Graduation Project 2021

Upper body teleoperation of a humanoid robot (avatar)

Department of Electrical Engineering, Mathematics and Computer Science (EEMCS)

Chair of Robotics and Mechatronics

Bachelor of Science, Creative Technology

University of Twente

Veronique Kochetov

s2168405

Client: i-Botics

Supervisor: Dr. Ir. Douwe Dresscher

Critical observer: Dr. Ing. Gwenn Englebienne

August, 2021

i-BOTICS
an initiative of TNO and University of Twente

UNIVERSITY OF TWENTE.

Abstract

As part of the ANA XPRIZE competition, the i-Botics group develops a telerobotic system to operate the humanoid robot EVE and create the feeling for the human operator of being present in a different environment. Telerobotic systems enable controlling robotics remotely and transferring the human operator's physical and mental capabilities to a geographically different location. This graduation project aims to develop the possibility of controlling the upper body posture of the humanoid robot over distance and in real-time. The project also includes research into communicating information in social contexts in relation to the upper body posture.

After background research and analysis of suitable motion capture hardware, the robot's motion capabilities, different motion mapping strategies, and possibilities of conveying human-like and social behavior in Telerobotics, concepts were developed and realized. In order to map the motion, a data-driven approach using Adaptive Neuro-Fuzzy interference systems (ANFIS) and direct angle mapping using the rotational position of the human chest applied to the hip joints of the robot were developed and assessed. An Xsens suit was used for motion capture. Literature research has shown that a valuable addition to conveying human-like behavior in Telerobotics are secondary actions, such as breathing, which is created and combined with the motion mapping algorithm.

The performance of the two motion mapping algorithms is compared based on simulation results and plots of human orientation vs. produced robot orientation. It can be concluded that the direct angle mapping approach involves less complexity and also performs slightly better than the ANFIS approach. There is, however, no visual difference. The performance of the breathing animation is tested in integration with the motion mapping algorithm and shows decent performance but has the pitfall of blocking real-time motion mapping.

As future work, it is particularly suggested to conduct user tests to evaluate the effectiveness of the breathing animation and integrate the developed motion mapping into the existing system of the i-Botics group.

Acknowledgments

I would like to thank a few people and express my appreciation for their support and guidance throughout this project. First, I would like to thank my supervisor Douwe Dresscher, who showed much support since the start of the graduation semester and gave me valuable feedback for my thesis. Moreover, I am thankful for the help and feedback from Robin Lieftink during the practical part of the project, who assisted me as an expert on the telerobotic system by i-Botics. In addition, I would like to thank my critical observer, Gwenn Englebienne, for the valuable insights into his current research regarding the humanoid robot in the social context. I am also grateful to Johnny Lammers van Toorenburg from the BMS lab of the University of Twente for providing me with the necessary equipment. Finally, I would also like to thank my friends and family for supporting me during this graduation semester.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

Telerobotics is a relevant and emerging field of robotics, where a robot can be operated over distance by a human. Telepresence describes the situation of truly feeling present at the robot's location to interact with the environment. Given this possibility, robots can be beneficially used in many areas, such as healthcare, military, or disaster relief.

The project is part of the ANA Avatar XPRIZE competition [13], where the goal is to develop a telerobotic system that allows you to transfer your physical and mental capabilities to a different location, as well as that creates an experience of physically being at that location to interact with the environment and other people. i-Botics is an open innovation Centre for research and development in the Netherlands, founded by TNO and the University of Twente, and participates in this competition.

When teleoperating a humanoid robot, the goal is to recreate the motions performed by the human operator as well as possible. The operator's body posture has to be captured and translated in a meaningful way since it is a relevant part of communicating information in social contexts and, e.g., showing human emotion when being present as a robotic avatar. The current system developed by i-Botics does not provide that feature yet and therefore has to be extended.

1.2 Problem definition

The telerobotic system by i-Botics includes the humanoid robot Halodi EVE[2]. It is currently possible to control the location of the robot’s hands and receive haptic feedback if the robot touches any object. Additionally, the robot's locomotion using the wheels is controllable with a device, which is part of the whole telerobotic system.

The focus and goal of this assignment is to find a way to teleoperate the upper-body posture of the EVE in real-time, which can be integrated into to existing system. The teleoperation of the upper body should be done by using the motion and position data of a human and reach an equivalent position by the robot. In order to reach this goal, sub-problems need to be solved and answered.

First, a suitable motion capture system has to be chosen to track the operator's upper body pose, keeping the availability and requirements of the hardware in mind. Additionally, a suitable method for motion mapping from the human motion to the motion of the humanoid robot EVE has to be found. Since the human body can move in many more ways than the robot, restrictions have to be identified, and solutions have to be found to map the motion successfully. Furthermore, an algorithm has to be developed to fuse the hand motion control with the upper body motion control. That way, both systems can work in combination and create proper reference positions for the robotic avatar.

Another goal of this project is to find effective ways to convey social and human-like behavior, e.g., emotions, given the physical limitations of the humanoid robot. Thus, possible adjustments or extensions to the motion mapping algorithm should be explored and implemented.

1.3 Research questions

Given the problem definition, the following research questions will be answered:

- Which body posture capturing hardware is most suitable given the requirements and availability for the existing controller system?
- How can the body posture of a human operator be efficiently mapped using an algorithm to translate motion to a humanoid robot?
- How can the upper body capture mapping algorithm be integrated into the existing control system?
- How can the chosen algorithm be adjusted in such a way that the humanoid robot can convey social behavior and gestures of the operator?

1.4 Approach and practical aspects

Regarding the realization of this project, certain practical aspects have to be taken into account. For instance, the hardware selection is bound to the availability of options provided by the Robotics and Mechatronics chair of the University of Twente. Moreover, testing of the implementation will mainly be done with simulation software. Due to the current situation regarding COVID-19, it is unsure if physical testing will be possible during the time frame of this project. It was announced that the robot EVE might become physically available on the university campus, allowing the conduction of physical user tests.

As part of implementing an algorithm, a kinematic model of a human and the robot should be explored to analyze the connection and restrictions of different body parts. Furthermore, potential hardware systems will be compared and chosen to capture motion data. Telerobotic systems described in literature will be used to explore different methods of how motion capture data can be translated to the right motion commands for the robot. Based on the chosen hardware, motion mapping strategy, and selected social behavior cues, the system can be implemented and evaluated afterward.

1.5 Report structure

Given the guidelines provided by the Robotics and Mechatronics faculty[14] and the Creative Technology design process by Mader et al.[15], the report will be oriented towards four phases: ideation, specification, realization, and evaluation. The ideation phase is about developing a concept for the practical realization and will be combined in a chapter together with the background research. Since the project consists of multiple parts, every partial concept follows after the required background information. The specification and realization are described in the implementation chapter, which specifies the created concepts and shows how they are realized. Finally, the implementation will be tested in the evaluation phase based on set requirements and test criteria. In conclusion, the research questions defined in section 1.3 will be answered, and suggestions for future work will be provided.

2 Background research and concept development

Within this chapter, the current situation of the telerobotic system by i-Botics will be explained, so that system requirements for the implementation of this project can be developed. After the stakeholders and system requirements are defined, available motion capture hardware, robot, and human kinematics, and existing motion mapping techniques will be described and analyzed. After the evaluation of the background research, the concept for this project will progressively be developed.

2.1 The current system and architecture

Telerobotic systems consist of various components, including hardware and software needed for human motion capturing, motion mapping, and visual, audible, haptic, or thermal feedback. The human operator is in a room, the cockpit, where the teleoperation takes place. The cockpit and used robot of the current system by i-Botics is provided in figure one and two.

The possibilities of movement of the human operator are limited due to the high chair the person leans on. The feet are placed on a locomotion plate to control the robot's wheels, so the rotation and movement forward and backward. Besides, the hands are attached to a system called virtuose by Haption[16] that captures the hand locations. This way, the position of the arms is already remotely controllable. H-gloves by Haption[17] provide haptic feedback by means of force. This way, the operator can feel if they encounter resistance by touching other objects. For instance, the operator can estimate the necessary pull force and open the oven easier when opening a heavy oven door. Additionally, the room involves multiple heaters for thermal feedback if the robot encounters high temperatures[1].

Figure 1: The current system: cockpit[1]

Figure 2: The humanoid robot: Halodi EVE[2]

Moreover, the operator wears a Head Mounted Display (HMD), enabling visual feedback and an immersive experience in which the user should feel as if they are placed inside the robot's body. The view of the user consists of a 3D representation of the remote environment, a direct stereo vision to observe interactions with the hands and environment and a virtual reality model of the avatar for self-view. In addition, sensors inside the HMD track the facial expressions of the operator, specifically the eye and mouth movement, which are mapped to EVE's face in animated form to create more natural interactions[1].

There are two PCs involved that run on Windows and Ubuntu. One is for the vision feature, and the other one is for control. The communication between the robot and PC is handled via ROS[18].

The humanoid robot involved is developed by Halodi and called EVE. EVE is 1.83 meters high and weighs 76 kg [2]. As shown in figure 2, EVE does not have two separate legs, but just one that can move down and back up. The robot drives from one place to the other using wheels. The upper body has more movement possibilities, such as movements of the hips in x-, y-, and z-direction, the up and down movement of the hand, and the arm and hand motion. It will be elaborated more on the motion capabilities of EVE in a later section. Since safety is also a relevant factor in teleoperation, emergency stop buttons are integrated into the back of the humanoid robot and wireless emergency stop buttons that can shut down the EVE remotely[1].

The new feature introduced within this project is the possibility of the human operator to control the upper body posture beside the head and arms. It will also be discussed which social impact the upper body posture has on the environment with which the robot is interacting.

2.2 User identification and stakeholders

Several parties can be identified as stakeholders. i-Botics is in possession of the cockpit and control interface to operate the robot, which means mainly i-Botics members will use the system and further develop it. Since this project is being developed in the context of the XPRIZE competition, the involved jury will rate the system based on certain requirements and functionalities. The robot itself is currently stationed at the headquarters of Halodi robotics in Norway. Therefore, the people that are physically interacting with the robot are members of Halodi Robotics. With future perspective, the University of Twente and other organizations could purchase and physically interact with this robot as well. All stakeholders are familiar with the subject of robotics and share equivalent interests regarding the system. The telerobotic system should be intuitive, the usage should be easy to learn, and next to the goal of immersion of the human operator, the system should copy and perform motion hints at autonomy.

2.3 System requirements

Since there is already an existing system, which has to be built upon, the new system should meet a few requirements. It has to be noted that many components are already involved that the human operator has to wear on the body, such as the head mounted display, the H-gloves by Haption and the attached virtuoses, and the locomotion plate. The system should be kept as non-invasive as possible, which means another motion capture hardware should possibly not interfere with other hardware or should capture the motion visually and from the outside. The operator should be free to move within the setup, not being obstructed by motion capturing hardware. Besides, aspects like the setup time of the system and ease of use are relevant as well. Costs are not relevant since there is motion capture hardware available to use.

In addition to that, the motion mapping should be accurate enough to reproduce the operator's posture naturally. The upper body posture should be controlled in terms of the robot's torso, e.g., hips and shoulder position. The position of the hands and the connected motion of the arms is already controlled in the current system. The new part of the system should be developed in such a way that it can be connected to the existing system. Besides, the motion commands should be computed and sent in real-time so that the motion of the human operator compared to the Eve has as little lag as possible.

2.4 Motion capture hardware

To be able to control a humanoid robot remotely, it is necessary to capture the motion, e.g., position and orientation coordinates of a human, which will be the input to the motion mapping algorithm. Regarding the availability of hardware, the Microsoft Kinect and the Xsens MVN motion capture suit can be utilized. Both of these systems are also widely used among many other telerobotic solutions. For this reason, both systems will be described and compared to draw a conclusion on which hardware will be used for this project.

2.4.1 Microsoft Kinect

The Kinect can be used as motion capture hardware based on visuals. It is a depth camera that recognizes the environment in 3D and can create a skeleton image of a person. Initially, the Kinect was developed to play video games, however, it is widely used among developers for different kinds of projects[19]. Therefore, there is a lot of different software available to interface the Kinect and extract motion data. The Kinect software is capable of automatically calibrating the sensor based on the person's physical environment and can trace up to 20 joints of the body with respect to its coordinate system[20] [21]. The Kinect provides the position and orientation of the joints. A disadvantage is that the Kinect is not as accurate as body suits, such as the Xsens motion capture suit, since it relies on vision, which can be obstructed by other body parts or objects[22].

2.4.2 Xsens MVN

The Xsens MVN motion capture suit is easy to use and comes with its own software. It is based on inertial sensors and wireless communication with the software that applies advanced sensor fusion algorithms. Besides the 17 small sensors that can be worn and only nine sensors to track the upper body without hands, there is no need for external cameras or markers. Thus, there are no restrictions regarding visual obstructions, such as objects or poor light conditions. There are a lot of different options regarding the data to be tracked. The translational position, the orientation in x-, y-, and z-direction, and even velocity and acceleration of different body parts can be obtained with respect to a global origin.[5] One disadvantage might be that the setup is more time-consuming than the Kinect since all sensors need to be properly attached to the body. For each use, the suit needs to be manually calibrated within the software application. Another aspect is invasiveness. In contrast to the Kinect, the user needs to wear the sensors on the body, which could be problematic if the system it is used for already involves much other invasive hardware. Nevertheless, the Xsens provides the best performance compared to the Kinect and other motion capture hardware[23].

2.4.3 Conclusion and concept development

On the one hand, the Kinect assures the significant advantage of non-invasiveness, since this is a system requirement of this project. On the other hand, the current telerobotic system involves much hardware, increasing the risk of visual obstructions so that the Kinect might produce more inaccurate results. The Kinect has to be placed at a certain distance to the human operator to capture the needed body parts, and at the same time, the vision has to be clear. No other person is allowed to walk in between and interfere with the system. More accuracy is provided by the Xsens suit, which lacks non-invasiveness but overcomes all pitfalls that the Kinect has.

Since both systems show certain benefits and drawbacks, both should be tested in practice to see how they perform. However, due to time constraints only the Xsens motion capture suit will be used and evaluated in this project. In future projects, the implementation with the Xsens

suit could be compared with the performance of the Kinect in order to conclude which hardware is the better choice.

2.5 Robot kinematics, human kinematics, and the EVE

In order to map the motion of a human to a robot, it is essential to analyze the kinematics of both, human and robot to find similarities and differences. When speaking about kinematics, it is meant to determine where and what kind of joints connect the different body parts and what movements are possible or restricted. A kinematic model of a robot will show the capabilities and limitations of motion. The human body significantly differs from the body of a robot in terms of size and motion capabilities. Therefore there are certain problems and challenges to be encountered in the procedure of motion mapping. Topics and terms such as degrees of freedom, joint angle limits, and workspace will be introduced.

2.5.1 Degrees of freedom

Degrees of freedom (DOF) in robotics and kinematics typically refer to the possibilities of motion of body parts. A joint connects two links/ body parts and thus limits the number of degrees of freedom between them[24]. Where a link can move freely in all directions without any other joints and links attached, it might be able to move in only in a limited amount of directions if certain joints constrain it. There are various kinds of joints to be found(Figure 3), such as revolute, prismatic, or spherical joints, which all provide different options of motion. Revolute and prismatic joints allow the movement in a single axis and, therefore, one degree of freedom. Spherical joints, sometimes referred to as ball joints, provide 3 degrees of freedom, which means rotation around the x-, y- and the z-axis is possible.[3]

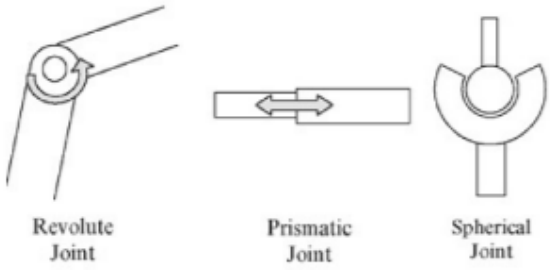


Figure 3: different types of joints[3]

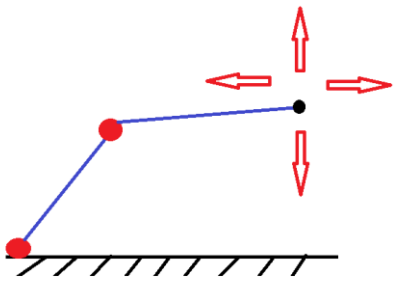


Figure 4: 2 DOF system with 2 joints & 2 links

Depending on how many and in which way links are connected in a chain, the degrees of freedom of the overall robotic system can increase or decrease. According to [25], the degrees of freedom of a mechanism is defined as “the number of coordinates or variables required to be specified such that the position and orientation of all the members of the mechanism can be stated as a function of time.” As an illustrating example, figure 4 shows two links (N =3, including ground) where each is connected to one joint (J = 2). Without joints, each link moves with three degrees of freedom in a 2D space (m = 3/ m=6, if 3D space). If the joints are chosen to be revolute (2 constraints per joint), and the mechanism is moving in a 2D plane, Grübler’s formula[26] states that this system has m * (N -1) – constraints = 3 * 2 - 4 = 2 degrees of freedom.

The humanoid robot EVE has 23 degrees of freedom and 24 motors to control different body parts (excluding hands)[2]. Figure 6 shows the human model imitated by the Xsens system. There are 23 body segments and 22 joints specified with six degrees of freedom for each joint[5]. Figure 5 shows a simplified kinematic model of a human, where the cylinders represent revolute joints and the balls represent spherical joints, where movement around all axis is possible. The simplified human model has "24 articulated rotational DOF in total, with addition of the three rotational DOF and three translational DOF of the pelvic segment which define the position and orientation of the body with regard to the reference coordinate system"[4], which means the EVE is quite similar to the simplified human model in terms of degrees of freedom. In reality, a human body has a lot more degrees of freedom, if, for instance, the spine, fingers, and toes are included as well.

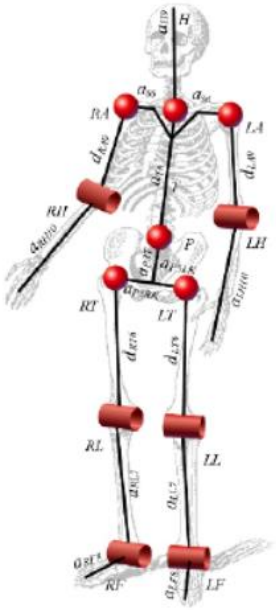


Figure 5: Simplified human kinematics model[4]

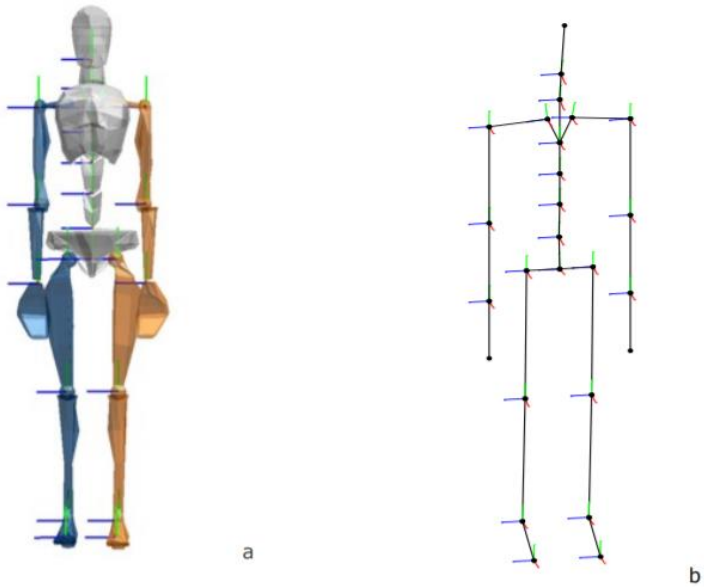


Figure 6: Human body modeled by Xsens[5]

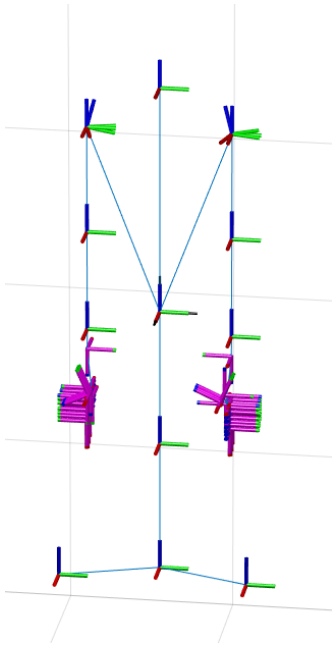


Figure 7: EVE model with frames assigned to links

Figure 8: Motion capabilities of the EVE[2]

When focusing on the upper body only, certain differences between the human body and the EVE can be noticed. More frames are assigned to the Xsens human body, especially along the spine and neck, reflecting that the human body has more body links and provides more flexibility in movement than the EVE (Figure 7).

According to figure 8 and the unified robot description format (URDF) model of the robot[27] (a file format that is, for example, used in ROS to describe all elements, dimensions, and movement capabilities of a robot[28]), there are three joints at the hip, for the x-, y-, and z-direction. They are not visible in figure 7 because they lay on top of each other, which is an equivalent attribute to the human body. It can be seen in figure 5 that the hips have a spherical joint allowing the movement in x-, y-, and z-direction of the upper body. There are also two spherical joints for the motion of legs, but which the EVE does not have. A movement of the shoulders in all three directions is, just like for a human, also possible for the EVE. Besides, the arms of a human and the EVE provide a very similar structure, where equivalent rotations are possible. However, the head of EVE can only move up and down, whereas a human can also move the head in left and right. As already mentioned, a human is capable of rotating spine segments without changing the position of the pelvis, though it influences the location of the shoulders. Consequently, this specific movement is not possible for the EVE and it has to be taken into account that the hips are not the only factor that can change the shoulder position and the position of the entire torso.

2.5.2 Joint angle limits and work space

Joint angle limits refer to the maximum allowed rotation by the joints of the robot and can be defined in the context of joint or configuration space[29]. The joint configuration is the joint angles for all joints. The configuration space is the space of all possible configurations. Differences between the joint/configuration space of a human and robot can be encountered. According to figure 8, the maximum allowed joint angles are shown for the EVE. Within the URDF model of the robot[27], some joint angle limits are narrowed even more, which might be due to safety reasons or the illustration pictures an older version of the EVE. One example of the differences in joint angle limits is in the hip joints. For instance, according to the URDF model, the movement in the y-direction, which is the motion to the front and back of the upper body, is limited to 10 degrees to the front and 90 degrees to the back (Figures 9,10,11).

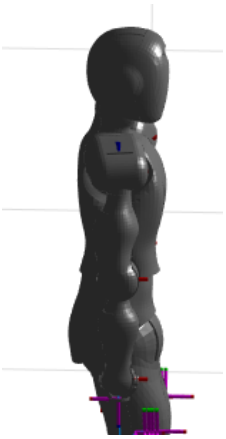


Figure 9: EVE standing straight

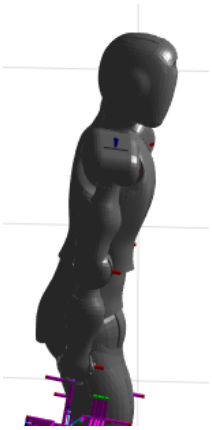


Figure 10: EVE leaned forward

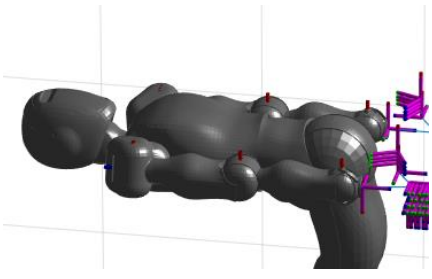


Figure 11: EVE leaned backward

One explanation of these limits could be for stability reasons of the robot. According to Figure 8, the EVE has a small support leg behind the wheels, which allows putting much weight on the back

side of the robot. If the movement to the front of the robot is too far, the robot could fall over. Figure 12 shows the general motion limitations of a human upper body when leaning forward and backward. In contrast to the robot, the human can move the upper body about 90 degrees to the front since the front part of the feet similarly provide stability as the support leg of the EVE. Depending on how much the individual is trained, the joint angle limit of the movement to the back is more restricted. In general, it is kinematically impossible to reach a 90-degree position of the back with respect to the legs.

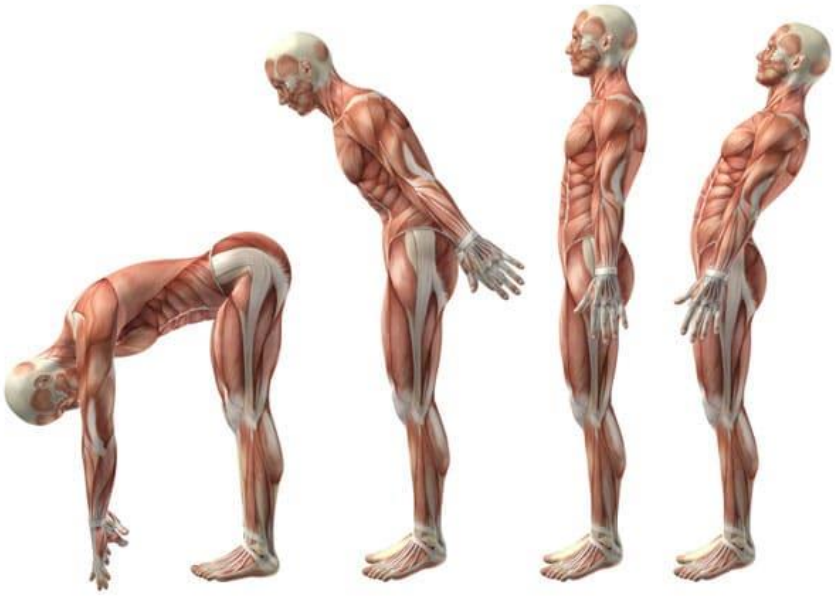


Figure 12: Human motion capabilities for flexion and extension of the upper body[6]

The workspace, also called the task space, refers to all possible positions and orientations of the end-effector (the body link to be controlled)[24]. Since the joint space differs between a human and robot, the workspace will show differences as well. Besides, the size difference between the human body and robot body plays a relevant role as well. A robot with arms that are half the size of a human arm will never reach the same translational position in space. Therefore a rescaling is required if the translational position is taken as the goal position for the robot. Methods for scaling and adjustments of the workspace are later described in the sections of chapter 2.4.

Regarding the joint space and joint angle limits, certain movements that the human operator performs will not be possible with the EVE, e.g., 90 degrees rotation to the front. Therefore, the only solution is to let the robot hit the joint angle limits if the human moves out of the range. Scaling the movement of the robot down could result in a minimally visible change of movement during the whole procedure of teleoperation, which is not desired for the project.

2.5.3 Forward and inverse kinematics

According to Aristidou et al. [30], kinematics is the translational and rotational motion of points, bodies, and groups of bodies without considering any reference to mass, force, or torque. Forward kinematics is defined as "the problem of locating the end effector positions after applying known transformations to the chain" [30]. The joint angles and link lengths are given. On the contrary, inverse kinematics is "the problem of determining an appropriate joint configuration for which the end effectors move to desired target positions, as rapidly and accurately as possible" [30] (Figure 13). In contrast to forward kinematics, inverse kinematics does not solely have a unique solution, but either multiple, a unique, or no solutions.

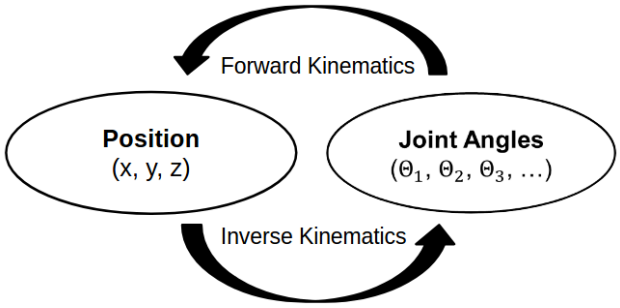


Figure 13: Visualization of inverse and forward kinematics [7]

Forward and inverse kinematics are relevant terms related to controlling body parts to a specific location or with a specific movement. If the goal is to reach a specific location with a robot's body part, methods of inverse kinematics are applied. More methods and applications of forward and inverse kinematics will be described in the following chapter.

2.6 Motion mapping techniques and algorithms

By mapping the motion of human to robot, it will be possible to imitate the human's behavior and transfer the physical capabilities of the human remotely to the robot. The robot should imitate the human motion in real-time and do the same tasks that the human is performing. The motion capture hardware provides position coordinates of human body parts in the absolute coordinate space. Motion mapping means processing these position coordinates in such a way that the output results in joint angles that can be applied to the robot joints. When applying the joint angles to the robot, it is expected to observe a similar or equivalent execution of motion by the robot. In other words, the robot should approach to reach the same position of the end-effector relative to the robot, as the position tracked by the motion capture hardware.

In the following sections, motion mapping techniques are explored that can be found in literature. Different studies describe the development of telerobotic systems, where within the context of motion mapping, also solutions for the problem regarding different kinematics between robot and human or other arising issues for motion mapping are addressed that were described previously in section 2.5. There are multiple steps that different papers describe and follow to create motion commands for the robot from motion capture of the human operator, which are generally

similar but differ in the specific kind of method. Figure 14 shows the general overview of steps to create a motion mapping. A few papers might not follow these steps in that particular order or might omit a step if it is not required in their approach.

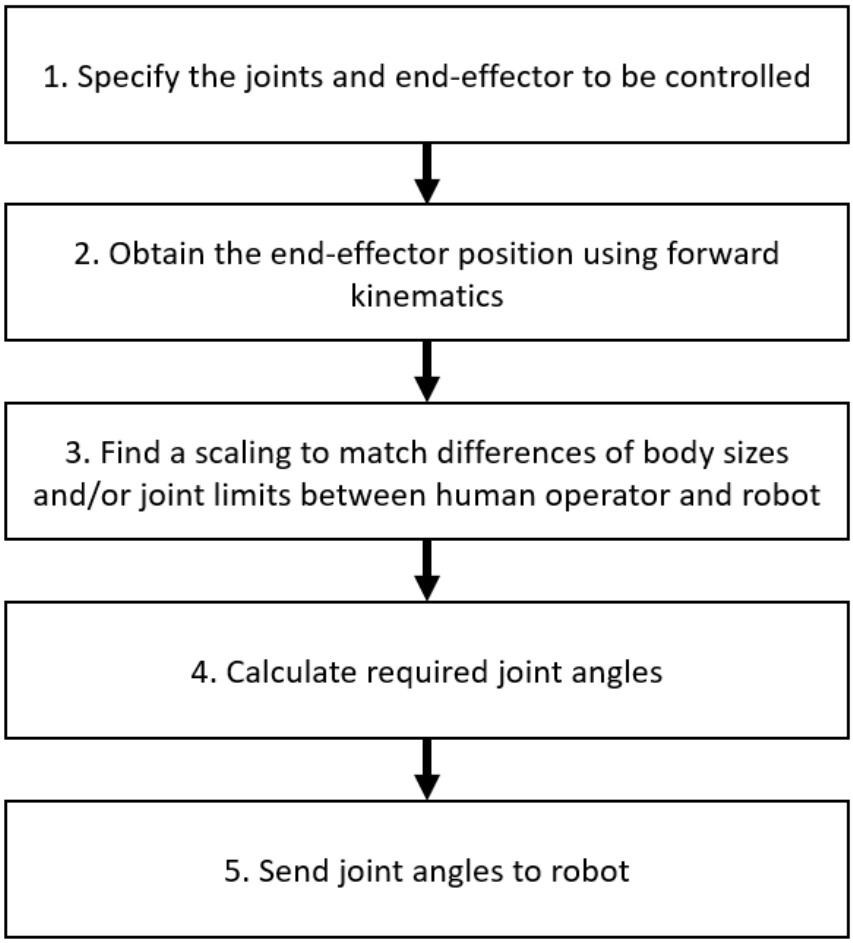


Figure 14: Pipeline for mapping motion from human to robot

The existing motion mapping techniques and their performance will be described and compared to draw a conclusion on the benefits and drawbacks of the methods. This will support the selection of a suitable technique for the particular case, the motion mapping of the EVE robot.

2.6.1 Step 1 to 3: End-effector, forward kinematics, and scaling

The preparation for mapping motion is performed by analyzing the differences of human and robot kinematics in chapter 2.5. This section elaborates on the first three steps of the motion mapping pipeline shown in figure 14 and addresses solutions to overcome the differences in human and robot kinematics. Firstly, the end-effector link to control is defined. Secondly, frames are assigned to links to calculate their position and the position of the end-effector. Lastly, scaling is performed to overcome differences in links length and joint limits to match a human body to the body of a robot.

In most of the found literature about the development of telerobotic systems, the robot link to control, the end-effector, is defined as the robot's hand. In this project, it is not necessary to control the hand but the shoulder position or any other part of the torso. However, the end-effector can usually be adjusted to any link, and therefore the methods in literature can be applied to control, e.g., the shoulder position solely.

One example is the system by Arduengo et al.[8], who control the posture of a single robotic arm with 7 degrees of freedom. The last link of the robot, the “hand”, is the end-effector. Therefore, the kinematic chain consists of all joints concerning the arm up to the end-effector. A kinematic chain is a series of connected links and joints that influence the possibilities of motion. On one end of the chain is the base, which is fixed, and on the other end is the end-effector, where no other link is attached to[24]. Similar to Arduengo et al., Mukherjee et al.[21] attempts to control the arm posture and therefore defined the hands of a NAO robot as end effector as well. Controlling both arms is divided into two subproblems, which means that the chains from left shoulder to left hand and right shoulder to right hand have to be individually mapped. Darvish et al. [9] present a whole-body teleoperation system for multiple robot models, but the 53 degrees of freedom iCub robot in particular. However, in this section, the focus will mainly be on upper body control.

Arduengo et al. [8] first describe finding a correspondence between the relative position and orientation of the human links and the robot's links up to a scaling factor. The following frames are defined for human and robot: arbitrary origin, virtual footprint, torso, shoulder, elbow, and wrist (Figure 15). Based on these frames, homogeneous transformation matrices are defined that describe the transformation from one frame to the other up to the wrist frame. Homogeneous transformation matrices are 4x4 matrices that incorporate a 3x3 matrix to describe the rotational transformation from one frame to the other (the orientation) and a vector to describe translational x, y, z transformation.

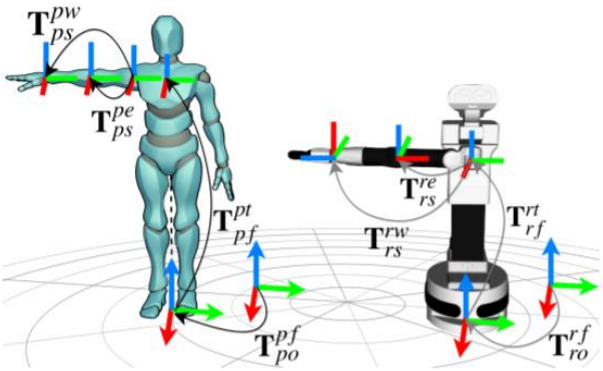


Figure 15: Assignment of frames to robot and human by Arduengo et al[8]

Arduengo et al. [8] depict dividing the translational components of the human transformation by the length of the link of the human and multiply it with the corresponding link length of the robot (1). The correspondence of orientation is calculated as in equation (2) when placing the robot and human in an equivalent pose.

$$\text{Position}_{\text{robot}} = \frac{\text{Position}_{\text{human}}}{\text{length}_{\text{human}}} * \text{length}_{\text{robot}} \quad (1)$$

$$Rotation_{robot_{l1}}^{robot_{l2}} = Rotation_{human_{l1}}^{robot_{l1}} * Rotation_{human_{l1}}^{human_{l2}} \quad (2)$$

Similar to Arduengo et al. [8], Darvish et al. [9] assign corresponding frames to the links of robot and human (Figure 16). However, instead of the position, Darvish et al. point out to solely use the rotation and angular velocity of the human links. To scale and map the motion of robot and human for the adjustment of the workspace, a fixed relative rotation between human links and robot links is

found (3) by positioning the robot and human subject in a similar joint configuration. The rotation is directly applied to the robot's URDF model.

$$Rotation_{origin}^{robot} = Rotation_{origin}^{human} * Rotation_{human}^{robot} \quad (3)$$

Even though, Arduengo et al. [8] reports good results and satisfactory motion matches between robot and human, [9] explains their different approach for initial scaling by pointing out that the workspace of the robot might be narrowed for reaching some points further away (if $\frac{\text{lengths}_{\text{Robot}}}{\text{lengths}_{\text{human}}} < 1$) or the precision might be lost for pinpoint manipulation tasks (if $\frac{\text{lengths}_{\text{Robot}}}{\text{lengths}_{\text{human}}} > 1$).

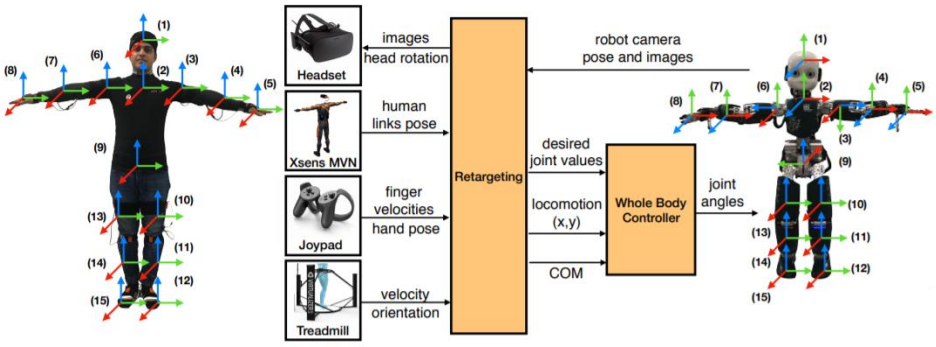


Figure 16: Telerobotic system and frame assignment by Darvish et al.[9]

Kim et al.[10] creates a scaling from human to humanoid robot (Figure 17) by multiplying the robot arm lengths with a constant c. This constant results from the division of the sum of the lengths of the upper and lower arm of a human subject by the sum of lengths of the humanoid robot arms in the same manner ($c = \frac{Length_{human}}{Length_{robot}}$). The orientation of human and robot links is directly mapped with the reasoning that the orientation is forced to match after the given scaling as well.

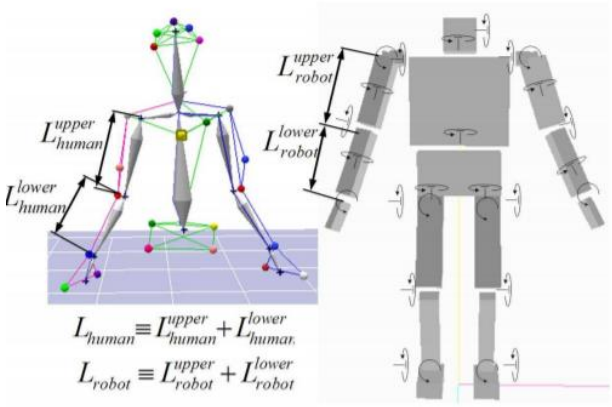


Figure 17: Motion mapping by Kim et al.[10]

According to Mukherjee et al.[21], there are four frames for each NAO arm that influence the position of the end-effector. Whereas Arduengo et al.[8] did not specify the derivation of the transformation matrices, Mukherjee et al. use a method called modified Denavit-Hartenberg (D-H)

Parameters to obtain homogenous transformation matrices for the forward kinematics so that the robot joints are calculated in reference to the previous joint[31]. This way, the x, y, and z coordinates of the end-effector frame could be obtained. For the D-H parameter method, it is required to provide the length of the lower and upper robotic arm. Mukherjee et al. decided to use the arm length of a human instead of the NAO since the coordinates of the wrist with respect to the shoulder that are given by the Kinect will be beyond the workspace of NAO's hands.

Stanton et al.[32] create a motion mapping between human and the NAO robot as well. Like Darvish et al. [9], Stanton et al. do not consider the translational position but focus solely on relative rotations between links. The absolute motion capture data is transformed to relative rotations by dividing one frame rotation by the previous one so that the kinematic mapping is not affected by the user's location and orientation in the absolute coordinate space.

Some approaches do not require any pre-processing steps before calculating the required joint angles at all. For instance, Sripada et al.[33] only utilize the orientation and position coordinates of the Kinect and directly applies the next step of calculating the required joint angles for motion mapping, which will be explained in 2.6.2.1.

2.6.2 Step 4: Calculate the required joint angles

In many cases, creating a mapping between the motion of a human and the corresponding robot links requires finding a solution to an inverse kinematics problem, which means the position and orientation of the body part to control is given, and the required joint angles need to be found, accordingly. There are several ways of solving inverse kinematics. However, while some papers describe complex ways to solve inverse kinematics, some methods do not target the task space (equivalence of end-effector position), but the configuration space, so the equivalence of the joint configuration (e.g., section 2.6.2.1).

Aristidou et al.[30] summarize inverse kinematic solvers in 4 different categories: Numerical solvers use an approximation for the forward kinematics first to iteratively solve the inverse kinematics, such as Jacobian, newton, and heuristic methods. Analytical solvers approach to find all possible solutions based on the lengths of the mechanism, the starting position, and the rotation constraints but usually approach to find a single solution built upon assumptions. Data-driven solutions aim to find a way of mapping motion, e.g., based on pre-learned postures that are matched with the positions of the robot joints.

2.6.2.1 Direct angle mapping

Mukherjee et al. [21] follows the approach of "direct angle mapping" and uses vector algebra to find the angles between the human links and maps them to the NAO robot arm with 6 DOF. The human joint coordinates involved are of the shoulder, elbow, and wrist and are captured by the Kinect. There are several studies following a similar approach. For example, the study by Sripada et al.[33] simply calculates the angles between joints after obtaining the position coordinates of the joints. This is then transformed to the motor speeds of the robot. The upper body control is performed with "appreciable accuracy" [33]. [21] reports about the requirement of needing three coordinates to determine four joint angles that define the position of the wrist, while other tested methods require less. Besides, this method resulted in jerky movements due to noisy Kinect readings and continuously changing joint values, which, however, could be solved by filtering the Kinect data.

Darvish et al. [9] does not apply a direct angle mapping approach but describes how configuration space retargeting works and what pitfalls could be encountered compared to task space

retargeting. By obtaining the human joint angles and velocities, a customized mapping step transforms them into the robot joint angles and velocities. Besides the consideration of dissimilarity between human and robot joints, a customized offset and scaling have to be found, and the robot joint constraints have to be taken into account as well.

2.6.2.2 Analytical methods

Analytical solutions are claimed to be mainly used for simple robotic systems[30]. Nunez et al.[34] creates an analytical solution for a humanoid robot with 18 DOF, where the arms have three DOF. The inverse kinematics of the used robot is divided into six subproblems: both arms, two feet with respect to the pelvis, and pelvis with respect to each foot. After assigning homogenous matrices to each frame of the robot, geometric formulas are derived to calculate the needed joint configuration. The implementation is described as straightforward and time-saving. The approach solved the kinematics successfully, however, there are no specific results mentioned about the performance. Kofinas et al.[35] reports about the advantages of the solution regarding accuracy and the elimination of singularities usually encountered at numerical solutions. Singularities are configurations in which there is a change in the expected number of degrees of freedom[36], which means certain movements become blocked. Mukherjee et al.[21], who also approaches to control a NAO robot, states that an analytical solution would be possible as developed by Kofinas et al., but it will require many different computations, which makes this solution rather time-consuming and laborious for more complex systems.

2.6.2.3 Numerical methods

Numerical solutions are relatively common among robotic systems, and different variations can be found. In the context of telerobotics and human-robot motion mapping, Arduengo et al.[8] propose a method where the inverse kinematics problem is solved by the Moore Penrose pseudo-inverse of the i-th task Jacobian, where a task can represent, e.g., the end-effector pose or available range of a joint. Similarly, Mukherjee et al. [21] compute the Jacobian matrix for a given robot configuration. The Moore-Penrose pseudo-inverse of the Jacobian matrix is created to calculate the change in joint angles of the robot to reach the desired position of a robot link. The used algorithm is created by Meredith and Maddock[37], where inaccuracies are checked in an iterative process after the pseudo-inverse is computed until the error is within an acceptable range.

Darvish et al.[9] describes an approach where the robot's joint positions are found by solving the inverse kinematics as an optimization problem using a dynamical optimization method utilizing a library for quadratic programming. The dynamical optimization method is similar to the usual iterative Jacobian, though it requires only a single iteration at each time step to find the solution, keeping the computational time constant, which ensures fast convergence of the error over time[38]. Comparable to Darvish et al. [9], Kim et al. use a dynamical optimization method as well by implementing the "SQP algorithm for nonlinear programming"[10]. The dynamical optimization method described by [9] and [10] aims to converge the frame orientation errors to a minimum. If the optimal posture is reached at a particular time grid point, it will be used as the initial value for the optimization problem for the next time grid point[10].

Regarding the performance, all methods show different results with certain limitations. The results of the method used by Arduengo et al. [8] show a successful and accurate imitation of motion. The mean of the absolute error for the end-effector position was about 11 cm and 0.05 radians for the elbow angle. However, the robot responded slow, which might be a physical constraint of the robot. Comparably, [21] also utilized the Jacobian pseudo inverse method and experienced a slow response of the NAO due to the number of iterations required at each step. Besides, the problem of singularities

was encountered. Using the dynamical optimization method, the upper-body retargeting performs in [9] and [10] well with low joints position error.

2.6.2.4 Data-driven methods

Over the last decade, the application of data-driven methods to solve inverse kinematics, in general, became more widespread [30]. Related to Telerobotics, a system was developed by Stanton et al. to control the NAO robot with a motion capture suit[32]. A feed-forward neural network with particle swarm optimization for each DOF of the robot is trained to find a mapping between human motion capture data, e.g., rotation of the human body links and robot motion, e.g., the angular position of each joint. For the learning and data collection process, the robot was programmed to slowly repeat a few different movements that the human operator has to repeat synchronized. The human imitates the motion of the robot and both, the robot's angular position data and the human motion capture data are logged for use in machine learning.

Moreover, there is a data-driven approach, where adaptive Neuro-Fuzzy Inference Systems (ANFIS) were trained using derived inverse kinematics equations and a set of joint angles with corresponding end-effector positions[21]. Adaptive Neuro-Fuzzy Inference Systems are used to map input to output and are similar to neural networks[39]. The trained systems received the position coordinates of the Kinect and returned the corresponding joint angles needed to reach the position. While [32] does not require any prior forward kinematic analysis, [21] describes deriving forward kinematics beforehand. Moreover, [21] does not mention the use of relative rotations, which might be due to the different motion capture hardware used. Additionally, the data-driven technique differs in both approaches as well. Another study tested multiple data-driven approaches for Telerobotics using a Vicon MX[40], resulting in a good performance and the preference over approaches different from data-driven ones[41]. Thus, data-driven approaches are available in different variations, nonetheless, so far, only a few were developed in combination with a motion capture suit or Kinect for a telerobotic system.

The application of data-driven inverse kinematic solutions in humanoid Telerobotics provides a promising and easy way of implementation. Based on the conclusions of [21], the neuro-fuzzy method was the most efficient and fastest out of three tested methods, e.g., Jacobian inverse and direct angle mapping. Since the systems are trained, the computation time is reduced, however, the training process might take a long time. More training data would result in higher accuracy. [32] agrees with this conclusion and reports an average mean error of solely 5.55% with 10 minutes of data collection time. The error is explained by differences in the repetitive motions creating multiple mappings while collecting data. In contrast to the other approaches, the main benefit of this method is that no mathematical modeling of inverse/forward kinematics is needed, as well as the flexibility to apply this method to any human subject, robot, and motion capture hardware. It can be claimed that the efficiency and benefits of data-driven solutions can be proved by multiple methods applied in practice.

2.6.3 Evaluation and concept development

In order to get more insights on the benefits and drawbacks to consider when choosing a suitable method for upper-body motion mapping with a humanoid robot, existing telerobotic systems were described and compared with each other. Each practical implementation provided certain benefits and drawbacks, which have to be considered and prioritized. Depending on the method chosen for acquiring the desired joint angles, the initial steps, e.g., forward kinematics, might be different. In any case, these steps are necessary to consider and solve issues regarding differences between human and robot kinematics.

First, it has to be defined what part of the robot should be controlled. Generally, it is desired to reach the same posture by the robot torso as the human torso. The goal is to allow the movement of the shoulders forward, backward, left, and right to the side and spinning based on the hip rotation. As concluded in chapter 2.5, the torso posture depends on three revolute joints located around the hips. Therefore the amount of degrees of freedom to control is 3. The end-effector can be any part of the torso since the whole torso is moving when manipulating the hip joints.

The advantage of this specific robot, the EVE, is that according to the URDF model, the three hip joints are placed at the same location, which means there is no significant displacement between the hip joints and frames that have to be taken into account, like it is the case with arms that have joints with a certain distance to each other. Besides, the EVE is about the same size as a human subject, which means additional scaling of the torso size might not be necessary or only necessary to a minimal extent. The idea of using the rotational position/orientation only for motion mapping as it is done by a few papers could be adapted as well since it makes scaling for the translational position redundant.

The parts of the torso, e.g., shoulders, chest, hips are rigidly connected, which means any part of the torso can be chosen as the final link. Therefore, it is the easiest to choose the same frame of the torso where the hip joints are located. The advantage of choosing the hips or pelvis as end-effector is that only the orientation has to be taken into account for the motion mapping. There is no translational change of the pelvis position with respect to the legs.

Regarding finding the desired joint angles, different categories such as performance, time and complexity of implementation and the computation time, as well as further benefits and drawbacks of the described methods of chapter 2.6 will be compared with each other in Table 1:

Method/category	Performance	Time and complexity of implementation	Computation/ response time	Overall score
Direct angle mapping by Mukherjee et al.[21], Sripada et al.[33]	+ Good accuracy	+ Fast and not very complex, only little calculations involved	++ No iterations, therefore fast response	+
Analytical methods by Nunez et al.[34], Kofinas et al.[35]	+/- Good accuracy, but better performance for simple systems	+/- Calculations might be more cumbersome than angle mapping	+ Fast response time	+/-
Moore Penrose pseudo inverse by Arduengo et al.[8], Mukherjee et al. [21]	+/- Satisfactory accuracy, but Singularity problems possible	- Rather complex algorithm	- Many iterations, therefore slow response possible	-

Dynamical optimization by Darvish et al.[9], Kim et al.[10]	++ Good performance, low position error	- Rather complex algorithm and implementation	+/- Faster than other iterative solutions[38]	+/-
Data-driven methods by Stanton et al. [32], Mukherjee et al. [21]	++ Good performance	+/- Generally easy implementation, provided there is background knowledge about feed-forward neural networks or ANFIS Data collection and training process might be rather time consuming	++ Solely assignment of Neural network/ ANFIS output, therefore fast response	+

Table 1: Comparison of different methods for finding the required joint angles

The direct angle mapping approach seems easy and quick in implementation with only little calculations involved. There are a few problems encountered and described by a few analyzed papers, but potential solutions are provided. The main challenge with this method is to consider the constraints of the robot kinematics, such as the joint angle and workspace limits, though, as mentioned in section 2.3.2, a solution could be just to let the robot hit the joint angle limits if the human moves out of the range.

The main benefit of the numerical solutions is that they apply to complex robotic systems and provide a satisfying accuracy as described in the reviewed approaches. On the other hand, e.g., calculating the Jacobean pseudo inverse can require a lot of computation time, which can be a cumbersome process. Furthermore, singularity problems were mentioned and a slow response of the robot due to many iterations in one computational step. On the contrary, analytical solutions can be implemented quicker than, e.g., numerical solvers, are generally reliable, and do not suffer from singularity problems. However, the more complex the robotic system is, the more computations are needed.

Furthermore, data-driven solutions have a crucial advantage of fast and easy implementation since the inverse kinematics are solved based on collected motion data of a human operator that correspond to the same humanoid robot motion. The results are reported to be accurate and better compared to other methods. One of the presented data-driven methods does not even require the formalization of forward and inverse kinematic equations. Another benefit is the scalability of the solution, which means that it can be applied to all kinds of robots and different human subjects. Nonetheless, on the one hand, data-driven approaches require much data to create an accurate mapping. On the other hand, repetitive movements might lead to multiple mappings for a particular motion.

Considering the scoring from Table 1 and the complexity and performance of the described methods, the direct angle mapping method and the data-driven solution using Adaptive Neuro-Fuzzy Inference Systems (ANFIS) seem to provide the best results with low complexity of implementation. While some motion mapping methods might require extensive pre-knowledge about machine learning

techniques, the adaptive Neuro-Fuzzy Inference Systems are based on a toolbox[42] in MATLAB and is simpler in application.

Since the kinematic problem is not very complex (three hip joints of the EVE that are all at the same location), numerical solutions are not required. Besides, they are also not always advised to be used based on the results of the analyzed papers and Table 1. An analytical solution could be possible but require more calculations than other methods, such as the direct angle mapping approach or the mentioned data-driven method. In conclusion, two methods will be developed, where one utilizes Adaptive Neuro-Fuzzy Inference Systems, and in the second method, direct angle mapping will be attempted.

Ultimately, the conceptual procedure looks as follows: For the ANFIS approach, homogenous transformation matrices between the robot base and end-effector have to be assigned to calculate the orientation of the torso given different joint configurations. The calculated orientations and the corresponding joint angles will be given as input to the ANFIS.

As described previously, the captured pelvis orientation does not always influence the position of the entire torso of the human since the spine can initiate the rotation of the shoulders as well. Therefore, instead of capturing the pelvis orientation, the orientation of the human chest will be used to determine the configuration of the robot hip joints.

The direct angle mapping approach can be realized without any calculations since the rotation of the chest in x-, y- and z-direction provided by the XSENS software can be directly applied to the hip joints of the robot within the range of the joint angle limits. Necessary scaling or adjustments for the mapping of robot and human orientation for both approaches can be only identified later in the implementation process.

2.7 Conveying social and human-like behavior in (tele-)robotics

The goal of conveying social behavior within the context of Telerobotics with the EVE is to make the interaction with the humanoid robot less artificial and produce a feeling for the people interacting with the robot of truly having another person being embodied by the robot with all aspects making the embodied person human-like, natural and social. Therefore, it will be examined how existing robotic systems convey social behavior and how social behavior can be applied to telerobotic systems.

2.7.1 Displaying body language and encountered limitations

Body posture and body language express the emotional state in a non-verbal way[43]. For instance, Figure 18 shows different emotions that can be observed only by taking the body posture into account and not considering the facial expression. For this reason, the motion mapping used for teleoperation should be able to translate the emotional state of the human operator to the robot. [43] points out that their test results have shown that by using a motion-capture suit for the control of an artificial agent, already a wide range of emotions could be detected since the system copies the movements and posture of the human operator.

Figure 18: Body language for emotional expression [11]

However, multiple papers inform about certain limitations regarding displaying body posture by a robot. For instance, physical constraints of the robot, e.g., another number of degrees of freedom of the robot compared to the human body, affect the expression of the emotional state[44]. Experiments with a NAO robot performing semantic gestures have shown that people's interpretation of the robot's gestures and the same gestures performed by a human are different, which could mean that the expression of emotion is not strong enough and decreases due to the physical limitations[45]. [44] agrees with the results of [45] after carrying out similar experiments. [46] examines more than 20 social robots and points out as well that the limitations in flexibility of movement affect the social presence and human-robot interaction.

According to [43], the use of motion capture leads to the loss of secondary cues and micro gestures. The recognition of emotions is not affected, but the emotional strength. For instance, during experiments, the actor was asked to perform emotions such as relief or sadness where visible sighs

were used along with other movements. These sighs were not present or heavily diminished by the artificial character. Furthermore, it is mentioned that breathing, which is a secondary cue as well, might be relevant since it also provides a good indication of emotional strength. Experiments of [43] have also shown that the smoothness of movements does not affect the correct identification of emotions or the naturalness and believability of those. Moreover, the paper claims that "characters should move the way they look"[43], which means the less realistic and the more cartoony the robot or character is, the more stylized movements it should have to express emotion in a better way.

2.7.2 Animation techniques in robotics

As claimed in the previous section, secondary cues are of great relevance for expressing emotion and human-like behavior. Those cues also appear in "Disney's Twelve Principles of Animation", which are rules that are usually applied in movies for character animation[47]. Several papers mention the potential of applying these principles to (social) robots[46]. However, they are not described in the context of teleoperated robotics. Some principles could be used in combination with Telerobotics, while other principles appear redundant. All of the principles[47] will be described and later evaluated if they can be applied to a telerobotic system.

"Squash and Stretch" is the principle for movement and liquidness of an object, where the principle states that the objects should keep the same volume while squashing and stretching. This principle is not applicable to robots since they are usually built of rigid parts. Another principle is "Anticipation", which should help viewers understand what the character is going to do. "Staging" should ensure that the expressive intention is clear to the viewer and is related to the general set-up in which the character expresses itself. In robotics, it relates to the way of expression, which can be with lights or sound.

Furthermore, "Straight ahead and Pose-to-Pose" relates to the animation method, where the animator either animates frame to frame and decides spontaneously how the next frame should look like (Straight ahead) or pre-plans the end and beginning poses and designs the frames that are needed in between (Pose-to-pose). Moreover, "Follow-Through and Overlapping Action" relates to avoiding abrupt stopping of motions, which is perceived as unnatural. "Slow In and Slow Out" should ensure that the motion has a natural speed and soft blending of motion. On the contrary, there is "Timing", which determines the speed of motion that is also relevant for expressing emotion.

The principle "Arcs" states that natural motions occur in arcs instead of linear motions. "Secondary Action" is related to passive behavior such as breathing or blinking. "Exaggeration" can be used to emphasize the robot's movements and expressions to make them more noticeable. According to the principle of "Solid drawing", a character should not stand stiff and still. Usually, people put, for instance, more weight on one leg, and poses are rarely symmetrical. Finally, the principle of "Appeal" determines the design of a character and how the viewers should feel about the character when looking at them and their behavior.

A literature review paper[46] identified the application of animation techniques, such as Arcs, Secondary cues, Anticipation, or Pose-to-Pose, as well as Motion capture in combination with other techniques for more than 20 robots. It concluded that all of these papers prove that animation techniques actually improve the interaction with robots and show the robot's emotional state. Terzioğlu et al.[48] implement breathing as a secondary cue for a robot by implementing small movements while the robot is idle. Testing showed that it has a positive effect on social presence and perceived sociability.

2.7.3 Evaluation and concept development

Literature research has shown that an accurate motion mapping using motion capture hardware is generally sufficient to represent a wide range of emotions (see 2.7.1), however, physical constraints of the robot and the loss of secondary cues and emotional strength compromise the visibility of social behavior. Different animation techniques, e.g., movement in arcs, solid drawing, secondary action, or exaggeration, might be applied to the humanoid robot to compensate for the mentioned limitations. Notwithstanding, since motion mapping is used as a technique for conveying social behavior, only a limited amount of animation principles can be potentially applied. Thus, the following table indicates which animation techniques can be applied to Telerobotics:

Animation technique	Applicable to Telerobotics?
1. Squash and stretch	No, robots are built of rigid parts
2. Anticipation	No, covered by motion mapping
3. Staging	No, covered by the telerobotic system
4. Straight ahead and pose-to-pose	No, but it can be useful for creating the animation for secondary cues
5. Follow-through and overlapping action	No, but it can be useful for creating the animation for secondary cues
6. Slow in and slow out	No, but it can be useful for creating the animation for secondary cues
7. Timing	No, but it can be useful for creating the animation for secondary cues
8. Movement in arcs	No, covered by motion mapping
9. Secondary action	Yes, e.g., breathing, sighs, laughing
10. Exaggeration	No, covered by motion mapping
11. Solid drawing	No, but it can be useful for creating the animation for secondary cues
12. Appeal	No, the design of the robot cannot be influenced

Table 2: Applicability check of animation techniques for telerobotic systems

As can be observed in Table 2, most animation techniques are already covered by the motion mapping and can be steered by the human operator. The technique that is not necessarily covered is "secondary action", which are motions that the human operator performs unconsciously, such as breathing, sighs, or upper body motion due to laughing. Breathing was mentioned as a secondary

action already used or described among multiple papers with positive results (see section 2.7.2) and implemented during the idle state of the robot, in which the robot is not moving in the actual trajectory. Therefore, the idle state could be used as a trigger to start the breathing animation. Triggers for laughing motion or sighs are rather hard to determine and probably require methods of sound recognition. Besides, the human operator is able to consciously exaggerate his sighs and other motions with the upper body if it is desired to portray it. For this reason, it is proposed to implement the breathing cue while the human operator is not moving in order to avoid inaccuracies while performing relevant tasks during motion mapping.

Based on the paper by Tsoli et al.[49] together with their video[12], it can be observed that during breathing, not only do the shoulders move back and forth, but the head and arms move slightly up when breathing in (Figure 19). The speed of these movements can indicate emotional strength, e.g., fast movements equal stress and anxiety, while slow movements equal calmness and relaxation. If the speed should be controlled, a heart rate sensor could be used, e.g., by means of a wristband. Alternatively, the breathing cue will remain at the same speed while the robot is in the idle state. The idle state would mean that all the robot links move with a velocity, which is close to 0. A threshold should make sure that noise does not reactivate the active state.

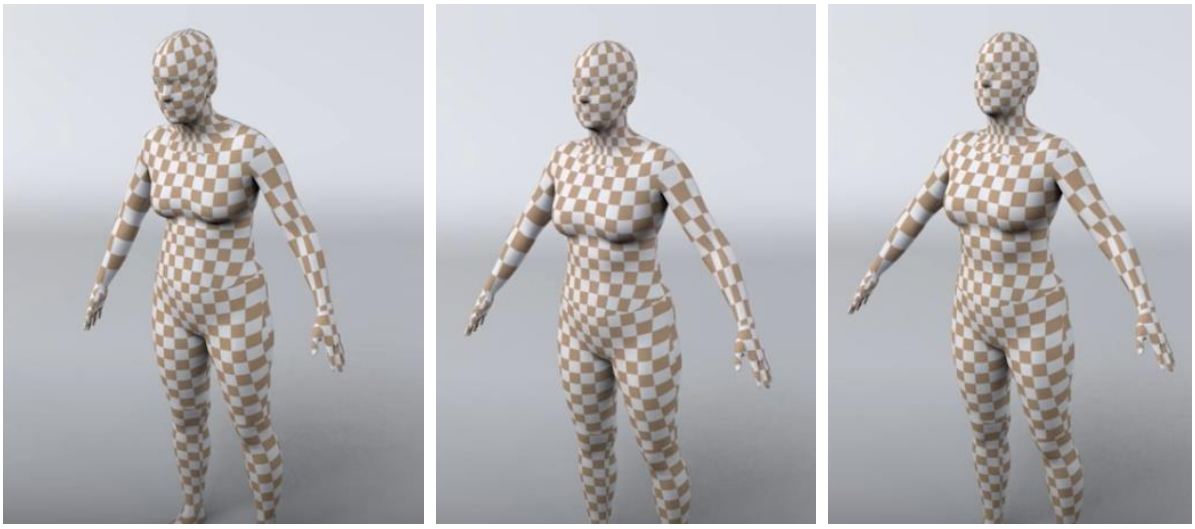


Figure 19: Screenshots from [12]*: Animated breathing motion*

Therefore, the breathing animation will be based on the defined motion by Tsoli et al.[49] and will be implemented as small movements of the upper body going forth and back. As Table 2 indicates, certain animation techniques can be applied for the creation of the breathing animation. The principle of pose-to-pose could be used to pre-define different postures during the breathing process, e.g., start and end position, and create the required joint configurations for in between these positions. Follow-through and overlapping action and slow in slow out draw attention to avoid abrupt motions and design the breathing animation smooth and natural. Timing can be related to the speed of breathing, which can be determined during the implementation and which influences the emotional expression. Finally, according to the principle of solid drawing, the upper body does not have to be perfectly symmetrical during the breathing motion since this is usually also not always the case in real life. Thus, during the implementation of the breathing, the described principles will be taken into account as well.

3 Implementation

In this chapter, the implementation of the developed concepts will be described. First, the general specification regarding the implementation will be explained. Afterward, further details will be stated within the description of realization.

3.1 Specification

The system will be developed with MATLAB since it provides the necessary toolboxes to implement the motion mapping techniques described previously in the concepts. Besides, MATLAB also offers the option to create a connection to ROS, which is required in order to apply the motion mapping to the physical robot and integrate it into the existing system. The used motion capture system will be the MVN Awinda Xsens motion capture suit together with the MVN Analyze software. The motion capture data, specifically the chest orientation, will be streamed in real-time into MATLAB, where the motion mapping has to be able to produce commands for the robot motion simulation in real-time. It should be possible to combine the motion mapping technique with the created breathing animation and later integrate it into the existing telerobotic system. The integration into the existing telerobotic system will, however, not be realized within this project.

3.1 Set up of the motion capture system

Setting up the XSENS motion capture suit starts with the correct placement of the wireless motion trackers on the upper body using straps and a shirt with Velcro patches (Figure 20, 21). The XSENS MVN analyze software has the limitation that upper body motion capturing of the torso without the arms and head is not possible. Thus, nine sensors had to be placed: head, chest, pelvis, left and right shoulder, upper arms, lower arms[5]. The sensors are recognized by the software (Figure 22) by means of the Awinda Station that is connected to the computer and receives the data wirelessly[5].

Figure 20: Xsens front sensor placement

Figure 21: Xsens back sensor placement

Figure 22: Active sensors(green) in Xsens software[5]

In order for the motion capture to work correctly, it is required to calibrate the suit within the software, which is facilitated with the assistance of another person. This is because software messages and instructions have to be confirmed during the calibration process, and the calibration can turn out wrong or bad with additional unnecessary movements. For the calibration, the human subject has to stand straight for a couple of seconds and then walk about 3 meters to the front and back again. The calibration ends with standing straight in a chosen direction, registered as the x-direction.

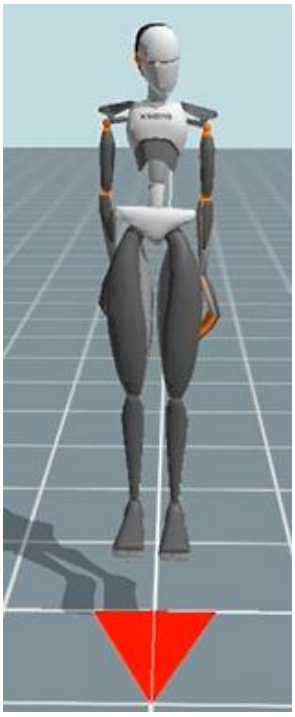


Figure 23: Xsens model with visual origin frame

The origin frame, marked as red arrow (Figure 23), lies directly underneath the captured human body. When selecting a body segment, e.g., pelvis, chest, or shoulder, the provided position and orientation are always in respect to the origin frame. The advantage of capturing only the upper body (starting from the pelvis) is that no translational change in position of the overall subject is possible in the software, where solutions are addressed by research papers, e.g. [32]. That means that it does not matter where the human subject sits or stands; the translational position of the pelvis will not change. Furthermore, when the upper body is moving in the x-, y-, or z-direction, only the orientation of the pelvis changes and not the translational position.

The next step is to stream the motion capture data into MATLAB in real-time. XSENS provides the necessary steps and code in their developer toolkit[50], which needs to be installed separately. In the XSENS software, the data streamed into MATLAB can be specified, e.g., position and orientation. The data of a specific body segment has to be defined within the MATLAB code (see Appendix F). As stated in the concept (section 2.4.3), the chest orientation will be streamed into MATLAB and has to be defined by its ID number. The code provided by XSENS is only an example code for plotting the data and is changed and adjusted for motion mapping as will be described in the following sections.

3.2 Motion mapping using ANFIS

This section describes the data generation and training of the ANFIS, which will be used and evaluated as approach to map motion from human captured data to the EVE robot. Moreover, it will be explained how the robot simulation is created and updated given the ANFIS output.

3.2.1 Data generation

Adaptive Neuro-Fuzzy Inference Systems require input and output data in order to be trained. The input data are all the possible position coordinates of the end-effector and the output data are the joint angles that are required to reach the corresponding position.

In order to produce this data, the first step is to import the URDF model of the robot[27] into MATLAB. The provided URDF model has the limitation that the pelvis is defined as the robot's base and is fixed to create a separation between the lower and upper body. Thus, in simulation, the lower body includes the hip joints that influence the position of the leg, while they do not influence the upper body at all. This was observed by experimenting with the robot model and changing the joint configuration in MATLAB. In order to fix this, the URDF file had to be modified so that the fixed base is at the leg, and the hip joints do have an influence on the torso posture. It is relevant to note that the URDF model is still correct, and solely the fixed base is moved to a lower location. The joint configuration can be changed by creating a structure array, where the joint name and value are given (Figure 24).

1x18 struct with 2 fields

Fields	JointName	JointPosition
1	'j_hip_y'	0.1700
2	'j_hip_x'	0.5164
3	'j_hip_z'	1.0328
4	'j_l_shoulder_y'	0
5	'j_l_shoulder_x'	0
6	'j_l_shoulder_z'	0
7	'j_l_elbow_y'	0
8	'j_l_elbow_z'	0
9	'j_l_wrist_y'	0
10	'j_l_wrist_x'	0
11	'j_neck_y'	0
12	'j_r_shoulder_y'	0
13	'j_r_shoulder_x'	0
14	'j_r_shoulder_z'	0
15	'j_r_elbow_y'	0
16	'j_r_elbow_z'	0
17	'j_r_wrist_y'	0
18	'j_r_wrist_x'	0

Figure 24: Structure array for the joint configuration of EVE model

The easiest way to create this array is by computing: config = homeConfiguration(robot), which sets every joint angle to 0.

Moreover, the URDF file also provides the joint angle limits of the three hip joints in radians. The goal is to create a data set that includes many joint angles within those limits as necessary. Values with an increment of 0.02 radians (about 1.1 degrees) were chosen to produce different joint angles:

```
hip_x_values = -0.523599: 0.02: 0.523599; % +/- 30 deg
hip_y_values =-1.57: 0.02: 0.174533; % + 10/- 90 deg
hip_z_values = -1.0472: 0.02: 1.0472; % +/- 60 deg
```

In another piece of code (Appendix C), iteration takes place through all possible combinations of the joint angle values, which were then applied to the joint configuration of the robot. After that, it was possible to obtain the homogenous transformation matrix of the current configuration from the base to the torso, which is defined as the pelvis in the URDF model (Figure 25):

This command creates the transformation matrix from base to pelvis for the current joint configuration:

```
tf_1 = getTransform(robot,c,'pelvis');
% c represents the current joint
configuration
```

Since only the rotation/ orientation is considered for the motion mapping, the rotation matrix is extracted from the homogenous transformation matrix and converted into the rotation in x-, y-, and z-direction (yaw, pitch, roll). The yaw, pitch, and roll values, as well as the three joint values, are all attached to separate data sets that are later combined into three data sets containing each all the possible orientations and one of the three hip joints since for each joint, an ANFIS has to be trained individually.

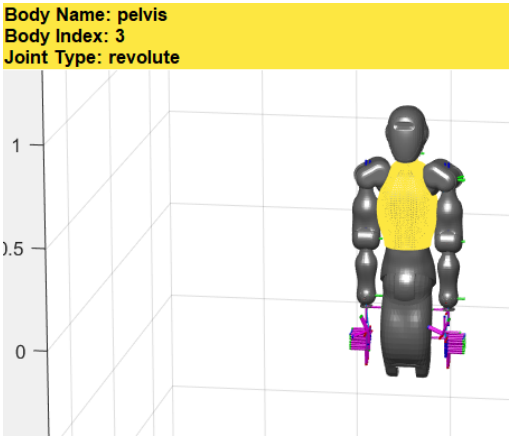


Figure 25: Pelvis segment of the EVE model

3.2.2 ANFIS training

The resulting data sets contain about 489720 rows of data, which is a considerable high amount and could lead to an interminable training time for the ANFIS. This was confirmed by a warning message when starting to train the ANFIS in MATLAB, which stated that MATLAB could run out of space during the training procedure. Thus, before training the ANFIS, an initial Fuzzy Inference System with subtractive clustering is trained that can be later finetuned using the "anfis()"[51] function. The next couple of paragraphs touch upon some theory behind Fuzzy Inference Systems and why an initial Fuzzy Inference System with subtractive clustering is a potential solution to avoid the encountered negative consequences when using a high amount of training data with the default training method, grid partitioning, for ANFIS.

Fuzzy inference systems are the foundation of Adaptive Neuro-Fuzzy inference systems. Fuzzy inference is the process of mapping an input to an output using Fuzzy logic[52]. ANFIS can then be created by tuning the Fuzzy inference systems utilizing neuro-adaptive learning techniques similar to those used for training neural networks[53]. The difference between Fuzzy logic to classic/Boolean logic is that not only true (1) or false (0) are possible values, but the value can be between 1 and 0, such as 0.2, 0.7, or 0.5[54]. Fuzzy inference systems consist of a list of if-then statements (If *x* is *A*,

then *y* is *B)* called rules, which are evaluated and interpreted in parallel using fuzzy reasoning[52], e.g., figure 26.

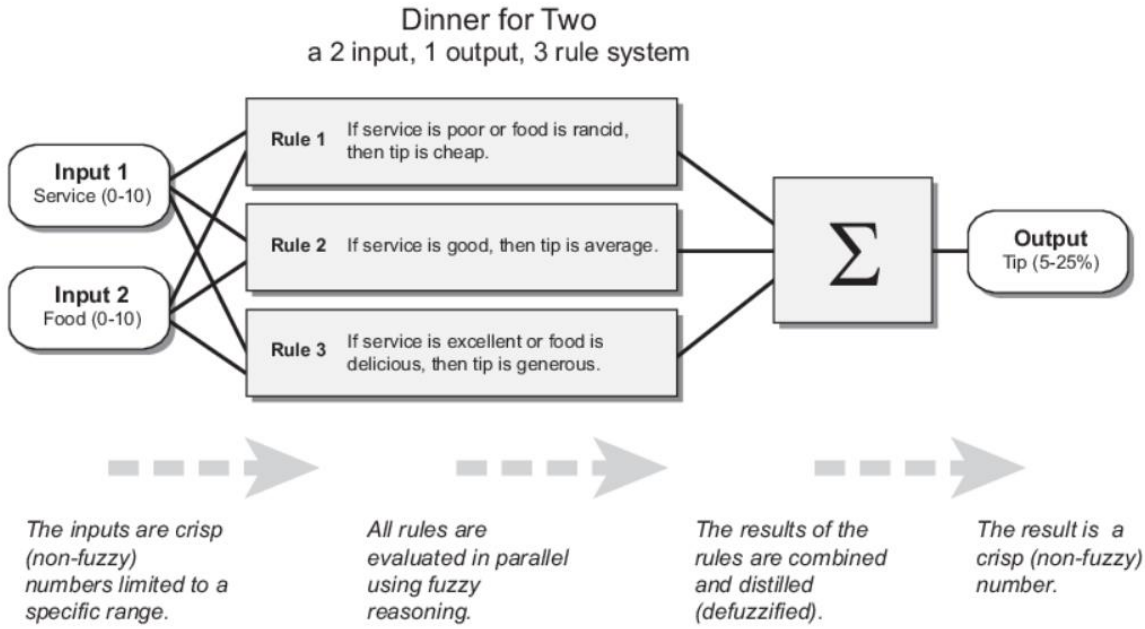


Figure 26: The concept of Fuzzy inference systems[52]

Membership functions are curves that define how each value in the input space is mapped to a degree of membership between 0 and 1[54]. For instance, figure 27 shows that March to June are 100 percent members of spring with Boolean logic, while fuzzy logic shows a transition and rates June to be a 50 percent member of spring. After creating the membership functions, the degree to which each input is satisfied for each rule is known[52], e.g., to what extent is the service poor, good, or excellent. Within the MATLAB Fuzzy logic toolbox, 11 different membership function types exist[54], whereas the simplest functions are formed using straight lines (e.g., in the form of a triangle). There are also membership functions based on the Gaussian distribution curve and have the benefit of being smooth. The rules and membership functions are created within the Fuzzy inference process given the entered input and output values.

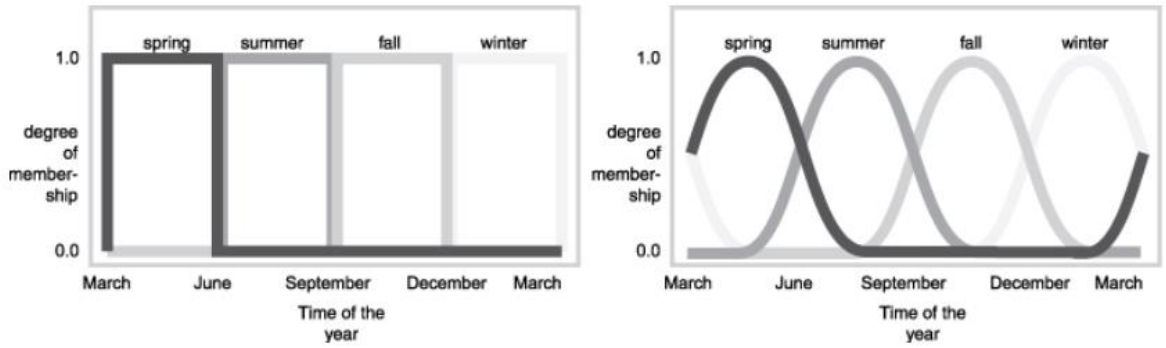


Figure 27: Membership function example (Left graph: Boolean logic, Right graph: Fuzzy logic)[54]

If the ANFIS are trained without an initial Fuzzy inference system, the function, "anfis()"[51], uses grid partitioning as default option for the training method. Grid partitioning generates input membership functions by uniformly partitioning the input variable ranges, while for the subtractive clustering method membership functions and rules are derived from data clusters[55]. Clustering aims to identify natural grouping of the data, which models the data behavior using a minimum number of

rules[56]. Subtractive clustering is a “fast, one-pass algorithm for estimating the number of clusters and the cluster centers for a set of data”[56], where each input variable has one gaussian input membership function for each fuzzy cluster. There is one rule per cluster[57].

Moreover, subtractive clustering allows to specify a clusters center range of influence in each of the data dimensions (e.g., the input data has three columns and the output has one), so the width of data space is times a scalar value between 0 and 1[58], [59]. By specifying a large cluster radius, a few large clusters with fewer rules are produced. Specifying a small radius usually leads to many small clusters in the data, producing a Fuzzy inference system with many rules[59]. According to [60], low radius values can affect the subtractive clustering model not to be mapped well and significantly increases the calculation time. In contrast, high radius values can lead to a non-good partitioning of the inputs.

A relevant advantage of using a clustering method to find rules is that the resulting rules are more tailored to the input data than grid partitioning. The total number of rules is reduced when the input data has a high dimension[61].

The Fuzzy Inference System will be created using the “genfis2()” function[58]. “genfis2()” generates a Sugeno-type Fuzzy inference system using subtractive clustering. To determine the number of rules and membership functions, the algorithm works as follows[59]:

1. Calculate the likelihood that each data point would define a cluster center based on the density of surrounding data points.
2. Choose the data point with the highest potential to be the first cluster center.
3. Remove all data points near the first cluster center within the defined clusters center’s range of influence
4. Choose the remaining point with the highest potential as the next cluster center.
5. Repeat steps 3 and 4 until all the data is within the influence range of a cluster center.

Given this information, the chosen scalar value for the influence range of a cluster is 0.7, which resulted in a fuzzy inference system training time of approximately 3 hours per system. Based on the resulting performance of the outcome, which will be evaluated in chapter 4, the scalar value could be adjusted and experimented with.

In order to improve the performance of the systems, they will be optimized using the “anfis()” function. It is possible to adjust, e.g., the optimization method, number of training epochs, and training error goal[62]. However, for this ANFIS training, it is decided to keep the default options and analyze the resulting performance first. The default number of training epochs is relatively short, with ten epochs. The higher the number of epochs, the longer the training time takes. There is no training error goal defined, and the number of rules and membership functions is pre-defined by the initial Fuzzy inference system created before.

The options for the ANFIS tuning were configured to set the initial fuzzy interference system using the following command:

```
opt = anfisOptions('InitialFIS',inFIS); % inFis represents the
initial Fuzzy inference system
```

The command `outFIS = anfis([input output],opt); %[input output] fuses the input and output data into one data set` was used to train the ANFIS. This was done for each of the three joints. The ANFIS tuning went faster than the initial Fuzzy inference system training with about 10 to 20 minutes per joint. The resulting training error or also

root mean square error (RMSE) of each ANFIS shows good results: The ANFIS for the x-joint returns an RMSE of 0.005, for the y-joint an RMSE of 0.021, and for the z-joint an RMSE error of 0.022 (Appendix A). The ANFIS output and performance will be analyzed in the evaluation chapter.

3.2.3 Robot simulation

In order to visualize and simulate the motion mapping with the EVE in MATLAB, the robot model has to be updated timely with the new configuration so that the change of motion can be seen. The joint configuration of the robot will be produced by the ANFIS based on the received orientation of the chest from the Xsens motion capture. The ANFIS and robot visualization will be implemented into the MATLAB code provided by Xsens for streaming position and orientation values (Appendix F).

The captured orientation is received in the representation of quaternions, which consists of 4 variables. Since the ANFIS were trained with Euler angles, a representation of the orientation in yaw (rotation in the z-direction), pitch (rotation in the y-direction), roll (rotation in the x-direction), the command:

```
[ xsens_ya, xsens_p,  xsens_r] = quat2angle([ ori(segmentId,1)
ori(segmentId,2)  ori(segmentId,3)  ori(segmentId,4)]);
```

was used to transform the received quaternion to Euler angles. Afterward, the three orientation angles were inserted into one input data set, which was then given as input to the three ANFIS to receive the predicted joint angles. These were then applied to the joint configuration of the robot.

With `show(robot,config,'PreservePlot',0, 'FastUpdate', 1);`, it is possible to update the robot visualization without closing the figure, which results in smooth transitions between different joint configurations. Lastly, in order for the simulation to work, it is necessary to restrict the frequency of the running code since it otherwise runs too fast for the visualization to be updated correctly. A frequency of 2000 is still fast enough for the visualization to update smoothly and corresponds to a period of 0.0005 seconds. For this simulation, the joint configuration may move out of the allowed range. If this system is applied to the physical robot, the joint angle limits will be reached, and the joint will not move further until the joint angles are again within the limits.

3.3 Direct angle mapping

The direct angle mapping approach is implemented faster and easier than the ANFIS approach since it is simply converting the received quaternions to Euler angles and applying it directly to the joint configuration of the robot. The joint for the y-direction is controlled by the received pitch angle, the roll is assigned to the x-joint, and the z-joint receives the yaw angle of the human chest orientation. The simulation works with the same principle as described in the previous approach using the show() command and frequency control.

3.4 Breathing animation

The breathing animation will be based on predefined poses, similarly to the poses defined in Figure 19, and include a change of position of head, arms, shoulders, and chest. In order to create a motion animation, a motion trajectory has to be built, which includes the three predefined poses with the corresponding joint configuration and all the joint configurations that are required for a smooth transition between the poses. It is possible to create these transitional joint configurations automatically using a trapezoidal velocity profile[63], which uses waypoints/predefined configurations

as input to create a trajectory between them. Therefore, it is first going to be described how to create the predefined poses for the EVE in MATLAB and furthermore how the velocity profile is configured.

By initializing an interactive figure of the robot with the command "interactiveRigidBodyTree()"[64], it is possible to move joints within the visualization and store the current joint configuration of the robot with the command "addConfiguration()", which adds the joint configuration to the "StoredConfigurations" attribute of the interactive robot object. The following figures show the predefined positions of the breathing motion:

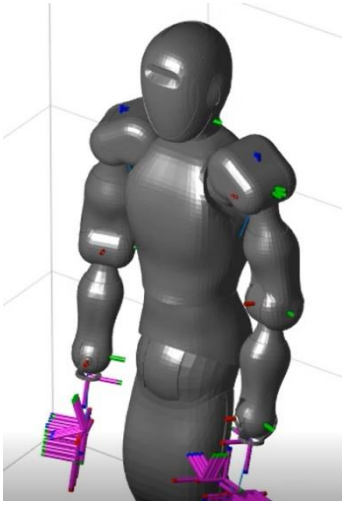


Figure 28: Starting pose of the breathing animation

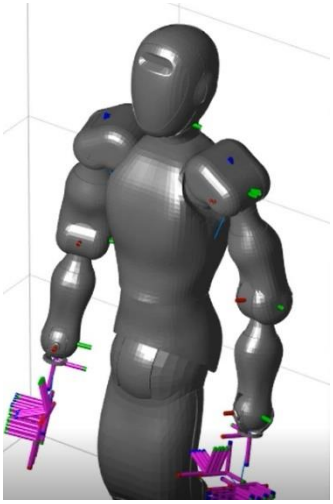


Figure 29: Pose after breathing in

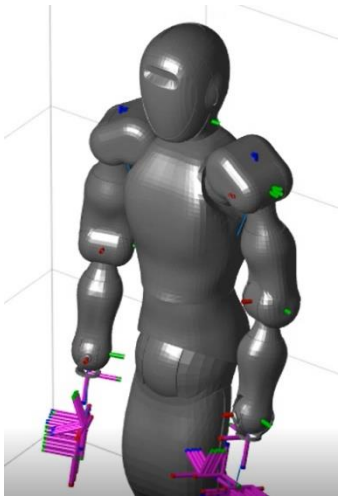


Figure 30: Pose after breathing out

In figure 28, it can be seen that the starting pose consists, corresponding to the first position of figure 19, of a slight side lift of the arms, a slight drop of the torso, and head to the front. Figure 29 shows a side lift of the arms to a larger extent and a rotation to the back. The torso and head are lifted to the back as well. The final pose (figure 30) is equivalent to the starting position (figure 28).

Once the poses are stored, a trajectory can be constructed using a trapezoidal velocity profile[63], which means that the robot stops smoothly at each pose but achieves a set speed when being in motion[65]. Given the number of stored configurations multiplied by a value between 10 and 50, a number of samples will be created that are part of the animation.

The trapezoidal velocity profile is created with the command:

```
[q,~,~, tvec] = trapveltraj(inte.StoredConfigurations,numSamples); % q is a matrix that represents a series of joint configurations that move between each pose, tvec is a vector/array of the time samples at which the output position q is sampled
```

To play back the animation, the joint configuration of each created frame of the interactive robot object is copied to the joint configuration of the initial robot visualization, which is updated with each frame (Appendix E).

When integrating this breathing animation into the motion mapping code, the rotational velocity of the human motion data should reach below a threshold close to 0, corresponding to the human being in idle state. While the velocity is below the threshold for a certain time, the breathing animation should start and continuously play until the human operator starts moving again.

There are two different versions of code implemented to check if the velocity is below a threshold for a certain amount of time, e.g., three to four seconds. The first version is defined by calculating the velocity, using the distance between the latest received rotational position and the rotational position received the previous time, divided by the number of frames, e.g., 10, counted between the old and new position. By looking at the velocity values during robot motion, a threshold for the idle state was extracted. An if statement checks if the velocity is below the threshold, which activates a counter, capturing the number of frames the robot is idle. If the counter reaches 15 frames, the motion mapping stops, and the breathing animation starts.

The second version of code is similar to the first version, however, it includes fewer computational steps and is therefore more efficient. This is because the velocity is not calculated separately but is directly integrated into the counter, which checks how long the robot is idle. It consists of an If statement, which calculates the distance between the latest received rotational position and the rotational position received the previous time. If the distance is below a value of 0.01 rad (= 0.57 degree), a counter starts and increases its value by one. If the distance goes above 0.01 rad, the counter resets back to 0. However, the breathing animation starts if the counter reaches a value above 70, which corresponds to a distance below 0.01 rad for approximately three to four seconds. This value can be adjusted later if it is desired that the breathing starts after a longer time being in the idle state.

Moreover, a smooth transition between the breathing postures and the captured posture is developed, which is done by creating another trajectory before the breathing animation starts and after it ends. The transitional trajectories use the last posture from the motion mapping to the first breathing position. After the breathing animation, the last breathing position smoothly changes to the first position of the motion mapping. The complete code can be found in Appendix F.

A pitfall of the previous breathing trajectory might be that the starting and end position (figures 28, 30) involve a specific posture, and after each breath, the robot returns to the motion capture position, which means that if the breathing animation takes longer than one breath and the motion capture position is not the same as the last or first breathing position, the transition from breath to breath could visually lose naturalness.

Therefore, an alternative breathing trajectory is created, where all joint angles in the starting and end position (figures 31, 33.) have the value zero. Like the previous breathing animation (figure 29), the second pose has a side lift of the arms and a rotation to the back. The torso and head are also slightly lifted. This alternative breathing allows taking the robot's current position directly as starting position for the animation. The second pose is reached by adding the change of joint angles to the starting position.

In the evaluation chapter, trajectory plots of both breathing animations will be analyzed to assess the differences and the integration into the motion mapping algorithm better.

Later, the motion mapping and breathing animation can be integrated into the existing telerobotic system, where the head and hand motion is captured by other systems. That would mean that the head and hand motion has to be taken into account as well in order for the breathing animation to work properly.

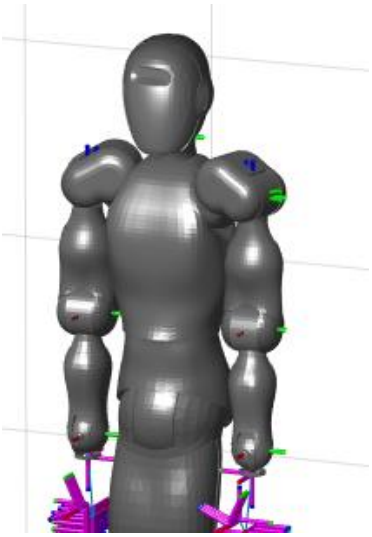


Figure 31: Starting pose of the alternative breathing animation

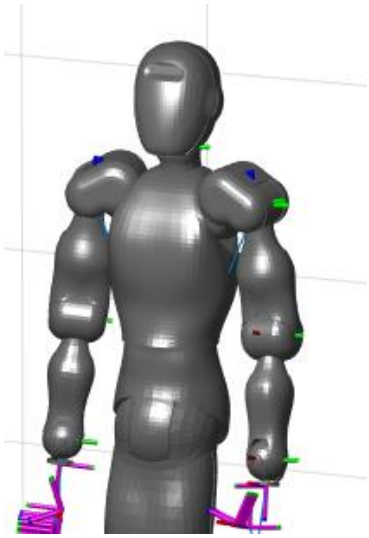


Figure 32: Pose after breathing in

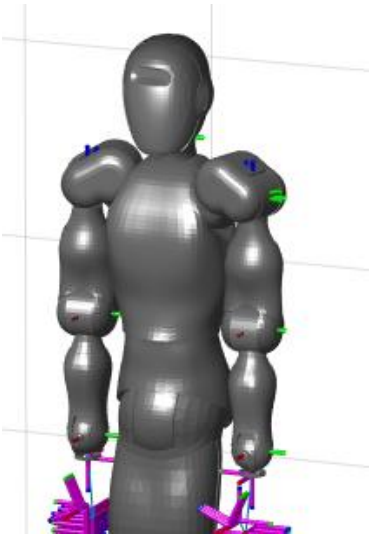


Figure 33: Pose after breathing out

4 Evaluation

In the previous chapter, two different approaches were developed to map the motion from human to robot, which will be evaluated and compared with each other in terms of performance and system requirements. The use of the Xsens motion capture system will be touched upon as well. Besides, two breathing animations have been created and integrated into the motion mapping algorithm, which will also be accessed.

4.1 Evaluation criteria

The motion capture hardware will be assessed based on the following criteria:

- Invasiveness
- set up time
- calibration process
- accuracy
- ease of use

The motion mapping techniques will be tested by performing different upper body motions and evaluated based on:

- Complexity
- Real-time mapping/response time and smoothness
- Accuracy (visual and value-based evaluation)

The upper body motions during testing will involve separate movements of the torso in x-, y- and z-direction, as well as combined movements in multiple directions.

The breathing animation will be assessed in relation to:

- Realness and naturalness
- Performance in combination with motion mapping

4.2 Xsens motion capture system

General benefits and drawbacks of the Xsens motion capture system were already discussed in chapter 2.4. However, during the implementation phase, further points of interest regarding the use of the Xsens system appeared.

For instance, the developed motion mapping system requires only data from one sensor, while the Xsens software does not enable motion capturing when wearing only one sensor. For upper body motion capturing, at least nine sensors have to be worn so that the calibration can take place. This is a negative aspect when rating the criteria of invasiveness. One of the system requirements states that the new motion mapping component should not result in too many gadgets that the human operator has to wear on the body. A solution to solve this problem would be to find a way that allows the operator to wear only the one sensor that is needed instead of nine. Besides, putting on all the needed sensors every time before teleoperating the robot increases the setup time significantly.

Additionally, calibrating the Xsens suit with good quality is hard to achieve by only one person. There has to be another person involved that helps with confirming software messages and guiding the human operator during the calibration process of what to do and when to do it, e.g., how long the

user has to stand straight and when the user is allowed to walk. The whole calibration process might take up to a couple of minutes, especially if it has to be repeated due to lousy calibration results, which appeared quite often during the implementation phase and testing.

The accuracy of the Xsens system is, as expected, good and solid but does also depend on the calibration quality. Besides, during movements, a sensor or strap may move, which influences the motion capture accuracy. Therefore, it has to be made sure that the straps are well and tight attached to the operator's body.

The use of the Xsens suit can be rated as very easy. The setup and interface are simple. Streaming data into MATLAB could be done in little time.

Overall, it can be stated that the use of the Xsens suit appeared to have a couple more drawbacks than initially assumed. Nevertheless, it is easy to use and provides accurate data if the setup and calibration are done correctly.

4.3 ANFIS validation

Before comparing the ANFIS method with direct angle mapping, the trained ANFIS are first going to be validated in terms of accuracy of the mapping from input data (roll, pitch, yaw of the hip) to output data (configuration of the hip joints).

The ANFIS output, the hip orientation that the system was trained with, will be entered into the system to receive the predicted joint angles. The predicted joint angles will be subtracted from the joint angles the system was trained with and used to calculate the hip orientation initially. It can be observed in figures 34 to 36 that the initial angle matches the predicted angle for the majority of the time with only a small error for all three joints.

However, every graph shows spikes that reach up to 0.166 radians (= 9.5 degrees) deviation for the z joint, whereas the x joint deviation shows the best results with 0.05 radians (= 3 degrees). Nevertheless, this error occurs only at a certain input value, while it is steady around a maximum deviation of +/- 0.02 to 0.06 radians for most of the predicted angles. In section 5.4, it will be observed if the deviations have a negative impact on the motion mapping.

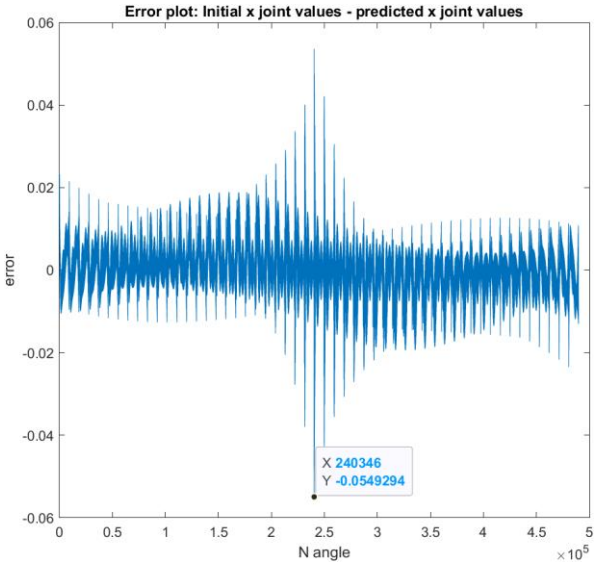


Figure 34: Error plot of initial values of the x joint and predicted joint values

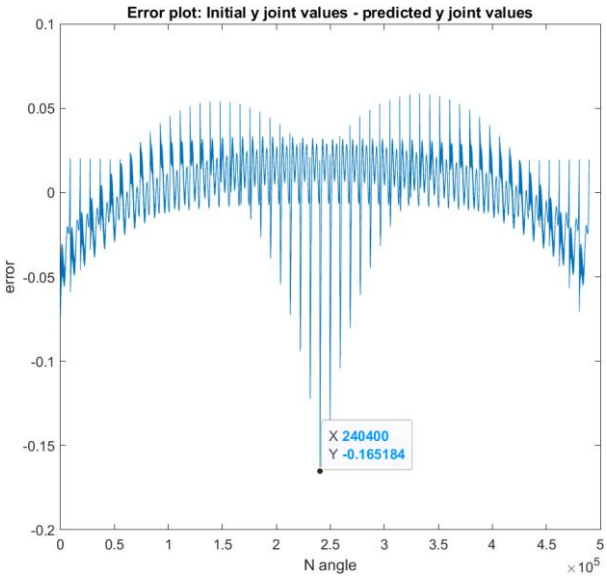


Figure 35: Error plot of initial values of the y joint and predicted joint values

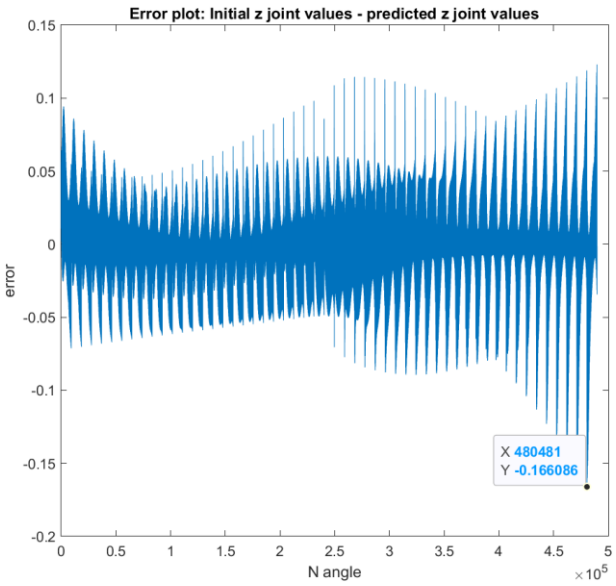


Figure 36: Error plot of initial values of the z joint

It could be possible to improve the ANFIS by not solely using the default options for training the (adaptive neuro) Fuzzy interference systems but optimize the options or try out different values for the cluster influence range regarding subtractive clustering. Additionally, it could be tested if a bigger step size for the data generation, leading to a smaller data set and, therefore, a shorter computation time, could affect the training performance. Overall, the ANFIS output shows decent results that are acceptable for this project.

4.4 Comparison of motion mapping techniques

Regarding the comparison of the complexity of the techniques, it can be claimed that the direct angle mapping technique is implemented a lot faster and easier and requires fewer computational steps than the ANFIS method. Generating data and training the adaptive neuro-fuzzy interference systems is done at one time. However, it takes 9 to 10 hours until all three ANFIS are trained, whereas the direct angle mapping approach only requires the assignment of the angles captured by the Xsens to the robot. Within the mapping algorithm, the ANFIS method calculates and assigns the joint positions within eight basic operations. In contrast, the direct angle method does it in four steps, excluding the Xsens data streaming loop, which is equal for both methods. However, both algorithms run smoothly and in real-time. Therefore, there is no adverse effect visible in the simulation.

For both techniques, no significant visual differences in motion mapping accuracy can be identified. Figures 37 and 38 show screenshots of the motion mapping with the ANIFIS technique, and figures 39 and 40 show the direct angle mapping technique. More postures can be found in Appendix B. Both techniques show good accuracy in mapping the motion between the human and robot. The accuracy can be confirmed by the orientation plots (Figures 41 to 46), where the human chest orientation is drawn in blue, and the orientation of the robot torso is shown in red.

It can be seen that the robot orientation is very close to the human orientation for both algorithms. However, overall, the direct angle mapping shows a better match than the ANFIS method. The worst orientation match of the direct angle mapping is the yaw orientation, while the pitch orientation shows the most significant deviation for the ANFIS method. The resulting roll orientation of the ANFIS method has the best performance, which is the expected outcome based on the error plot after the ANFIS training in figure 34, which shows the lowest error out of the three ANFIS. Another influence on the ANFIS performance is given by the joint angle ranges of the input data. Currently, the motion mapping does not filter out captured orientation values outside the joint limits, which means that it is possible to obtain inaccurate mapping if the incoming orientation data is far away from the data range with which the systems were trained, e.g., figure 45.

It can be concluded that given the complexity and accuracy of the algorithms, the direct angle mapping approach performs better and should preferably be used to teleoperate the torso posture of the robot. Nevertheless, the results of the ANFIS method are not incorrect or unacceptable, and this approach can definitely be considered for more complex kinematic problems, e.g., with more than three DOF and displacements between joints. For this project, it appeared that the more straightforward method does provide more reliable results and computational efficiency.

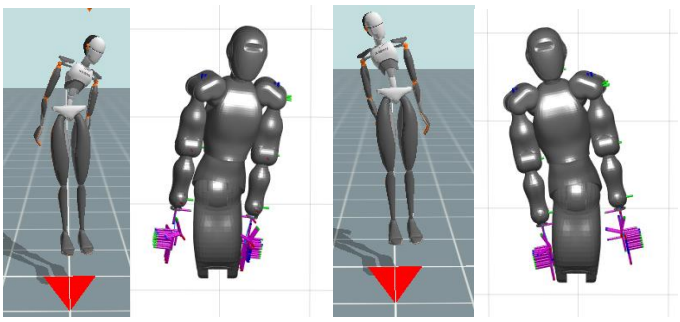


Figure 37: Visual ANFIS motion mapping accuracy (motion sideways)

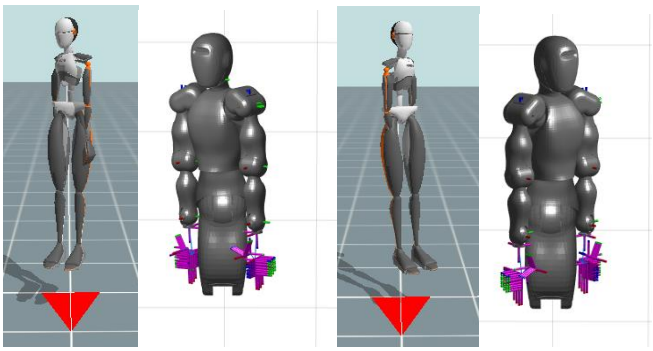


Figure 38: Visual ANFIS motion mapping accuracy (spinning motion)

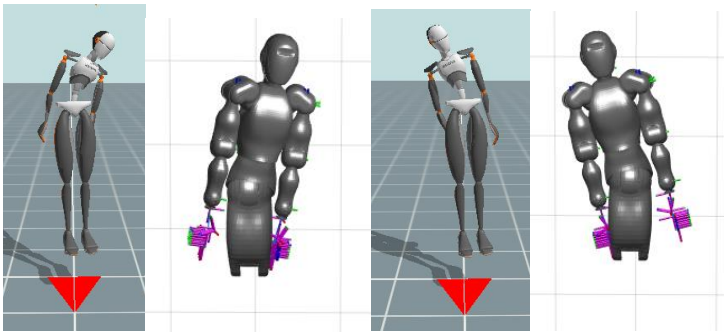


Figure 39: Visual direct angle mapping accuracy (motion sideways)

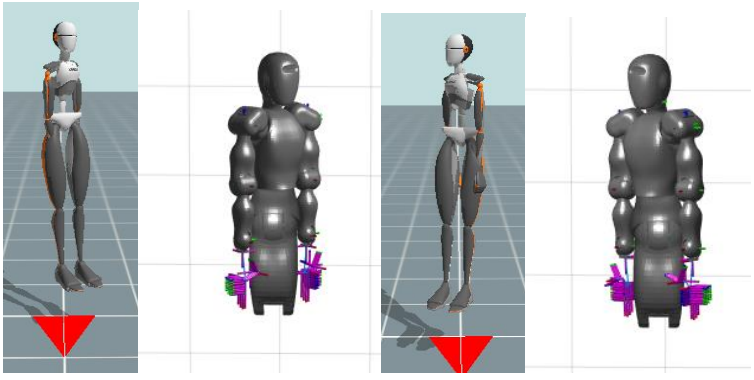


Figure 40: Visual direct angle mapping accuracy (spinning motion)

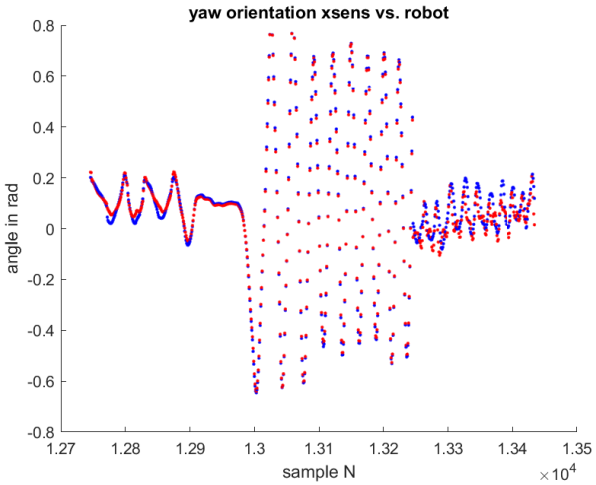


Figure 41: ANFIS motion mapping accuracy (yaw, Xsens = blue, robot = red)

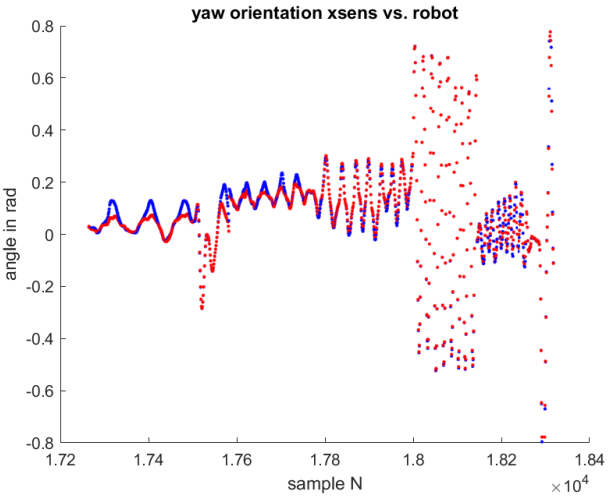


Figure 42: Direct angle mapping accuracy (yaw, Xsens = blue, robot = red)

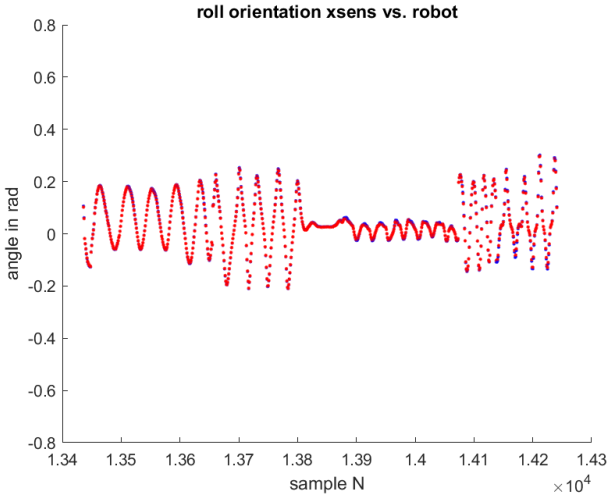


Figure 43: ANFIS motion mapping accuracy (roll, Xsens = blue, robot = red)

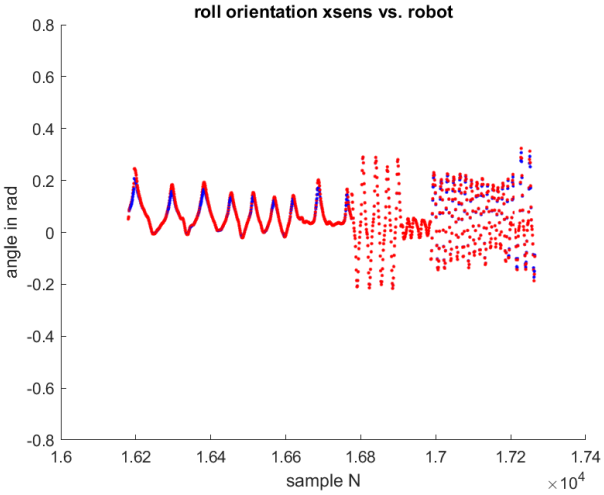


Figure 44: Direct angle mapping accuracy (roll, Xsens = blue, robot = red)

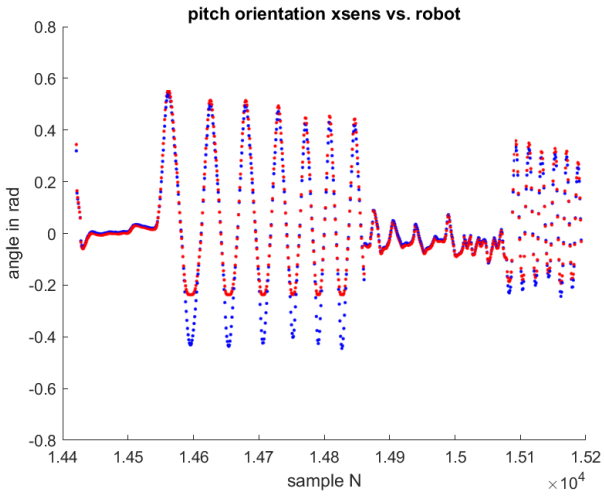


Figure 45: ANFIS motion mapping accuracy (pitch, Xsens = blue, robot = red)

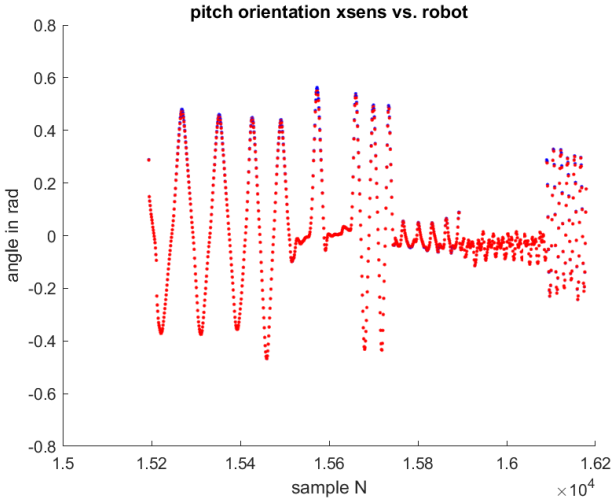


Figure 46: Direct angle mapping accuracy (pitch, Xsens = blue, robot = red)

4.5 Breathing animation

The goal of the breathing animation is to convey the feeling of more natural humane and social behavior during interactions with the robot. Two different breathing trajectories were developed and implemented in combination with the motion mapping. The difference between both animations is that the second one (figures 31 to 33) is less conspicuous than the first one (figures 28 to 30), which is achieved by keeping the joint positions at zero radians for the starting posture. For instance, it can be seen in figure 48 that the head orientation of the first animation starts from about -0.5 radians and reaches a maximum of about 0.2 radians, while the second animation (figure 51) starts at 0 radians and reaches a maximum of 0.25 radians. The starting posture of the first animation is created by moving the head and chest slightly down and lifting the arms at a slight angle. The next posture moves the head and chest up and increases the arm lift even more, corresponding to the breathing animation created by [49]. When implementing the first animation into the motion mapping algorithm, the robot first has to change its position to the first breathing posture before the animation can start. To visualize the breathing trajectory of both animations, figures 47 to 56 show the position and orientation trajectory plots of several robot links.

The most significant deviation can be seen in the orientation of the left wrist (figures 47, 50). Besides the fact that the yaw and pitch orientation of the first animation starts at higher angle values, it can be observed that the change of orientation during the trajectory also provides only a little volume compared to animation two. Therefore, if the robot model transitions from the last motion mapping posture to the starting posture of animation one, e.g., from 0 rad to 0.4 rad, the transition has a higher impact on the robot motion than the breathing animation itself. This is also the case if the robot goes back in the human captured position after each breath, which can visually introduce unnatural movement if the human captured position is not similar to the last breathing posture. On the contrary, the second breathing animation takes the robot's current position into account by replacing the joint positions at the starting position (= 0 radians) and adding the change of joint positions during the trajectory to it. This eliminates the issue of the first animation regarding the transition from motion mapping to breathing and contributes to more naturalness and realness of the breathing behavior.

Furthermore, the position plots of the left wrist of animations one and two show slight differences as well (figures 53, 55). The orientation and position of the left shoulder of both animations show no relevant differences. It can be stated that animation two is more suitable to be applied in combination with the motion mapping algorithm. However, there is still room for improvement and adjustments of the second breathing posture of figure 32.

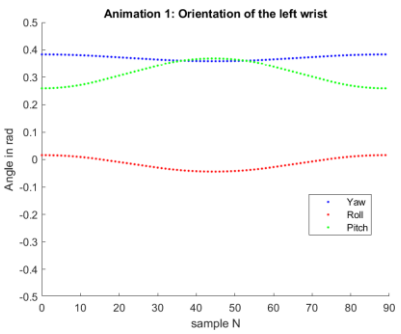


Figure 47: Wrist orientation of breathing animation 1

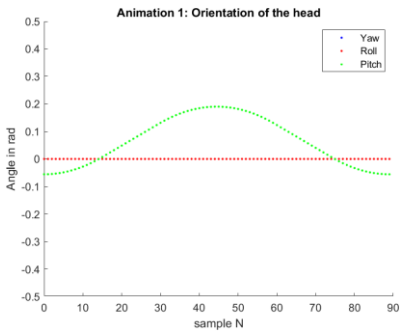


Figure 48: Head orientation of breathing animation 1

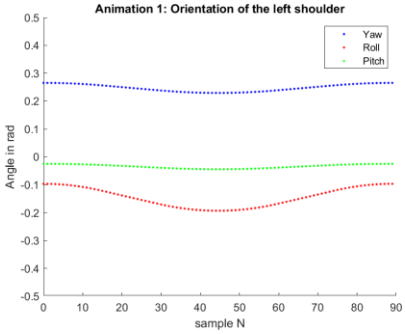


Figure 49: Shoulder orientation of breathing animation 1

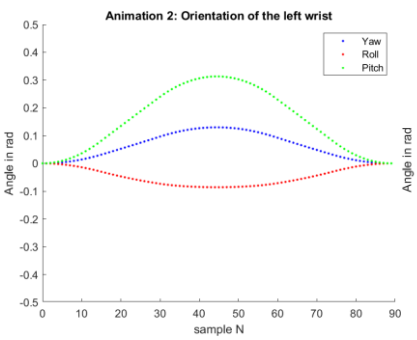


Figure 50: Wrist orientation of breathing animation 2

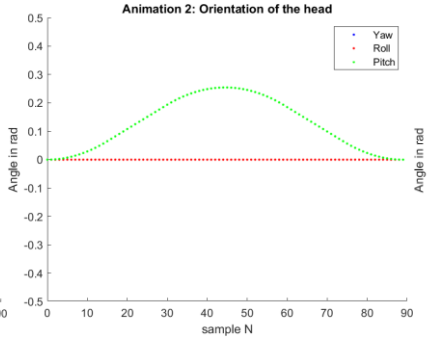


Figure 51: Head orientation of breathing animation 2

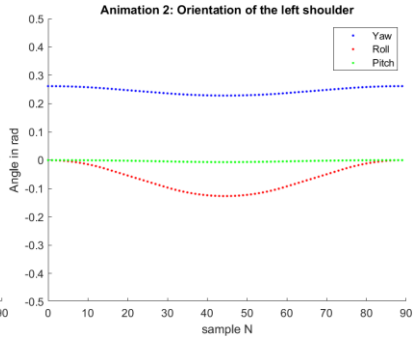


Figure 52: Shoulder orientation of breathing animation 2

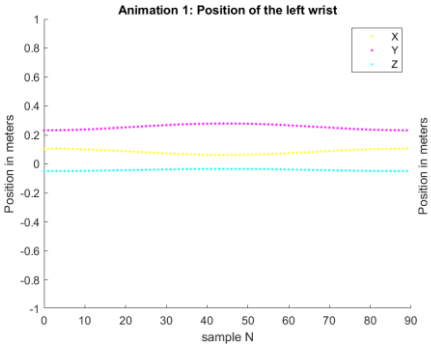


Figure 53: Wrist position of breathing animation 1

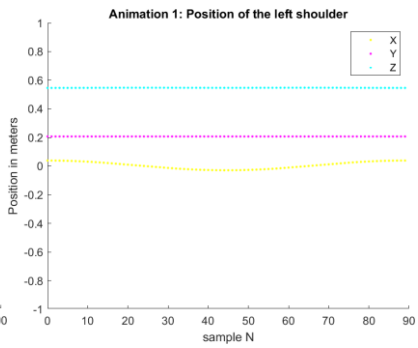


Figure 54: Head position of breathing animation 1

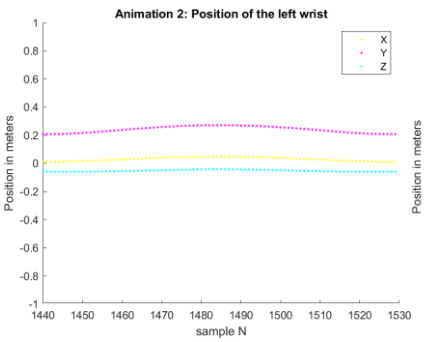


Figure 55: Wrist position of breathing animation 2

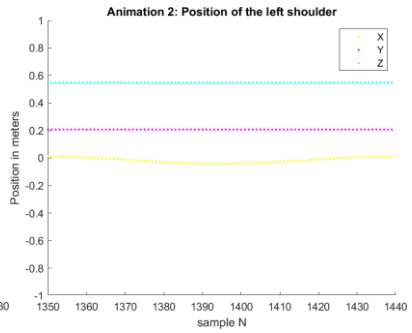


Figure 56: Head position of breathing animation 2

Moreover, two versions of the algorithm for combining the breathing animation and motion mapping were developed, as described in section 3.4. After testing the code, it can be concluded that both versions work equally well. The breathing animation starts smoothly if the human operator keeps the chest still during different postures. However, as mentioned in 3.4, the second algorithm, which does not calculate the velocity separately, is computationally more efficient.

One drawback is that the breathing animation has to entirely end before it is possible to use the captured motion data for teleoperation again. So in the case that the human operator starts moving again while the robot is still breathing, the motion mapping will not work directly. Therefore, this process has to be optimized in the future, and it has to be ensured that the one breath does not take too long. The visual performance of the breathing animation and the motion mapping can be seen in this video: https://youtu.be/RFGkooaRQIY

5 Conclusion

This project aimed to contribute to the development of the telerobotic system by i-Botics with the possibility of operating the upper body posture remotely and in real-time. Moreover, research was done on how the upper body posture of the EVE can convey social behavior. To conclude the findings, the formulated research questions in chapter 1.3 will be discussed and answered.

The research question "*Which body posture capturing hardware is most suitable given the requirements and availability for the existing controller system?*" was discussed in the background research, where advantages and disadvantages related to the Xsens suit and the Kinect were described. It was concluded that in order to answer the question, both systems had to be tested in application. However, within this project, only the Xsens suit was evaluated. The motion mapping with the Xsens suit works successfully, though some further drawbacks in using the system were discovered. Besides, the Xsens suit was not tested in combination with the existing controller system. Therefore, only assumptions could be made about the suitability of the hardware. Overall, it can be claimed that the Xsens suit is a decent and suitable hardware that can be potentially used for the existing system.

Furthermore, an answer was found for the question "*How can the body posture of a human operator be efficiently mapped using an algorithm to translate motion to a humanoid robot?*". Literature research contributed to identifying potential challenges when mapping the motion of human to robot, which involve, e.g., differences in degrees of freedom or joint angle limits. Several research papers that developed such a motion mapping presented their solutions for mapping motion successfully. A general mapping pipeline could be developed that almost every paper follows. It was decided to use two different approaches and analyze which one performs the best. The direct angle mapping approach proved not only to involve a fast and easy implementation but, at least for this project, also match the human captured torso orientation better than the ANFIS approach. Therefore, the technique to efficiently translate motion to a humanoid robot is the direct angle mapping approach.

Unfortunately, the question "How can the upper body capture mapping algorithm be integrated into the existing control system?" could not be answered due to time limitations. However, after research and consultation with an i-Botics researcher, integration into the existing control system is theoretically possible. The algorithm might need to be rewritten in the C language so that MATLAB does not have to be used as an additional program involved. The use of the Haption virtuoses will not be obstructed if the human operator moves slow enough. Besides, the virtuoses capture the hand position, which does not always move, if the torso moves. Further research on the system integration should be done in the future.

Finally, the question *"How can the chosen algorithm be adjusted in such a way that the humanoid robot can convey social behavior and gestures of the operator?"* could be answered as well. Background research showed that there are not many adjustments that can be done in the context of Telerobotics since the human operator steers the postures of the robot. However, by implementing the secondary action of breathing, a humane characteristic could be included, supporting the perception of human-like behavior. This ensures that the robot does not stand like a statue during conversations with other people. After integration into the algorithm, the robot starts to breathe while the human is in the idle state with good performance. However, the pitfall is that the motion mapping only continues after the breathing motion is finished, which affects the requirement of accuracy and control on time.

6 Discussion and future work

As touched upon in the previous chapter, there are some limitations concerning this project influencing the conclusions. Firstly, due to time limitations, the integration of the torso motion mapping into the existing telerobotic system could not be realized, and testing of the motion mapping was solely done in simulation without taking any other systems into account. Therefore, the suitability of the developed algorithm could not be thoroughly evaluated. However, it is crucial to ensure the different components work together and know if adjustments have to be made. This also includes accessing the motion capture hardware within the existing Telerobotics setup since only assumptions could be made if the system meets the requirements. Thus, future work should include the research into combing the new system into the existing system.

Another aspect of limitation is the issue when combining the breathing animation with the motion mapping. The breathing animation has to finish until the motion mapping can continue and negatively impact timely motion mapping. Multithreading might be a potential solution for this, but more research has to be done. Moreover, the perception of the breathing animation is based on the subjective opinion of the author of this thesis since no testing with human participants took place to access different points of view. With future perspective, the breathing animation should be tested with human subjects to research if the breathing animation indeed increases the perception of social or human-like behavior. Optimizations and fine-tuning might need to be applied.

Additionally, it was suggested to control the speed of the breathing animation with a heart rate sensor, which appeared to not be in the scope of this project. Nonetheless, future research can include this topic as well. Next to the integration of the torso motion mapping and breathing animation into the existing system, the position of the elbows could be mapped as well. The virtuoses by Haption capture the hand position only and calculate the required joint positions. If it is desired to control the position of the elbows actively, this might be a point of interest for future work. Finally, testing out different motion capture hardware, such as the Kinect, and comparing it to the Xsens suit is also a relevant aspect to conclude which hardware is more suitable in application with the existing telerobotic system.

Nonetheless, the results of the research in this project include relevant findings, which provide a relevant foundation for future work and the integration of new features for the telerobotic system by the i-Botics group.

References

[1] i-Botics, “i-Botics XPRIZE - Semifinalist Selection Submission - Full.” [Online]. Available: https://www.youtube.com/watch?v=ddcS92uzGCc

[2] “EVEr3 Humanoid Research Robot — Halodi Robotics.” [Online]. Available: https://www.halodi.com/ever3

[3] M. Moeini, “Dynamics Analysis for a 3-RPS1 Parallel Manipulator Wearable Thimble,” p. 28.

[4] T. Koritnik, T. Bajd, and M. Munih, “A Simple Kinematic Model of a Human Body for Virtual Environments,” 2010, pp. 401–408. doi: 10.1007/978-90-481-9262-5_43.

[5] Xsens technologies, “MVN User Manual,” Apr. 01, 2021. https://www.xsens.com/hubfs/Downloads/usermanual/MVN_User_Manual.pdf (accessed Jul. 29, 2021).

[6] “Most Common Physical Therapy Issues in NYC — HI Physio Advance Physio NYC,” *HI Physio*. [Online]. Available: https://hiphysio.com/most-common-physical-therapy-issues-in-nyc

[7] A. D’Souza, S. Vijayakumar, and S. Schaal, “Learning inverse kinematics,” in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, 2001, vol. 1, pp. 298–303. doi: 10.1109/IROS.2001.973374.

[8] M. Arduengo, A. Arduengo, A. Colomé, J. Lobo-Prat, and C. Torras, “A Robot Teleoperation Framework for Human Motion Transfer,” *arXiv*, Apr. 2019, [Online]. Available: http://arxiv.org/abs/1909.06278

[9] K. Darvish *et al.*, “Whole-Body Geometric Retargeting for Humanoid Robots,” in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, Oct. 2019, pp. 679–686. doi: 10.1109/Humanoids43949.2019.9035059.

[10] C. Kim, D. Kim, and Y. Oh, “Solving an inverse kinematics problem for a humanoid robot\u2019s imitation of human motions using optimization.,” Jan. 2005, pp. 85–92.

[11] K. Schindler, L. van Gool, and B. de Gelder, “Recognizing emotions expressed by body pose: A biologically inspired neural model,” *Neural Networks*, vol. 21, no. 9, pp. 1238–1246, Nov. 2008, doi: 10.1016/j.neunet.2008.05.003.

[12] Michael Black, “Breathing Life into Shape (SIGGRAPH 2014).” May 2014. [Online]. Available: https://www.youtube.com/watch?v=4CRMgPcgKp4

[13] “ANA Avatar XPRIZE | XPRIZE Foundation.” https://avatar.xprize.org/prizes/avatar (accessed Apr. 19, 2021).

[14] “RAM Wiki.” [Online]. Available: https://wiki.ram.eemcs.utwente.nl/index.php/Main_Page

[15] A. Mader and W. Eggink, “A design process for Creative Technology,” *Proceedings of the 16th International Conference on Engineering and Product Design Education: Design Education and Human Technology Relations, E and PDE 2014*, no. September, pp. 568–573, 2014.

[16] "Virtuose™ 6D - HAPTION SA." [Online]. Available: https://www.haption.com/de/products-de/virtuose-6d-de.html

[17] "HGlove™ - HAPTION SA." [Online]. Available: https://www.haption.com/de/products-de/hglove-de.html

[18] "ROS.org Powering the world's robots." [Online]. Available: https://www.ros.org/

[19] "What is Kinect? - Definition from WhatIs.com," *SearchHealthIT*. [Online]. Available: https://searchhealthit.techtarget.com/definition/Kinect

[20] W. Song, X. Guo, F. Jiang, S. Yang, G. Jiang, and Y. Shi, "Teleoperation humanoid robot control system based on kinect sensor," *Proceedings of the 2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2012*, vol. 2, pp. 264–267, 2012, doi: 10.1109/IHMSC.2012.159.

[21] S. Mukherjee, D. Paramkusam, and S. K. Dwivedy, "Inverse kinematics of a NAO humanoid robot using kinect to track and imitate human motion," Apr. 2015. doi: 10.1109/RACE.2015.7097245.

[22] L. Fritsche, F. Unverzag, J. Peters, and R. Calandra, "First-person tele-operation of a humanoid robot," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Nov. 2015, pp. 997–1002. doi: 10.1109/HUMANOIDS.2015.7363475.

[23] "Comparing Motion Capture Techniques for Movement Art." [Online]. Available: https://studio.knightlab.com/results/oscillations-vr/oscillations-mocap-comparison/

[24] W. Khalil and E. Dombre, "Chapter 1 - Terminology and general definitions," in *Modeling, Identification and Control of Robots*, W. Khalil and E. Dombre, Eds. Oxford: Butterworth-Heinemann, 2002, pp. 1–12. doi: 10.1016/B978-190399666-9/50001-4.

[25] "Kinematics – Design of Mechanisms: Degrees of Freedom." Sep. 2008. [Online]. Available: https://www.brighthubengineering.com/machine-design/6634-kinematics-design-of-mechanisms-degrees-of-freedom/

[26] "Chapter 4. Basic Kinematics of Constrained Rigid Bodies." [Online]. Available: https://www.cs.cmu.edu/~rapidproto/mechanisms/chpt4.html

[27] "Halodi Robot Models." Halodi Robotics, Jul. 2021. [Online]. Available: https://github.com/Halodi/halodi-robot-models/blob/1a11056bd7bec48ca3198d53c6b36d0b4d1436d9/eve_r3_description/urdf/eve_r3.urdf

[28] "Gazebo : Tutorial : URDF in Gazebo." [Online]. Available: http://gazebosim.org/tutorials/?tut=ros_urdf

[29] "Task and Configuration space Robot Academy." Mar. 2017. [Online]. Available: https://robotacademy.net.au/lesson/task-and-configuration-space/

[30] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, "Inverse Kinematics Techniques in Computer Graphics: A Survey," *Computer Graphics Forum*, vol. 37, no. 6, pp. 35–58, Apr. 2018, doi: 10.1111/cgf.13310.

[31] "How to Calculate a Robot's Forward Kinematics in 5 Easy Steps." [Online]. Available: https://blog.robotiq.com/how-to-calculate-a-robots-forward-kinematics-in-5-easy-steps

[32] C. Stanton, A. Bogdanovych, and E. Ratanasena, “Teleoperation of a humanoid robot using full-body motion capture, example movements, and machine learning.” [Online]. Available: http://www.measurand.com/shapetape.htm

[33] A. Sripada *et al.*, “Teleoperation of a Humanoid Robot with Motion Imitation and Legged Locomotion,” in *2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM)*, Jul. 2018, pp. 375–379. doi: 10.1109/ICARM.2018.8610719.

[34] J. v Nunez, A. Briseno, D. A. Rodriguez, J. M. Ibarra, and V. M. Rodriguez, “Explicit Analytic Solution for Inverse Kinematics of Bioloid Humanoid Robot,” in *2012 Brazilian Robotics Symposium and Latin American Robotics Symposium*, Oct. 2012, pp. 33–38. doi: 10.1109/SBR-LARS.2012.62.

[35] N. Kofinas, E. Orfanoudakis, and M. G. Lagoudakis, “Complete analytical inverse kinematics for NAO,” in *2013 13th International Conference on Autonomous Robot Systems*, Apr. 2013, pp. 1–6. doi: 10.1109/Robotica.2013.6623524.

[36] P. Donelan, “Kinematic Singularities of Robot Manipulators,” 2010. doi: 10.5772/9548.

[37] M. Meredith and S. Maddock, “Real-time inverse kinematics: The return of the Jacobian,” Jan. 2004.

[38] L. Rapetti *et al.*, “Model-Based Real-Time Motion Tracking Using Dynamical Inverse Kinematics,” *Algorithms*, vol. 13, no. 10, p. 266, Apr. 2020, doi: 10.3390/a13100266.

[39] “Neuro-Adaptive Learning and ANFIS - MATLAB & Simulink - MathWorks Benelux.” [Online]. Available: https://nl.mathworks.com/help/fuzzy/neuro-adaptive-learning-and-anfis.html

[40] “Vicon Award Winning Motion Capture Systems,” *Vicon*. [Online]. Available: https://www.vicon.com/

[41] R. P. Khurshid and K. J. Kuchenbecker, “Data-Driven Motion Mappings Improve Transparency in Teleoperation,” *Presence: Teleoperators and Virtual Environments*, vol. 24, no. 2, pp. 132–154, May 2015, doi: 10.1162/PRES_a_00223.

[42] “Fuzzy Logic Toolbox.” [Online]. Available: https://nl.mathworks.com/products/fuzzy-logic.html

[43] A. Beck, B. Stevens, K. A. Bard, and L. Cañamero, “Emotional body language displayed by artificial agents,” *ACM Transactions on Interactive Intelligent Systems*, vol. 2, no. 1, pp. 2:1–2:29, Mar. 2012, doi: 10.1145/2133366.2133368.

[44] Z. Zhang, Y. Niu, S. Wu, S. Lin, and L. Kong, “Analysis of Influencing Factors on Humanoid Robots’ Emotion Expressions by Body Language,” in *Advances in Neural Networks – ISNN 2018*, vol. 10878, T. Huang, J. Lv, C. Sun, and A. v Tuzikov, Eds. Cham: Springer International Publishing, 2018, pp. 775–785. doi: 10.1007/978-3-319-92537-0_88.

[45] M. Zheng and M. Q.-H. Meng, “Designing gestures with semantic meanings for humanoid robot,” in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec. 2012, pp. 287–292. doi: 10.1109/ROBIO.2012.6490981.

[46] T. Schulz, J. Torresen, and J. Herstad, “Animation Techniques in Human-Robot Interaction User Studies: a Systematic Literature Review,” *arXiv*, Dec. 2018, doi: 10.1145/3317325.

[47] T. Ribeiro and A. Paiva, “The illusion of robotic life: Principles and practices of animation for robots,” in *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Mar. 2012, pp. 383–390. doi: 10.1145/2157689.2157814.

[48] Terzioğlu et al., “Designing Social Cues for Collaborative Robots Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction.” [Online]. Available: https://dl-acm-org.ezproxy2.utwente.nl/doi/abs/10.1145/3319502.3374829

[49] A. Tsoli, N. Mahmood, and M. J. Black, “Breathing life into shape: capturing, modeling and animating 3D human breathing,” *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 52:1–52:11, Jul. 2014, doi: 10.1145/2601097.2601225.

[50] Xsens, “Software downloads.” [Online]. Available: https://www.xsens.com/software-downloads

[51] “Tune Sugeno-type fuzzy inference system using training data - MATLAB anfis - MathWorks Benelux.” [Online]. Available: https://nl.mathworks.com/help/fuzzy/anfis.html

[52] “Fuzzy Inference Process - MATLAB & Simulink.” [Online]. Available: https://www.mathworks.com/help/fuzzy/fuzzy-inference-process.html

[53] “Neuro-Adaptive Learning and ANFIS - MATLAB & Simulink.” [Online]. Available: https://www.mathworks.com/help/fuzzy/neuro-adaptive-learning-and-anfis.html

[54] “Foundations of Fuzzy Logic.” [Online]. Available: https://nl.mathworks.com/help/fuzzy/foundations-of-fuzzy-logic.html

[55] “Option set for genfis command - MATLAB genfisOptions.” [Online]. Available: https://www.mathworks.com/help/fuzzy/genfisoptions.html

[56] “Fuzzy Clustering - MATLAB & Simulink.” [Online]. Available: https://www.mathworks.com/help/fuzzy/fuzzy-clustering.html

[57] “Generate fuzzy inference system object from data - MATLAB genfis.” [Online]. Available: https://www.mathworks.com/help/fuzzy/genfis.html

[58] “(To be removed) Generate Fuzzy Inference System structure from data using subtractive clustering - MATLAB genfis2 - MathWorks Benelux.” [Online]. Available: https://nl.mathworks.com/help/fuzzy/genfis2.html

[59] “Find cluster centers using subtractive clustering - MATLAB subclust - MathWorks Benelux.” [Online]. Available: https://nl.mathworks.com/help/fuzzy/subclust.html#bvm9zpz-5

[60] K. Benmouiza and A. Cheknane, “Clustered ANFIS network using fuzzy c-means, subtractive clustering, and grid partitioning for hourly solar radiation forecasting,” *Theoretical and Applied Climatology*, vol. 137, no. 1, pp. 31–43, Jul. 2019, doi: 10.1007/s00704-018-2576-4.

[61] “Model Suburban Commuting Using Subtractive Clustering and ANFIS - MATLAB & Simulink.” [Online]. Available: https://www.mathworks.com/help/fuzzy/model-suburban-commuting-using-subtractive-clustering.html

[62] “Option set for anfis command - MATLAB anfisOptions.” [Online]. Available: https://www.mathworks.com/help/fuzzy/anfisoptions.html

[63] “Generate trajectories with trapezoidal velocity profiles - MATLAB trapveltraj - MathWorks Benelux.” [Online]. Available: https://nl.mathworks.com/help/robotics/ref/trapveltraj.html

[64] “Interact with rigid body tree robot models - MATLAB - MathWorks Benelux.” [Online]. Available: https://nl.mathworks.com/help/robotics/ref/interactiverigidbodytree.html

[65] “Interactively Build a Trajectory for an ABB YuMi Robot - MATLAB & Simulink - MathWorks Benelux.” [Online]. Available: https://nl.mathworks.com/help/robotics/ug/interactively-build-a-trajectory-abb-yumi.html

Appendix A: ANFIS training error

```
ANFIS info:
    Number of nodes: 38
    Number of linear parameters: 16
    Number of nonlinear parameters: 24
    Total number of parameters: 40
    Number of training data pairs: 489720
    Number of checking data pairs: 0
    Number of fuzzy rules: 4


Start training ANFIS ...

1    0.0738706
2    0.064934
3    0.0550998
4    0.0470065
Step size increases to 0.011000 after epoch 5.
5    0.0405036
6    0.0353633
7    0.0309028
8    0.0274181
Step size increases to 0.012100 after epoch 9.
9    0.0247041
10   0.0225789

Designated epoch number reached. ANFIS training completed at epoch 10.

Minimal training RMSE = 0.0225789
```

ANFIS training for joint z

```
ANFIS info:
    Number of nodes: 30
    Number of linear parameters: 12
    Number of nonlinear parameters: 18
    Total number of parameters: 30
    Number of training data pairs: 489720
    Number of checking data pairs: 0
    Number of fuzzy rules: 3


Start training ANFIS ...

1    0.0721219
2    0.0644714
3    0.0552745
4    0.0468706
Step size increases to 0.011000 after epoch 5.
5    0.039858
6    0.0342347
7    0.0294566
8    0.0258836
Step size increases to 0.012100 after epoch 9.
9    0.0232327
10   0.021242

Designated epoch number reached. ANFIS training completed at epoch 10.

Minimal training RMSE = 0.021242
```

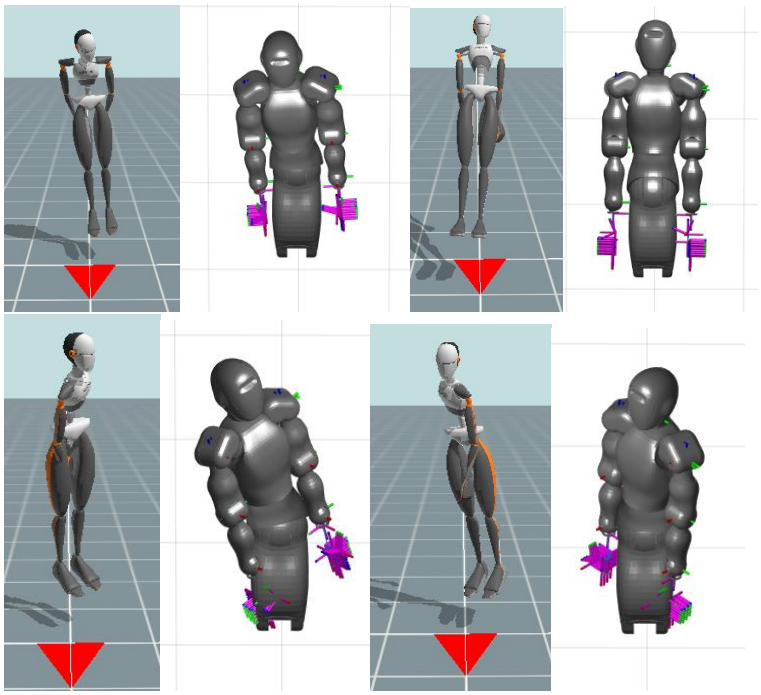
ANFIS training for joint y

```
ANFIS info:
    Number of nodes: 46
    Number of linear parameters: 20
    Number of nonlinear parameters: 30
    Total number of parameters: 50
    Number of training data pairs: 489720
    Number of checking data pairs: 0
    Number of fuzzy rules: 5


Start training ANFIS ...

1    0.00793833
2    0.00760698
3    0.00728597
4    0.00697518
Step size increases to 0.011000 after epoch 5.
5    0.00667446
6    0.00638365
7    0.00607511
8    0.00577834
Step size increases to 0.012100 after epoch 9.
9    0.00549331
10   0.00522011

Designated epoch number reached. ANFIS training completed at epoch 10.

Minimal training RMSE = 0.00522011
```

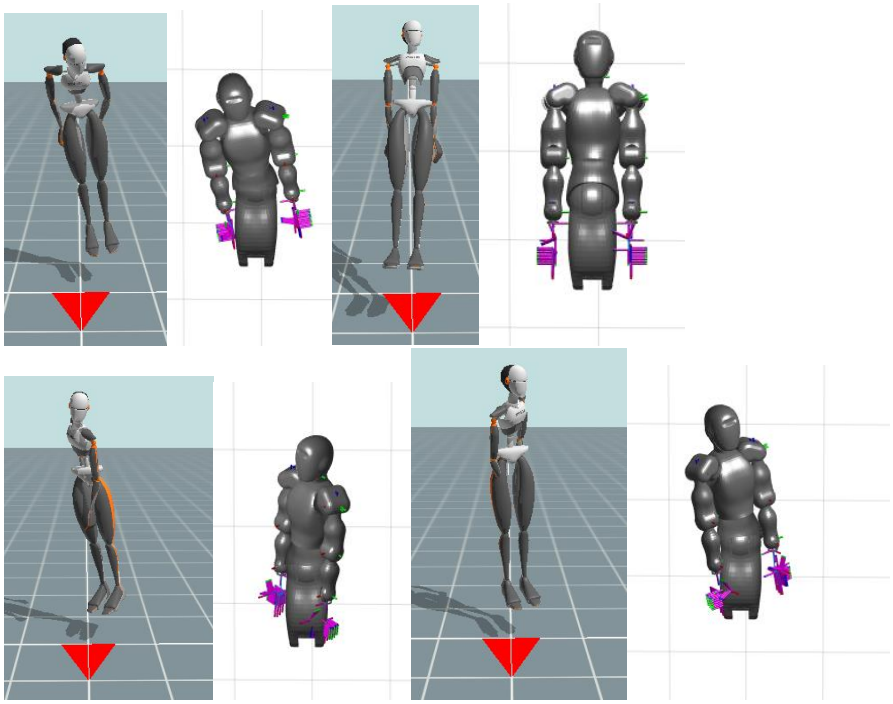
ANFIS training for joint x

Appendix B: Screenshots of ANFIS motion mapping and direct angle mapping

ANFIS motion mapping:



Direct angle mapping:



Appendix C: MATLAB code – Data generation

```
%%import robot
robot = importrobot('customEVE.urdf','MeshPath','meshes');
```

```
%% Data generation: Joint values
hip_x_values = -0.523599000000000: 0.02: 0.523599000000000; % +/- 30 deg
hip_y_values =-1.570000000000000: 0.02: 0.174533000000000; % + 10/- 90 deg
hip_z_values = -1.047200000000000: 0.02: 1.047200000000000; % +/- 60 deg
```

```
%% Data generation: Orientation

left_r = [];
left_p = [];
left_y = [];
x_joint = [];
y_joint = [];
z_joint = [];
%iterate through all joint values
for x = hip_x_values
    for y = hip_y_values
        for z = hip_z_values
            c = homeConfiguration(robot);
            %assign joint angles to robot configuration
            c(1).JointPosition = y;
            c(2).JointPosition = x;
            c(3).JointPosition = z;
            %calculate transformation matrix
            tf_1 = getTransform(robot,c,'pelvis');
            %extract rotation matrix
            rotm = tform2rotm(tf_1);
            % tranform to Euler angles
            [yaw, pitch, roll] = dcm2angle(rotm);
            %create 6 data sets
            left_y(end +1) = yaw;
            left_p(end +1) = pitch;
            left_r(end +1) = roll;

            x_joint(end +1) = x;
            y_joint(end +1) = y;
            z_joint(end +1) = z;

        end
    end
end
```

```
%% Data generation: input and output for Fuzzy inteference system
data_hip_in = [left_r(:) left_p(:) left_y(:)];
data_hipx_out = x_joint(:);
data_hipy_out = y_joint(:);
data_hipz_out = z_joint(:);
```

Appendix D: MATLAB code – ANFIS training and validation

```
%% Training of ANFIS - y joint and validation
%create FIS with subtractive clustering
fis_y = genfis2(data_hip_in,data_hipy_out,0.7);
% create ANFIS with initial FIS
opt = anfisOptions('InitialFIS',fis_y);
outFIS_y = anfis([data_hip_in data_hipy_out],opt);
%ANFIS validation
anfisOutput_y = evalfis(outFIS_y,data_hip_in);
ded_y = data_hipy_out -  anfisOutput_y;
plot(ded_y);
title("Error plot: Initial y joint values - predicted y joint values");
ylabel("error");
xlabel("N angle");
```

```
%% Training of ANFIS - z joint and validation
%create FIS with subtractive clustering
fis_z = genfis2(data_hip_in,data_hipz_out,0.7);
% create ANFIS with initial FIS
opt = anfisOptions('InitialFIS',fis_z);
outFIS_z = anfis([data_hip_in data_hipz_out],opt);
%ANFIS validation
anfisOutput_z = evalfis(outFIS_z,data_hip_in);
ded_z = data_hipz_out -  anfisOutput_z;
plot(ded_z);
title("Error plot: Initial z joint values - predicted z joint values");
ylabel("error");
xlabel("N angle");
```

```
%% Training of ANFIS - x joint and validation
%create FIS with subtractive clustering
fis_x = genfis2(data_hip_in,data_hipx_out,0.7);
% create ANFIS with initial FIS
opt = anfisOptions('InitialFIS',fis_x);
outFIS_x = anfis([data_hip_in data_hipx_out],opt);
%ANFIS validation
anfisOutput_x = evalfis(outFIS_x,data_hip_in);
ded_x = data_hipx_out -  anfisOutput_x;
plot(ded_x);
title("Error plot: Initial x joint values - predicted x joint values");
ylabel("error");
xlabel("N angle");
```

Appendix E: MATLAB code – Breathing animation

```
%% Breathing animation

%Create and store configurations using
%inte = interactiveRigidBodyTree(robot);
%addConfiguration(inte);
numSamples = 30*size(inte.StoredConfigurations, 2) ;
[q,~,~, tvec] = trapveltraj(inte.StoredConfigurations,numSamples);
% q is a matrix that represents a series of joint configurations that move between each pose,
%tvec is a vector/array of the time samples at which the output position q is sampled

for m = 1:5 % 3 times breathing
    rateCtrlObj = rateControl(25); %control breathing speed
    for i = 1:numSamples
        c1 = homeConfiguration(robot);
        for b = 1:18
            c1(b).JointPosition = q(b,i);
        end
        show(robot,c1,'PreservePlot',0, 'FastUpdate', 1);
        waitfor(rateCtrlObj);
    end
end
```

Appendix F: MATLAB code – Motion mapping + breathing

```
%% Streaming Definitions (adopted from base.xsens.com/knowledgebase)

% Definition of the Streaming Parameters (change these to reflect your own settings)
port = 9763;        % port number as defined in the Network Streamer in MVN
timeout = 3000;     % maximum time to wait to receive a package in miliseconds
segmentId = 2;      % Segment ID -> stern/chest

% Parameters to run the code
lastReceived = NaN;
counter = 1;
maxPacketLength = 2000;
```

```
%% Rate control object and robot simulation initiation

config = homeConfiguration(robot);
show(robot,config,'PreservePlot',0, 'FastUpdate', 1)
r = rateControl(2000);
reset(r)
n = 0;

newPacketFlag = 1;
```

```
%% ANFIS motion mapping (adopted from base.xsens.com/knowledgebase, adjusted version)

while 1

    try
        % Kevin Bartlett (2020). A simple UDP communications application
        %(https://www.mathworks.com/matlabcentral/fileexchange/24525-a-simple-udp-communications-application),
        %MATLAB Central File Exchange. Retrieved May 1, 2020.
        message = judp('receive', port, 8*maxPacketLength, timeout);
    catch e
        disp(e.message)
        continue
    end

    % conversion of the UDP messages into parameters
    [pos, ori, lastReceived, timeCode, newPacketFlag] = parse_position_packet(message, lastReceived);

    % Skip unknown packets
    if newPacketFlag ~=0

        % transform orientation in quarternions to Euler angles
        [ xsens_ya, xsens_p,  xsens_r] = quat2angle([ ori(segmentId,1)  ori(segmentId,2)  ori(segmentId,3)  ori(segmentId,4)]);
        % combine angles to 1 data set
        input = [ (xsens_r)  (xsens_p)  (xsens_ya)];
        % calculate joint positon with ANFIS
        valx = evalfis(outFIS_x, input);
        valy = evalfis(outFIS_y, input);
        valz = evalfis(outFIS_z, input);
        % Apply joint positions to the robot configuration
        config(1).JointPosition = valy *-1;
        config(2).JointPosition =valx *-1;
        config(3).JointPosition = valz *-1;

        show(robot,config,'PreservePlot',0, 'FastUpdate', 1);

        %     tf_1 = getTransform(robot,config,'pelvis');
        %     rotm = tform2rotm(tf_1);
        %     [yaw, pitch, roll] = dcm2angle(rotm);
        %     tf_2 = getTransform(robot,config,'l_shoulder');
        %      rotm2 = tform2rotm(tf_2);
        %     [yaw2, pitch2, roll2] = dcm2angle(rotm2);

        %  hold all;
        %   ylim([-0.8 0.8]);
        %     plot(n, pitch2 *-1 , '.b');
        %     plot(n, pitch * -1, '.r');
        %     title( "pitch orientation xsens vs. robot");
        %     ylabel("angle in rad");
        %     xlabel("sample N");
        %     pause(0.01);
        %      n= n+1;

        waitfor(r);
    end
    drawnow
end
```

```
%% Direct angle mapping (adopted from base.xsens.com/knowledgebase, adjusted version)

while 1

    try
        % Kevin Bartlett (2020). A simple UDP communications application
        %(https://www.mathworks.com/matlabcentral/fileexchange/24525-a-simple-udp-communications-application),
        %MATLAB Central File Exchange. Retrieved May 1, 2020.
        message = judp('receive', port, 8*maxPacketLength, timeout);
    catch e
        disp(e.message)
        continue
    end

    % conversion of the UDP messages into parameters
    [pos, ori, lastReceived, timeCode, newPacketFlag] = parse_position_packet(message, lastReceived);

    % Skip unknown packets
    if newPacketFlag ~=0

        [ xsens_ya, xsens_p,  xsens_r] = quat2angle([ ori(segmentId,1)  ori(segmentId,2)  ori(segmentId,3)  ori(segmentId,4)]);
        config(1).JointPosition = xsens_p;
        config(2).JointPosition =xsens_r;

        config(3).JointPosition = xsens_ya ;

        show(robot,config,'PreservePlot',0, 'FastUpdate', 1);
        waitfor(r);
    end
    drawnow
end
```

```
%% Breathing animation integrated into motion mapping (Version 1)
%(adopted from base.xsens.com/knowledgebase, adjusted version)
r_old =0;
y_old =0;
p_old =0;
timr =0;
timr2 = 0;
figure

while 1

    try
        % Kevin Bartlett (2020). A simple UDP communications application
        %(https://www.mathworks.com/matlabcentral/fileexchange/24525-a-simple-udp-communications-application),
        %MATLAB Central File Exchange. Retrieved May 1, 2020.
        message = judp('receive', port, 8*maxPacketLength, timeout);
    catch e
        disp(e.message)
        continue
    end

    % conversion of the UDP messages into parameters
    [pos,ori,sampleCounter,timeCode,newPacketFlag] = parse_position_packet(message,lastReceived);

    % Skip unknown packets
    if newPacketFlag ~=0
```

```
[ xsens_ya, xsens_p,  xsens_r] = quat2angle([ ori(segmentId,1)  ori(segmentId,2)  ori(segmentId,3)  ori(segmentId,4)]);
%capture old orientation values
if ( timr == 0)
    r_old = xsens_r;
    p_old = xsens_p;
    y_old = xsens_ya;
end
%start timer
timr = timr +1;
% if timer counts more than 10 frames, calculate velocity
if (timr >= 10)
    vel = 10* (norm( [ y_old p_old r_old] - [xsens_ya xsens_p  xsens_r] )/timr);
    timr = 0; %reset timer
end
disp(timr2);
% if threshold is reached, start second timer, otherwise reset second
% timer
if (vel < 0.003 && timr2 < 10000)
    timr2 = timr2 +1;
else
    if(timr2 < 10000)
        timr2 = 0;
    end
end
```

```
if (timr2 >= 15 && timr2 < 10000)
    % store current joint configuration to be applied for the breathing
    % animation
    for b = 1:18
        curr(b,1)= config(b).JointPosition;
    end

    % if the first animation is used, replace [curr curr +inte.StoredConfigurations(:,2) curr] by
    %[curr curr+inte.StoredConfigurations(:,1)+inte.StoredConfigurations(:,2) +inte.StoredConfigurations(:,3)]

    numSamples = 15*size([curr curr +inte.StoredConfigurations(:,2) curr], 2) ;
    [q,~,~, tvec] = trapveltraj([curr (curr +inte.StoredConfigurations(:,2)) curr],numSamples);

    for i = 1:numSamples
        inte.Configuration = q(:,i);
        c1 = homeConfiguration(robot);
        for b = 1:18
            c1(b).JointPosition = inte.Configuration(b);
        end
        % REPLACE above by: (+ add individual rate control object to control speed)
        %for i = 1:numSamples
        % c1 = homeConfiguration(robot);
        %     for b = 1:18
        %     c1(b).JointPosition = q(b,i);
        %     end
```

```
            show(robot,c1,'PreservePlot',0, 'FastUpdate', 1);
            waitfor(r);
        end
        %after animation, set second timer to 1000 to create a transition
        %back to motion mapping
        timr2 =10000;

    elseif  (timr2 == 10000)
        config(1).JointPosition = xsens_p;
        config(2).JointPosition =xsens_r;
        config(3).JointPosition = xsens_ya ;
        for b = 1:18
            curr(b,1)= config(b).JointPosition;
        end
        for b = 1:18
            last(b,1)= c1(b).JointPosition;
        end

        numSamples = 15*size([last curr], 2) ;
        [q,~,~, tvec] = trapveltraj([last curr],numSamples);
        for i = 1:numSamples
            inte.Configuration = q(:,i);
            for b = 1:18
                c1(b).JointPosition = inte.Configuration(b);
            end
            show(robot,c1,'PreservePlot',0, 'FastUpdate', 1);
            waitfor(r);
        end
        %set second timer back to 0 to enable a new breathing motion
        timr2 = 0;

    else
        %if motion capture data is above threshold/ no breathing
        config(1).JointPosition = xsens_p;
        config(2).JointPosition =xsens_r;
        config(3).JointPosition = xsens_ya ;

        show(robot,config,'PreservePlot',0, 'FastUpdate', 1);
        waitfor(r);


    end
  end
end
```

```
%% Breathing animation integrated into motion mapping (Version 2)
%(adopted from base.xsens.com/knowledgebase, adjusted version)
r_old =0;
y_old =0;
p_old =0;
timr =0;

while 1

    try
        % Kevin Bartlett (2020). A simple UDP communications application
        %(https://www.mathworks.com/matlabcentral/fileexchange/24525-a-simple-udp-communications-application),
        %MATLAB Central File Exchange. Retrieved May 1, 2020.
        message = judp('receive', port, 8*maxPacketLength, timeout);
    catch e
        disp(e.message)
        continue
    end

    % conversion of the UDP messages into parameters
    [pos,ori,sampleCounter,timeCode,newPacketFlag] = parse_position_packet(message,lastReceived);

    % Skip unknown packets
    if newPacketFlag ~=0

        [ xsens_ya, xsens_p,  xsens_r] = quat2angle([ ori(segmentId,1)  ori(segmentId,2)  ori(segmentId,3)  ori(segmentId,4)]);
```

```
% if distance is below threshold, start timer and count
if   abs(norm([xsens_ya xsens_p  xsens_r] - [ y_old p_old r_old])) < 0.01 && timr < 10000
    timr = timr +1;
else
    if(timr < 10000)
        timr = 0;
    end
end
if (timr >= 70 &&timr < 10000 )
    for b = 1:18
        curr(b,1)= config(b).JointPosition;
    end
    numSamples = 15*size([ curr+inte.StoredConfigurations(:,1) curr +inte.StoredConfigurations(:,2) curr +inte.StoredConfigurations(:,3)], 2) ;
    [q,~,~, tvec] = trapveltraj([ curr+inte.StoredConfigurations(:,1) (curr +inte.StoredConfigurations(:,2)) curr +inte.StoredConfigurations(:,3)],numSamples);
    for i = 1:numSamples
        inte.Configuration = q(:,i);
        c1 = homeConfiguration(robot);
        for b = 1:18
            c1(b).JointPosition = inte.Configuration(b);
        end
        % REPLACE above by: (+ add individual rate control object to control speed)
        %for i = 1:numSamples
        % c1 = homeConfiguration(robot);
        %     for b = 1:18
        %     c1(b).JointPosition = q(b,i);
        %     end
        show(robot,c1,'PreservePlot',0, 'FastUpdate', 1);
```

```
            waitfor(r);
        end
        %after animation, set timer to 1000 to create a transition
        %back to motion mapping
        timr =10000;


    elseif  (timr == 10000)

        config(1).JointPosition = xsens_p;
        config(2).JointPosition =xsens_r;
        config(3).JointPosition = xsens_ya ;
        for b = 1:18
            curr(b,1)= config(b).JointPosition;
        end
        for b = 1:18
            last(b,1)= c1(b).JointPosition;
        end
        numSamples = 12*size([last curr], 2) ;
        [q,~,~, tvec] = trapveltraj([last curr],numSamples);
        for i = 1:numSamples
            inte.Configuration = q(:,i);
            for b = 1:18
                c1(b).JointPosition = inte.Configuration(b);
            end
            show(robot,c1,'PreservePlot',0, 'FastUpdate', 1);
            waitfor(r);
        end
        timr= 0;

    else
        config(1).JointPosition = xsens_p;
        config(2).JointPosition =xsens_r;
        config(3).JointPosition = xsens_ya ;

        show(robot,config,'PreservePlot',0, 'FastUpdate', 1);
        waitfor(r);


    end
    r_old = xsens_r;
    p_old = xsens_p;
    y_old = xsens_ya;

end
end
```