction.

The device that was chosen for further implementation was the ZED Camera. The HoloLens was still new at the point in time, when the prototype was built, so there was only little reference material about the meshing functionality, as for example usage in other projects. Furthermore, equipping the surrogate robot with a headset is not very intuitive, since the Baxter Mobility Base does not have a head or any similar part that the HoloLens could directly be attached to. The ZED Camera on the other hand can easily be attached to the Mobility Base. Videos of the live scanning capabilities of even large scale outdoor environments show evidence that the ZED Camera is more than qualified to be used for 3D reconstruction[20]. The ZED Camera also comes with support for Unity and many other common programming languages, making the integration into the overall system easy. For these reasons, the reconstruction of an environment as a point cloud using the provided functionality of the ZED Camera was chosen as the next approach to be implemented.

The integration of the ZED Camera works similar to the previous integration of the Tango device. Figure 4.26 shows the architecture of the prototype implemented in this approach. The ZED Camera is attached to the robot, such that the environment can be scanned with the camera, while the robot navigates. As the ZED Camera is busy constructing point clouds from the environment, a separate camera is attached to the robot to record live video for the AR video stream. Both point cloud data, as well as video frames are sent via the TECS server to the client as both camera devices are not connected to the ROS operating system.

Environment Scanning and Point Cloud Extraction

The ZED Camera provides Unity assets to integrate the device into the software framework. Therefore, the first idea was to create a Unity application running on a machine attached to the robot that runs the ZED Camera. When trying to access the point cloud functionalities of the camera, it turned out that the Unity assets provided by the developers of the camera are missing the required functionality. While it was not mentioned that the Unity support would not feature the full set of capabilities, the resulting problem of how to implement the ZED application was avoided by using the

[18]Microsoft HoloLens, Retrieved via https://goo.gl/hB1vhN on 22/12/2017.

[19]Stereolabs ZED Camera, Retrieved via https://www.stereolabs.com/ on 22/12/2017.

[20]Live 3D reconstruction of an environment as point cloud, Retrieved via https://goo.gl/SkvBZL on 22/12/2017.

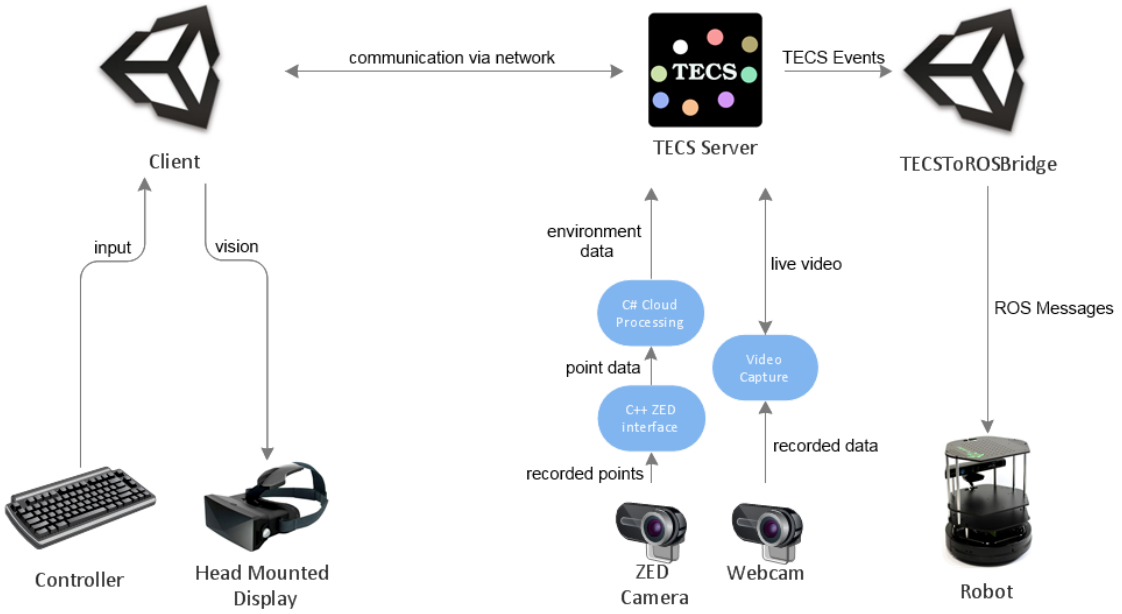


Figure 4.26: Prototype after completion of the third step with use of the ZED camera.

support for other programming languages.

Since the extraction of point clouds can be a performance critical task due to the potentially huge amounts of data, the programming language C++ was used. The developers provide a sample program, showing the integration of the ZED Camera with the *Point Cloud Library*[21]. Based on the sample provided by the creators of the ZED Camera, a program that records point clouds with the ZED Camera was implemented.

The whole implementation of the prototype up to this point was performed on Windows. For the transmission of the data, TECS was included into the C++ application. Due to complications with the project dependencies, it was not possible to compile the project on Windows. Therefore, the C++ application was made a ZED Camera interface that handles extraction of point cloud data from the device.

For transmission of the point cloud data via TECS, the point cloud data has to be handed from the C++ interface to another application that can handle possible processing and transmission. To hand data from one process to another, a C# program was created as the master, running the C++ ZED interface program. The C++ interface extracts point cloud data from the camera, encodes the single point's data as byte strings and writes them to the standard output. The master program reads those byte strings from that output, parses them and bundles the point data to recreate the original point clouds. By taking this detour, it is then possible to transmit the point data to the client via TECS.

The ZED camera provides a frequency at which new data is generated. If this frequency is not met, the camera's internal buffer for outputting the data will overflow, resulting in corrupt data. Since data output is slower than the extraction of the data from the camera, data that is extracted while old data is still being output has to be

[21]Sample code for ZED, Retrieved via https://goo.gl/Sk8acD on 22/12/2017.

discarded. The C++ ZED interface therefore consists of two threads organized in a monitor pattern. Figure 4.27 shows a flow chart of the two threads. One thread keeps extracting the point cloud data and trying to write the data to shared memory for outputting, if the other thread is not still busy with the output process. Otherwise, the newly extracted data will be discarded. As consecutively recorded data is unlikely to contain major changes, it is unlikely that discarded data will have much impact on the 3D reconstruction. The output thread will output data, whenever new data is available in the shared memory.

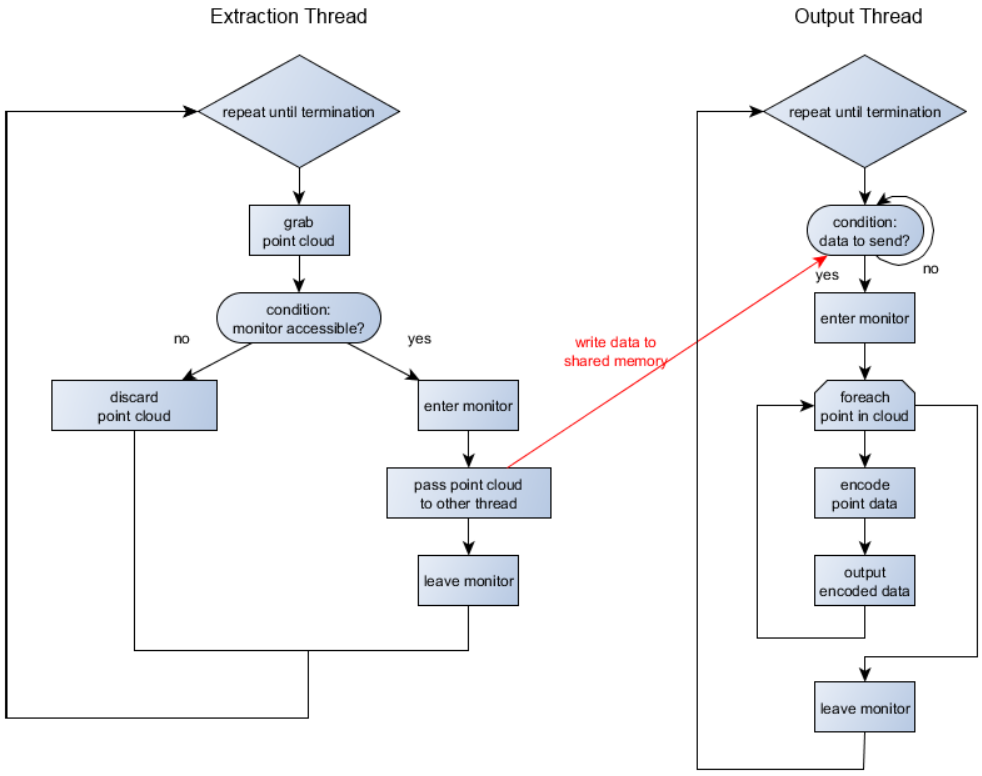


Figure 4.27: Flow chart of the threads providing point cloud data.

Processing of Points and Transmission

Given the point cloud data extracted by the C++ interface, all that is left is to process the data if necessary and send it. To speed up this process, three separate threads were designed to split the workload: The *receiver*, reading data from the C++ interface, the *processor*, processing the point cloud if necessary and the *sender*, handling the sending of environment data. The environment data is sent via a TECS connection, as the application is running on a separate machine from the surrogate's ROS system.

The *receiver* runs the C++ ZED interface and reads its standard output. The output contains the single points of the point clouds, separated by terminators. The single points of the same cloud are read and grouped in a point cloud data structure representing the full cloud. This data structure is then handed to the *processor* for possible processing of the cloud.

The *processor* is a component that can be implemented to process the point clouds if necessary. Once, a cloud has been processed, it is handed to the *sender* to be transmitted to the client. Simple processing could for example be the removal of redundancies or invalid data to reduce the network load.

The *sender* sends the environment data that results from processing the extracted point cloud data. To avoid transmission of redundant data, the *sender* keeps track of which points he has already sent to the client and which were newly acquired since the transmission. The points are sent in the form of *ZEDPointCloudEvent* by publishing them to their respective topic. As shown in figure 4.28 the event is made up of an arbitrary number of point events.

```
struct ZEDPoint {
    1: required double x;
    2: required double y;
    3: required double z;
    4: optional double rgb;
}

struct ZEDPointCloudEvent {
    1: required list<ZEDPoint> points = [];
  2: required ZEDPoseEvent pose;
}
```

Figure 4.28: Thrift definition of the *ZEDPointCloudEvent* message type for transmitting point clouds.

For some reconstruction schemes, it might be interesting to delete points in the reconstructed environment to remove errors or adapt to changes. Using a transmission scheme that only sends each point to the client once, conflicts with this idea. For example, a point that was mistaken for an error and deleted, will never be resent and thus forever be lost from the virtual reconstruction. As evidence shows that the ZED Camera works with a high precision and creates little erroneous data, using this simple transmission scheme and a matching reconstruction scheme will suffice for a first prototype.

3D Reconstruction in Unity

For visualization of the surrogate's environment, the point clouds that are recorded by the ZED Camera should be displayed in the client scene, where the avatar and thus the operator can perceive them. To place the single points from the point clouds, the *EnvironmentManager* from the preceding Tango approach is adjusted to create small sphere objects for each point in space, such that together they resemble the surrogate's real world environment.

The point cloud data is received via a connection to the TECS server. The EnvironmentManager subscribes to the *ZEDPointCloudEvent* topic, where the point clouds are published. During the teleinteraction session, the EnvironmentManager keeps waiting

for *ZEDPointCloudEvents* to extract the points from it and create objects accordingly, as soon as they arrive.

As with the Tango approach, coordinates from the ZED Camera have to be translated to the Unity coordinate space first. The ZED Camera uses metric measurements and provides the point coordinates in meters accordingly.

Position Tracking

In addition to the point cloud data, the ZED Camera's interface provides functionality to retrieve the pose, in which each point cloud was recorded. This pose can be used to provide regular feedback about the camera's and thus the surrogate's pose. The extracted pose is added to the output of the point clouds and and then grabbed by the C# application that is reading and processing the camera's cloud data. Whenever the *receiver* reads a pose, he prepares a *PositionEvent* and has it sent by the application's TECS connection. Once, received the client can then use the pose feedback for correction of the avatar's placement in the virtual environment.

Adding Live Video Streaming

The ZED Camera provides functionality to extract captured image via the interface. However, the ZED Camera is busy with extracting the point cloud data. To not interfere with that task and avoid possible slowdown of the reconstruction process, the video capturing will be outsourced to an additional program running on the separate machine attached to the robot. The additional program written in C# and is connected to a webcam to record video frames. The single camera frames are sent via the TECS connection as *TextureFrameEvent*.

Due to the modular implementation and the unchanged interface, no changes on the client side are required to get the video streaming to work. Depending on the encoding of the frames, the client might have to adjust the decoding of the image data.

Implemented Functionality

Figure 4.29 shows a sample reconstruction using the raw visualization of point clouds. The sample was created using the prototype implemented in this approach. While the result of the reconstruction is fairly recognizable, the creation process has some issues.

A single point cloud during the reconstruction tests already contained around 11.000 points. Processing the cloud and creating a game object for each of these points induced a huge toll on the performance resulting in high delays for displaying even single clouds. The huge amounts of data being sent via the network, inducing high network latency, delaying the reception of the cloud data even more to this delay. Keeping to send further point clouds during a teleinteraction session reduces the performance even more resulting in many frames being dropped in the client. For proper immersion using a head mounted display, this is not acceptable.

The ZED SDK came with a sample software that was shown in the video evidence, showing the live reconstruction capabilities of the ZED Camera. The provided software is closed source, so it was not possible to check the developer's code that is capable of reconstructing the environment with little delay. It is likely that it includes some way of compressing the the data amount to allow for a faster visualization to the user.

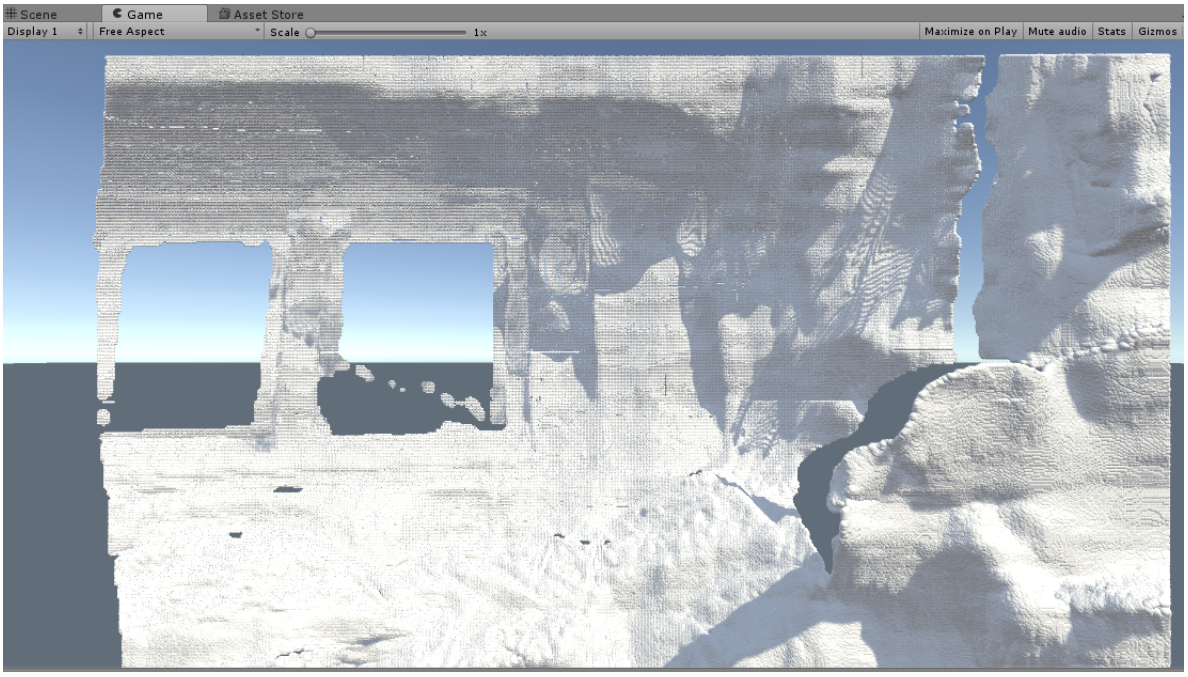


Figure 4.29: Environment reconstruction using point clouds.

Therefore, a different approach applying compression of the extracted clouds is proposed in the next chapter.

Compressed Point Clouds: Blocks World

The main problem of the point cloud reconstruction approach is the huge data amount that has to be sent and displayed for each extracted point cloud. In this approach, the point cloud data will be compressed to reduce the amount of data that has to be sent via the network. Furthermore, the sphere game objects that are created to represent the points will be joint to reduce the amount of game objects that have to be instantiated for reconstruction.

The idea to compress the point cloud data is to reduce the level of detail of the reconstruction by joining closely located points in space. To do so, the world is segmented in a three dimensional grid, creating a world that consists of blocks instead of single small points. The idea is inspired by the construction of the world in the game *Minecraft* (see 4.30). Points that are located in the same sector will create a block, if there are enough points to surpass a certain threshold in the same sector. By regulating the threshold jitter can be filtered and the reconstruction can be tweaked.

The implementation of the approach using compressed point clouds is based on the implementation of the point cloud approach. The difference between the two approaches is the processing of the point clouds to reduce the amount of data and the reconstruction of the sent environment data in the Unity client. As the other components of the system remain the same, this chapter will focus solely on the changes made from the implementation in the previous chapter.

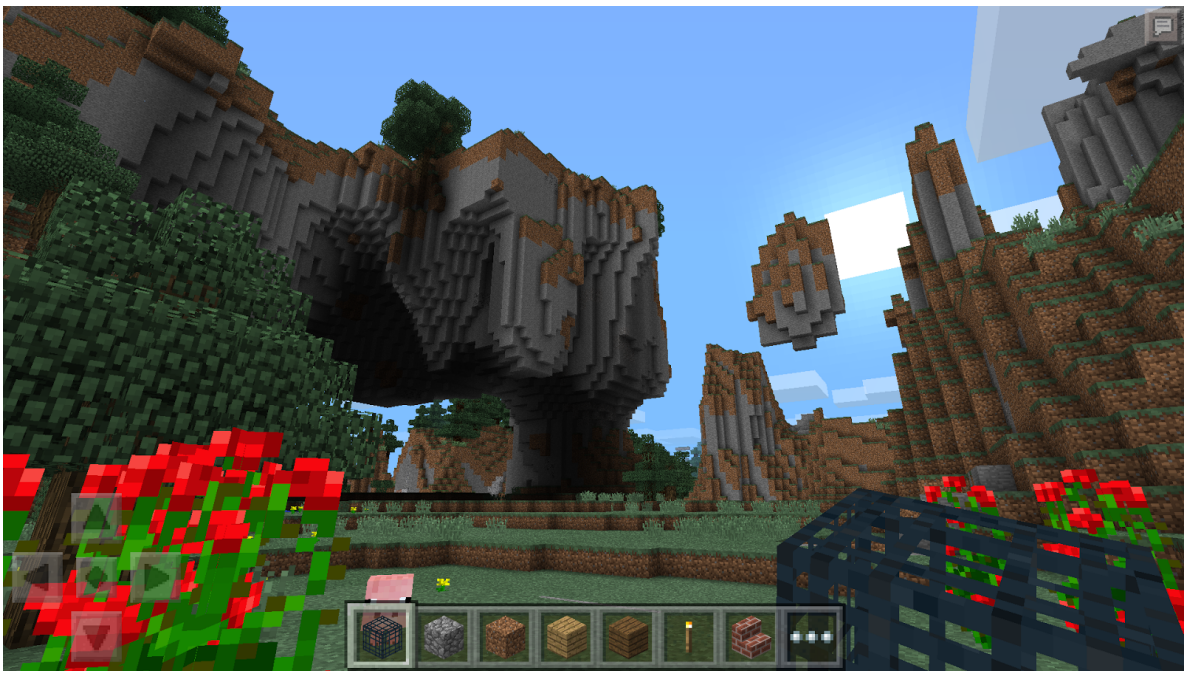


Figure 4.30: Sample game world of the game *Minecraft*, Retrieved via https://goo.gl/VJ2o44 on 24/12/2017

Compression of Point Clouds to Blocks and Transmission

The compression of the point clouds can be implemented by replacing the *processor* of the previously implemented approach. The previous *processor* did only perform very basic processing as for example filtering redundancies. By using a processor that joins the points in the same sector and lets only sectors that contain enough points be sent, this approach can easily be implemented. Figure 4.31 shows a flow chart of the compression process.

The *processor* is provided with an object representing the recorded point cloud by the *receiver*. For each of the contained points the sector of the three dimensional grid, which contains the point, is calculated. The sectors that contain points are stored in a dictionary together with the amount of points each sector contains. If the amount of points contained by a sector exceeds the defined threshold, the sectors index in the grid is handed to the *sender* for transmission.

Since points keep being added to the grid, eventually all sectors would contain enough points to produce a block. Thus, it is important to take into account the history of added points in some way. If only one point was added to a sector often enough over the teleinteraction session due to jitter, it would not be desirable to place a block for that sector. If a high amount of points was added to a sector at one time, the probability that a real world object was recorded in that sector is very high. A decaying process of points is implemented that reduces the count of the points in each sector with each received cloud. Blocks are then only created, if a lot of points are added to a sector in a short time.

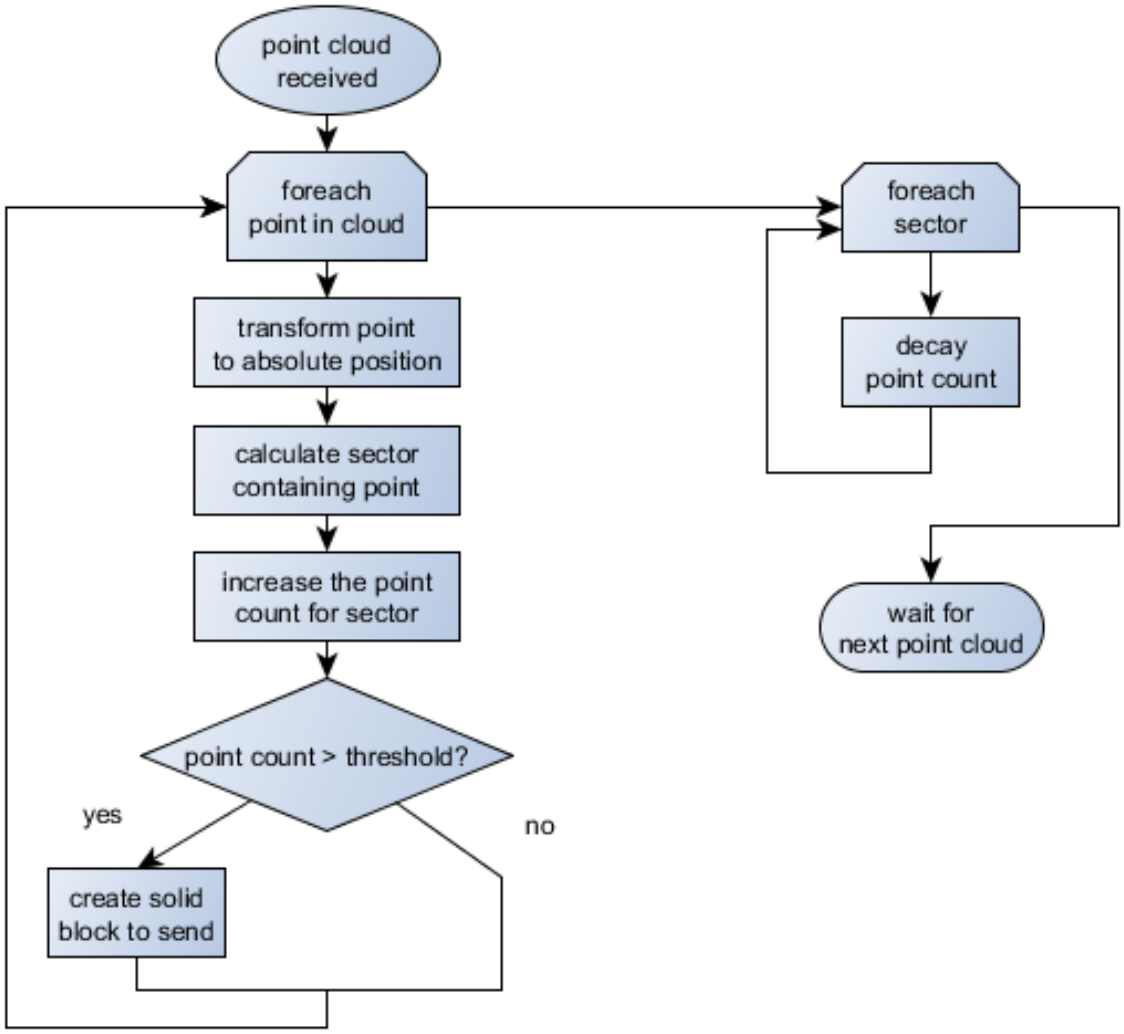


Figure 4.31: Flow chart of the compression process.

3D Reconstruction in Unity

The reconstruction scheme in the Unity client for the creation of a blocks world is similar to the reconstruction using point clouds. Instead of creating a sphere for each received point, a block is created for each sector that is to be made solid in the virtual environment. As the compression of the point clouds creates indexes in the three dimensional grid for indicating sectors that should become a blocks, the received indexes have to be translated to the sector's position in the Unity space before placing the block.

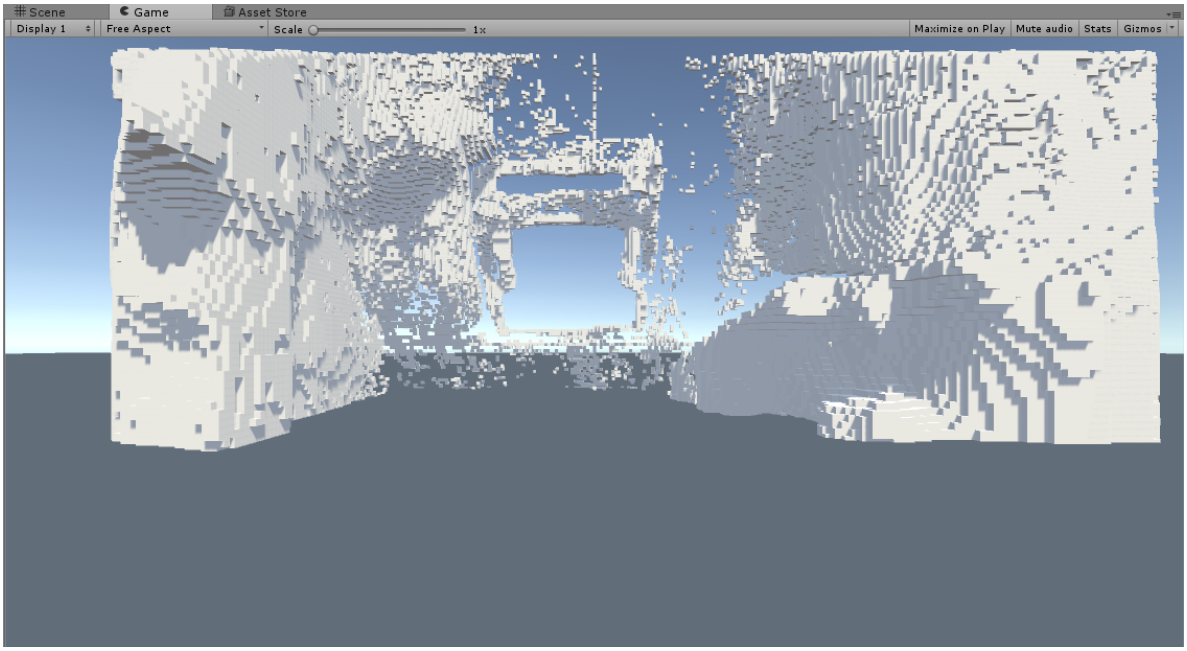


Figure 4.32: Environment that was reconstructed using the blocks world approach.

Implemented Functionality

Figure 4.32 shows a reconstruction sample of an office. Further reconstruction samples can be found here[22,23,24].

The implementation of this approach still has some delay in receiving the environment data from the surrogate due to network latency and processing latency of the point cloud. Because of the reduced amount of data to be displayed, no frames are dropped anymore by the client, avoiding the issue of a bad immersion experience for the operator. As this is the basic requirement for a working prototype, this prototype will be used for the evaluation.

[22]Recording of a sample reconstruction while standing still, Retrieved via https://youtu.be/9ZKyv-VzhG4

[23]Recording of a sample reconstruction while moving, Retrieved via https://youtu.be/EvjkncdWID8

[24]Recording of sample reconstruction while turning, Retrieved via https://youtu.be/dbn2oiIeVck

5 Evaluation

The implementation of the proposed teleinteraction system was performed using three different approaches to 3D reconstruction. Since several issues were encountered with the different approaches, a technical evaluation of each approach will be conducted. Flaws in the design of the system, as well as problems with the utilized technology will be located and analyzed. In addition to the technical evaluation, a performance evaluation of the implemented prototype will be conducted. As latency is the central factor, all involved latencies including time-wise performance of each component of the system will be measured. A user study for evaluation of the teleinteraction experience would be of interest as well. The study is not conducted in the context of this thesis. However, the study was prepared including a plan of execution, questionnaires and scenarios. The prepared study will be discussed at a later point in the future work chapter.

5.1 Technical Evaluation

3D reconstruction is the central technology at the core of the proposed teleinteraction approach. In the context of this thesis, four possible approaches were mentioned, of which three were actually implemented. During the implementation of two approaches, critical issues were encountered that impeded the successful completion of the prototype using these approaches. In the following, an evaluation for each of the implemented approaches will be given, to summarize the advantages and problems of each approach for future reference.

Meshing using Google's Project Tango

Google's *Project Tango* is a technology platform that was created for easy development of AR applications. The most common Tango device, the Tango tablet, is equipped with a stereo camera and a fish eye camera for perception of the environment and a gyroscope for tracking the device's movements. Using these it is possible for the Tango device to record video, track its position in space or even perform area learning. The provided APIs enable developers to easily create applications that use these features to support the making of AR based applications.

The Project Tango supports common development environments and programming languages, namely Unity, C and Java. The APIs are easily integrable, as the use of each feature of the Tango device can directly be implemented by adding handlers from the provided API interfaces. Extraction of the reconstructed environment in form of a mesh, pose data and live image can quickly be implemented with the provided API.

A finished Tango application, can be installed to any Tango device via the build APK file. However, the application implemented in this thesis would crash either on

the startup or after running for some seconds without performing its purpose. Provided example applications showed the same behaviour, indicating technical issues with the hardware. The documentation mentions known issues, such as sudden rapid movements or recording of objects from up close to cause crashes of the Tango service. The Tango logs could not provide any helpful information about the crashes.

The Project Tango showed severe issues preventing the functional integration of a Tango device into the teleinteraction system. The Project Tango was announced to be deprecated as of the 1st March 2018[1]. This supports the conjecture that the Project Tango has severe technical issues that prevent proper usage of the technology.

Point Cloud and Compressed Point Cloud Visualization using Stereolabs' ZED Camera

Stereolabs' *ZED Camera* is a stereo camera that provides depth perception and related image processing capabilities. The camera additionally contains a gyroscope for tracking of movements and its pose in space. Using these features, it is possible to perform a SLAM task and reconstruct the camera's environment.

The ZED Camera comes with SDKs for Windows and Linux, providing support for integration into third party systems. These third party systems include, amongst others, Jetson, Unity, ROS, *Point Cloud Library (PCL)*, OpenCV, Matlab and the Oculus Rift. The SDK additionally provides a sample program demonstrating the realtime reconstruction of environment using the ZED Camera. Unfortunately, the reconstruction cannot be extracted on-the-fly. It would be favourable to integrate the provided reconstruction scheme into other projects, but this is not possible due to the provided program being closed source.

When developing a Unity application with the SDK's Unity support, it turned out that the SDK for the third party framework Unity was lacking a set of the advertised core functionalities. Namely the reconstruction of the scanned environment, as well, as the extraction of the respective point cloud cannot be performed from within the Unity application. Because of this, the integration of the ZED Camera required a workaround reducing the efficiency of the 3D reconstruction. For other third party support, it should thus be checked in advance, whether the advertised functionality is actually provided, before deciding to use the ZED Camera.

The camera enables recording of the environment at a frame rate of 15, 30 or 60 frames per second. A frame provides roughly 11.000 points in the environment that were recorded by the camera. For each frame, the camera's current pose for translation of the points can additionally be provided. Recorded data contains little errors and is quite precise, as the raw displaying of the recorded point clouds shows.

Visualizing this big amounts of data in Unity or transmitting them via network can lead to performance issues, preventing a fluent live reconstruction, as the implementation of the point cloud approach has shown. As the sample program provided with the SDK cannot be integrated into other projects, processing or compression of the data is required. By reducing the amount of data, a successful implementation of a teleinteraction system is feasible. Lag and dropped frames in the teleinteraction client could be avoided. While showing some downsides regarding performance, the ZED Camera has

[1]Project Tango cancelled, Retrieved via https://goo.gl/cgkMfo on 21/12/2017.

shown to enable the implementation of 3D reconstruction schemes that suffice for the purpose of long-distance teleinteraction.

Microsoft's HoloLens as an Alternative

During implementation of the other three approaches, more or less severe issues have been encountered, especially with the Project Tango. Thus, Microsoft's *HoloLens* might be a good alternative to use for the implementation of an alternative 3D reconstruction approach. By using the meshing capabilities of the HoloLens, it might be possible to implement an approach similar to the meshing approach of the Project Tango. If the meshing works efficiently, it might be possible to get a better visualization of the surrogate environment than with the point cloud based approaches. Therefore, the HoloLens should definitely be considered for extension and improvement of the proposed teleinteraction system.

5.2 Performance and Latency Measurements

Analysis and evaluation of the performance of the implemented prototype is an important measure to identify and locate bottlenecks and weaknesses with the approach or the specific implementation. Delay induced by the network connection and time that is required to perform processing of data in the overall program flow can provide the necessary information to improve the prototype.

```
...
long start = System.DateTime.UtcNow.Ticks;

/* code to measure */

long end = System.DateTime.UtcNow.Ticks;
file.Write("<action>: " + (end - start) + "\n");
...
```

Figure 5.1: Code to instrument the C# based applications.

To measure the performance, the code is instrumented. Figure 5.1 shows the instrumentation that is used to measure elapsed time between two points in the program flow. The program extracts the ticks in UTC time before and after an action is finished. The difference, representing the elapsed milliseconds, is then stored in a file. It is important to use a universal time, as some components might run on machines in different timezones. By using the local time for measurements, errors would be introduced in such cases.

Measuring the elapsed time of actions that involve different system components, as for example transmission of an event for example, has to be performed slightly differently. When a start time is measured, it can be attached to the transmitted data as a timestamp. By calculating the elapsed time between the end of an action and the transmitted timestamp, the duration of the action can be determined. Since different

machines are involved in these measurements, it is important to synchronize the system time of each of them.

Figure 6.1 shows a list of the actions, whose performance was measured.

Action	Semantic
sending a command	time between the sending and the reception of a command
applying pose feedback	time between the sending and the incorporation of pose feedback
reconstruction of environment data	time between the extraction of recorded data and the creation of a virtual object from it
transmission of video	time between the recording of a video frame and the rendering at the client

Figure 5.2: Table of the actions to be measured.

Measured Results

A teleinteraction session was started to perform the measurements. Each teleinteraction session for evaluation approximately took two and a half minutes. An operator would navigate the surrogate through an environment and have it run the reconstruction. The code instrumentations would then record and store the time data to files on the hard drive. The recorded times were averaged to minimize the impact of potential deviations.

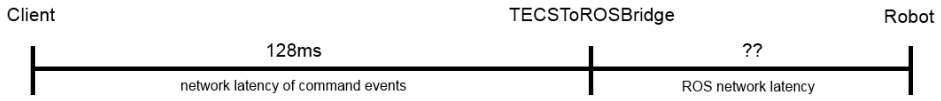


Figure 5.3: Measured performance of the action *sending a command*.

Figure 5.3 illustrates the action of sending a command to the robot and the involved latencies. The transmission of a command consists of the transmission of the TECS event from the client to the TECSToROSBridge and the transmission from there to the robot via the ROS network. The network latency for the transmission of the TECS event is $128ms$. The ROS network latency could not be measured, as the ROS nodes responsible for movements of the surrogate robot could not be instrumented. However, running the teleinteraction prototype suggests that the ROS network introduces a major additional latency of up to a second. This is obviously a delay that is way too high for properly controlling the surrogate. This ROS network latency is introduced due to the use of websockets for communication in the ROS network. To speed up the latency for sent commands, it would be desirable to avoid the use of ROS websockets.

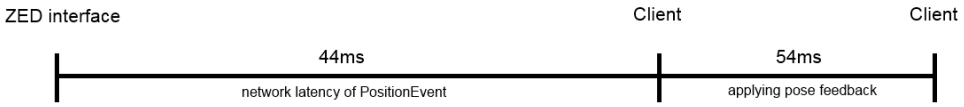


Figure 5.4: Measured performance of the action *applying pose feedback*.

If TECS events could be directly injected into the surrogate's ROS nodes, the major ROS network latency might be avoidable.

Figure 5.4 illustrates the action of applying pose feedback and the involved latencies. Extracted pose feedback is immediately converted into a PositionEvent and sent to the client, where it is then factored into the avatar's position to reduce drift. Transmission of the feedback takes $44ms$, while the time to apply the feedback is $54ms$. The time required to adjust the avatar's pose is directly connected to how drift is reduced. While such small latencies will only have little impact on the teleinteraction, based on the approach that is chosen, delay might be further reduced.

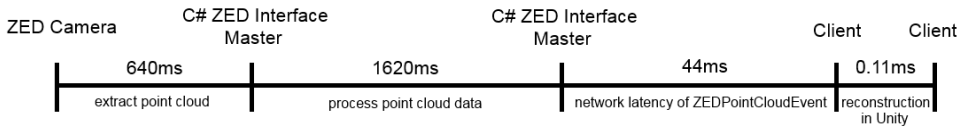


Figure 5.5: Measured performance of the action *reconstruction of environment data*.

Figure 5.5 illustrates the action of reconstructing environment data and the involved latencies. A recorded point cloud is extracted from the ZED Camera's memory and handed over for processing from the ZED interface program. After processing, environment data is sent to the client, where it is used to create a virtual reconstruction of the environment. The extraction of a single point cloud takes $640ms$. This includes the workaround to hand data from the program extracting the data to the program processing the data as byte strings. This is very inefficient, but it was necessary to get access the point data at all. A more direct solution that combines extraction and processing of data in one program would allow for a major speedup. Processing of the point cloud for the reconstruction takes $1620ms$. Since the processing of a point cloud takes longer than it takes for point clouds to be extracted, this creates a bottleneck causing incoming clouds to pile up. The processing certainly needs to be improved to enable proper handling of incoming point cloud data. Due to the reduction of the point data to blocks, network latency is relatively low with $44ms$. The same goes for the creation of the virtual environment from the received data in Unity with a time of $0.11ms$.

Figure 5.6 illustrates the action of transmitting video and the involved latencies. To display a video frame to the operator, it is first extracted from the camera's memory, sent to the client via TECS and then rendered to the display. The extraction of the video

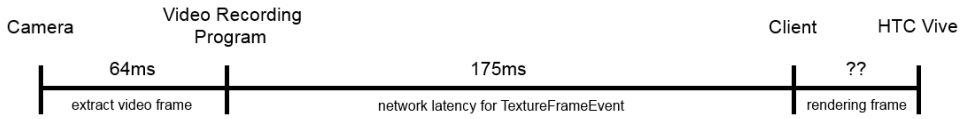


Figure 5.6: Measured performance of the action *transmission of video.*

frame data is performed in $64ms$. Transmission of the frame data to the client takes $175ms$. The time required for rendering the frame to the display was not measure., however it can be assumed to be trivial, as the application successfully runs with a frame rate of 60 frames per second.

The sending of TECS events involves transmission of data over a network. Thus, the network latency naturally increases for longer distances. In this evaluation scenario, surrogate and client were on the same local network. Therefore, the measured latency is relatively small. The difference in the network latency between different evaluated actions is based on the amount of data that has to be transmitted and the current network congestion. While it would be desirable to reduce these high latencies, the underlying network infrastructure can often not be changed. Therefore, it is usually only possible to improve the overall network latency by reducing the amount of data that needs to be transmitted.

6 Conclusion

In this thesis, an approach to long-distance teleinteraction was proposed that aims to avoid the end-to-end latency of visual data between operator and surrogate. The approach uses 3D reconstruction to create a local copy of the surrogate's environment, enabling visualization with low latency. Since latency induced simulator sickness is one of the major challenges, when working with head mounted displays, the proposed approach provides a potential solution to allow the use of head mounted displays for long-distance teleinteraction.

A concept of the proposed teleinteraction approach was designed and then implemented. As table 6.1 shows, the 3D reconstruction process was implemented using three different approaches. A fourth approach was considered but not actually implemented.

	Tango	**ZED Camera**	**Compressed Clouds**	**HoloLens**
integration	Unity application	separate programs for scanning and processing	separate programs for scanning and processing	?
data type	segmented mesh	point cloud	block data	?
success	no	no	yes	?
performance	?	dropped frames in client	sufficient but room for improvements	?

Figure 6.1: Table of the actions to be measured.

The Project Tango scored with simple integrability providing detailed mesh data for reconstructing the surrogate's environment. Still, due to technical issues with the device, the approach would never reach a functional state, preventing the development of a functional prototype.

The ZED Camera succeeded in providing sensible environment data in form of point clouds that could be used for reconstruction. While integrability was lacking, a workaround enabled the integration of the ZED Camera into the teleinteraction system. Due to the large amounts of point cloud data produced by the camera, the actual reconstruction of a virtual environment suffered from severe performance issues preventing a proper application of the prototype.

By developing a compression algorithm to drastically reduce the amount of data necessary for the actual reconstruction, it was possible to improve the overall performance of the teleinteraction system. Using this approach, a functional teleinteraction system, solving the issue of high end-to-end latency in long-distance teleinteraction scenarios,

could be created. The system was evaluated and performance issues and bottlenecks in the different workflows for further improvement were discovered.

The fourth approach, the use of the HoloLens was not implemented. As only one of the implemented approaches yielded a functional prototype, the HoloLens should certainly be considered as an alternative for future research. Using the HoloLens' meshing capabilities, the overall performance of the teleinteraction system could be improved.

7 Future Work

The teleinteraction system that was designed and implemented in the context of this thesis has to be interpreted as a first prototype to build a proof of concept. A user study would be of interest to evaluate user experiences during teleinteraction using the proposed system to locate possible issues with the application of the prototype. Such a user study was prepared, but could not be conducted in the context of this thesis. It should also be worthwhile to try and find ways to improve the overall system. Still, the limitations of current technology and of the approach in general should be kept in mind to identify potential bottlenecks that can and cannot be overcome by improvements of the system itself.

7.1 User Study

A user study was prepared to evaluate the user experience of potential operators of the teleinteraction system. The goal of this thesis is to find a way to counter the network induced latency to avoid the risk of simulator-sickness. However, the user study does not aim to measure the degree of simulator-sickness induced by the teleinteraction prototype. Since local reconstruction is used and visualization of the environment is performed locally, the network latency and the risk of simulator-sickness are avoided. Therefore, what is left to check is whether the implemented approach and the prototype provide sufficient usability and applicability.

Design of the Study

In appendix 8, the study documents can be found. The study consists of a personality test, three scenarios that provide the test subject with a task related to the teleinteraction system, a short feedback on the usability and lastly a feedback on whether the test subject could imagine the application of the system in given scenarios.

The inclusion of a personality test is not directly related to the teleinteraction. However, it is often useful to have information on the subjects personality to deduce possible reasons on why the subject perceives the proposed prototype or approach as it does. The included test is an established quick test that was developed in the context of the *socio-economic panel (SOEP)* [20].

Three scenarios provide slightly different navigation tasks to measure and evaluate the performance of the operator, when trying to navigate to certain locations. Relevant factors are for example, the time required for the task, how often and how long video streaming was used to support the perception of the environment and possibly the number of mistakes the operator makes during execution of the task. Additionally the test subject is asked for feedback on the usability of the system to the task. To take into

account the familiarization of the operator with the prototype, the tasks are shuffled for different test subjects, to normalize the subjects' familiarity over the different scenarios.

After completion of the tasks, some general feedback on the usability is requested. This part of the study also contains short questions to check for potential motion-sickness to make sure there is actually no increased risk of simulator-sickness, when using the implemented prototype.

At the end of the study, the subject is presented four scenarios that describe situations, where the application of a teleinteraction system might be of benefit. The subject is asked to rate the applicability of the system to the described scenario, as well as to rate the if effort can be saved and safety for the operator can be increased.

Conduction of the Study

The test subject is placed in front of a computer with connected HTC Vive and keyboard. The subject will first fill out the participation form, including the *prior experiences* section and the personality test. Then the subject is equipped with the HTC Vive and the keyboard and informed on how to control the surrogate. Then, for each scenario, the subject is informed about the task to perform, the scenario is started and stopped, when the task has been fulfilled. After each scenario, the feedback for the respective scenario will be filled out by the participant. Once all scenarios have been finished, the subject will fill out the general feedback form and lastly rate the described scenarios.

7.2 Improvements and Other Approaches

The implemented teleinteraction system is still a prototype at this point. While the basic functionality is provided, improvement of the prototype might be worthwhile. By improving the overall performance of the system, especially the performance of the 3D reconstruction scheme, the teleinteraction experience could be improved. Also the implementation of alternative reconstruction approaches is of interest. The integration of the HoloLens into the system for 3D reconstruction should be tested as a potentially superior technology compared to the ZED Camera and especially the Project Tango.

Furthermore, the proposed system itself could be extended. As described in the concept, actual interaction with the remote environment via the surrogate is one of the ultimate goals. Thus, the integration of different robots with actual physical interaction capabilities should be researched.

A deficiency of the current system that is amplified by the introduction of actual physical interaction with the environment, is the vulnerability of the controls to latency. A delayed reaction to the operator's commands can impede with the execution of complex or sensitive actions. It would be of interest to research the effects of partial automation of commands. For example, being able to instruct the surrogate to grab a specified item and having the robot carry out the action by its own would remove any impact of the network latency on the action. An improved resistance of the system against latency would support the usability of the system.

Acknowledgements

I would now like to thank the people that supported me during the course of this thesis. First, I would like to thank Professor Wahlster from DFKI Saarbrücken, who gave me the opportunity to write this thesis as an international cooperation with the National Institute for Informatics in Tokyo. I would also like to thank Professor Inamura for welcoming me at his chair in Tokyo, where I spent a major amount of time during the making of this thesis. I would like to thank Boris Brandherm, Rie Ayuzawa and Yukiko Mizoguchi, who helped me with the organization of my stay in Japan. Next, I would like to thank my supervisor Christian Bürckert, who advised me to the full extent of his capabilities. I would also like to thank Dennis Königsmark for assisting me in the spell checking of this thesis. Next I would like to thank my dear friends, who accompanied me during the making of this thesis and kept me motivated. Last but not least, I would like to thank my parents Ullrich Barth and Bettina Barth for all the support up to now, be it mental or financial. I could not have made it this far without them.

Saarbrücken, 17th January 2018

Marvin Barth

8 Appendix

Fragebogen zur Studie Long-Distance Teleinteraction:

Nummer: _____

Alter: _____

Geschlecht: _____

Höchster Bildungsabschluss: _____

Studienfach (wenn Student): _____

			1	2	3	4	5	
1.	Erfahrung mit Computern	keine Erfahrung						viel Erfahrung
2.	Erfahrung mit Virtual Reality	keine Erfahrung						viel Erfahrung
3.	Erfahrung mit Steuerung von Robotern	keine Erfahrung						viel Erfahrung

Persönlichkeit Teil A:

Nummer: _____

Hier sind unterschiedliche Eigenschaften, die eine Person haben kann. Wahrscheinlich werden einige Eigenschaften auf Sie persönlich voll zutreffen und andere überhaupt nicht. Bei wieder anderen sind Sie vielleicht unentschieden. Antworten Sie bitte anhand der folgenden Skala. Der Wert 1 bedeutet "trifft überhaupt nicht zu". Der Wert 7 bedeutet "trifft voll zu". Mit den Werten zwischen 1 und 7 können Sie Ihre Meinung abstufen.

			1	2	3	4	5	
1.	Ich bin jemand, der gründlich arbeitet.	trifft nicht zu						trifft zu
2.	Ich bin jemand, der kommunikativ, gesprächig ist.	trifft nicht zu						trifft zu
3.	Ich bin jemand, der manchmal etwas grob zu anderen ist.	trifft nicht zu						trifft zu
4.	Ich bin jemand, der originell ist, neue Ideen einbringt.	trifft nicht zu						trifft zu
5.	Ich bin jemand, der sich oft Sorgen macht.	trifft nicht zu						trifft zu
6.	Ich bin jemand, der zurückhaltend ist.	trifft nicht zu						trifft zu
7.	Ich bin jemand, der eher faul ist.	trifft nicht zu						trifft zu
8.	Ich bin jemand, der aus sich herausgehen kann, gesellig ist.	trifft nicht zu						trifft zu

Persönlichkeit Teil B:

Nummer: _____

			1	2	3	4	5	
9.	Ich bin jemand, der künstlerische Erfahrungen schätzt.	trifft nicht zu						trifft zu
10.	Ich bin jemand, der leicht nervös wird.	trifft nicht zu						trifft zu
11.	Ich bin jemand, der Aufgaben wirksam und effizient erledigt.	trifft nicht zu						trifft zu
12.	Ich bin jemand, der verzeihen kann.	trifft nicht zu						trifft zu
13.	Ich bin jemand, der rücksichtsvoll und freundlich mit anderen umgeht.	trifft nicht zu						trifft zu
14.	Ich bin jemand, der eine lebhafte Phantasie, Vorstellung hat.	trifft nicht zu						trifft zu
15.	Ich bin jemand, der entspannt ist, mit Stress gut umgehen kann.	trifft nicht zu						trifft zu

Task_A:

Nummer: _____

			1	2	3	4	5	
1.	Ich denke, dass ich dieses System gerne regelmäßig nutzen würde.	trifft nicht zu						trifft zu
2.	Ich fand das System unnötig komplex.	trifft nicht zu						trifft zu
3.	Ich denke, das System war leicht zu benutzen.	trifft nicht zu						trifft zu
4.	Ich denke, ich würde die Unterstützung einer fachkundigen Person benötigen, um das System benutzen zu können.	trifft nicht zu						trifft zu
5.	Ich fand, die verschiedenen Funktionen des Systems waren gut integriert.	trifft nicht zu						trifft zu
6.	Ich halte das System für zu inkonsistent.	trifft nicht zu						trifft zu
7.	Ich glaube, dass die meisten Menschen sehr schnell lernen würden, mit dem System umzugehen.	trifft nicht zu						trifft zu
8.	Ich fand das System sehr umständlich zu benutzen.	trifft nicht zu						trifft zu
9.	Ich fühlte mich bei der Nutzung des Systems sehr sicher.	trifft nicht zu						trifft zu
10.	Ich musste viele Dinge lernen, bevor ich mit dem System arbeiten konnte.	trifft nicht zu						trifft zu

Seite 4 von 9

Task_B:

Nummer: _____

			1	2	3	4	5	
1.	Ich denke, dass ich dieses System gerne regelmäßig nutzen würde.	trifft nicht zu						trifft zu
2.	Ich fand das System unnötig komplex.	trifft nicht zu						trifft zu
3.	Ich denke, das System war leicht zu benutzen.	trifft nicht zu						trifft zu
4.	Ich denke, ich würde die Unterstützung einer fachkundigen Person benötigen, um das System benutzen zu können.	trifft nicht zu						trifft zu
5.	Ich fand, die verschiedenen Funktionen des Systems waren gut integriert.	trifft nicht zu						trifft zu
6.	Ich halte das System für zu inkonsistent.	trifft nicht zu						trifft zu
7.	Ich glaube, dass die meisten Menschen sehr schnell lernen würden, mit dem System umzugehen.	trifft nicht zu						trifft zu
8.	Ich fand das System sehr umständlich zu benutzen.	trifft nicht zu						trifft zu
9.	Ich fühlte mich bei der Nutzung des Systems sehr sicher.	trifft nicht zu						trifft zu
10.	Ich musste viele Dinge lernen, bevor ich mit dem System arbeiten konnte.	trifft nicht zu						trifft zu

Seite 5 von 9

Task_C:

Nummer: _____

			1	2	3	4	5	
1.	Ich denke, dass ich dieses System gerne regelmäßig nutzen würde.	trifft nicht zu						trifft zu
2.	Ich fand das System unnötig komplex.	trifft nicht zu						trifft zu
3.	Ich denke, das System war leicht zu benutzen.	trifft nicht zu						trifft zu
4.	Ich denke, ich würde die Unterstützung einer fachkundigen Person benötigen, um das System benutzen zu können.	trifft nicht zu						trifft zu
5.	Ich fand, die verschiedenen Funktionen des Systems waren gut integriert.	trifft nicht zu						trifft zu
6.	Ich halte das System für zu inkonsistent.	trifft nicht zu						trifft zu
7.	Ich glaube, dass die meisten Menschen sehr schnell lernen würden, mit dem System umzugehen.	trifft nicht zu						trifft zu
8.	Ich fand das System sehr umständlich zu benutzen.	trifft nicht zu						trifft zu
9.	Ich fühlte mich bei der Nutzung des Systems sehr sicher.	trifft nicht zu						trifft zu
10.	Ich musste viele Dinge lernen, bevor ich mit dem System arbeiten konnte.	trifft nicht zu						trifft zu

Allgemein:

Nummer: _____

Nach dem wievielten Versuch hattest du das Gefühl mit dem System gut zu Recht zu kommen?

1	2	3

Wie sehr treffen folgende Aussagen Ihrer Meinung nach zu?

			1	2	3	4	5	
1.	Mir wurde bei der Benutzung des Systems übel.	trifft nicht zu						trifft zu
2.	Mir wurde bei der Benutzung des Systems schwindelig.	trifft nicht zu						trifft zu
3.	Ich finde die Steuerung des Systems zu kompliziert.	trifft nicht zu						trifft zu
4.	Ich wurde bei der Benutzung des Systems seekrank.	trifft nicht zu						trifft zu
5.	Ich finde die Qualität der 3D rekonstruierten Umgebung ausreichend, um mich zu Recht zu finden.	trifft nicht zu						trifft zu

Szenarien:

Nummer: _____

Im Folgenden werden einige mögliche Szenarien beschrieben, in denen der Teleinteraktionsansatz, der in dieser Studie evaluiert werden soll, eingesetzt werden könnte. Lesen Sie die Beschreibung der Szenarios und kreuzen Sie für die jeweilige Aussage an, wie zutreffend Sie diese finden.

Ein deutsches Technikunternehmen sendet Mitarbeiter zur Inspektion eines neu erbauten Fertigungswerkes in Amerika. Da die Inspektion rein observativer Natur ist, könnte statt dem Mitarbeiter der Teleinteraktionsroboter der bereits in Amerika ist, genutzt werden, um die Inspektion aus der Ferne durchzuführen.

			1	2	3	4	5	
1.	Ich glaube, dass es Sinn macht, in diesem Szenario Teleinteraktion zu nutzen.	trifft nicht zu						trifft zu
2.	Ich glaube, dass die Nutzung des Roboters die Aufgabe im Szenario erleichtert.	trifft nicht zu						trifft zu
3.	Ich glaube, dass der Einsatz des Roboters die Arbeitssicherheit erhöht.	trifft nicht zu						trifft zu

Ein Unfall in einem Atomkraftwerk sorgt für radioaktive Verseuchung in der weiträumigen Umgebung. Um die Situation unter Kontrolle zu bringen, müssen Mitarbeiter die verseuchte Anlage betreten. Ein Teleinteraktionsroboter der mit den nötigen Befähigungen ausgestattet ist, könnte statt einem Mitarbeiter die Anlage betreten, um die Situation zu entschärfen.

			1	2	3	4	5	
1.	Ich glaube, dass es Sinn macht, in diesem Szenario Teleinteraktion zu nutzen.	trifft nicht zu						trifft zu
2.	Ich glaube, dass die Nutzung des Roboters die Aufgabe im Szenario erleichtert.	trifft nicht zu						trifft zu
3.	Ich glaube, dass der Einsatz des Roboters die Arbeitssicherheit erhöht.	trifft nicht zu						trifft zu

An einem Bahnhof wird eine Bombe entdeckt und alle Zivilisten werden evakuiert. Spezialisten müssen die Bombe nun entschärfen. Statt selbst die Bombe zu entschärfen, könnten die Spezialisten einen Roboter mit hochauflösender Kamera und entsprechenden Werkzeugen zur Entschärfung fernsteuern, um selbst nicht die Gefahrenzone zu betreten.

			1	2	3	4	5	
1.	Ich glaube, dass es Sinn macht, in diesem Szenario Teleinteraktion zu nutzen.	trifft nicht zu						trifft zu
2.	Ich glaube, dass die Nutzung des Roboters die Aufgabe im Szenario erleichtert.	trifft nicht zu						trifft zu
3.	Ich glaube, dass der Einsatz des Roboters die Arbeitssicherheit erhöht.	trifft nicht zu						trifft zu

Zur Erkundung eines anderen Planeten soll ein entsprechend ausgerüsteter Weltraumroboter eingesetzt werden. Aufgrund der großen Distanz gibt es eine hohe Latenz zwischen Roboter und Erde (3D Rekonstruktion ist weniger stark von Latenz betroffen).

			1	2	3	4	5	
1.	Ich glaube, dass es Sinn macht, in diesem Szenario Teleinteraktion zu nutzen.	trifft nicht zu						trifft zu
2.	Ich glaube, dass die Nutzung des Roboters die Aufgabe im Szenario erleichtert.	trifft nicht zu						trifft zu
3.	Ich glaube, dass der Einsatz des Roboters die Arbeitssicherheit erhöht.	trifft nicht zu						trifft zu

Bibliography

[1] Evan Ackerman. Oculus Rift-Based System Brings True Immersion to Telepresence Robots - IEEE Spectrum, 2015.

[2] Robert S Allison, Laurence R Harris, Michael Jenkin, Urszula Jasiobedzka, and James E Zacher. Tolerance of Temporal Delay in Virtual Environments. 2001 Proceedings of IEEE, 2001.

[3] Pierpaolo Baccichet, Jeonghun Noh, Eric Setton, and Bernd Girod. Content-Aware P2P Video Streaming with Low Latency. Proc. IEEE International Conference on Multimedia & Expo (ICME'07), pages 2–5, 2007.

[4] G. Baier, M. Buss, F. Freyberger, and G. Schmidt. Benefits of combined active stereo vision and haptic telepresence. Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000), 1:702–707, 2000.

[5] Randall E Bailey, J J Trey Arthur Iii, and Steven P Williams. Latency Requirements for Head-Worn Display S / EVS Applications. Proceedings of SPIE, 5424, 2004.

[6] G H Ballantyne. Robotic surgery , telerobotic surgery , telepresence , and telementoring Review of early clinical results. Surgical endoscopy, pages 1389–1402, 2002.

[7] Emily Brown and Paul Cairns. A Grounded Investigation of Game Immersion. CHI'04 extended abstracts on Human factors in computing systems, pages 1297–1300, 2004.

[8] Erik Bylow, Jürgen Sturm, Christian Kerl, Frederik Kahl, and Daniel Cremers. Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions. Robotics: Science and Systems, 2, 2013.

[9] Jean-Rémy Chardonnet, Mohammad Ali Mirzaei, and Frédéric Merienne. Visually Induced Motion Sickness Estimation and Prediction in Virtual Reality using Frequency Components Analysis of Postural Sway Signal. International Conference on Artificial Reality and Telexistence Eurographics Symposium on Virtual Environments, 2015.

[10] Jessie Y C Chen, Ellen C Haas, and Michael J Barnes. Human Performance Issues and User Interface Design for Teleoperated Robots. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 37(6):1231–1245, 2007.

[11] Nusrat Choudhury, Nicholas Gélinas-Phaneuf, Sébastien Delorme, and Rolando Del Maestro. Fundamentals of neurosurgery: Virtual reality tasks for training and evaluation of technical skills. World Neurosurgery, 80(5):9–19, 2013.

[12] Rafael Cisneros, Shin Nakaoka, Mitsuharu Morisawa, Kenji Kaneko, Shuuji Kajita, Takeshi Sakaguchi, Fumio Kanehiro, Rafael Cisneros, Shin Nakaoka, Mitsuharu Morisawa, Kenji Kaneko, Shuuji Kajita, and Takeshi Sakaguchi. Effective teleoperated manipulation for humanoid robots in partially unknown real environments : team AIST-NEDO ' s approach for performing the Plug Task during the DRC Finals real environments : team AIST-NEDO ' s approach for performing the Plug Task. Advanced Robotics, 1864(November):1–15, 2016.

[13] Angela Dai, Matthias Nießner, and Christian Theobalt. BundleFusion : Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration. ACM Transactions on Graphics, 2017.

[14] S. C. de Vries and P Padmos. Steering a simulated unmanned aerial vehicle using a head-slaved camera and HMD. International Society for Optics and Photonics, 3362, 1998.

[15] John V. Draper, David B. Kaber, and John M. Usher. Telepresence. Human Factors: The Journal of the Human Factors and Ergonomics Society, 40(3):354–375, sep 1998.

[16] Mark H Draper, Erik S Viirre, Thomas A Furness, and Valerie J Gawron. Effects of image scale and system time delay on simulator sickness within hea. Human Factors; Spring, 43(1), 2001.

[17] Maksym Dzitsiuk, Robert Maier, Lingni Ma, and Daniel Cremers. De-noising, Stabilizing and Completing 3D Reconstructions On-the-go using Plane Priors. 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017.

[18] Mark Fiala. Pano-Presence for Teleoperation. 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2170–2174, 2005.

[19] Bernd Girod, Mark Kalman, Yi J. Liang, and Rui Zhang. Advances in channel-adaptive video streaming. Wireless Communications and Mobile Computing, 2(6):573–584, 2002.

[20] Angelika Glöckner-Rist. Zusammenstellung sozialwissenschaftlicher Items und Skalen. 2008.

[21] Mario Hermann, Tobias Pentek, and Boris Otto. Design Principles for Industrie 4.0 Scenarios. In 2016 49th Hawaii International Conference on System Sciences (HICSS), pages 3928–3937. IEEE, jan 2016.

[22] Tetsunari Inamura. Simulator platform that enables social interaction simulation — SIGVerse : SocioIntelliGenesis simulator. IEEE/SICE International Symposium on System Integration (SII), 2010.

[23] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion : Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. Proceedings of the 24th annual ACM symposium on User interface software and technology, 2011.

[24] Mark Kalman, Eckehard Steinbach, and Bernd Girod. Adaptive media playout for low-delay video streaming over error-prone channels. IEEE Transactions on Circuits and Systems for Video Technology, 14(6):841–851, 2004.

[25] Michelle R. Kandalaft, Nyaz Didehbani, Daniel C. Krawczyk, Tandra T. Allen, and Sandra B. Chapman. Virtual reality social cognition training for young adults with high-functioning autism. Journal of Autism and Developmental Disorders, 43(1):34–44, 2013.

[26] Christian Kerl and Daniel Cremers. Dense Visual SLAM for RGB-D Cameras. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013.

[27] Sven Kratz. Evaluating Stereoscopic Video with Head Tracking for Immersive Teleoperation of Mobile Telepresence Robots. Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts, pages 4–5, 2015.

[28] Sven Kratz and Fred Rabelo Ferriera. Immersed Remotely : Evaluating the Use of Head Mounted Devices for Remote Collaboration in Robotic Telepresence. 2016 25th IEEE International Symposium, 2016.

[29] Oh Hun Kwon, Seong Yong Koo, Young Geun Kim, and Dong Soo Kwon. Telepresence robot system for english tutoring. Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts, ARSO, pages 152–155, 2010.

[30] Daniel Leyzberg, Samuel Spaulding, Mariya Toneva, and Brian Scassellati. The Physical Presence of a Robot Tutor Increases Cognitive Learning Gains. 34th Annual Conference of the Cognitive Science Society, (1):1882–1887, 2012.

[31] Y. J. Liang and B. Girod. Network-adaptive low-latency video communication over best-effort networks. IEEE Transactions on Circuits and Systems for Video Technology, 16(1):72–81, 2006.

[32] Yi J. Liang, Markus Flierl, and Bernd Girod. LOW-LATENCY VIDEO TRANSMISSION OVER LOSSY PACKET NETWORKS USING RATE-DISTORTION OPTIMIZED REFERENCE PICTURE SELECTION. IEEE Proc. International Conference on Image Processing, 2002.

[33] Maria G. Martini, Chaminda T E R Hewage, Moustafa M. Nasralla, Ralph Smith, Iain Jourdan, and Timothy Rockall. 3-D robotic tele-surgery and training over next generation wireless networks. Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS, pages 6244–6247, 2013.

[34] Elias Matsas and George Christopher Vosniakos. Design of a virtual reality training system for human-robot collaboration in manufacturing tasks. International Journal on Interactive Design and Manufacturing, 2015.

[35] F Michaud, P Boissy, and D Labont. Telepresence Robot for Home Care Assistance. AAAI Spring Symposium: Multidisciplinary Collaboration for Socially Assistive Robotics, 2007.

[36] Marvin Minsky. Telepresence, 1980.

[37] Yoshiaki Mizuchi and Tetsunari Inamura. Cloud-based Multimodal Human-Robot Interaction Simulator Utilizing ROS and Unity Frameworks. RoboCup International Symposium, 2017.

[38] Jason Moss. CHARACTERISTICS OF HEAD MOUNTED DISPLAYS AND THEIR EFFECTS ON. Human factors, 2011.

[39] Laurent A. Nguyen, Maria Bualat, Laurence J. Edwards, Lorenzo Flueckiger, Charles Neveu, Kurt Schwehr, Michael D. Wagner, and Eric Zbinden. Virtual reality interfaces for visualization and control of remote vehicles. Autonomous Robots, 11(1):59–68, 2001.

[40] E. Paulos and J. Canny. Delivering real reality to the World Wide Web via telerobotics. Proceedings of IEEE International Conference on Robotics and Automation, 2(April):1694–1699, 1996.

[41] Tomislav Pejsa, Julian Kantor, Hrvoje Benko, Eyal Ofek, and Andrew Wilson. Room2Room : Enabling Life - Size Telepresence in a Projected Augmented Reality Environment. Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing, (Figure 1):1716–1725, 2016.

[42] Corey Pittman and Joseph J Laviola Jr. Exploring Head Tracked Head Mounted Displays for First Person Robot Teleoperation. AeroSense'97, International Society for Optics and Photonics, 2014.

[43] Irene Rae and Leila Takayama. In-body Experiences : Embodiment , Control , and Trust in Robot-Mediated Communication. In-body Experiences : Embodiment , Control , and Trust in Robot-Mediated Communication, 2013.

[44] Soc Robot, Annica Kristoffersson, Kerstin Severinson-eklundh, and Amy Loutfi. Measuring the Quality of Interaction in Mobile Robotic Telepresence : A Pilot ' s Perspective. International Journal of Social Robotics, 2013.

[45] Radu Bogdan Rusu and Steve Cousins. 3D is here : Point Cloud Library (PCL). 2011 IEEE International Conference on Robotics and automation (ICRA), 2011.

[46] Thomas Sch, H Christian, and Marc Pollefeys. 3D Modeling on the Go : Interactive 3D Reconstruction of Large-Scale Scenes on Mobile Devices. 2015 International Conference on 3D Vision (3DV), 2015.

[47] E Setton, Taesang Yoo, Xiaoqing Zhu, A Goldsmith, and B Girod. Cross-layer design of ad hoc networks for real-time video streaming. IEEE Wireless Communications, 12(4):59–65, 2005.

[48] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. Thrift: Scalable cross-language services implementation. Facebook White Paper, page 8, 2007.

[49] Eckehard Steinbach, Niko Färber, and Bernd Girod. Adaptive Media Playout for Low Latency Video Streaming. IEEE Proc. International Conference on Image Processing, 2001.

[50] Jonathan Steuer. Defining Virtual Reality: Dimensions Determining Telepresence. Journal of communication, pages 1–25, 1992.

[51] Jeffrey Tan, Chuan Too, and Tetsunari Inamura. SIGVerse - A Simulation Platform for Human-Robot Interaction. pages 3–6, 2011.

[52] Graham Walker. Telepresence — the Future of Telephony. In Digital Media: The Future, pages 281–296. Springer London, London, 2000.

[53] Mingxin Yu, Yingzi Lin, Xiangzhou Wang, David Schmidt, and Yu Wang. Human-Robot Interaction Based on Gaze Gestures for the Drone Teleoperation. Journal of Eye Movement Research, 7(4):1–14, 2014.

[54] Yizhong Zhang. Online Structure Analysis for Real-time Indoor Scene Reconstruction. ACM Transactions on Graphics, 2015.