TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Teleoperation of a Humanoid Robot Using Oculus Rift and Leap Motion

DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Ao. Univ.-Prof. Dipl.-Ing. Dr. techn. M. Vincze
Dipl.-Ing. Dr. techn. M. Zillich

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Matthias Hirschmanner
Matr.-Nr.: 0927170
Grassigasse 7/15
1140 Wien

Wien, 8. Juni 2017

Vision for Robotics Group
A-1040 Wien, Gusshausstr. 27, Internet: http://www.acin.tuwien.ac.at

Vorwort

Diese Diplomarbeit wurde als Abschlussarbeit des Studiums Energie- und Automatisierungstechnik verfasst. Ich möchte mich an dieser Stelle bei allen bedanken, die zum Entstehen dieser Arbeit beigetragen haben.

Mein Dank gilt meinem Professor Markus Vincze, der diese Arbeit ermöglicht hat und auch einen neuen Roboter aus dem Ärmel gezaubert hat, als derjenige den Geist aufgab, mit dem ich diese Arbeit begonnen habe. Ich möchte mich auch bei meinem Betreuer Michael Zillich bedanken, der mich besonders am Ende dieser Arbeit unterstützt hat. Sie gaben mir außerdem die Chance zur Mitarbeit bei Projekten in der Vision for Robotics Group, die mir Einblicke in den wissenschaftlichen Alltag gewährte.

Ebenfalls möchte ich mich bei Dimitrios Prodromou bedanken, der die Idee zu dieser Arbeit hatte. Besonders in der Anfangsphase hat er mich unterstützt und sein Enthusiasmus hat mir immer wieder neue Motivation gegeben. Er hat es mir auch ermöglicht, die Arbeit in Paris bei der Pepper World 2017 vorzustellen.

Ich möchte mich bei meinen Studienkollegen bedanken, die mich einen Teil des Weges begleitet haben, allen voran Felix, Lukas und Simon. Ohne sie wäre ich wohl an einigen Lehrveranstaltungen verzweifelt und auch außerhalb der Universität habe ich sie als Freunde sehr zu schätzen gelernt.

Ein ganz besonderer Dank gilt meiner Familie und insbesondere meinen Eltern, die mir das Studium ermöglicht haben und mich dabei unterstützt haben, es so zu betreiben, wie ich es für richtig hielt. Danke auch an Monika, die mich in schwierigen Phasen während dieser Arbeit motivierte und zur Seite stand.

Matthias Hirschmanner

Wien, 8. Juni 2017

Abstract

This thesis presents a system to operate a humanoid robot from a distance. Teleoperated robots are used in environments inaccessible to humans for tasks which cannot be performed autonomously. Because of the many degrees of freedom of humanoid robots, manual teleoperation methods with keyboards, joysticks and screens are cumbersome to use. The introduced solution is a first-person teleoperation application where the robot imitates the movements of the user's upper body. The focus is on providing an intuitive user experience with commercially available consumer-grade electronics: the Oculus Rift virtual-reality headset with the Leap Motion 3D hand tracking sensor mounted. The user receives visual feedback from the robot's cameras inside the Oculus Rift which tracks the user's head pose. The robot's head pose and therefore the camera direction is controlled intuitively by the user turning his head. We introduce a method for a representation of the video feed in a virtual 3D space to avoid cyber-sickness. The user's arm movements are tracked by the Leap Motion sensor. We derive the necessary robot joint angles for natural imitation of the user's arm posture from the tracked joint positions and direction vectors. The system is able to imitate human motions with high precision and low latency. It was implemented for the humanoid robots Romeo and Pepper manufactured by SoftBank Robotics. Over 100 users tested the system with each robot. Each of them was able to control the robots' upper-body within seconds without any training.

Kurzzusammenfassung

In dieser Arbeit wird ein System vorgestellt, um einen humanoiden Roboter von einem entfernten Ort aus zu steuern. Teleoperierte Roboter werden in Umgebungen eingesetzt, die für Menschen nicht zugänglich sind, um Aufgaben zu erfüllen, die ein Roboter nicht autonom verrichten kann. Teleoperation per Tastatur, Joystick und Bildschirm ist aufgrund der vielen Freiheitsgrade eines humanoiden Roboters umständlich. Beim vorgestellten System imitiert der Roboter die Bewegungen des Oberkörpers des Benutzers, welcher die Umgebung des Roboters aus dessen Perspektive wahrnimmt. Der Fokus liegt darauf, eine intuitive Art der Steuerung mit günstiger Unterhaltungselektronik umzusetzen: die Oculus Rift Virtual-Reality-Brille und der Leap Motion 3D Hand-Tracking Sensor. Die Kamerabilder des Roboters werden in der Oculus Rift dargestellt, die auch die Orientierung des Kopfes misst. Die gemessen Pose des Kopfes wird auf den Roboter übertragen. Damit beeinflusst der Benutzer die Kamerarichtung auf eine natürliche Weise. Wir stellen eine Möglichkeit zur Darstellung des Videos in einer virtuellen Welt vor, um Unwohlsein zu vermeiden. Die Armbewegungen des Benutzers werden mit der Leap Motion gemessen. Wir berechnen die nötigen Gelenkwinkel aus den Messungen der Gelenkpositionen und der Richtungsvektoren so, dass der Roboter die Stellung des Armes imitiert. Das System ist in der Lage, menschliche Bewegungen mit hoher Präzision und niedriger Latenz zu imitieren. Es wurde für die humanoiden Roboter Romeo und Pepper der Firma SoftBank Robotics implementiert. Über 100 Personen pro Roboter haben das System bereits getestet. Jeder von ihnen konnte innerhalb kürzester Zeit, ohne jegliches Training, den Oberkörper der Roboter kontrollieren.

Contents

1 Introduction 1

1.1 Motivation . . . 1

1.2 Problem Statement . . . 2

1.3 Scope of This Thesis . . . 2

1.4 Chapter Organization . . . 3

2 Related Work 5

2.1 History of Teleoperation . . . 5

2.2 Current Work with Humanoid Robots . . . 6

2.2.1 Robot Kinematics for Teleoperation . . . 8

2.2.2 Capture Systems for Teleoperation . . . 10

2.2.3 Applications Using Teleoperation . . . 17

3 Concept 20

3.1 Leap Motion . . . 21

3.1.1 Leap SDK and Orion . . . 23

3.2 Oculus Rift . . . 24

3.2.1 Oculus Rift Development Kit 2 . . . 25

3.2.2 Virtual Reality Considerations . . . 26

3.2.3 Oculus SDK 0.5 . . . 27

3.3 Humanoid Robots by SoftBank Robotics . . . 28

3.3.1 Project Romeo . . . 29

3.3.2 Pepper Robot . . . 30

3.3.3 NAOqi . . . 33

3.4 Teleoperation Setup . . . 35

4 Implementation 37

4.1 Imitation of User Motion . . . 37

4.1.1 Homogeneous Transformation . . . 37

4.1.2 Algorithm for Robot Joint Angle Calculation . . . 40

4.1.3 Filter . . . 45

4.2 User Interface . . . 45

4.2.1 OpenGL API . . . 46

4.2.2 Virtual World for Video Feed . . . 49

4.3 Integration of Software Modules 52

5 Experiments **55**

5.1 Experiments in the Simulator 56

5.2 Experiments on the Robot 60

5.3 User Study 63

5.3.1 Future User Study 65

6 Conclusion **67**

6.1 Imitation of User Motion 67

6.2 User Interface 68

6.3 Future Work 68

List of Figures

2.1 Handyman twirling a hoola hoop 6
2.2 Robonaut mounted on vehicle and teleoperated 8
2.3 Teacher (left) and Android Repliee Q2 (right) with markers [27] 11
2.4 Sarcos humanoid robot and actor with motion capture markers performing “little teapot” dance [29] 12
2.5 Online Imitation by humanoid ARMAR-IIIb using a markerless vision system for motion capturing by Do et al. [30] 13
2.6 Imitation with Nao Robot and Kinect Camera by Zuher and Romero [34] 14
2.7 Whole-body imitation with Nao Robot and Kinect camera by Ou et al. [35] 14
2.8 Jaco robotic arm controlled with Leap Motion sensor [38] 15
2.9 Nao robot controlled with Leap Motion sensor by Yu et al. [39] . 16
2.10 Whole-body imitation with Nao Robot and Xsens MVN motion-capture suit by Koenemann, Burget and Bennewitz [41] 17
2.11 HRP-1S performs a Japanese dance which was recorded beforehand with a marker-based motion capturing system [43] 18
2.12 HOAP-2 lifting a box after learning the movement from a human demonstrator [44] 18
2.13 The user wears an Oculus Rift virtual-reality headset and a sensor glove to teleoperate the iCub robot; the user’s movements are tracked by the Kinect camera [45] 19

3.1 Hardware setup: the user wears the Oculus Rift virtual-reality headset with the Leap Motion sensor attached; the Pepper robot imitates the user’s movements and provides the camera images displayed in the Oculus Rift 20
3.2 Opened Leap Motion controller exposes two infrared cameras and three infrared LEDs 22
3.3 Visualisation of a tracked arm by the Leap Motion sensor 23
3.4 Oculus Rift Development Kit 2 25
3.5 The rendered barrel distortion cancels out the pincushion distortion introduced by the lenses of the Oculus Rift 27

3.6 A scene rendered for the Oculus Rift with barrel distortion and colour shift applied to cancel out effects of the lenses 28
3.7 Humanoid robots by SoftBank Robotics (from left) Nao, Romeo and Pepper 29
3.8 Joints of Romeo's upper body 30
3.9 Joints of Pepper 32
3.10 Structure of the NAOqi Framework 34
3.11 Communication inside NAOqi framework 35
3.12 Setup of the different hardware components used for teleoperation and which kind of data is sent between the devices 36

4.1 Vector relative to frame and coordinate transformation. 38
4.2 Coordinate frames on the user side, and the measured positions and directions of the Leap Motion that are used in the algorithm 40
4.3 Coordinate frames on the robot side 42
4.4 The stages of the OpenGL rendering pipeline 46
4.5 Transformation matrices used in OpenGL 47
4.6 Perspective Projection 48
4.7 Textures used for the skybox 49
4.8 Virtual screens with robot's camera images and virtual camera moving in skybox 50
4.9 Rendered scene with robot's camera images in the virtual environment without distortion correction 51
4.10 Rendered scene with robot's camera images in virtual environment for Oculus Rift 52
4.11 The software modules for Imitation of User Motion and User Interface run in parallel in different threads and receive inputs from the used hardware components; Camera Thread and Joint Thread buffer the camera images and the joint measurements respectively to overcome latency issues 53

5.1 Motion sequence imitated by Pepper 55
5.2 Screenshot of Choregraphe with simulated robot 56
5.3 Comparison of the palm position of the user and the simulated Romeo robot during horizontal eight movement 57
5.4 Comparison of the tracked palm position, the simulated Romeo's palm position and Romeo's scaled palm position during a movement for each coordinate axis 58
5.5 Desired and measured joint angles of the simulated Romeo robot during movement 59

5.6 Comparison of the tracked palm position, Pepper's palm position and Pepper's scaled palm position during a movement for each coordinate axis 61
5.7 Desired and measured joint angles of Pepper during movement . 62
5.8 Demo setting of teleoperation system with Romeo robot 64

List of Tables

3.1 Comparison of the range of arm joints between Romeo and a human . . . 31
3.2 Comparison of the range of head joints between Romeo and a human . . . 31
3.3 Comparison of the range of arm joints between Pepper and a human . . . 33
3.4 Comparison of the range of head joints between Pepper and a human . . . 33

1 Introduction

Today, we are on the verge of having robots in our everyday life. Autonomous vacuum cleaners or lawn mowers are available for consumers and so affordable that they can be found as presents under lazy college students' Christmas trees. Humanoid robots are a category of robots that resemble humans in appearance. A wide variety of humanoid robots exist. However, their capabilities are still very limited and they do not meet many people's expectations which are based on science fiction movies. Teleoperation is a method, where the robot is controlled by a human being from a distance to complete tasks which cannot be performed autonomously and to overcome problems the robot might run into.

1.1 Motivation

Humanoid robots have a few advantages over specialised machine-like robots such as industrial robots. Humans are accustomed to interacting with each other, and therefore a humanoid robot might appeal more to us than a faceless machine. Additionally, our surroundings are customised for the human body (stairs, dimensions of doors, tools). A humanoid robot can, in theory, operate in the same environment and use the same tools as we do [1].
However, the capabilities of robots to act autonomously are still very limited, especially in an unknown environment. A human supervisor can monitor and support the robot. The act of controlling a system from a distance is called teleoperation. This can be useful if the environment is too dangerous for humans, for example in disaster-response scenarios[1]. A human can analyse complicated situations much faster than a robot. Camera images are sent to the teleoperator who is able to analyse the situation and react appropriately.Additionally, the teleoperated robot can be used to communicate with people from a distance. In contrast to a simple videoconference, a mobile robot enables the user to look and move around freely. A humanoid robot may also enable nonverbal communication such as gestures or facial expressions. This form of teleoperation is called telepresence, due to the similarities of being present in person.
The introduced teleoperation application imitates human movements which

[1] `http://www.darpa.mil/program/darpa-robotics-challenge`

can be used to bring human-like movements to the robot. The many degrees-of-freedom of a humanoid robot make it difficult to program specific motions manually. Because of the similarities to the human body, an approach is to have the robot learn motions by imitating human motions. This is called learning from demonstration and seems natural to us because we learn by observing and being shown how to do something. Nobody learns how to ride a bike by reading a book. The first step of this learning process for the robot is to copy the human motion as best as possible and then adapt it in a way to accomplish the desired task.
Another application of teleoperation is to use it in human-robot-interaction research in so-called Wizard of Oz settings [2]. An examiner controls the robot from a different room to simulate a more advanced robot. It is observed how a test person interacts with the robotic puppet. This way, social interaction can be studied safely for example in an early development stage of a robot.

1.2 Problem Statement

The many degrees of freedom of a humanoid robot pose a problem when operating the robot from a distance. Classic teleoperation applications with a joystick and camera images displayed on a computer screen are cumbersome and difficult to use.
To generate human-like motions on the robot by learning through demonstration, a system for human imitation is necessary.
In this thesis, an intuitive method to control the upper body of a humanoid robot is introduced, utilising the capabilities of modern consumer hardware.

1.3 Scope of This Thesis

Intuitive Control: The focus of this thesis is to have an intuitive way to control a humanoid robot. Therefore, we want to imitate the posture of the user's upper body as closely as possible. A different approach would be to follow the end-effector position of the user. However, it was deemed more important to give the user the feeling of the robot copying his movements instead of optimising for the end-effector position. This way, the user acts as a feedback loop. If he wants to reach a specific Point, he would adjust his hand position according to the visual feedback of the robot's hand. Low latency and hardly any jitter are required for this to work. The Leap Motion hand tracking sensor is used to track the user's arm movements. It shows good results and costs a fraction of the price of professional motion tracking hardware.

Virtual Reality: The Oculus Rift virtual-reality (VR) headset is used to display the camera images of the robot's cameras as a visual feedback for the user. Additionally, the user's head pose is tracked by the headset. The robot copies the head movements of the user. This way, the user naturally adjusts the direction of the cameras, positioned on the head of the robot. The Leap Motion sensor is attached to the headset because it is optimised for this use case.

Telepresence for Object Manipulation: The user can teleoperate the upper body of the robot for example to manipulate an object. The system is not optimised for communication with other people via the robot. No audio transmission or imitation of gaze direction is implemented. The teleoperation is restricted to the head and the arms. So the movements of the feet, as well as movements of the upper body such as leaning forward, are not imitated by the robot. The system is not optimised to be used over a long distance. The delays introduced by the application itself are kept to a minimum, but delays resulting from data communication are not addressed (for example by compression of the robot's video feed).

Robots: The introduced system is implemented for the humanoid robots Romeo[2] and Pepper[3] from SoftBank Robotics. We tested it with these robots in the simulator and on the real robots. Over 100 persons used the teleoperation with each robot in short demo settings. The algorithms and methods introduced in this thesis are independent of a specific robotic platform. An interface to use the teleoperation algorithm with other robots for example via the Robotic Operating System (ROS) is not implemented yet.

1.4 Chapter Organization

Chapter 2 introduces different systems for teleoperation and human imitation with a focus on humanoid robots. It starts with a short history of teleoperation. The recent works are grouped into interesting approaches in the domains of robot kinematics, capture systems and complete systems. In Chapter 3 the used hardware is described with its corresponding software tools, and how the different components are linked together. Chapter 4 presents the implementation of the teleoperation system. First, an introduction to coordinate transformation is given followed by the methodology to convert the sensor

[2]`http://projetromeo.com/en`
[3]`https://www.ald.softbankrobotics.com/en/cool-robots/pepper`

information to motion on the robot. Afterwards, the design decisions for the user interface as well as its implementation are explained. The experiments for the teleoperation system are described in Chapter 5, with results from the simulator, a real robot and user testing. Chapter 6 concludes the thesis, and the introduced teleoperation system is analysed. Possibilities for future work are discussed.

2 Related Work

Robots excel at performing the same acts over and over again, e. g. industrial robotic arms for welding or spray-painting. However, many tasks are unpredictable, and humans are in general better at adapting to unexpected circumstances than an automatic device. Teleoperation combines the capabilities of robots and humans. A remote operator oversees and controls the actions of a robot. In this chapter, related work in the domains of teleoperation and human imitation is introduced with a focus on teleoperation of humanoid robots, the used sensors and the different solutions of the inverse kinematic problem.
The Greek prefix tele means “at a distance” and the combined word teleoperation stands for operating at a distance. Sheridan [3] defines teleoperation as “the extension of a person’s sensing and manipulation capability to a remote location”. Telepresence enhances these capabilities, so “the operator feels physically present at the remote site”.

2.1 History of Teleoperation

Robots can be used in hazardous and dangerous surroundings like space, deep ocean or nuclear environments. Therefore, teleoperation has been researched for the past 60 years. According to Sheridan [3], the first master-slave teleoperators were developed by Goertz in the mid-1940s. It was a mechanical linkage to handle nuclear material from outside the room. In 1954 Goertz and Thompson [4] replaced the mechanical connection with an electrical servomechanism. In the 1960s a lot of research has been done in teleoperated devices. A notable example is the Handyman by Mosher in 1964 [5]. It is a teleoperated machine with two electromechanical arms with ten degrees of freedom each (two in each of the two fingers) and force feedback (Figure 2.1). These servomechanisms were also applied in human prostheses. The first significant arm prosthesis that used electrical signals created by the human body to control it was built by Kobrinski in 1960 [6].
A head mounted display was introduced in 1968 by Sutherland [7]. It consists of two miniature CRT (cathode ray tube) screens and an ultrasonic head position sensor to recognise changes in the head position and adjust the shown image accordingly. In the 1960s telemanipulators and cameras were attached

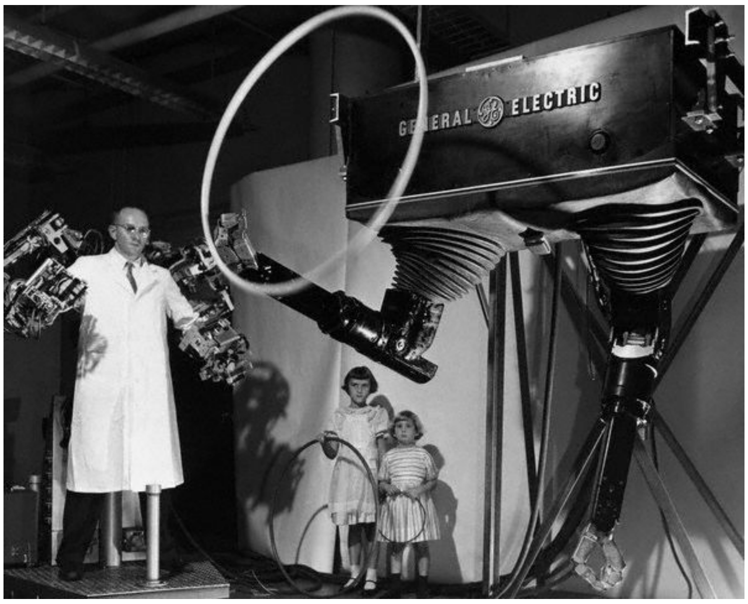


Figure 2.1: Handyman twirling a hoola hoop[1]

to submarines for deep sea exploration as well as rescue and recovery missions. In 1966 the US Navy's CURV (Cable-Controlled Underwater Recovery Vehicle) recovered a nuclear bomb that was dropped in the Mediterranean Sea after an accident while refuelling in mid-air [3]. Teleoperated devices were also used in space exploration. Following the first missions that performed soft landings on the moon (Surveyor by the USA, Luna by the Soviet Union) the first telerobot landed on the moon on November 17, 1970. Lunokhod 1 was an unmanned moon rover by the Soviet Union that was able to move and transmit camera images and other sensor information back to earth. It was operated by a ground crew of 5 members [8], [9].

2.2 Current Work with Humanoid Robots

Humanoid robots are robots that resemble humans in appearance. Today, most robots in use are specialised for a particular task, which often defines the shape of the robot. Therefore, they do not look human-like, e.g. industrial robotic arms, robotic vacuum cleaners and surgical robots.
However, there are several reasons to design humanoid robots. The world

[1]http://cyberneticzoo.com/man-amplifiers/1958-9-ge-handyman-ralph-mosher-american/

around us is created for human locomotion. A human-like structure would enable robots to move around a house with stairs, doors and furniture, which often cause difficulties for wheeled robots [10], [11]. They are also able to manipulate and interact with objects designed for humans. Research is done for using humanoid robots in disaster response scenarios for situations like ladder climbing or car ingress [12].
Another aspect of using humanoid robots is human-robot interaction. A humanoid robot enables natural interaction, which increases social acceptance [13]. A humanoid robot enhances vocal interaction by sending paralinguistic communication signals, such as facial expression, gestures, gaze direction and posture [14]. It is even argued, that anthropomorphic design increases trust in a machine [15].
There is a wide variety of applications for teleoperation of humanoid robots, from using them in hazardous disaster situations in unknown environments (e.g. nuclear accidents) to interacting with other humans from far away. Humanoid robots have many degrees of freedom (e.g. the SoftBank Robotics Romeo has 37 degrees of freedom in total) so they are hard to be operated with controls, such as joysticks and keyboards. Different sensory systems (e.g. cameras, tracking suits) are used to capture human motion and transfer it to the robot. These motion sequences are also often used to teach the robot human-like movements, a technique called "Learning from Demonstration" [16]. Already in 1997, Schaal [17] showed its capabilities with a robotic arm which was able to balance an inverse pendulum on the first try after a 30 seconds demonstration.
A well-known example of teleoperation of a humanoid robot is NASA's Robonaut [18], [19]. It was developed to work in space alongside astronauts and use tools made for humans. The second version of Robonaut was deployed on the International Space Station in February 2011 [20]. The main operational mode is teleoperation from the accompanying crew as well as from ground control. The images of the robot's stereo cameras can be displayed in a head mounted display worn by the teleoperator to provide visual feedback (Figure 2.2). The operator's head position is tracked by a six-axis magnetic posture sensor on the helmet. This information is used to control the neck. The same sensors are positioned on the back of the operator's hand to track its position. The user wears a glove which captures the finger motion with resistive flex sensors which is mapped to the robot's hand [21]. This results in a very natural teleoperation. However, many different devices need to be attached to the operator, which might feel intrusive and encumbering. Therefore Martinez et al. [22] developed a system to track the arm movement with a monocular camera. They used a model of the right arm of a human and applied the Maximum-Likelihood motion estimation algorithm to control the right arm of a virtual Robonaut. They describe a tracking error of the position of shoulder, elbow and wrist in the

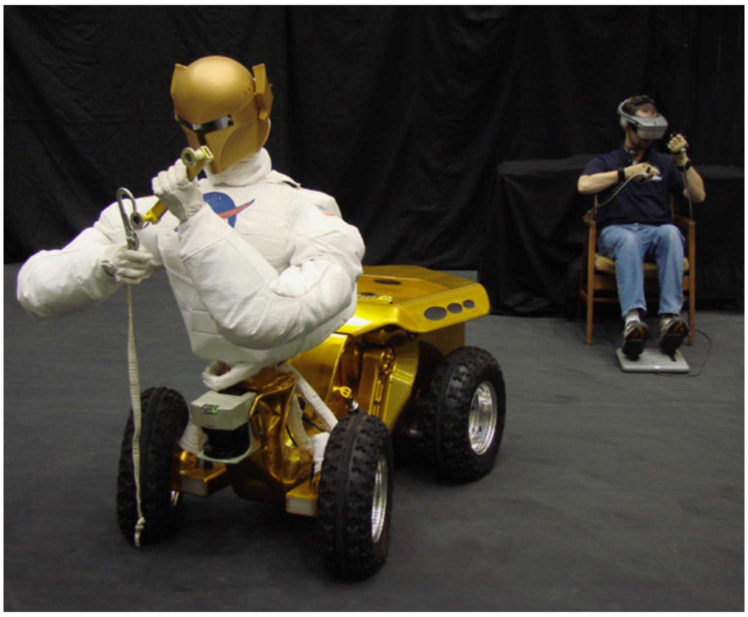


Figure 2.2: Robonaut mounted on vehicle and teleoperated[2]

millimetre range when using synthetic image sequences. However, this system is not able to track fingers, must be configured to the anatomical measurements of the teleoperator and the camera is fixed. In our teleoperation application, only the shoulder position is dependent on the dimensions of the teleoperator, the fingers are tracked and the camera is moved when turning the head.

2.2.1 Robot Kinematics for Teleoperation

Teleoperation requires transferring human motions to robot motions. A motion-capture system delivers a representation of human movements, often as joint positions in a Cartesian space. A few common motion-capture systems are described in Section 2.2.2. For now, consider the human motion data is given. A robot is usually controlled by setting the angles of the different joints. Forward kinematics is the domain of determining the pose of the end-effectors in Cartesian space for given joint parameters. The calculation from Cartesian space to robot joint variable space is called inverse kinematics. Different approaches exist to solve the inverse kinematic problem either analytically or numerically [11].
The most intuitive method is to use the forward kinematic equations which

[2] `http://robonaut.jsc.nasa.gov/R1/sub/telepresence.asp`

calculate the position and orientation $\mathbf{x}$ of the end-effectors from the joint angles $\boldsymbol{\theta}$

$$\mathbf{x} = f(\boldsymbol{\theta}).$$

The equations depend on the robot's dimensions and joints. These equations are solved for the joint angles analytically [23]. However, for complex kinematic chains, it is often too complicated to obtain an analytic solution for the inverse kinematic problem due to non-linear terms. Robot manipulators are often redundant, which means the robot has more degrees of freedom than would be required to reach a specified position in Cartesian Space. This enables the robot to avoid obstacles, but is problematic for solving the inverse kinematic problem, because there are multiple solutions in the joint space for one pose of the end-effector.

The inverse kinematic problem can also be solved by numerical methods, by treating it as an optimisation problem. A specific cost function is defined, usually with the robot's joint angles as input arguments. A solution is considered optimal if it minimises this function. Baillieul and Martin [24] distinguish between global and local methods. Global methods calculate an optimal path in joint space for a given movement in operational space. This approach is not suitable for real-time applications because of its non-causality. Local methods compute an optimal change of the joints $\dot{\theta}$ and therefore reach the desired position iteratively. Whitney [25] proposed the rate control scheme in 1969 which uses the Jacobian and its pseudoinverse.

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\boldsymbol{\theta}}, \qquad \mathbf{J} = \frac{\partial}{\partial \theta} f(\boldsymbol{\theta}).$$

If $\mathbf{J}$ is invertible at a certain point, $\dot{\theta}$ can be calculated by

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}\dot{\mathbf{x}}.$$

If $\mathbf{J}$ is rank deficient at a point, it cannot be inverted. Such a point is called singularity and should be avoided in a path. For redundant manipulators $\mathbf{J}$ is not quadratic and therefore not invertible. In this case, additional optimality criteria are added to obtain a pseudo-inverse.

Capturing systems used for teleoperation of humanoid robots usually provide more information than the pose of the end-effector, e. g. the position of various markers placed on the human. Riley et al. [26] use a suit with coloured patches and a 3D vision system to detect motion and transfer it to the Sarcos humanoid robot. Either the robot's internal cameras or external cameras are used. The positions of 7 patches on the human body are obtained. The equivalent positions on the robot are estimated for the current joint configuration. The cost function is the difference between the estimated positions of the patches on the robot

and the measured positions on the human. The joint angles are calculated by minimising this function iteratively by using the Jacobian similar to the approach described above. The problem is split into simpler subproblems by exploiting the dependencies of body geometry. The operator has to start from a defined starting position. If the operator moves too fast, the robot will stop or return to this starting position and wait for the human to go back to this position as well. This approach enables a real-time imitation of full-body motion, but due to the coloured patches, only 7 degrees of freedom are controlled. In our teleoperation application for the Romeo robot 19 degrees of freedom are controlled.
Another possibility to calculate the inverse kinematics in a teleoperation setting is to use neural networks. These networks have to be trained in the beginning to map the human's posture to the robot. Matsui et al. [27] use neural networks to generate motion for a humanoid robot with a focus on appearing as human-like as possible. They developed the android Repliee Q2 which looks like a Japanese woman and can be seen in Figure 2.3. They use a marker-based system to capture motion. They argue that a transformation of the obtained joint angles of the human teacher to the robot would yield unnatural movements of the robot because of the different kinematic structure. Additionally, human joint's centre of rotation changes when rotating, which is not considered by other joint-based approaches. Therefore, markers are attached to the robot in the same positions as on the teacher. A three-layer neural network is trained to construct a mapping from the performer's posture to the robot's input. The training sequence consists of 50000 samples. The robot moves randomly and the error between the marker positions of the human and the robot is used to train the neural network. In the experimental setup, 21 degrees of freedom of the robot are controlled[3]. The wrist and the fingers are not controlled. The results show that human motion can be transferred to the android but is restricted by the possible speed of motion.

2.2.2 Capture Systems for Teleoperation

Motion capture systems can roughly be divided into optical and non-optical systems. Moeslund, Hilton and Krüger [28] group the applications of motion capture systems into three groups: surveillance, control and analysis. Surveillance applications automatically monitor places such as airports and parking spaces. In control applications, the body posture and movements are used for controlling functionalities. For example, the user is interacting with a

[3]Each shoulder joint has 5 degrees of freedom. Additionally to the rotation, it can move upwards/downwards and back/forth.

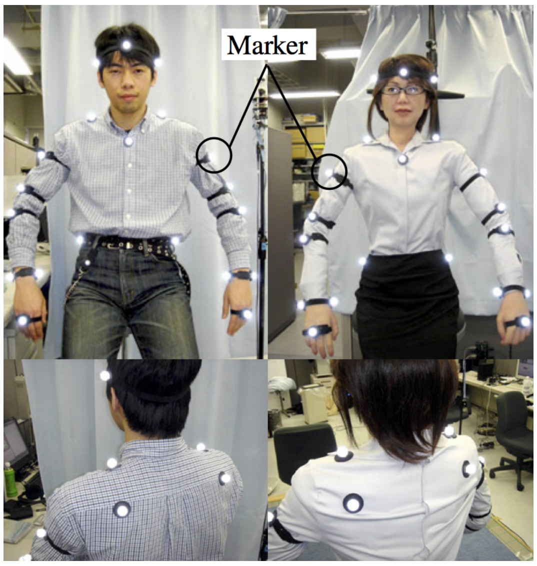


Figure 2.3: Teacher (left) and Android Repliee Q2 (right) with markers [27]

game or controls a virtual character. Motion capturing of actors to create animated characters for a movie or game are also part of this category. Analysis applications contain automatic diagnostics of orthopaedic patients or analysis of athletes' movements. The car industry is also a driving factor for analysis applications such as pedestrian detection or driver fatigue detection. Due to this widespread application area, there are some highly sophisticated motion capture systems commercially available.

Optical Systems In optical systems, cameras are used to track human motion. Professional systems that track motion accurately for animating characters in movies normally use markers attached to the actor and several cameras for triangulation. These markers are either active (illuminated by LEDs) or passive (retroreflective) and can be seen in Figure 2.3. Pollard et al. [29] use such a commercially available marker-based motion capture system from Vicon to map a human dancing motion of the upper body to a Sarcos humanoid robot shown in Figure 2.4. The focus is on adapting the human motion to the limited range of joint motion and velocity of the robot. The resulting trajectory should be as similar to the original movement as possible. The same dance was performed by different actors and the various dance styles should be distinguishable in the robot motion sequences. The motion capturing system setup consists of eight cameras and 35 markers. The raw marker data is mapped to the joint angles of a skeleton by the provided software. These angles are scaled down by

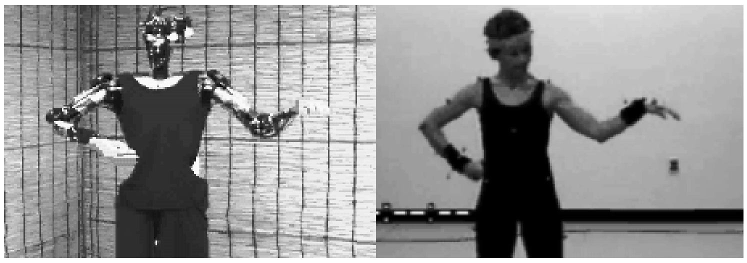


Figure 2.4: Sarcos humanoid robot and actor with motion capture markers performing “little teapot” dance [29]

a scaling factor to be in the range of the robot’s joints. The joint velocity is bound to an upper limit. This way, the motion is scaled down and therefore, the position of the robot’s hand won’t necessarily match that of the actor’s as closely as possible. The results are satisfying for many poses. It falls short for some cases when the actor exceeds the limits of a degree of freedom of a joint but another degree of freedom is in bound, because each degree of freedom is scaled independently. So, if the actor is leaning forward and to the side, it might result in the robot mostly leaning forward. Another drawback is the expensive motion capture system as well as the lack of real-time processing which would be required for teleoperation. However, the latter might be solved with modern hardware.
Marker-based motion capture systems are very expensive and inconvenient. They need to be set up in a fixed lab environment. Therefore, there is a big interest in affordable and easy to set up markerless motion capture systems. Do et al. [30] present a system for human motion imitation with markerless and marker-based motion capture systems. They use the humanoid robot ARMAR-IIIb for experimental testing (Figure 2.5). The markerless system has a virtual human model and compares the filtered input from a stereo camera system with this model involving a particle filter. This approach recognises only 4 degrees of freedom for each arm. Details on this algorithm are given in [31]. The acquired motions from both systems are translated to the unifying master motor map, which consists of 58-dimensional vectors, each vector describing a joint angle configuration of a human model. The joint angles of the robot are determined by finding the optimal solution of a similarity function which considers the robot angles and the position of the Tool Center Point (TCP). The Levenberg-Marquardt method is used for solving the optimisation problem. This is done because of different kinematic structures of a human and the robot, a one-to-one mapping of joint angles would often result in unsatisfactory reproduction of motion. Experimental results show that this approach is a

good compromise between correct joint angles and a correct position of the end effector.

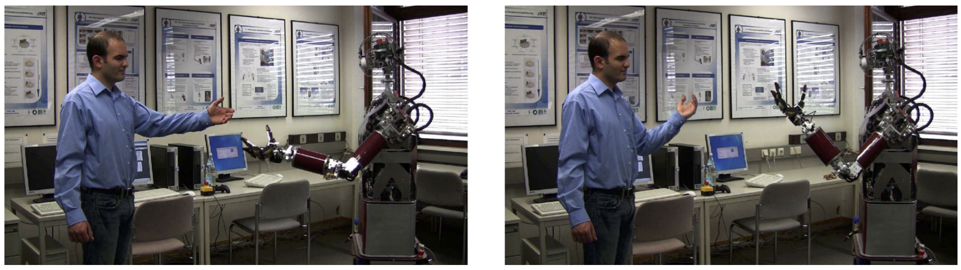


Figure 2.5: Online Imitation by humanoid ARMAR-IIIb using a markerless vision system for motion capturing by Do et al. [30]

The data acquired by markerless motion capture systems that are based on RGB images is often too imprecise. In the approach above Do only used 4 degrees of freedom. In 2010 Microsoft released the Kinect camera to be used for the Xbox 360 gaming system as a controller alternative. It uses an infrared projector which emits an infrared pattern and an infrared camera to acquire a depth image. The obtained data is used to map the movements of the user to a virtual skeleton. The joint angles are calculated in real time. For more details see [32]. The Kinect is well suited for teleoperation applications because it is affordable and provides the human joint angles. Suay and Chernova [33] presented one of the first implementations for the Nao humanoid robot in 2011. Zuher and Romero [34] use the Kinect to recognise human motion and control a Nao robot in real-time (Figure 2.6). They use different techniques to determine the human joint angles, namely trigonometry, the rule of three and the method provided by the used SDK (Software Development Kit). An imitation mode and a gesture mode are implemented. In imitation mode, the robot copies the head, hand and leg motions of the user. In gesture mode, different robot behaviours are initiated with gestures. For example, the robot starts walking if the arm is pointed forward. The system was tested with 10 users who reported satisfactory teleoperation capabilities. However, leg movements were disabled during user testing because they are too unstable. The robot would fall over. Additionally, no collision detection for the legs is performed. The wrist is not used and therefore only 4 degrees of freedom for each arm are controlled.
Ou et al. [35] imitate human motion with the Nao robot using the Kinect Camera as well. The focus is on whole-body imitation with proper balancing (Figure 2.7). The human motion is captured by the Kinect camera and the corresponding joint angles are calculated by minimising a similarity error function with the Levenberg-Marquardt algorithm. The Kinect does not

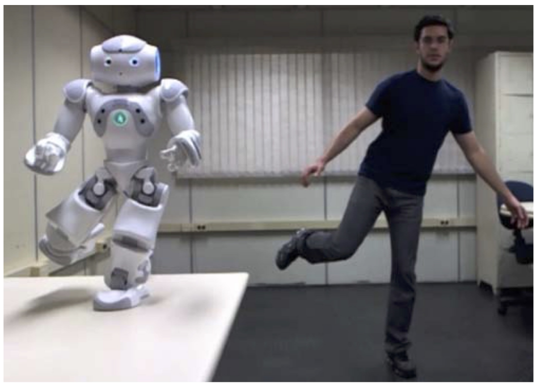


Figure 2.6: Imitation with Nao Robot and Kinect Camera by Zuher and Romero [34]

provide satisfactory information of the user's ankle. The robot's ankle joint is controlled in order to maintain balance. A stable state is achieved if the projection of the centre of mass lies within the supporting foot in the single-foot supporting phase or on the line between the feet in the double-foot supporting phase. The required angles of the ankle and hip joints are calculated by solving an optimisation problem with the Levenberg-Marquardt method. If a collision of links would occur, the corresponding motors stop. The experiments show promising results. The robot is able to maintain balance most of the time while the motion still is similar to the human's. However, the speed of the legs is reduced.

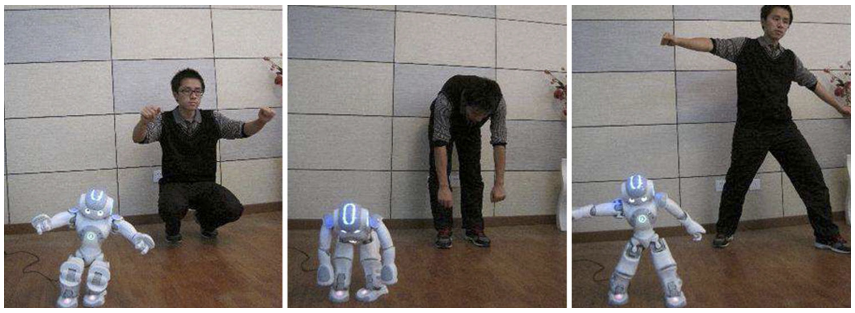


Figure 2.7: Whole-body imitation with Nao Robot and Kinect camera by Ou et al. [35]

The Kinect camera is an affordable and easy to set up system for capturing human motion. The accuracy of the Kinect is limited. The random error of depth measurement increases quadratically with increasing distance from the

sensor from a few millimetres to 4 cm at the maximum range of 5 m. The low resolution of 640 × 480 for the depth image also limits the accuracy especially for objects farther away [36]. The second version of Kinect is improved in these domains [37]. However, the standard SDK still does not provide the possibility to track each finger[4].
In 2012 the Leap Motion sensor was introduced which specialises in hand tracking. It enables tracking of the singular bones in the fingers with sub-millimetre accuracy. For more details on the Leap Motion refer to Section 3.1. Bassily et al. [38] use the Leap Motion to control the Jaco robotic arm. The aim is to provide an easy interface for elderly and disabled people to control the robotic arm to help them with simple tasks. The Leap Motion is positioned on a table, and the user moves their hand above it as shown in Figure 2.8. If the hand is moved in a direction, the end-effector of the robotic arm will move in this direction as well. Angular information (roll, pitch, yaw) and grasping motion are also mapped to the robotic arm. To neglect involuntary oscillating movements caused by diseases such as Parkinson's Disease, a threshold value for each motion type is set in an initialising step. The Leap Motion is used as an interface to control a robotic arm, but only the palm's position and orientation are considered. The posture of the rest of the arm is not imitated.

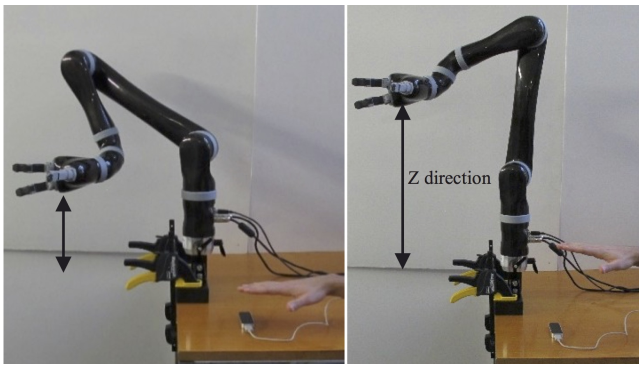


Figure 2.8: Jaco robotic arm controlled with Leap Motion sensor [38]

Yu et al. [39] propose a method to control a Nao humanoid robot with the Leap Motion controller (Figure 2.9). The Leap Motion is again positioned on a table with both hands moving above it. Gestures of the right hand are recognised by the Leap Motion SDK and used to control leg movements. For example, if a swipe is recognised, the robot walks forward, and if a circle gesture is registered, the robot turns around. The user's left hand movements are mapped

[4]https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

to the robot's left hand. The Cartesian position and the angular motion are considered, comparable to [38].

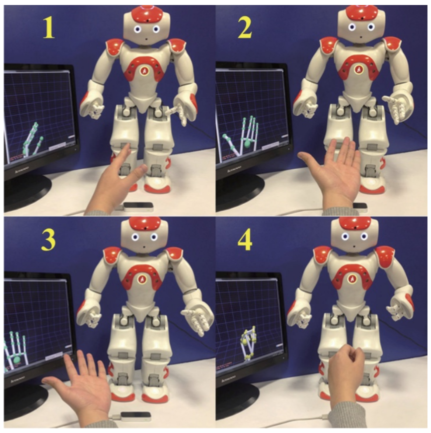


Figure 2.9: Nao robot controlled with Leap Motion sensor by Yu et al. [39]

Non-Optical Systems Motion capture can also be done without cameras. Usually, these non-optical systems distribute several sensors on the user's body. Sensor fusion is used to combine the data from different sensors such as 3D gyroscopes, accelerometers and magnetometer to compensate drift. In the NASA's Robonaut teleoperation mentioned above, such a system is used. A commercially available motion-tracking suit is the Xsens MVN suit [40]. Koenemann, Burget and Bennewitz [41] use this suit to imitate whole-body motions with the Nao robot in real-time (Figure 2.10). Only the position of the end-effectors (hands and feet) and the position of the centre of mass are considered. The orientation of end-effectors is neglected. The joint angles are calculated by a numerical inverse kinematics solver based on damped least squares method. The actor's body proportions are measured while performing a "T-pose" in an initialisation step. For posture stabilisation, only the robot's leg chain is influenced to keep similarity to the human motion. Similar to Ou's approach, the centre of mass is manipulated. Its projection is kept near the line connecting the feet or inside one foot in single-foot support mode. The results show a high similarity between robot and human motions, while maintaining the robot's balance. Stanton, Bogdanovych and Ratanasena [42] use the Xsens MVN motion capture suit as well to teleoperate Nao. A neural network is trained in the beginning by analysing a training sequence in which the human actor imitates movements of the robot. Afterwards, a real-time teleoperation is possible. No balance controller is used. The ankle joints are adapted to the hip and knee joints in order to keep the feet parallel with the ground. The downside of the motion-capture suit is its high prize.

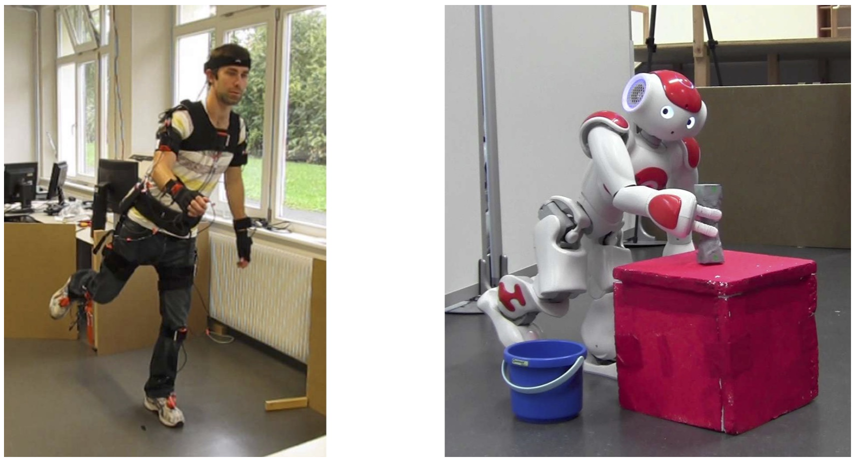


Figure 2.10: Whole-body imitation with Nao Robot and Xsens MVN motion-capture suit by Koenemann, Burget and Bennewitz [41]

2.2.3 Applications Using Teleoperation

Nakaoka et al. [43] adapted and extended Pollard's algorithm [29] for mapping dancing motion to generate whole body motions for the HRP-1S humanoid robot (Figure 2.11). The focus is on ensuring stability when moving the feet. Motions are acquired by a marker based motion capturing system with eight infrared cameras. For the particular Japanese dance three leg positions are dominant and correspondent primitives are defined (stand, squat, step). To ensure stability, trajectories of the "zero moment point"[5] are created between these primitives. In the simulation, the whole dance sequence can be recreated by the robot. The real robot is only able to imitate the hand motions with a reduced speed of motion by the factor 2.5 while standing.
Shon, Storz and Rao [44] use a motion capture system to imitate human arm movements in real-time. Their approach focuses on learning the shown motion from demonstration. The inverse kinematic problem is solved by a commercially available motion-capture system. They use Forward Model Learning with Hidden Markov Models to accomplish a goal. The demonstrator

[5]The zero moment point is the point at which the moments introduced by gravity and inertia cancel each other out. This point should be in the area of the sole for a stable state.

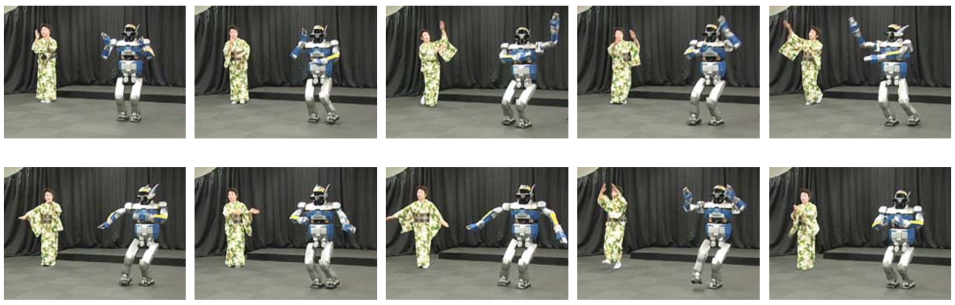


Figure 2.11: HRP-1S performs a Japanese dance which was recorded beforehand with a marker-based motion capturing system [43]

shows the robot how to lift a box with one hand or two hands. Afterwards, the robot performs this task 40 times and learns, that it is not capable of lifting the box with one hand and therefore will only use two hands consequently (Figure 2.12).

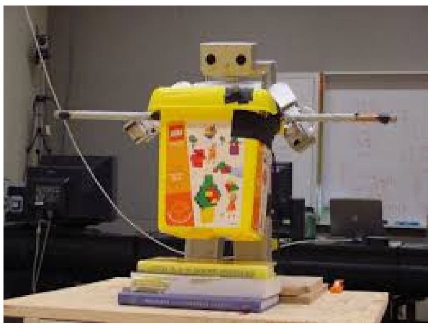


Figure 2.12: HOAP-2 lifting a box after learning the movement from a human demonstrator [44]

Fritsche et al. [45] present a first-person teleoperation interface for the iCub humanoid robot. The human motions are captured by the Kinect v2. A sensor glove is used to track the individual fingers and provide haptic feedback to the user. The robot's camera streams are shown in a head mounted display (Oculus Rift DK2). The setup is shown in Figure 2.13. Four degrees of freedom are controlled in each arm (pitch, roll, yaw of the shoulder and yaw of the elbow). The wrist joint is neglected. The torso has 3 degrees of freedom. The arm and torso motions are tracked by the Kinect camera and the required joint angles are calculated from the tracked positions. The operator's head movements are tracked by the sensors in the Oculus Rift and mapped to 3 degrees of freedom of the neck. If the operator would exceed the limits of the neck joint angles, the eyes are moved to increase the field of view. Each hand has 7 degrees of freedom. The user's fingers are tracked by the SensorGlove which also provides haptic feedback via vibration. The intensity of the vibration is determined by tactile sensors on iCub's fingertips. After adapting to the dimensions of the robot and to the delay introduced by filtering and processing sensor data, a test user was able to pick up an object with the robot's hand and place it at a different location. No stereo vision is used in this approach, so the depth perception is limited. Furthermore, the delay between the operator's movements obtained by the Kinect and the robot's movements is about 800 ms.

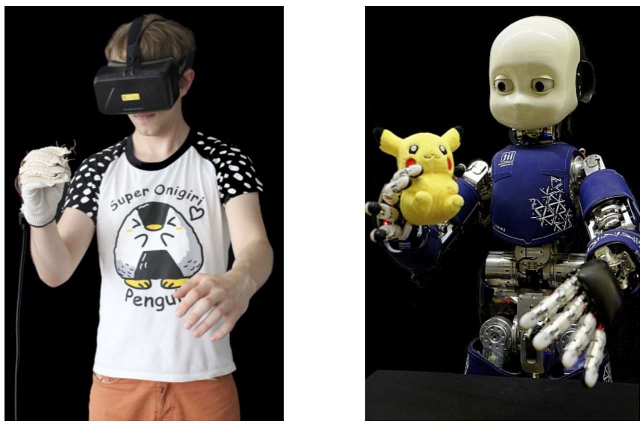


Figure 2.13: The user wears an Oculus Rift virtual-reality headset and a sensor glove to teleoperate the iCub robot; the user's movements are tracked by the Kinect camera [45]

Tomić et al. [46] tested different inverse kinematics algorithms with the Romeo humanoid robot for human imitation. A marker-based motion capture system is used to track human movements. Virtual markers are placed on the robot. To imitate human motion, the distance between the user's and the robot's markers is minimised by solving an optimisation problem. Usually, there is an infinite number of solutions which satisfy this condition because of the high number of degrees of freedom of humanoid robots. Therefore, Tomić et al. introduce an additional criterion to choose one solution. Different criteria (for example minimising joint velocities or minimising kinetic energy) are tested and compared for best human imitation. Every algorithm is able to imitate human motion. The experiments were performed on a simulation of the Romeo robot. No information for real-time capabilities of these algorithms is given, making them unsuitable for teleoperation.

3 Concept

In this chapter, the hardware and software components used for the intuitive teleoperation application are explained. The user wears the Oculus Rift DK2 virtual-reality headset with the Leap Motion hand tracking sensor attached for a first-person experience shown in Figure 3.1. The Leap Motion controller is used to track the movement of the hands. The obtained joint positions are sent to a computer which calculates the corresponding joint angles. These are sent via a network connection to a humanoid robot (e. g. Romeo, Pepper), which mirrors the movements of the user. The head movements of the user are tracked with sensors in the Oculus Rift virtual-reality headset. The camera images captured by the robot are sent back to the computer and displayed on the Oculus Rift to provide feedback to the user.

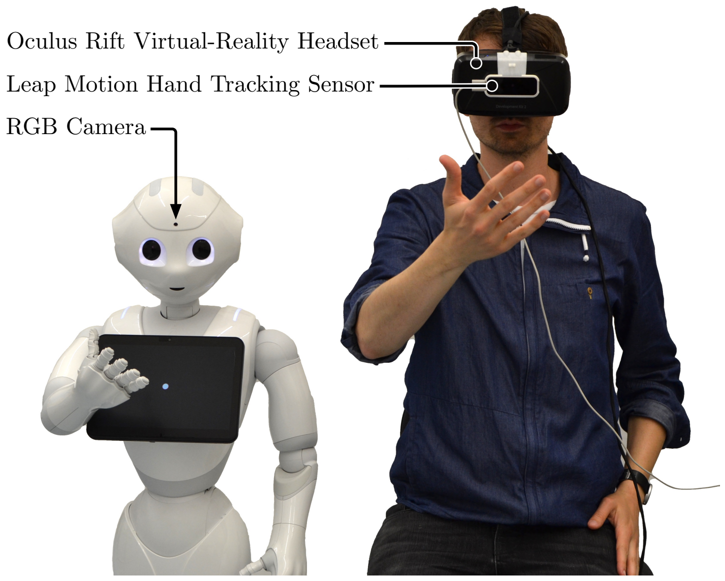

Oculus Rift Virtual-Reality Headset
Leap Motion Hand Tracking Sensor
RGB Camera

Figure 3.1: Hardware setup: the user wears the Oculus Rift virtual-reality headset with the Leap Motion sensor attached; the Pepper robot imitates the user's movements and provides the camera images displayed in the Oculus Rift

3.1 Leap Motion

The Leap Motion[1] controller is a consumer-grade hand tracking device. The device was initially developed to be placed on a desk in front of a screen and to be used as an input device. The latest software releases focus on virtual reality. In this setup, the Leap Motion is mounted on a VR headset, which is the way it is used in the teleoperation system.

The basic setup consists of two infrared cameras and three infrared LEDs displayed in Figure 3.2. The scene is illuminated by the LEDs which emit infrared light with a wavelength of 850 nm only when a picture is taken[2]. This narrow band and the pulsed light ensure that noise from other infrared sources is negligible, such as remote controls or VR headsets. According to the manufacturer, the sensor's range is 25 mm to 600 mm above the device with sub-millimetre accuracy. The range is limited by the infrared light propagation. For hands farther away, a higher intensity is needed which is limited by the power that can be drawn over the USB connection. The short light pulses improve the available intensity. The images of both cameras are synchronised and saved in the USB controller's memory. The hardware itself only makes necessary resolution adjustments and sends the stereo image via USB connection to the computer[3]. The software removes background objects and reconstructs a 3D representation. The data is compared with an internal model of a human hand to calculate the position of the real hand. Filtering techniques are applied to ensure temporal coherence and overcome problems resulting from occlusions. The exact functionality of the algorithm is not disclosed, due to patent and trade secret restrictions.

Different studies have been conducted to measure the precision. Guna et al. [47] found the standard deviation in a static scenario to be less than 0.5 mm at all times for a maximum distance of 250 mm. When moving to a discrete position on a path the standard deviation was below 0.7 mm per axis [48]. The standard deviation corresponds to the variability of the tracked position. In these studies, tools were tracked that resemble fingertips which were positioned in parallel to the sensor. In non-optimal settings, e. g. the fingers are not spread out, the tracking performance and the robustness substantially decrease. These studies were conducted with an old version of the SDK. In February 2016 Orion was introduced, the newest version of the SDK[4]. This release focused on VR applications and greatly improved tracking and latency. The range was also

[1]`https://www.leapmotion.com`

[2]`http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/`

[3]`http://blog.leapmotion.com/understanding-latency-part-1/`

[4]`http://blog.leapmotion.com/orion/`

significantly extended up to 800 mm[5]. To the author's knowledge, there have not been any new studies published yet, analysing the improved capabilities. However, a clear difference is observable, especially concerning the improved sensory space, latency and robustness.

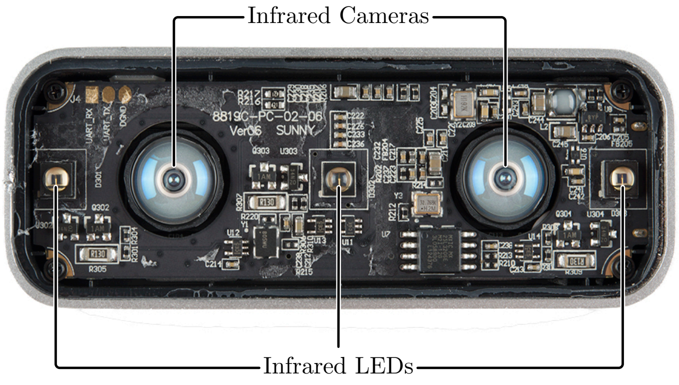


Figure 3.2: Opened Leap Motion controller exposes two infrared cameras and three infrared LEDs[6]

The obvious alternative to the Leap Motion sensor is the Microsoft Kinect camera with its provided skeletal tracking capabilities. The current version tracks 25 skeletal joints of the whole body when the user is standing. This would simplify the calculation of the shoulder joint angles in comparison with the method introduced in this thesis because the Leap Motion does not provide the position of the shoulder joint. However, the precision of the tracked joints by the Kinect skeletal tracker is inferior to the Leap Motion. According to Wang et al. [49] the mean joint position offset for most of the joints for a user in motion is 50 mm to 100 mm and the standard deviation is 10 mm to 50 mm. The Leap Motion has a standard deviation of the distance for movement in the magnitude of ≈1 mm as mentioned above in comparison. To be fair, these values were measured for fingertips. The positions of some other joints (e. g. elbow) have a visible offset. The Leap Motions tracks the human hand in great detail down to the position and direction of the single finger bones. The Kinect only tracks the hands, the thumbs and the tips of the hands. The robots on

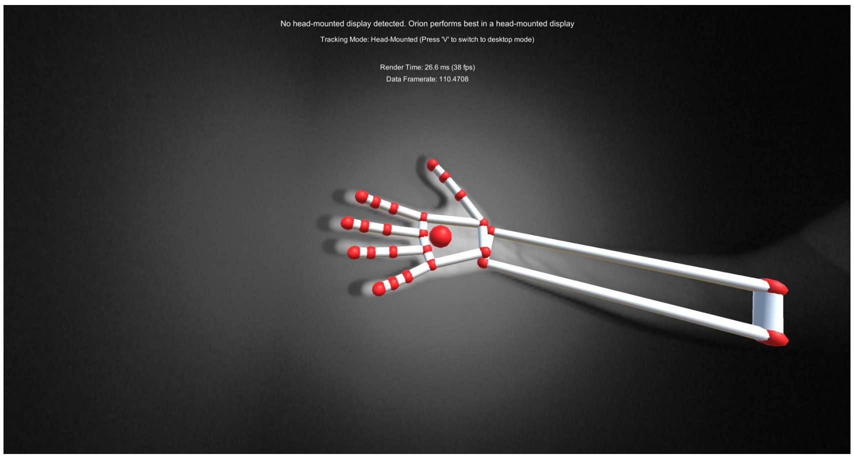
[5]`http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/`

[6]`https://learn.sparkfun.com/tutorials/leap-motion-teardown`

which we tested the teleoperation application are only able to open and close their hands. To control these simple end-effectors, the joints provided by the Kinect might be sufficient. For more complex hands with independent fingers, the Leap Motion sensor data is needed. Another advantage is the portability of the Leap Motion. It is positioned on the Oculus Rift, so no additional camera has to be set up opposite of the user.

3.1.1 Leap SDK and Orion

Leap Motion provides an SDK with two basic libraries, one written in C++ and the other in C. Language bindings for these libraries are also available for C#, Objective-C, Java and Python. A WebSocket server and a client-side JavaScript library are available for developing JavaScript and web applications. Leap motion data can be used in the game engines Unity and Unreal with plugins[7].
The provided software runs on the client computer as a service on Windows or as a daemon on Unix-based systems. It connects to the controller device over USB and provides two APIs (application programming interface) to access the sensor data: a native interface and a WebSocket interface. An application connects to the native interface through a dynamically loaded library. The WebSocket server sends tracking data in the form of JSON messages[8].



```
No head-mounted display detected. Orion performs best in a head-mounted display
Tracking Mode: Head-Mounted (Press 'V' to switch to desktop mode)

Render Time: 26.6 ms (38 fps)
Data Framerate: 110.4708
```

Figure 3.3: Visualisation of a tracked arm by the Leap Motion sensor

[7]`https://developer.leapmotion.com/documentation/cpp/devguide/Leap_SDK_Overview.html`
[8]`https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Architecture.html`

To establish a connection to the Leap Motion device, a controller object has to be created. This controller provides snapshots of the sensory data, so-called frames. There are two possibilities of obtaining this data, either event-based or by polling. The former requires a listener object defined by the Leap SDK. The controller calls the relevant listener callback function when an event occurs, e. g. on each new frame. These functions are called from a thread created by the Leap Motion library. The other possibility is to call the controller's `frame()` function when the application needs the data. This is the simpler option and is easy to use when the application already has a loop for example for rendering. It is possible to get the same frame several times if the frame rate of the application exceeds the frame rate of the Leap Motion. This effect is avoided when using callback functions.
A class is defined for each tracked physical entity, like hands, fingers and tools. The arm object describes the forearm and includes the position of the elbow, of the wrist and the direction of the arm. The other objects contain similar information down to the individual bones of the fingers. In Figure 3.3 a visualisation of the tracked human arm is shown.

3.2 Oculus Rift

In order to control a robot in a teleoperation setting, the user has to see the surroundings of the robot. A virtual reality headset has many advantages over a conventional monitor. The head of the robot and therefore the orientation of the cameras can be controlled by the motion sensors in the headset. This is a very intuitive way of manipulating the field of view. Two cameras can be used to create a stereoscopic 3D image, which makes it easier to complete certain tasks like grabbing. The whole setup creates a sense of immersion for the user.
The Oculus Rift is a head-mounted display (HMD) and was first launched on the crowd-funding website Kickstarter in 2012[9]. The Development Kit 1 started shipping on 29 March 2013 with a resolution of 1280x800 (640x800 per eye). The updated Development Kit 2 was released in July 2014. The resolution was increased to 1920x1080 (960x1080 per eye), the display technology changed from LCD to OLED and a camera was included to track the user's translational motion. The Consumer Version started shipping on 25 March 2016. It uses two panels, one for each eye, instead of one like the previous versions. These panels have a resolution of 1080x1200 and a refresh rate of 90 Hz.

[9]`https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game/`

3.2.1 Oculus Rift Development Kit 2

For our implementation, we use the Oculus Rift Development Kit 2 (DK2) pictured in Figure 3.4. The improved Consumer Version was not available when the project started, and the DK2 still has some advantages. Oculus stopped supporting Linux operating systems with version 0.6 of their SDK, which is required for the new headset. Furthermore, due to the lower screen resolution, the DK2 does not need as much computation power.

Figure 3.4: Oculus Rift Development Kit 2

The screen used in the DK2 is the same panel that is built in the Samsung Galaxy Note 3 phone. Its diameter is 140 mm with a resolution of 1920x1080. In front of the display are two lenses, which increase the perceived field of view because the light reaches the eyes from the sides as well as from straight ahead. Furthermore, the lenses[10] create collimated light which means the rays have been aligned in parallel. This way, the display seems to be infinitely far away instead of approximately 4 cm in front of the eyes, so the user's eyes are able to focus on the image. The lenses introduce distortions and chromatic aberration, which have to be accounted for when rendering [50].
For an immersive VR experience, the user's head orientation and position have to be tracked fast and precisely. The DK2 has a gyroscope, accelerometer[11] and a magnetometer built in. The sensors provide angular velocity, linear acceleration and magnetic field strength at a refresh rate of 1000 Hz[12]. The

[10]`http://vrtifacts.com/oculus-dk2-lens-characteristics/`
[11]`https://www.invensense.com/products/motion-tracking/6-axis/mpu-6500/`
[12]The magnetometer readings only change at a rate of 220 Hz

samples are sent to the host at a rate of 500 Hz[13]. Each package contains two samples. The user's head orientation is calculated by sensor fusion of the different inputs to minimise the drift. Predictive filtering is applied to reduce latency [51]. However, the head position cannot be calculated with these sensor readings accurately. An offset in acceleration results in a quadratic error growth of the position. The DK2 comes with a camera that has to be placed in front of the user and tracks an array of infrared LEDs in the faceplate of the headset to estimate the head position [52].

3.2.2 Virtual Reality Considerations

When designing an application for virtual reality, many things have to be taken into account, that can be neglected when developing for traditional systems. Many users experience symptoms of cybersickness when using VR headsets comparable to motion sickness. These symptoms include nausea, disorientation, eye strain and headaches. The cause of cybersickness is still not completely understood. Different theories exist to explain the phenomenon, with sensory conflict theory being the most prominent. It states that conflicts between visual and vestibular systems are the main cause, so basically what you see differs from what you feel [53], [54].
A key factor for minimising the user's discomfort is to keep the latency very low. Ideally, a motion-to-photon latency of less than 20 ms is recommended. This is the total time between the user's head movement and an updated image being displayed on the screen [55]. The application's frame rate should also be at least at the display's frame rate of the DK2 of 75 Hz to avoid judder. This high frame rate is essential in VR settings, more modern headsets have increased it to 90 Hz.
The described factors lead to a few consequences in a teleoperation setting. It is not advisable to take the two camera images of the robot and display them to the left and right eye of the user. The distance of the left and right camera of the robot probably differs from the user's interpupillary distance (IPD). Therefore, the images probably would not result in one stereoscopic image. More importantly, there will be a significant latency between the head movements of the robot and of the user caused by computation time and limited motor velocity. In the worst case, the user tries to look somewhere, where the robot cannot reach because of kinematic restrictions. Additionally, the used robots' cameras support a maximum frame rate of 30 Hz with a low resolution as stated in Section 3.3. To overcome these problems, a virtual environment is created that reacts to movements of the user almost instantly. The robot's

[13]`http://doc-ok.org/?p=1405`

video feed is displayed on a floating canvas which position depends on the robot's head pose. This way, the user's discomfort is kept to a minimum while providing the robot's camera readings with a feedback in which direction the robot is looking. The detailed implementation is described in Section 4.2.2.

3.2.3 Oculus SDK 0.5

The Oculus Rift is still a very young product. Therefore, a lot of changes occur between different versions of the SDK including major paradigm shifts. In May 2015 Oculus VR announced to drop support for Linux and macOS[14]. The so-called extended mode which made it possible to use the Oculus Rift as an external monitor was removed as well, in favour of direct rendering to the Oculus Rift which is more efficient. We wanted to implement the teleoperation on Windows as well as on Ubuntu which forced us to stay at version 0.5 of the SDK. This does not have any major restrictions besides that we are not able to use the newest hardware as well as some optimised functions.

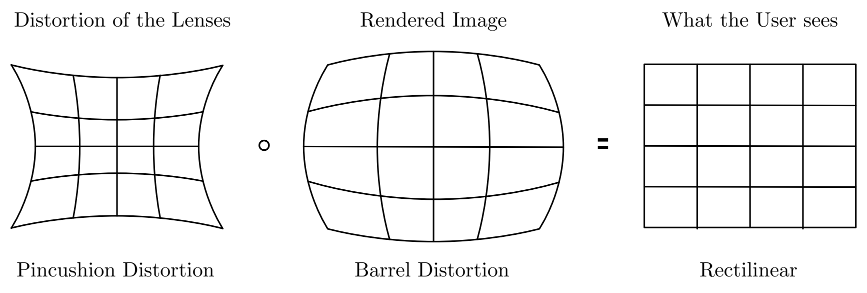


Figure 3.5: The rendered barrel distortion cancels out the pincushion distortion introduced by the lenses of the Oculus Rift

In the application, an HMD object has to be created to establish a connection to the Oculus Rift. The API provides functions to get the current pose of the user's head. To create a stereoscopic image the virtual scene has to be rendered twice from different camera positions for the two eyes. The SDK provides information for the distance of the cameras and the field of view. As mentioned above, the lenses magnify the image to enhance the field of view. This introduces distortions that have to be counteracted by post-processing the rendered view. The pincushion distortion is cancelled out by applying the inverse barrel distortion to the image as seen in Figure 3.5. Chromatic aberration is an effect of lenses to focus light with different wavelengths to

[14]`https://www.oculus.com/en-us/blog/powering-the-rift/`

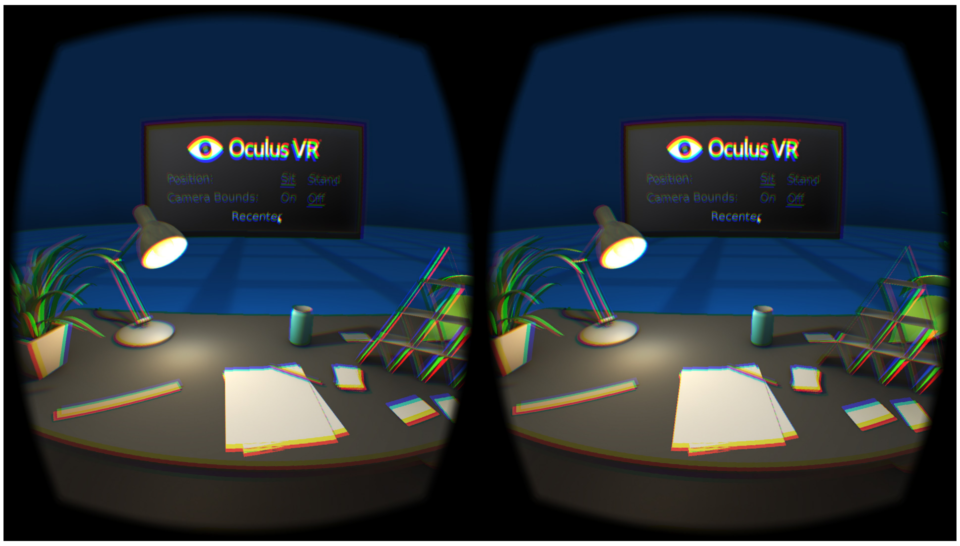


Figure 3.6: A scene rendered for the Oculus Rift with barrel distortion and colour shift applied to cancel out effects of the lenses

different convergence points. In the Oculus Rift, it is responsible for colour fringes around edges near the lens periphery. It can be corrected by shifting the different colour channels of the image in the opposing direction of the chromatic aberration. The Oculus SDK takes care of these corrections. The developer just needs to provide the two textures for the left and the right eye. In Figure 3.6 a rendered scene for the Oculus Rift is shown. The barrel distortion and the colour shift are applied to each image.

3.3 Humanoid Robots by SoftBank Robotics

Romeo is a humanoid robot by the corporation SoftBank Robotics (formerly Aldebaran Robotics) based in France. It's the successor of the 58 cm tall Nao robot which was released in 2008 to the public. Nao is mainly used as a research platform and is the official robot for the RoboCup soccer tournament[15]. Pepper is the newest robot of SoftBank focused on perceiving human emotions and reacting accordingly to these emotions[16]. All of them are shown in Figure 3.7.

[15]https://www.ald.softbankrobotics.com/en/cool-robots/nao
[16]https://www.ald.softbankrobotics.com/en/cool-robots/pepper

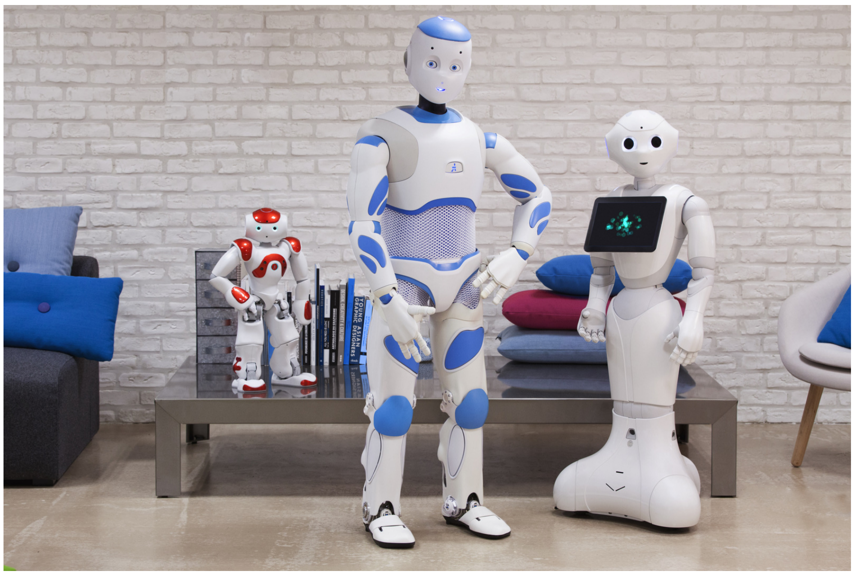


Figure 3.7: Humanoid robots by SoftBank Robotics (from left) Nao, Romeo and Pepper[17]

3.3.1 Project Romeo

The Romeo project started in 2009 with the goal to create a humanoid robot to assist people with loss of autonomy, such as elderly or disabled persons[18]. The first prototype that is in use by different laboratories in Europe was introduced in 2014. It is approximately 140 cm tall and has 37 degrees of freedom. It is controlled by 4 CPU boards based on the Intel Atom (Z500 series) processor[19], which are assigned to different domains like audio, video and artificial intelligence respectively [56]. The robot has a total of four 2D cameras (ON Semiconductor MT9M114[20]), two in the forehead and one in each eye. These cameras provide a maximum frame rate of 30 Hz which is far below the recommended frame rate for virtual reality applications of 75 Hz. Optionally it is also equipped with a 3D camera (ASUS Xtion[21]). Romeo has four microphones and two loudspeakers on his head. You can connect to it via Ethernet or WiFi.
In our teleoperation application, only the head and arms are used. Therefore, the further explanations will focus on these parts. The kinematic chain of Romeo

[17]https://www.ald.softbankrobotics.com/en/press/gallery/romeo
[18]https://www.ald.softbankrobotics.com/en/cool-robots/romeo
[19]http://doc.aldebaran.com/2-1/family/romeo/motherboard_romeo.html
[20]http://www.onsemi.com/pub_link/Collateral/MT9M114-D.PDF
[21]http://doc.aldebaran.com/2-1/family/romeo/video_3D_romeo.html

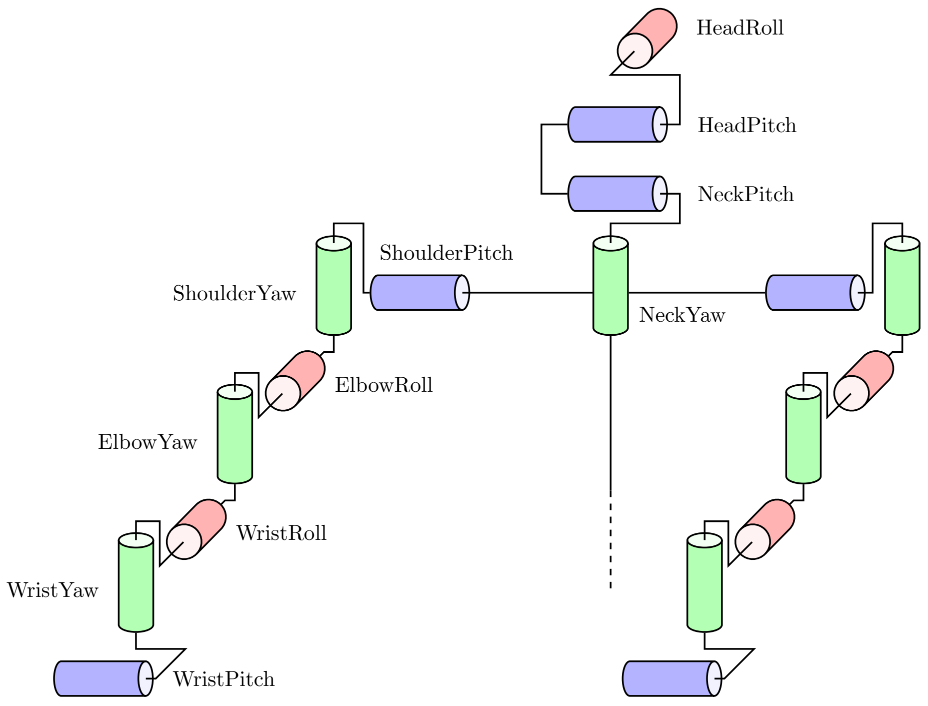


Figure 3.8: Joints of Romeo's upper body

is illustrated in Figure 3.8. Each arm has 7 degrees of freedom comparable to the human arm. The joints are listed in Table 3.1 with their corresponding range. For comparison, the equivalent human joints that cause the same movement are shown. Overall, the range is quite similar. Romeo exceeds in roll motions but falls behind in most of the other motions. In Table 3.2 the different joints of the head and neck are shown. Romeo has two separate joints to change the pitch.

3.3.2 Pepper Robot

Pepper was unveiled to the public in June 2014. It is a humanoid robot. It is commercially available and is used in SoftBank's own stores as well as in stores for overpriced coffee in capsules. It is approximately 120 cm tall and has 20 degrees of freedom. In contrast to Romeo and Nao, Pepper is a wheeled

Joint Name	Range of Romeo	Range of human [57]
ShoulderPitch	$-82.7°$ to $127.2°$	$-90°$ to $135°$
ShoulderYaw	$-24.7°$ to $65.3°$	$-50°$ to $120°$
ElbowRoll[22]	$-120°$ to $120°$	$-80°$ to $110°$
ElbowYaw	$-90°$ to $0°$	$-145°$ to $0°$
WristRoll	$-210°$ to $30°$	$-180°$ to $5°$
WristYaw	$-25°$ to $25°$	$-15°$ to $30°$
WristPitch	$-56°$ to $56°$	$-85°$ to $85°$

Table 3.1: Comparison of the range of arm joints between Romeo and a human

Joint Name	Range of Romeo	Range of human [58]
NeckYaw	$-90°$ to $90°$	$-70°$ to $70°$
NeckPitch	$-20°$ to $40°$	$-40°$ to $55°$
HeadPitch	$-20°$ to $16°$	-
HeadRoll	$-20°$ to $20°$	$-35°$ to $35°$

Table 3.2: Comparison of the range of head joints between Romeo and a human

robot with 3 multi-directional wheels and a maximum speed of 3 km/h[23]. It is equipped with two 2D cameras (OmniVision OV5640[24]) and a 3D depth camera (ASUS Xtion[25]). Pepper uses an Intel Atom E3845 quad-core processor with 4 GB of RAM. Four microphones on its head enable Pepper to locate the direction of sound. For sound output, two speakers are located in its ears. Pepper connects to a network via WiFi. For a faster connection, an Ethernet port is available. It is powered by a 795 W h battery which lasts for approximately 12 hours according to the manufacturer. In contrast, Romeo does not have any battery and needs to stay connected to a power source. For obstacle avoidance, Pepper is equipped with laser and sonar sensors. A gyroscope and an accelerometer provide sensor information for the robot to balance itself. A tablet is fixed to Pepper's chest as an input method, for

[22]Romeo's ElbowRoll results in a rotation of the fore-arm. The origin of the same movement for humans is in the shoulder, so the joint would be called ShoulderRoll. To compare the range they are listed in the same line.

[23]`https://www.ald.softbankrobotics.com/en/cool-robots/pepper/find-out-more-about-pepper`

[24]`http://doc.aldebaran.com/2-5/family/pepper_technical/video_pep.html`

[25]`http://doc.aldebaran.com/2-5/family/pepper_technical/video_3D_pep.html`

visualising information or transporting emotions.

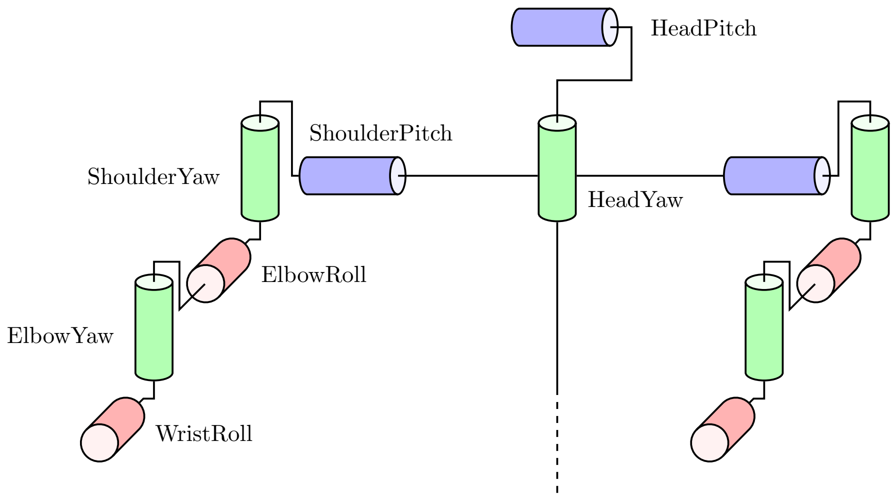


Figure 3.9: Joints of Pepper

The discussion will focus on Pepper's upper body and how it compares to Romeo's upper body. The joints are shown in Figure 3.9. Each arm has five degrees of freedom, not counting the opening and closing of the hand. The head is moved by two different joints. In comparison to Romeo, Pepper lacks two arm joints and two head joints. As a result, Pepper is not able to tilt its head and perform a pitch or yaw movements of the hand. In Table 3.3 and Table 3.4 the range of the joints is shown. The human joints have a similar range in comparison, with some major differences. ShoulderYaw cannot decrease below 0.5° because it would collide with the tablet. WristRoll range differs vastly from the human equivalent probably to compensate for the rotation of the wrist humans can induce in the shoulder joint. These adjustments make the pouring motion possible for the robot and increase the human imitation performance as shown in Chapter 5. The realisable angles for HeadPitch and ElbowRoll are for various positions of the head and arm smaller than stated in the table, to avoid collisions with the tablet.
For teleoperation, the operational space of Pepper is more restricted than Romeo's operational space. On the positive side, complexity is reduced due to fewer joints. Pepper's 2D cameras are positioned in its forehead and in its mouth. Romeo has two stereo pairs of 2D cameras which can be used to improve the horizontal field of view by displaying the left camera image on the left side of the virtual reality headset and the right image on the right side.

This is not possible for Pepper, and the horizontal field of view is therefore only 57.2°. Due to Pepper's multidirectional wheels, a simple keyboard control for locomotion can be implemented. Currently, a comparable solution for Romeo is not possible, because there are no methods for walking available in the SDK.

Joint Name	Range of Pepper	Range of human [57]
ShoulderPitch	−119.5° to 119.5°	−90° to 135°
ShoulderYaw	0.5° to 89.5°	−50° to 120°
ElbowRoll	−119.5° to 119.5°	−80° to 110°
ElbowYaw	−89.5° to −0.5°	−145° to 0°
WristRoll	−104.5° to 104.5°	−180° to 5°
WristYaw	-	−15° to 30°
WristPitch	-	−85° to 85°

Table 3.3: Comparison of the range of arm joints between Pepper and a human

Joint Name	Range of Pepper	Range of human [58]
HeadYaw	−119.5° to 119.5°	−70° to 70°
HeadPitch	−40.5° to 36.5°	−40° to 55°
HeadRoll	-	−35° to 35°

Table 3.4: Comparison of the range of head joints between Pepper and a human

3.3.3 NAOqi

The operating system of SoftBank robots is NAOqi OS which is an embedded GNU/Linux distribution based on Gentoo[26]. It runs different programs and libraries including NAOqi which is the main software that controls the robot. The NAOqi framework is provided to develop for these systems and can be used on Windows, Linux or macOS. It supports the programming languages C++, Python Java and JavaScript.

The basic structure is shown in Figure 3.10. The NAOqi executable is a broker which provides access to different methods over a network connection. When the application launches it loads the preferences file `autoload.ini` that defines which libraries are loaded. The libraries contain modules which are basically

[26]`http://doc.aldebaran.com/2-1/dev/tools/opennao.html`

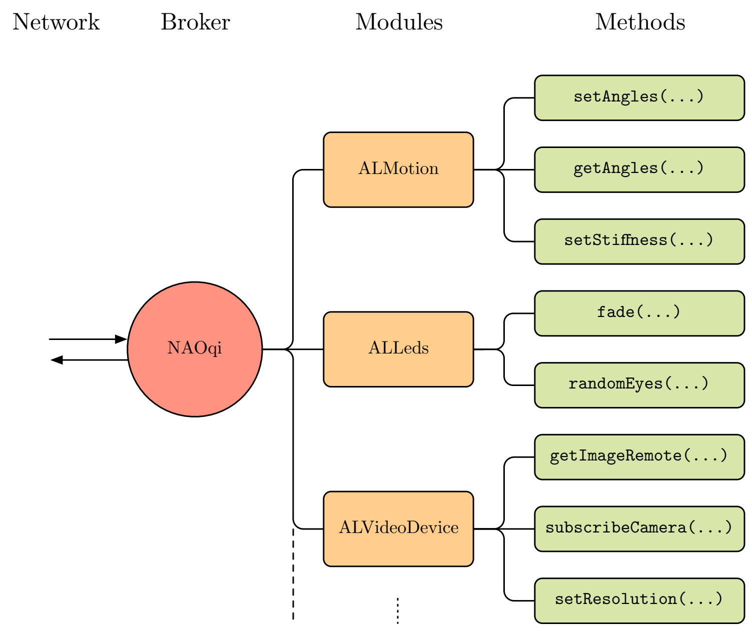


Figure 3.10: Structure of the NAOqi Framework

classes within the corresponding library (e. g. ALMotion, ALVideoDevice). These modules can either be used locally or remotely. A local module is compiled as a library and can only run on the robot directly. A remote module is compiled as an executable and can be used on an external computer. It communicates with the robot over a network connection. In the teleoperation application, the modules are used remotely because information from the Leap Motion and Oculus Rift is needed to control the robot. These devices are connected to a computer which calculates the required movements and sends the corresponding instructions remotely to the robot. Vice versa, sensor data from the robot is needed for rendering to the Oculus Rift such as images from the cameras and the head pose.
To exchange information with the robot, a proxy to the needed module has to be created. A proxy is an object that contains all methods of the corresponding module (Figure 3.10). It can either be created locally or remotely. The latter requires the IP and port of a broker, and the module must be in this broker[27]. The internal communication between the upper-level software (NAOqi modules) and the low-level electronic devices (boards, sensors, actuators) is handled by the DCM[28] (device communication manager). It sends commands

[27]`http://doc.aldebaran.com/2-1/dev/naoqi/index.html`
[28]`http://doc.aldebaran.com/2-1/naoqi/sensors/dcm.html`

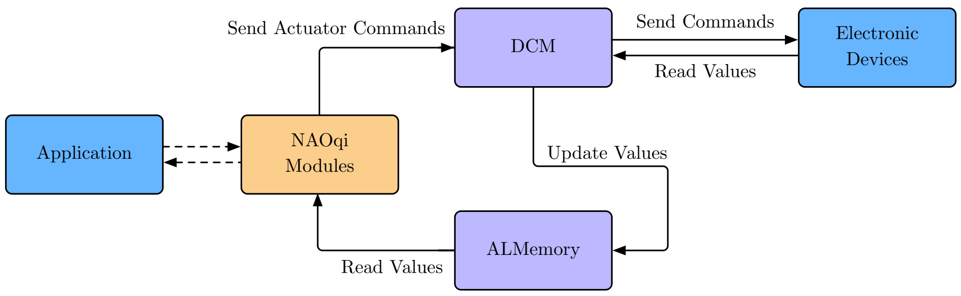


Figure 3.11: Communication inside NAOqi framework

to the actuators and receives sensor data. The data is saved in a centralised memory called ALMemory which can be accessed by upper-level modules[29]. The communication process is shown in Figure 3.11.
In NAOqi two possibilities exist for calling functions. Blocking calls will be executed sequentially. Non-blocking calls will initiate a task in a parallel thread. When the task is complete, the corresponding thread will be terminated. This allows, for example, to move different joints and access the video feed simultaneously.

3.4 Teleoperation Setup

The complete hardware setup is shown in Figure 3.12. The teleoperation application is running on a computer. The Leap Motion sensor is connected via USB and sends the measured arm pose at a rate of approximately 115 Hz. The Oculus Rift virtual-reality headset is connected via USB and HDMI. The head pose is read by the teleoperation application. An instance of NAOqi runs on the robot which communicates with the controlling computer via a network connection. The teleoperation application polls the current joint angles and the camera images of the robot. This data is used to calculate the desired joint angles and render the images for the Oculus Rift headset.

[29]`http://doc.aldebaran.com/2-1/naoqi/core/almemory.html`

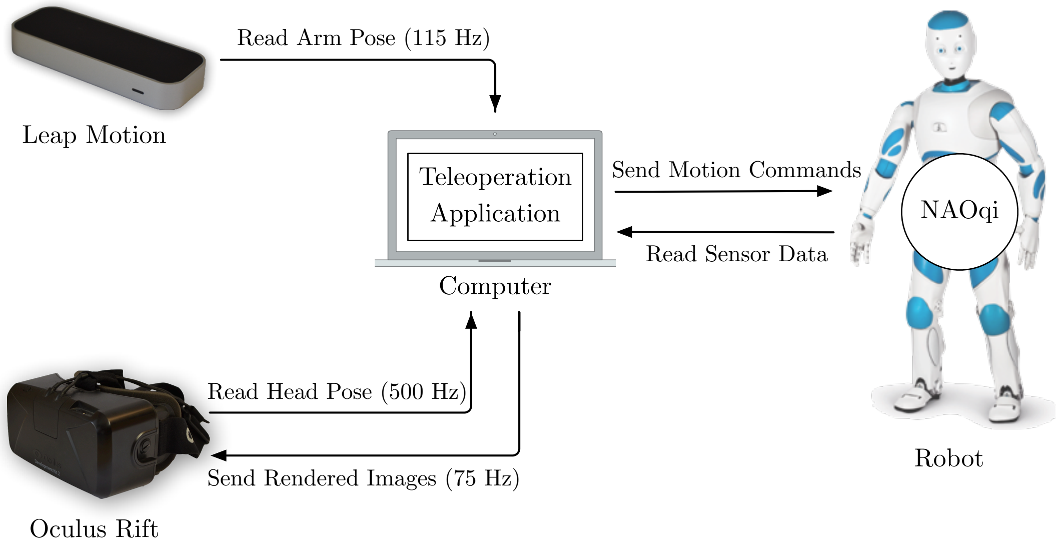


Figure 3.12: Setup of the different hardware components used for teleoperation and which kind of data is sent between the devices

4 Implementation

The components described above are combined to create the teleoperation application. In this chapter, we describe the algorithm for calculating the joint angles of the robot, how the camera images of the robot are displayed inside the virtual-reality headset and how the different software modules are combined. The focus is on providing the user with an intuitive method to control the upper body of a humanoid robot. Low latency and small amount of jitter of the robot's movements as well as an implementation to avoid cybersickness are key factors to accomplish a satisfying experience.

4.1 Imitation of User Motion

Forward Kinematics is the problem of calculating the end-effector position from the values of the joint parameters. This is done by solving the specific kinematic equations of the robot. Inverse Kinematics is the process of computing the required joint angles to reach a desired position of the end-effector. In the teleoperation application, the sensor information of the Leap Motion haas to be translated to the appropriate joint angles. NAOqi does not provide a solver for inverse kinematic problems for Romeo and Pepper. There are different possibilities to solve this problem. It can be calculated analytically by solving trigonometric equations for a specific kinematic chain or approximated iteratively by minimising an error function. The latter numerical solution is often used in complex kinematic chains, due to the difficulty of determining the inverse function of the nonlinear forward kinematic equations. For more details, refer to Section 2.2.1.

4.1.1 Homogeneous Transformation

The following mathematical description and notation is based on [59]–[61]. A position in space can be described by a 3×1 vector. It is important, to attach this vector to a specific coordinate system because in robotics there are typically multiple coordinate systems. For example, each joint has at least one local coordinate frame. In this work, only Cartesian coordinate systems are used and the terms “reference frame”, “coordinate frame” and “coordinate system” are used equivalently. To show the coordinate frame the vector is defined in, a subscript is written. For example,

$$\mathbf{p}_A = \begin{bmatrix} p_{Ax} \\ p_{Ay} \\ p_{Az} \end{bmatrix} \tag{4.1}$$

defines a vector in the coordinate system $\{A\}$. The three values p_{Ax}, p_{Ay}, p_{Az} are the distances along the axes of $\{A\}$.

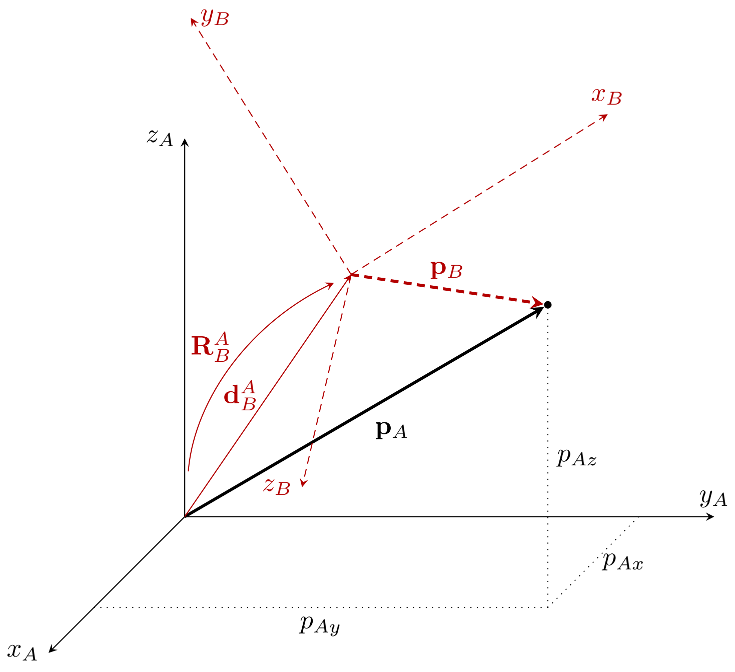


Figure 4.1: Vector relative to frame and coordinate transformation.

A vector can be transformed to a different coordinate system $\{B\}$ by

$$\mathbf{p}_B = \mathbf{R}_B^A \mathbf{p}_A + \mathbf{d}_B^A \tag{4.2}$$

illustrated in Figure 4.1. The rotation transformation matrix

$$\mathbf{R}_B^A = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.3}$$

rotates the axes of the coordinate system $\{A\}$ to equal the orientation of the frame $\{B\}$. For example, the matrix

$$\mathbf{R}_{x,\psi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \tag{4.4}$$

rotates the reference frame counter-clockwise by an angle ψ around the x-axis (Roll). Every rotation in space can be executed by performing rotations around three coordinate axes subsequently. The rotation matrices around y (Pitch) and z (Yaw) are

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \quad \mathbf{R}_{z,\phi} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.5}$$

Matrix multiplication is not a commutative operation and therefore, the order of rotations is important. The order used in this work is $x - y - z$, as used in the NAOqi documentation and most of the robots' joints, so

$$\mathbf{R} = \mathbf{R}_{z,\phi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\psi}. \tag{4.6}$$

In the standard definition of a rotation matrix, the columns have unit magnitude and are orthogonal. A consequence is that

$$\mathbf{R}_B^A = \left(\mathbf{R}_A^B\right)^{-1} = \left(\mathbf{R}_A^B\right)^T, \tag{4.7}$$

which means the rotation matrix from coordinate frame $\{A\}$ to $\{B\}$ is the transpose of the rotation matrix from coordinate frame $\{B\}$ to $\{A\}$.
The origin is moved by adding the translation vector

$$\mathbf{d}_B^A = \begin{bmatrix} d_{Ax} \\ d_{Ay} \\ d_{Az} \end{bmatrix}. \tag{4.8}$$

The summation in (4.2) can be included in a matrix operation by introducing homogeneous coordinates and the homogeneous transformation:

$$\begin{bmatrix} \mathbf{p}_B \\ 1 \end{bmatrix} = \mathbf{T}_B^A \begin{bmatrix} \mathbf{p}_A \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & \mathbf{R}_B^A & & \mathbf{d}_B^A \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} \mathbf{p}_A \\ 1 \end{bmatrix}. \tag{4.9}$$

This simplifies the transformation to a pure matrix multiplication, but because the 4×4 matrix ${}_A^B\mathbf{T}$ is not orthogonal, (4.7) is not applicable anymore:

$$\mathbf{T}_B^{A^{-1}} \neq \mathbf{T}_B^{A^T}. \tag{4.10}$$

4.1.2 Algorithm for Robot Joint Angle Calculation

The imitation of the user's head orientation is straightforward. The Oculus Rift provides the head pose of the user. The sensor readings are converted to Euler angles and sent to the robot. For Romeo, $NeckYaw$, $HeadPitch$ and $HeadRoll$ are set to the corresponding measured angles. Pepper lacks the possibility of tilting its head, so only $HeadYaw$ and $HeadPitch$ are set to the corresponding values.
For the robot's arm movements, the focus in our application is to be a good imitation of the user's arm movements. It is more important for the different parts of the arm (upper arm, forearm, hand) to point in the right direction, than the position of the end-effector to be precise. If only the position of the user's hand is considered for solving the inverse kinematic problem, the position of the robot's hand might be more accurate, but the robot's arm posture might differ a lot from the one of the user due to different physical dimensions of the arms. Therefore, the direction vectors of the upper arm, forearm and hand are used to calculate the joint angles of the shoulder, elbow and wrist respectively. This results in a natural movement of the robot's arms. If a precise position of the hand is needed, for example for grabbing, the user will act as a feedback loop and adjust the hand position due to the visual information.

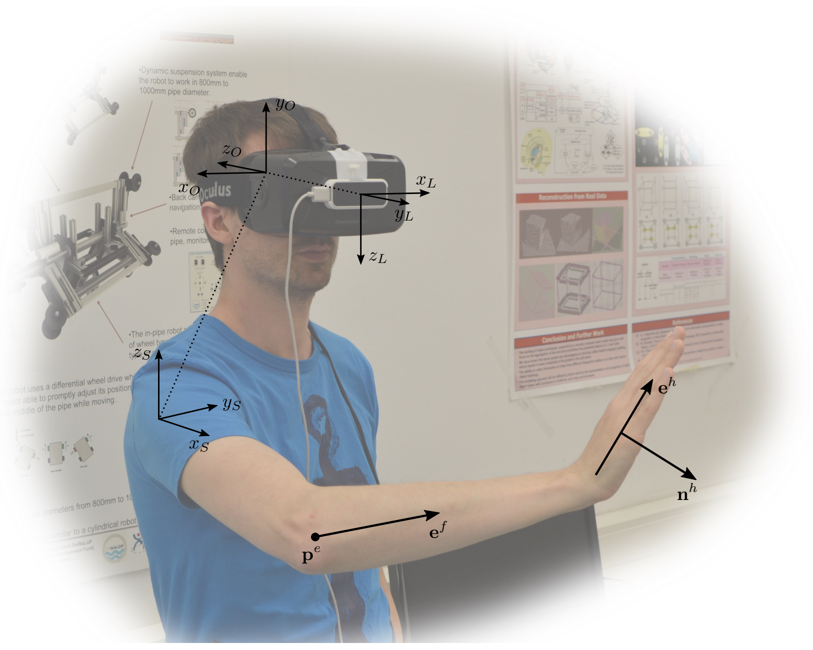


Figure 4.2: Coordinate frames on the user side, and the measured positions and directions of the Leap Motion that are used in the algorithm

Leap Motion provides many different measurements besides the palm position. For the motion calculation, the measured elbow position, forearm direction and hand direction and hand normal are used. This way, the inverse kinematic problem breaks down to solving simpler equations which can be solved analytically with trigonometry. The Leap Motion sensor is positioned on the front side of the Oculus Rift as shown in Figure 4.2. The sensor information needs to be transformed to the different frames of reference in the joints.
The position of the user's elbow is used to calculate the desired angles of the shoulder joints. The elbow position in the Leap frame $\mathbf{p}_L^e$ needs to be transformed to the shoulder frame by

$$\begin{bmatrix}\mathbf{p}_S^e \\ 1\end{bmatrix} = \mathbf{T}_S^O \mathbf{T}_O^L \begin{bmatrix}\mathbf{p}_L^e \\ 1\end{bmatrix}. \tag{4.11}$$

The matrix $\mathbf{T}_O^L$ transforms from the Leap to the Oculus frame. The first step is to scale the Leap data from millimetres to metres, then rotate the coordinate system to equal the one used for OpenGL and afterwards translate 21 cm in the direction of the negative z-axis to move the origin to approximately the middle of the head:

$$\mathbf{T}_O^L = \underbrace{\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & -0.21\\ 0 & 0 & 0 & 1\end{bmatrix}}_{Translation} \underbrace{\begin{bmatrix}-1 & 0 & 0 & 0\\ 0 & 0 & -1 & 0\\ 0 & -1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}}_{Rotation} \underbrace{\begin{bmatrix}0.001 & 0 & 0 & 0\\ 0 & 0.001 & 0 & 0\\ 0 & 0 & 0.001 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}}_{Scaling}. \tag{4.12}$$

The 21 cm are composed of 8 cm to get to the height of the eyes[1] and 13 cm to reach the middle of the head. Subsequently, the origin is moved to the shoulder with

$$\mathbf{T}_S^O = \underbrace{\begin{bmatrix}0 & 0 & -1 & 0\\ -1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}}_{Rotation} \underbrace{\begin{bmatrix}1 & 0 & 0 & \pm 0.21\\ 0 & 1 & 0 & 0.225\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}}_{Translation} \underbrace{\begin{bmatrix} & & & 0\\ & \mathbf{R}(q) & & 0\\ & & & 0\\ 0 & 0 & 0 & 1\end{bmatrix}}_{Rotation}. \tag{4.13}$$

The matrix $\mathbf{T}_S^O$ makes the sensor data independent of the head position. It transforms the elbow position to a fixed coordinate frame in the shoulder. $\mathbf{R}(q)$ is calculated from quaternions obtained by the Oculus sensors. For details refer to the literature e. g. [1]. A final rotation has to be performed to get to the robot's coordinate system as used by NAOqi.

[1] `http://blog.leapmotion.com/vr-essentials-need-build-scratch/`

The translations used above are based on the average dimensions of German males aged 26-40 [62]. Additionally, the position of our shoulder changes when we are moving. Therefore, the described transformation moves to origin only approximately to the centre of rotation of the shoulder joint. However, the approximation only influences the shoulder joint angles. Direction vectors are used to calculate the desired angles of the other joints. Therefore, no translation has to be performed as described below. Additionally, the user can easily adapt his arm posture to the error. Overall, the error made by the approximation is negligible for our teleoperation system and an additional sensor to get the shoulder position is not necessary.

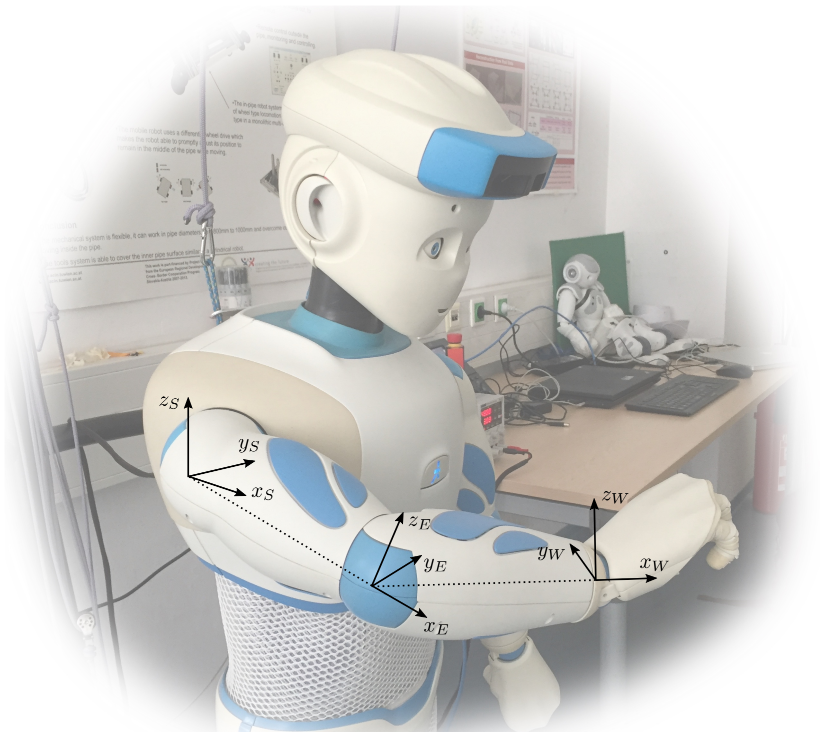


Figure 4.3: Coordinate frames on the robot side

The required pitch angle of the robot's shoulder can be calculated by

$$ShoulderPitch = \theta_S = \text{atan2}(-p^e_{Sz}, p^e_{Sx}) \tag{4.14}$$

where p_{Sx}, p_{Sy}, p_{Sz} are the coordinates of the elbow's position in the shoulder frame. The robot's $ShoulderYaw$ joint rotates with the arm if $ShoulderPitch$ is changed. Therefore, the coordinate system has to be rotated before calculating

$ShoulderYaw$. This is done by calculating

$$\mathbf{p}^e_{S'} = \mathbf{R}_{y,\theta_S}\mathbf{p}_S = \begin{bmatrix} \cos\theta_S & 0 & -\sin\theta_S \\ 0 & 1 & 0 \\ \sin\theta_S & 0 & \cos\theta_S \end{bmatrix} \mathbf{p}^e_S. \tag{4.15}$$

The updated elbow position $\mathbf{p}^e_{S'}$ is used for

$$ShoulderYaw = \phi_S = \operatorname{atan2}(p^e_{S'y}, p^e_{S'x}). \tag{4.16}$$

To calculate the elbow angles, the Leap Motion provides the direction of the forearm $\mathbf{e}^f_L$. This direction needs to be transformed to the robot's elbow coordinate system as seen in Figure 4.3. It is only necessary to apply the various rotational matrices and no translation because it is a direction and not a position:

$$\mathbf{e}^f_E = \mathbf{R}^S_E \mathbf{R}^O_S \mathbf{R}^L_O \mathbf{e}^f_L. \tag{4.17}$$

$\mathbf{R}^O_S$ and $\mathbf{R}^L_O$ are the rotations of $\mathbf{T}^O_S$ and $\mathbf{T}^L_O$ respectively. The only new matrix is $\mathbf{R}^S_E$, which transforms from the shoulder frame to the elbow frame. It depends on the robot's current joint angles of the shoulder and is calculated by

$$\mathbf{R}^S_E = \mathbf{R}_{z,\phi_S}\mathbf{R}_{y,\theta_S} = \begin{bmatrix} \cos\phi_S & \sin\phi_S & 0 \\ -\sin\phi_S & \cos\phi_S & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_S & 0 & -\sin\theta_S \\ 0 & 1 & 0 \\ \sin\theta_S & 0 & \cos\theta_S \end{bmatrix}, \tag{4.18}$$

Two different approaches exist to calculate the rotation matrices which are dependent on the robot's joint angles. Either the current angles measured by the robot's sensors or the calculated desired angles can be used. Both methods have advantages. When using the measured angles, the robot adapts the joints further down the kinematic chain to the currently measured angles of the joints that come before. This might reduce the error of the end-effector position during fast movements of the shoulder joints while the elbow joints hardly change. In a perfect world without latency, this method would be preferable. Due to the latency of sending the measured joint angles to the teleoperation computer, the robot adjusts the joint angles to measurements from the past. This results in an overshooting of the end-effector position and an additional delay because the correct end-positions of the joints can only be calculated when the measurements of the joints further up the kinematic chain are available. These problems do not occur when using the calculated desired joint angles. These are the angles of the joint configuration the robot will be at after the movement has finished. Of course, the calculated angles have to be bound to the range of the robot's joints. This is more complicated for Pepper because the range of $ElbowYaw$ changes depending on the value of *ElbowRoll*. However,

even if the changing range is ignored, the results are still satisfying. Therefore, this method is implemented in the teleoperation system.
ElbowRoll is computed by the equation

$$ElbowRoll = \psi_E = \operatorname{atan2}(e^f_{Ez}, e^f_{Ey}) \tag{4.19}$$

As before, the joint for *ElbowYaw* rotates when *ElbowRoll* changes. The forearm direction $\mathbf{e}^f_E$ needs to be rotated by

$$\mathbf{e}^f_{E'} = \mathbf{R}_{x,\psi_S}\mathbf{e}^f_E = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\psi_E & \sin\psi_E\\ 0 & -\sin\psi_E & \cos\psi_E\end{bmatrix}\mathbf{e}^f_E, \tag{4.20}$$

before calculating

$$ElbowYaw = \phi_E = \operatorname{atan2}(e^f_{E'y}, e^f_{E'x}). \tag{4.21}$$

Leap Motion provides a vector pointing from the wrist position to the palm centre, as well as a vector perpendicular to the plane formed by the hand. This normal vector $\mathbf{n}^h_L$ is used to calculate *WristRoll* after bringing it to the wrist coordinate frame by

$$\mathbf{n}^h_W = \mathbf{R}^E_W\mathbf{R}^S_E\mathbf{R}^O_S\mathbf{R}^L_O\mathbf{n}^h_L \tag{4.22}$$

with

$$\mathbf{R}^E_W = \begin{bmatrix}\cos\phi_E & \sin\phi_E & 0\\ -\sin\phi_E & \cos\phi_E & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\psi_E & \sin\psi_E\\ 0 & -\sin\psi_E & \cos\psi_E\end{bmatrix}. \tag{4.23}$$

For Pepper, $\mathbf{R}^E_W$ contains an additional rotation of $-9°$ around the y-axis due to its structure. Now, we can calculate

$$WristRoll = \psi_W = \operatorname{atan2}(n^h_{Wy}, -n^h_{Wz}). \tag{4.24}$$

For Pepper, the calculation of the arm joints ends at this point. Romeo has two additional degrees of freedom in his wrist. To calculate the angles for the two additional joints the direction vector $\mathbf{e}^h_L$ pointing from the position of the wrist to the palm centre is used. It is transferred to the wrist coordinate frame rotated by *WristRoll* by

$$\mathbf{e}^h_{W'} = \mathbf{R}_{x,\psi_W}\mathbf{R}^E_W\mathbf{R}^S_E\mathbf{R}^O_S\mathbf{R}^L_O\mathbf{e}^h_L \tag{4.25}$$

with

$$\mathbf{R}_{x,\psi_W} = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\psi_W & \sin\psi_W\\ 0 & -\sin\psi_W & \cos\psi_W\end{bmatrix} \tag{4.26}$$

which is used for

$$WristYaw = \phi_W = \operatorname{atan2}(e^h_{W'y}, e^h_{W'x}). \tag{4.27}$$

$WristPitch$ is calculated by

$$\mathbf{e}^h_{W''} = \mathbf{R}_{z,\phi_W}\mathbf{e}^h_{W'} = \begin{bmatrix} \cos\phi_W & \sin\phi_W & 0 \\ -\sin\phi_W & \cos\phi_W & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{e}^h_{W'} \tag{4.28}$$

and

$$WristPitch = \theta_W = \operatorname{atan2}(-e^h_{W''z}, e^h_{W''x}). \tag{4.29}$$

4.1.3 Filter

The calculated joint angles are sent to the robot through the NAOqi SDK with the function `ALMotionProxy::setAngles()`. This function is a non-blocking call which means the calling thread does not need to wait for the robot to reach the desired location. As soon as new desired joint angles are available due to new measurements, the updated values are sent to the robot. This ensures the fastest possible translation from human motion to robot motion.
Sensor noise results in jitter of the robot motion. A filter can be added to reduce this jitter. However, a filter adds time latency (so called lag) to the system. The filter should be simple to limit computation time. The 1€ filter introduced by Casiez, Roussel and Vogel [63] is specifically developed for tracking human motion. It is a first order low-pass filter with an adaptive cutoff frequency. An ordinary low-pass filter would reduce jitter at slow speeds but would not be able to follow fast movements. The 1€ filter adapts the cutoff frequency for the updated value depending on an estimate of the speed. When holding the hand steady, a low cutoff frequency is used to reduce jitter. When moving fast, the cutoff frequency is increased to reduce lag. A higher cutoff frequency increases jitter, but humans are more sensitive to lag when moving fast according to Casiez, Roussel and Vogel.
In our implementation, a separate filter is used for each joint angle. The 1€ filter has two tuneable parameters. These are set to the same values for every joint angle.

4.2 User Interface

The goal is to provide the user with a natural and intuitive interface to control the robot without causing discomfort. The Oculus Rift is used to display the camera feed of the robot to the user. There are two APIs supported by the Oculus SDK for rendering to the head mounted display: OpenGL and Direct3D. The latter is officially only implemented on the Windows platform[2]. OpenGL is an open standard and is available on multiple platforms. To ensure cross-platform compatibility, OpenGL was the obvious choice to use in the teleoperation application.

4.2.1 OpenGL API

The following description of the OpenGL API is based on [64]. The OpenGL API is a software library to access the features of graphics hardware in order to render to a display. Rendering is the process of creating an image from models. The models used in OpenGL are constructed from geometric primitives (points, lines and triangles) defined by the corresponding vertices. A vertex is basically a bundle of data values that are processed together, such as positional and colour information. The rendering process consists of many different stages. The sequence is called rendering pipeline and is shown in Figure 4.4.

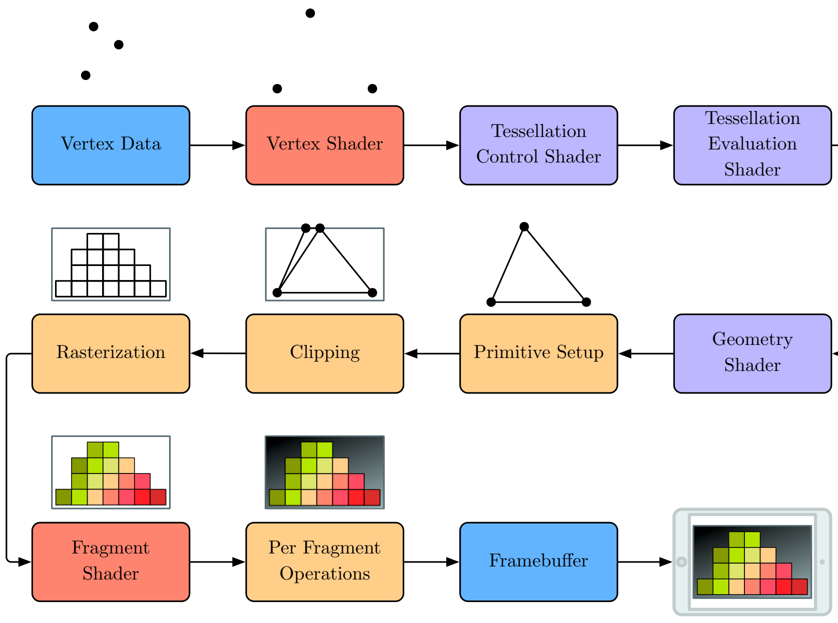


Figure 4.4: The stages of the OpenGL rendering pipeline

An essential part of the rendering pipeline are shaders, which are special functions that the graphics hardware executes. They are compiled by OpenGL to be

[2]https://blogs.msdn.microsoft.com/windows-embedded/2009/06/25/component-tales-directx/

used by the graphics processing unit (GPU). A vertex shader is called for each vertex and processes the data associated with the vertex. It basically determines the position of primitives on the screen usually by using transformation matrices. The tessellation shader continues to process this data and increases the number of primitives by subdividing primitives and generating a mesh of triangles, for example. The geometry shader accepts complete primitives as a collection of vertices as input. It can change the number and type of the primitives. The tessellation and geometry shader are optional and are not used in the teleoperation application.
The previous shader stages operated on vertices and the information how to build primitives are carried along internally. The primitive assembly stage organises the vertices to create primitives. The clipping stage takes care of vertices outside of the rendering target and is done automatically by OpenGL. Rasterization creates fragments from the primitives. A fragment is a "candidate pixel" which might become a real pixel on the screen, but can still be rejected by further steps. Rasterization can be understood as the transformation from primitives in the 3D space to a 2D image space. The fragment shader determines the fragment's colour. For example, an image can be mapped onto a surface (texture mapping). The last stage is called per-fragment operations, which determines the fragment's visibility by various tests (depth testing[3] and stencil testing).

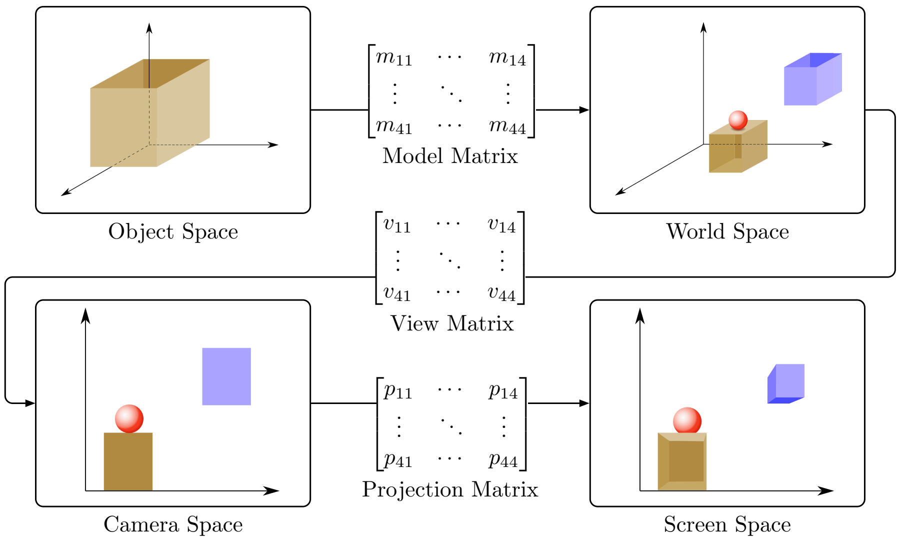

$\begin{bmatrix} m_{11} & \cdots & m_{14} \\ \vdots & \ddots & \vdots \\ m_{41} & \cdots & m_{44} \end{bmatrix}$
Model Matrix
Object Space
World Space
$\begin{bmatrix} v_{11} & \cdots & v_{14} \\ \vdots & \ddots & \vdots \\ v_{41} & \cdots & v_{44} \end{bmatrix}$
View Matrix
$\begin{bmatrix} p_{11} & \cdots & p_{14} \\ \vdots & \ddots & \vdots \\ p_{41} & \cdots & p_{44} \end{bmatrix}$
Projection Matrix
Camera Space
Screen Space

Figure 4.5: Transformation matrices used in OpenGL

[3]Depth testing is commonly known as z-testing

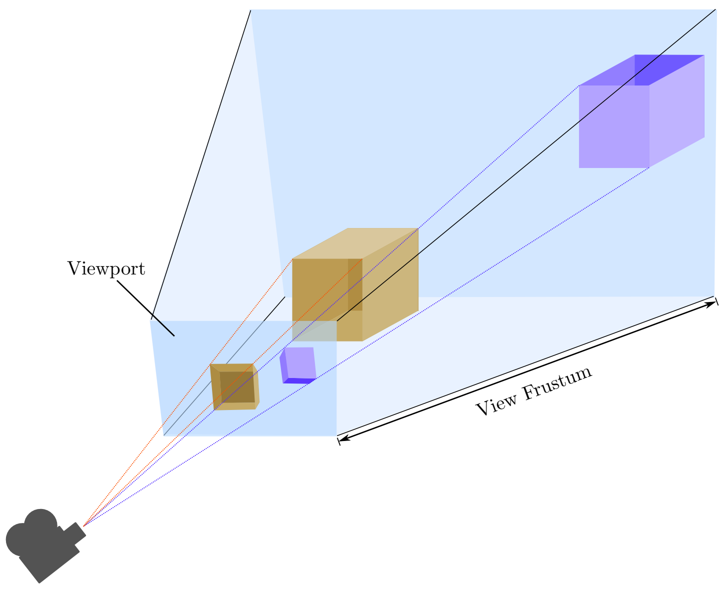


Figure 4.6: Perspective projection[4]

The most important stages in the rendering pipeline that can be influenced by the programmer are the vertex shader and the fragment shader. The position of primitives on the screen is calculated by projecting the models in 3D space onto a 2D image space. The primitives are defined by various vectors and are transformed by several matrix multiplications in the vertex shader (similar as described in Section 4.1.1).
Usually, the transformation is separated into three steps as seen in Figure 4.5. The model matrix moves an object such as the screen for our video feed to a position in the world space by translation, rotation and scaling. The view matrix moves the virtual camera position in space. In OpenGL, the camera is fixed at the origin, so instead of moving the camera, the world around it is moved, resulting in the same effect. This is needed in the teleoperation application to adjust the position of the camera during the user's head movements. Additionally, when rendering to the Oculus Rift two cameras with different view matrices are required, one for each eye. In other words, the same scene is rendered twice with two different camera positions for the two eyes.
The projection matrix performs the perspective transformation from 3D space to the 2D image space as illustrated in Figure 4.6. Objects further away from the camera are smaller than objects in the front. The view frustum is the area

[4]adapted from `http://www.xojo3d.com/tut002.php`

that can be seen from the camera. It is dependent on the field of view and the near and far plane. Objects outside of this area cannot be seen in the camera image. The projection matrix is also a 4×4 matrix and is provided by the Oculus SDK[5].

4.2.2 Virtual World for Video Feed

The camera feed of the robot has to be displayed in the Oculus Rift headset. The easiest possibility of doing so is to show the left and right camera image on the left and right side of the HMD respectively for a robot with a stereo camera setup. This would cause major discomfort to the user as described in Section 3.2.2 because the user's and the robot's movement speed and range differ. Therefore, the perceived and real motion would diverge. The implemented solution to this problem is a moving reference world frame that reacts quasi-instantly on the user's motion. The video feed is displayed on a floating screen that adjusts to the robot's head pose. The idea is adapted from the implementation for live webcam video in the Oculus Rift from [50].

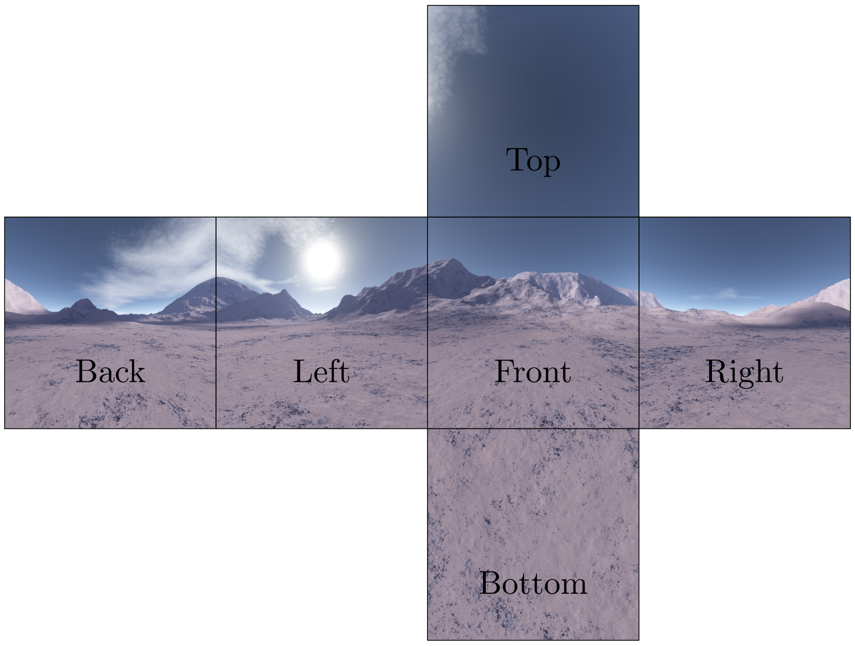


Figure 4.7: Textures used for the skybox[6]

[5]For details on the calculation of the projection matrix, please refer to relevant literature.
[6]Created by Chris Matz `http://www.custommapmakers.org/skyboxes.php`

As a reference world, we use a so-called skybox. A skybox is a cube with textures on each side with a seamless transition between them. Typical textures are the sky, clouds and distant mountains. The textures used for the teleoperation application are shown in Figure 4.7. The camera is positioned in the centre of this cube, and it appears as if the user is in an environment of distant 3D surroundings.
To render a skybox, a vertex with the coordinates of a unit cube is created. In the vertex shader, the model-view-projection transformation is performed. The model matrix scales the cube to be far away. The view matrix is mainly built up from the rotation matrix obtained by the Oculus SDK of the user's head position. This way, the skybox moves quasi-instantly as the user moves to ensure a natural feeling and avoid cybersickness. In the fragment shader, the textures are projected onto the sides using a special texture type from OpenGL called Cubemap Texture[7].

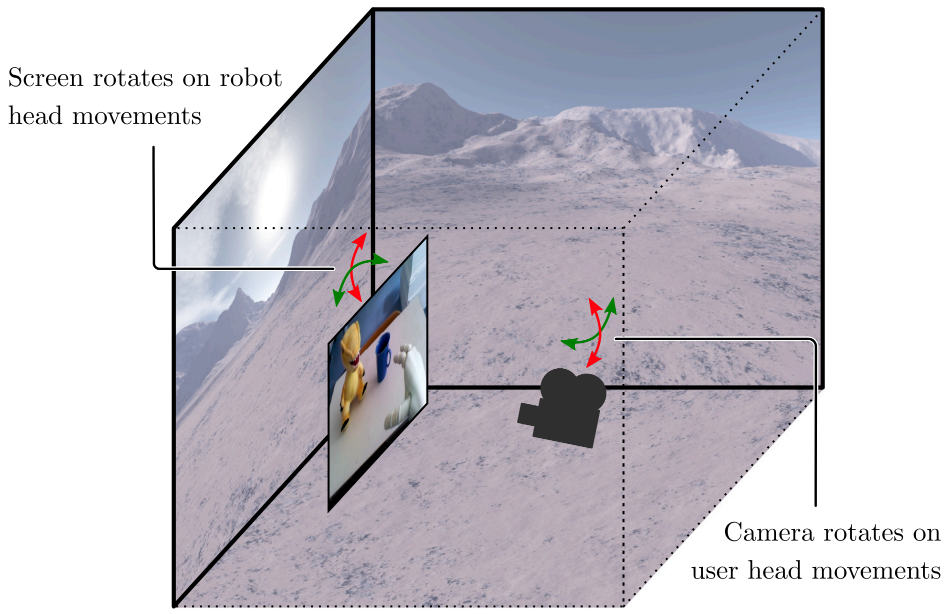

Screen rotates on robot head movements
Camera rotates on user head movements

Figure 4.8: Virtual screens with robot's camera images and virtual camera moving in skybox

The floating screen is a rectangle with the same aspect ratio as the robot's cameras. In the vertex shader, the view matrix is the same as before. The model matrix is a combination of translating the rectangle in front of the virtual

[7]`www.opengl.org/wiki/Cubemap_Texture`

camera and rotating it according to the sensor readings of the robot's head pose. This way, the screen moves in the world space to the position the robot is actually looking at. If the robot has a stereoscopic camera setup, the image of the left camera is displayed on the virtual screen in the texture rendered for the left eye and the image of the right camera is displayed on the virtual screen in the texture rendered for the right eye. A horizontal translation of the screens in the two rendered images is added to get a stereoscopic effect. This increases the otherwise fairly small horizontal field of view of only 50°. In the fragment shader, the current video image is projected onto the rectangle. The described solution is illustrated in Figure 4.8.
The rendering process is done in a loop, so the frame is updated in every iteration. The scene has to be rendered for both eyes separately with a small offset in the virtual camera position. These rendered images are displayed next to each other to create a stereoscopic image. The rendered scene is shown in Figure 4.9. The Oculus SDK applies the necessary corrections to counteract the distortions introduced by the lenses in the Oculus Rift (pincushion distortion, chromatic aberration) as described in Section 3.2.3. The final result that is displayed in the Oculus Rift is shown in Figure 4.10. The barrel distortion counteracts the pincushion distortion introduced by the lenses. At the edges in the periphery red and blue fringes can be seen. These are introduced by shifting the different colour channels in different directions to negate the chromatic aberration.

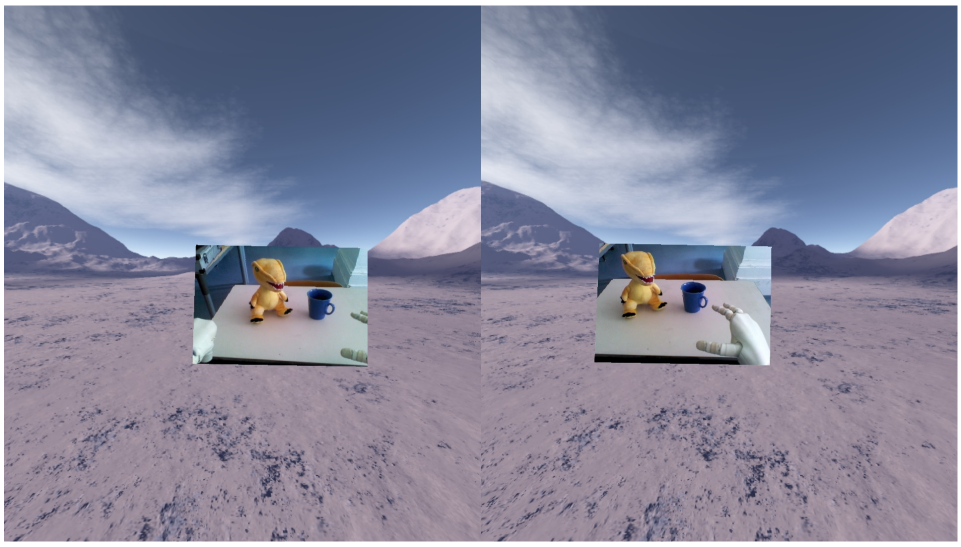


Figure 4.9: Rendered scene with robot's camera images in the virtual environment without distortion correction

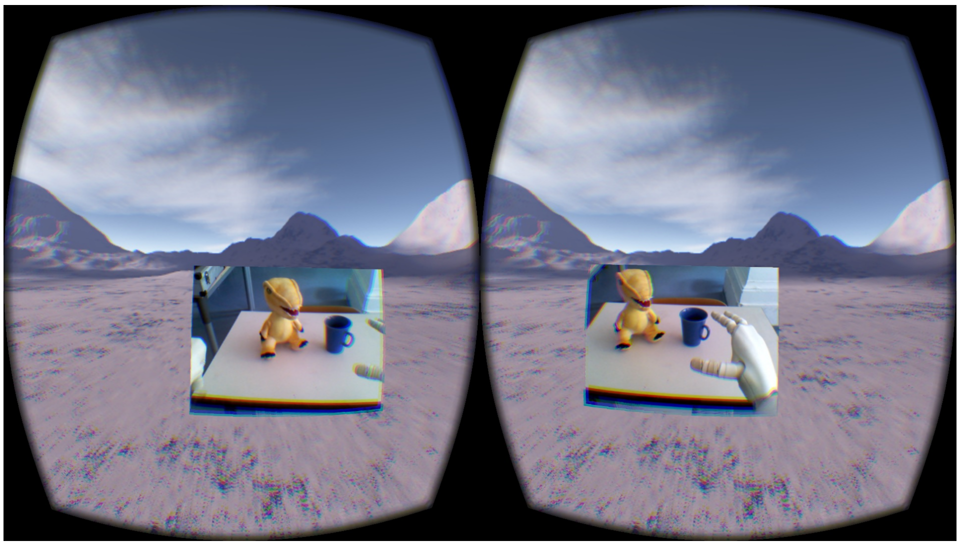


Figure 4.10: Rendered scene with robot's camera images in virtual environment for Oculus Rift

4.3 Integration of Software Modules

In this section, the integration of the software modules described above into one teleoperation application is explained. The two major software modules Imitation of User Motion and User Interface are easily separable. Therefore, they can be run simultaneously in parallel using multithreading. A program can consist of multiple threads that execute tasks concurrently. Threads access the same address space and therefore can interact with each other [65]. We use this technique to minimise latency.
In Figure 4.11, the structure of the teleoperation application and how it interacts with the external devices is illustrated. The User Interface Thread contains the rendering loop which updates the images for the Oculus Rift in every iteration. Therefore, it needs the current sensor readings of the head joints of the robot, the camera images of the robot and the measured head pose of the user. Due to the latency of the network connection to the robot, the data from the robot is obtained by two separate threads. The Joint Thread requests the latest joint sensor readings from the robot and saves them in an object that is accessible from the other threads. The Camera Thread retrieves the most recent images from the robot's cameras and loads them directly to the GPU. In the User Interface Thread, the joint object with the robot's sensor readings is used to calculate the model matrix and therefore the position of the virtual screen. The measured head pose of the user provided by the Oculus Rift is used to

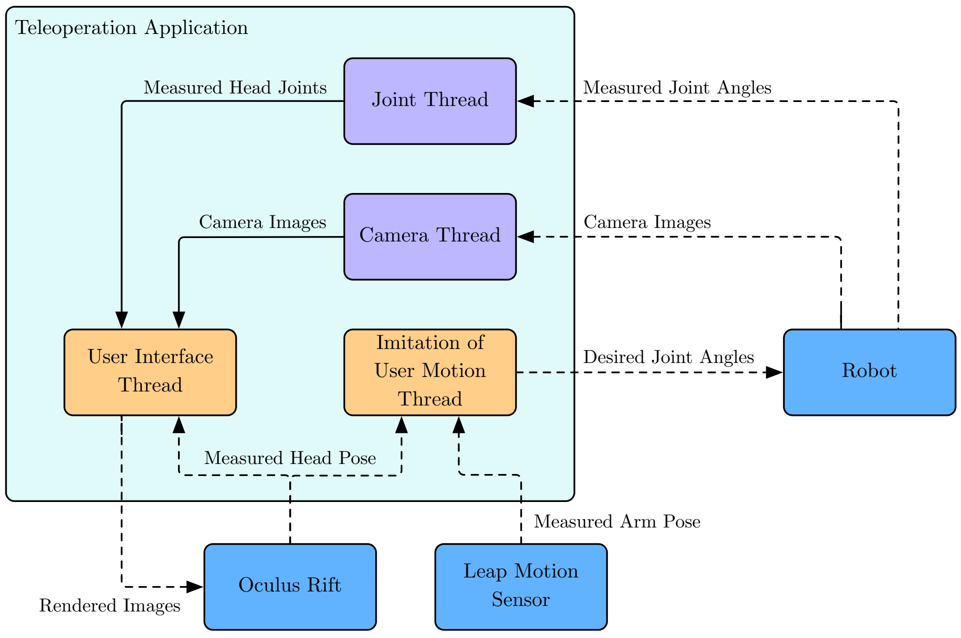


Figure 4.11: The software modules for Imitation of User Motion and User Interface run in parallel in different threads and receive inputs from the used hardware components; Camera Thread and Joint Thread buffer the camera images and the joint measurements respectively to overcome latency issues

determine the orientation of the virtual camera. Additionally, the function to display the robot's camera image on the virtual screen is called. The rendered images are sent to the Oculus Rift at the end of the rendering loop.
Without the buffering performed by the Camera Thread and the Joint Thread, the rendering loop would have to wait for the robot to send the requested data via the network connection. Additionally, the robot's camera only updates at a rate of 30 Hz. Therefore, latency is introduced, and it would not be possible to accomplish a frame rate of 75 Hz. For the sensor readings from the Oculus Rift, no separate thread is needed because the data is sent via USB connection at a rate of 500 Hz and very low latency.
The Imitation of User Motion Thread calculates the desired joint angles from the measured head pose provided by the Oculus Rift and from the measured arm pose provided by the Leap Motion sensor. Both devices use a USB connection for data transfer, and therefore no additional threads are needed. The Imitation of User Motion Thread does not need the sensor readings of the robot's joints in our implementation. The transformation matrices for the different coordinate

systems of the robot are generated from the angles calculated inside this thread. as described in Section 4.1.2.

5 Experiments

In this chapter, the experimental setup and results are described. It is grouped in experiments in the simulator, experiments on the real robot and a user study. The focus of the teleoperation system is to provide an intuitive and satisfactory user experience. The system has already been used by over a hundred people in demo settings such as the open door day or the Pepper World 2017 event. For these settings, the available data is limited to observations of the researcher. It is planned to do a more thorough user study focused on task performance at a later stage. The measurements described are meant to show characteristics of the application and not to provide an accurate analysis of the precision of the system for which a more complex experimental setup would be needed.
In Figure 5.1, a motion sequence is performed, and Pepper imitates the motion.

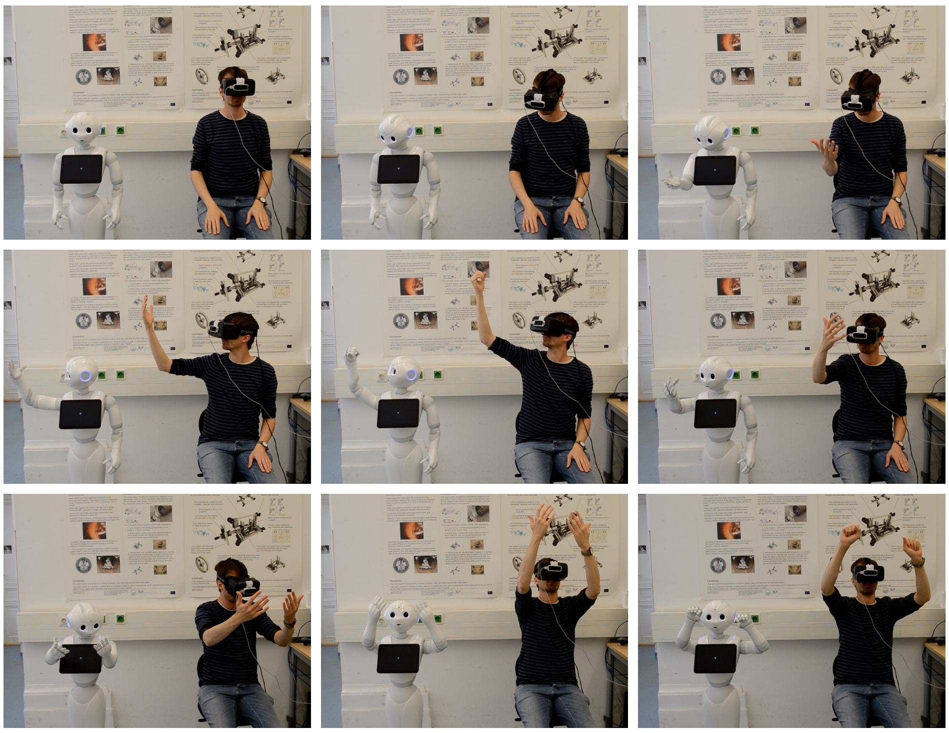
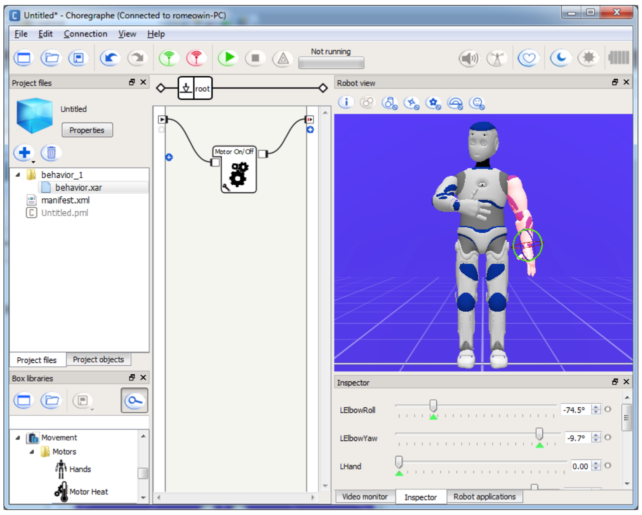


Figure 5.1: Motion sequence imitated by Pepper

5.1 Experiments in the Simulator

Most of the development process was done on a simulated robot in the provided software Choregraphe [66]. A virtual robot is created by running an instance of NAOqi on the simulating computer. Choregraphe connects to this instance and shows the robot's movements, as displayed in Figure 5.2. Unfortunately, no video images of the simulated robot's cameras are provided, so there is no feedback of the movements displayed in the Oculus Rift. Other simulation software is available to accomplish this but was not used for the development process.



LElbowRoll -74.5°
LElbowYaw -9.7°
LHand 0.00

Figure 5.2: Screenshot of Choregraphe with simulated robot

To discuss several aspects of the arm movement, the position of the palm position of the right hand is plotted. In Figure 5.3 the trajectory of a horizontal eight movement is shown. The origin of the coordinate system is in the shoulder (compare Figure 4.3). The blue trajectory is the palm position data captured by the Leap Motion controller. The red trajectory is the actual position of the Romeo robot's palm calculated by forward kinematics with the sensor readings of the robot's joint angles. The trajectories match only approximately due to kinematic differences of the human and the robot. The arms of the robot are shorter than the user's arms. Furthermore, the imitation algorithm used tries to copy the pose of the user's arm instead of minimising the error of the end-effector position as described in Section 4.1.2. Nevertheless, the overall shape of the trajectory is reasonably copied.
In Figure 5.4, a different movement is shown divided into the three coordinate axes. We recorded the movement performed by the user beforehand, so we

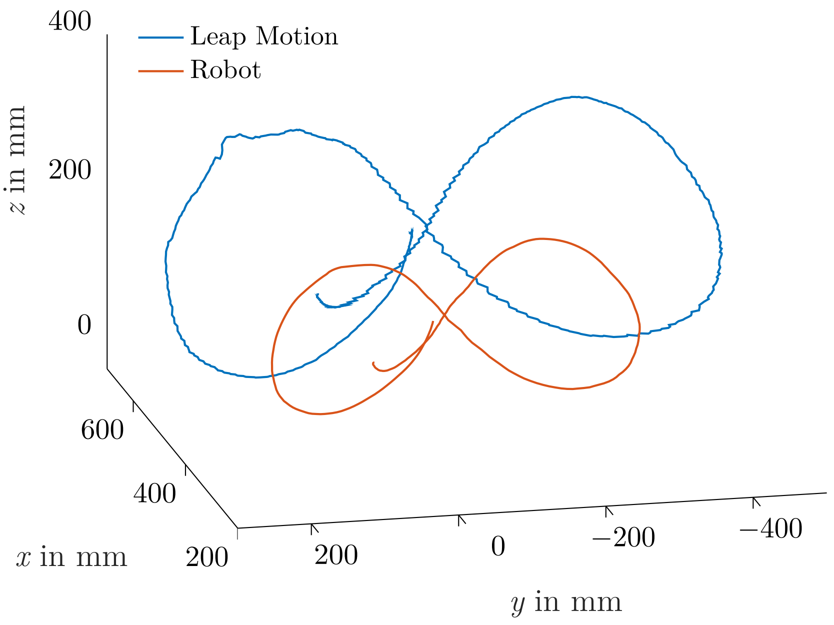


Figure 5.3: Comparison of the palm position of the user and the simulated Romeo robot during horizontal eight movement

can compare how the different robots handle the same input. Additionally, we see differences of the real robot and the simulation. The yellow line is a scaled version of the robot motion. It is calculated by forward kinematics from the robot's joint readings and the user's arm dimensions. The tracked palm position of the Leap Motion is transformed to the shoulder coordinate system. This transformation is an approximation because the shoulder position is not measured as described in Section 4.1.2. The same transformation is used for the calculation of the desired joint angles, so the error made by the approximation is cancelled out. However, as mentioned above the measurements are meant to show characteristics of the robot and the algorithm and not to determine the precision of the end-effector position accurately. Especially, since the focus of the approach is not to approximate the end-effector position of the user but rather imitate the arm posture.

The movement starts with the arm stretched out in front of the shoulder ($x \approx 620$, $y \approx -210$, $z \approx 20$). Then the hand is moved horizontally to the right (seconds 3-6) and raised afterwards (seconds 6-10). At second 16 the arm is stretched out in the front of the user again. Then the hand is moved in front of the face with the palm facing the user (seconds 23-27). Afterwards, it is returned to the starting position again. The movement was chosen to be in

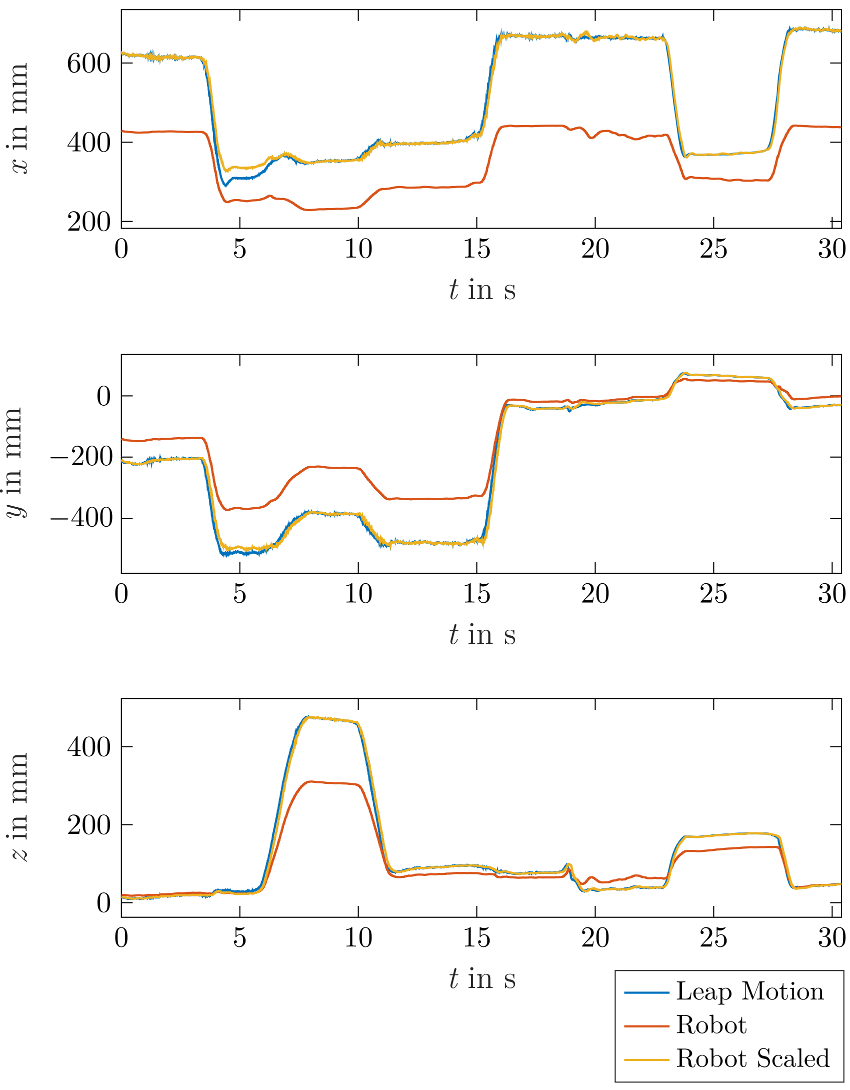


Figure 5.4: Comparison of the tracked palm position, the simulated Romeo's palm position and Romeo's scaled palm position during a movement for each coordinate axis

the operational space of Romeo and Pepper, be easily reproducible and use all joints except $WristPitch$ and $WristYaw$ because they are not available on Pepper. We recorded the sensor measurements of the Oculus Rift and Leap Motion sensor and then fed it to the algorithm. We chose this approach so that we can use the exact same sensor readings with Pepper and Romeo. This way, we can compare the different characteristics of the robots. The recorded motion can also be used in the simulator as well as on a real robot to compare

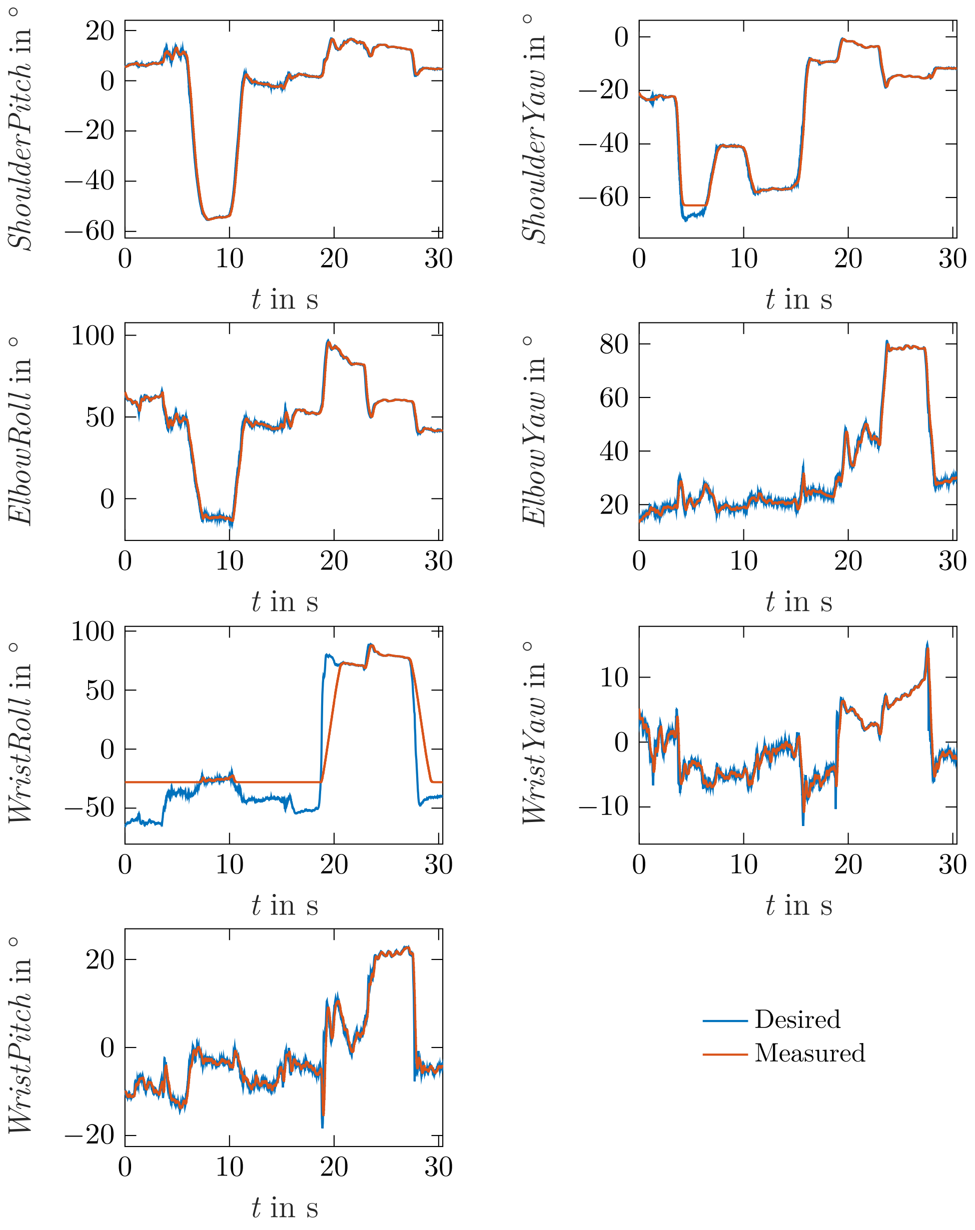


Figure 5.5: Desired and measured joint angles of the simulated Romeo robot during movement

the latency.
The robot is able to follow the movement for the most part. The mean Euclidean distance between the measured palm position of the user and the scaled palm position of the robot for this movement is 10.56 mm with a standard deviation

of 15.4 mm. If the time delay of ≈83 ms is cancelled out, the mean Euclidean distance is reduced to 5.68 mm with a standard deviation of 8.45 mm.
The biggest error occurs between 4 s and 6 s, especially in the x-axis. This error can be described when looking at the desired and measured angles shown in Figure 5.5. The robot's *ShoulderYaw* cannot reach the desired angle due to the joint's limits. This results in an error of the palm position mainly in the x and y coordinate.
Another problem is that the *WristRoll* joint cannot follow the desired joint angle over the whole course of the movement even though it exceeds the range of the human equivalent as shown in Table 3.1. The reason is the different structure of the robot's arm and the human arm. We can provoke a rotation of the palm in the shoulder joint[1]. Romeo lacks this degree of freedom in the shoulder and instead has an additional degree of freedom in the elbow. Therefore, the algorithm tries to compensate this rotation with its origin in the shoulder by decreasing the joint angle of *WristRoll*. Additionally, the motor in *WristRoll* is slower than the other motors, which can be seen at 19 s to 27 s where the robot is not able to follow the desired angle. These problems do not influence the palm position for this movement severely, but the orientation of the palm is impaired. For the newer Pepper robot, the range of this joint is different, and the motor's speed is increased, to overcome these problems. It is also planned to change the range for future Romeo prototypes from −210° to 30° to −140° to 80°.The desired angles in Figure 5.5 are discontinuous which is a result of the sensor noise of the Leap Motion data. The influence of the noise on the robot's motion is negligible because the motors are not able to follow these required fast movements and therefore perform a smoothing action. To further reduce jitter due to noise, a filter can be applied to the desired joint angles. A filter will introduce time latency. The 1€-filter is a simple filter for tracking human motion which can be tuned for a good tradeoff between reducing jitter and introducing lag. It is described in Section 4.1.3. We decided against using the filter in favour of lower latency.

5.2 Experiments on the Robot

The teleoperation system was tested with the humanoid robots Romeo and Pepper of SoftBank Robotics. Teleoperation with Romeo was shown to a broad audience during demonstrations and tours through the lab. Teleoperation with Pepper was presented at the Pepper World 2017 event. The experiences from these user tests are summarised in Section 5.3. This section focuses on

[1]Try to rotate the hand outside with the upper arm pressed against the body and the elbow at 90°. Then try to perform the same movement with the upper arm loose in the air.

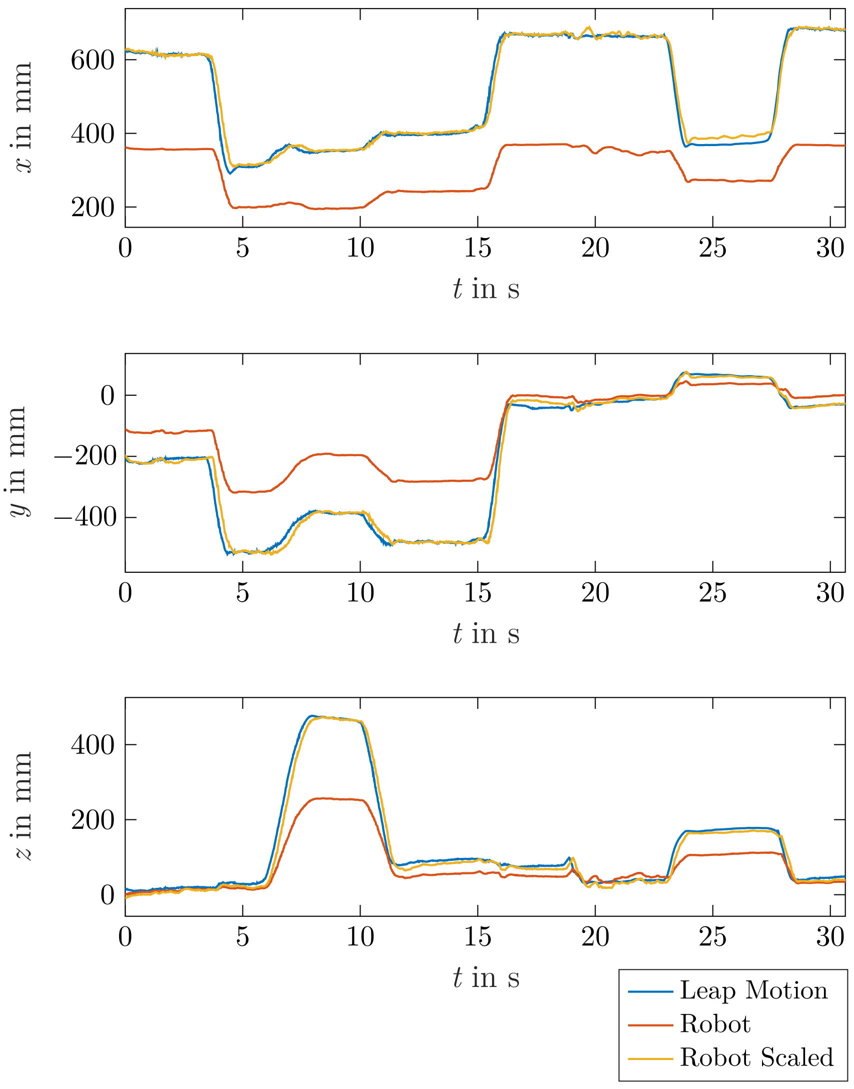


Figure 5.6: Comparison of the tracked palm position, Pepper's palm position and Pepper's scaled palm position during a movement for each coordinate axis

measurements.
The measurements were done with Pepper because Romeo was not available at the time of the experiments. In Figure 5.6 the same recorded sensor data as used for Figure 5.4 was played. Pepper was connected to a wireless network. The teleoperation computer was connected to the same network via Ethernet connection to minimise delay.
The mean Euclidean distance between the position measured by the sensor

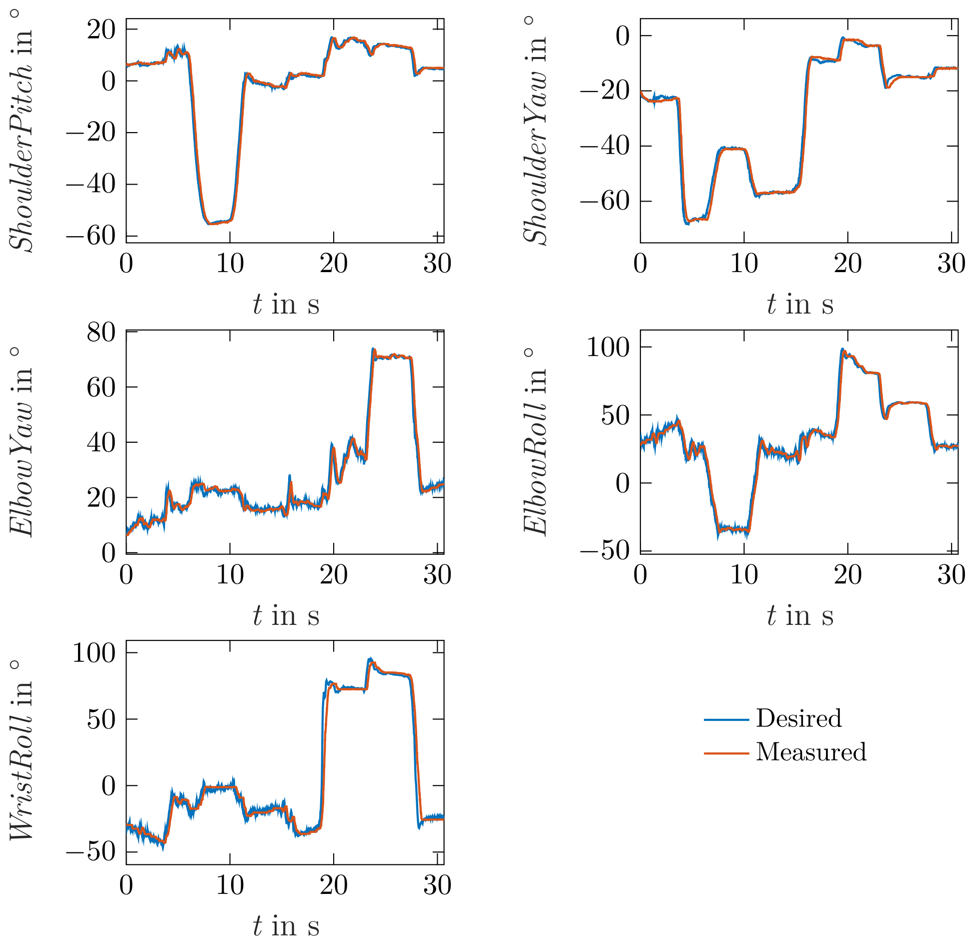


Figure 5.7: Desired and measured joint angles of Pepper during movement

and the scaled robot's position is 21.66 mm with a standard deviation of 21.44 mm. The time delay is estimated by using an approach similar to the method proposed by Viola and Walker [67]. The sample points of the reference signal obtained by the Leap Motion are interpolated by cubic splines for each coordinate axis. The obtained functions are delayed to minimise the mean Euclidean distance to the scaled up version of the measured position of the robot's end-effector. The minimum is reached if the functions are delayed by 150.2 ms. Therefore, this value is the estimated time delay of the whole system including sending the sensor information back from the robot to the computer. The minimum Euclidean distance is 13.93 mm, and the standard deviation equals 8.06 mm after cancelling out the time delay.
The remaining error after cancelling out time delay effects is a result of kinematic

differences between the user and the robot and the restrictions of the robot's motors. As can be seen in Figure 5.7, the *WristRoll* joint is not able to follow the fast movements performed by the user at second 10 and 15. The value of *WristRoll* does not affect the end-effector position. Therefore, the lagging joint does not result in an error of the end-effector position. It is notable that Pepper's *WristRoll* is much faster than the equivalent joint of Romeo (shown in Figure 5.5). Additionally, due to the different range of the joint as mentioned above, the desired joint angles stay in the attainable range.

Pepper misses two degrees of freedom in her wrist in comparison to Romeo. Even though the performed movement was chosen to contain no $WristYaw$ and $WristPitch$ movements, they are not entirely avoidable. These missing degrees of freedom result in an error of the end-effector position. The error at 22 s to 26 s can be explained by this circumstance. The simulated Romeo robot applies $WristYaw \approx 10°$ and $WristPitch \approx 20°$ to follow the movement in this time interval.

For comparison, we used the same recording of the sensor data on a simulated Pepper robot. The time delay in the simulation is 82.5 ms. The mean error is 14.12 mm with a standard deviation of 12.47 mm which decreases to 9.65 mm with a standard deviation of 5.71 mm when cancelling out the time delay. So, the network connection adds ≈70 ms of latency and the real robot kinematics add an error of the palm position. However, the mean error of the simulated Pepper is bigger than for the simulated Romeo due to the two missing degrees of freedom in the wrist.

In this setup, the palm position of the robot was calculated by using forward kinematics with the current sensor readings of the robot's joints. Therefore, errors of the end-effector position resulting from sensor errors of the robot's motors do not show up in the measurements. Errors in the calculation of the elbow joints due to the approximation of the shoulder joints and sensor errors of the Leap Motion cannot be seen in the measurements as well. Consequently, the measurements done in the previous sections are not meant to be seen as an accurate comparison of the position of the user's hand and the robot's hand in real space. It was not a goal to measure the precision of the robot kinematics or the precision of the Leap Motion sensor. However, the experiments show how the robots are able to follow the available sensor readings within their limits and expose specific characteristics like motor restrictions and latency.

5.3 User Study

The teleoperation application was developed from the beginning to be intuitive and comfortable to use. The aim is for a novice user to pick up the Oculus Rift

and start using the application without any previous knowledge of robotics. Effects of cybersickness have to be averted as discussed in Section 3.2.2.
The teleoperation application with Romeo was used with over 100 users in demo and presentation settings. Each user controlled the robot about 1 min to 2 min. No questionnaires were used to receive feedback from the users. Below are qualitative observations. It is planned to do a more thorough user study in the future which is described in Section 5.3.1.

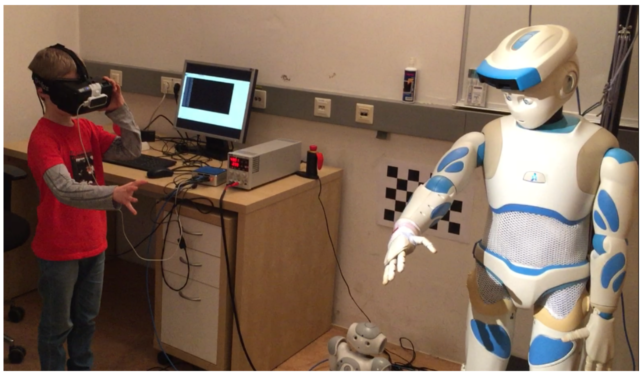


Figure 5.8: Demo setting of teleoperation system with Romeo robot

The demo setting is shown in Figure 5.8. A demo starts with Romeo greeting the audience, doing some gymnastics and talking about his various sensors. The presenter continues to talk about the research goals and explains the teleoperation application. The test users put the virtual reality headset on and control the right arm of the robot. The left arm is not activated to avoid weird movements when the robot tries to avoid self-collision of the arms.
We presented the teleoperation application with Pepper at the "Pepper World 2017" event in Paris. 24 companies showed their applications for Pepper to over 500 visitors and more than 100 SoftBank Robotics employees. The teleoperation system could be tested in a booth. The users controlled both arms in this setup. This was possible because the tablet on Pepper's chest limits the range of the arms in a way, that collision avoidance only has to be performed in edge cases. The feedback for the teleoperation was overwhelmingly positive.
All of the users were able to grab a cup or a piece of cloth, that was handed to them. The users were also able to perform a handshake motion of the robot with another human being. Some users were asked to control the robot to pick something up from a table. This is problematic because the range of the robot in a plane parallel to the floor is very narrow. This is due to the fact that no lean forward or a comparable motion is implemented. The human shoulder joint is also able to have a translational movement additionally to rotation

which is not possible for the robot.
Many of the test users were children. As described in Section 4.1.2, we tuned the calculations for of the robot's joint angles for average male adults. The different dimensions of children result in a shifted origin of the shoulder coordinate system. However, only the shoulder joints are influenced by this circumstance because for the calculation of the other joint angles the direction vectors received by the Leap Motion are used. These direction vectors are independent of the position of the origin. When a child stretches out the arm horizontally in front of it, the robot's upper arm will be slightly tilted upward and to the outside. This behaviour could be avoided, by starting the teleoperation with an initialisation step. The user would have to do a specific pose to calibrate the origin of the shoulder coordinate system. That would be in contrast to the goal for the application to be intuitive. Therefore, no initialisation step is implemented in order to keep the application easy to use. The user receives feedback of the robot's arm position and adjusts his arm accordingly to reach the desired position.
The sensor range is problematic as well. The Leap Motion has a limited field of view and looks for a shape that resembles a hand. So, the hand needs to be at some distance in front of the Leap Motion. For small children, that is often only achievable when stretching their short arms straight forward which limits the possibilities for teleoperation. At the beginning, the users often put their hand right on the sensor or move it outside the field of view. This results in the robot not reacting to the hand movements. After an explanation of the functionality of the Leap Motion, the users learn to look at their hands with a fair distance to the sensor. None of the users noted that they were experiencing symptoms of cybersickness. This might change if the users are exposed to the teleoperation application for a longer period of time. It will be examined in a future user study.

5.3.1 Future User Study

A more thorough user study is planned to be performed in the future. It was not possible to perform it for this thesis, because of the unavailability of the Romeo robot. The focus of the designed user study is on task performance, usability and user experience. A comparable user study was performed by Takayama et al. [68] to evaluate assisted driving of a mobile robot presence system.
The study design is as follows: Before starting the task, the participants have to complete a questionnaire about demographic information and technology usage. The instructor explains how the teleoperation system works, especially focusing on limitations like sensor range or the narrow operational space of the

robot. The participant puts on the Oculus Rift and gets accustomed to the teleoperation application. For training, a plastic cup is handed to the robot, and the participant has to control the hand in order to grab the cup. The task itself is to rearrange three cards on a table in front of the user. This task was chosen because of the poor grabbing performance of Romeo. Cards can be manipulated without closing the hand. An open hand improves the tracking stability of the Leap Motion as well.
As suggested by Steinfeld et al. [69], the workload on the user and human performance is measured by the NASA-Task Load IndeX (NASA-TLX) [70]. The usability and user experience are retrieved with the "IBM Computer Usability Satisfaction Questionnaires" [71].

6 Conclusion

In this thesis, a teleoperation system for humanoid robots was implemented. The capabilities of modern consumer electronics for real-time human upper body imitation were investigated. In this chapter, the strengths and weaknesses of the introduced approach, as well as possible future work, are discussed. It is roughly grouped into topics concerning the imitation of user motion and the user interface which is displayed inside the Oculus Rift.

6.1 Imitation of User Motion

The proposed algorithm for imitating the user's motion shows promising results. An executed trajectory by the user is followed with great precision. The sensor data from the Leap Motion sensor enables a simple algorithm due to the great number of tracked points (e. g. elbow, wrist, palm). The simplicity of the algorithm minimises the time delay which is approximately 150 ms for the whole system, including sending the robot's sensor information back to the computer. Therefore, the time delay of the motion imitation is smaller.
The Leap Motion sensor provides a stable position of the tracked entities in normal lighting conditions. The robot's dynamics act as smoothing. Jitter is hardly noticeable, and additional filtering of the sensor data is not necessary in favour of reducing time delay. For applications that require a more stable position of the robot's hand, the 1€ was implemented and tested. This filter is specialised on filtering human input and is easily tuneable.
The inverse kinematic algorithm was designed to imitate the user's arm posture instead of optimising for a minimum error of the end-effector position. If a precise end-effector position is needed, the human operator will act as a feedback loop and adjust the position of his hand accordingly. The experiments have shown that the scaled up end-effector position is precise despite these design decisions.
The proposed algorithm is not dependent on a specific robot and was tested with the humanoid robots Romeo and Pepper of SoftBank Robotics. To date, there is no easy interface implemented to adapt the algorithm to other robots. Especially, an interface for the Robot Operating System (ROS) would be interesting.

6.2 User Interface

A commercially available virtual reality headset was used to display the robot's camera images and to measure the user's head movements. This technology provides an intuitive interface to control the robot's head orientation and therefore the camera direction. The VR headset should also create an immersive feeling while not causing cybersickness. The problem of different range and speed of the user's and the robot's head motion was solved by displaying a virtual screen with the camera images of the robot in a virtual 3D world. This world would move accordingly to the user's head movements while the screen would move to where the robot is looking.
The small field of view and low refresh rate of the robot's camera are not ideal. This results in feeling like peeking through a window into the robot's world rather than being there in person. Interaction with objects is cumbersome because the hands are easily outside the field of view of the cameras. The low frame rate increases the probability for symptoms of cybersickness. Possible solutions for these problems are presented in Section 6.3.
The manipulation of the camera position is very intuitive, as expected beforehand. An implementation with VR headsets excels at this task. The concept of changing the camera orientation by turning the head did not pose a problem for the test users due to the similarity to what we are used to in everyday life. However, the gaze direction is not tracked inside the VR. Robots with movable eyes (e. g. Romeo, iCub) could benefit from gaze tracking and imitate the user's eye movements. First VR headsets with built-in eye tracking are currently in development[1].
The user interface utilises OpenGL for rendering and is easily portable to comparable VR headsets. The teleoperation system was tested with the Oculus Rift Development Kit 2 and the Oculus Rift Consumer Version 1. The visualisation was tested with a stereoscopic as well as a monoscopic camera system.

6.3 Future Work

In this section, suggestions for further improving the teleoperation system are presented. The system can be enhanced to whole-body imitation by using additional sensors like the Kinect camera to do skeleton tracking. Through sensor fusion, this data can be used to do whole-body imitation comparable

[1]e. g. `https://www.getfove.com`

to [35]. Even with the existing sensors in the system, a forward leaning motion can be implemented, which would enhance the narrow operational space. The optional external camera of the Oculus Rift provides sensor data of the head's translation. Currently, this data is not used in favour of portability. For wheeled robots like Pepper a simple Joystick control for locomotion might be more intuitive and easier to use when wearing the VR headset than the already implemented Keyboard control. Visual feedback when moving might also be interesting and should be considered.
As mentioned above, the immersion is not complete. The visual feedback is sufficient to control the robot but the user does not feel present in the scene. The immersion is dependent on the robot's camera setup. Pepper's cameras provide a monoscopic image with 57.2° horizontal field of view and a frame rate of 30 Hz. Romeo has at least a stereoscopic camera setup which improves the horizontal field of view slightly and adds depth perception. Immersion can be enhanced by using a different camera setup instead of the built in one. For example, wide angle cameras can be mounted on the robot's head. The virtual screen in the user interface would extend and bend to a concave surface. A wide-angle lens attached to the built-in cameras might have a similar effect. Another possibility would be to utilise the built in depth camera to display a point cloud of the surroundings. This could improve depth perception but would increase latency due to additional data transmission and computation time. The problem of the robot's hands being easily outside the field of view of the cameras can be tackled by displaying virtual representations of the tracked user's hands. Imitation of the gaze direction might improve immersion and increase liveliness of the robot. A VR-headset with eye tracking would be required for this endeavour. Another possibility to utilise the robot's movable eyes is to extend the range of the observable area if the neck joints reach their limits comparable to the approach introduced in [45].
To use the system as a telepresence system to communicate with other people a few features need to be added. The audio stream recorded by the robot's microphones has to be transmitted to the user. A possibility for the user to make the robot talk has to be added. The type of implementation is dependent on the use case. In a "Wizard of Oz" setting the robot's voice should be generated by its text-to-speech module, so the robot seems to be autonomous. If the robot is used as a form of communication, it might be advisable to record the user's voice and play it over robot's speakers. An interesting research topic is how the robot can be used to transport emotions.
A future research goal is to generalise the introduced teleoperation system to make it easily portable to other robotic platforms. A wrapper is planned for the Robot Operating System which is an open source software project that provides a structured communication layer between processes on multiple hosts [72].

For other robots (especially non-humanoid robots) different algorithms for the calculation of the joint angles might perform better. Therefore, we will evaluate inverse kinematic algorithms based on local optimisation of the error of the end-effector position.
Another ongoing research project is for the robot to learn action concepts from observing a human tutor[2]. The here introduced teleoperation system is used to track hands and transform the movements to the robot. It is coupled with an object tracker which tracks the objects the user interacts with. A natural language processing engine is used to analyse the verbal description of the performed action. The robot shall learn from the accumulated data to perform new tasks. A research question is if this kind of showing and explaining actions comparable to how we teach children is suitable for robots to acquire new actions.

[2]`http://ralli.ofai.at`

Bibliography

[1] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2016.

[2] L. D. Riek, „Wizard of Oz studies in HRI: A systematic review and new reporting guidelines,“ *Journal of Human-Robot Interaction*, vol. 1, no. 1, 2012.

[3] T. B. Sheridan, „Telerobotics,“ *Automatica*, vol. 25, no. 4, pp. 487–507, 1989.

[4] R. C. Goertz and W. M. Thompson, „Electronically controlled manipulator,“ *Nucleonics (US) Ceased publication*, vol. 12, 1954.

[5] R. S. Mosher, „Industrial manipulators,“ *Scientific American*, vol. 211, pp. 88–95, 1964.

[6] K. J. Zuo and J. L. Olson, „The evolution of functional hand replacement: From iron prostheses to hand transplantation,“ *Plastic Surgery*, vol. 22, no. 1, p. 44, 2014.

[7] I. E. Sutherland, „A head-mounted three dimensional display,“ in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, ACM, 1968, pp. 757–764.

[8] S. Kassel, „Lunokhod-1 soviet lunar surface vehicle,“ DTIC Document, Tech. Rep., 1971.

[9] T. B. Sheridan, „Teleoperation, telerobotics and telepresence: A progress report,“ *Control Engineering Practice*, vol. 3, no. 2, pp. 205–214, 1995.

[10] T. Fong, I. Nourbakhsh, and K. Dautenhahn, „A survey of socially interactive robots,“ *Robotics and autonomous systems*, vol. 42, no. 3, pp. 143–166, 2003.

[11] O. E. Ramos Ponce, „Generation of whole-body motion for humanoid robots with the complete dynamics,“ PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2014.

[12] K. Bouyarmane, J. Vaillant, F. Keith, and A. Kheddar, „Exploring humanoid robots locomotion capabilities in virtual disaster response scenarios,“ in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, IEEE, 2012, pp. 337–342.

[13] J. Fink, „Anthropomorphism and human likeness in the design of robots and human-robot interaction,“ in *International Conference on Social Robotics*, Springer, 2012, pp. 199–208.

[14] C. Breazeal, „Emotion and sociable humanoid robots,“ *International Journal of Human-Computer Studies*, vol. 59, no. 1, pp. 119–155, 2003.

[15] A. Waytz, J. Heafner, and N. Epley, „The mind in the machine: Anthropomorphism increases trust in an autonomous vehicle,“ *Journal of Experimental Social Psychology*, vol. 52, pp. 113–117, 2014.

[16] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, „A survey of robot learning from demonstration,“ *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[17] S. Schaal *et al.*, „Learning from demonstration,“ *Advances in neural information processing systems*, pp. 1040–1046, 1997.

[18] R. O. Ambrose, H. Aldridge, R. S. Askew, R. R. Burridge, W. Bluethmann, M. Diftler, C. Lovchik, D. Magruder, and F. Rehnmark, „Robonaut: NASA’s space humanoid,“ *IEEE Intelligent Systems*, vol. 15, no. 4, pp. 57–63, 2000.

[19] W. Bluethmann, R. Ambrose, M. Diftler, S. Askew, E. Huber, M. Goza, F. Rehnmark, C. Lovchik, and D. Magruder, „Robonaut: A robot designed to work with humans in space,“ *Autonomous robots*, vol. 14, no. 2-3, pp. 179–197, 2003.

[20] J. Badger, M. Diftler, S. Hart, and C. Joyce, „Advancing robotic control for space exploration using Robonaut 2,“ 2012.

[21] S. Goza, R. O. Ambrose, M. A. Diftler, and I. M. Spain, „Telepresence control of the NASA/DARPA robonaut on a mobility platform,“ in *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 2004, pp. 623–629.

[22] G. Martinez, I. A. Kakadiaris, and D. Magruder, „Teleoperating Robonaut: A case study.,“ in *BMVC*, 2002, pp. 1–10.

[23] R. P. Paul, *Robot manipulators: Mathematics, programming, and control: The computer control of robot manipulators.* Richard Paul, 1981.

[24] J Baillieul and D. Martin, „Resolution of kinematic redundancy,“ in *Proceedings of Symposia in Applied Mathematics*, vol. 41, 1990, pp. 49–89.

[25] D. E. Whitney, „Resolved motion rate control of manipulators and human prostheses.,“ *IEEE Transactions on man-machine systems*, 1969.

[26] M. Riley, A. Ude, K. Wade, and C. G. Atkeson, „Enabling real-time full-body imitation: A natural way of transferring human movement to humanoids,“ in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, IEEE, vol. 2, 2003, pp. 2368–2374.

[27] D. Matsui, T. Minato, K. F. MacDorman, and H. Ishiguro, „Generating natural motion in an android by mapping human motion,“ in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2005, pp. 3301–3308.

[28] T. B. Moeslund, A. Hilton, and V. Krüger, „A survey of advances in vision-based human motion capture and analysis,“ *Computer vision and image understanding*, vol. 104, no. 2, pp. 90–126, 2006.

[29] N. S. Pollard, J. K. Hodgins, M. J. Riley, and C. G. Atkeson, „Adapting human motion for the control of a humanoid robot,“ in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, IEEE, vol. 2, 2002, pp. 1390–1397.

[30] M. Do, P. Azad, T. Asfour, and R. Dillmann, „Imitation of human motion on a humanoid robot using non-linear optimization,“ in *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2008, pp. 545–552.

[31] P. Azad, T. Asfour, and R. Dillmann, „Robust real-time stereo-based markerless human motion capture,“ in *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2008, pp. 700–707.

[32] Z. Zhang, „Microsoft Kinect sensor and its effect,“ *IEEE multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[33] H. B. Suay and S. Chernova, „Humanoid robot control using depth camera,“ in *2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, IEEE, 2011, pp. 401–401.

[34] F. Zuher and R. Romero, „Recognition of human motions for imitation and control of a humanoid robot,“ in *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian*, IEEE, 2012, pp. 190–195.

[35] Y. Ou, J. Hu, Z. Wang, Y. Fu, X. Wu, and X. Li, „A real-time human imitation system using Kinect,“ *International Journal of Social Robotics*, vol. 7, no. 5, pp. 587–600, 2015.

[36] K. Khoshelham and S. O. Elberink, „Accuracy and resolution of Kinect depth data for indoor mapping applications,“ *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[37] L. Yang, L. Zhang, H. Dong, A. Alelaiwi, and A. El Saddik, „Evaluating and improving the depth accuracy of Kinect for Windows v2,“ *IEEE Sensors Journal*, vol. 15, no. 8, pp. 4275–4285, 2015.

[38] D Bassily, C Georgoulas, J Guettler, T Linner, and T Bock, „Intuitive and adaptive robotic arm manipulation using the Leap Motion controller,“ in *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, VDE, 2014, pp. 1–7.

[39] N. Yu, C. Xu, K. Wang, Z. Yang, and J. Liu, „Gesture-based telemanipulation of a humanoid robot for home service tasks,“ in *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015 IEEE International Conference on*, IEEE, 2015, pp. 1923–1927.

[40] D. Roetenberg, H. Luinge, and P. Slycke, „Xsens MVN: Full 6DOF human motion tracking using miniature inertial sensors,“ *Xsens Motion Technologies BV, Tech. Rep*, 2009.

[41] J. Koenemann, F. Burget, and M. Bennewitz, „Real-time imitation of human whole-body motions by humanoids,“ in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 2806–2812.

[42] C. Stanton, A. Bogdanovych, and E. Ratanasena, „Teleoperation of a humanoid robot using full-body motion capture, example movements, and machine learning,“ in *Proc. Australasian Conference on Robotics and Automation*, 2012.

[43] S. Nakaoka, A. Nakazawa, K. Yokoi, H. Hirukawa, and K. Ikeuchi, „Generating whole body motions for a biped humanoid robot from captured human dances,“ in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, IEEE, vol. 3, 2003, pp. 3905–3910.

[44] A. P. Shon, J. J. Storz, and R. P. Rao, „Towards a real-time Bayesian imitation system for a humanoid robot,“ in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 2847–2852.

[45] L. Fritsche, F. Unverzag, J. Peters, and R. Calandra, „First-person teleoperation of a humanoid robot,“ in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, IEEE, 2015, pp. 997–1002.

[46] M. Tomić, C. Vassallo, C. Chevallereau, A. Rodić, and V. Potkonjak, „Arm motions of a humanoid inspired by human motion,“ in *New Trends in Medical and Service Robots*, Springer, 2016, pp. 227–238.

[47] J. Guna, G. Jakus, M. Pogačnik, S. Tomažič, and J. Sodnik, „An analysis of the precision and reliability of the Leap Motion sensor and its suitability for static and dynamic tracking,“ *Sensors*, vol. 14, no. 2, pp. 3702–3720, 2014.

[48] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, „Analysis of the accuracy and robustness of the Leap Motion controller,“ *Sensors*, vol. 13, no. 5, pp. 6380–6393, 2013.

[49] Q. Wang, G. Kurillo, F. Ofli, and R. Bajcsy, „Evaluation of pose tracking accuracy in the first and second generations of Microsoft Kinect,“ in *Healthcare Informatics (ICHI), 2015 International Conference on*, IEEE, 2015, pp. 380–389.

[50] B. A. Davis, K. Bryla, and P. A. Benton, *Oculus Rift in Action.* 2015.

[51] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov, „Head tracking for the Oculus Rift,“ in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 187–194.

[52] I. Goradia, J. Doshi, and L. Kurup, „A review paper on Oculus Rift & Project Morpheus,“ *International Journal of Current Engineering and Technology*, vol. 4, no. 5, pp. 3196–3200, 2014.

[53] J. J. LaViola Jr, „A discussion of cybersickness in virtual environments,“ *ACM SIGCHI Bulletin*, vol. 32, no. 1, pp. 47–56, 2000.

[54] S. Davis, K. Nesbitt, and E. Nalivaiko, „A systematic review of cybersickness,“ in *Proceedings of the 2014 Conference on Interactive Entertainment*, ACM, 2014, pp. 1–9.

[55] R. Yao, T. Heath, A. Davies, T. Forsyth, N. Mitchell, and P. Hoberman, „Oculus VR best practices guide,“ *Oculus VR*, 2014.

[56] N. Pateromichelakis, A. Mazel, M. Hache, T Koumpogiannis, R. Gelin, B. Maisonnier, and A. Berthoz, „Head-eyes system and gaze analysis of the humanoid robot Romeo,“ in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 1374–1379.

[57] I. Kapandji, *The Physiology of the Joints: Annotated Diagrams of the Mechanics of the Human Joints*, ser. The Physiology of the Joints: Annotated Diagrams of the Mechanics of the Human Joints. Churchill Livingstone, 1982, ISBN: 9780443025044. [Online]. Available: `https://books.google.at/books?id=-zYCjwEACAAJ`.

[58] T. Kurz, *Stretching scientifically: A guide to flexibility training.* Stadion Publishing Company, Inc., 2003.

[59] M. Böck and A. Kugi, „Fachvertiefung: Automatisierungs- und Regelungstechnik,“ Automation and Control Institute, TU Wien, Lecture Notes, 2016.

[60] J. J. Craig, *Introduction to robotics: Mechanics and control.* Pearson Prentice Hall Upper Saddle River, 2005, vol. 3.

[61] R. N. Jazar, *Theory of applied robotics: Kinematics, dynamics, and control.* Springer Science & Business Media, 2010.

[62] H. W. Jürgens, *Erhebung anthropometrischer Maße zur Aktualisierung der DIN 33 402-Teil 2.* Wirtschaftsverl. NW, Verlag für Neue Wiss., 2004.

[63] G. Casiez, N. Roussel, and D. Vogel, „1 filter: A simple speed-based low-pass filter for noisy input in interactive systems,“ in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2012, pp. 2527–2530.

[64] D. Shreiner, G. Sellers, J. M. Kessenich, and B. Licea-Kane, *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3.* Addison-Wesley, 2013.

[65] B. Stroustrup, „The C++ programming language,“ 2013.

[66] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier, „Choregraphe: A graphical tool for humanoid robot programming,“ in *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, IEEE, 2009, pp. 46–51.

[67] F. Viola and W. F. Walker, „A spline-based algorithm for continuous time-delay estimation using sampled data,“ *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 52, no. 1, pp. 80–93, 2005.

[68] L. Takayama, E. Marder-Eppstein, H. Harris, and J. M. Beer, „Assisted driving of a mobile remote presence system: System design and controlled user evaluation,“ in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 1883–1889.

[69] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M. Goodrich, „Common metrics for human-robot interaction,“ in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, ACM, 2006, pp. 33–40.

[70] S. G. Hart and L. E. Staveland, „Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research,“ *Advances in psychology*, vol. 52, pp. 139–183, 1988.

[71] J. R. Lewis, „IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use," *International Journal of Human-Computer Interaction*, vol. 7, no. 1, pp. 57–78, 1995.

[72] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, „ROS: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, vol. 3, 2009, p. 5.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.
Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 8. Juni 2017

Matthias Hirschmanner