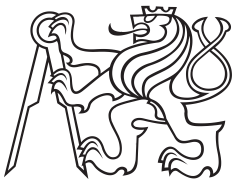
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Real-Time Teleoperation of a Robot Arm for Self-Contact

Bc. Adam Rojík

Supervisor: doc. Mgr. Matěj Hoffmann, Ph.D.
Supervisor–specialist: MSc. Jason Khoury
Study program: Cybernetics and Robotics
May 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Rojík Adam** Personal ID number: **466026**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Real-Time Teleoperation of a Robot Arm for Self-Contact

Master’s thesis title in Czech:

Teleoperace robotické ruky pro sebe-kontakt v reálném čase

Guidelines:

The mechanisms of how humans localize touch on their bodies are not fully understood. To increase understanding, specific manipulations of tactile localization are needed. One possibility is to exploit self-contact when the human is sliding over its skin surface, but insert a robot arm in the middle [CAT22]. However, state-of-the-art studies lack the ecological conditions of free movement. A motion capture system (Qualisys) coupled with a teleoperated robotic manipulator [KIN3] equipped with an artificial finger will bridge the gap and allow more freedom to study the influence of proprioception vs. touch during self-touch by changing forward kinematics parameters [PAT12].
Instructions:
1. Familiarize yourself with the Kinova Gen3 robotic platform [KIN3], Kortex API [KOR] and Qualisys motion tracking system [QUA].
2. Familiarize yourself with the psychological experiment procedures [CAT22].
3. Develop an interface for real-time control of the robot end effector position based on the arm joint positions and transform it into position/speed commands. Optimize the robot’s control so it performs movement with the lowest possible lag.
4. Assess and ensure the safety of this application.
5. Set up a pilot tactile localization experiment and logging of the results.
6. Optionally, conduct and evaluate experiments (with your supervisors).
7. Optionally, create a graphical interface for running the experiments with human participants.

Bibliography / sources:

[1] [CAT22] Cataldo, A., Dupin, L., Dempsey-Jones, H., Gomi, H., & Haggard, P. (2022). Interplay of tactile and motor information in constructing spatial self-perception. Current Biology, 32(6), 1301-1309.
[2] [PAT12] Patane, L., Sciutti, A., Berret, B., Squeri, V., Masia, L., Sandini, G., & Nori, F. (2012, June). Modeling kinematic forward model adaptation by modular decomposition. In 2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob) (pp. 1252-1257). IEEE.
[3] [KIN3] Kinova gen3: https://www.kinovarobotics.com/product/gen3-robots
[4] [KOR] https://github.com/Kinovarobotics/kortex
[5] [QUA] Qualisys system - https://www.qualisys.com/cameras/, https://docs.qualisys.com/qtm-rt-protocol/

Name and workplace of master's thesis supervisor:

doc. Mgr. Matěj Hoffmann, Ph.D. Vision for Robotics and Autonomous Systems FEE

Name and workplace of second master's thesis supervisor or consultant:

MSc. Jason Khoury Vision for Robotics and Autonomous Systems FEE

Date of master's thesis assignment: **01.02.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Mgr. Matěj Hoffmann, Ph.D. Supervisor's signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

First, I would like to thank my supervisor Matěj Hoffmann for his unwavering support and guidance throughout my thesis. He has provided me with an excellent topic for the thesis and inspired me with a passion for science. The years in the laboratory have been an invaluable gift that I will always cherish.

I am also very grateful to my supervisor specialists and co-working psychologists, Jason Khoury, Valentin Marcel, and Sergiu Tcaci Popescu. Working with them was both enjoyable and educational. Their expertise in the psychological aspects of this work and willingness to discuss technical difficulties greatly enriched my understanding of the topic.

I am grateful to Krištof Pučejdl for his instrumental role in designing and 3D printing an elastic tool for the robot that simulates self-touch on the participant's arm.

My thanks go to Zdeněk Hurák for his key input on system control. His expert advice was instrumental in improving system performance and minimizing control delays.

Lukáš Rustler deserves credit for his critical role in accurately measuring the robot's forces and ensuring its safety, elements that were vital to the project's success.

I want to extend my gratitude to the laboratory team. Their camaraderie made my work experience enjoyable and I'm fortunate to have had the opportunity to meet such incredible individuals.

I would be remiss not to acknowledge the unwavering love and support of my family. I am deeply thankful for their sacrifices and for always being there to cheer me on.

Finally, my sincerest thanks go to everyone who supported me throughout this journey. Your collective contributions have made this accomplishment possible.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 26. května 2023

Abstract

The mechanisms by which humans localize touch on their bodies are not yet fully understood. Specific manipulations of tactile localization are needed to improve understanding. One possibility is to exploit self-contact when humans slide over their skin surface, but use a robotic arm in the middle [1, 2]. However, current state-of-the-art studies lack the ecological conditions of free movement to investigate the influence of proprioception versus touch during self-touch by changing forward kinematic parameters [3]. This gap can be bridged by using a motion capture system in conjunction with a teleoperated robotic manipulator equipped with an artificial finger.

This thesis focuses on the technical part, a real-time teleoperation task with the additional transformation of a motor-to-touch gain, with latency minimized to levels indistinguishable for participants of an experiment as timing is crucial in such studies.

It further emphasizes safety and usability, with thorough risk mitigation and the creation of a simple to use graphical user interface. The effectiveness of the tool was verified through pilot experiments, validating its potential to advance research in human sensory perception, which has a potential impact on the fields of human-robot interaction, prosthetic design, and rehabilitation interventions.

Keywords: real-time teleoperation, kinematics, self-contact, human-robot interaction, tactile feedback, motion capture, safety in robotics, robot-mediated self-touch

Supervisor: doc. Mgr. Matěj Hoffmann, Ph.D.

Abstrakt

Mechanismy, kterými lidé lokalizují dotek na svém těle, nejsou dosud zcela objasněny. Jedním ze způsobů, jak prohloubit naše chápání hmatové lokalizace, je zavést roboticky zprostředkovaný sebedotek (tj. dotyk směrem k vlastnímu tělu), aby se pohyby oddělily od jejich hmatových důsledků [1, 2]. Dosavadní studie však postrádají vhodné podmínky volného pohybu, aby bylo možné zkoumat vliv propriocepce oproti dotyku při sebedotyku při změně kinematických parametrů [3]. Tento nedostatek lze překlenout použitím systému pro snímání pohybu ve spojení s teleoperovaným robotickým manipulátorem s umělým prstem.

Tato práce se zaměřuje na technickou část, tedy úlohu teleoperace v reálném čase s přidanou transformací mezi pohybem a dotykem, s minimálním zpožděním na úroveň nerozlišitelnou pro účastníky experimentu, protože čas je v takových studiích kritický.

V této práci je kladen důraz na bezpečnost a použitelnost s důrazem na zmírněním rizik a na vytvoření uživatelsky jednoduchého grafického uživatelského rozhraní. Efektivita nástroje byla ověřena pilotními experimenty, které potvrdily jeho potenciál pro rozvoj výzkumu v oblasti lidského smyslového vnímání, což má potenciální dopad na oblasti interakce člověka s robotem, navrhování protéz a na rehabilitační intervence.

Klíčová slova: teleoperace v reálném čase, kinematika, sebe-kontakt, interakce člověk-robot, hmatová zpětná vazba, snímání pohybu, bezpečnost v robotice, dotyk sám sebe zprostředkovaný robotem

Překlad názvu: Teleoperace robotické ruky pro sebe-kontakt v reálném čase

Contents

Glossary

GUI graphical user interface. 15–17, 21–26, 32–34, 40, 42, 43, 47, 49, 54

IK inverse kinematics. 8, 9, 14, 27, 31, 36, 41

Chapter 1

Introduction

1.1 Motivation

The mechanisms by which humans localize touch on their bodies are not yet fully understood. Specific manipulations of tactile localization are needed to improve understanding. One possibility is to exploit self-contact when humans slide over their skin surface, but use a robotic arm in the middle [1, 2]. However, current state-of-the-art studies lack the ecological conditions of free movement to investigate the influence of proprioception versus touch during self-touch by changing forward kinematic parameters [3]. This gap can be bridged by using a motion capture system in conjunction with a teleoperated robotic manipulator equipped with an artificial finger.

This work focuses on the technical part, a real-time teleoperation task with the additional transformation of a motor-to-touch gain; for example, a gain of 1.5 would mean that a movement of the arm by 1 cm causes the robot to move 1.5 cm. In this context, minimizing the delay between the participant's movements and the robot manipulator's responses is critical so that they are virtually imperceptible. This principle is similar to the rubber hand illusion [4], where the touch sensation on both hands must occur simultaneously to produce a coherent experience. This synchronization leads to a perception in which it feels as if the moving hand is the source of the touch sensation from the other hand.

Applying the same concept should create a seamless and immersive teleoperation experience where the user's actions and the robot's responses feel like one continuous motion. Furthermore, it should be noted that humans cannot perceive differences of less than 12.5 ms that appear to them as synchronous [5].

The program developed as part of this thesis provides psychologists with a valuable tool to study and gain knowledge about how people perceive and interpret tactile information. Understanding how humans process touch can foster the development of technologies and interventions to improve human-robot interaction, prosthetic design, and rehabilitation approaches. This research contributes to a deeper understanding of the human sensory system and its applications in various fields.

1.2 Objectives

The primary goals of this work were to provide a framework for the experimental psychologists working in the team and other cognitive scientists to conduct an experiment in which the robot acts as a proxy for the self-touch behavior of the participant. Its functionality was subject to the following constraints:

- It is an interface for real-time teleoperation of a robot arm via feedback from a motion capture system.
- The control of the robot is optimized so that the delay is not perceptible to the participant.
- Evaluation and assurance of the safety of the application.
- Optional execution and evaluation of the experiments with supervisors.
- Optional creation of a graphical interface for the execution of the experiments.

Chapter 2

Related work

2.1 Psychology behind the experiment

The basis of this work was to replicate the experimental paradigm that was used in two experimental studies by Cataldo et al. [2, 1]. The studies aimed to determine what most influences our spatial perception of the extent of a touch, in this case specifically of a self-touch: the sensations of proprioception (perception of the relative positions of one's body parts) and extent of tactile sensation or the extent of our motor actions. They used robot-mediated self-touch to touch the participant's left arm, as in Figure 2.1. This setup is similar to natural self-touch, for example, when the right hand touched the left arm, where the spatial coupling between movement and touch is fixed, but decoupling of these movements is possible. Two conditions were studied: passive and active. In the passive case, the researchers moved the participant's right hand, which controlled the robot; in the active case, the participant moved it. These movements were in sync with the arm of the robot to which a brush was attached; this brush touched the participant's left arm. The relationship between the the right arm's movement and the touch felt on the left arm was altered by introducing five motor-to-touch gain conditions.

The participants were consistently and automatically affected by the other signal in all conditions. The active movement has more influence on the judgement of the touch extent and is more immune to external influence in case of the movement extent than passive movement. Interestingly, the precision of movement length evaluation seems worse in active compared to passive condition. This suggests that when we move voluntarily, our brain uses several signals to help us perceive the space around us.

In our daily experiences, we frequently adjust our handling of different tools or pointing devices of various sizes and types without requiring extensive experience. These adjustments, as exemplified by the findings of Cataldo's studies [2, 1], can influence our perception of the extent of touch and have a notable impact on our spatial awareness. Patanè's study [3] further complements this understanding by suggesting that humans manage the calculations involved in forward kinematics, specifically the position and orientation of an object based on joint angles and link lengths, by initially learning individual

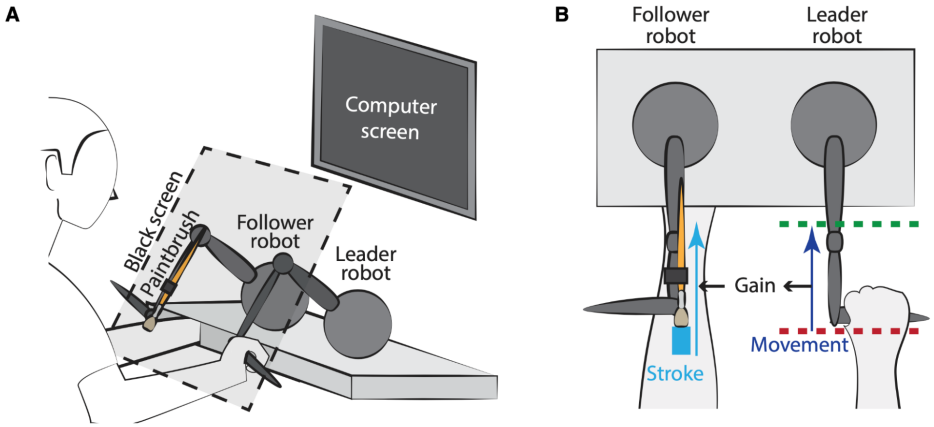


Figure 2.1: The setup of the replicated experiment: (A) shows the leader robot moved by the participants' right hand while feeling a corresponding stroke on the left forearm from a brush attached to the follower robot. A black screen covered the participants' arms and the robotic setup throughout the experiment. (B) The physical extent of the right arm movement was controlled by the position of two programmable "virtual walls" that defined unpredictable start and end positions for each trial. The spatial relation between the extent of movement and touch depended on the gain of the leader: follower robot coupling, which was randomized across trials. From Cataldo [2].

modules, then subsequently adjusting the interrelationships among these modules.

However, this process exhibits varying learning speeds. While the adjustment of the parameters that orchestrate the interaction among the modules is fast, the learning process for individual modules is slower. This could be explained by the presence of intra-module adjustments in our day-to-day activities, which come naturally and are quickly adapted to. On the contrary, inter-module adjustments, which involve coordination between different modules, are harder to encounter naturally and hence take longer to master. This aligns with Cataldo's finding [2, 1] that active movements, or intra-module adjustments, have more influence on our perception and are more resistant to external influence than passive movements.

Moreover, DiMercurio [6] emphasizes the role of self-touch for infants as a critical foundation for developing future behaviors. Given the importance of self-touch and the primary groundwork for this thesis, it was crucial to see how the robot's end-effector could be moved in coordination with the participant's moving arm. The control of the robot is related to teleoperation, described in the next section.

2.2 Teleoperation

This section is divided into two subsections, where the first focuses on the history of teleoperation, and the other on studies with similar experimental

setups.

2.2.1 History of teleoperation

Teleoperation comes from the combination of two words, tele and operation. Tele is a prefix from the Greek word "tēle," that means "far off" or "at a distance", and in English it is used as a prefix to denote something related to distance, remote communication, or transmission over a distance, such as in words such as "television" [7]. The operation has its roots in the Latin word "operātiō," meaning "application of effort, functioning (of natural forces)" [8]. Here, it refers to controlling or manipulating a machine, system, or process. Combined, it represents the act of remotely controlling or operating a machine, system, or process from a distance.

Figure 2.2: The first modern master-slave pantograph made by Goertz at Argonne National Laboratory, from [9].

The first modern master-slave teleoperated system came in the 1940s when Goertz at Argonne National Laboratory made the first pantograph shown in Figure 2.2. Later on, in 1954, they made the first electromechanical manipulator with feedback servo control. Between 1965 and 1971, there was the first attempt to build an exoskeleton by General Electric. Even though it did not succeed, such developments eventually led to the creation of a teleoperated robot arm for wheelchairs, controllable by tongue or other motor signals. The next application was by the US Navy, where they built the CURV, a vehicle to retrieve an accidentally dropped nuclear bomb from the deep bottom of the ocean. In the 1970s, the Soviet's LUNOCHOD project explored instability problems in teleoperation with time delays. Later, devices were built for oil and gas operations under the sea. [9]

This section was a brief introduction to the teleoperation field, which is now expanding much faster. Thus, the next part aims to explore studies only with a similar experimental setup.

2.2.2 Studies with similar experimental setups

A study by Chotiprayanakul [10] in 2009 explored using a sizeable remote robot in hazardous environments with haptic-based force-feedback. Liarokapis [11] in 2013 developed a human-to-Mitsubishi PA-10 robot motion mapping scheme that guarantees anthropomorphism, the attribution of human-like characteristics and behaviors to non-human entities, such as robots. However, their published study does not go into much detail. A year later, Reddivari [12] published an experiment where they controlled a Baxter robot using input from Kinect, which is similar to the goal of this study. To control the robot, they used Python `rospy`; however, they mention the appearance of lag in the robot's motion due to a nonexistent inverse kinematics solution given some combination of the inputs.

In 2016, Maric [13] made RGBD palm tracking working at 125 FPS program for Kinova Jaco 6DOF robot, where they controlled the robots end-effector velocity at 100 Hz using high-level control in ROS provided by Kinova robotics. Although the robot used in this work is Kinova, it cannot be controlled at 100 Hz, as it is a different robot arm with a different API.

Rakita [14] in 2017 used different controllers to control a robot arm in real-time while relaxing constraints on the mapping between a user and the robot's arm. They used the literature for animation, as retargeting motion between characters is a well-studied problem. In their study, they were able to reliably measure the delay to be around 130 ms and 140 ms. Based on the literature they reviewed, it was noted that latency does not become a factor until it becomes much worse than the system's performance. However, their application may not have been as time-critical since their focus was relaxed control for novice users.

In 2018 a study by Zhang [15] made a real-time whole-body task-oriented imitation program with the robot Nao. They had to solve a quadratic programming problem, on average taking 20 ms, with very fast inverse kinematics. But they did not mention lag of the whole system. Similar study, focusing more on the teleoperation and less on IK conducted by Prota [16] in 2022, they controlled a Pepper robot in healthcare settings. The delays induced by the control loop around 200 ms and for the visual feedback it was 100 ms. However, since both studies mention different delays, they are not comparable. For the first study, they did not mention it as a problem and for the latter, they mention it did not cause any problems during the experiments.

Gutzeit [17] had a similar setup to the one used in this work. They used motion capture system to track arm movements and transform those into robot motion by having three markers on the hand in triangular shape, one on the elbow, one on shoulder and three markers on the back as synchronization. The arm was representing the end-effector. However, their main focus was segmenting the movements into a block, where the participant would create the blocks using imitation learning, then motion plan refinement and creating a template from it, thus it was not real-time operation.

Qualisis was also utilized in a study by Sandoval [18], where they created a robot-assisted camera stand during surgery in combination with a Franka

robot controlled at 1 kHz using torque control, controlled via input from the motion capture system at 100 Hz, which is lower than the robot's control frequency. They do not mention the safety of such application, but the robot joints were roughly within $\pm 10^{\circ}$ during the whole experiment.

In addition, Darvish [19] conducted a teleoperation survey, in which they made studies comparable by joint-level control, torque, or position, which may help select the appropriate controls and parameters of the setup. For example, most studies used the joint position when comparing the joint-level control. They also mentioned the variety of teleoperation applications, such as telepresence, teleoperation in hazardous environments, manufacturing & research, telenursing, space applications, and service robotics. However, none of the studies addressed the safety of the application, primarily because there was no direct contact between humans and robots, or the motion was somewhat limited. Another part of a study by Darvish [19] summarizes the findings on latency. Low latency can lead to strong telepresence, also known as tele-embodiment, which is one of the goals of this work. In the studies in this section, the delay averaged about 100 ms. In some studies, the source of the delay was divided into the control delay and the delay of the sensors, such as cameras. The primary source of control delay usually results from the computation of the inverse kinematics, which is highly dependent on the computational speed.

2.3 Inverse kinematics (IK)

The robot's goal position is in Cartesian coordinates. However, the robot is controlled by joint angles, thus there is a need to convert the goal position to robot's joint angles. The process of calculating them is called inverse kinematics. The robot has two groups of control modes that deal with it, high-level and low-level control modes. When controlling a robot in high-level control mode, the robot calculates IK by itself at 40 Hz and the calculation takes place inside the robot. Therefore, any delay affects only the goal position, but not the joint angles, which is better from the safety perspective. However, when faster 1 ms low-level joint control is activated, it no longer uses the robot's end-effector position as a goal. Instead, the robot uses joint angles, leading the robots end-effector to the desired position, which have to be calculated. It is necessary to look at current studies that solve the IK problem.

In 2019, Carpentier [20] made a whole framework, Pinocchio, where they achieved very fast, 1 µs-10 µs computation speed by unrolling most of the calculations during the build of the program. For the computation itself, they used the recursive Newton method. The laptop they used for the speed measurement was equipped with an Intel Core i7 CPU @ 2.4 GHz.

Lloyd [21] in 2022 used a novel approach to solving IK using Halley's Method. In the study, they compared their method with open-source Orocos Kinematics and Dynamics Library (KDL), which is arguably the most popular library used in ROS libraries, ML-BFGS and ML-LMA, which are part

of Mathworks' Matlab Robotics Toolbox and outperformed all of them in robustness and speed. This novel approach is slower when compared to Pinocchio, 50 µs-100 µs, on faster Intel Core i9 CPU @ 2.3 GHz- However, since Pinocchio uses the Newton method as in KDL-NR, which was less robust closer to singularities, it is likely the same for Pinocchio library. Furthermore, the computation speed is below 1 ms, making it sufficient as the robot cannot be controlled any faster.

Given the safety perspective, it makes sense to see what kind of delay is acceptable during a user interaction scenario and if it is possible to utilize high-level control.

2.4 Lag during user interaction

Most of these works were described in Section 2.2, but their main focus was on teleoperation. The quality of interaction with a participant during the experiment is strongly influenced by the delay of the robot during the interaction, as in the case of the rubber hand illusion [4], where synchrony between the touch felt by both arms is required to produce coherent experience. In the replicated study [2], the delay was approximately 2.5 ms.

A study by Jay [22] showed that participants only detect time latencies of 25 ms or longer when haptic feedback is involved. A greater delay in haptic feedback resulted in a greater performance degradation than with visual feedback. Moreover, the more difficult the task, the greater the impairment due to latency.

In a later study by Kaaresoja [23], in which they tested touchscreen display delay, they found that the delay acted as if the button had a weight - the lower the delay, the lower the weight. The consequences of having a delay can have unprecedented effects, thus for unbiased study, the delay must be negligible.

In a study by Kuroki [5], they tested people's ability to distinguish touch sensations that were 12.5 ms apart, and found that they were correct only 75 % of the time.

In applications such as the tactile internet studied by Junior [24], the maximum delay ranges from 1 ms to 10 ms, depending on the requirements of the application.

As described in a survey by Darvish [19], low latency can lead to strong telepresence, as mentioned in Section 2.2 on teleoperation. This leads to a conclusion, that consequences of can be unprecedented and the smaller the delay the better, but it heavily depends on the application. Since the robot in high-level control mode contains 25 ms control delay and the study by Jay [22] mentions it as a treshold, it is a good target to aim at. However, the newer studies mention smaller delays up to detectable 12.5 ms or even lower for the tactile internet.

The following essential aspect is safety.

2.5 Safety

Safety is a critical aspect of human-robot interaction, especially in applications where the robot is in close proximity to humans or interacts directly with them. Ensuring the safety of both the human user and the robot is paramount. Haddadin and Croft [25] discuss cooperative proximate human-robot interaction and emphasize the importance of considering the worst-case scenarios in the design process. In particular, they highlight the distinction between constrained impact or clamping and unconstrained design, with the latter being preferable for safety reasons.

To accurately measure the forces exerted by the robot and ensure safe interaction, it is necessary to have a model of the robot and measure external forces. International Organization for Standardization (ISO) has developed standards to guide the design and operation of robots with respect to safety. ISO 10218 focuses on identifying hazards and managing risks [26, 27], while ISO/TS 15066 provides complementary guidelines for designing collaborative robot systems, including maximum limits on tissue pressure to ensure human safety [28].

One approach to enhancing safety in human-robot interaction is through lightweight design, which results in lower inertia, which is the case for the robot used. This design philosophy minimizes the potential damage caused by the robot in case of unintended contact or collisions with humans. The reduced inertia allows for quick stops and changes in direction, further enhancing the safety of interactions.

However, the robot alone is unsafe and the application must be considered. Furthermore, if the robot is considered collaborative, it provides safety guarantees. However, it is not the case for the used robot here. One possible measure is using a passive skin on the robot, working as a cushion or active skin detecting the touch and acting on it [29]. Next is the end-effector's position, where the position affects the robot's applied forces [30].

Thus during the robot design, it is essential to consider possible hazards and eliminate them as much as possible during the design phase.

2.6 Conclusion

Various presented studies from different fields lead to the importance of low latency for strong telepresence and the impact of it on user interaction. In conclusion, the choice of control mode, as described in Section 4.3.1, for the robot will need to be carefully considered, balancing safety and speed. The first proposed goal is to use high-level control, as it is safer by design and its control delay is 25 ms, which was a threshold of detectable delay in study by Jay [22]. Low-level control is faster but does not retain many safety features; thus, it is the subject of testing. The hardware used determines those limits and is given as a constraint. It is described in the next chapter.

Chapter 3

Hardware, Setup and Software Platforms

This chapter describes the hardware, the experiment setups and the software platforms used, with their interaction over a local network.

3.1 Hardware

This section focuses on the cameras, the robot and the laptops used.

3.1.1 Cameras – Qualisys 3D Motion Capture

The cameras used are part of the Qualisys 3D Motion Capture system. The system includes 8x Miqus M3 with resolution 2 MP (1824x1088) and a capture rate of 340 Hz in full resolution, which was used, and 650 Hz in reduced 0.5 MP resolution. Additionally, a Miqus video camera with a resolution of 2 MP (1920 x 1080) and a capture rate of 85 Hz, or 1 MP and 180 Hz in reduced resolution was included, but not used for the experiments. The faster modes with reduced resolution come at the price of detection accuracy, which was the reason for not selecting them.

The camera setup in the laboratory room can be seen in Figure 3.1. The cameras are positioned around the room to capture a full view of the space.

Figure 3.1: 8x Miqus M3 cameras in the laboratory room.

The data captured by the cameras were processed using a Dell Latitude 7490 laptop running Windows 10 Pro. This computer was equipped with an i7-8650U CPU running at 1.9 GHz and 16 GB RAM. Two Ethernet ports are needed to connect the cameras and a local network. The second Ethernet port was not available on the laptop; alternatively, it was provided via a docking station that powers it. The software controlling the cameras, Qualisys Track Manager, is described in Section 3.4.1. The software mainly calculates 3D

positions of motion capture markers and streams the positions to another laptop controlling the robot.

3.1.2 Robot – Kinova Gen3 with Robotiq 2F-85 gripper

In addition to the cameras, a Kinova Gen3 robot with a Robotiq 2F-85 two-finger gripper was utilized as in Figure 3.2. The robot was controlled through another laptop, MacBook Pro 16 inch, 2019, with an i9 CPU running at 2.3 GHz and 16 GB RAM. This laptop was running MacOS Ventura. The main focus of this work, software controlling the robot, described in Section 4.1.2, was running inside a docker container on this machine.

Figure 3.2: Kinova Gen3 robot in the laboratory.

The Kinova Gen3 robotic platform is commonly used in research and education as it is ultralightweight and built for human-robot interaction [31]. Its Kortex API includes high-level control containing a solver for inverse kinematics, taking care of additional safety, such as limiting the velocity of the end effector. It also includes a low-level controller, which does not have those features. The control modes are described in Section 4.3.1.

The following section will explain how these devices worked together during the experiments.

3.2 Setup of the experiments

There were two pilot experiments. The first pilot experiment took place in March, while the final experiment was conducted in May (video). The final experiment was different from the first, as some problems had been discovered in the first experiment. The source code used for both experiments is marked with the appropriate date as a tag [32]. There was an additional experiment in March, but only minor changes were made.

3.2.1 First pilot setup

The first pilot experiment is shown in Figure 3.3. As described before, the robot's purpose was to slide over the participant's left arm according to the movements of the participant's right arm, which controlled the robot motion. During this first experiment, the participant was supposed to move his arm by a distance represented by a line on a screen. The gain applied to the robot's movements between each trial varied.

On the right arm, there were three markers during the pilot experiment and two additional markers on the robot. A calibration triangle with four markers was used as a fixed point in the corner of the table. The Qualisys Track Manager captured those points, as described in Section 3.4.1. The line was displayed by an additional laptop, since no graphical user interface (GUI) was implemented on the laptop that controls the robot. Thus, the experiment required two operators synchronizing the two laptops for the participant, marking start and stop of each trial, while the participant was only moving their hand.

Figure 3.3: The first pilot experiment setup, where the robot slides over the participant's left arm according to the movements of the right arm. There are three Qualisys markers on the right arm, two on the robot, and four on the calibration triangle.

The synchronization of those computers was very focus-intensive for the

operators when running the experiment. Furthermore, this experiment was evaluated on internal personnel only as a way to check the basics of the setup; it's functionality was limited. The final setup reflected those points and is described in the following subsection.

3.2.2 Final setup

During the final setup, the participants were seated in front of a 65 inch Samsung television screen with their right arm on a sponge to support the arm and their left arm placed in an articulated arm support Ergorest, series 330 011, Finland [33] held a Bluetooth mouse as in Figure 3.4a. As a remark, the arms are now flipped in comparison to the previous setup, limiting the shared workspace between the participant and the robot. The right arm was the touched one, while the left arm was the moving arm. It also differed in the task. The participants freely moved their arm, given only basic instructions, such as going further than in the previous trial. Subsequently, they estimated the length of the movement of the left arm or the length of the touch on the right arm, represented by a line on a screen that they controlled by pressing pedals placed under their feet (Figure 3.5). The pedals were made from keypads, where all keys except one were removed, a cushion was added acting as a spring and all covered by a cut in half sponge. The GUI was now running in the robot controller program, thus it required no input from the experimenter during the trials.

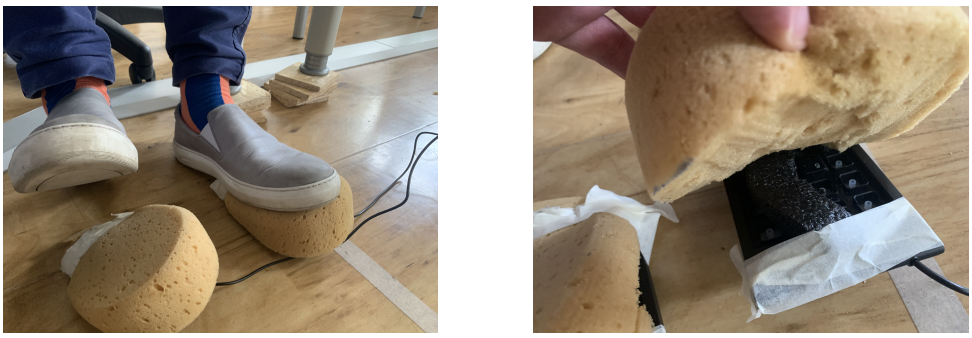
Moreover, the participant's direction was no longer perpendicular to the table as shown in Figure 3.3 and Figure 3.4, to prevent an accidental collision with the robot by minimizing the shared workspace with the robot and farther from the participant's head. More about the safety is provided in Section 4.4.3.

(a) **(b)**

Figure 3.4: Final experiment setups. **(a)** Robot slides over participant's right arm mimicking the movements of the left arm. Markers on arm, robot, and table captured by Qualisys Track Manager. Arm support table houses a mouse. **(b)** Robot touching participant's arm, participant wears noise-canceling headphones and tunnel-vision blinders.

During the trials, participant wore noise-canceling headphones with a white

noise on and tunneling-vision blinders limiting the participants' field of view to the screen and concealed the robotic setup and their arms (Figure 3.4b). The tunneling-vision blinders were hand made from bottles, tape, spring and a piece of fabric made by Jason Khoury.



(a) **(b)**

Figure 3.5: Final experiment setups. **(a)** Pedals used as controllers to judge moved distance by the participants. **(b)** The pedal underneath the cut in half sponge is a keypad with removed buttons except one.

The robot provided tactile stimulation on the right arm congruent with the hand movement of the left arm thus emulating self-touch, as if the left hand was touching the right arm. There were three markers on the robot and four on the participant's hand, where the extra marker was to measure the arm's length. The triangle on the table was replaced with one marker as the other markers were unnecessary.

A flexible 3D printed elastic stick with a holder for the robot, made by Krištof Pučejdl, was used to stroke the right forearm. The ratio between the movements of the participant and the robot was manipulated, as in the Cataldo study [1, 2] with configurable ratios. The gains used were $\frac{1.5}{1}$, 1, $\frac{1}{1.5}$ producing various combinations of motor and tactile displacements.

The controller program behind the robot is described in Section 4.1.2 and GUI in Section 4.4.2. The hardware part connections were described here; the next section is about software connections, specifically network connections.

3.3 Network

In the setups, there were four devices communicating and one virtual as shown in Figure 3.6.

The Qualisys notebook communicated with the cameras and the notebook running the robot controller. The laptop running the robot controller was a proxy for a virtual Docker machine running it. Although the virtual machine introduced a slight delay of a few milliseconds, it enabled the controller program to function across multiple operating systems. It allowed replacing the notebook as a proxy with just about any other performant laptop. The robot program inside the Docker container communicated with the cameras

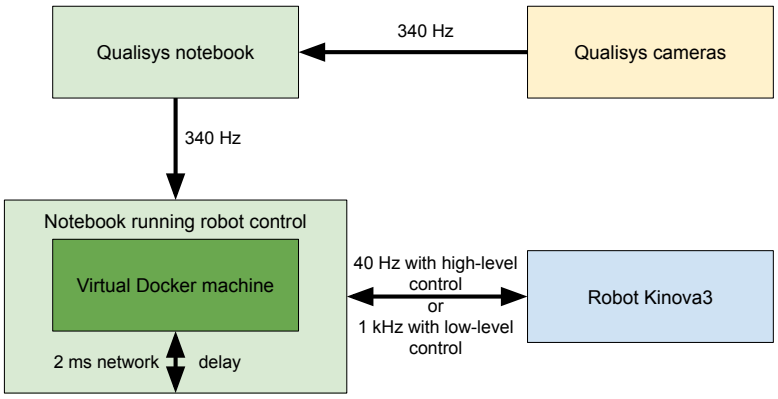


Figure 3.6: Networking scheme: green blocks represent a machine, dark green is a virtual machine, yellow are the cameras, blue is the robot.

and the robot. It translated the position of 3D keypoints into the motion of the robot.

Data from the camera and the robot were coming through the network in both directions every millisecond. All of this had to be reliable and fast for real-time interaction. Thus, the communication was happening over the Ethernet network. However, the software itself did not distinguish this as it is a matter for the operating system to solve. The software was provided just with the IP addresses of the corresponding end-points. Although Docker had its benefits, it had a downside by adding a significant delay to the communication up to a few milliseconds when measured via ping command, which is significant when the robot is controlled at rate of 1 kHz, but still possible to deal with, since the cameras are working at 340 Hz anyway. The software for processing data from the cameras is in the next section.

3.4 Software

This section describes the software for gathering the data from cameras and introduces the robot controller program.

3.4.1 Gathering data from cameras

The Qualisys Track Manager was used for data acquisition and analysis. It can be used in animation, and here, it was used to track the movements of the participant and the robot using the Qualisys reflective markers as shown in Figure 3.7, which the program then translated into 3D keypoint positions as shown in Figure 3.8. It comes with real-time streaming and capturing of the data. This program was controlled from the robot controller code using Qualisys SDK. As a side note, Windows 10 machine was required to run the Qualisys Track Manager, making it cumbersome to run it all on a single machine.

Figure 3.7: Four Qualisys markers attached to a left arm.

The cameras required calibration to establish a coordinate system. Once the calibration was done, it was saved and reused across the experiments. Next, Automatic Identification and Measurement (AIM) model calibration was needed to label keypoints in space, such as the center of the hand or joints in motion capture systems. This model helped to track specific body parts during the experiment. The model was flexible, making it possible for a single model to be used by all participants. The keypoints were reconstructed from reflected light by the spherical markers from Qualisys as in Figure 3.7. It was possible to adapt the tracking settings, set the exposure and threshold for the detection, or have a mask to select the tracked area for each camera. The masks were useful in eliminating reflective light from other objects inside the room, which would otherwise influence the measurements by adding "ghost" keypoints.

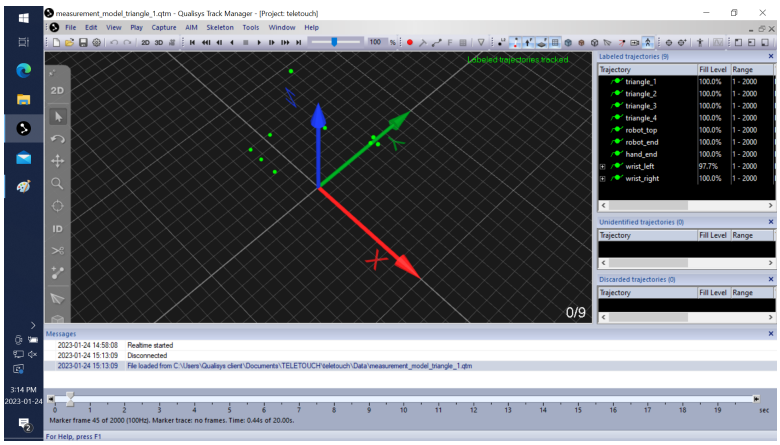


Figure 3.8: Screenshot from Qualisys interface during recording playback.

One of the features of the program was the ability to replay a measurement as in real-time and even change the playback sped, which made it possible to debug or measure the forces produced by certain movements. It further allowed analysis of the movements, such as computing the velocities, smoothing out trajectories and with specialized tools it can even measure forces, however

those tools were not available. Mainly the program was used to stream the detected labeled keypoints to the program controlling the robot, introduced in the next section.

3.4.2 Robot controller

The development of the main application has gone through many iterations and even in programming language changes. The first language of choice was Python, which is quite simple to read and write. From a performance perspective, Python is often sufficient, although interpreted language. And with the benefit of readability and a large community, where it can solve many tasks using libraries, it was the first choice of programming language. However, for this application, it turned out to be too slow, as the communication with the robot reached only about 20 Hz, despite the documentation for the robot stated it would run at 40 Hz [34]. Later discovering that having 50 ms delay does not work for this application, because the delay would influence the experiment too much led to rewriting the code to C++.

It is a much faster, compiled language, also with a large community. Even though the code is not as readable as the one in Python, having the Python code made the rewriting process much faster and simpler. The same loop in C++ runs at the declared 40 Hz. The lag was two times lower, but it was still a noticeable delay for a task where the robot has to go at the speed of a moving arm while touching the non-moving arm and ideally start and stop at the exact moment as the moving arm.

The robot had several control modes described in Section 4.3.1, with the final solution using a low-level controller running at 1 kHz. Defining the precise and optimal method the final method for optimal control of the robot was the largest part of this work, which is described in the next section.

Chapter 4

Implementation

This chapter deals with the whole implementation process of the final code [32]. It covers the implementation strategy process, the frameworks, and libraries used in the code. Then it describes the code itself, its structure, and its configuration. The next part is about the optimization options for the robot controller and the trajectory interpolation for the low-level controller. The penultimate section deals with the experiments, how and what data was recorded during the experiment, the graphical user interface (GUI), and safety considerations.

Everything is summarized in the last section of this chapter.

4.1 Introduction

This section explains the implementation strategy and the frameworks and libraries used in the code.

4.1.1 Implementation strategy

In planning the implementation strategy for the project, the process was guided by agile principles. The idea was to keep development flexible and responsive to change, focusing on iterative development and continuous feedback. This section provides an in-depth look at key elements of the implementation strategy.

1. **Iterative Development:** An iterative process was adopted, each iteration involving first a prototype, then feedback from the supervisors, and after that, the complete implementation was done. After each iteration, working software was produced, which was improved in the next iteration. The advantage of this approach is that it enables changes and improvements to be made continuously as the software evolves while keeping the time cost of prototypes low. For example, during the first pilots, the application had a separate program on a separate computer to navigate the participant. It was found to be too exhausting for the experimenters that the time cost of developing GUI was justified.

2. **Incremental Feature Addition:** Rather than trying to create a full system in one go, the approach was to build the system incrementally, adding one feature at a time. It was beneficial in maintaining control over the project's complexity and ensuring that each feature was appropriately integrated before moving on to the next. It was achieved by having multiple objects that could run as a thread while having shared memory in between.

3. **Continuous Testing and Integration:** A vital part of the implementation strategy was maintaining a strong focus on testing. Each feature was tested to ensure that the different parts of the system worked well together. Testing sometimes required disabling other parts of the code or keeping only specific hardware connections, such as the cameras or the robot, so it was done manually.

4. **Adaptive Planning:** The project planning was flexible and adaptive. It was understood from the beginning that the design details could change as the project evolved. As a result, project planning was continuously reviewed and updated throughout the implementation phase. However, the big picture of having a robot controlled by a motion of a hand remained the same during the process.

5. **Feedback Loops:** Feedback sessions were held to improve the setup of the experiment and to check with the objectives continuously. Feedback was collected from both experimenters and participants during the pilot phases. This feedback was then used to guide future iterations of the software.

This agile development strategy was integral in addressing challenges and efficiently achieving the project goals. As an example already mentioned, the first language of the project was Python, but it ended up being rewritten in C++. The following section discusses the frameworks and libraries used in the final code.

4.1.2 Frameworks and Libraries

The robot was controlled by a C++ application running inside a Docker container, a lightweight, portable, self-contained unit that runs software applications consistently in different environments [35]. The application used the X Window System, also known as X11 or X, to display a graphical interface. Through this interface, the user could interact with the application inside a Docker as if it were running on the local machine using the X client [36]. For the GUI, the C++ library wxWidgets was used because it provides high-level platform-independent classes and functions. It supports event callbacks to detect keystrokes or mouse clicks that can be processed further.

The application could be configured using an `ini` file. To facilitate the parsing of the ini file, the mINI library was used [37]. Within the configuration,

there was a JSON string that required parsing. To handle this, the Lohmann's JSON library was used [38].

The code uses multiple threads to split tasks into several logical parts, e.g., `UI` thread for processing user input that may come from GUI or the console. One of the threads for communicating with the robot was provided by KortexAPI [39]. It provided several control modes – mainly high-level and low-level modes. The chosen mode, low-level position control necessitated the computation of joint velocities. In the calculation of the velocity Proportional-Integral-Derivative (PID) controllers were used. PID controllers were using Bradley's implementation [40]. To obtain the 3D position of the hand from Qualisys Track Manager, Qualisys SDK [41] was used.

Therefore, Docker was required on the machine to run the program. If the X client was set up properly, it could run seamlessly. The next section is about the code structure, program states and configuration of the program.

4.2 Code, States and Configuration

This section describes how multiple threads were used and why, the states in the program, and finally configuration of the program.

4.2.1 Code structure

For the program to handle multiple inputs and outputs in parallel, multiple threads were used. The main thread was the GUI thread, as required by the wxWidgets plugin [42]. During its initial process, it initialized all the other threads, except the low-level controller that ran 1 kHz communication with the robot.

The low-level controller is represented by a class `KortexController`, as can be seen in Figure 4.1. It was initialized by `RobotController`, which took care of computing goal position for the robot's end-effector, immediate goal position during low-level control, and setting up the robot, including calibration if necessary.

`KortexController` can act as a class and as a thread. When used as a class, it has commands for the robot such as "set the end-effector's velocity", "set the end-effector's position", etc. which used the robot's high-level control described in Section 4.3.1.

However, when started as a thread, it started controlling the robot in the low-level joint position control mode, sending joint angle positions and velocities at 1 kHz. Joint positions were read from shared memory. When the robot was operated in low-level mode, it could not be operated in high-level mode, which would have been useful when going to the robot's home position. This was solved by computing linearly interpolated trajectory between the robots initial position and the goal position within configured time.

Making the threads communicate with each other required having shared data structures, and mutex locks. If a thread wanted to read the data, it had to wait for a free lock, locked access to the memory, read the data, and unlock

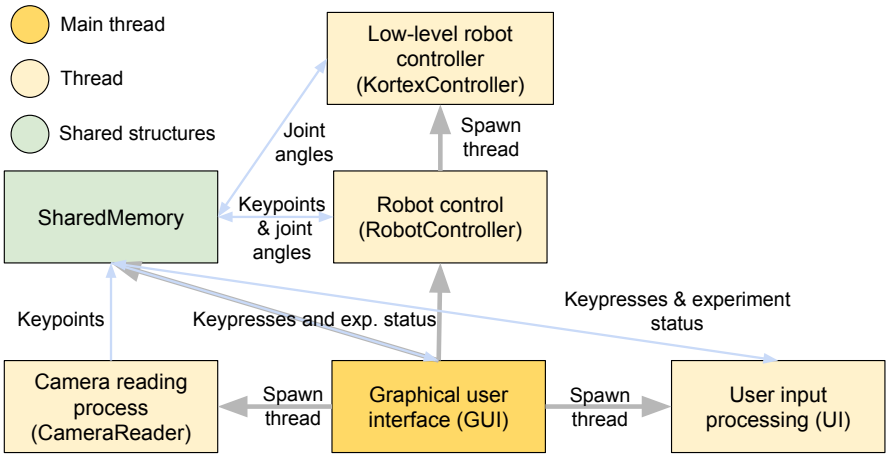


Figure 4.1: Program diagram: blue arrows represent communication through shared memory and the gray arrows represent which thread spawns which.

it for other threads. There were multiple shared structures, depending on the needs of each thread. There was a lot of data being communicated across the threads when the program was running, such as robot's joint angles, position data from the camera, user input, etc.

`KortexController` read the goal joint angles every millisecond. These were being written by the `RobotController` thread and computed based on the data from the cameras. But none of those processes had access to the cameras directly. Instead, another thread, `CameraReader`, was reading the data from the cameras and storing them in the shared memory.

The user input could come from two sources, GUI or the console. To process them, the `UI` thread was specialized to do so in one place. It also acted as the main controller, taking care of the robot's gain, measuring the arm's length, and taking care of the current state of the program, which are be described in the next section.

Consequently, having it all separated made it structurally understandable and modular. For instance, if an update would make the robot's high-level API communicate at 100 Hz, it might be worth replacing the current low-level controller with a high-level velocity controller and omit the immediate goal computation. Another use case might be to replace cameras with different types of sensing.

4.2.2 States of the program

For the final experiments it was necessary to add states to the program as it contained too many variables. The states were designed to accommodate the experiment scenarios and variants of them. In the experiment, there were four scenarios in total and three variants. The variants represented the current objective of the experiment:

1. **Familiarisation:** this was the default variant when the program started.

Here, the participant got to know the robot and its movements.

2. **Tactile:** the participant focused and evaluated the length of touch provided by robot's tool.

3. **Movement:** the participant was focusing on and evaluating the length of the arm movement.

The scenarios were as follows:

1. **Introduction:** when the application was opened, this was the first state. It described the instructions for the selected variant. This was the only state that allowed changing the variants.

2. **Push to start:** this state was only for the tactile and movement variants, it contained a shorter version of the instructions shown between each trial.

3. **321+:** in this state the robot started touching the participants arm and started following the hand.

4. **Judgement:** here the participant was judging the length of touch or movement based on the instructions. This was exclusive to tactile and movement variants only.

The scenarios depended on the selected variant, where the user was switching between the states by clicking the mouse held in the left arm. The interaction flow diagram is shown in Figure 4.2.

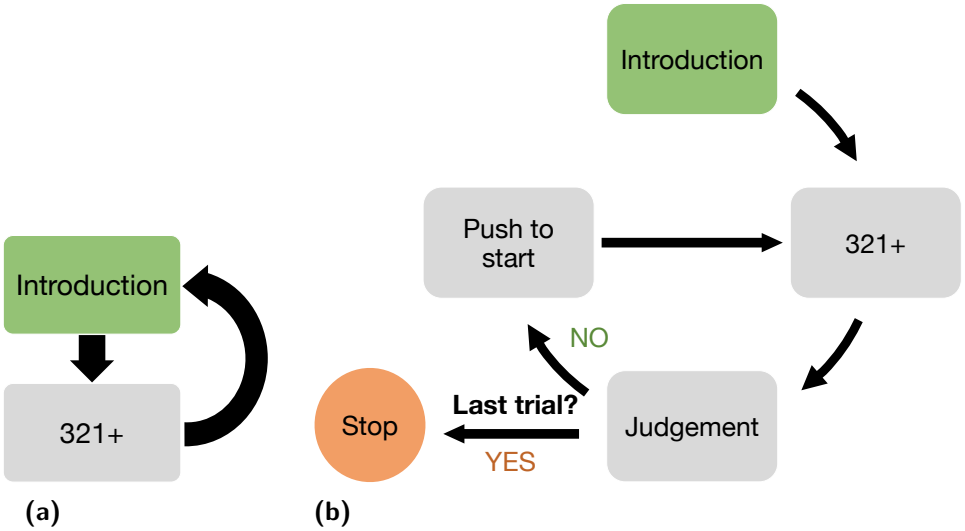


Figure 4.2: Scenarios diagram for different variants. The participant goes through them by click the mouse. **(a)** When familiarisation is active, the program loops between the two scenarios. **(b)** For variants tactile and movement, the scenarios loop until the participant goes through all trials.

Given the combination of variation and state, GUI was changing its layout.

`UI` thread, which was the processing unit for the user input, changed the function of buttons. In addition, it was in charge of triggering the logging of the experimental data, changing the states and variants, and more. All of which depended on the current state, variant combination.

The `RobotController` thread, which takes care of the robot's end-effector position, changed between following the hand, going to the initial position, and the idle state. When the 321+ scenario was on, it went towards the hand and started tracking the hand. When this scenario was changed, the robot went to its initial position away from the hand. In the next trial the initial position was randomized based on a given configuration. The configuration is described in the following section.

4.2.3 Configuration

Operation of the robot contained configurable parameters; therefore, it was useful to have a separate configuration file. Inside the "settings.ini" file, there were sections determined by the class names.

First, there was the configuration of the GUI, containing settings of the screen height in centimeters, pixels, and a denominator. Inside the code, the screen height in pixels was divided by the height of the screen in centimeters and by the denominator multiplier. The denominator multiplier would usually be 1, but for retina screens it was 1.5.

Next, there was the `KortexController` class that communicates with the robot. It contains settings for the robot's IP, required ports, timeouts, and parameters for the PIDs, including the maximal speed for the joints. Although each joint had a separate PID, all shared the same parameters, since only proportional gain of 1 was used.

Continuing to the `RobotController` class, it contained the transformation from the cameras to the robot, the number of positions measured to calculate a calibration transformation if it is not set, the parameters of the moving window to limit the speed of the robot, the initial position of the robot during the experiment, the velocity limits, the calibration constants, the boundary box in terms of the minimum and maximum values for each axis, and the random offset between the trials.

When changing the boundary box, it is important to make sure it is set correctly by checking the corner coordinates, by moving the robot to corner coordinates with the controller or the robot's web controller.

Lastly, there was the `CameraReader` class, which reads position keypoints from the cameras. It contained only basic connection information such as IP, port, version info, endian and password.

The next section describes the steps taken to find the optimal robot control.

4.3 Robot Control

Controlling the robot with a delay imperceptible to the participant while ensuring safety played a vital role. Originally, the delay threshold was set to be 25 ms, as studies had shown that it was a detectable treshold for some application, while achievable when using high-level control mode. However, it was shown through pilot experiments, that it was not the case for this application.

Thus it was necessary to use the low-level controller running at 1 kHz. However, the low-level controller itself does not take the 3D position as input but rather the joint angles. Moreover, the robot's target position had to be split into smaller steps for this controller to work. The next part will describe how all this works.

4.3.1 Optimizing the robot control

The first iteration in Python was made with the high-level Kortex API [39]. It allowed controlling the position and speed of the robot's end-effector without worrying about the inverse kinematics (IK) and offered built-in safety features. For example, it was cheking joint speeds, self-collision, and workspace boundary box, which was configurable through a robot's web interface. However, those safeties were not guaranteed for the Kinova Gen3 robot.

Despite a low-level API could operate at 1 kHz, it was not selected for the first few iterations because it seemed too high for the task. However, the high-level control was too slow when examined in pilot testing. With low-level control modes, there are no safety features or the calculation of IK. They only accept joint angles, joint velocities, torques or currents.

Calculating the joint angles for the IK itself is not trivial. Fortunately, there are libraries that can be used to solve it very quickly. In 2022, Lloyd et al. [21] published a novel approach to solve it using Halley's method. They also provided the algorithm in C++ and a Matlab wrapper, publicly available on GitHub [43]. The algorithm can compute IK for the Kinova Gen3 robot on average in 16 µs while having a strong performance near singularities. There is also the Pinocchio library by Carpentier et al. [20] from 2019, in which they unroll most of the computations directly at compile time, achieving even better performance in terms of computation speed, but it may not be as robust near singularities as Halley's method, since it uses Newton method, which was outperformed.

Calculating joint angles for the low-level 1 kHz control loop was now possible. The robot offered a variety of control modes. The first recommended mode for controlling the robot, which was joint position control, did not work well initially. The initial trials were problematic because the robot needed joint velocities in addition to joint angles, which was not mentioned in the documentation. Without them, the robot seemed to brake every millisecond and sometimes the joints would get stuck and stop moving during startup.

Other modes were explored as options, but none of them worked. The low-level velocity control mode has a bug, so the developers of the firmware recommended not using it. Torque control would not be practical for controlling the robot's arm position and lastly, controlling motor currents goes far beyond this task.

Servoing mode	Control mode	Frequency	Advantages	Disadvantages
High-level	Position control (end-effector)	40 Hz	+ End-effector position as input	- Works only at 40 Hz - Stops after reaching goal position - Blocking
High-level	Velocity control (end-effector)	40 Hz	+ Continuous + Non-blocking	- Works only at 40 Hz
Low-level	**Position control (joint angles & joint velocities)**	1 kHz	+ Works at 1 kHz	- Inverse kinematics to be solved in 1 ms - Positions must be continuous - Uses maximal torque
Low-level	Joint velocity control (internal velocity loop)	1 kHz	+ Works at 1 kHz	- Inverse kinematics to be solved in 1 ms - Gravity not compensated
Low-level	Torque control (joint torques)	1 kHz	+ Works at 1 kHz	- Inverse kinematics to be solved in 1 ms - Robot dynamics computation

Table 4.1: Comparison of different control modes for the Kinova Gen3 robot. Low-level position control was used.

In Table 4.1, there is a comparison of all possible modes with their advantages and disadvantages, except for the current mode, which is not documented. The selected mode was the position control mode since it worked at 1 kHz, because turned out to be the most important factor.

However, this required to split the movements into smaller steps using trajectory interpolation.

4.3.2 Trajectory interpolation

When controlling the robot in position low-level control mode, the robot used joint angles and joint velocities to move to the target position. If the joint angles were too far from the current values, the robot would go into joint fault mode and stop moving.

Thus, the goal position had to be close to the current position of the robot. If the position was too distant, an immediate goal was used instead, as can be seen in Figure 4.3. Both the goal and the immediate goal position were always limited by a configurable boundary box as a safety measure.

Calculating the immediate goal position was tricky because all sources of information had different timestamps.

The cameras sent the keypoints at 340 Hz, while the robot was controlled at 1 kHz. On top of that, those timestamps were rarely in phase as they are

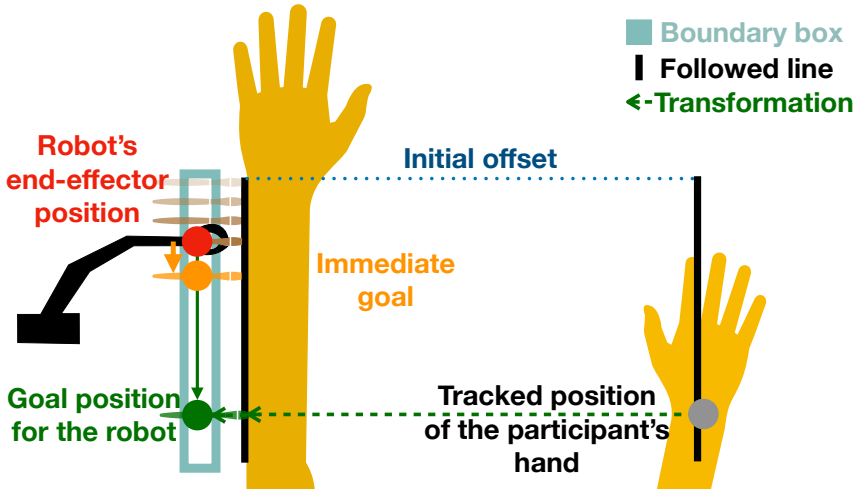


Figure 4.3: One frame from trajectory following process computed every 1 ms. The robot is following participant's hand, but it is too far, thus the robot uses immediate goal instead of the goal. Distances are exaggerated for more clarity of the drawing.

not multiples of each other and there was delay in the network. When the robot sent its joint angle positions, it sent them as a callback after setting the future joint angles and joint velocities with an additional delay. Since the network, described earlier in Section 3.3, contained a delay higher than 1 ms, it always made the control loop lag behind. A very basic model that represents this can be seen in Figure 4.4.

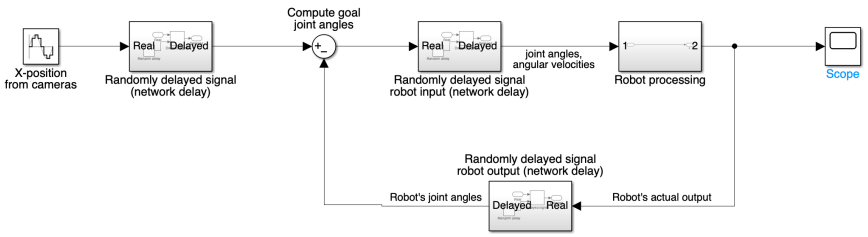


Figure 4.4: Simple model of control loop for the robot with the added delay.

On top of the delay, it was necessary to ensure that the robot's end-effector does not move too fast. In ideal case, the robot would send its joint angles every millisecond, but the end-effector speed had to be limited to a configurable speed. Let us say that it is configured to 0.4 m/s, which is equivalent to 0.04 cm/ms.

However, changes of 0.04 cm cannot be accurately measured due to the delay and precision of the measurements. Instead, if the robot's position is within 1 cm after 25 ms, its speed is within limits, and its position can be measured. The robot's previous measured positions, for example, up to 50 ms ago, can be used to limit the robot's end-effector speed.

This was implemented as a moving window that stores the robot's positions and timestamps from the measurements. If the robot's immediate goal position would be outside the limits of the moving window, the robot would be going too fast and thus had to slow down. Pseudocode of the Algorithm 1 shows explains this in more detail.

Algorithm 1 Compute intergoal

procedure CMPINTERGOALPOS($mw, goal, measTime, wasLimited$)
  $interGoal \leftarrow goal$
  $robot \leftarrow mw.points.back$ ▷ get robot's last position
  $dt \leftarrow currTime - measTime$ ▷ Δt in seconds
  $avgVel \leftarrow$ Average velocity over valid historical points in mw

  $dist \leftarrow norm(robot - interGoal)$
  $maxDist \leftarrow MAX\_VEL\_MULTI * MAX\_VEL\_NORM * dt$
    ▷ max distance robot can travel
  if $wasLimited$ **then**
    $maxDist \leftarrow maxDist * MAX\_DIST\_RECOVERY\_MULTI$
  end if

  if ($dist < maxDist$ and $avgVel < MAX\_VEL\_NORM$ **then**
    $wasLimited \leftarrow false$
  else
    $wasLimited \leftarrow true$
    $distMulti \leftarrow maxDist * MAX\_VEL\_DECREASE\_MULTI$

    $interGoal \leftarrow$ Position based on robot, goal and distMulti
  end if
  $interGoal \leftarrow$ keep inside boundary box($interGoal$)
  return $interGoal, wasLimited$
end procedure

First, the target position was checked to determine whether it stayed within a threshold distance $maxDist$ (m), which depended on whether the previous iteration was already limited.

The initial threshold distance was calculated as the maximum velocity MAX_VEL_NORM (m/s) times the time difference dt (s) between now and the time the joint angles were received to calculate the position of the end-effector. On top of that, there had to be "compensation multiplier" constant $0 < MAX\_VEL\_MULTI$, to compensate for the errors in the real measured speed versus the configured speed. MAX_VEL_MULTI was empirically derived to match the measured robot's speed from Qualisys cameras with the configured speed.

If the goal position was within the limit of $maxDist$, the robot was immediately sent to that position.

On the contrary, if the participant went too fast, either by breaking the

$maxDist$ limit or the moving window limit, the robot's goal position was moved closer to the robot, thus decreasing the speed. It was done by shortening the length of the step size by $0 < MAX\_VEL\_DECREASE\_MULTI < 1$, making the step size following:

$$\begin{aligned} distMulti &= MAX\_VEL\_DECREASE\_MULTI \cdot maxDist \\ &= MAX\_VEL\_DECREASE\_MULTI \cdot \\ &\quad MAX\_VEL\_NORM \cdot \\ &\quad MAX\_VEL\_NORM \cdot \\ &\quad dt \end{aligned}$$

Once the robot got outside those limits, it had to get closer to the hand than initial $MAX\_VEL\_MULTI \cdot MAX\_VEL\_NORM \cdot dt$ to start going to the goal position immediately again. The new allowed distance was multiplied by another constant, $0 < MAX\_DIST\_RECOVERY\_MULTI \leq 1$, which required the robot to be at least $MAX\_DIST\_RECOVERY\_MULTI \cdot maxDist$ meters from the goal position. This ensured that the robot would not move by jerky movement.

Once the immediate goal was computed, its position was limited to be within a configured boundary box. IK computation followed, which calculated the joint angles of the goal, and if there was no error in the IK, they got stored in shared memory. The next section is about what was done with them afterwards.

4.3.3 Low-level controller

During the startup, the low-level controller set its goal joint angles in shared memory to its current joint angles. Subsequently, the shared memory was read every millisecond. Next, those joint angles were checked to see whether they are within the robot's joint angle limits, and if not, it would not send the command and stop the robot instead as seen in the Algorithm 2, Line 9.

Otherwise, the difference between the current joint angle positions and the goal joint angle positions was calculated and used as input for PID, which was used to compute each joints velocity.

Each joint had its own configurable PID for controlling the joint velocities, limited by configurable maximum. However, in the experiment, only the proportional gain was used, and thus the configuration was shared for all joints.

The PID's outputs had a shared configurable limit, which was $4\,°/s$ as it prevented the robot from entering joint fault modes; however, $50\,°/s$ was stated in the official documentation [34].

The following section contains the implementation details related closely to the experiments.

Algorithm 2 Update Actuator Positions and Velocities

```
1:  procedure UpdateActuators(rob, pid)
2:      ik_angles ← read goal joint angles from shared memory in degrees
3:      send_command ← true
4:      if robot is connected and ik_angles are ready then
5:          for i = 0 to actuator_count − 1 do
6:              goal_angle ← ik_angles(i)  mod 360
                                              ▷ wrap the goal joint angle
7:              if goal_angle is inside the forbidden angle for joint i then
8:                  send_command ← false
9:                  Log error, command will not be sent, robot stops
10:             end if
11:             rob.set_position(goal_angle)
12:             ang_error ← substracted actual from goal angle
                                ▷ angles are subtracted and wrapped around 2π
13:             rob.set_velocity(i) ← pid[i].calculate(ang_error)
14:         end for
15:     end if
16: end procedure
```

4.4 Experimentation

In this section, there is information about the data that were recorded during the experiments, graphical user interface used by the participants and the safety of the implementation.

4.4.1 Recording the experiments

During the experiment, all trials were recorded using the Qualisys Track Manager, described in Section 3.4.1. The laptop controlling the robot runs a thread `CameraReader`, communicating with the cameras using the Qualisys SDK. The SDK allowed sending an event with a text label, which got stored directly inside the Qualisys recording with the corresponding timestamp. This method was used to label all the necessary events during the experiment, such as current trial started, trial stopped, current trial number, current gain, arm's length, robot started moving, current selected variant, judged length, and number of total trials.

These files were stored on the laptop, controlling the cameras, making it convenient since all the data was stored only on one computer, while all the events were synchronized directly with the recording. Furthermore, if the controller crashed, the data would still be kept. Using the Qualisys Track Manager also made it possible to replay of the participant's movements, which

was used during development as a way of debugging and tweaking parameters in the configuration of the application.

Fields	Label	Frame	Time
1	'ARM_LENGTH_CM_23'	1.2059e+04	35.4660
2	'TRIAL_WILL_START;RATIO_1.00'	1.2662e+04	37.2373
3	'TRACKING_STARTED'	1.3341e+04	39.2359
4	'EXPERIMENT_START'	1.5891e+04	46.7359
5	'TRIAL_WILL_START;RATIO_1.00'	1.8471e+04	54.3236
6	'TRACKING_STARTED'	1.9491e+04	57.3235
7	'EXPERIMENT_START'	2.0465e+04	60.1889
8	'VARIANT_TOUCH;TRIAL_1;TOTAL_TRIAL...	2.2873e+04	67.2702
9	'TRIAL_WILL_START;RATIO_1.00'	2.8382e+04	83.4746
10	'TRACKING_STARTED'	2.9403e+04	86.4766
11	'TRIAL_DONE'	3.0462e+04	89.5902
12	'TRIAL_2;JUDGEMENT_LENGTH_CM_9'	3.3651e+04	98.9701
13	'TRIAL_WILL_START'	3.8363e+04	112.8280
14	'TRACKING_STARTED'	3.9384e+04	115.8311
15	'TRIAL_DONE'	4.1504e+04	122.0664
16	'TRIAL_3;JUDGEMENT_LENGTH_CM_26'	4.8076e+04	141.3978
17	'TRIAL_WILL_START;RATIO_1.00'	5.0918e+04	149.7553
18	'TRACKING_STARTED'	5.1938e+04	152.7553
19	'TRIAL_DONE'	5.6154e+04	165.1564

Figure 4.5: Example of the exported data in Matlab.

Within the Qualisys Track Manager, it was possible to export the recorded data containing positions and events to Matlab format ".mat" as in Figure 4.5 or ".tsv", which could be processed in Python or even Excel, although it may be too slow for the amount of data. Processing itself was not part of this work, but there were multiple ways for the experimenters to process the data further.

During the experiments, the participant used GUI to interact with the robot controller application. Any of those events could be recorded using the method above. In the following section, the GUI is described and how the participant used it.

4.4.2 Graphical user interface (GUI)

It was certain that GUI was needed after doing the first pilots described in Section 3.2.1, as it was a focus intensive task for the experimenters. For the final pilots (Section 3.2.2), GUI was implemented within the robot controller program using the wxWidges library, taking care of displaying, i.e., a line or text on a screen.

Without scenarios and variants, it would be hard to keep up with the logic of the experiments. It was the main impulse to create them and rewrite the whole base logic behind all the other components (`UI` and `RobotController`) in the first place, as they were stateless before creating the GUI.

The GUI itself was quite simple, there were three "placeholder" elements as in Figure 4.6. There was a logical switch for each element: whether to show the current trial number, or the line, whether the line should be on the left or right side, or whether to have some text in the center. This was found to be sufficient to satisfy all possible combinations of variants and scenarios.

Figure 4.6: Three basic elements of the GUI.

The line had to be drawn in a way that 1 cm was always shown as 1 cm on a screen, as the participant was evaluating a length and not a scaled proportion of it. The length of the line was allowed to be between 1 cm and the height of the display. Therefore, the display had to be higher than the length of the arm. The screen used in the final experiment was 81.5 cm, so there was no problem.

For the text, it had to be possible to contain Czech characters, since the text was in the Czech language. Luckily, wxWidgets has a way of passing Unicode characters to its strings.

For interaction, the participant had only three buttons available, of which two were pedals and one was the Bluetooth mouse click. The pedals were only for the judgement of the line length, and the mouse click worked similarly to mouse click in a presentation.

When the 321+ scenario was on, there was a countdown and during which the screen and controls would be locked, so the robot could finish its trajectory motion. It was timed in a way so that even when the participant would just skip everything, the robot would be ready. This was related to safety, which is described in the next section.

4.4.3 Safety considerations

Safety during the experiment was crucial as the participant was in direct contact with the robot. Furthermore, the robot was controlled in real-time by the participants' hand movements, which adds risks, such as unexpected movements, when the participant might want to scratch their nose, for example. To mitigate risks, safety measures were taken at multiple levels from a control point of view.

Implemented safety features

Let's assume a worst case scenario and go through the control loop, what were the measures taken:

1. **Issue with Qualisys:** The path of the data would begin in the `CameraReader`, where if the received data were wrong, they would be sent further.

2. **Robot's current state:** Then, the `RobotController` will read them. The robot would be either executing some trajectory, is idle, or follows the hand. Unless the robot follows the hand, it would be ignored.

3. **Computing goal position a):** For the robot to execute some wrong trajectory is impossible, as the trajectory is pre-programmed and the program keeps track of the previous and current trajectory. Furthermore, the trajectories are allowed to be only in certain order for the robot to start executing them. However, if this fails, the robot will set the position as a goal.

4. **Computing goal position b):** Return to the case where the robot follows the hand. The robot reads the wrong position, computes the goal position, and tries to send the robot it. However, the goal will keep only the followed X-axis, since all other axes are fixed.

5. **Computing goal position:** Next, the goal position is limited to being within the configured boundary box. If the configuration was wrong, it would be possible for the robot to go to places where it shouldn't.

6. **Computing immediate goal:** The position of the limited goal is sent to calculate the immediate goal. In short, it uses a moving window to check whether the robot's speed would exceed the configured speed limit given the current goal position and an additional check uses robot's current position, but it has to use "magic" constants, since the signal is noisy and delayed. This was explained in more detail in Section 4.3.2.

7. **Computing immediate goal a):** If the robot was in slow mode, the new immediate goal would be calculated from the previous measured position and the current position, and again, the axes are limited, so only the X-axis could be changed. If that happens when the robot is following a trajectory, it will go into joint fault mode and stop. Thus, the trajectory has to be designed already with the limits taken into account.

8. **Computing immediate goal b):** Otherwise, if it is within the limits, the goal position is used as next goal immediate goal position.

9. **Computing immediate goal:** The calculated immediate goal is again limited to being within the boundary box.

10. **Validation before computing IK:** Before calculating IK for immediate goal, it is first verified that it lies within the boundary box. If not, this position is ignored.

11. **IK:** joint angles are thereafter calculated from the limited immediate goal position and the last measured robot joint angles. This makes the library find the closest solution to the current robot position; however, this is not guaranteed. If the library reports an error, such as max iterations exceeded, it is not proceed further. Otherwise, they are stored in shared memory.

12. **Low-level controller:** the calculated joint angles are read by the `KortexController`. It checks whether the joint angles lie within the joint limits for each joint. If that is the case, the robot will stop moving and exit, as this should never happen.

13. **Sending commands:** Otherwise, the joint angles are sent directly to the robot's own controller with the calculated joint velocities, where the velocity limit is from the configuration.

14. **Inside the robot's controller:** Now, if the robot receives the setpoint of the joint angle, which is not within 3° to 5° of the current angle, the joint fault mode is triggered and the robot stops. This could happen in a case where the delay is too big and the robot does not follow fast enough.

To sum up, the safety measures were taken by limiting the position to be within a boundary box on multiple layers, the robot follows only in one direction, and other directions are fixed. These measures limit the robot's workspace. Then, the speed of the robot's end-effector is limited by the moving window and the current position, which can be noisy. If the robot would exceed the speed limit, there was a mechanism that prevented the robot from going back to fast following quickly. The joint angles are then calculated and sent to the low-level controller only if IK did not report an error. In the low-level controller, those joint angles are checked to be within the limits of the robot. If the joint angles are too far from the current robot's joint angles, the robot will stop moving and go into joint fault mode.

Risk assessment

In addition, a risk assessment was performed to check for potential hazards, which is shown in Table 4.2. In summary, all the threads ended up being negligible, mainly because the experimental setup was changed, so participant's head is outside the robot's workspace and having the boundary box on the end-effector helped further.

			Risk evaluation				Evaluation after mitigation		
ID	**Risk**	**Threat**	**Severity**	**Probability**	**Rating**	**Measures**	**Severity**	**Probability**	**Rating**
1	Wrong position from the camera input (e.g. mismatched points, when some hidden)	Robot moving in unexpected way	Low	Probable	Moderate	Boundary box was added on the robot's goal end-effector position	Negligible	Probable	Negligible
2	Missing position from the camera (e.g. cameras/computer disconnected)	Robot not moving, or jerky movements	Negligible	Rare	Negligible	None, robot stops moving.	Negligible	Rare	Negligible
3	Robot restarts during the experiment	Robot opens/closes the arm, which changes Z-position	Negligible	Occasional	Negligible	Robot must be rebooted, experiment repeated	Low	Occasional	Negligible
4	Robots shuts down	Robot starts slowly falling using its recuperation breaks	Negligible	Occasional	Negligible	Robot must be rebooted, experiment repeated	Negligible	Occasional	Negligible
5	Robot stops receiving data from controller laptop	Robot keeps joint velocities, but breaks after joint angle difference is above 3 degrees	Low	Rare	Low	Updated experiment setup, smaller shared workspace	Negligible	Rare	Negligible
6	Controller sends wrong joint angles	Robot stops and goes into fault state, but might go little lower in Z-axis	Negligible	Rare	Negligible	Robot must be rebooted, experiment repeated	Negligible	Rare	Negligible
8	Controller sends low joint velocities, while the joint angle difference is high	Robot starts shaking, eventually goes into fault state	Negligible	Rare	Negligible	Robot must be rebooted, experiment repeated	Negligible	Rare	Negligible
9	Robot stops being in control of the controller (e.g. using web-interface)	Zero joint angles are received from the robot and the robot is controlled by the web-interface	Negligible	Rare	Negligible	Robot must be rebooted, experiment repeated	Negligible	Rare	Negligible
10	Controller sends the robot to wrong position	Robot going to places where it should not be	Low	Rare	Low	Robot's goal is within boundary box	Negligible	Rare	Negligible
11	Fast moving end-effector, unexpected movement by the participant	Robot hitting the participant	Low	Probable	Moderate	Velocity limit, changed layout. If robot is too far, the robots follows slowly.	Negligible	Probable	Negligible
13	The participant gets into robots operation space. Possibly head or the other parts.	Collision of the robot and the participants head	Mild	Rare	Low	Changed layout of the experiment, robot now can not reach the participant	Negligible	Never	Negligible
14	Metal end-effector	Getting hit by the end-effector might be the most painful	Low	Rare	Low	Lower speeds of the end-effector	Negligible	Rare	Negligible
15	If the robot gets stuck, it has to be always rebooted afterwards.	Robot moving in unexpected way	Low	Occasional	Low	Rebooting the robot always.	Low	Never	Negligible

Table 4.2: Risk assessment of the application.

Measuring the impact forces

To further assess the risks, the forces were measured to see if they are within the limits of the ISO/TS 15066 [28] technical specification. The maximum exerted force can be 135 N on the hands and fingers for quasi-static contact. The hands and fingers have the lowest threshold of all parts that could come into contact. The lower legs and abdomen have lower limits, 125 N and 110 N respectively. The head is part of the critical zone that cannot be touched by the robot at all.

Forces were measured using a certified measurement device, CBSF-75-Basic, which is a force transducer. It measures forces up to 500 N and the maximal error is 3 N. The setup of the measurements was the same as in [30], with the help of Lukáš Rustler. All cases measured worstcase scenario, where the part hitting the participant would be the metalic end-effector.

There were two measured directions, one from the top, that is, if the robot would go into joint fault mode. During this measurement, the robot was turned off and the forces of a passive falling robot were measured as in Figure 4.7. There were 10 measurements, where the maximal force was 97 N. After the collision, the applied force remained at 34.4 N on average, as shown in Figure 4.8.

The next measured direction was towards the participant. It was measured by replaying a prerecorded movement of a participant and sped up 1.3 ×. There were 11 measurements and the maximal measured force was 110 N. Since there was no clamping, which would be a situation in which the hand or any part gets constrained from both sides, the force was just a fast bump as in Figure 4.9. The setup is shown in Figure 4.10.

It was also interesting to see the forces when clamping was present, although in the real experiment there was no clamping. The setup was the same as in the previous measurement, but the string that holds the measurement device was tightened. There were 25 measurements and the maximal force applied by the robot was 187 N, which was much higher. In Figure 4.11, it shows that there were measurements in which the robot lost contact with the measuring device. That was because the prerecorded movements just moved away from the robot.

In conclusion, safety guards were implemented within the robot's controller code, the safety of the application was assessed, and possible hazards were eliminated by changing the design of the experiment and adding safety guards to the code. Lastly, if the robot would come in contact, the applied forces would stay within the limits of ISO 15066 [28] technical specification, if there would be no clamping involved. The application is safe, but one has to make sure that there is no clamping, be vigilant, ready to stop the program or, if necessary, press the stop button, and hold the robot so it does not fall, in case it is needed.

Figure 4.7: Passive robot falling onto the measuring device from the top.

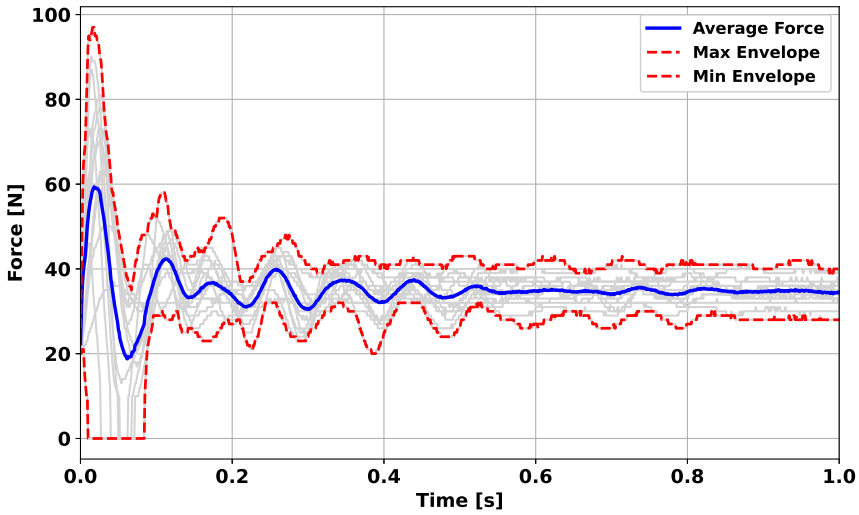


Figure 4.8: Measurements of the force, when the robot fell on the device from the top.

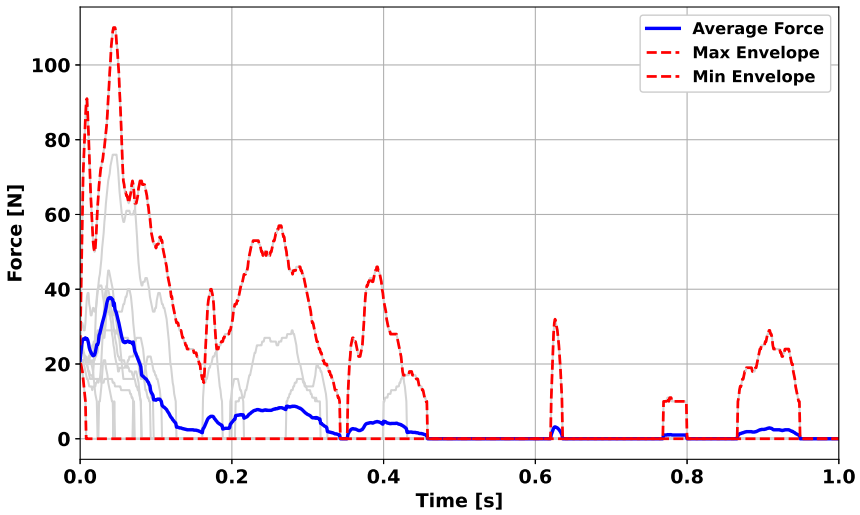


Figure 4.9: Measurements of the force, when the robot hits the device from the direction of the participant without clamping.

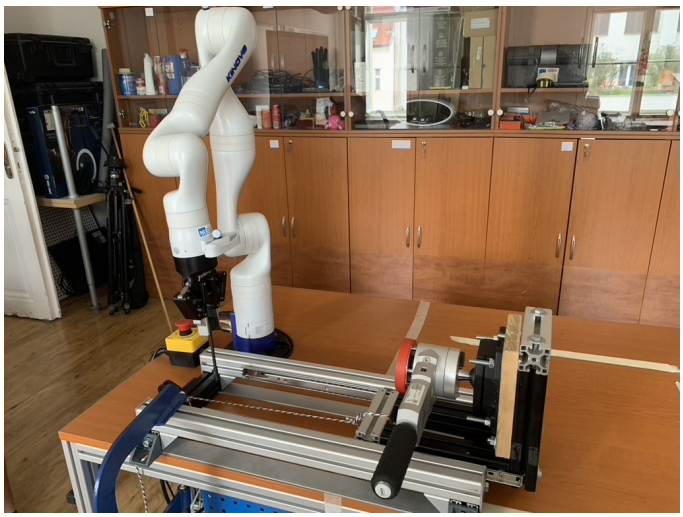


Figure 4.10: Setup for measuring the forces in the direction of the participant. Device was hitted by the metalic end-effector.

4.5 Summary

This chapter covered a wide variety of topics, thus a quick recapitulation of all the main points from each section follows:

1. **Implementation strategy:** agile development was used, due to the nature of the experiment, as it required frequent changes and there were many unknowns at the beginning.

2. **Frameworks and Libraries:** robot was in C++, the wxWidgets library was used for the GUI and all run inside Docker container, so X client

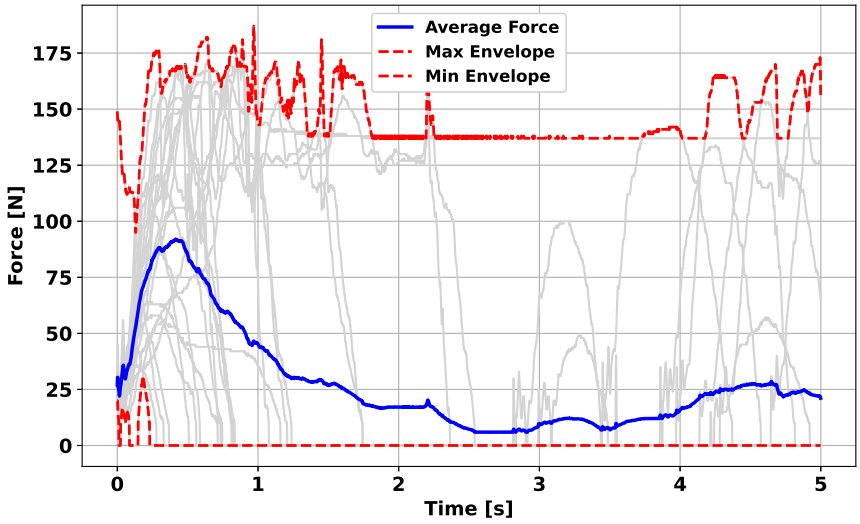


Figure 4.11: Measurements of the force, when the robot hits the device from the direction of the participant with clamping.

was used to tunnel the interface into the host machine. KortexAPI was used to communicate with the robot and Qualisys SDK to communicate with the Qualisys Track Manager.

3. **Code structure:** the program was running multiple threads, allowing flexibility and interoperability between them.

4. **States of the program** the state of the program had a variant and scenario. They represent the current state within an experiment, which makes it easier to react to changes of them.

5. **Configuration:** the configuration was stored in a "ini" file, which was divided by class names. An important note is that one should be careful when changing the boundary box limits as they affect the safety of the experiment.

6. **Optimizing the robot's control** different modes were discussed, but low-level API was used, running at 1 kHz, which required computing IK. It was done using the public algorithm on GitHub [43].

7. **Trajectory interpolation:** the control loop contained a delay, which made speed measurements harder. That was important for safety, so there were multiple ways to tackle this, such as using moving window and constants that would allow for a noise in the signal.

8. **Low-level controller:** the joint angles were restricted to be within hardware limits and maximal joint velocity was also limited.

9. **Recording the experiments:** the experiments were recorded using the Qualisys Track Manager, where all the important information would

be stored inside one Qualisys file. Exporting to other formats is also possible directly from the Qualisys Track Manager.

10. **Graphical user interface GUI:** creation of it led to the need to have the state of the program. It displays three basic elements, determined by the current scenario and variant.

11. **Safety considerations:** there were multiple safety guards at multiple levels of the code, potential hazards were evaluated and mitigated, and lastly, the impact forces were measured, where all would be within the limits of standards, except for the clamping scenario, which does not occur.

Chapter 5

Assessment of the implementation

5.1 Interface for real-time teleoperation of the Kinova Gen3 robot

One of the main goals was to build an interface for the real-time control of the robot. This part focused on the problem of taking arm joint positions and transforming them into position or speed commands for the robot.

The quality of the solutions was checked with the supervisors' psychologists by performing multiple pilot experiments. From the first trial of tests conducted in the beginning of March 2023, it was certain that the high-level control, even in C++, was not sufficient due to high delays and low-level control was necessary. More about the control methods is provided in Section 4.3.1.

By the end of March, there was a second run of the experiments using low-level joint position control. This time the latency was unnoticeable and sufficient for the first pilot experiments to be conducted. It was also found that having a GUI for the participant included in the C++ code would make running the experiment a lot easier as for the pilots it required two people to control two computers in sync. One computer was running the GUI for the participants written in Python and the other was running the controller for the robot. Both computers had to be in sync, which was a focus-intensive task for the two operators.

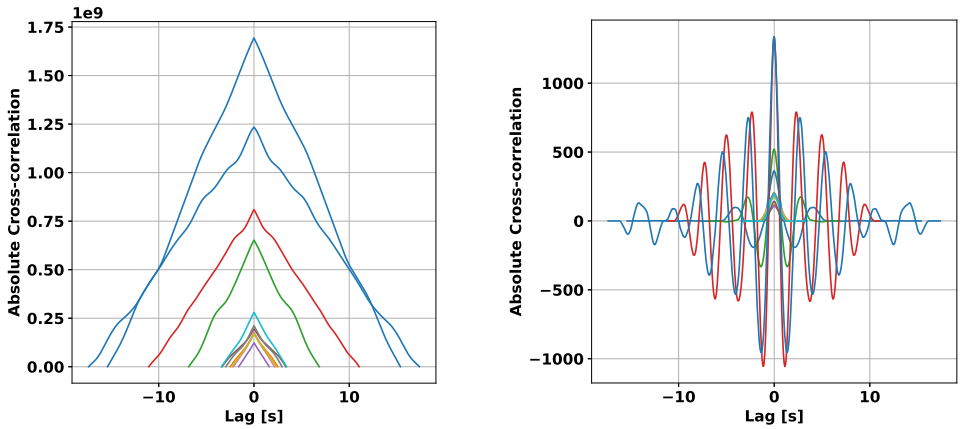
From the point of quality, it was working very well, with the exception of some jerkiness of the end-effector during the trials. There was still room for improvement from the perspective of safety, as the velocity limits were only partially implemented.

In May, the final pilots were conducted. For these pilots, a GUI was implemented, and there were no problems with it, or with the robot following the hand during the experiment. It made the experiment much easier as the participant was in charge of all necessary actions. Furthermore, the experiment changed a bit in the setup, which was explained in Section 3.2.2.

Overall, the interface was built with safety in mind and the latency was not noticeable. In the next part, the latency is evaluated.

5.2 Latency of the whole control loop

The total latency was evaluated using Qualisys recording from a dedicated experiment, where 11 trials with a gain of 1 were examined from the point of delay and position error in the direction of followed movement. Motion mainly resembled standard trials, but varied in having additional motion up and down, as shown in Figure 5.2. The first and last second of each trial were removed, as there would be added lag from the synchronization period where the robot synchronizes with the participants' hand and the event sent from the robot controller. The latter should be much smaller. The accuracy was limited by the capture rate of the cameras, which was 340 Hz.



(a) **(b)**

Figure 5.1: Cross-correlation of **(a)** the hand and robot position in followed direction, **(b)** speeds of the hand and the robot in the followed direction.

Finding the delay between two signals can be solved by looking at the maximal value of cross-correlation between the signals. When the position data were examined, as shown in Figure 5.1a, it was found that the maximal cross-correlation was with zero offset, meaning the delay in the position was less than 3 ms in all trials.

To examine further, Figure 5.2 shows the trajectories of each trial with their corresponding error in the position difference. Table 5.1 shows the average of these absolute errors in position and the maximum absolute error. Trial number 7 had a 0 ms error between the speed of the hand and the speed of the robot on the axis followed. When examined closer, which was because the hand was going at almost constant speed most of the time. The weighted average of the position error when following the hand was (1.45 ± 0.60) mm with a maximum of 5.8 mm, which proved to be very small.

Another important characteristic was the speed cross-correlation, giving insight into how fast the robot reacts to changes. This is shown in Figure 5.1b and Table 5.1. The delay between the speed of the hand and the speed of the robot on the axis followed was (16.70 ± 3.82) ms on average, which was little above the detectable threshold [5].

To conclude, the delay of the loop was very low, while the controller was proved robust as the error in position difference was also very small.

Trial	Time lasted [s]	Delay in speed [ms]	Avg. position error [mm]	Max. position error [mm]
1	15.38	17.65	0.7	4.9
2	2.5	14.71	1.0	2.6
3	6.85	14.71	1.6	4.3
4	11.05	17.65	2.4	5.8
5	1.6	23.53	0.4	5.3
6	3.36	23.53	1.0	3.9
7	2.15	0.0	1.1	2.9
8	2.96	14.71	1.3	3.0
9	2.3	17.65	1.0	4.0
10	3.42	11.76	1.2	3.4
11	17.37	17.65	1.9	5.2

Table 5.1: Statistical data from each trial, showing delay in speed, which represents how well does the robot reacts to acceleration and the position error between the hand and the robot.

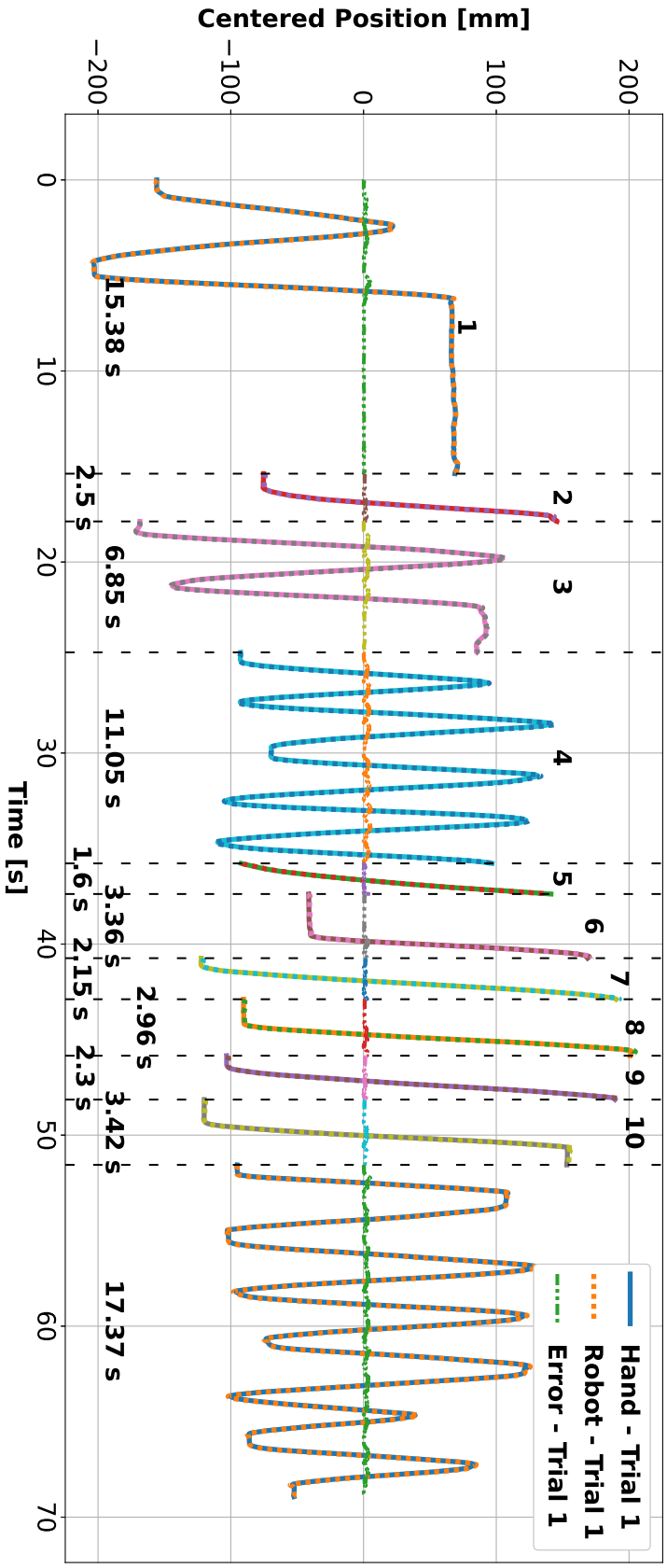


Figure 5.2: Trajectories from the trials, delimitered by a line. The number above each trajectory is the trial number, and on the bottom it is the length of the trial. Each trajectory shows the robot's position, hand position and the absolute error between those positions.

5.3 Safety of the experiment

The safety of the experiments was checked by going through the worstcase scenarios, that is, sending wrong joint angles to the controller or unsafe positions while there were no participants. During this phase, a bug was found that would allow the robot to enter forbidden joint angles, and it was fixed thereafter. The next test was conducted by running a test trial, where the tracked hand position would be moved in unexpected directions, using an artificial hand, made from a plastic tube with the markers. Sometimes this led to the robot entering joint fault mode, which could sometimes result in the robot moving unexpectedly. The forces at which the robot could hit even in these situations were measured using a certified device. It was found that if there is no clamping, or the robot goes into fault mode, the forces are small.

Since the design eliminates clamping in the motion towards the participant, which is the only direction in which it could happen as all the other axes are fixed, it was concluded that this application is safe. With that being said, experimenters should always be vigilant during experiments and be ready to stop the program or, if necessary, press the stop button, and hold the robot so it does not fall.

5.4 Graphical user interface usability

The usability of the user interface was tested during the final pilot experiments. There were no problems with it, and the participants had no problem controlling it. One suggestion during the pilots was that the improvised pedals could have provided more support and could have been harder to press. On the positive side, they felt very soft since they were literally made from sponges.

For the experimenters, it was a nice feature to have a help ready on the console by pressing “h”. Starting the experiment was sometimes harder as it uses the X client and on MacOS it does not act as a standard application. The windows had to be set up in a very specific way to see on one monitor the console and on the other the GUI.

5.5 Evaluating the experiments

There were two pilot experiments, where the first iteration of the experiments led to many changes from the preliminary experimental setup. In the first experiment, there were 4 participants. It had the main feature where the robot follows the hand done. The results have been used for a abstract for International Multisensory Research Forum 2023.

The final pilot experiment (video) included GUI, which changed changing the whole control of the experiment. This experiment had 3 participants and its main purpose was to test the final setup. In this setup, it was now

the participant who was in charge of it, by having the pedals and a mouse. Furthermore, safety of the application has improved greatly and it was now ready to get the ethical approval and have the framework ready for future experiments.

Chapter 6
Conclusion

6.1 Accomplishments

The journey of this thesis was defined by careful planning and management of unexpected challenges. The reality of software development and research was met head on with diligent time management and a four-week reserve plan that compensated for unexpected delays. Throughout the process, tasks such as developing a graphical user interface (GUI) and implementing low-level C++ code for robot control served as obstacles that were successfully overcome.

The constant need for communication with supervisors during the finalization of the experimental design and the high stakes of developing a safe application for direct robot-human interaction contributed to healthy pressure. This challenging environment honed skills such as project planning, communication, and reliability, having a profound impact on both the project and personal development.

6.2 Meeting the Objectives

The objectives established at the beginning of the project serve as a benchmark to measure the success of this study. Let's evaluate the project based on these objectives:

- It is an interface for real-time teleoperation of the Kinova3 robot via feedback from the Qualisys motion capture system.
- The robot's control is optimized, so the lag is unnoticeable for the participant.
- Assessed and ensured the safety of the application.
- Optionally, conduct and evaluate the experiments with the supervisors.
- Optionally, create a graphical user interface for running the experiments.

6.2.1 Development of a Real-time Interface for Teleoperation

The interface for real-time teleoperation of the Kinova3 robot was successfully built using feedback from the Qualisys cameras.

6.2.2 Optimization of Robot's Control

A significant accomplishment of this project was optimization of the robot control. The delay between the speed of the hand and the speed of the robot on the axis followed was (16.70 ± 3.82) ms on average. When the robot followed the hand, the error in position tracking was (1.45 ± 0.60) mm on average and its maximum was 5.8 mm. Thus, the objective was achieved through many iterations of trying different control modes and finally optimizing the low-level joint position control mode.

6.2.3 Assessment and Ensurance of Application Safety

Given the direct contact between the participant and the robot, safety was a paramount concern. The application was rigorously tested and evaluated, ensuring that the potential hazards were known and reduced to a negligible degree. The application is safe, yet as it includes a human in the loop, the experimenters should always proceed with vigilance and be ready to stop the program or, if necessary, press the stop button and hold the robot so it does not fall, in case it is needed.

6.2.4 Conducting and Evaluating Pilot Experiments

Optional pilot experiments were conducted and their results were recorded. Data from the first pilots were already submitted in a conference abstract to International Multisensory Research Forum 2023. The final pilots (video) were to ensure that the experiment setup is prepared for future experiments. The successful execution and evaluation of these experiments demonstrated the system's readiness and ability to be a helpful interface allowing to explore our biological wirings of our senses.

6.2.5 Creation of a Graphical Interface for Experiments

An additional achievement was the creation of a graphical interface to run the experiments. This feature not only enhanced the user experience but also allowed the experimenters to focus on the experiment itself rather than the execution of it.

6.2.6 Comparative Analysis with Cataldo's Study

Given the fact that study by Cataldo et al. [2, 1] has been an important reference point for this study, a comparison is indispensable. In the final pilots, the hand was flipped due to safety concerns about the robot's workspace. Due to time constraints, the participant was not seated on the right side of the

robot, as it was not known whether the cameras would catch the participants' movements properly. In their study, the participant held a joystick-like handle, which is not the case here. The participant was holding a mouse in one hand and the other hand was empty.

Next, by changing the robot, it makes the experiment more flexible and allows for 3D in the future. The delay was higher with this setup, 16.7 ms when compared to Cataldo's 2.5 ms. Here the robot carried an elastic flexible tool, but not a paintbrush, as the ending was too sharp.

In the original study, there were two virtual walls with force-feedback, which was not possible here. There was a limit on the robot's movement, constrained by its boundary box, and a tape on the table with the mouse acting as a physical barrier. However, the movement of the robot was controlled and decided by the participant, which could be more natural.

Furthermore, the experimental design contained only the active movement scenario, but not the passive as Cataldo did. In their design, they used two robot arms to achieve passive movement which would not be possible here.

In summary, the designs vary at multiple levels but are similar in others. The degree of it may change in the future, and the final decision whether the study is comparable is not conclusive. It is possible to argue on both sides as the setups were not intended to be the same.

Chapter 7
Discussion

7.1 Limitations

7.1.1 Latency

As mentioned in Section 3.3 about the network setup, the program controlling the robot was running on a MacOS machine with a Docker inside. There was additional lag added by the Docker container, even though it was setup to use the hosts network, which should have limited the added delay. However, the delay added by this was around 2 ms, which might have made the control of the robot at 1 kHz further complicated. There was a possible workaround by removing the Docker from the loop, but that would have made the program work only on a specific Ubuntu version and made it more cumbersome to workout the correct versions of libraries that work together.

7.1.2 Cameras frequency

The cameras were running at 340 Hz, which is very fast, but the controller was able to run at 1 kHz. It was possible for the cameras to run at 500 Hz, however, they did not properly mark the 3D keypoints. If there was a faster way to measure the keypoint positions, it could have improved the following error and made the tracking simpler, as there could have been only one timestamp for all the keypoints, robot's end-effector included.

7.1.3 Flexibility of the robot's end-effector tool

The robot's end effector flexible tool is very bendable. This can result in position errors when following the participants' hand, making the felt touch delayed and lag behind. This has to be taken into account when placing the participant's hand during the beginning of the experiment.

The next possible improvement might be done by having a stiffer tool, but flexible enough, not hurting the participants arm.

7.2 Future work

7.2.1 Replication of experiment by Cataldo et al.

With the setup ready, the most straight forward direction is to replicate the experiment done by Cataldo et al. [2] and compare the results by the psychologists.

7.2.2 Following in 2D plane and 3D space

One of the future directions of the project is to expand the hand following the experiment to 2D to accommodate small movements made in the other direction. This expansion is highly probable and is likely to be a follow-up project. The next expansion would be into 3D space. That would allow having complex transformation between the arm and the robot. In the project code, it was already taken into consideration; however, the safety of such experiment would require additional testing for safety and there would have to be a different model of instructions given to the participant. On top of that, the Z-axis direction was a little more problematic, as the robot tends to act slower in the Z-axis in comparison to the other axes most likely due to gravity.

7.2.3 GUI for setting up the experiment

Another possibility for expansion could be having a graphical user interface for scientists to set up the experiment, where it would be possible to write an arbitrary transformation using the 3D keypoints and converting it into the robot goal position. However, it is currently unclear, how the setup of the instructions for the participant would look like and what capabilities would be needed. One of the possible paths might be having GUI for that, similar to creating a presentation or an API allowing it to communicate with more accessible languages, such as Python.

7.2.4 Improving control and the network delay

There are possible improvements for tracking the hand and tracking the position of the joint angles of the robots. The hand position and its speed can be modeled by either constant velocity or constant jerk motion model, and then it is possible to applay Kalman filter to it. Tracking of the robot's position could also be improved by using the Kalman filter.

Finally, it is possible to reduce the network delay by running the Docker container on a Linux host machine that supports using the host network. On other platforms, such as MacOS or Windows, it is not supported, and the alternative adds delay up to a few milliseconds, which is a lot when the robot is controlled at 1 kHz.

Bibliography

[1] A. Cataldo, L. Dupin, H. Gomi, and P. Haggard, "Sensorimotor signals underlying space perception: An investigation based on self-touch," *Neuropsychologia*, vol. 151, p. 107729, 2021.

[2] A. Cataldo, L. Dupin, H. Dempsey-Jones, H. Gomi, and P. Haggard, "Interplay of tactile and motor information in constructing spatial self-perception," *Current Biology*, vol. 32, pp. 1301–1309.e3, Mar 2022.

[3] L. Patane, A. Sciutti, B. Berret, V. Squeri, L. Masia, G. Sandini, and F. Nori, "Modeling kinematic forward model adaptation by modular decomposition," in *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, (Rome, Italy), p. 1252–1257, IEEE, Jun 2012.

[4] T. Dummer, A. Picot-Annand, T. Neal, and C. Moore, "Movement and the rubber hand illusion," *Perception*, vol. 38, no. 2, pp. 271–280, 2009. PMID: 19400435.

[5] S. Kuroki and S. Nishida, "Human tactile detection of within- and inter-finger spatiotemporal phase shifts of low-frequency vibrations," *Scientific Reports*, vol. 8, p. 4288, Mar 2018.

[6] A. DiMercurio, J. P. Connell, M. Clark, and D. Corbetta, "A naturalistic observation of spontaneous touches to the body and environment in the first 2 months of life," *Frontiers in Psychology*, vol. 9, p. 2613, Dec 2018.

[7] "Definition of TELE." `https://www.merriam-webster.com/dictionary/tele.` Accessed: 2023-4-23.

[8] "Definition of OPERATION." `https://www.merriam-webster.com/dictionary/operation.` Accessed: 2023-4-23.

[9] M. Raul, "History of telerobotics." `http://alvarestech.com/temp/raul/CursoRobotica2016/Class2_HistoryTelerobotics_RaulMarin_UJI_06.ppt.pdf`, 2016. Accessed: 2023-4-28.

[10] P. Chotiprayanakul and D. Liu, "Workspace mapping and force control for small haptic device based robot teleoperation," in *2009 International Conference on Information and Automation*, pp. 1613–1618, 2009.

[11] M. V. Liarokapis, P. K. Artemiadis, and K. J. Kyriakopoulos, "Mapping human to robot motion with functional anthropomorphism for teleoperation and telemanipulation with robot arm hand systems," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Tokyo), p. 2075–2075, IEEE, Nov 2013.

[12] H. Reddivari, C. Yang, Z. Ju, P. Liang, Z. Li, and B. Xu, "Teleoperation control of baxter robot using body motion tracking," in *2014 International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI)*, pp. 1–6, 2014.

[13] F. Marić, I. Jurin, I. Marković, Z. Kalafatić, and I. Petrović, "Robot arm teleoperation via rgbd sensor palm tracking," in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1093–1098, 2016.

[14] D. Rakita, B. Mutlu, and M. Gleicher, "A motion retargeting method for effective mimicry-based teleoperation of robot arms," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, (New York, NY, USA), p. 361–370, Association for Computing Machinery, 2017.

[15] Z. Zhang, Y. Niu, Z. Yan, and S. Lin, "Real-time whole-body imitation by humanoid robots and task-oriented teleoperation using an analytical mapping method and quantitative evaluation," *Applied Sciences*, vol. 8, p. 2005, Oct 2018.

[16] F. Porta, C. T. Recchiuto, M. Casadio, and A. Sgorbissa, "Towards a framework for the whole-body teleoperation of a humanoid robot in healthcare settings," in *Social Robotics: 14th International Conference, ICSR 2022, Florence, Italy, December 13–16, 2022, Proceedings, Part II*, (Berlin, Heidelberg), p. 288–298, Springer-Verlag, 2023.

[17] L. Gutzeit, A. Fabisch, M. Otto, J. H. Metzen, J. Hansen, F. Kirchner, and E. A. Kirchner, "The besman learning platform for automated robot skill learning," *Frontiers in Robotics and AI*, vol. 5, p. 43, 2018.

[18] J. Sandoval, M. A. Laribi, J. Faure, C. Breque, J.-P. Richer, and S. Zeghloul, "Towards an autonomous robot-assistant for laparoscopy using exteroceptive sensors: feasibility study and implementation," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6473–6480, 2021.

[19] K. Darvish, L. Penco, J. Ramos, R. Cisneros, J. Pratt, E. Yoshida, S. Ivaldi, and D. Pucci, "Teleoperation of humanoid robots: A survey," *IEEE Transactions on Robotics*, pp. 1–22, 2023.

[20] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their

analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, 2019.

[21] S. Lloyd, R. A. Irani, and M. Ahmadi, "Fast and robust inverse kinematics of serial robots using halley's method," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 2768–2780, 2022.

[22] C. Jay, M. Glencross, and R. Hubbold, "Modeling the effects of delayed haptic and visual feedback in a collaborative virtual environment," *ACM Transactions on Computer-Human Interaction*, vol. 14, p. 8, Aug 2007.

[23] T. Kaaresoja, E. Hoggan, and E. Anttila, *Playing with Tactile Feedback Latency in Touchscreen Interaction: Two Approaches*, vol. 6947 of *Lecture Notes in Computer Science*, p. 554–571. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[24] J. C. V. S. Junior, S. N. Silva, M. F. Torquato, T. Mahmoodi, M. Dohler, and M. A. C. Fernandes, "Fpga applied to latency reduction for the tactile internet," *Sensors*, vol. 22, no. 20, 2022.

[25] S. Haddadin and E. Croft, *Physical Human–Robot Interaction*, pp. 1835–1874. Cham: Springer International Publishing, 2016.

[26] ISO Central Secretary, "Robots and robotic devices - safety requirements for industrial robots - part 1: Robots," ISO Standard ISO 10218-1:2011, International Organization for Standardization, Geneva, Switzerland, 2011.

[27] ISO Central Secretary, "Robots and robotic devices - safety requirements for industrial robots - part 2: Robot systems and integration," ISO Standard ISO 10218-2:2011, International Organization for Standardization, Geneva, Switzerland, 2011.

[28] ISO Central Secretary, "Robots and robotic devices - collaborative robots," ISO Standard ISO/TS 15066:2016, International Organization for Standardization, Geneva, Switzerland, 2016.

[29] P. Svarny, J. Rozlivek, L. Rustler, M. Sramek, Özgür Deli, M. Zillich, and M. Hoffmann, "Effect of active and passive protective soft skins on collision forces in human–robot collaboration," *Robotics and Computer-Integrated Manufacturing*, vol. 78, p. 102363, Dec 2022.

[30] P. Svarny, J. Rozlivek, L. Rustler, and M. Hoffmann, "3d collision-force-map for safe human-robot collaboration," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3829–3835, IEEE, 2021.

[31] Clearpath Robotics, "Kinova Gen3." `https://store.clearpathrobotics.com/products/kinova-gen-3`, 2023. Accessed: 2023-4-23.

[32] A. Rojík, "Teleoperation of Kinova3 Robot using Qualisys Motion Cameras." https://gitlab.fel.cvut.cz/body-schema/qualisys-kinova-interface/, 2023.

[33] "Ergorest 330-series." https://www.ergorest.fi/products/330-series/, 9 2021. Accessed: 2023-5-8.

[34] Kinova Robotics, "Kinova3 User Guide." https://www.kinovarobotics.com/uploads/User-Guide-Gen3-R07.pdf, 2022. Accessed: 2023-3-11.

[35] "Docker - what is a container?." https://www.docker.com/resources/what-container/, 2 2023. Accessed: 2023-5-8.

[36] "X.org." https://www.x.org/wiki/, 5 2023. Accessed: 2023-5-8.

[37] D. Durakovic, "mINI." https://github.com/pulzed/mINI, 2018. Accessed: 2023-05-26.

[38] N. Lohmann, "JSON for Modern C++." https://github.com/nlohmann/json, 2013. Accessed: 2023-05-26.

[39] Kinova Robotics, "KINOVA® KORTEX™ API." https://github.com/Kinovarobotics/kortex, 2021.

[40] B. J. Snyder, "PID C++ implementation." https://gist.github.com/bradley219/5373998, 2019. Accessed: 2023-05-26.

[41] Qualisys AB, "Qualisys Realtime SDK." https://github.com/qualisys/qualisys_cpp_sdk, 2 2022.

[42] wxWidgets Development Team, "wxwidgets." https://www.wxwidgets.org, 2018. Accessed: 2023-4-28.

[43] S. Lloyd, R. A. Irani, and M. Ahmadi, "QuIK Codebase." https://github.com/CarletonABL/QuIK, 2022.