Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Ove Baardsgaard
Kia Brekke

Development of Robot Based Rehabilitation Apparatus for Whiplash Injured Patients

Master's thesis in Product Development
Supervisor: Knut Einar Aasland
June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Development of Robot Based Rehabilitation Apparatus for Whiplash Injured Patients

Ove Baardsgaard
Kia Brekke

Master's thesis

Submission date:	11.06.2019
Supervisor:	Knut Einar Aasland, MTP
Co-supervisor:	Aleksander Lillienskiold, SINTEF Ocean

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Preface

Development of Robot Based Rehabilitation Apparatus for Whiplash Injured Patients has been a master's thesis collaboration between Ove Baardsgaard and Kia Brekke. An engagement of research and write-up lasting from January 15th 2019 till June 11th 2019.

Our work has been conducted at the Department of Mechanical and Industrial Engineering (MTP) at the Norwegian University of Science and Technology (NTNU). This is the 6th master's thesis on this topic, initiated by Firda Physical Medicine Center back in 2014. Their request was research into the development of a new, improved and versatile rehabilitation apparatus for whiplash injured patients.

From the beginning it was clear that this was more than a mechanical engineering task. The motion platform to engage neck exercises on the patient was a 7-DOF robotic arm. The previous master's thesis has focused on establishing methods for interaction with human subjects, establishing freedom and motion range, and a most of the hardware except the robot.

We have built on this work, and will in this thesis document: How we have built a smooth controller allowing for good uninterrupted head motion, how we have developed a graphical interface enabling tuning and recording of motion, how we have looked at the human-robot-interface, and how we have thought of and taken into account health and safety of the patients and operators.

Problem Description

The long term vision is to make a rehabilitation apparatus for whiplash injured patients, which is a comprehensive and complicated problem. The concept idea is to use a robot arm with a helmet-like device for fastening the head to guide a patient through neck exercises. This master thesis focus on some parts of the apparatus, which has been seen as most important at the current stage for developing a minimum viable product (MVP).

The areas of focus were:

- Control of the robotic arm, making it compliant with the ability to adjust resistance, and guide the movement through a recorded motion.
- A graphical user interface to control the functions of the robot, and store patient info such as custom programs and exercises.
- Helmet mounting mechanism which securely fastens the helmet to the robot, while keeping patient EHS in focus by disconnecting in case of emergencies or malfunctioning.

Summary

Firda Physical Medicine Centre (Firda Fysmed) is a medical clinic that specializes in whiplash injuries. They experience that the current rehabilitation apparatus in use, the Multi Cervical Unit (MCU), is not able to give the desired motion range. Firda Fysmed initiated a collaboration with NTNU to research into the development of a new, improved and versatile rehabilitation apparatus for whiplash injured patients. Previous theses have explored various motion platforms, resulting in the idea of using a robot arm with a helmet-like head mounting tool. A chair with a frame structure for mounting the robot arm was also constructed in the latest thesis.

This thesis is a continuation of the ongoing project of making a rehabilitation apparatus for whiplash injured patients. The main focus has been to develop a control system for the robotic manipulator Panda by Franka Emika, a user interface to control the robot, and further develop the head attachment connected to the robot.

After developing and testing different robot control systems, a dynamic model made the best force controller in the Cartesian space to make the robot move with very low resistance. A controller for the rotation of the end-effector was also made. A few methods for constraining the motion of the robot to a recorded path was also developed. The best method used an algorithm for finding the closest point on a path and a PID controller for constraining the end-effector position. With the force controller and this method, the robot was easy to move through any recorded path. A method for constraining the joint positions with a PID controller was developed to make the robot apply a smooth resistance if the joints approached their limit.

A user interface linking the user and the apparatus has been developed using gtkmm and Glade. The interface controls the different states of the robot and lets the user record movements and follow a constrained path. Functionality for performing an exercise with adjustable resistance, progress bar, and counter is also included.

A new helmet was designed with dimensions to fit all humans, with a thin-walled structure to minimize the distance from the head to the robot. A new helmet mounting mechanism using magnets was developed with a focus on safety and rigidity. The magnets make the connection between the helmet and the robot rigid and provide easy disconnection in case of a sudden movement from the robot.

To complete the MVP, the Cartesian force controller needs to be made more stable. The controller for rotation also needs to be improved. The user interface has some of the functionality needed in an MVP and needs to be finished, and the helmet needs to be redesigned to securely fasten the head of the patients.

Sammendrag

Firda fysikalsk-medisinsk senter (Firda Fysmed) har spesialisert seg på diagnostisering og rehabilitering av personer med nakkeskader. De opplever at dagens løsning, treningsapparatet Multi Cervical Unit (MCU), ikke gir tilbyr ønsket bevegelsesområde. Firda Fysmed startet et samarbeid med NTNU for å utvikle et nytt, forbedret og allsidig rehabiliteringsapparat for nakkeslengskadde pasienter. Tidligere oppgaver på prosjektet har utforsket ulike løsninger som resulterte i den nåværende ideen om å bruke en robotarm med en hjelm-aktig hodeinnfestning. En stol med tilhørende rammekonstruksjon for å sette roboten på ble også konstruert i den forrige oppgaven.

Denne oppgaven er en fortsettelse på det pågående prosjektet om å lage et rehabiliteringsapparat for nakkeslengskadde pasienter. Hovedfokuset har vært å utvikle et kontrollsystem for robotarmen Panda av Franka Emika, et brukergrensesnitt for å kontrollere roboten, og videreutvikle hodeinnfestningen som er koblet til roboten.

Etter å ha utviklet og testet ulike kontrollsystemer for roboten, ble en dynamisk modell det beste valget som kraft-kontroller i kartesisk rom for å styre roboten med minst mulig motstand. En kontroller for å rotere roboten ble også laget. Noen metoder for å begrense robotens bevegelse til å følge en forhåndsinnspilt bane ble også utviklet. Metoden som oppnådde best resultat brukte en algoritme for å finne det nærmeste punktet på en bane, og en PID kontroller for å begrense posisjonen til roboten. Denne metoden kombinert med en kraft-kontroller gjorde det lett å bevege roboten gjennom en innspilt bane. For å begrense leddposisjonene ble det utviklet en metode med en PID kontroller, slik at roboten påfører en jevn kraft hvis leddene nærmer seg ytterpunktene sine.

Brukergrensesnittet for kontroll av roboten ble utviklet i gtkmm og Glade. I brukergrensesnittet kan man ta opp en bestemt bevegelse, følge en begrenset bane og justere motstand. Verktøy som fremdriftsindikator, informasjon om robot status, telling av antall repetisjoner og graf som viser kraft påført av pasienten er implementert for å gjøre brukeropplevelsen bedre.

Det ble designet en ny hjelm som skal passe alle hodestørrelser og minimere avstanden fra hodet til roboten. Festemekanismen mellom hjelm og robot er re-designet med tanke på sikkerhet og minimering av slark. Den nye løsningen bruker magneter for å holde hjelmen og roboten sammen, noe som også tillater at hjelmen lett bryter av hvis roboten gjør en brå bevegelse.

For at produktet skal komme nærmere enn MVP, må den kartesiske robotkontrolleren og rotasjonskontrolleren gjøres mer stabil. Brukergrensesnittet har noe av funksjonaliteten som trengs, men må fullføres. Hjelmen må også re-designes slik at pasientens hode blir sikkert festet til roboten.

Acknowledgement

We would like to thank our supervisors Knut Einar Aasland and Aleksander Lillienskiold for their support, assistance and expertise. We would also like to thank our contact person at Firda Physical Medicine Centre, Morten Leirgul, for patience, support and valuable feedback. Thanks to Magnus Helgeby from Spark NTNU for assistance and guidance.

Table of Contents

List of Figures

List of Listings

Abbreviations

API	=	Application Programming Interface
AROM	=	Active Range Of Motion
CAD	=	Computer Aided Design
CPU	=	Central Processing Unit
DOF	=	Degrees of Freedom
EHS	=	Environment, Health and Safety
FCI	=	Franka Control Interface
GUI	=	Graphic User Interface
IK	=	Inverse Kinematics
LGPL	=	Lesser Genereal Public License
MCU	=	Mutli Cervical Unit
MTP	=	Department of Mechanical and Industrial Engineering
MVP	=	Minimum Viable Product
NTNU	=	Norwegian University of Science and Technology
OS	=	Operating System
PID	=	Proportional Integral Derivative
PLA	=	Polyactic acid
RTOS	=	Real-time Operating System
UI	=	User Interface

Chapter 1

Introduction

This master's thesis is a continuation of 5 previous master's theses and lastly a project thesis where Ove Baardsgaard contributed as one of the authors. The project thesis, available in Appendix B, continued the work of developing a rehabilitation apparatus for whiplash injured patients.

1.1 Background and motivation

Whiplash injury is a common problem around the world. Morten Leirgul, the project instigator, works at Firda Physical Medicine Centre (Firda Fysmed), a medical clinic that specializes in whiplash injuries. They experience that the patients have difficulties finding treatment for their illness. Firda Fysmed currently owns an exercise apparatus called the Multi Cervical Unit (MCU)

They say that there is a lot of issues with the MCU. It can only move around static axes, while in the most natural neck movements, the axes translate from their original position. Therefore the MCU is not able to give the desired motion range that Firda Fysmed requires.

The company that produces the MCU states on their website that: ”The BTE MCU Multi-Cervical Unit is the most effective and complete system for the assessment and rehabilitation of patients suffering from neck pain, whiplash associated disorders (WAD), and general cervical spine disorders. The MCU is an unmatched, research-backed tool that empowers physical therapists and chiropractors alike to objectively evaluate, strengthen, and restore the ailing neck and cervical spine.” (BTE, 2019b)

This project is intended to develop a product that challenges the MCU, which is the only competitor in the category. The MCU is therefore an important reference for the quality of the outcome of this project. To make a product that outperforms the MCU, it must be competitive in either value, price or both.

BTE and others (Centeno-Schultz Clinic, 2019) claim that the MCU works as intended and gives good results in regards to treating neck-related injuries. BTE has also collected a wide range of papers where the MCU has been used in successful treatment, or as neck mobility and strength testing device (BTE, 2019a).

Firda Fysmed is now able to treat patients with the MCU with good results, but they can't recommend the MCU to any other clinics because of its many issues. The goal for Firda Fysmed and this project is therefore to make an apparatus that is easy to use and outperforms the MCU so other clinics are also able to use it and can offer treatment closer to the patients.

1.2 Previous work

The previous master thesis proposed a quite specific solution, which was based on using a robotic arm to guide and give resistance to the patient's neck movement, as shown in Figure 1.1. The concept included an alpine helmet with inflatable air pockets to secure the head, a mounting mechanism to fix the helmet to the robotic arm, and a chair with a mount for the robot and the ability to raise and lower the seat. A specific robotic arm was also chosen for this concept: Panda by Franka Emika GmbH.

1.3 Problem description

The long term vision is to make a rehabilitation apparatus for whiplash injured patients, which is a comprehensive and complicated problem. The concept idea is to use a robot arm with a helmet-like device for fastening the head to guide a patient through neck exercises. This master thesis focus on some parts of the apparatus, which has been seen as most important at the current stage for developing a minimum viable product (MVP).

The areas of focus were:

- Control of the robotic arm, making it compliant with the ability to adjust resistance, and guide the movement through a recorded motion.
- A graphical user interface to control the functions of the robot, and store patient info such as custom programs and exercises.
- Helmet mounting mechanism which securely fastens the helmet to the robot, while keeping patient EHS in focus by disconnecting in case of emergencies or malfunctioning.

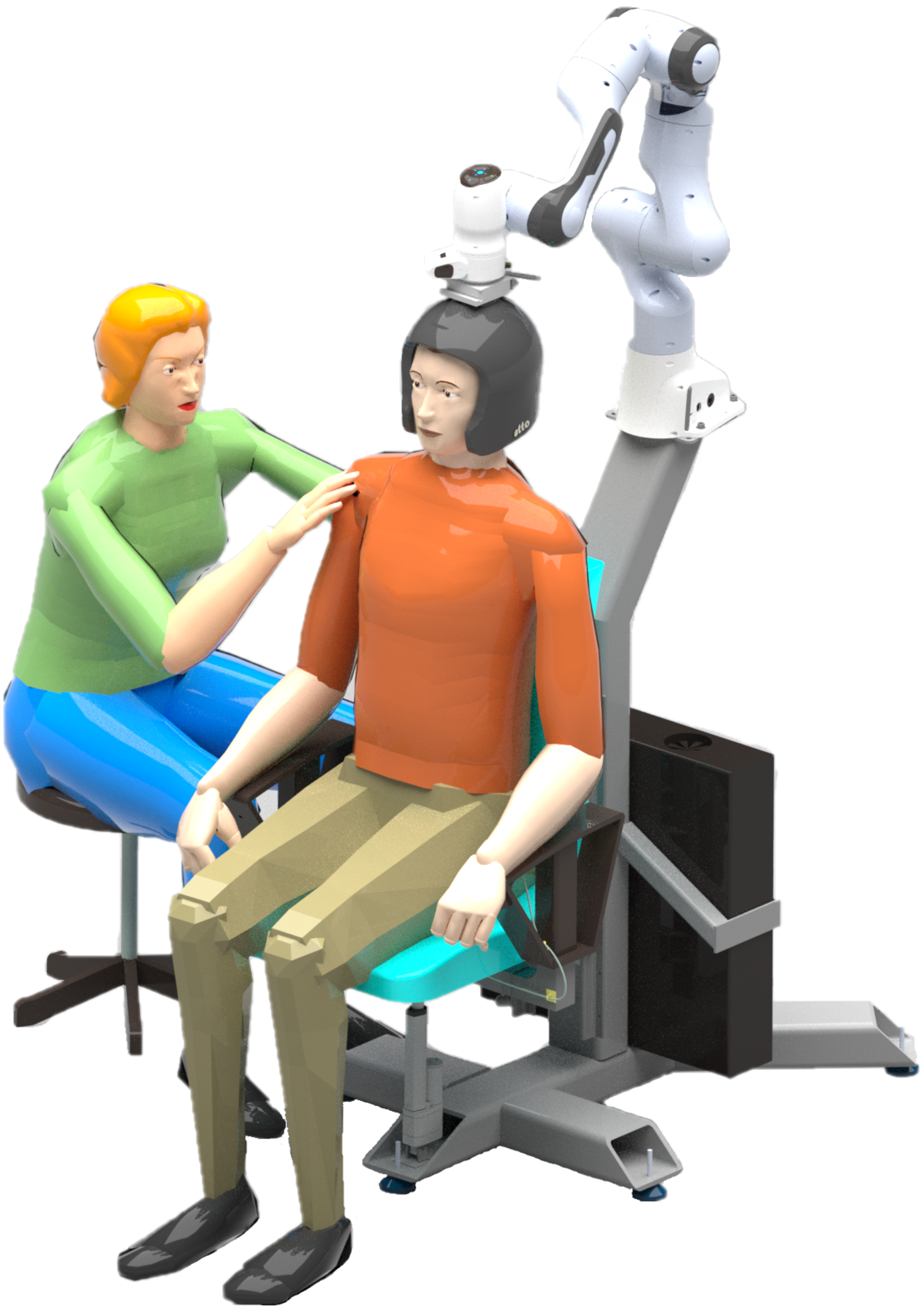


Figure 1.1: Concept from previous master's thesis by Brattgjerd and Festøy (2018)

Chapter 2

Theory

The approach to this project has been very experimental, but with some theory in mind. This chapter covers the theory which the development of a robot control system and a user interface has been based on.

The previous theses on this project by Slåttsveen and Tolo (2015), Berg and Sunde (2016), Gælok and Strand (2017), and Grøtte (2018) has gone in-depth on the theory of the human neck and the whiplash injury. Therefore, we refer to them for readings on this subject.

2.1 Terms and concepts

Flange	-	The end of the last joint of a robot arm where an end-effector can be mounted.
End-effector	-	A tool the robot uses to interact with the environment mounted at the flange of the robot arm
Task space	-	Defines the position and rotation of the end-effector
Joint space	-	Defines the joint angles of the robot arm, that together forms the robot arm pose.
Robot	-	Refers to the 7 DOF robotic arm, Panda by Franka Emika which is utilized in this project.
Jerk	-	Time-derivative of acceleration
libfranka	-	A C++ library that comes with the Panda robot to control and communicate with the robot
Struct	-	A grouped list of variables that can be accessed through a single pointer

2.2 Robot control

2.2.1 PID controller

A PID controller is a widely used control method for decreasing the error towards a set-point of a system. The controller consists of three components: a proportional (P) term, an integral (I) term, and a derivative (D) term. To use a PID controller, you first calculate an error, the difference between a desired set-point and the current state. The error is then sent to the controller, which outputs a correction to minimize the error. Figure 2.1 shows the block diagram for a typical PID controller.

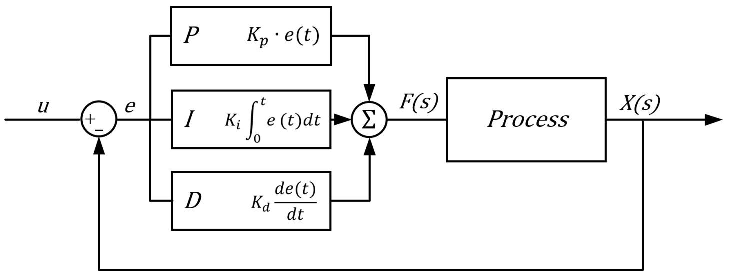


Figure 2.1: Block diagram of a PID controller from DEWESoft (2019)

Each term of the controller has a coefficient to determine their influence on the output, called K_p, K_i, and K_d. A common method of tuning a PID controller is by starting with all coefficients at zero, and slowly increasing K_p until oscillations occur. Then increase K_d to dampen the oscillations. For a robotic controller, the I term is often left at 0.

2.2.2 Admittance control

In the project thesis by Khan (2018), an admittance controller was proposed as a solution for making a stable compliant force controller. An admittance controller is based on the principles of a mass-spring-damper system (Pertuz et al., 2018). This means that the robot is controlled to behave like a mass connected to a spring and a damper which is illustrated in Figure 2.2.

From Newton's second law we have the relationship described in Equation 2.1. The force exerted by a spring is described in Equation 2.2. The force exerted by a damper is described in Equation 2.3.

$$\vec{F_{tot}} = m\vec{a} \tag{2.1}$$

$$\vec{F} = k\vec{r} \tag{2.2}$$

$$\vec{F} = c\vec{v} \tag{2.3}$$

Using newton's third law on the system, the dynamics can be written as

$$\vec{F_{ext}} = k\vec{r} + c\vec{v} + m\vec{a} \tag{2.4}$$

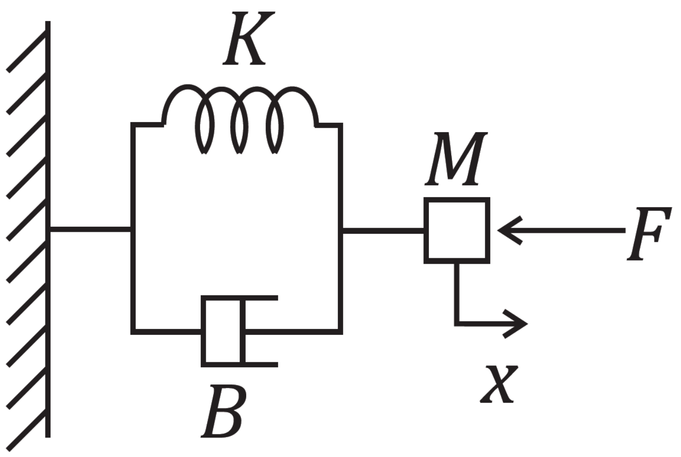


Figure 2.2: Illustration of a mass-spring-damper system from Pertuz et al. (2018)

The equation for the desired acceleration of the robot (Equation 2.5), is obtained by rewriting Equation 2.4.

$$\vec{a} = 1/m(\vec{F_{ext}} - k\vec{r} - c\vec{v}) \tag{2.5}$$

The spring will try to force the robot back to the initial position, which is undesirable in our case. The appropriate admittance controller can therefore be written as Equation 2.6

$$\vec{a} = 1/m(\vec{F_{ext}} - c\vec{v}) \tag{2.6}$$

2.2.3 Signal filtering

Torque sensors are prone to having measurement noise in their signals. The Panda robot uses force estimation to calculate forces in cartesian directions from the internal torque sensors of the robot. To get a more useful and smooth force signal from the robot, the signal can be filtered. A simple and common filter is a low-pass filter. Libfranka comes with a simple lowpass filter, `franka::LowPassFilter`. This function uses the time step, current and previous value, a cutoff frequency, and outputs a filtered signal where frequencies above the cutoff frequency are damped.

2.2.4 Jacobian

The Jacobian is a matrix that is used to translate task space velocities into joint space velocities by Equation 2.7

$$\vec{v} = J\vec{\dot{q}} \tag{2.7}$$

$\vec{v}$ is a 1×6 vector of Cartesian velocities, $\vec{\dot{q}}$ is a 1×7 vector of joint velocities (since the robot has 7 joints), and J is therefore a 6×7 Jacobian matrix. The Jacobian is dependent on joint positions, $J(q)$, and needs to be accurately defined by the physical properties of the robot.

2.3 Software technologies

This section describes the tools and software technologies used and discussed in the thesis.

2.3.1 User interface

User interface (UI) is the platform where interactions between humans and machines occur. When talking about a rehabilitation apparatus for whiplash injured patients, the user interface is the platform between the physiotherapist (the user) and the robot arm (the machine).

A Graphical User Interface (GUI) is the type of user interface that allows the users to interact through graphical icons and visual indicators (e.g windows, menus, buttons).

2.3.2 GUI tools and libraries

Gtk

GTK is a multi-platform toolkit for creating graphical user interfaces. GTK is written in C and is supported in many languages like C/C++, Java, JavaScript, Python, Ruby, etc. (The GTK Team, 2019a)

Gtkmm is the C++ interface for GTK, making it possible to create user interfaces either in C++ code or with the Glade User Interface designer, using Gtk::Builder. A user interface designer makes it easier and quicker to get started with creating a user interface without any prior knowledge to the tools or coding. Glade produces an XML-file that can be loaded by gtkmm applications.

Gnuplot

Gnuplot is a multi-platform command-line driven graphing utility. It is free to use and was created to primarily visualize mathematical functions interactively. It supports both 2D and 3D plot. It is easy to set up and easy to use. The 3D plots can either be a surface plot based on a grid of lines or 2D projection creating contour lines or heat maps (Williams and Kelley, 2018). Gnuplot can't project 3D-models from e.g obj-files.

OpenGL

OpenGL (Open Graphics Library) is a multi-platform application interface for 2D and 3D graphics. OpenGL interacts with the graphics processing unit (GPU) to acquire hardware-accelerated rendering. Hardware acceleration involves computing tasks in hardware (here the GPU) to decrease latency and increase throughput. OpenGL is therefore powerful, but it is more complex to set up and use, compared to for example Gnuplot. For clarification, throughout the project OpenGL refers to modern OpenGL (version 3.3 or higher).

2.3.3 Threading

Threading is the technique of splitting the queue of actions happening in the processor. For the user, it seems like different sequences are happening in parallel. In theory, there is no limit to how many threads you can have, but the practical restriction of the number of threads is limited by the resources of the computer.

2.3.4 Real-time operating system

A real-time operating system (RTOS) is an operating system (OS) using a real-time kernel to handle the processing of the system. RTOS uses priorities to achieve the lowest possible latency at any cost (wiki.archlinux.org, 2019). The highest priority task gets access to the resources needed in the CPU. Features like reliability, power-saving, and throughput are necessarily sacrificed in an RTOS (Ubuntu Documentation, 2019).

Most operating systems (OS) do not include a real-time kernel but rather a standard time-sharing kernel that schedule tasks for efficient use of the system. Real-time patches for Linux are needed for operating the Panda robot (Franka Emika GmbH, 2017a).

Chapter 3

Development Process

The objective of this project was to continue developing an apparatus towards an MVP. Also, it was important to make a good foundation for further development. Both students were involved in all processes. It was a strategy to work closely together, to be able to collaborate, delegate and discuss solutions on all areas.

This chapter starts with a description of the working methods applied, then it goes into detail about the development of the product, divided into robot control, user interface, helmet and helmet mounting mechanism.

3.1 Development methods

During the development process of the apparatus, different development methods have been applied. This section will describe the methods and how they were used in this specific project.

3.1.1 Agile development methods

To keep an organized overview of what was being worked on, and to divide tasks into manageable pieces, the working method Scrum, often used in software development, was applied.

Scrum is an iterative agile framework for managing product development (Scrum.org, 2019). It consists of several elements, whereas only some of them were applied. Backlog, Scrum board and a lightweight version of sprints were elements explored in this project.

Backlog

A backlog is simply an ordered list of tasks to be done. All tasks are created to achieve a product requirement. The advantage of having a backlog is the constant overview of what is left to be done. It is easy to compare tasks and requirements to evaluate what is most important to do next when it is visual and written down.

The challenge when creating a backlog is to keep the tasks small and descriptive enough. Even though the end goals are known, it can be difficult to find out how to get there and break down the challenges into something manageable. During the project, the backlog was edited and updated once a task became clearer, or a method didn't work as expected.

Sprint

A sprint is a time-based iteration. In this project, one week was the most common duration of a sprint. During that one week, tasks from the backlog were picked towards a goal that would be accomplished a certain week.

Sprints enable predictability and low complexity due to reachable goals set for a limited period. After each sprint, it is possible to adjust future goals and tasks ensuring progress towards the main objectives.

Scrum board

The tasks chosen for a sprint was placed in a scrum board. The application Trello (Trello.com, 2019) was used to organize a scrum board. The board consisted of columns. One column for all tasks anticipated that week (TO-DO), one column for tasks being worked at (IN-PROGRESS) and finally a column for tasks that were completed (DONE). In Trello, it is possible to assign users to a task, so it was easy to keep track of who was working on which task at all times.

Ideally, when using a scrum board, it is easy to begin a new task once one is done. You simply pick the next one in the TO-DO column. The tasks are supposed to be ordered. However this is tedious to keep updated at all times. Sometimes tasks were bigger than anticipated or unexpected challenges arose.

Daily log

To make it easier to remember what had been done during the project, and how problems were solved, small daily logs were written at the end of each day. The logs were a great opportunity to formulate any problems, making the problems easier to face or to see new solutions. The logs were also an asset when describing the development process.

3.1.2 Rapid prototyping

Rapid prototyping is a method where prototypes are made quickly to test concepts and ideas. In this project, CAD and 3D printing were used as rapid prototyping tools. With 3D printing, a prototype can be made in a few hours and give instant feedback of the concept, how it works and what can be improved.

3.2 Compliant Robot Control System

The first objective of the problem formulation was to make a control system that made the end-effector move with as little resistance as possible. This problem can be split into three parts: getting a Cartesian force reading from the sensors in the robot, converting the force readings into a Cartesian velocity control signal, and making the robot perform the desired Cartesian velocity.

Also, there has been made an attempt at making the same system with rotation. Rotation is important to make an MVP but is also a more complex task. Lastly, a system for constraining the motion of the robot to follow a pre-recorded path was developed.

3.2.1 Robot control loop

The robot is communicating with a computer by an Ethernet cable, sending data at a rate of 1000 Hz. The computer runs a C++ program with the libfranka library. The program contains a control loop that reads and sends data to the robot. If the calculations in this loop are too time-consuming, causing the loop to run slower than 1000 Hz, the robot will stop and throw an error.

In the robot loop, all the information about the robot can be obtained from `franka::RobotState` which is saved in `robot_state` every iteration. In this project, the program is divided into two threads to be able to run the robot loop simultaneously with the user interface. To share data, such as information from the robot to the GUI, a global struct `robot_data` is passed to the threads. Information added or edited in this struct can then be accessed by the threads simultaneously.

Many calculations in the control loop are often dependent on the time interval between each loop iteration. The time interval, Δt, is defined by Franka to always be 0.001s (Franka Emika GmbH, 2017a).

3.2.2 Velocity Controller

To make the robot move, one can either use velocity or position control, in either task space or joint space. Libfranka has built-in controllers for all these methods. A desired velocity or acceleration is given in task space from the force controller described later. Therefore a task space velocity controller needed to be developed.

Cartesian velocities

The initial idea was to use the built-in controller called `franka::CartesianVelocities` which takes a 1×6 array of velocities in X, Y and Z direction and rotation about those axis (Franka Emika GmbH, 2017b).

When using this controller, there are strict rules for which values that can be commanded. The

Franka Docs states that in Cartesian space, there are maximum limits for velocity, acceleration, and jerk (Franka Emika GmbH, 2017a). By sending velocity values that exceed these limits, the robot immediately throws a `cartesian_motion_generator_velocity_limits_violation` error, and similar for acceleration and jerk.

To avoid these errors, functions for limiting the velocity, acceleration, and jerk were made. After setting a desired velocity, `vel_desired` ($\vec{v}_{desired}$) and getting the robot's previous velocity `robot_velocity` ($\vec{v_0}$) from `robot_state.O_dP_EE_d`, the desired acceleration $\vec{a}_{desired}$ and the desired jerk $\vec{j}_{desired}$ were calculated. The jerk is then limited to the values mentioned in the Franka Docs.

$$\vec{a}_{desired} = \frac{\vec{v}_{desired} - \vec{v}_0}{\Delta t} \tag{3.1}$$

$$\vec{j}_{desired} = \frac{\vec{a}_{desired} - \vec{a}_0}{\Delta t} \tag{3.2}$$

$$\vec{j}_{limited} = \min(j_{limit}, \left\|\vec{j}_{desired}\right\|)\hat{j} \tag{3.3}$$

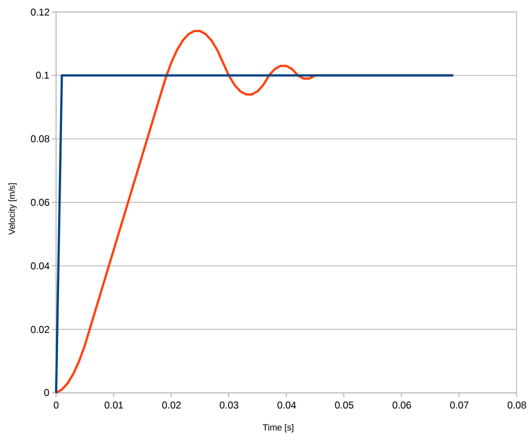
Furthermore, the new acceleration and the new velocity were calculated based on the limited vectors.

$$\vec{a} = \vec{a_0} + \vec{j}_{limited}\Delta t \tag{3.4}$$

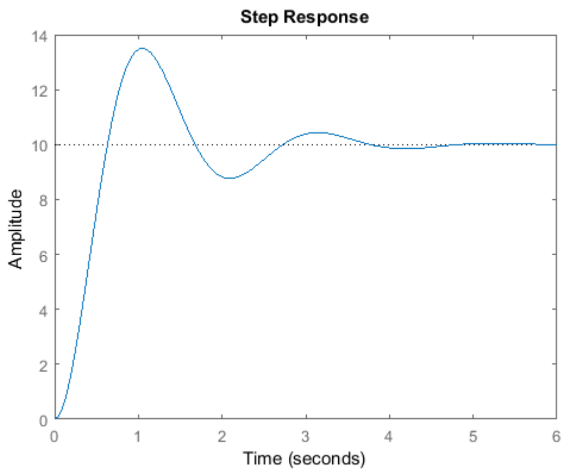
$$\vec{a}_{limited} = \min(a_{limit}, \|\vec{a}\|)\hat{a} \tag{3.5}$$

$$\vec{v} = \vec{v_0} + \vec{a}_{limited}\Delta t \tag{3.6}$$

The limiters did not work as intended, as the discontinuity errors were still present. It was hard to find an explanation of why, as there were no apparent faults in the code. Therefore, a spreadsheet to simulate the behavior of the robot and limiters were made. On the spreadsheet, it was possible to see the values of position, velocity, acceleration, and jerk of each iteration, and plot them. When simulating a step response in the spreadsheet, the plot looks similar to other step responses found online as shown in Figure 3.1.



(a) Step response simulated in a spreadsheet



(b) Step response from Control Tutorials for MATLAB and Simulink (2019)

Figure 3.1: Representation of measured force applied to the end effector

When trying to run the robot through the same step response test, with the same limiting functions and the same parameters as in the spreadsheet, the velocities did not match as shown in Figure 3.2. The correlation between the commanded acceleration and the performed acceleration was found to be a factor of about 2.59155. In an email to the customer support of Franka Emika, the correlation factor was presented and questioned, but their response did not explain or solve the issue (Cristian Kodura, Franka Emika, 2019).

By multiplying the commanded acceleration with the correlation factor, the issues were solved. The reason behind the factor was still a mystery. The velocity, acceleration and jerk limiters now worked as intended, but other errors appeared. Even though the Cartesian limits were satisfied, there was still no guarantee that the joint space limits were satisfied. The simplest way to avoid this was by setting the Cartesian limits low enough, so the limits of each joint were never reached. By gradually lowering the Cartesian limits, a stable state was achieved with an acceleration limit value of $2\ \mathrm{m/s^2}$, and a jerk limit value of $1000\ \mathrm{m/s^3}$, 6.5 times lower than the limits provided by Franka Docs (Franka Emika GmbH, 2017a).

By setting the Cartesian limits much lower than the maximum values mentioned on Franka Docs, we feel the potential of the robot is constrained. With the heavy limits needed on acceleration and jerk, a solution where all the joints are limited independently might be less restrictive. The `jointVelocities` interface needed to be explored.

Joint Velocities

The joint velocities interface requires inverse kinematics to transform the task space coordinates to joint space. Inverse kinematics is the problem of finding the joint variables in terms of the end-effector's position and orientation (Spong et al., 2006). To transform the Cartesian veloc-

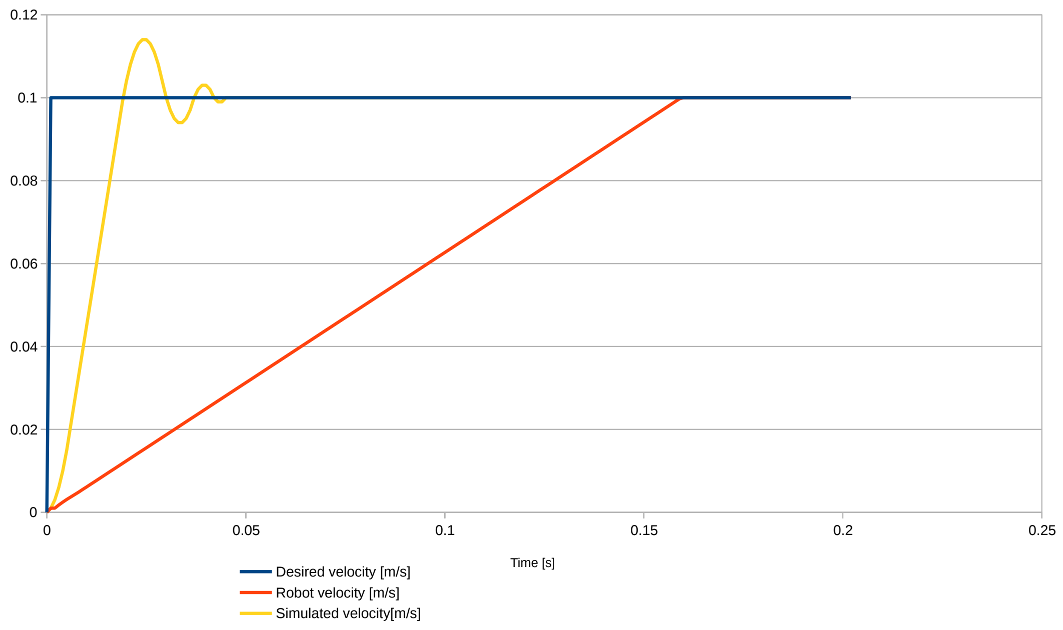


Figure 3.2: Plot showing the performed velocity of the robot during a step response test, compared to the simulated values. Both have limiters with acceleration limit $6m/s^2$ and jerk limit $1000m/s^3$

ities to joint velocities, the Jacobian is used, which is described further in Chapter 2, Section 2.2.4.

The Jacobian can be complex to calculate, but libfranka provides functions to solve it for the Panda. The `franka::Model` class has a function called `zeroJacobian` which outputs the Jacobian.

To get joint velocities from a set Cartesian velocity, the inverse of the Jacobian is needed. As the Jacobian is not square, the inverse cannot be calculated directly (Spong et al., 2006, p. 151). One solution is to calculate the pseudo-inverse of the matrix (Spong et al., 2006, p. 151). A forum post on Franka Community provided an easy to use C++ function for calculating the pseudo-inverse (Franka Community, 2018).

The robot could now be controlled by setting a desired velocity in Cartesian space, calculating joint velocities using Equation 3.7, and sending $\vec{\dot{q}}$ to the `jointVelocities` interface. J^+ is the pseudo-inverse Jacobian.

$$\vec{\dot{q}} = J^+\vec{v} \tag{3.7}$$

Strangely, the robot did not perform the velocities commanded. When comparing the pseudo-inverse Jacobian to the results of an online matrix inversion calculator (Comnuan, 2018), the values of the Jacobian was much lower. Another method of solving the pseudo-inverse attempted by using functions from the Eigen package as proposed by an answer on StackOverflow (Sean, 2017). This solution worked perfectly, and the velocity of the robot was equal to the desired value.

A previously unknown benefit of using the JointVelocities interface is that the velocity, acceleration, and jerk of all joints are automatically limited. As Franka Docs states: ”As of version 0.4.0, rate limiters for all real-time interfaces are running by default.” (Franka Emika GmbH, 2017a). This means no encounter with the velocity, acceleration or jerk errors previously faced while using the Cartesian interface.

3.2.3 Force input

The Panda has a built-in command for getting an estimated force input at the flange, `robot_state.K_F_ext_hat`. This is a 1×6 vector containing forces in X, Y, and Z direction at the flange, as well as torque around those axes. The Panda does not have a force sensor at the flange, rather the values are calculated from torque sensors in the joints (Cristian Kodura, Franka Emika, 2019).

The values from `K_F_ext_hat_K` seems to be a bit inaccurate. During a test, the robot was controlled to move with constant velocity in a step response as shown in Figure 3.3. As shown on the same figure, the values from `K_F_ext_hat_K` jumps when the robot is accelerating and no external forces acting on the robot. In an ideal situation, the values would stay at zero at all times when no external forces act on the robot.

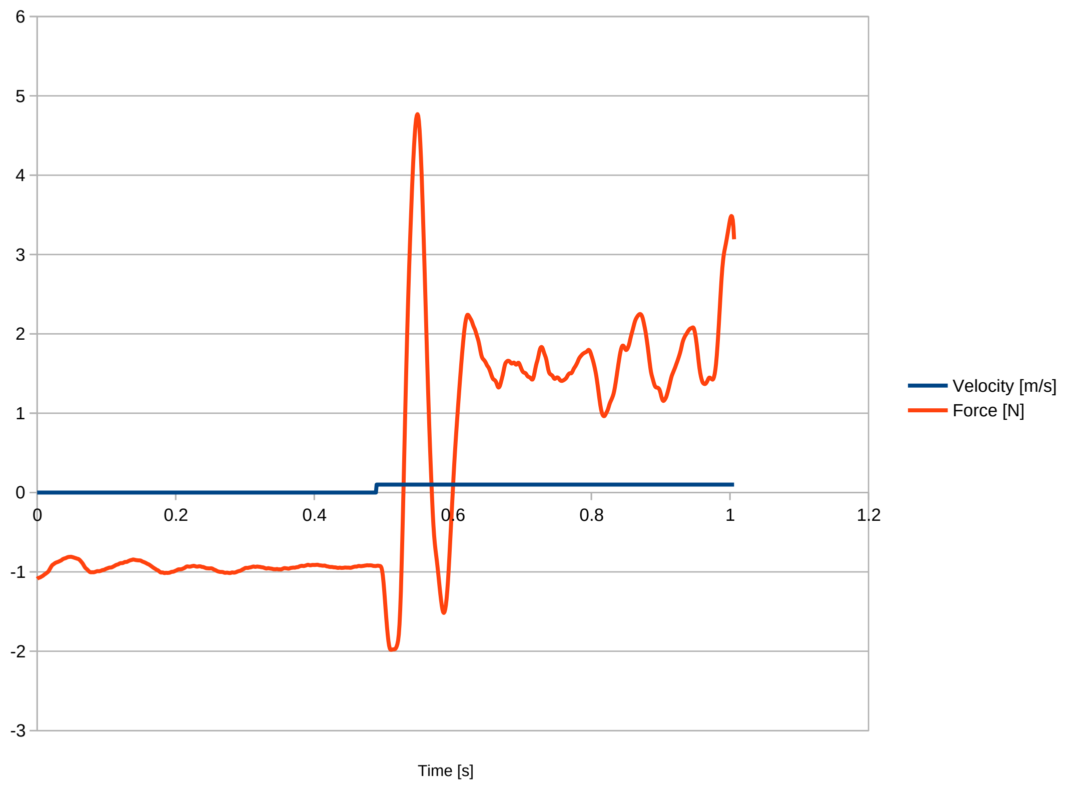


Figure 3.3: Plot of force estimation and velocity during a step response test

When using this type of force estimation in a force controller, the inaccuracy when the robot is moving is an issue. As the robot moves, the force estimation outputs a force that is sent to the force controller. The force controller again makes the robot move, and the effect propagates. This is a feedback issue that causes the robot to oscillate, inducing an unstable behavior.

It is possible to after-mount a force-torque sensor on the flange, such as an ATI Axia80 Force-torque sensor (ATI Industrial Automation, 2019). The advantage of an external sensor is that the sensor is most likely more sensitive, and is not affected by the robot's own acceleration. One disadvantage is that it adds an extra distance of 25mm between the flange and the patient's head. This reduces the torque the robot can apply. It also adds an extra cost to the final product.

An article about force-torque sensors (Maw, 2018) mentions the need for an external force-torque sensor in collaborative robots with built-in joint-torque sensors. They say that you don't get good results with joint-torque force approximation, especially at low-cost. And that the best low-cost solution is to use a low-cost robot with an after mounted precise force-torque sensor. Worth noting is that the interviewees are representatives for ATI and Robotiq, two force-torque sensor companies.

A force-torque sensor was not obtained for this project. Instead, the controller was further developed using libfranka's force estimation, with the possibility of adding a force-torque sensor later.

3.2.4 Force controller

Our main goal is to reduce the force required to move the end-effector as much as possible. One way to do that is to use the force input and feeding it back to the motors in the robot. The simplest type of control is by Newton's second law: $F = ma$, where F is the force sensed by the robot, a is the desired acceleration of the robot, and m is a virtual mass.

To reduce the apparent inertia, m needs to be reduced. But as m reaches zero, the desired acceleration increases to infinity. This makes the robot react abruptly to even small force inputs. Along with the feedback issue of the force input, the robot moved with increasing oscillations. Such oscillations are highly undesirable, and methods for making a more stable controller needed to be introduced.

The most common method for developing a force controller is an admittance controller or a PID controller. The last alternative is making a custom controller based on a dynamic model, which models the desired physical behavior of a virtual object.

The development of the force controller is divided in two, where the first part is translating the end-effector in Cartesian space, and the second is rotating the end-effector. The focus in this thesis was on developing the translation force controller, as rotation is more complex and difficult to develop.

Admittance controller

The admittance controller was tested by manually defining the parameters m, the virtual mass, and c, the damping coefficient. The testing started with a high m value of 10kg, and a c value of 0. The motion was stable but had a slow response due to the high inertia. By reducing m, a quicker response was achieved, but at some point, the response was overcompensating and the robot started oscillating. Landi et al. (2017) mentions that this is because the virtual mass is lower than the physical mass of the robot.

To reduce the oscillations, damping to the system can be introduced by increasing c. The system can be tuned by increasing c while lowering m until minimum resistance, but still stable motion. With the best combination of values achieved, the compliance still wasn't satisfactory as a considerable force was required to move the robot.

PID controller

Khan (2018) suggested using a PID controller as a force controller. In this case, the error to be fed into the controller is the difference between the current force exerted on the robot F_{ext}, and the desired force, which is zero. The output of the controller is then the control signal, which in this case is the desired velocity of the robot (Khan, 2018).

When increasing the K_p value, the robot quickly started oscillating. In theory, this could be dampened using the D term, but trying to increase K_d made the motors in the robot emit loud unpleasant noise due to rapid, small velocity changes. The D term is highly sensitive to noise (Cooper, 2015). The force input was filtered as much as possible, as described in Section 2.2.3, but the D term still had too much noise.

Dynamic modelling

Control systems are often based on dynamic modeling, which means making a dynamic model of a system and transferring its physical properties to the control system. The admittance controller is an example of a dynamic model of a mass-spring-damper system.

In this project, the desired behavior was to move the robot with as little resistance as possible, while having a stable behavior and don't make any movement without any force input. A dynamic model of this can be an object of little mass on a surface with friction. This object will be easy to move but will stop and stand still if not pushed. This could be visualized with for example moving a piece of cardboard over a table.

To make a dynamic model of this object, the inertia of the mass (Equation 3.8), the Coulomb friction with static friction (Equation 3.9) and the kinetic friction (Equation 3.10) are combined.

$$\vec{F_m} = m\vec{a} \tag{3.8}$$

$$\vec{F_{fs}} = -\vec{F_{ext}} \tag{3.9}$$

$$\vec{F_{fk}} = -\mu m g \hat{v} \tag{3.10}$$

The dynamic equation for the model is shown in Equation 3.11, where $\vec{F_f}$ is either $\vec{F_{fs}}$ or $\vec{F_{fk}}$. The static friction is used when the applied force is lower than $\vec{F_{fk}}$ or the velocity is zero. The kinetic friction is therefore used when the applied force is over $\vec{F_{kf}}$ or the object is still moving.

$$\vec{F_{ext}} = m\vec{a} + \vec{F_f} \tag{3.11}$$

To translate the dynamic model into commands for the robot, the Equation 3.11 is rewritten to Equation 3.12.

$$\vec{a_{desired}} = \frac{1}{m}(\vec{F_{ext}} + \vec{F_f}) = \frac{1}{m}\sum \vec{F} \tag{3.12}$$

By this equation, when static friction is applied, the acceleration is zero. This would mean that the robot could have a constant velocity, but by defining the kinetic friction to be applied if the robot had a velocity, the kinetic friction makes the robot decelerate until it stops.

In this controller, the parameters m and μ needed to be tuned. It started with a large m of 10 kg and μ value of 0. When decreasing m, μ was increased to dampen oscillations. The friction seemed to be very effective in damping oscillations without adding too much extra force. The best values obtained was $m = 2.0$ kg and $\mu = 0.01$. The object simulated with these valued would equal for example a small steel cube sliding on ice. After testing this, the robot seems to behave about the same as a steel cube would do.

This controller was the most satisfactory as the resistance needed to move the robot was the lowest. The robot behaves differently when moved to different positions, and in some positions, the robot starts to oscillate. There seem to be no simple solutions to avoid oscillations totally. Landi et al. (2017) mentions the same issue with a similar robot and develops a method for dynamically changing the damping coefficient of an admittance controller to dampen any oscillations.

To use the robot as an exercise apparatus, it is necessary to be able to add resistance to the movement. One simple approach is by adding another friction force with the wanted resistant force, F_w. This will not simulate the behavior of a cable apparatus such as the MCU since there will not be applied any resistance force if the patient stops the movement. We believe this is a safer method for this kind of exercise, but it remains to be tested and discussed with Firda Fysmed.

The resistance is only wanted when the patient moves from the start point to the end point of the path, and not when they move back to the start position again. Therefore, the fraction of completion of the motion is calculated, and when 99 % of the motion is completed, the resistance is set to zero. When the patient moves back to the start position again, the resistance is set to the desired resistance once more.

Joint impedance

Before the robot loop is started, the impedance of the joints needs to be defined. This defines the stiffness in a spring-damper model which controls the joints. It was discovered that setting a low joint impedance dampens the oscillations caused by the force controller. A lower impedance makes the joints less stiff.

By experimenting with the impedance parameters, the controller became more stable. One issue with low impedance is that the accuracy of the robot is lowered due to the softness, but it can also make the robot feel more human and safe for the patient.

Joint position limits

If the joints are moved outside the limits described in Franka docs (Franka Emika GmbH, 2017a), the robot stops and throws a `joint_motion_generator_position_limits_violation` . To avoid this, the joints could instead resist movement approaching those limits. One idea of how to implement this is to use a PID controller.

First, the desired joint positions need to be calculated from the desired joint velocities. Then, the positions are clamped to the limiting values. If the clamped values are sent directly to the robot, a very hard deceleration and jerk occur, causing an unstable behavior. Instead, the difference between the clamped positions and the desired positions is sent as an error input to a PID controller. The PID output is then added to the desired position, which corrects the desired position to not move outside the limits.

When tuning the PID controller, increasing K_p value was used to increase the stiffness of the stopping motion when a joint hits its limit. By increasing the K_d value, the damping is increased. The values were adjusted until an appropriate behaviour was achieved, with $K_p = 0.001$, and $K_d = 0.00002$. This gave a firm, but not harsh stop, and it did not spring back too much.

3.2.5 Force control rotation

For simplicity, the force controller developed in this thesis does not include rotation, keeping the orientation of the end-effector constant. Being able to rotate the end-effector is important to make an MVP. An attempt at including rotation in the force controller has been made, based on the same principles as the Cartesian force controller.

For rotation, it was tested to use the same dynamic model as developed for the Cartesian controller but using inertia, I, instead of mass. The inertia is a constant that is manually defined, but to get a feeling of the physical properties, the equation for the inertia of a solid ball can be used: $I = \frac{2}{3}mr^2$

After testing this out, a radius of minimum 0.4 m was needed to make the controller stable.

Compared to the low resistance of the Cartesian force controller, rotation is much heavier. It seemed difficult to find any combination of variables that lowered this resistance.

3.2.6 Constraining motion

One of the requirements for the exercise apparatus was that the patients should be able to precisely repeat a custom motion with the help of the apparatus. One of the ideas for how this could be done is to first track the motion while the robot is moving without resistance and then constrain the motion to that tracked path.

Recording a path

To record a path which the robot is to follow afterward, the Cartesian positions of the end effector are obtained with `robot_state.O_T_EE`. The positions are pushed to a vector in the shared `robot_data.tracking_data` which can be read from another thread. This is needed because saving the data may take some time and exceed the 0.001s time limit for the robot loop.

When the tracking is done, the data is then saved to a .csv file which then contains one set of X, Y and Z coordinates on each row. The .csv format was chosen since it is easy to write to with simple delimiters, and can also be easily opened in spreadsheet software to visualize the points for debugging.

The position is initially obtained at the same rate as the robot loop, which is 1000 times per second. To avoid an unnecessarily detailed path and a large amount of data, a quick solution is to sample the data. A sampling rate of 100 times per second produced a smooth path. A low sampling rate leads to a more coarse path where the sampled points are felt by hand when moving the robot through the path.

Constraining velocity direction

Our initial idea was to take the output of the force controller, a Cartesian velocity, and manipulate that vector to be in the same direction as the wanted constraining direction. The first test was done by making a unit vector $\hat{\vec{u}}$ in some direction. The Cartesian velocity, $\vec{v}$, was then projected onto $\hat{\vec{u}}$ using the dot product $\vec{v_p} = \vec{v} \cdot \hat{\vec{u}}$. This worked as intended, and the robot was constrained to only move along the chosen vector.

The flaw with this method is that it does not take into account the position of the robot. So it does not follow the path, but rather follows the direction of the path. This can lead to an offset from the desired position that also varies due to inaccuracies in the control system.

Following a path of multiple points

An exercise motion is made up of many points that together form a set of lines, called a path. Previously, the robot followed a single vector. To make it follow a path, one approach is to find the point in the set of points that is closest to the position of the end-effector. Then, a vector can be made from that point, and the second closest point in the set as illustrated in Figure 3.4. By constraining the velocity to this vector, the constrained direction should update and make the robot follow the path.

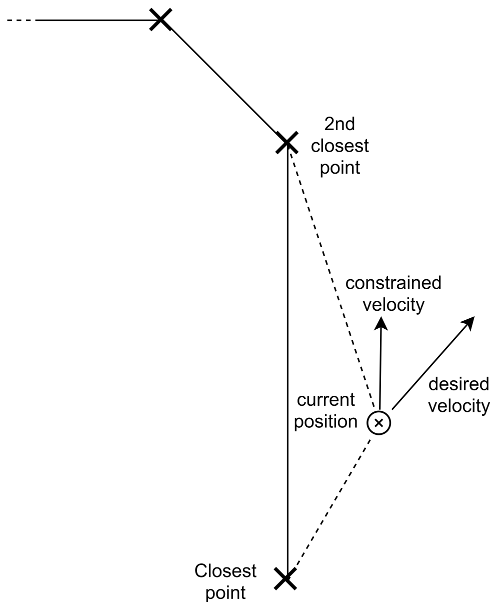


Figure 3.4: Method of constraining velocity in the direction of the closest line

When this method was tested, it worked to a certain degree but struggled in the corners of the path. Because the robot could have an offset, and also move a short distance between the time steps, the robot can skip between the two vectors as shown in Figure 3.5 and move in another direction than desired.

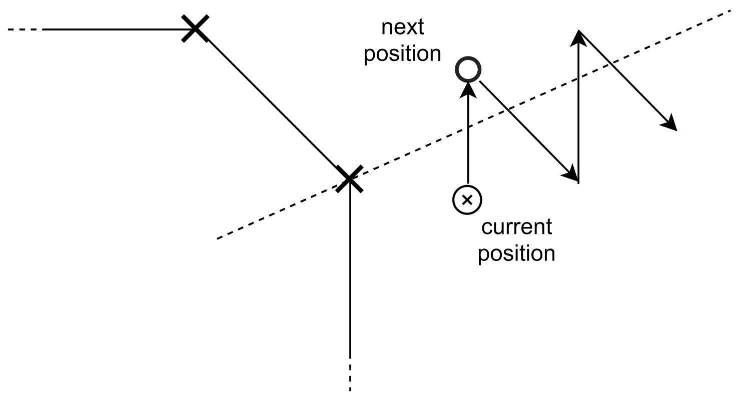


Figure 3.5: Issue with velocity constrain method when approaching corners of the path

To avoid this behavior, a new approach was developed based on position control instead of velocity control. First, the desired position was calculated by integrating the desired velocity from the force controller. Then the distance from the desired position to the closest point on each line segment of the path was found. To find the closest point on a line segment, the desired position is projected onto a line made from two line points. If the projected point is between the two line points, the function should return the projected point, or else return the closest line point. The calculated point from all line segments with the smallest distance was chosen to be the new desired position, as shown in Figure 3.6.

To make the robot move to the desired position, it was initially thought that the new desired

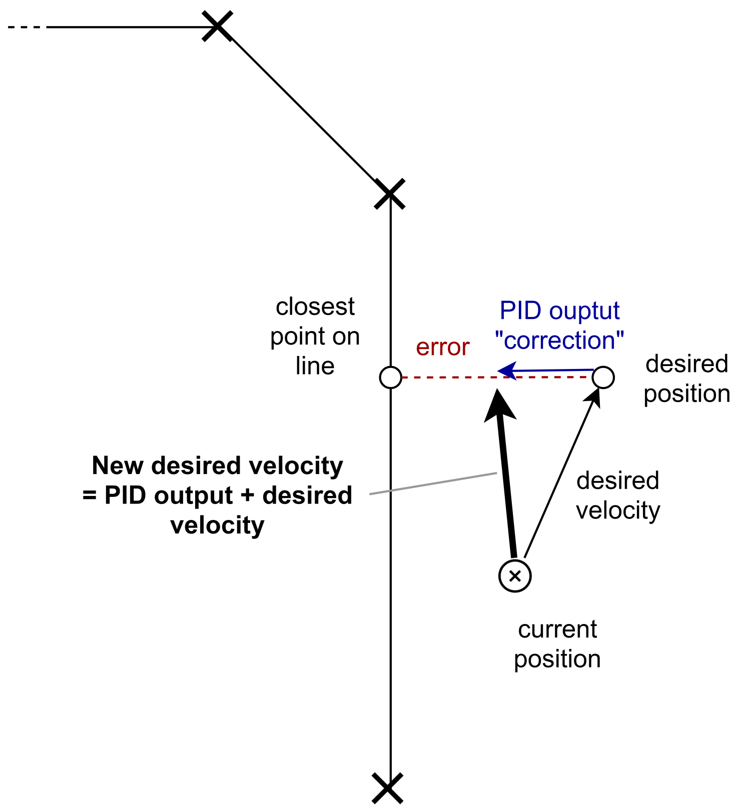


Figure 3.6: Method of constraining the position of the robot with a PID controller, to steer into the desired path

position could be calculated back to a new desired velocity and then sent to the robot. However, this causes an unstable behavior since it is an underdamped system (Rao, 2007). A common solution is to use a PID controller, with the difference between the desired position and current position as the error input, and a ”correction” as output. This correction is added to the desired velocity, to steer the robot into the path in a controlled manner.

After tuning the PID controller, the robot can follow the path perfectly. With the force controller and this method, the robot was easy to move through any recorded path. However, the derivative term of the controller generates an audible noise in the motors of the robot if tuned too high.

3.3 User Interface

The second main goal of this project was to make a user interface for controlling the robot. The user interface is intended to be the primary tool used by the medical staff to conduct a rehabilitation process. The idea is an application where each patient has personalized pages where programs can be edited and carried out, where exercises can be adjusted and the training program can be documented. It is also supposed to be the interface towards the robot, to monitor the robot state and control its behavior and fetch sensor information.

During this project period, a user interface was created with two objectives in mind; as an aid for the developers and as an interface for the end user (a physiotherapist or other medical staff).

3.3.1 Previous work

The design of the user interface is based on research done in a previous master thesis by Gælok and Strand (2017). They collected valuable information from Firda Fysmed on the subject of what they wanted in a user interface. It resulted in a set of requirements and a prototype of a design. At the beginning of this project, the list was reevaluated and updated in cooperation with Firda Fysmed. The requirements are reproduced in the list below.

- Personalized page for the patient
- Adjustable resistance from the robot
- Possibility to measure the active range of motion (AROM)
- Possibility to measure the force applied by the patient
- Possibility to illustrate and measure deviation from ideal movement
- Possibility to make custom exercise programs
- Possibility to make an exercise by guiding the robot manually, record movement and save as an exercise option
- 3D projection of movement performed by the patient

The most important requirements were to create a GUI that supported control of different robot states, tracking of movements and running an exercise. The other requirements were considered secondary, but to get a realistic impression of the concept idea, some features were added without functionality.

3.3.2 Graphical plotting

Several of the requirements are ideally presented through graphs (AROM, deviation from ideal movement, etc.). Graphs can be a great tool during development as well. When searching for a suitable graphical tool, the possibility for projecting 3D models was also taken into account in addition to graphical plotting abilities.

Another concern that needed to be considered, was compatibility with a specific programming language. Libfranka is a provided C++ library needed to achieve low-level control of the robot (Franka Emika GmbH, 2017a). Therefore a graphics tool for C++ was sought-after.

All the different factors were considered, and an ordered list of requirements for a graphics library was made:

- C++ library
- Possible to make linear graphs
- Possible with live updating of graphs
- 3D plotting possibilities

Choice of graphics library

Searching for options resulted in two apparent alternatives: Gnuplot and OpenGL. They are two commonly used tools and they are both well documented. Gnuplot is simple to use, but only available for plotting, not model rendering. OpenGL is more complicated, but also more powerful and is mainly a 3D graphics library.

Gnuplot was the easiest to try, as a plotting tool for developing purposes was needed initially. It was easy to set up and didn't require many lines of code to get going. Gnuplot was implemented to plot different values from the robot state, but the amount of data quickly affected the performance. It was a noticeable delay after only a few seconds because of the increasing amount of data to plot.

It is stated in the requirements that the possibility for live updates of the graph is an important feature. With a long term solution in mind, Gnuplot wouldn't suffice.

OpenGL is not made as a plotting tool, but it is possible to draw lines. It is therefore possible to use OpenGL in this application for both 2D plotting and 3D model rendering. Working with OpenGL requires more investment in time and resources before a result can be expected. However, since OpenGL offers solutions for both options, spending time learning and mastering even another tool is avoided. It fitted better with more of the requirements, even if it lacked some qualities of 2D plotting and graphs (like an interactive area, grid, axis, possibility to read values directly, etc.). The time and resources were therefore put in action and OpenGL was chosen as the preferred graphics library.

2D plot

As stated earlier, a simple 2D plot, or a graph, can be useful for both developers and therapists. A plot consisting of straight lines between points, with new points added continually was created.

In OpenGL, all the vertices must be contained in a vertex buffer object, and a vertex array object contains one or more buffer objects. The vertex array object stores the information for a complete rendered object. The buffer is specified through the function `glBufferData`, and `glVertexAttribPointer`. Listing 3.1 shows an excerpt of the function `Plot2d::initializeBuffer()` where buffers for the plot are generated and defined.

```
1  glGenVertexArrays(1, &dataArray);
2  glGenBuffers(1, &dataBuffer);
3  glBindBuffer(GL_ARRAY_BUFFER, dataBuffer);
4  glBufferData(GL_ARRAY_BUFFER, buffersize*NUM_PLOTS, NULL,
    ↪  GL_DYNAMIC_DRAW);
5  glVertexAttribPointer(
6      0,                          // attribute
7      VERTEX_COORDINATE_COUNT,    // size (3 coordinates)
8      GL_FLOAT,                   // type
9      GL_FALSE,                   // normalized
10     0,                          // stride
11     (void*)0                    // array buffer offset
12 );
13 glEnableVertexAttribArray(0);   // attribute from the buffer
```

Listing 3.1: Generating buffers and defining attributes, excerpt from graphics/plot2d.cpp

In the function `glBufferData` (Listing 3.1, line 4), the buffer is generated with a size `buffersize` defined earlier.

```
buffersize =
 ↪  sizeof(GLfloat)*SAMPLES_PER_FRAME*VERTEX_COORDINATE_COUNT;
```

`databuffer` is a storage for a fixed number of points. C++ properties causes the need for the number of plotting points to be decided when initializing the buffer. The plot is updated with points continually, so a method for adding new points to the buffer and filling it incrementally, was made, as shown in Listing 3.2.

The buffer can only contain a certain amount of data, that is why the `INDEX_OF_LAST_ENTRY` is always increasing but with a modulus of `SAMPLES_PER_FRAME`. When the buffer is full, the next point will overwrite the first point in the buffer.

Once a new point has been added, the graph needs to be redrawn to take effect. The draw

```
void Plot2d::graph_update(Point values[]) {
    GLfloat point[3];
    float current_index_of_buffer = sizeof(GLfloat) *
     ↪  INDEX_OF_LAST_ENTRY * VERTEX_COORDINATE_COUNT;
    float size_of_new_data = sizeof(GLfloat) *
     ↪  VERTEX_COORDINATE_COUNT*SAMPLES_PER_FRAME;
    float value_x = (float) values[0].first;
    float value_y = (float) values[0].second;

    // Calculation of point to fit inside OpenGL window
    point[0] = GRAPH_WIDTH * INDEX_OF_LAST_ENTRY / SAMPLES_PER_FRAME
     ↪  -1.0f;
    point[1] = 2 * (value_y - MIN_EXP_VALUE) / (MAX_EXP_VALUE -
     ↪  MIN_EXP_VALUE) - 1.0f; // [1,-1]
    point[2] = 0.0f;

    glBindVertexArray(dataArray);
    glBindBuffer(GL_ARRAY_BUFFER, dataBuffer);
    glBufferSubData(GL_ARRAY_BUFFER, current_index_of_buffer,
     ↪  size_of_new_data, point);
    INDEX_OF_LAST_ENTRY = (INDEX_OF_LAST_ENTRY + 1)  %
     ↪  (SAMPLES_PER_FRAME);
}
```

Listing 3.2: Updating the graph with a new point

call is shown in Listing 3.3. When the robot communication operates at 1000 Hz, the graph is fed with new points at the same rate. OpenGL renders quickly and can keep up the desired frame rate. OpenGL is made to render complex 3D environments, so making a simple plotting tool with lines is not an issue.

Multiple plots in one graph

Not many alterations must be made to create two or more plots in the same graph. In addition to visually present each value and tendency over time, it can be used to compare two (or more) different parameters.

A way to implement this is to augment the buffer size, with enough space to cover the number of plots

```
glBufferData(GL_ARRAY_BUFFER, buffersize*NUM_PLOTS, NULL,
 ↪  GL_DYNAMIC_DRAW);
```

When adding a new set of values, each point must be placed in the right part of the buffer.

```
for(int i=0; i<NUM_PLOTS; i++) {
    float value_y = (float) values[i].second;

```

```
void Plot2d::drawGraph() {
    // Clear the screen
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Use the shader
    glUseProgram(programID);

    // Send the transformation to the currently bound shader,
    // in the "MVP" uniform
    glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);

    // Bind right render object
    glBindVertexArray(dataArray);

    GLfloat red[4] = {1, 0, 0, 1};
    glUniform4fv(uniform_color, 1, red);
    glDrawArrays(GL_LINE_STRIP, 0, INDEX_OF_LAST_ENTRY);

}
```

Listing 3.3: Function drawGraph() in graphics/Plot2d.cpp

```
    point[0] = GRAPH_WIDTH * INDEX_OF_LAST_ENTRY / SAMPLES_PER_FRAME
     ↪  -1.0f;
    point[1] = 2 * (value_y - MIN_EXP_VALUE) / (MAX_EXP_VALUE -
     ↪  MIN_EXP_VALUE) - 1.0f; // [1,-1]
    point[2] = 0.0f;

    glBufferSubData(GL_ARRAY_BUFFER, current_index_of_buffer +
     ↪  buffersize*i,    size_of_new_data, point);
}
```

Finally, when drawing the plots, each plot has its own draw call.

```
for (int i=0; i<NUM_PLOTS; i++) {
    glUniform4fv(uniform_color, 1, color[i]);
    glDrawArrays(GL_LINE_STRIP, SAMPLES_PER_FRAME*i,
     ↪  INDEX_OF_LAST_ENTRY);
}
```

Alternative presentation of the graph

To create something informative, a different approach to presenting the graph was developed. Instead of filling the screen from left to right, the current value was placed in the middle of the frame and the previous values to the left. It makes the current value the most interesting at all times since it is centered on the screen.

3D model

In the same way as OpenGL was used to plot 2D graphs, it is possible to plot with 3 coordinates. OpenGL is a popular choice for rendering 3D models, in e.g games, so import functions of 3D models are easily accessible. The most common choice is to upload the model as an obj file.

We used a single header obj-loader (Smith, 2018). It is simple, easy to incorporate with existing code and its MIT license permits free use, modification and publication of code.

After loading the obj-file with the external OBJ-function, all vertices, normals and texture coordinates were added in a coherent vertex buffer object.

```
void Plot3d::drawModel() {
    // Clear the screen
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    ModelMatrix = getModelMatrix();
    ProjectionMatrix = getProjectionMatrix();
    ViewMatrix = getViewMatrix();
    MVP = ProjectionMatrix * ViewMatrix * ModelMatrix;

    glUseProgram(programID);

    // Send our transformation to the currently bound shader,
    // in the "MVP" uniform
    glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
    glUniformMatrix4fv(ModelMatrixID, 1, GL_FALSE,
     ↪  &ModelMatrix[0][0]);
    glUniformMatrix4fv(ViewMatrixID, 1, GL_FALSE, &ViewMatrix[0][0]);

    lightPos = glm::vec3(4, 4, 4);
    glUniform3f(LightID, lightPos.x, lightPos.y, lightPos.z);

    glBindVertexArray(VertexArrayID);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, Texture);
    glUniform1i(TextureID, 0);

    glEnableVertexAttribArray(0);
    glEnableVertexAttribArray(1);
    glEnableVertexAttribArray(2);

    // Draw triangles from model
    glDrawArrays(GL_TRIANGLES, 0, num_model_vertices);
    }
```

Listing 3.4: Funtion drawModel() in graphics/Plot3d.cpp

Transformation of model

One of the GUI requirements is projecting a 3D model to follow the real movement and replay a recorded movement. The job of transforming 3D points into 2D coordinates on a screen is accomplished through matrix transformation. The transformations consist of a model matrix, a view matrix and projection matrix put together in an MVP matrix (Shreiner et al., 2013). The MVP matrix can be used to manipulate the vertex positions without changing the model coordinates.

Model matrix: constructed of a rotation matrix, a translation matrix, and a scaling matrix. It can be used to rotate, translate and scale the model based on its vertices.

View matrix: a matrix to describe where the ”camera” is looking at the screen by stating a position of the camera, a position to where the camera looks and finally a vector to define if it is looking up or down.

Projection matrix: describes how the model should be projected on the screen. It is possible to adjust the field of view, the screen ratio, and the display range.

It is possible to change the view of the robot and translate, scale or rotate the model as shown in Listing 3.5. This was tested with an obj-file of the robot and keyboard inputs to control different parameters.

```
    // Projection matrix : 45° Field of View, 4:3 ratio, display range : 0.1 unit <-> 100 units
    ProjectionMatrix = glm::perspective(glm::radians(FoV), 4.0f / 3.0f, 0.1f, 100.0f);
    // Camera matrix
    ViewMatrix = glm::lookAt(
                    position,           // Camera is here
                    glm::vec3(0,y,0),   // and looks here : at the same position, plus "direction"
                    up                  // Head is up (set to 0,-1,0 to look upside-down)
                           );

    RotationMatrix = glm::eulerAngleXYZ(mod_angle_x, mod_angle_y, mod_angle_z);
    TranslationMatrix = glm::translate(glm::mat4(1.0), glm::vec3(mod_x, mod_y, mod_z));
    ScalingMatrix = glm::mat4(1.0);

    ModelingMatrix = TranslationMatrix * RotationMatrix * ScalingMatrix;
```

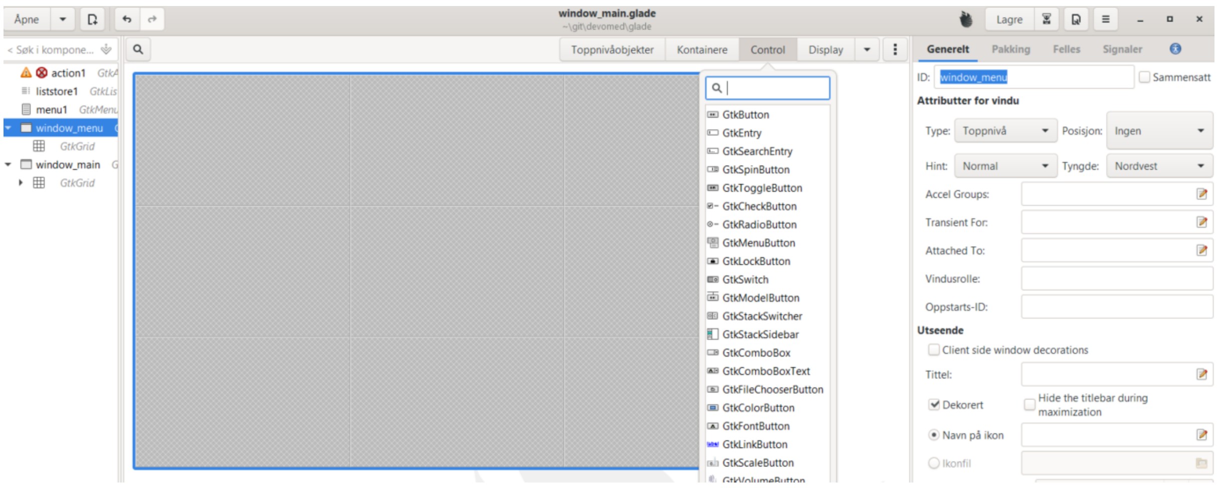
Listing 3.5: Definition of the different matrices

3.3.3 GUI tool

When some of the functionality was developed, looking for a suitable GUI design tool was started. A C++ solution was preferred since the rest of the code base is written in C++. That the solution was compatible with OpenGL was also essential.

After looking into the different possibilities, an option fitting the requirements stood out. GTK is free software and is commonly used and recommended. It has a Lesser General Public License (LGPL) which allows it to be used for developing proprietary software without any license fees or royalties (The GTK Team, 2019b). Some other free options operate under stricter license for device creation. Considering current and future work, a software with a liberal license like LGPL is favorable.

Gtkmm is the C++ interface for GTK, making it possible to create user interfaces either in C++ code or with the Glade User Interface designer, using `Gtk::Builder`. A user interface designer makes it easier and quicker to get started with creating a user interface without any prior knowledge to the tools or coding. Figure 3.7 shows a screenshot of the user interface designer with one of the menus for choosing different widgets is displayed.



ID: window_menu
Sammensatt
Attributter for vindu
Type: Toppnivå
Posisjon: Ingen
Hint: Normal
Tyngde: Nordvest
Accel Groups:
Transient For:
Attached To:
Vindusrolle:
Oppstarts-ID:
Utseende
Client side window decorations
Tittel:
Dekorert
Hide the titlebar during maximization
Navn på ikon
Ikonfil

Figure 3.7: Screenshot of Glade user interface designer

Developing a GUI

The requirement list for a GUI is described in Section 3.3.1. It contains many requirements, so the most important ones were addressed first. It was decided to mainly concentrate on the ones regarding the control of the robot and support for other features developed in this project.

The requirements are quite different, which makes it natural to divide the functionality into separate pages. To achieve this, a notebook widget with tabs was chosen, making it easy to change between different pages and still keep an overview of what the user interface involves.

With the notebook and the different tabs, it was also possible to create pages suitable for development while keeping them separated from the pages contributing to an MVP.

Record a movement

With the functionality to record a movement in place, it was developed a GUI exercise page to handle the following requirement:

- Possibility to make an exercise by guiding the robot manually, record movement and save as an exercise option

It was decided to have a toggle button handle the recording of a movement. The toggle button has two states, it can be activated (start tracking) or deactivated (stop tracking). To not waste resources and memory with automatic saving, a separate button to save the movement was made.

```
    Gtk::Button     *btn_save_exercise;
    Gtk::ToggleButton     *btn_record;

    builder->get_widget("btn_save_exercise", btn_save_exercise);
    builder->get_widget("btn_record", btn_record);

    btn_record->signal_clicked().connect([&]() {
      if (btn_record->get_active()) {
        robot_data->tracking_data.clear();
        robot_data->track_position = true;
        btn_record->set_label("Stopp tracking");
      } else {
        robot_data->track_position = false;
        btn_record->set_label("Start tracking");
      }
    });
```

Listing 3.6: Record a movement, excerpt from graphics/gui.cpp

Listing 3.6 shows the implementation of this functionality. When the Boolean `robot_data->tracking_data` is set to true, it activates a sequence in the robot control thread that starts pushing the current positions into an array. When `robot_data->track_position` is set to false again, the array of recorded positions are saved in a CSV file. The CSV files are stored in the separated folder *exercises* under the build folder.

Adjustable resistance

With an exercise recorded, it was useful to make it possible to perform an exercise. A drop-down list for saved exercises and a start button were made. During the performance of an exercise, one requirement was:

- Adjustable resistance from the robot

The resistance was adjusted with a `GtkSpinButton`. It permits the user to adjust the value with up and down buttons or type the wanted resistance directly in the value field, as shown in Figure 3.8. The weight incrementing step was set to 50 g, a barely noticeable change, making progress achievable. The code to set up the resistance adjustment is shown in Listing 3.7

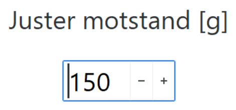


Figure 3.8: Spin button and label

```cpp
Gtk::SpinButton *spin_resistance;

builder->get_widget("spin_resistance", spin_resistance);

spin_resistance->signal_value_changed().connect([&](){
    wanted_force = spin_resistance->get_value_as_int();
});
```

Listing 3.7: Adjust resistance with spin resistance, excerpt from graphics/gui.cpp

Measure force applied by the patient

Other requirements during the performance of an exercise were the following:

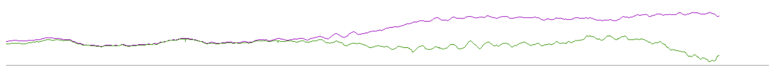
- Possibility to measure the force applied by the patient
- Possibility to illustrate and measure deviation from ideal movement

A 2D plot and a written value (Figure 3.9b) was created to represent the measured force applied by the patient. The 2D plot was made with OpenGL. GTK has a widget called `gl_area` that can render OpenGL content. The `gl_area` needs a handle to realize, unrealize and render the OpenGL content, the code is shown in Listing 3.8. During realization, the OpenGL content is created and the buffers are generated. During unrealization (when the program quits) the buffers are deleted and the array attributes are disabled. During render the plot is updated and the content is drawn.

```cpp
Gtk::GLArea     *gl_area;
builder->get_widget("gl_area", gl_area);

gl_area->signal_realize().connect(sigc::mem_fun(*this,
 ↪  &Gui::onRealize));
gl_area->signal_unrealize().connect(sigc::mem_fun(*this,
 ↪  &Gui::onUnrealize));
gl_area->signal_render().connect(sigc::mem_fun(*this, &Gui::onRender),
 ↪  false);

void Gui::onRealize(){
    gl_area->make_current();
    plot.realize();
}

void Gui::onUnrealize(){
    gl_area->make_current();
    plot.unrealize();
}

bool Gui::onRender(const Glib::RefPtr<Gdk::GLContext>& /* context */)
 ↪  {
    gl_area->make_current();
    update_plot();
    plot.gl_draw();
    return true;
}
```

Listing 3.8: Creat OpenGL environment in GUI, excerpt from gui.cpp

The deviation was illustrated by adding an extra plot in the graph. It shows the component of the force applied in the right direction. If all the force applied by the patient is making the robot move in the right direction, the plots will overlap, if the force is applied in another direction, it will be a gap in the graph, as shown in Figure 3.9a. The pink plot is the force applied, the green plot is the component of the force in the exercise direction.



(a) 2D graph of the force applied

Påført kraft [g]

760

(b) Instant measurement of force applied

Figure 3.9: Representation of measured force applied to the end effector

Lambda functions

The actions of a GTK widget (buttons, gl_areas etc.) can be connected to a signal handler (functions or expressions). Gtkmm documentation (The GNOME Project, 2014) suggests to use `sigc::mem_fun()` and `sigc::ptr_fun()` to connect the signal handlers. However, since C++ 11 it is possible to use lambda functions instead (cppreference.com, 2019). It compacts the code and it is easier to send parameters to a function. Parts of Listing 3.8 is rewritten using lambda functions and shown in Listing 3.9 to state the difference.

```
// Recommended method in gtkmm documentation
gl_area->signal_realize().connect(sigc::mem_fun(*this,
 ↪  &Gui::onRealize));

void Gui::onRealize(){
    gl_area->make_current();
    plot.realize();
}

// Use of lambda functions
gl_area->signal_realize().connect([&](){
    gl_area->make_current();
    plot.realize();
});
```

Listing 3.9: Lambda expressions vs sigc::mem_fun()

Lambda functions are efficient to use, especially if the connected function consists of few lines, and is only called one or a few times. When it was discovered that Lambda functions made it easier to connect signal handlers, it was the preferred choice.

GUI for demonstration

To prepare for a realistic demonstration of the GUI, with a wholesome experience, some of the requirements were added as pages with a graphical profile and limited functionality. The styling of the user interface was done in CSS.

3.4 Software Architecture

When writing a program from scratch, it is important to think beyond the functionality presently needed. The software architecture needs to be constructed in a way that permits alterations and scalability for future requirements. This section elaborates on the tools used and decisions made to build robust software easy to continue working with.

3.4.1 Git

Git is a highly popular version-control system for software development. The project was incorporated with Git quite early. It increased the robustness of the program by tracking changes and making it possible to work on the same source code from different devices. It is recommended to continue using Git for future continuation with the project.

3.4.2 Code editor and build tool

The code editor Visual Studio Code (Microsoft, 2019) was chosen for this project due to the familiarity, git support, and great overview. However, Visual Studio Code doesn't include a build automation tool and thus CMake was chosen as the tool for managing the build process. CMake has multi-platform support, good documentation and it is widely distributed. In addition, it is the tool used in libfranka, which makes it easier to integrate libfranka in the project.

CMake

The developer writes a text-file (Cmakelists.txt), and based on that CMake will automatically create the other files needed to compile the program. In CMake, the external packages are added (e.g libfranka, Eigen3 and OpenGL), all c++ files are linked with their executable. It is also used to copy files with other file extensions than runnable programs (e.g shaders, obj-files) from source folder to build directory.

During development, different set-ups of CMake was tested. One possibility is to write several C++ files as executables, another possibility is to have one main application file and rather call all functionality from different files there (one EXAMPLE and several SOURCES as found in CMakeLists.txt in Appendix A). The latter option compiles much faster in CMake since executables take longer time to build than other files. It is possible to comment out non-relevant executables as well, but it is rather exhausting to edit or check CMakeLists.txt before each compile.

3.4.3 Threading

The communication between the computer and the robot operates at 1000 Hz. A slow program leads to communication errors due to the robot experiencing packet losses.

One way of overcoming the communication errors due to heavy computing is threading. Every core operation got its own thread, one thread for communication with the robot and one thread to run the GUI-application, as shown in Listing 3.10. Threading makes it possible for other processes to operate at a lower frequency, which can avoid unnecessary use of computer power. Threading is also advantageous for testing and debugging, making it easier to isolate portions of the code.

It was created a common struct, `robot_data`, that can be reached and updated from any thread. The struct is passed to each thread as an argument, as shown in Listing 3.10 lines 5 and 8. The common struct makes it possible for the GUI to get live updated values from the robot communication thread without any direct connection with that thread.

```
1   pthread_t threads[NUM_THREADS];
2
3   for( i = 0; i < NUM_THREADS; i++ ) {
4       if (i == 0) {
5           rc = pthread_create(&threads[i], NULL, GuiThread ,
             ↪  &robot_data);
6       }
7       else if (i == 1) {
8           rc = pthread_create(&threads[i], NULL, RobotLoopThread ,
             ↪  &robot_data);
9
10      ...
```

Listing 3.10: Excerpt from main_start_threads.cpp in src/apps

3.5 Helmet mounting mechanism and helmet

In the thesis of Brattgjerd (2017), a mechanism for mounting the helmet on the robot was developed. The project thesis of Baardsgaard and Hafskolt (2018) discovered severe flaws in this design, mainly regarding looseness and weight. Therefore it was considered crucial to further develop this mechanism to approach an MVP. It was also natural to develop the helmet further simultaneously with the mounting mechanism.

3.5.1 Helmet mounting mechanism

Brattgjerd (2017) made a list of requirements for the helmet mounting mechanism. This list was reconsidered and now contains the following requirements:

- Weight: As the robot has a limited payload, the mechanism needs to be lightweight.
- Easy and fast to connect/disconnect: This was based on the assumption that the patient would put on the helmet before connecting to the robot. With a compliant robot, it may be just as easy to have the helmet mounted on the robot while putting it on the patient's head. This point is therefore a nice-to-have rather than a must-have.
- Low profile: As Baardsgaard and Hafskolt (2018) discovered, a large distance between the flange and the head limits both the possible workspace and the torque of the robot. Therefore a short distance between the flange and the head could be important for this particular robot.
- Safety: With the patient's head connected to a powerful robot, an important safety feature is a self-actuating mechanism for disconnecting the head from the robot if the robot suddenly moves uncontrollably.
- Looseness: To provide accuracy in both motion and force transfer between the robot and the head, minimal play in the mechanism during normal operation is crucial.
- Adjustable: The ability to adjust the position of the connection is a nice-to-have while trying to address the issue of a limited work-space.

From these functions, the safety issue seemed most important to address. To ensure that the mechanism never fails, any solutions containing electronics or complex mechanics were regarded unfavorable. An idea was therefore to use magnets to provide a connection that releases on excessive force/torque.

The mechanism was developed using rapid prototyping and testing, as magnetic properties are complex in theory but cheap and easy to test practically.

The first prototype was made of two parts, one mounted to the robot and one mounted to the helmet. Four 10 mm $\times$ 2 mm neodymium disc magnets placed symmetrically provided the connection force. The magnets seemed to have better pulling force than shear force. Therefore, two small bumps were added to make the connection more rigid in the shear direction, as shown in Figure 3.10.

The prototype was 3D-printed with PLA plastic and tested. After attempting to objectively test the mechanism using a luggage weight to measure the force required to release the connection, it was quickly observed that it was hard to get accurate results, and it would take too much time to perform. It was rather decided to test the connection with our own heads and feel if the connection is too loose or too hard to disconnect.The following prototypes were compared against each other, to find where they could improve.

The first prototype was too weak in shear, pulling and torsional strength, so either stronger magnets or more magnets were needed. It was also observed that shear strength seemed to be less of an issue than first thought, and the main weakness was rather tilting. To address this, the

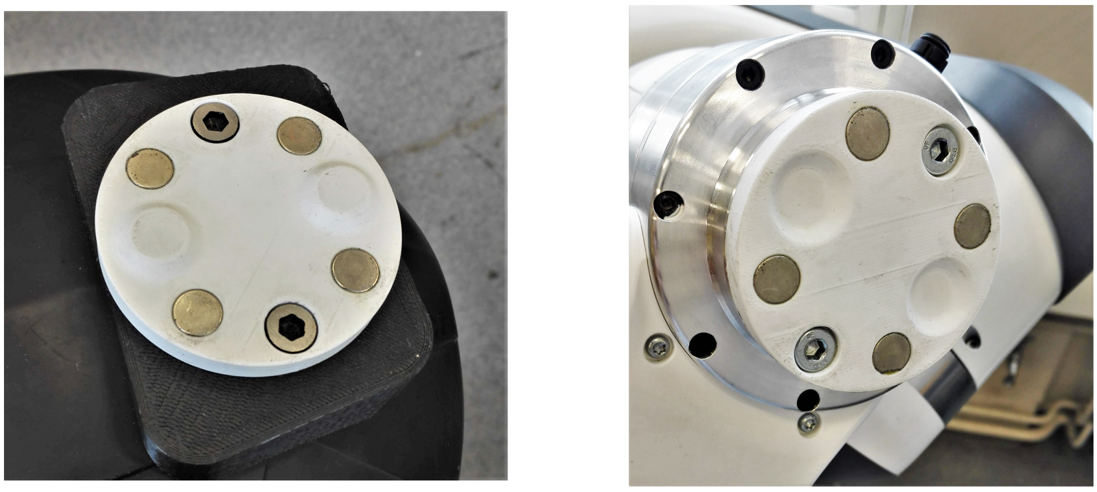

(a) Part on the helmet (b) Part on the robot

Figure 3.10: Magnetic connection prototype 1

diameter of the connection was increased. This increases the lever length and thereby increases the force needed to tilt the connection. The distance from the center to the magnets does not affect tilting strength in theory, as the total force vector from the magnets will always be positioned in the center of the circle. But the radius does affect the torsional strength along the connection axis. Therefore this distance was also increased since the torsional strength seemed to be too low.

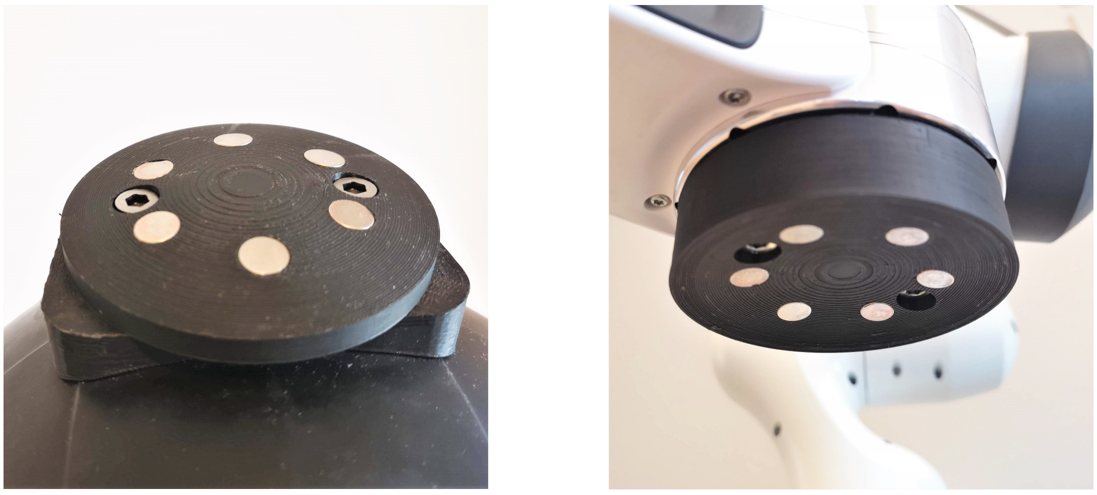
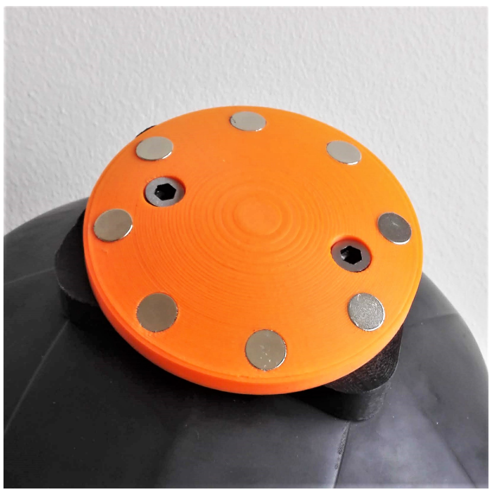

(a) Part on the helmet (b) Part on the robot

Figure 3.11: Magnetic connection prototype 2

To keep some rigidity in the shear direction, different profiles for the connection surface were brainstormed. As the helmet was developed simultaneously with an almost spherical shell, the idea of a concave surface was most interesting. The magnets could be placed directly on the helmet, where the part on the flange had a corresponding concave surface. With multiple rings of magnets on the helmet, the connection can easily be moved to different positions. Figure 3.11 shows the second prototype.

The second prototype did perform better in the areas that were attempted improved but was still quite weak in all directions except pulling along its axis. Therefore, the number of magnets were increased, the diameter increased further and the radius of the magnet ring was increased to the maximum. The third prototype is shown in Figure 3.12.


(a) Part on the helmet

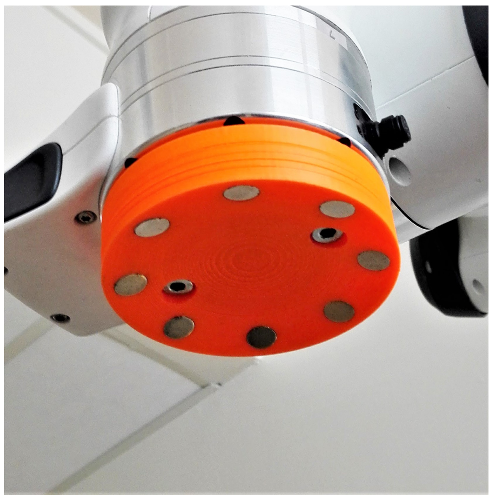
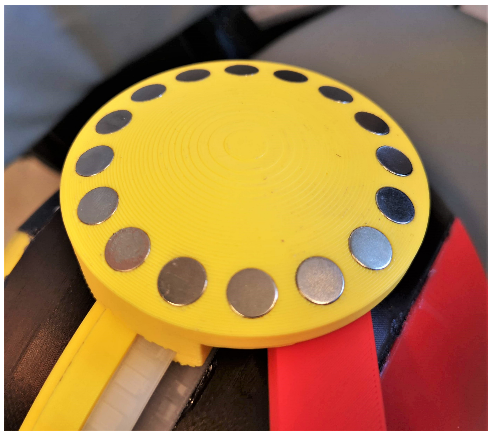

(b) Part on the robot

Figure 3.12: Magnetic connection prototype 3

The third prototype was better as expected but still lacked the general strength in most directions as it was too easy to release the connection. To increase the force, it was considered getting stronger/larger magnets. Stronger and larger magnets are more expensive and need to be ordered online, so an easy solution was to double the number of magnets. It was realized that the position of the connection may need to be fine-tuned. One idea was to have multiple rings of magnets over the helmet, but that will need a lot of magnets and may be hard to fine tune. Instead, a slider and locking mechanism on the helmet was tested out. The fourth prototype is shown in Figure 3.13.

The magnets used were labeled with a strength of 0.9kg each. With 16 magnets, this means the pulling force required to release the connection is 14.4 kg. Even though this seems way too high, it appears to be sufficient for keeping a rigid connection while still releasing with excessive force. This is because the most common releasing motion is tilting when not all the magnets are pulled apart at the same time.

A cross section of the magnetic connection and helmet with the slider for adjusting position is shown in Figure 3.14. The slider is designed to be as slim as possible to not increase the flange-to-head distance too much. The handle is designed as a cam that can tighten the slider to


(a) Part on the helmet

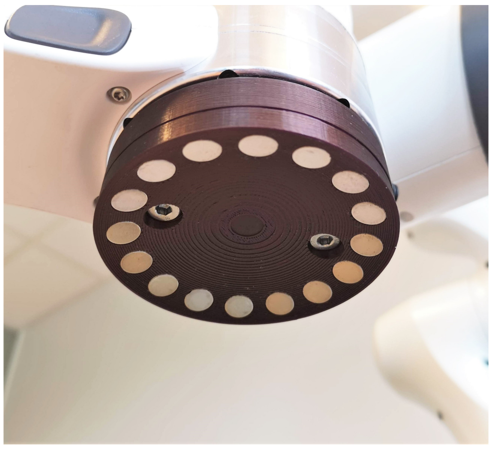

(b) Part on the robot

Figure 3.13: Magnetic connection prototype 4

fasten it in the desired position.

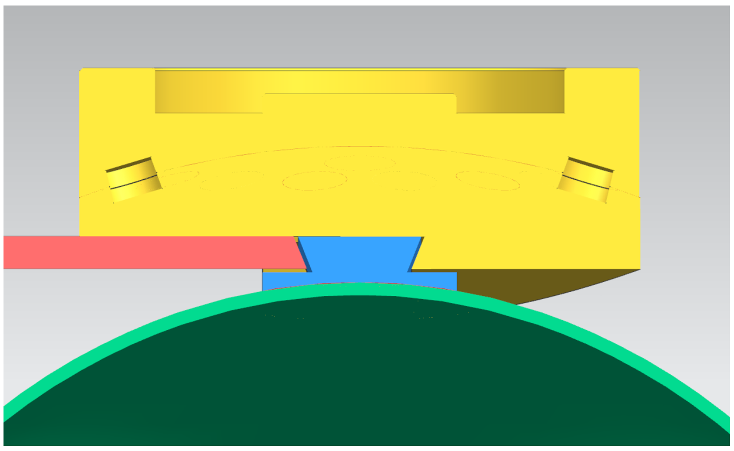


Figure 3.14: Cross section of the helmet with magnetic connection and slider

The slider for adjusting the position did not work as expected. The locking handle can unlock itself when the mechanism is pushed backwards since it rotates in the same direction as the sliding motion. This also adds 19 mm to the flange-to-head distance and reduces the sturdiness of the connection. An easy solution is to just put magnets on the helmet shell, but alternative solutions should be explored further.

Without the slider, the connection could be very low profile, with a flange to helmet distance of 5 mm. The part on the robot only weighs 66 g, and the 16 magnets on the helmet weigh 22 g in total.

3.5.2 Helmet

A helmet concept was developed during the thesis of Brattgjerd and Festøy (2018). This consisted of an Etto Twister alpine helmet with inflatable air bags inside to securely fasten the head. As the helmet mounting mechanism was developed, it was noticed that the flange-to-head distance could be reduced significantly by designing a new helmet. An attempt was also made at making a universal helmet that fits all head shapes and sizes, as this seemed desirable.

The Etto Twister is made of an outer ABS plastic shell, with a polystyrene foam padding for impact cushioning. The only part of the helmet needed in this concept is the outer shell. Gælok and Strand (2017) mentioned a possible need for multiple helmets in different shapes and sizes. That would increase cost, time spent with the patient, and storage space needed at the clinic. A universal helmet may need more development but is beneficial for the clinic.

It was designed a new shell using Siemens NX CAD software, which was dimensioned to fit all human heads. The dimensions were based on figures in the thesis by Gælok and Strand (2017). To quickly test the helmet, the design was 3D printed in several parts which were glued together. The 3D printed shell works good with the inflatable air bags, but since the distance between the head and the shell has increased slightly, the air inside the bags is easier to compress. The air is also able to move around inside the bags, which causes noticeable looseness.

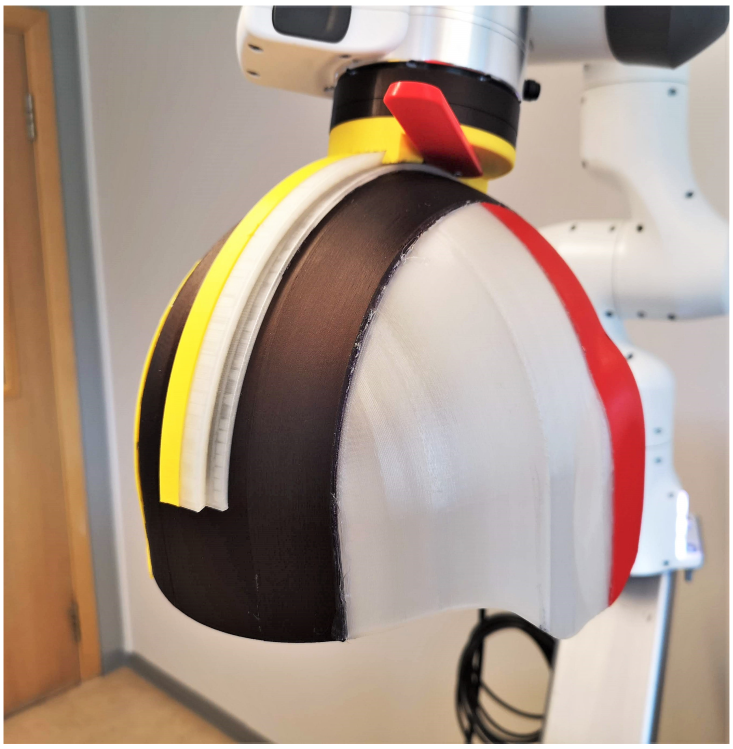


Figure 3.15: The 3D printed helmet with slider and magnetic connection mounted to the robot

The prototype of the helmet and slider with the final magnetic connection is shown in Figure

3.15. With a new helmet, and a new helmet mounting connection, the flange-to-head distance could be as low as 10 mm. The previous solution had a distance of 64 mm. An estimated average distance from the neck to the top of the head is 230 mm (Neuhaus, 2014). The lever then changes from 294 mm to 240 mm, which reduces the torque needed from the neck by 18 %.

Chapter 4

Current Concept

The work committed throughout this project resulted in the current concept of the product. The previous solution was developed and improved with a focus on the areas mentioned in Section 1.3. An overview of the current concept is presented in this chapter. Each component is presented and described in depth.

4.1 Overview

The apparatus consists of a robot arm mounted on a frame structure attached to a chair. At the flange of the robot a magnetic connection is attached, to connect the helmet to the robot. The helmet is made to fit all head sizes and tightened with air-pockets.

A system for controlling the robot has been made. The robot can be moved by hand with low resistance, due to a force controller based on a dynamic model. Additional desired resistance can also be added. A path can also be recorded, and the robot can be constrained to only move through that path. Also, the joints individually provide resistance to not move outside their limits.

The robot can easily be operated through a custom GUI. The GUI is made as a foundation for a complete system, where a user can choose between patient records, and view, edit and save exercises and programs for each patient. The exercises can be recorded, saved and performed from the GUI, where resistance can be added. An OpenGL framework is also made to visualize the robot performing a saved exercise, and as a plotting tool for visualizing force, speed or other parameters.

Figure 4.1 gives an overview of the product.

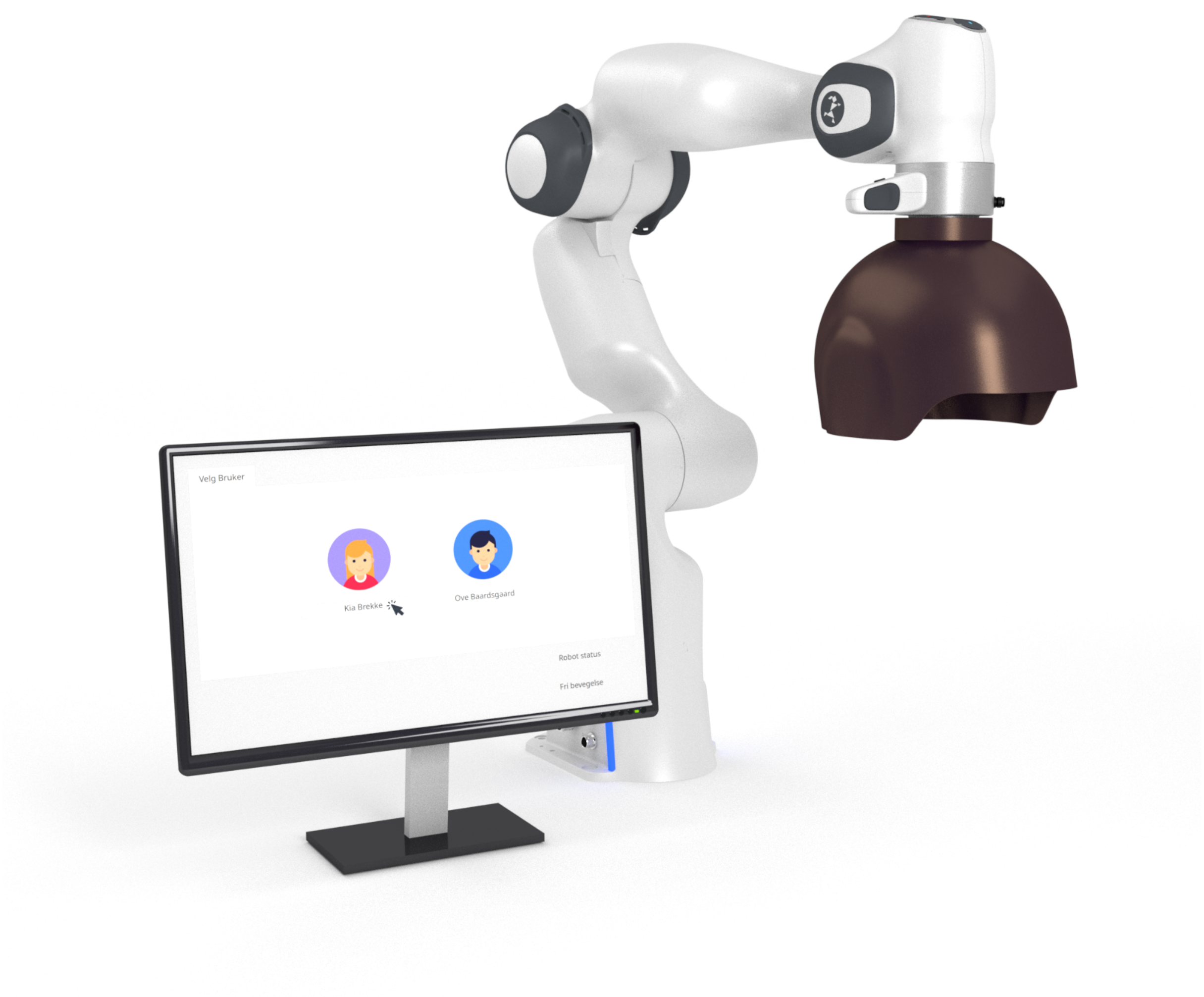


Figure 4.1: Current solution with the Panda robot arm, helmet and magnetic connection, and the GUI start page.

As defined in the problem description, the chair and frame structure was not further developed during this thesis. The solution and design decisions made are thoroughly described in chapter 5.3 and 5.4 in the thesis by Brattgjerd and Festøy (2018). Figure 4.2 shows the current concept with chair and frame structure.

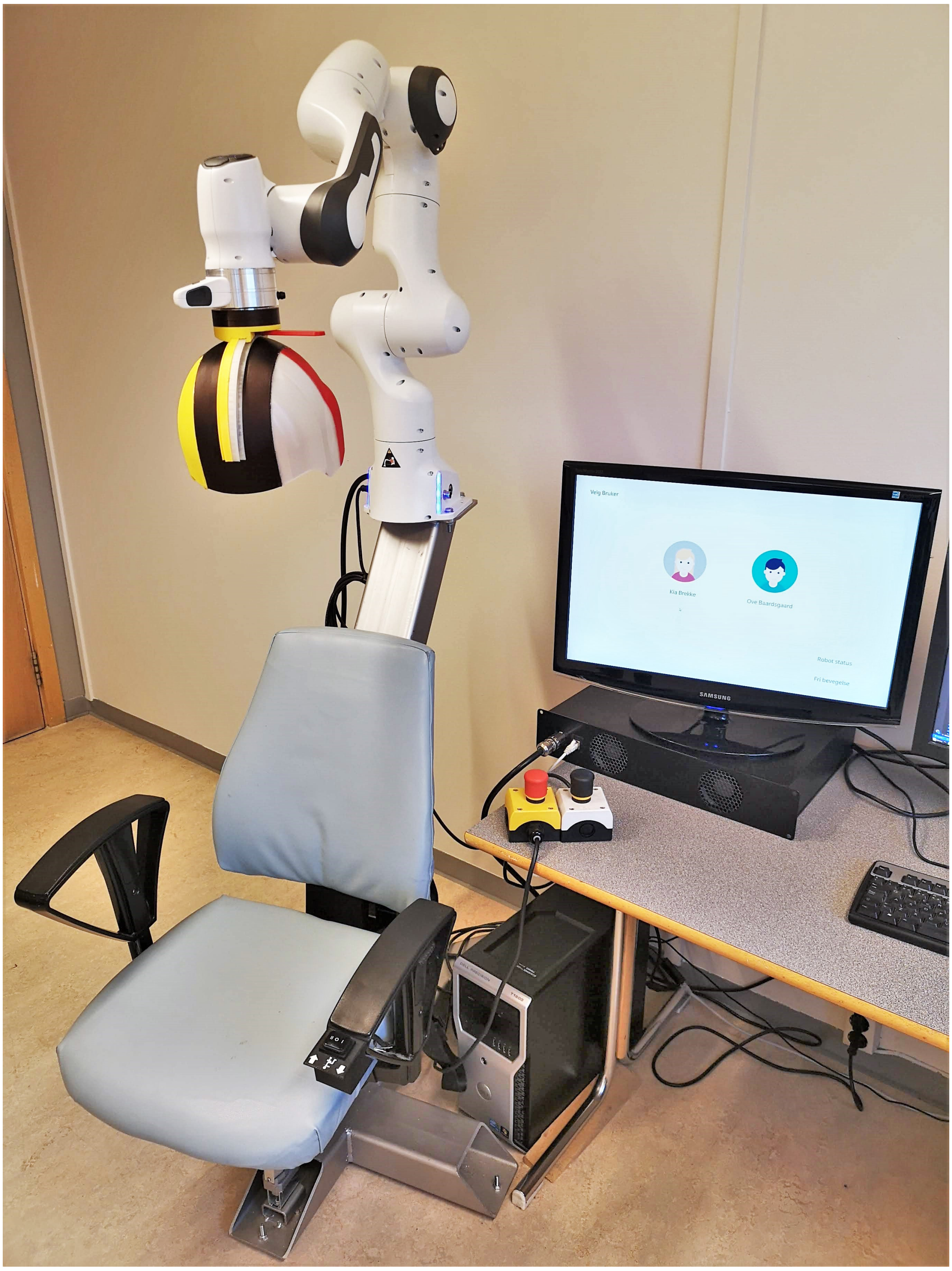


Figure 4.2: Current solution with the chair from the Brattgjerd and Festøy (2018) thesis, the Panda robot arm, helmet and magnetic connection, and the GUI start page.

4.2 Compliant robot control system

The task was to make a compliant robot control system where you can move the end-effector with very low resistance. The current concept is a control system that can be divided into three parts:

- Getting information from the internal sensors of the robot and processing the data.
- Using the data in a force controller which converts the force input into a desired cartesian velocity.
- Converting the cartesian velocity into joint velocities, and controlling the robot arm to perform these velocities.

In addition, our solution involves a method for recording a motion path that is saved to a file, constraining the robot motion to the recorded path, and add resistance to the motion.

4.2.1 Velocity controller

The best solution was to use the `franka::jointVelocities` interface. As the exercise motions are performed in task space, it is needed to convert between joint space and task space velocities.

The robot is commanded with joint velocities, $\vec{\dot{q}}$. The inverse Jacobian is used with Equation 3.7 to transform the cartesian velocities into joint velocities. As the Jacobian is not directly invertible, the pseudo-inverse is used instead as shown in Listing 4.1.

```
const Eigen::MatrixXd pseudo_inverse =
 ↪  jacobian.completeOrthogonalDecomposition().pseudoInverse();
Eigen::VectorXd qd_desired = pseudo_inverse*desired_velocity;
```

Listing 4.1: An excerpt from the robot loop showing how the Jacobian is inverted to transform task space velocities to joint space velocities.

4.2.2 Getting sensor information

Using libfranka, getting sensor data is often very straight forward. A lot of values are easy to obtain from the `robot_state`, which are documented on libfranka docs (Franka Emika GmbH, 2017b). Using the Eigen library has been very useful to easily do matrix and vector calculations. The values from `robot_state` will then need to be mapped into Eigen vectors/matrices.

As the `franka::JointVelocities` interface was used, the cartesian velocities of the robot were not obtainable directly from `robot_state`. Instead, joint velocities and the Jacobian is used to calculate cartesian velocity using Equation 2.7 as shown in Listing 4.2.
The external force applied to the flange of the robot is easily obtained using `robot_state.K_F_ext_hat_K`. The force signal is corrected for gravity, inertia, Coriolis and centrifugal

```cpp
std::array<double, 42> jacobian_array =
  ↪ model.zeroJacobian(franka::Frame::kEndEffector, robot_state);
Eigen::Map<const Eigen::Matrix<double, 6, 7> >
  ↪ jacobian(jacobian_array.data());
Eigen::Map<const Eigen::Matrix<double, 7, 1> >
  ↪ dq_d(robot_state.dq_d.data());

const Eigen::VectorXd robot_velocity = jacobian*dq_d;
const Eigen::Vector3d robot_cart_vel = robot_velocity.head(3);
const Eigen::Vector3d robot_ang_vel = robot_velocity.tail(3);
```

Listing 4.2: Getting the jacobian and joint velocities to calculate cartesian velocity

effects (Cristian Kodura, Franka Emika, 2019). To reduce noise from the signal, a function from libfranka called `franka::LowpassFilter` was used. The force signal was filtered with a cutoff at 10Hz. This produces a smooth signal, without too much delay.

4.2.3 Force controller

To make the force controller, a dynamic model of an object with the desired behavior was made. The model consists of a mass on a surface with friction. The dynamic properties of the model are shown in Equation 4.1, where m is the virtual mass of the object, $\vec{a}$ is the acceleration of the object, $\vec{F_{ext}}$ is the externally applied force, and $\vec{F_f}$ is the friction force.

$\vec{F_f}$ is divided into kinetic friction $\vec{F_{fk}}$, and static friction $\vec{F_{fs}}$. Static friction is applied when the velocity of the object is zero, and $\vec{F_{ext}}$ is less than $\vec{F_{fk}}$.

$$m\vec{a} = \vec{F_{ext}} + \vec{F_f} \tag{4.1}$$

If the wanted resistance force is added and the equation is reorganized, it results in an equation for the desired acceleration of the robot shown in Equation 4.2.

$$\vec{a} = \frac{1}{m}(\vec{F_{ext}} + \vec{F_f} + \vec{F_w}) \tag{4.2}$$

To calculate the desired velocity, the kinematic Equation 3.6 is used. The dynamic model was implemented in the robot loop as shown in Listing 4.3.

The function used on lines 7 and 8 in Listing 4.3 is a custom function for calculating the friction as mentioned in Section 3.2.4, shown in Listing 4.4.

```cpp
//F_fk = mu*m*g
double friction = 0.01*virtual_mass*9.81;
//Wanted force in grams to act as an extra friction
double wanted_force = (robot_data->wanted_force/1000)*9.81;

//Convert friction into static or kinetic friction based on velocity
const Eigen::Vector3d cart_friction_force =
 ↪  frictionForce(robot_cart_vel, external_force, friction);
const Eigen::Vector3d cart_wanted_force =
 ↪  frictionForce(robot_cart_vel, external_force, wanted_force);

//F_tot = F_ext + F_f + F_w
const Eigen::Vector3d total_force = external_force +
 ↪  cart_friction_force + cart_wanted_force;
//a = 1/m * F_tot
const Eigen::Vector3d acc = (1.0/virtual_mass)*total_force;
//v = v_0 + a * dt
const Eigen::Vector3d cart_vel_desired = robot_cart_vel + acc * dt;
```

Listing 4.3: Force controller

```cpp
//Return a simple friction force with static friction up to a desired
 ↪  "max_friction_force", and then kinetic friction.
Vector3d frictionForce(Vector3d velocity, Vector3d external_force,
 ↪  double max_friction_force) {
    //Kinetic friction if we have velocity, or force exceeds the
     ↪  desired force value
    if (velocity.norm() > 0.001 || external_force.norm() >
     ↪  max_friction_force) {
        //Apply friction in the opposite direction of velocity
        return -velocity.normalized()*max_friction_force;
    } else {
        //Static friction: sum F = 0, and there is no movement.
        return -external_force;
    }
}
```

Listing 4.4: Function frictionForce() in functions/functions.cpp

4.2.4 Constraining motion

To constrain the motion of the robot to a recorded path, the path is first read from a file and put into a matrix with the function `readMatrix`, shown in Listing 4.5. The matrix is then a $3 \times n$ matrix with n rows containing X, Y, and Z values for each point.

```cpp
MatrixXd readMatrix(std::string filename) {
    std::ifstream indata;
    try {
        indata.open(filename.c_str());
    } catch (std::system_error& e) {
        std::cerr << e.what() << std::endl;
    }
    std::string line;
    std::vector<double> values;
    uint rows = 0;
    while (std::getline(indata, line)) {
        std::stringstream lineStream(line);
        std::string cell;
        while (std::getline(lineStream, cell, ';')) {
            std::stringstream cellstream(cell);
            double value;
            cellstream >> value;
            values.push_back(value);
        }
        ++rows;
    }
    return Map<const MatrixXd>(values.data(), values.size()/rows,
     ↪  rows);
};
```

Listing 4.5: Function readMatrix in functions/functions.cpp

When starting to constrain the motion, the path matrix is initialized. This means that the path is translated such that the start point of the path is at the same position as the robot's current position. This is done by subtracting the first row of the matrix over the whole matrix, which translates the path to origo. Then, the robot's current position is added to all the rows in the matrix, as shown in Listing 4.6.

```
if (!trackingpath_is_initialized) {
    Eigen::Vector3d first_column = robot_data->track_path.col(0);
    robot_data->track_path = robot_data->track_path.colwise() -
     ↪  first_column;
    robot_data->track_path = robot_data->track_path.colwise() +
     ↪  robot_cart_pos;
    trackingpath_is_initialized = true;
}
```

Listing 4.6: Excerpt from the robot loop in robot_loop_thread.cpp, where the tracking path is initialized.

The function `closestPointOnLineSegment` then finds the point on the path which is closest to the desired position. The desired position is calculated from the desired velocity from the force controller. `closestPointOnLineSegment` loops through all the points on the path, finds the closest point on all line segments made from those points, and then returns the point which is closest of all. The code for this function is shown in Listing 4.7. To find the closest point on each line segment, the function runs the function `closestPointOnLine`, which is based on code from OpenGL Mathematics (2018)

After finding the closest point on the path, the difference between the closest point and the current position is sent as an error input into a PID controller. The PID output is then added to the old desired position as a correction, which causes the new desired position to follow the path. With this method, the robot can be moved by hand through a pre-recorded path smoothly and securely.

```
Vector3d closestPointOnLine(const Vector3d linePoint1, const Vector3d
 ↪ linePoint2, const Vector3d point) {
  Vector3d lineDirection = (linePoint2 -  linePoint1).normalized();
  double projectedDistance = (point - linePoint1).dot(lineDirection);
  if (projectedDistance <= 0) return linePoint1;
  if (projectedDistance >= (linePoint2 - linePoint1).norm()) return
   ↪ linePoint2;
  return linePoint1 + lineDirection*projectedDistance;
}

Vector3d closestPointOnLineSegment(const MatrixXd lines, Vector3d
 ↪ point, double &fractionCompleted) {
  int closestIndex = 0;
  double closestDistance = 10000;
  Vector3d closestPointOfAll;
  int num_lines = lines.cols()-1;
  for (int i = 0; i < num_lines; i++) {
    Vector3d closestPoint = closestPointOnLine(lines.col(i),
     ↪ lines.col(i+1), point);
    double distance = (closestPoint - point).squaredNorm();
    if (distance < closestDistance) {
      closestDistance = distance;
      closestPointOfAll = closestPoint;
      closestIndex = i;
    }
  }
  fractionCompleted = (double)closestIndex/(num_lines-1);
  return closestPointOfAll;
}
```

Listing 4.7: Function closestPointOnLineSegment in functions/functions.cpp

4.2.5 Clamping joint positions

A method for limiting which joint positions each joint can move to was implemented. This causes the joint to just stop at the limit instead of throwing an error. The desired joint positions are calculated from the desired velocities from the force controller. Then, the desired joint positions are clamped to their limits. An error is defined by the difference between the clamped positions and the desired positions and sent to a PID controller. As the PID controller tries to minimize the error, the output is added to the desired joint positions as the ”correction”.

The code for this method is shown in Listing 4.8. The PID controller is tuned with $k_p = 0.002$ and $k_d = 0.00001$, which makes the joint spring back from its limits with a distinct motion, but with pleasant damping.

```
//Calculate desired joint positions
Eigen::VectorXd q_desired = q_d + qd_desired*dt;
//Clamp the desired joint positions to the limits of each joint
Eigen::VectorXd q_clamped = q_desired.cwiseMin(q_max).cwiseMax(q_min);
Eigen::VectorXd joint_error = q_clamped - q_desired;

Eigen::VectorXd q_pid = jointPid.computePID(q_clamped - q_desired, dt);

//Add the "correction" from the PID controller to the desired joint position
q_desired += q_pid;
qd_desired = (q_desired - q_d)/dt;
```

Listing 4.8: Excerpt from robot loop where the motion of the robot joints are clamped

4.2.6 PID controller

To make the use of the PID controller easier, a custom class was made with an implementation of a simple PID controller. The code for the PID class is shown in Appendix A. A PID controller can then be initialized and used as shown in Listing 4.9. Both the input error and the output is a vector of size n, which has to be defined on line 2 in the example. This is practical for making a PID controller when using multiple values, for example, a 3D vector.

```
1  Pid some_pid = Pid();
2  some_pid.init(Eigen::VectorXd::Zero(7));
3  some_pid.setParameters(0.1, 0, 0.02);
4  Eigen::VectorXd pid_result = some_pid.computePID(my_error);
```

Listing 4.9: Example use of the pid class

4.3 User interface

The user interface is the application that links the user and the apparatus. It is considered the starting point of operating the machine. The different states of the interface are presented in a natural order of how someone would use the program.

4.3.1 GUI page: Select Patient

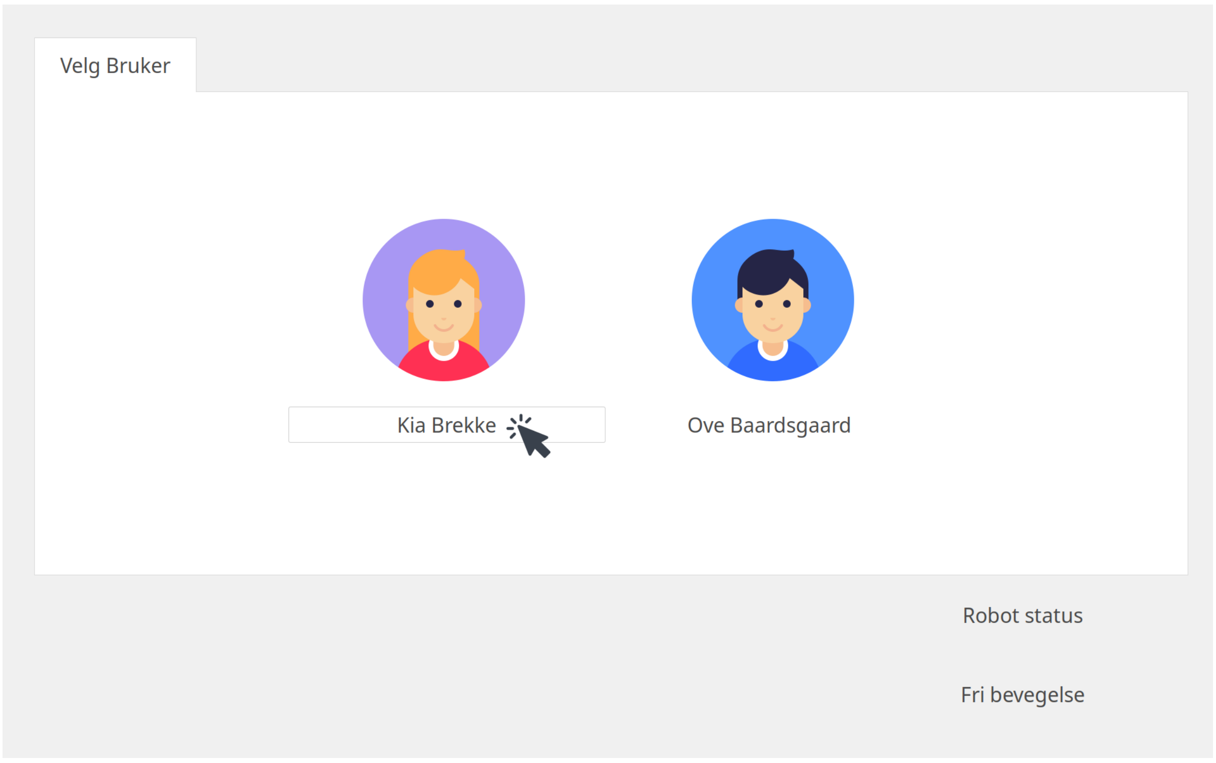
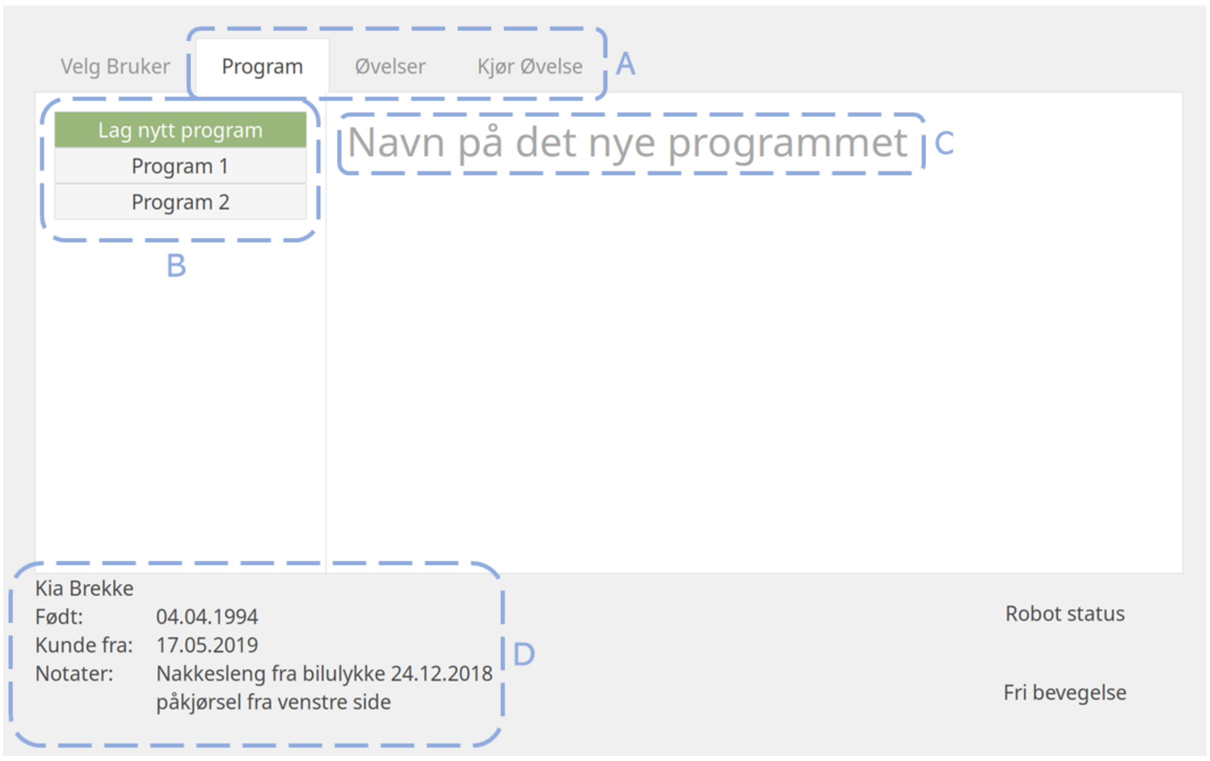


Figure 4.3: Screenshot of GUI page where a patient is selected

When the application is started, the GUI opens and fills the whole screen. The first page appearing is where a patient is selected, as shown in Figure 4.3. On this page, the user is to select a patient which then opens the personalized program page. The patients are represented with a photo and a name for identification.

In the bottom right corner, the state of the robot (free movement, constrained movement, idle state, disconnected, error, etc.) is shown.

4.3.2 GUI page: Program



Kia Brekke

Født:	04.04.1994
Kunde fra:	17.05.2019
Notater:	Nakkesleng fra bilulykke 24.12.2018 påkjørsel fra venstre side

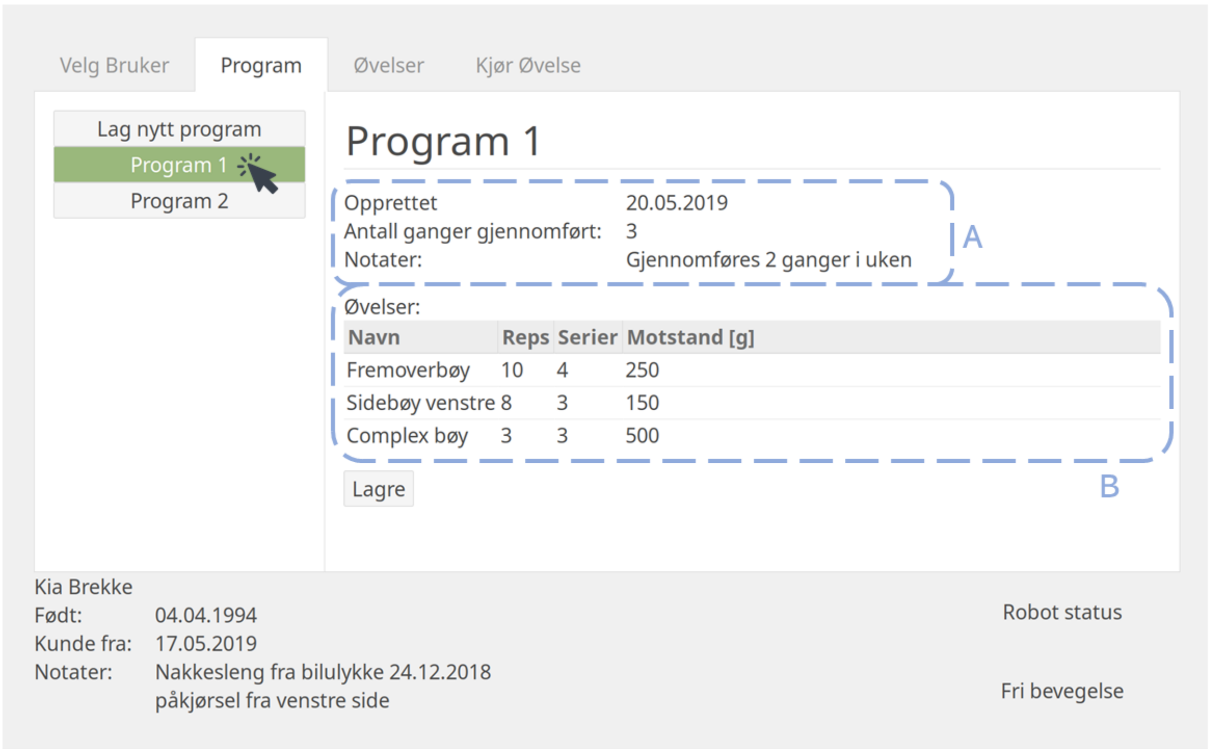
Figure 4.4: GUI Tab: Program

When a user is chosen, the program page is opened automatically, and the other tabs in the program are revealed (Figure 4.4-A). The program page consists of a main frame divided into two as shown in Figure 4.4. The program page is intended to be a personal page where information about programs are available and new programs can be created.

The column on the left is the program chooser, where a previously made program can be reloaded or a new program can be made (Figure 4.4-B). The column on the right contains information about the currently chosen program. In Figure 4.4 ”Lag Nytt Program” is chosen, and therefore no information is displayed yet. There is a text section (Figure 4.4-C) to input a name for a new program.

The footer on the bottom of the page is universal, it is shown on all pages. The footer has on its left side information about the selected patient (Figure 4.4-D). On the right, the robot status is displayed.

Previously created program



Program 1

Opprettet	20.05.2019
Antall ganger gjennomført:	3
Notater:	Gjennomføres 2 ganger i uken

Øvelser:

Navn	Reps	Serier	Motstand [g]
Fremoverbøy	10	4	250
Sidebøy venstre	8	3	150
Complex bøy	3	3	500

Kia Brekke

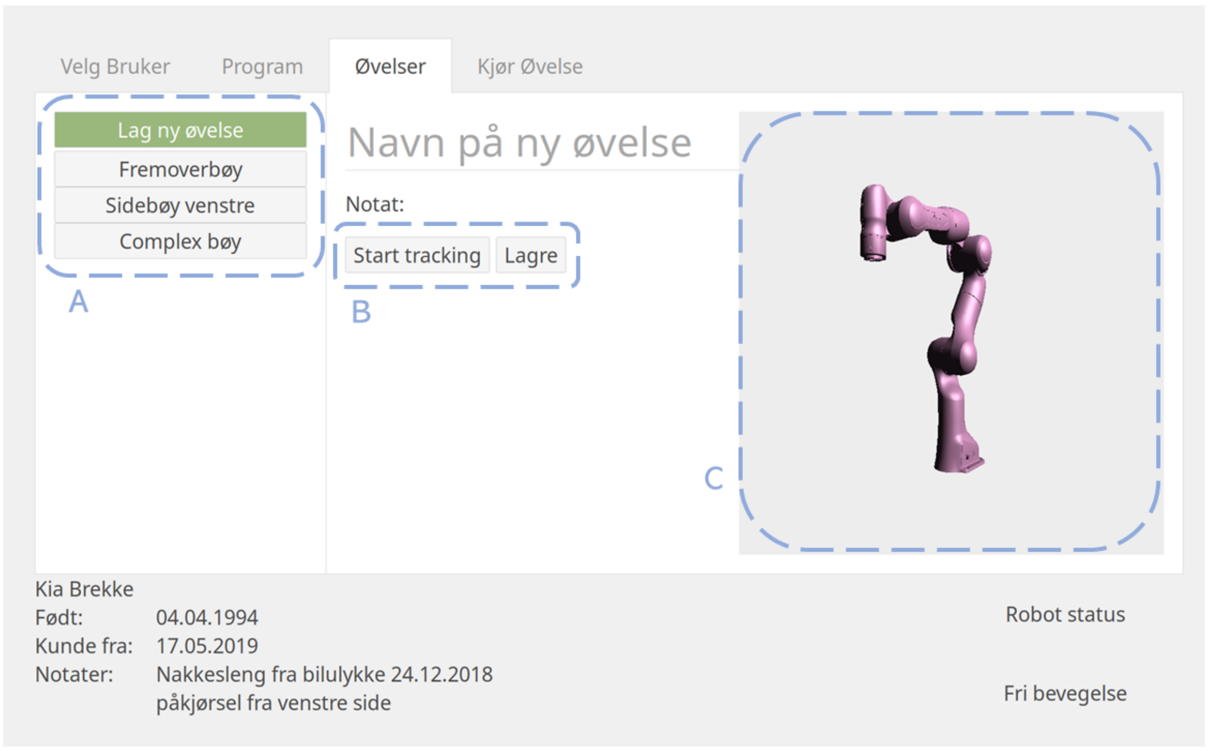
Født:	04.04.1994
Kunde fra:	17.05.2019
Notater:	Nakkesleng fra bilulykke 24.12.2018 påkjørsel fra venstre side

Figure 4.5: GUI Tab: Program, info Program 1

When a previously made program from the list (Figure 4.4-B) is chosen in the program page, corresponding information is displayed in the right area, as shown in Figure 4.5.

Underneath the program title, general information about the program is displayed (Figure 4.5-A). A program consists of one or more exercises, displayed in Figure 4.5-B. The information regarding the exercises is displayed as a table with the exercise name, number of repetitions in a series, number of series to perform and resistance for each movement. The idea is that everything on this page should be editable, hence the button $\boxed{\text{Lagre}}$.

4.3.3 GUI page: Exercises



Kia Brekke
Født: 04.04.1994
Kunde fra: 17.05.2019
Notater: Nakkesleng fra bilulykke 24.12.2018
påkjørsel fra venstre side

Figure 4.6: Screenshot of exercises page in GUI

The exercise page is reached by selecting the tab marked *Øvelser*. On the page, as shown in Figure 4.6, one can study a previously recorded movement or choose to record a new movement. The menu on the left (Figure 4.6-A) contains a list of saved exercises as well as an option to create a new exercise. The idea is to make the exercise information accessible and the exercises editable.

When Lag ny øvelse is selected, the area to the right contains buttons to record a new movement (4.6-B). The exercise is given a title by marking ”Navn på ny øvelse” and typing in the desired name. Below the title of the exercise is a field to keep notes about the exercise.

The page also contains an OpenGL section (Figure 4.6-C) where a demo of the exercise is supposed to be shown using a 3D model of the robot. Currently, it shows a 3D model of the robot which can be turned and moved using the keyboard.

Track movement and save as a new exercise

Tracking of a new movement starts by clicking on the button $\boxed{\text{Start tracking}}$. This action will make the button turn green, and the button label will change to ”Stopp tracking”. The recording of the movement will proceed until the button is clicked again. When a movement has been recorded, one can decide to track again or save it as a new exercise.

A movement (or path) is saved when the button $\boxed{\text{Lagre}}$ is pushed. The exercise will be saved as the name given in the exercise title. The button label will simultaneously change to $\boxed{\text{Lagret!}}$. The instant feedback from the button label, reassures the user of a successful saving of the exercise. Figure 4.7 demonstrates the different states of the buttons.

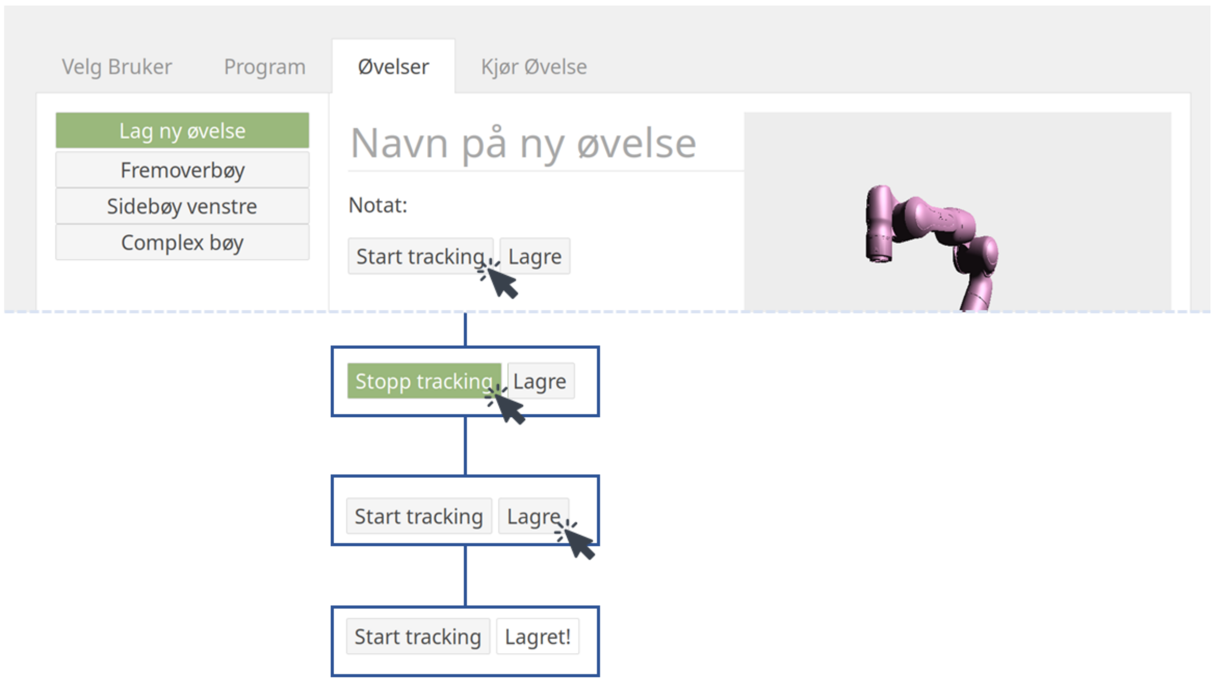
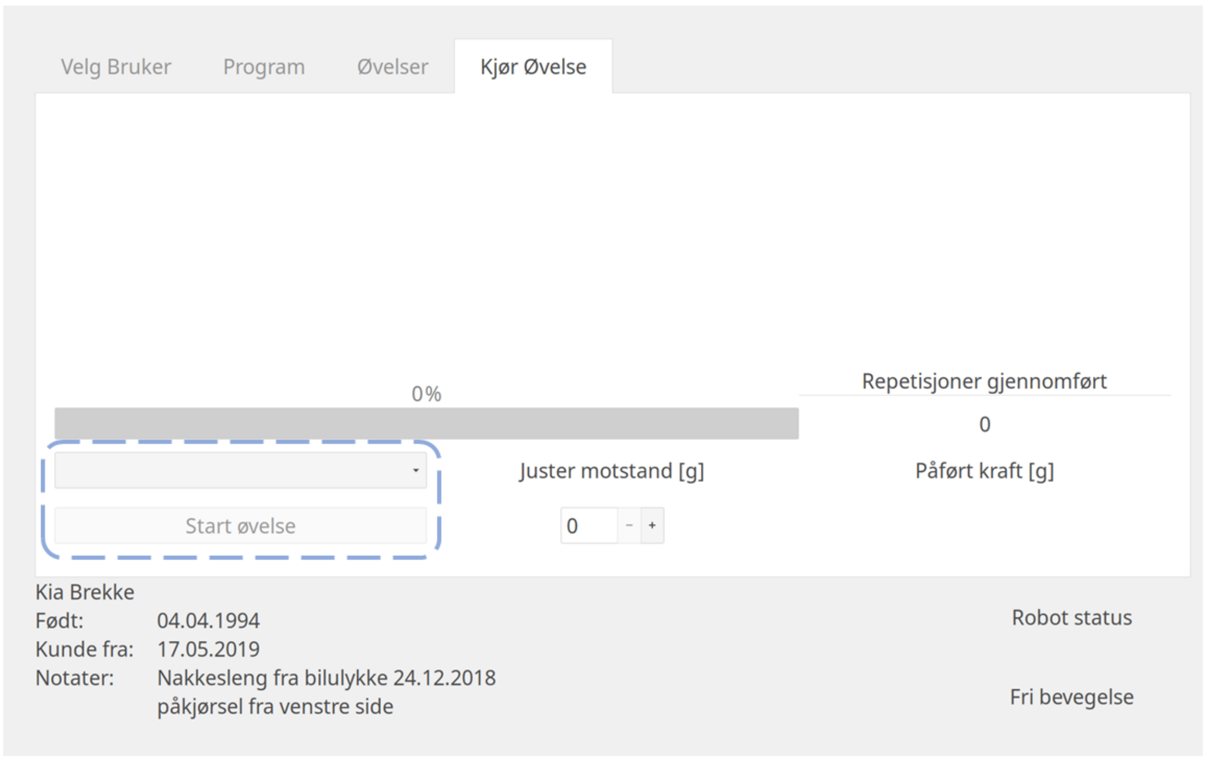


Figure 4.7: Screenshot of exercise page: Button states when tracking a movement

4.3.4 GUI page: Run exercise

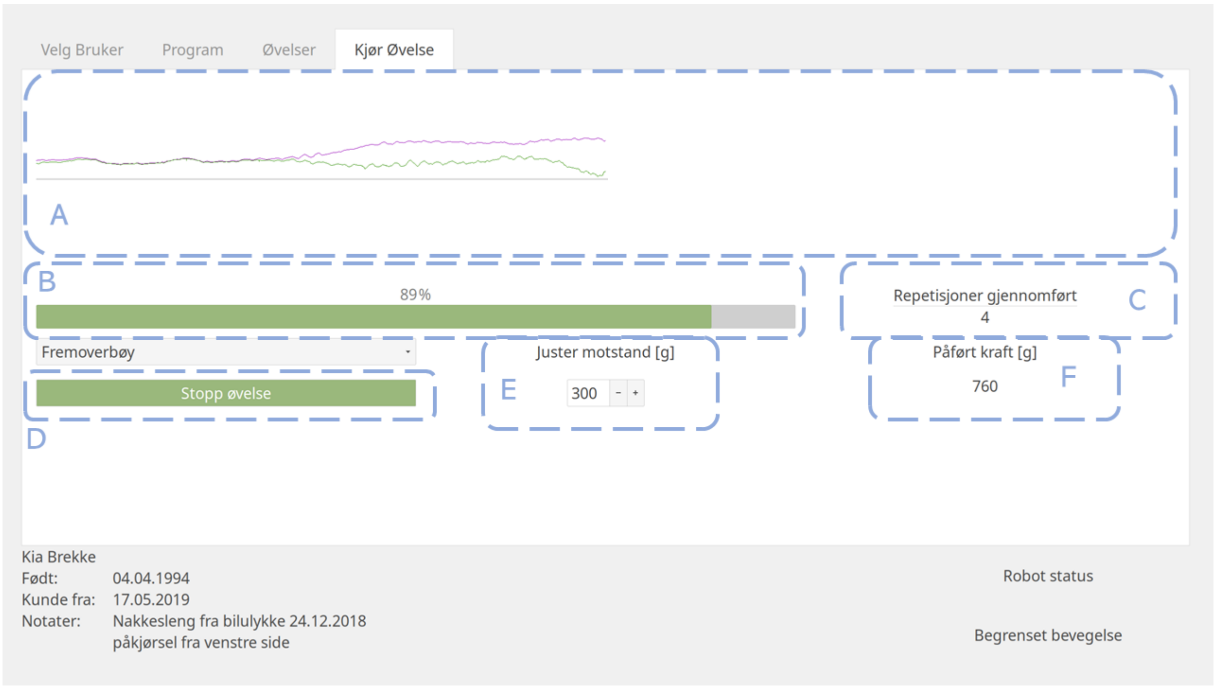


Kia Brekke
Født: 04.04.1994
Kunde fra: 17.05.2019
Notater: Nakkesleng fra bilulykke 24.12.2018
påkjørsel fra venstre side

Figure 4.8: Screenshot of initial frame in ”run exercise” page

The page where an exercise (or program) can be performed, is reached by selecting the tab marked ”Kjør Øvelse”. Figure 4.8 shows a screenshot of the initial state of the run exercise page. The area marked in Figure 4.8 is where an exercise is chosen. The button $\boxed{\text{Start Øvelse}}$ is deactivated until an exercise is chosen. The dropdown menu contains the same exercises as in the exercises page. When an exercise is picked from the dropdown menu, the button $\boxed{\text{Start øvelse}}$ is activated and becomes clickable. When pushing the button $\boxed{\text{Start øvelse}}$, the robot state will change to a constrained state, where it is only possible to follow the path of the chosen exercise.

Run an exercise



Kia Brekke
Født: 04.04.1994
Kunde fra: 17.05.2019
Notater: Nakkesleng fra bilulykke 24.12.2018
påkjørsel fra venstre side

Figure 4.9: Screenshot of running an exercise in ”run exercise” page

Figure 4.9 shows a screenshot of a exercise in motion. The robot status has changed to constrained movement. The upper half of the page contains a 2D plot of the force applied by the patient (pink plot) and a 2D plot of the force component applied in the exercise direction (green plot), as shown in Figure 4.9-A. The current force value is always displayed in numbers (Figure 4.9-F). The value is rounded for legibility.

Underneath the plot is a progress bar (Figure 4.9-B) . It shows how much of the exercise is completed. When the exercise is completed and the progress bar shows 100%, The exercise counter (Figure 4.9-C) increases by one. After reaching 100% the progress bar must get below 2% before another repetition can be counted.

The resistance from the robot is adjusted with the spin button (Figure 4.9-E). It is also possible to write the desired resistance directly in the number field.

When an exercise is started, the button previously marked $\boxed{\text{Start Øvelse}}$ changes to $\boxed{\text{Stopp øvelse}}$. The button $\boxed{\text{Stopp Øvelse}}$ can be pushed at any phase of the exercise. Pushing the button will change the robot state to floating mode, and the features will be deactivated.

4.3.5 Developer mode

When pressing the key $\boxed{\text{D}}$, some additional pages appear which can be used during development. The additional pages consist of a model page and a PID page.

Developer page: Model

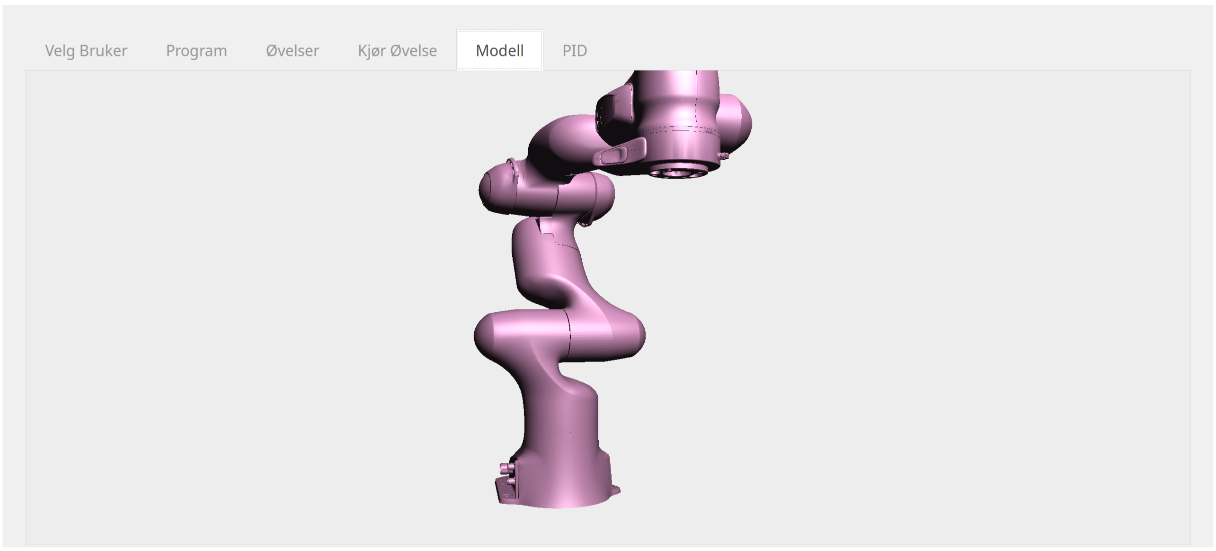


Figure 4.10: Screenshot of developer page: Model

The model page consist of a OpenGL environment (`gl\_area`) to test new OpenGL features, as shown in Figure 4.10. It is a clean canvas to test new code and features without any distractions.

Developer page: PID

The PID page is used to adjust the programming parameters during run-time. The buttons can be used to change the corresponding values in the code, where the letter represents which parameter it adjusts.

P, **I** and **D** represents the parameters for the PID-regulator k_p, k_i and k_d, described in Section 2.2.1.

M represent the virtual mass, described in Section 3.2.4.

R_i represent the radius of the simulated object, described in Section 3.2.5.

F represent the cutoff frequency, described in Section 2.2.3.

The labels underneath each set of buttons get updated to the current value every time a button is pushed. The page facilitates smoother development and testing.

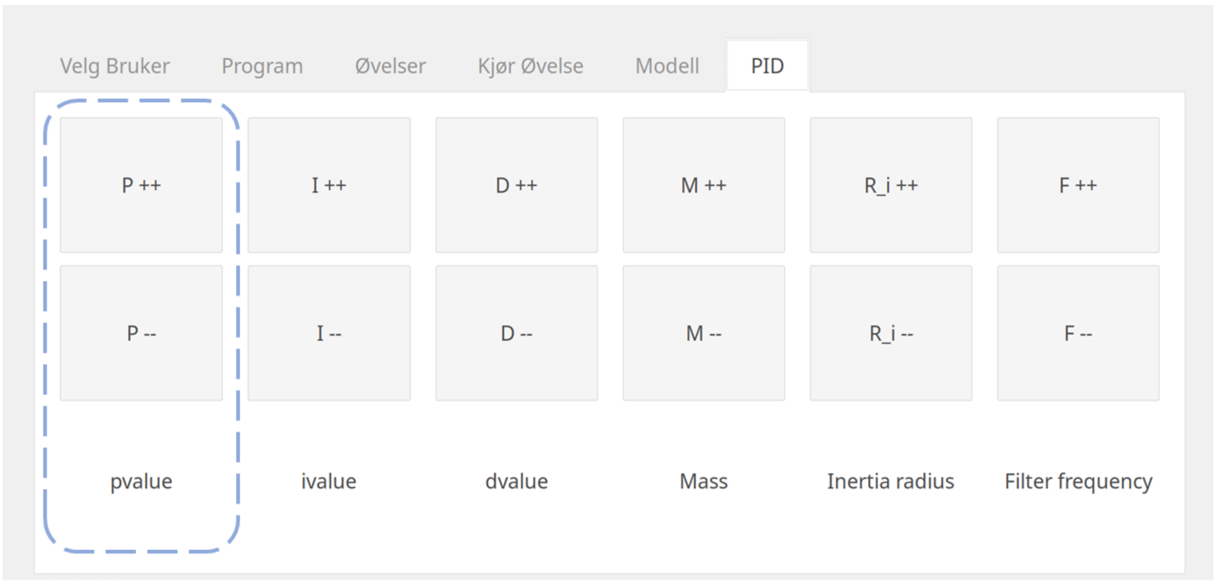


Figure 4.11: PID values adjustment

4.4 Helmet and helmet mounting mechanism

The current solution includes a helmet-like plastic shell with inflatable air pockets and a magnetic connection to the robot flange. To attach the helmet to the robot, the magnetic connection simply snaps the helmet into place and forms a rigid connection. In the case of unwanted robot behavior that causes too large forces on the head, the magnetic connection will disconnect the helmet from the robot. This secures that the robot is unable to cause serious injuries to the patient.

4.4.1 Helmet

The helmet is designed based on the previous used Etto Twister alpine helmet, but with dimensions to fit all humans and a thin-walled structure. The top has a slider mechanism to adjust the position of the magnetic connection, which is a circle with 16 magnets which makes the helmet connect with the robot. A picture of a 3D printed prototype of the helmet is shown in Chapter 3, Figure 3.15.

4.4.2 Magnetic connection

The main function of the magnetic connection is to add safety to the product since a robot with powerful motors is to be connected to a human head. Also, the connection adds the possibility of changing different end-effectors quickly.

The connection consists of a 3D printed structure with a conical surface with 16 magnets with a pulling force of 0.9kg each along the perimeter as shown in Chapter 3, Figure 3.13. The part has two holes for mounting to the robot with screws. The type, number, and placement of the magnets causes the magnets to make a rigid connection but is still able to disconnect with a reasonable force in all directions.

Chapter 5

Set-up Guide

This is a guide on how to set up the project and the current solution for further development.

Requirements

This section specifies the equipment needed to set up the Panda robot and the hardware needed to run the current project.

Ethernet cable

The Ethernet cable between the Panda Control and the computer needs to be able to operate at least 1000 Hz. Franka states that the cable needed is a ”high-performance Ethernet cable” (Cristian Kodura, Franka Emika, 2019). Good results were experienced with a Cat5e cable. If you experience a lot of communication errors, you should check if the Ethernet cable is damaged, or if the cable is too long. The cable cannot exceed 100 m (Panduit, 2013).

Graphics card

To run OpenGL, a graphics card is needed. Most modern computers and laptops meet the requirements. To check whether a computer is compatible, check the requirements on the Focus multimedia web page (Focus Multimedia Ltd., 2015).

Network card

The Franka Docs requirements (Franka Emika GmbH, 2017a) states that the minimum requirements for network cards are 100BASE-TX. Franka mentioned by mail that they experienced good results with ”Ethernet Server Adapter I350-T4V2 from Intel” (Cristian Kodura, Franka Emika, 2019).

Computer and operating system

The operating system needs to be Linux based, and it needs to be an RTOS (described in Section 2.3.4).

Powering up Panda robot

Place the Panda robot on top of the frame structure and secure it with bolts, to make sure the robot does not get damaged during use.

Follow further instructions on how to connect the robot with the computer on Franks Docs (Franka Emika GmbH, 2017a).

Download project on GitHub

Download the GitHub repository of the project from https://github.com/kiabrekke/devomed. To clone the project to your computer, write the following in the terminal.

```
>> git clone https://github.com/kiabrekke/devomed.git
```

I addition to the project file, some additional packages must be installed (Eigen, Poco, OpenGL, and gtkmm).

```
>> sudo apt-get update
>> sudo apt-get install libeigen3-dev
>> sudo apt-get install libpoco-dev
>> sudo apt-get install freeglut3-dev
>> sudo apt-get install libgtkmm-3.0-dev
```

To build the project, create a build folder, then build the project from that folder.

```
>> mkdir build
>> cd build
>> cmake build .
```

To start the program, navigate to the build folder and type:

```
>> ./main_start_threads
```

Chapter 6

Discussion and Further Work

In this section, we will discuss the results and solutions made, but also discuss ideas and possibilities for further work, with the main goal of achieving an MVP in mind.

Demonstration of current solution

At the end of the project, the current solution was presented with demonstrations of all functionality developed. The supervisors and Morten Leirgul from Firda Fysmed were present, as well as other interested parties.

Presenting the product was in the form of a use case, a patient coming into a clinic for a rehabilitation session, to show a realistic use of the concept. The intention was to test if the concept idea applied in practice still fulfilled the expectations of Firda Fysmed. The demonstration resulted in a lot of valuable comments and discussion on further work. The feedback for each part is presented under the coherent section.

6.1 Product

6.1.1 Compliant robot control system

In this thesis, a compliant robot control system in the Cartesian space was developed. The focus was to make the robot able to move by hand with little resistance, which was achieved. The control system still has some flaws which cause unstable behavior in some positions. Also, the ability to rotate the end-effector needs to be improved. In this section, the parts of the control system are further discussed.

Getting sensor data

The current solution uses the built-in function from libfranka for calculating forces in task space from joint torques. It was discovered that this method is a bit inaccurate, especially when the robot is moving, which is also confirmed by Franka in an email conversation (Cristian Kodura, Franka Emika, 2019). We still do not know the degree of inaccuracy, how much this affects the force controller, or how much accuracy is needed for the force controller we want.

We chose to filter the force signal using a low-pass filter to remove measurement noise. This does remove some of the accuracy from the signal. Signal filtering is a complex subject, and there might be other filters that are better at removing noise and preserving signal accuracy.

Velocity controller

The Franka Panda comes with a built-in interface for controlling Cartesian velocities. It seemed easiest to use this at first, but we encountered a lot of problems which seems hard to overcome. We contacted Franka about this issue, but they could not explain the source of our issues. The simplest solution was instead to use joint velocities.

To calculate the pseudo-inverse Jacobian, we used a method from the Eigen library as proposed on Stack Overflow by Sean (2017). He also comments that other methods should be used instead of this. We felt this method gave good results, and don't see the need for a new method, but it can be kept in mind for further work.

By using joint velocities, the joints are velocity, acceleration and jerk limited individually. That means, if you command a Cartesian velocity where one or more joints gets limited, the direction and magnitude of the performed Cartesian velocity are different. We suspect that in this use case, the speed of motion is so slow that it will not cause an issue.

The velocity controller is capable of producing the desired velocities with a fast response and does not need much further work before an MVP.

Force controller

The force controller which worked best was based on a dynamic model of an object on a surface with friction. This method of controlling a robot arm has not yet been researched as far as we know. The friction force seemed to dampen the oscillations of the force controller much better than the more commonly used admittance controller.

However, to make the controller stable, it was necessary to lower the impedance of each joint, which also contributes to the damping of the force controller. For further work, it might be useful to investigate how this works and see if a better system can be made based on these observations.

When lowering the joint impedance, the robot feels less rigid and more like a human arm. In this project, accuracy is important as small movements need to be controlled. But a more human-like arm may improve the comfort of the patient. It may be interesting to test if this can be a benefit in the final product.

To figure out how the force controller can be improved and how rotation can be included in the force controller, a force-torque sensor might be useful. By making a force controller with an external force-torque sensor, the force estimation issues can be ruled out of the equation. If then, the force controller is successfully made, one can go back to the force estimation and see if it can be fitted to work on the new controller.

To add the wanted resistance force, we added it as a friction force. This adds a constant force when the patient is moving the head but stops adding force if the patient stops. The MCU uses cables and weights, which gives a different experience. There the patient has to provide static force to hold the weight when stopping.

We think our method feels safer for the patient since you can stop moving at any time and not hold any force. In regular weightlifting exercises, you have a concentric and eccentric phase, which is that the muscle is contracted while providing force, and extended while providing force. In our method, the patient never experiences the eccentric phase. If this is an issue needs to be discussed and researched further, and maybe also test the different alternatives in adding force.

Tracking a path

To avoid large amounts of points when tracking the path, we sampled the recording at 100Hz. This provided a smooth motion, but the path is very detailed and can be unnatural to follow afterward. A suggestion is therefore to use a smoothing algorithm, which reduces the detail level but keeps the overall shape of the path. A common algorithm for such task is the Douglas Peucker algorithm, where there exists some free to use C++ implementations online.

After tracking a path, the list of points needs to be saved as an exercise motion. In this thesis, we chose to save it as a .csv file, since it is easy to implement and debug. For further work, it is recommended to save this in a database, along with user info, programs and exercise info, etc.

The privacy laws have been briefly discussed regarding how user info is stored in a database, and if the info is to be used between different clinics. An online solution could be used, but the security of the info needs to be considered. An alternative is that the patient keeps a memory stick with their own info, which is more inconvenient but improves security.

6.1.2 User interface

GTK

GTK with the C++ library gtkmm and GUI design tool Glade did work as intended for this project. There are however not many tutorials for these tools, and the documentation can be hard to understand. Other tools may be easier to get started with, but after the initial learning period, we felt that gtkmm and Glade are well suited to make a GUI for this application.

OpenGL / graphics library

Grid and axis
Since OpenGL isn't a natural graph tool, all elements need to be added separately. Grid and axis are created by drawing lines between fixed vertices. This wasn't added to the current solution, due to priorities.

3D model
The main focus when developing the 3D model was to lay a good foundation for continued development. In this project, the import of a 3D model of the robot was achieved, as well as applying texture, lighting, and controlling the model.

The 3D model of the robot has been split into 7 obj-files, one for each joint. The files can be found in the *objfiles* folder in the project folder. To make the model move as the robot, all joints need to be imported into OpenGL as 7 different models. The models then need to be positioned in the 3D environment to form the complete robot.

The position and angle of each joint can be provided from the robot data. The dimensions of the parts are already known. A function to transform the joint angles into translation and rotation matrices for each joint is needed. All these steps combined should make it possible to fulfill the requirement of projecting a movement on the 3D model.

GUI design

The current GUI design was primarily made to test the concept idea, and only the most important features were added. The GUI received great feedback from Morten Leirgul and confirmed that the components implemented were what they wanted. Working on the GUI should therefore be continued in future work, by improving the design and developing more functionality.

User test
To ensure a simplistic and intuitive user interface, a user test could be performed. A good way to test if the GUI is intuitive is to invite testers that are not very familiar with the concept and then giving them access to the GUI, giving them exercises to complete before explaining how it is supposed to work. This will give the developers a great insight into how people think when working with the GUI, and how pleasant they find the design components.

Working prototype
To make a working prototype for the MVP, some of the functionality left to be implemented are:

- A log-in page for the physiotherapist/medical staff to secure that the current user is allowed to access the patient's data.
- A database of patients with associated programs and exercises.
- Functionality for creating and editing patient data, programs, and exercises in the database.

- Functionality for adding exercises to programs, and the possibility of running all exercises in a program consecutively.
- Preview of the exercises using the 3D model area in the GUI.
- Presentation and mapping of the patient's AROM, and progress over time

Graphics libraries not based on C++

Graphics libraries in C++ were mainly evaluated when choosing a suitable tool, and eventually, OpenGL was chosen for the current solution. However, it is possible to use plotting tools in other languages than C++. Other languages like Python have a bigger selection of plot tools that are easy-to-use, modern and well documented. To incorporate two applications with different languages a server solution with an API interface is needed.

We considered making a user interface using other languages but wanted to test using C++ first. Using gtkmm and Glade was a promising solution for this project, and we recommend using this in further development.

6.1.3 Helmet and helmet mounting mechanism

Magnetic connection

To maximize the force transmission from the head to the robot's flange, the flange-to-head distance needs to be as small as possible. Our solution reduced this distance to 39mm, but without the slider for adjusting the position of the connection, the distance can be reduced to 10mm. We suggest that putting the magnets directly on the helmet is a better option, but the ability to adjust the position then needs to be investigated.

We were quite satisfied with the amount of force needed to release the connection in this prototype. The force also seemed to be even in almost all directions. In addition, the connection is very rigid in all directions except rotation along its axis. This could maybe be improved a bit by experimenting with the placement of magnets and the polarity of the magnets. Another solution might be to experiment more with the addition of small bumps as we used in the first prototype, to increase torsional strength.

The only direction that required a high force to release the connection was when pulling along its axis. Then, all the magnets need to disconnect at the same time, requiring about 14.4kg to pull the connection apart. But as we discovered, the connection can much easier be tilted to disconnect. Therefore we regard this to not be an issue.

This version of the magnetic connection was 3D printed with PLA plastic but is very rigid. For the MVP, 3D printing seems to be good enough, but as a final product, more long-lasting materials such as aluminum should be considered.

In the future, it might be an option to use the robot as a rehabilitation apparatus for other injuries as well. Therefore, such a magnetic connection can be designed as a universal connection to attach other tools also, such as an arm or hand fixture.

Helmet

The helmet was developed further to reduce the flange-to-head distance, and to see if it was possible to make one universal helmet that fits all head shapes and sizes.

With the current concept of inflatable air bags to secure the head in the helmet, it is important to consider that air is compressible and can easily move inside the air pockets. By making a large helmet, the air pockets get bigger, and the air is easier to compress, which causes looseness. The thesis of Gælok and Strand (2017) proposed to solve this by making multiple helmets of different shapes to ensure that the air pockets are small enough.

We see that the concept of air pockets does not work with a universal helmet, but we believe that making multiple helmets is unnecessary. The air pockets may be improved by further testing with multiple small air pockets or filling the bags with a foam substance etc.

Gælok and Strand (2017) also developed a concept with a more direct mechanical solution with straps. As it may be hard to make the air pockets rigid enough, a mechanical solution might be interesting to investigate further.

In addition, we discovered that the previous theses had not addressed how to deal with patients with long hair. Slåttsveen and Tolo (2015) mentioned that hair between the helmet and the head provided severe looseness and needed to be avoided. Long hair needs to be put into a ponytail etc. In a closed helmet, there might be difficulties in fitting all the hair, which is something that needs to be considered.

6.1.4 Chair and frame structure

Chair solution

As outlined in the problem description, the chair wasn't one of the priorities to work on, but we have some thought on possible adjustments to be made.

The frame structure is not sturdy enough to withstand forces applied to the chair or structure. The robot is mounted on top of a tall steel column, which makes it vulnerable to vibrations. When rocking the chair (which can happen if a patient abruptly sits down or rises) the frame structure will shake as well, making the robot base move. These movements interfere with the force sensors in the robot, and for large enough movements, the robot stops and throws a reflex error.

The frame structure may need a total redesign, to ensure that the platform for the robot is totally vibration free. A solution where the robot frame and the chair are separated or at least vibration damped may be beneficial. This can also enable the possibility of changing the position of the patient relative to the robot.

Adjustable robot platform

Another suggestion to improve the chair solution is to make the robot platform adjustable instead of the seat. The robot is much lighter than a person, requiring a less powerful actuator. Another benefit is that it would make it easier to change the chair with a design that enables the patient to either stand or sit during the treatment. This solution was also discussed in the thesis by Brattgjerd and Festøy (2018) as a ”hybrid design” as shown in Figure 6.1.

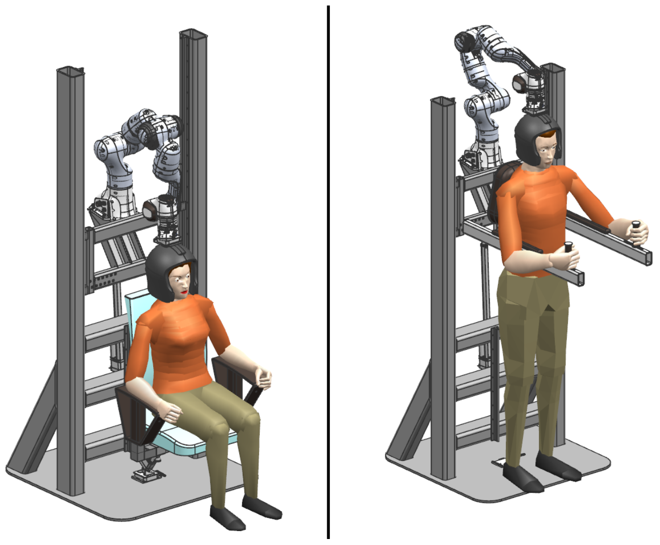


Figure 6.1: CAD sketch of the hybrid seat design from Brattgjerd and Festøy (2018)

The robot can measure the optimal position once the patient is well seated, and self adjust by moving the robot platform. This feature would let the robot operate with maximum range for all patients.

Placement of robot platform

The previous master thesis Brattgjerd and Festøy (2018) explores different ways to position the robot. However, Franka Emika states on Franka docs (Franka Emika GmbH, 2017a) that placing the robot any different than perpendicular to the ground will void the warranty.

Even if the warranty is voided, it doesn’t necessarily mean it has to be dangerous or difficult to change the inclination of the robot platform. However, it should thoroughly be simulated and tested before attempting to vary the placement. The position of the robot in the current concept seems to have a sufficient range of motion for further developing and initial testing.

6.2 Market analysis

During this thesis, it was considered to make the project commercial. Together with a Spark* NTNU (Spark NTNU, 2018) supervisor, the market potential for whiplash rehabilitation apparatuses was analyzed. Both companies offering treatment of whiplash injuries, organizations for whiplash injured people and whiplash injured patients were contacted and interviewed about the process of treating a whiplash injury in Norway. The interviews intended to find the most pressing areas to be improved and to evaluate the market potential for such a product.

Several whiplash injured patients report about difficulties getting a diagnosis, finding information about possible treatment options and lack of expertise at many clinics. As injured, the information available is limited, and a lot of patients search for years to find a treatment that might help. If they do find a good specialist, it is quite expensive and they might have to travel long distances. The organizations are frustrated that it is so hard to get help and support through the Norwegian welfare system for these specific injuries.

The industry also admits a lack of knowledge and information. A staff member from a chiropractor clinic talked about the lack of a standardized and promised treatment of whiplash injuries. He emphasized varying results from different studies and the controversial opinions regarding rehabilitation methods as big challenges for increasing the competence of whiplash injuries. The clinic offers treatment for whiplash injuries, but it is not their specialty. They are not primarily interested in an apparatus, but very interested in the field being more researched.

Based on the interviews, it seems like many people could benefit from access to an apparatus like this, but few clinics have the expertise to handle such a device. It would therefore be hard to sell more than a couple of devices in Norway. It might be possible to market the project towards an international market or educational purposes and researching projects.

The market potential is definitely present, but the product is in the introduction stage of the product life cycle (Living Better Media, 2019) which requires a high cost of research and testing before any return can be expected.

The market needs a product to drive research and development in the field of whiplash injuries. The current concept should therefore initially be developed into a research tool rather than a commercial product.

Chapter 7

Conclusion

Problem description

The long term vision is to make a rehabilitation apparatus for whiplash injured patients, which is a comprehensive and complicated problem. The concept idea is to use a robot arm with a helmet-like device for fastening the head to guide a patient through neck exercises. This master thesis focus on some parts of the apparatus, which has been seen as most important at the current stage for developing a minimum viable product (MVP).

The areas of focus were:

- *Control of the robotic arm, making it compliant with the ability to adjust resistance, and guide the movement through a recorded motion.*
- *A graphical user interface to control the functions of the robot, and store patient info such as custom programs and exercises.*
- *Helmet mounting mechanism which securely fastens the helmet to the robot, while keeping patient EHS in focus by disconnecting in case of emergencies or malfunctioning.*

7.1 Compliant robot control system

In this thesis, a control system for the robotic manipulator Panda by Franka Emika has been developed. The control system is a force controller in the Cartesian space with very low resistance. The controller for making the rotation of the end-effector still needs to be improved. In addition, the controller needs to be made more stable.

It is interesting to be able to use the internal force estimation of the robot, to lower the cost and complexity of the product. However, the force estimation signal may not be accurate enough, and a force-torque sensor could be necessary.

The force controller uses a velocity control based on the pseudo-inverse method from the Eigen library. This method inverts the Jacobian matrix acquired from libfranka, which is used to transform Cartesian velocities into joint velocities. This method worked well and provides robust control of the Cartesian velocity.

A method of constraining the robots motion to a recorded path was also developed, using an algorithm for finding the closest point on a path and a PID controller for constraining the end-effector position. With the force controller and this method, the robot was easy to move through any recorded path.

Since the robot stops and throws an error if the joints are moved outside their limits, a method for constraining the joint positions with a PID controller was developed. This made the robot apply a smooth resistance if the joints approached their limit.

The concept of using a robot arm as a motion platform for the rehabilitation apparatus is further developed towards an MVP, but there are still some issues left to be solved.

7.2 User interface

A user interface linking the user and the apparatus has been developed. The interface controls the different states of the robot and lets the user record movements and follow a constrained path.

The graphical interface has been made using Glade as a foundation for further work and contains different pages where exercises and exercise programs are shown. OpenGL was used to display a 3D model of the robot and a 2D plotting area. Functionality for performing an exercise with adjustable resistance, progress bar, and counter is also included. The user interface has some of the functionality needed in an MVP and needs to be completed.

7.3 Helmet and helmet mounting mechanism

The previous helmet mounting mechanism was struggling with looseness and weight. A new mounting mechanism using magnets was developed with a focus on safety and rigidity.

The magnetic connection consists of a 3D printed structure with a conical surface with 16 magnets inserted along the perimeter. The type, number, and placement of the magnets causes the magnets to make a rigid connection but is still able to disconnect with a reasonable force in all directions. The easy disconnection of the magnets is a safety improvement, in case of a sudden movement from the robot or the patient.

The helmet is designed based on the previous used Etto Twister alpine helmet, but with dimensions to fit all humans, and a thin-walled structure to minimize the distance from the head to the robot. The top has a circle with 16 magnets which makes the helmet connect with the mounting mechanism attached to the robot.

Bibliography

ATI Industrial Automation, 2019. F/t sensor: Axia80-m20. https://www.ati-ia.com/products/ft/ft_models.aspx?id=Axia80-M20, Accessed: 2019-6-3.

Baardsgaard, O., Hafskolt, H. E., 2018. Development of a rehabilitation apparatus for whiplash injured patients. Master's thesis, NTNU.

Berg, O. J., Sunde, Ø. K., 2016. Utvikling av apparat for behandling av nakkeskadde. Master's thesis, NTNU.

Brattgjerd, S. K., 2017. Development of a new machine for rehabilitation of whiplash patients. Master's thesis, NTNU.

Brattgjerd, S. K., Festøy, A. M., 2018. Development of rehabilitation apparatus for whiplash patients. Master's thesis, NTNU.

BTE, 2019a. Evidence-based practice initiative – mcu. https://media.btetech.com/wp-content/uploads/20170821140651/Evidence-based-Practice-Initiative_MCU_2017-Q3.pdf, Accessed: 2019-2-1.

BTE, 2019b. Mcu multi-cervical unit. https://www.btetech.com/product/mcu/, Accessed: 2019-1-30.

Centeno-Schultz Clinic, 2019. Multi-cervical unit. https://centenoschultz.com/multi-cervical-unit/, Accessed: 2019-2-1.

Comnuan, K., 2018. Jacobian calculator. https://comnuan.com/cmnn04/cmnn04003/, Accessed: 2019-6-3.

Control Tutorials for MATLAB and Simulink, 2019. Extras: Generating a step response in matlab. http://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_step, Accessed: 2019-6-6.

Cooper, D., 2015. Derivative action and pid control. https://controlguru.com/measurement-noise-degrades-derivative-action/, Accessed: 2019-6-4.

cppreference.com, 2019. Lambda expressions, c++ 11. https://en.cppreference.com/w/cpp/language/lambda, Accessed: 2019-6-4.

Cristian Kodura, Franka Emika, 2019. private mail conversation.

DEWESoft, 2019. Pid control. https://dewesoft.pro/online/course/pid-control, Accessed: 2012-6-2.

Focus Multimedia Ltd., 2015. Check for opengl compatibility. https://focusmm.zendesk.com/hc/en-us/articles/203200801-Check-for-OpenGL-compatibility, Accessed: 2019-6-5.

Franka Community, 2018. Libfranka jacobian inverse. https://www.franka-community.de/t/libfranka-jacobian-inverse/468, Accessed: 2019-6-3.

Franka Emika GmbH, 2017a. Franka control interface documentation. https://frankaemika.github.io/docs/, Accessed: 2019-6-3.

Franka Emika GmbH, 2017b. libfranka: C++ library for franka emika research robots. https://frankaemika.github.io/libfranka/index.html, Accessed: 2019-6-3.

Gælok, T. E. L., Strand, M., 2017. Utvikling av maskin for opptrening av nakkeslengskadde. Master's thesis, NTNU.

Grøtte, H., 2018. A virtual security net and soft motion controller for a 7-dof redundant cooperative robotic manipulator. Master's thesis, NTNU.

Khan, S., 2018. Development of a compliant control system for rehabilitation of whiplash injuries. Master's thesis, NTNU.

Landi, C. T., Ferraguti, F., Sabattini, L., Secchi, C., Fantuzzi, C., 2017. Admittance control parameter adaptation for physical human-robot interaction. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 2911–2916.

Living Better Media, 2019. Product life cycle stages. http://productlifecyclestages.com/, Accessed: 2019-6-5.

Maw, I., 2018. Force and torque sensors for industrial robotics. https://www.engineering.com/AdvancedManufacturing/ArticleID/16602/Starter-Guide-to-Six-Axis-Force-and-Torque-Sensors.aspx, Accessed: 2019-6-4.

Microsoft, 2019. Visual studio code. https://code.visualstudio.com/, Accessed: 2019-6-6.

Neuhaus, J., 2014. Dimensions of the human figure. https://www.jneuhaus.com/human.html, Accessed: 2018-12-20.

OpenGL Mathematics, 2018. glm/closest_point.inl. https://github.com/mosra/glm/blob/master/glm/gtx/closest_point.inl, Accessed: 2019-6-7.

Panduit, 2013. The evolution of copper cabling systems from cat 5 to cat 5e to cat 6. https://web.archive.org/web/20130314150935/https://www.gocsc.com/UserFiles/File/Panduit/Panduit098765.pdf, Accessed: 2019-6-5.

Pertuz, S., Llanos, C., Muñoz, D., 04 2018. Simulation and implementation of impedance control in robotic hand.

Rao, S. S., 2007. Vibration of continuous systems. Vol. 464. Wiley Online Library.

Scrum.org, 2019. What is scrum? https://www.scrum.org/resources/what-is-scrum, Accessed: 2012-6-3.

Sean, 2017. Eigen library - pseudo-inverse of matrix (matlab - pinv). https://stackoverflow.com/a/44465287/5586153, Accessed: 2019-6-4.

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

Slåttsveen, K. B., Tolo, S. F., 2015. First development of new machine for rehabilitation of whiplash patients. Master's thesis, NTNU.

Smith, R., 2018. Obj-loader. https://github.com/Bly7/OBJ-Loader, Accessed: 2019-6-6.

Spark NTNU, 2018. Spark — veiledning for entreprenørskap. https://www.sparkntnu.no/, Accessed: 2019-6-5.

Spong, M. W., Hutchinson, S., Vidyasagar, M., et al., 2006. Robot modeling and control.

The GNOME Project, 2014. Connecting signal handlers. https://developer.gnome.org/gtkmm-tutorial/stable/sec-connecting-signal-handlers.html.en, Accessed: 2019-6-6.

The GTK Team, 2019a. Language bindings. https://www.gtk.org/language-bindings.php, Accessed: 2012-6-2.

The GTK Team, 2019b. What is gtk, and how can i use it? https://www.gtk.org/, Accessed: 2019-6-6.

Trello.com, 2019. Trello. https://trello.com/, Accessed: 2012-6-3.

Ubuntu Documentation, 2019. Ubuntustudio/realtimekernel. https://help.ubuntu.com/community/UbuntuStudio/RealTimeKernel, Accessed: 2012-6-2.

wiki.archlinux.org, 2019. Realtime kernel patchset. https://wiki.archlinux.org/index.php/Realtime_kernel_patchset, Accessed: 2012-6-2.

Williams, T., Kelley, C., 2018. Gnuplot 5.2.
URL http://www.gnuplot.info/docs_5.2/Gnuplot_5.2.pdf

Appendix A

Code

pid.h

```
#ifndef PID_H
#define PID_H

#include <Eigen/Dense>

class Pid {

    public:
        Pid();
        void init(Eigen::VectorXd zero_vector);
        Eigen::VectorXd computePID(Eigen::VectorXd error, double dt);
        void setParameters(double kp, double ki, double kd);

    private:
        double kp;
        double ki;
        double kd;
        Eigen::VectorXd pid;
        Eigen::VectorXd past_error;
        Eigen::VectorXd integral;
        Eigen::VectorXd derivative;

};

#endif
```

pid.cpp

```cpp
#include "pid.h"

Pid::Pid(){
    //init();
    //setParameters(0.0,0.0,0.0);
}

void Pid::init(Eigen::VectorXd zero_vector) {
    this->pid = zero_vector;
    this->past_error = zero_vector;
    this->integral = zero_vector;
    this->derivative = zero_vector;
}

void Pid::setParameters(double kp, double ki, double kd) {
    this->kp = kp;
    this->ki = ki;
    this->kd = kd;
}

Eigen::VectorXd Pid::computePID(Eigen::VectorXd error, double dt){
    //Get new pid only if dt > 0
    if (dt) {

        integral += error*dt;
        derivative = (error - past_error)/dt;

        pid = kp*error + ki*integral + kd*derivative;

        this->past_error = error;
    }

    return pid;

}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(asdf)

list(INSERT CMAKE_MODULE_PATH 0 ${CMAKE_SOURCE_DIR}/cmake)

set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
set(CMAKE_BUILD_TYPE Release)

## Dependencies
find_package(Franka REQUIRED)
find_package(Poco REQUIRED COMPONENTS Net Foundation)
find_package(Eigen3 REQUIRED)
find_package(OpenGL REQUIRED)

set(THREADS_PREFER_PTHREAD_FLAG ON)
find_package(Threads REQUIRED)

find_package(PkgConfig)
pkg_check_modules(GTKMM gtkmm-3.0)

add_library(examples_common STATIC
  src/functions/source/examples_common.cpp
)

add_subdirectory (externalGL)

include_directories(
    common/headers/
    src/functions/headers/
    src/threads/headers/
    cmake/
    externalGL/glfw-3.1.2/include/
    externalGL/glm-0.9.7.1/
    externalGL/glew-1.13.0/include/
    graphics/headers/
    .
    ${EIGEN3_INCLUDE_DIR}
  ${GTKMM_INCLUDE_DIRS}
)

target_link_libraries(examples_common PUBLIC Franka::Franka
 ↪  Eigen3::Eigen3)

set(EXAMPLES
    main_start_threads
)

set(SOURCES
common/source/shader.cpp
common/source/common.cpp
common/source/controls.cpp
common/source/texture.cpp
graphics/source/plot2d.cpp
graphics/source/plot3d.cpp
graphics/source/gui.cpp
src/functions/source/pid.cpp
src/functions/source/functions.cpp
src/threads/source/robot_loop_thread.cpp
src/threads/source/gui_thread.cpp
)

ADD_CUSTOM_TARGET(copy_shader_dir ALL
COMMAND ${CMAKE_COMMAND} -E copy_directory
"${CMAKE_CURRENT_SOURCE_DIR}/graphics/shaders" "${CMAKE_BINARY_DIR}"
COMMENT "copy shaders to build directory"
VERBATIM)

ADD_CUSTOM_TARGET(copy_glade_dir ALL
COMMAND ${CMAKE_COMMAND} -E copy_directory
"${CMAKE_CURRENT_SOURCE_DIR}/glade" "${CMAKE_BINARY_DIR}"
COMMENT "copy glade to build directory"
VERBATIM)

ADD_CUSTOM_TARGET(copy_objfiles_dir ALL
COMMAND ${CMAKE_COMMAND} -E copy_directory
"${CMAKE_CURRENT_SOURCE_DIR}/objfiles" "${CMAKE_BINARY_DIR}/objfiles"
COMMENT "copy objfiles to build directory"
VERBATIM)

foreach(example ${EXAMPLES})
    add_executable(${example} src/apps/${example}.cpp ${SOURCES})
  target_link_libraries(${example} OpenGL  Franka::Franka
    Threads::Threads glfw GLEW_1130 examples_common Eigen3::Eigen3
     ↪  ${GTKMM_LIBRARIES})
  ADD_DEPENDENCIES(${example} copy_shader_dir)
endforeach()
```

Appendix B

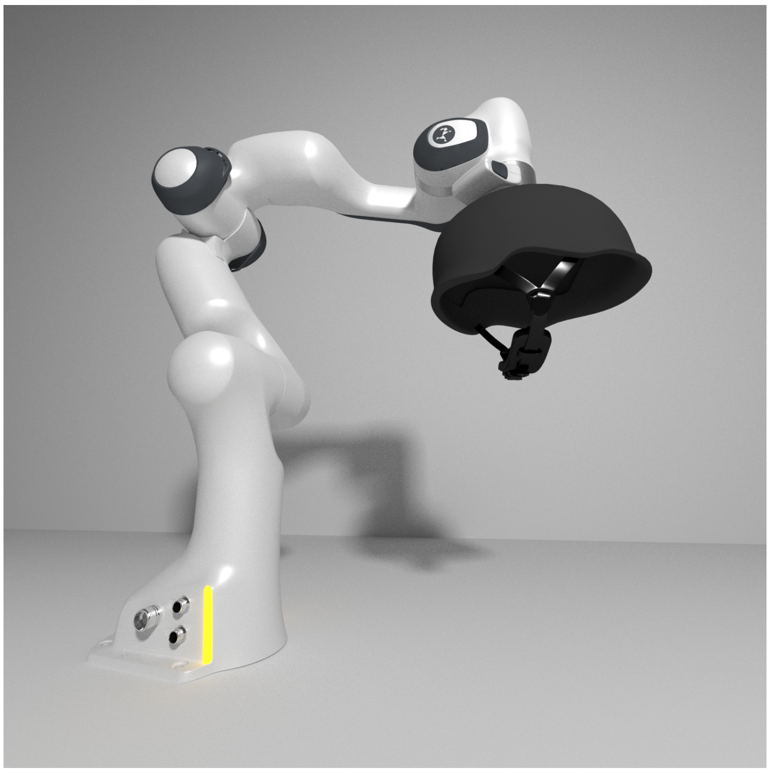
Specialization Project

NTNU

Norwegian University of
Science and Technology

Development of a rehabilitation apparatus for whiplash injured patients

Ove Baardsgaard
Hans Edvard Hafskolt



Project Thesis
Submission date 20.12.2018
Supervisor Knut Aasland

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Preface

Several persons and institutions have contributed to understand the problem both academically and practically. We would like to thank Shahrukh Khan, Eirik Njåstad, Adam Leon Kleppe for providing useful insights about the nature of the problems we intended to solve. Further more we would like to thank Aleksander Lillienskiold for providing us with both great insights about the Franka Emika API and concrete suggestions for the future implementation of force control to the Franka Emika robotic manipulator.

Finally we are grateful for Knut Aasland from the Department of Mechanical and Industrial Engineering (MTP), and Morten Leirgul from Firda Physical Medical Centre (Firda fysmed) for providing a inspiring problem, being available and assistive through the project and believing that our efforts will amount to something useful.

Summary

This is a specialization project concluded the fall of 2018 by Ove Baardsgaard and Hans Edvard Hafskolt. It is a part of an ongoing project at the Department of Mechanical and Industrial Engineering (MTP). This semester was intended to continue the development of an apparatus for rehabilitation of whiplash injured patients. We have continued on the work of several other theses where the robotic manipulator Panda by Franka Emika GmbH was chosen as a motion platform for the apparatus. To decide if the Panda is the best option for a motion platform for this project has been the main focus of this thesis. In addition, we continued the work on making a head/neck attachment to ensure good force transfer between the head and the robot.

A promising prototype for the head/neck attachment has been made with an inflatable element with two air pockets inside an alpine helmet. This provided reasonable force transfer, while being comfortable and universal. The prototype is low level, and needs further development before a final prototype.

The Panda is the most competitively priced robotic manipulator in its category, and provides a lot of functionality and sensors. We have tried to test if the robot is able to fulfill all the needed requirements as a motion platform for this apparatus. One of the questions was if the Panda is able to reach the sufficient range of motion. The reach of the Panda is limited, but we believe that it is able to cover the sufficient range of motion. Another question is if the Panda is able to record motions without restricting the patients movement. From our tests and research, we think that it is possible. But we were not able to implement the appropriate controller in this time period, thus we cannot be entirely confident.

Table of Contents

List of Tables

List of Figures

Abbreviations

PD	=	Proportional Derivative
IK	=	Inverse Kinematics
FCI	=	Franka Control Interface
MTP	=	Department of Mechanical and Industrial Engineering
DOF	=	Degrees of Freedom

Chapter 1

Introduction

The project is part of an ongoing cooperation with Firda Physical Medical Centre at Sandane in Norway, represented by physiotherapist Morten Leirgul to create a rehabilitation apparatus for whiplash injured patients. It continues on the work of Brattgjerd and Festøy (2018), where it was decided that a robotic manipulator by Franka Emika GmbH should be utilized.

1.1 Problem description

Since the overall task of the project presented to us was a bit vague: To further develop the rehabilitation apparatus for whiplash patients, we felt the need to set a more defined goal for this project. After the robotic manipulator Panda by Franka Emika GmbH was acquired, we quickly realized that there had not been done enough research to reason that this robot is actually suited for this task. Therefore we set our main goal for this project thesis to:

Investigate the possibilities of using the robotic manipulator Panda by Franka Emika GmbH as the main motion platform for a whiplash rehabilitation apparatus

As we started our research, an important problem with using the Panda was that it required too much force to guide it by hand. Therefore the main goal of the project was focused on lowering the resistance of the robot, or in other words increasing the compliance.

Before the robot was acquired, we also looked into other problems, such as:

- How to fix the patient's head inside a contraption that allows force transfer to the robot arm.
- How said contraption should be connected to the robot arm.

1.2 Background and Motivation

We entered the project with no prior knowledge about the task and were presented the notion that perhaps the next step in the project was to acquire the Panda robot. Previous work had been done: simulations, various approaches, and a feasibility study of the Panda robotic manipulator as a part in the final product. We suspect there was a misconception that the product would rapidly progress as soon as we acquired the robot. At this time we were not sufficiently experienced to question this assumption, nor did we fully understand the complexity of the task.

Our background is mixed, but mainly consists of studying mechanical engineering. Hans Edvard Hafskolt started at Engineering and ICT, then chose ICT and Mechanical Engineering for his specialization. Hans Edvard has experience with programming and user experience design. He does not have prior experiences with control engineering or robotics. Ove Baardsgaard is in the final year of a masters degree in mechanical engineering, and has programming experience from spare-time projects and work at the geomatics department of Trondheim Municipality. He has not been involved in any prior cybernetics projects, but believes that a strong motivation for this project and creativity will make this into a great learning experience.

The progress of this project during the semester has been very varying. It started out with realizing that we needed to acquire the Panda as quickly as possible. While the Panda had a delivery time of 3 weeks, we started to do some research and preparations. After the Panda arrived, we worked day by day trying to learn how the robot worked, and it was hard to make any long-term plans. This was a very unstructured way of working, but we believe it was the only way for us to progress as our lack of knowledge was large.

Chapter 2

Theory

2.1 Previous work

As there have been made 5 masters and 5 project theses on this project, and a lot of work has been done already, we needed to extract as much knowledge from those theses as possible. To get an overall view of previous work, we have summarized the most important findings:

2.1.1 Head/Neck attachment

The masters thesis by Slåttsveen and Tolo (2015) tested a lot of different solutions for how to make a contraption that is able to transfer force between the head and the motion platform. The two main solutions were a semi-flexible cap tightened by straps, and a more rigid cap still tightened by straps. Their conclusion was that it was hard to provide adequate fastening for every motion of the head.

Berg and Sunde (2016) made a new and improved solution based on a helmet with inflatable pillows to secure the head. They stated it had better force transmission from the head to the motion platform, while also being more comfortable.

Gælok and Strand (2017) tested a large range of different concepts, and landed on two main concepts: a helmet with inflatable elements, and a helmet with four rigid elements tightened to the head. The inflatable solution is similar to the solution of Berg and Sunde (2016), but with multiple air pockets to avoid the movement of air inside the inflatable element. They also noted the need for multiple helmets to reduce the distance between the helmet and the head because of the large variation of shape and size of the patient's heads.

Brattgjerd and Festøy (2018) developed the concept of inflatable pillows inside a helmet further, and came up with some promising concepts. The inflatable pillow needs to have a slot for the ear for comfort, and it needs to cover the forehead, back, and temple. They concluded that the design needs to be developed further, and a testing rig is needed to find an optimal design.

2.1.2 Connection between motion platform and head/neck attachment

Slåttsveen and Tolo (2015) presented a solution for detaching the head/neck attachment from the motion platform based on a mounting mechanism from a surfboard. They mentioned that this solution should be further explored.

Berg and Sunde (2016) needed to make a new connection for their concept, and tested different solutions. They concluded that a permanent connection was best suited for their motion platform, while also eliminating a time-consuming step for the physiotherapist. Since Gælok and Strand (2017) stated a need for multiple helmets, they mentioned that the connection between the helmet and motion platform needs to be quick and easy to handle, while being rigid to avoid looseness.

A quick and rigid connection mechanism was the basis for the work of Brattgjerd (2017) where he made a prototype of the mechanism in his project thesis as seen in Figure 2.1. After we tested the mechanism, it seemed to have severe problems with looseness in planar rotation and lacked the rigidity needed for the application. In addition it was heavy(755g) and quite tall(30mm), so it is obvious that further development is needed.

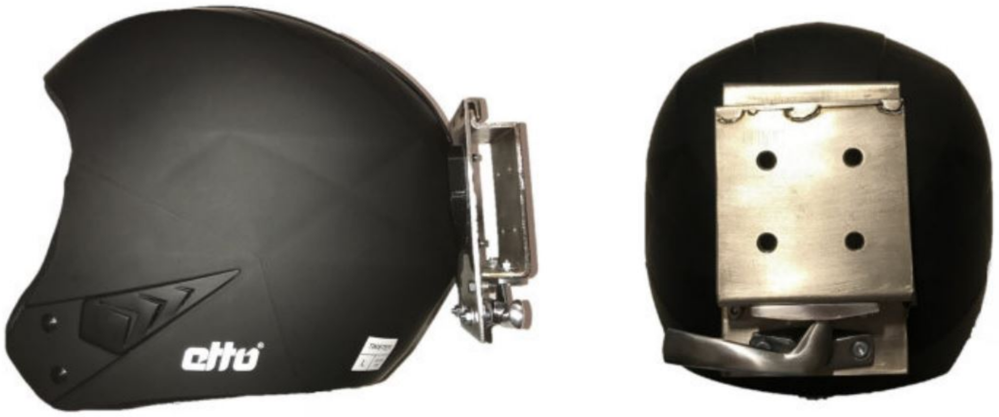


Figure 2.1: Mounting mechanism prototype(Brattgjerd, 2017)

2.1.3 Motion Platform

The masters thesis of Slåttsveen and Tolo (2015) explored various motion platforms, and recommended the use of a Stewart Platform as it has a large work space and a great force-control ability. The masters thesis of Berg and Sunde (2016) came to the conclusion that the Stewart platform was not suitable as motion platform as it was unable to cover the necessary motion space. By exploring various concepts they came up with a new motion platform called MASNAK. They concluded that the MASNAK was able to fully cover the motion space, and that it was a promising solution.

After Gælok and Strand (2017) investigated the previous theses, they found flaws in both the Stewart platform concept and the MASNAK concept that made it necessary to find another motion platform to be able to fulfill the requirements for the product. Therefore they started exploring new concepts, such as a cable-driven parallel robot (Gælok and Strand, 2017). After some research, they found that this type of robot is too demanding to make. Meanwhile they discovered the robotic arm, Panda by Franka Emika GmbH, which was likely to be much more suited for this task. During a mail conversation with Dr. Simon Haddadin, CEO of Franka Emika GmbH they became certain that the resistance of guiding the robot can be minimized to almost no resistance. The only possible limitations seemed to be the low payload of the robot.

Brattgjerd (2017) did physical tests with the Panda during a trip to Franka Emika GmbH, and concluded that the ”results were promising”, but mentions that further examination is needed. Brattgjerd and Festøy (2018) executed these further examinations by simulating the Panda with a Matlab toolbox (Corke, 2018). Their conclusion was that ”Panda fulfills most of the requirements for the motion platform”(Brattgjerd and Festøy, 2018, p. 51). The payload seemed to not be of an issue, but the motion space was not fulfilled as the patients was not able to rotate their head backwards because of the robot self-colliding. They stated that a solution to this could be to place the robot in a different position, but this was not tested.

2.2 Neck and whiplash

To understand the background and motivation for this project, we give you a short introduction to how the neck works, and what whiplash is.

The neck consists of 7 vertebrae that allows it to cover a great range of motions. In addition, the vertebrae has to support the weight of the head and protect the spinal cord. The two topmost vertebrae called Atlas and Axis, contributes most to allow the great range of neck motions, and is also the most vulnerable joints in the human body (LFN, 2005).

The rotation of the head can be divided into three main motions as shown in Figure 2.2. An article from NASA (2000) provides data about the range of neck motion that the apparatus should be able to handle. Table 2.1 shows the range of motion that the subjects' necks was able to reach in the motions shown in Figure 2.2. It is worth noting that this is a simplified version of the neck's movement. As the neck is able to move in a large range of positions and rotations, we only focus on the extreme values because they are a limiting factor in this project.

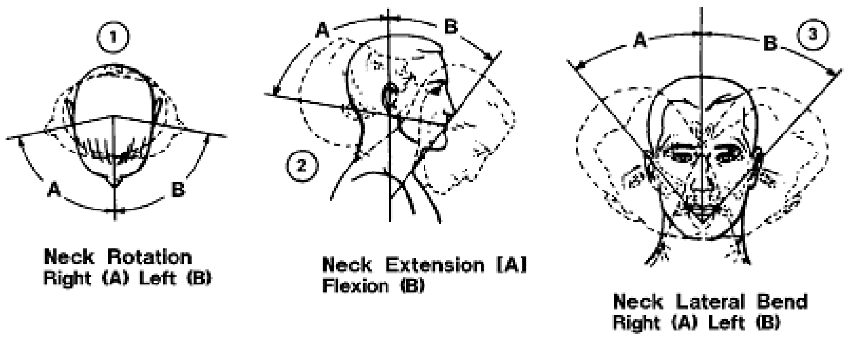


Figure 2.2: Neck motions(NASA, 2000)

Joint movement	1A/B	2A	2B	3A/B
Range [°]	109	103	84.4	77.2

Table 2.1: Neck range(NASA, 2000)

Whiplash is an injury mechanism that can be described as a sudden acceleration of the head, followed by a sudden deceleration (Louw, 2018). This often caused by car accidents. During this movement, large forces are applied to the neck which can cause a variety of damage in the neck. The neck's ligaments can be stretched and deformed plastically, or muscles, nerves and joints can get damaged (Barnsley et al., 1994). This makes the outcome of the injury very individual, which arises the need for individually adapted recovery programs.

2.3 Robotic manipulators

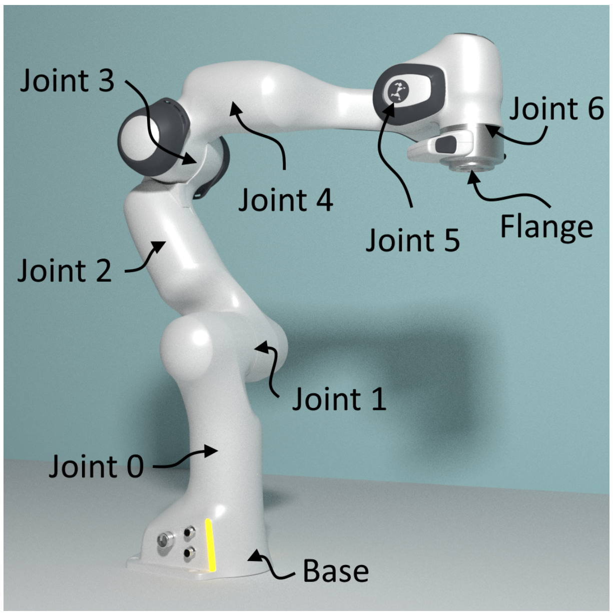


Figure 2.3: Naming of the different parts of the Panda

There are many variants of robotic manipulators. A robotic arm with 6 degrees of freedom (DOF) is an approach that will allow an end-effector (tool at the end of the robot arm) to move to any pose and orientation. Among the benefits of implementing a robot arm are the availability of reliable, finished products that perform precisely with good repeatability. While recording movements by guiding the robot, repeatability is the robot's ability to reproduce the exact same movement as recorded measured by the deviation. If the apparatus should rely on the internal sensors of the robotic manipulator, good repeatability is important. Morten Leirgul stated that deviation should not be larger than a millimeter when the apparatus recreates a recorded movement. Available robotic manipulators often specify the highest measured end-effector deviation as reapeatability. Franka Emika specifies a repeatability of +/- 0.1 mm (ref), which we assume is sufficient.

In most applications the robot produces a movement commanded by a computer. The challenge in this project is to let the patient produce the movement and the robotic manipulator to follow the movement without inhibiting the patient. This requires a control system for the robot. Force control, admittance and compliance are notable keywords for this approach. Furthermore the robotic manipulator should be able to produce adjustable resistance against the patient's movement.

A new generation of collaborative robots are made to work safely alongside humans. These robots, referred to as collaborative robots, or ”cobots”, have built in safety features; features such as integrated sensors, passive compliance or sensing high current in the motors. This way the robot will sense external forces and stop if the force exceeds a set threshold. This means that you can stop the robot with your hands. These features are not necessary for all approaches to control the robot, but might be deemed necessary for the safety of the patient and physiotherapist.

The Panda was chosen as a motion platform for this project. While studying the previous work, the benefits of choosing the Panda were not obvious to us and thus we wish to include some considerations about the Panda compared to other robotic manipulators with adequate reach and payload. In the previous work by Brattgjerd and Festøy (2018) the requirements specified the need for 6 degrees of freedom. Due to our findings further described in Experiments and Motion Control we adjust this requirement to 7 DOF's. In theory a 6 DOF mechanical system is sufficient, however with a robotic arm the the joints turn in different planes according to the pose and will frequently loose one or more DOF's. When the robot arm looses DOF's and gets less than 5 DOF's, it reaches a kinematic singularity which means it is not able to produce every desired movement anymore.

The Panda has 7 joints, which yields more than the minimal number required to complete a set of tasks. This means that the end-effector can reach any position and rotation with one redundant DOF, and these robotic manipulators are commonly referred to as redundant manipulators. The extra degree of freedom is exhibited as a joint velocity that does not contribute to the velocity of the end-effector, and adopting redundancy helps avoiding kinematic singularities.

In practical terms, reaching a singularity while running the robot means that the robot cannot continue moving towards the desired pose, no matter how it moves its joints. The inability to move or rotate the end-effector in certain ways is considered a lowered manipulability. On the other hand, the extra joint makes the inverse kinematic problem have infinite solutions. Redundancy can be conveniently exploited to maintain manipulability or to meet additional constraints (Grøtte, 2018, p. 8).

The Panda has a payload of 3 kg, 800mm maximum reach, +/- 0.1 mm repeatably and torque sensors in each joint. At a price point quite low compared to similar robots we could argue these specifications to be the minimum requirements for this apparatus. Among other viable options are the Rethink Robotics Sawyer and KUKA LBR, they produce the same repeatablility and sensoring, but they are in a different price range (ROBOTIC IQ, 2017). Most other available robotic manipulators do not offer 7 DOFs or torque sensoring.

2.4 Motion Planning

To specify the movement of a robotic manipulator such that the end-effector accomplishes a set task is known as motion planning. Forward kinematics makes use of joint parameters to compute the pose of the end-effector. By pose we mean the orientation and position of the end-effector. Inverse Kinematics (IK) uses kinematic equations to find suitable joint parameters to let the end-effector reach the desired pose.

Singularities are also a challenge in the mathematical representation of robotic manipulators. One should verify that singularities in the representation are handled correctly. The end-effector rotates around three axes and there are three common ways of representing rotation: Euler angles, quaternions and rotation matrices.

Euler angles is a minimal representation of relative orientation and perhaps the most intuitive of the three. Euler angles introduce a problem with representing angles around 90 degrees, commonly referred to as gimbal lock (CHRobotics LLC, 2018). This problem is solvable implementing extra parameters and checks.

Rotation matrices, (3x3) matrices rely on storing 9 scalar values, and are more computationally expensive.

Unit quaternions, quaternions with absolute value 1 is a third way of representing orientation, sometimes referred to as a compromise between rotation matrices and Euler angles in terms of advantaged and disadvantages (CHRobotics LLC, 2018). If you consider a complex number to be a two-dimensional number, a quaternion can be considered as a three-dimensional number represented by four values and multiplying quaternions is less computationally expensive than multiplying rotation matrices. To read the robot orientation and avoid singularities we would suggest using unit quaternions to represent the orientation.

2.4.1 PD controller

To make the robot track a reference point, a PD controller can be used. This is also suggested by Khan (2018). A PD controller is a virtual spring and damper system, often used in control systems (Kelly and Saund, 2018). The desired reference point may change quickly, faster than the robot is able to move. The PD controller can control the robot such that it tries to move to the reference point as fast as possible. Also, before the robot reaches that point, it has to slow down to avoid passing the point, called overshooting. In our case we could want the robot to keep a constant force. The PD controller will then calculate the error between the measured force and the desired force, and control the robot to minimize that error.

2.5 Interviews

Throughout the project we conducted many interviews. First to understand the the problem and the requirements for an adequate solution. Then to find theoretical and practical approaches to solve the problem.

2.5.1 Visit to Firda Physical Medical centre in Sandane

Tuesday 18th of September we met with Physiotherapist Morten Leirgul. The visit to Sandane was intended to motivate us, and gather knowledge about the treatment of whiplash injured patients. Morten showed us typical treatments to strengthen and engage patient movement, and explained what might be the causes of the patient's symptoms.

He stated the long term intentions for the project: An easy to use apparatus for rehabilitation, that can precisely reproduce exercises and provide adjustable resistance. On of the most challenging aspects of the rehabilitation is to engage the desired muscle response from the patient. The specific response is necessary strengthen the patient to perform certain movements that the injuries inhibits. To achieve this the apparatus should provide support while the patient exercises, and adjust the resistance as the patient's rehabilitation progresses. The patient will do exercises together with Morten, and then the apparatus should be able to reproduce the exercises without supervision.

The apparatus could have the ability to measure the performance of the patient and provide useful data, but most importantly the apparatus would be an improvement to existing equipment (Khan, 2018, p. 3) and enable patients to perform the required rehabilitation training.

We talked about our educational backgrounds before discussing what could be the possible approaches to the project. Morten wanted the apparatus to move accordingly to the patients head movement and record the trajectory. The apparatus should then let the patient recreate the same exact movement precisely so the head follows the same trajectory. Morten wanted to know if we thought the Panda could produce low resistance force in the trajectory while inhibiting movements that deviate from the trajectory. We told him that our experience with robotics and controllers were very limited, but from our backgrounds studying mechanical engineering and watching demonstrations of the Panda we could reason about the possibilities. We did not know how to make the Panda behave the way Morten desired but we thought it would be possible.

Morten asked us about the the Stewart Platform as he stated he liked some of the previous work. Ee agreed upon that the Stewart Platform seemed promising at did not know of any disadvantages other than a slightly restricted rotation. Morten then stated that Firda Fysmed would value a very good solution rather than a faster solution. One of the desired qualities was that the end product should be highly automated and require minimal adjustments from the patient and physiotherapist. And if the a robotic arm like the Panda could support a larger range of motion than a Stewart Platform that could be a better solution.

2.5.2 Interview with Eirik Njåstad, PhD Candidate at MTP, NTNU

Eirik Njåstad works with industrial robots for welding and machining applications. At our first meeting with Eirik we wanted to know more about programming robot arms, the safety aspect of attaching a robot arm to the human head and how we familiarize ourselves with the practical and theoretical aspects of moving a robot arm with external force applied to the end-effector.

We learned that even though the popularity and demand of colaborative robots is increasing, human-robot interaction was not a familiar topic at this department of the university. Eirik was not specifically familiar with the Panda as they were first released in late 2017, but theorized an approach to control the position of the robot using a 6-axial force/torque sensor mounted between the robot flange and the end-effector.

2.5.3 Testing the Panda with Morten Leirgul

Morten Leirgul at Firda Fysmed wanted to test the robot as soon as we had it running. Monday morning the 5th of November we met in the Realization Laboratory at MTP, NTNU. We used the proprietary hand guiding of the Panda and Morten tried manipulating the end-effector pose with his head in the mounted helmet. We received three immediate responses:

- The end-effector does not move quite where I want it to move.
- The minimum force required to change the position is too large.
- The force required varies both in magnitude and direction, and this does not comply with the behavior needed to engage neck movement from a patient.

We had no knowledge about how the compliance of the proprietary hand guiding worked, or whether it was possible to lower the resistance of the hand guiding. Information and documentation to this hand guiding is not accessible according to Franka.

At this stage we could not know whether it was possible to guide the end-effector in a way suited to the needs of Firda Fysmed. Until this point our intention was to track the patient's movement and letting the patient reproduce the movement while the robotic manipulator provided virtual walls to help the patient exercise the correct movement. One of the most important aspects of Firda Fysmed's rehabilitation of whiplash injuries is engaging movements that the patient struggles with. Realizing that the resistance of the end-effector, varying in both direction and magnitude would rather inhibit the desired response from the patient we needed to change focus.

We decided to focus on making a apparatus capable of moving with the patient with equal resistance in all directions and equal resistance in all rotations. Morten underlined the importance of this as the first step towards developing a useful product.

2.5.4 Adam Leon Kleppe, Head Engineer at MTP

We consulted Adam Leon Kleppe about approaches to control robotic manipulators. Initially we talked about inverse kinematics and PD controllers, then together we tested the Panda. Adam helped us reason about the proprietary hand guiding mode of the Panda. He reasoned that to increase compliance we needed to compute the necessary torque in each joint, and that we could find this through inverse kinematics. Then that we might go at it with a different approach considering the torque measurements from each of the joints. Adam suggested that the solution to our goal was to create out own mass-spring-damper system and calculate appropriate values for the stiffness matrix.

2.5.5 Visiting Aleksander Lillienskiold, Researcher at SINTEF Ocean

Monday the 3rd of December we met with Aleksander Lillienskiold at SINTEF Ocean. They have solved various tasks related to fishing and processing of fish, as well as other automation projects. They were early adopters of the Panda and have been developing controllers for robotic manipulators. Aleksander made great efforts to explain their applications, experiences and code for the Panda and proposed that we could make use of and contribute to the same code base pending on a non disclosure agreement. Controlling robotic manipulators while considering the forces applied to the end-effector is of great interest to SINTEF Ocean, and cooperation could be mutually beneficial. Aleksander introduced us to some of the people researching similar topics and helped us understand our problem and the libraries for the Panda. In their experience the Panda should serve very well as a chosen platform for the whiplash rehabilitation apparatus. The Panda seemed robust, the programmable interface is quite open and the high communication rate of the provides solid grounds for developing a well performing apparatus.

Aleksander suggested that mounting a small 6-axis force/torque sensor between the flange and the end-effector would ensure complete control over the input parameters. And that we should pursue this to create a force control system for the Panda, before we made any further reasoning of whether applying a force/torque sensor is necessary. That would provide data to determine if the internal torque sensors of the Panda will suffice in the intended use of the whiplash apparatus.

Chapter 3

Experiment

3.1 Positioning of the Panda

The previous work by Brattgjerd and Festøy (2018) showed that the Panda was unable to cover the motion space from the position on the current prototype seen on Figure 3.1.

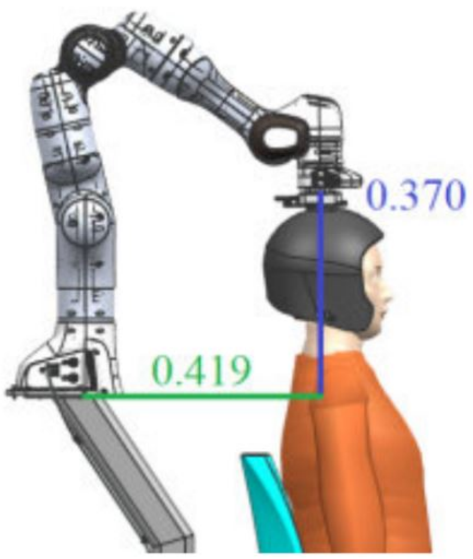


Figure 3.1: Position of the Panda as suggested by Brattgjerd and Festøy (2018)

Finding an optimal position for the Panda is not an easy problem, and we chose to address the problem iteratively by using a cardboard model since the robot was not yet attained. A cardboard model was built using the parameters provided by Franka seen in figure 3.2. This allowed us to test different positions rapidly.

To test how well the Panda would be able to cover the motion space from the various positions, we mounted an iPhone on the helmet with the app Angle Meter(nakhon phagdeechat, 2018) that shows rotation in three axis. The angle from the initial position to eight extreme positions was measured and noted. The eight positions were repeated two times, and an average

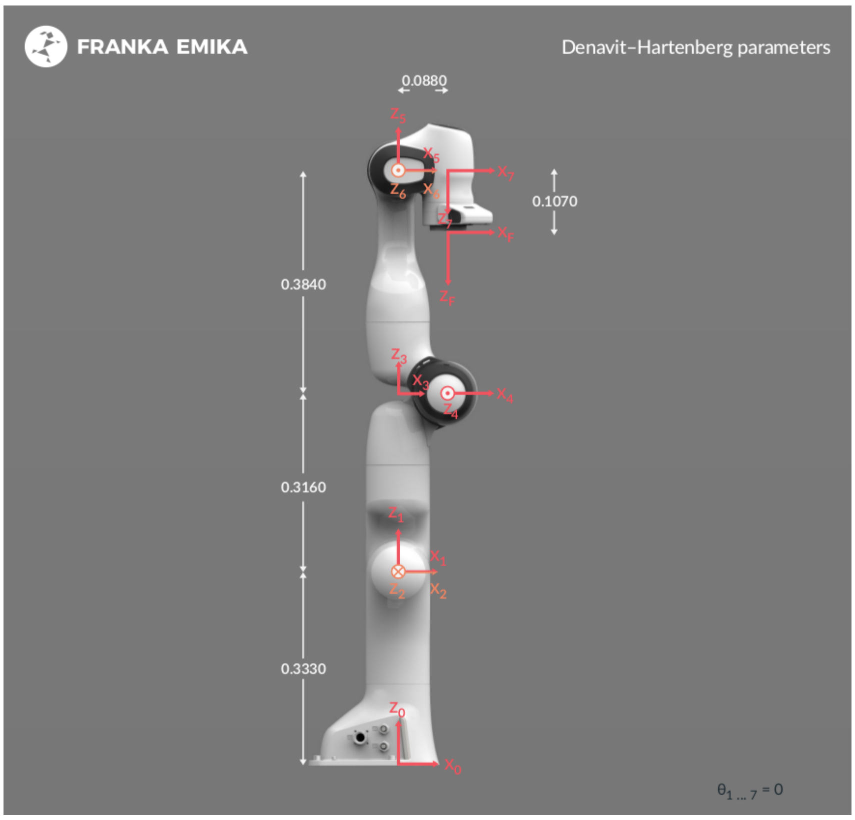


Figure 3.2: Parameters of the Panda (Franka Emika GmbH, 2017d)

value was calculated from the results. To test the desired motion space, a test without the model attached was done as shown in Figure 3.3.

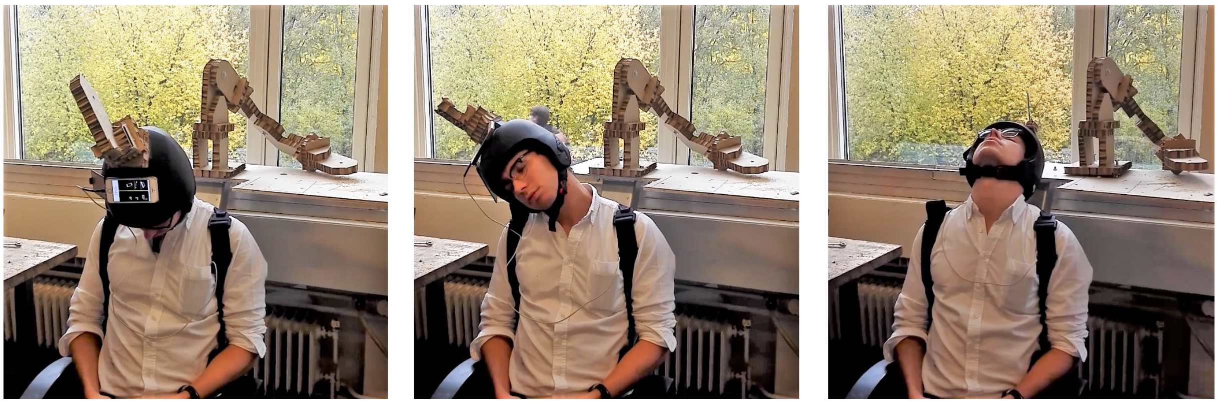


Figure 3.3: Three of the extreme positions of the head tested without the model attached

Then we needed to test the motion space of the robot in the original position. This was done in the same way as described above, but with the cardboard model attached to the helmet as seen in Figure 3.4. It was clear that the robot was self colliding when rotating the head backwards.

Our initial guess was then to position the model with an offset to one of the sides, because this would solve the problem of the robot self-colliding when rotating the head backwards. After moving the cardboard model to a new position offset 130mm to the side and 270mm forward,

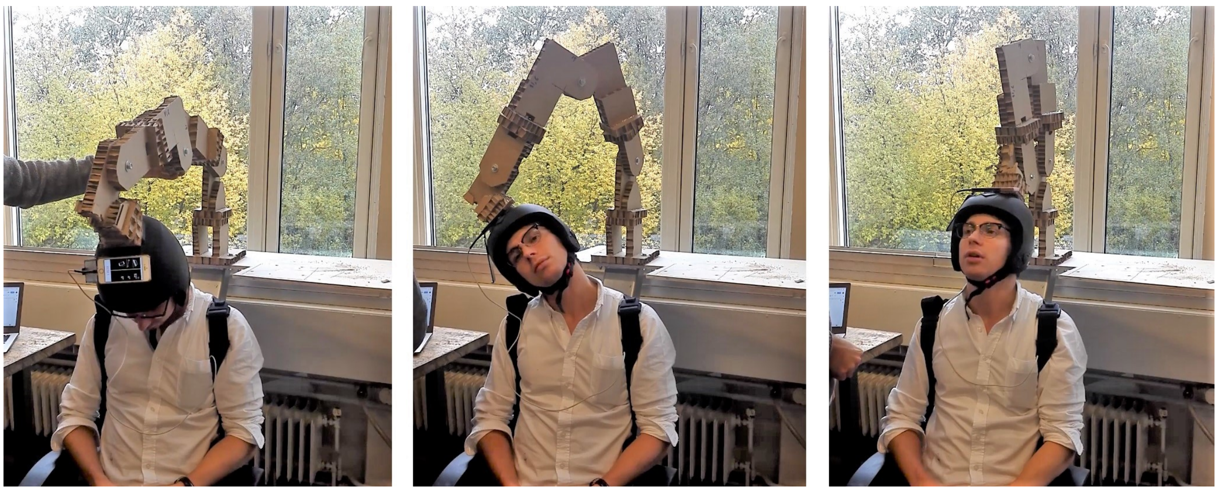


Figure 3.4: Three of the extreme positions of the head tested with the model attached in the original position

position 1, we did the test again as shown in Figure 3.5. The model was fully able to handle the backwards bending, but self-collided when the head was rotated towards the model. Due to this, another position was tested, position 2, offset 150mm to the side and 315mm forward. Then the model did not self collide, but had some minor troubles reaching the opposite side.

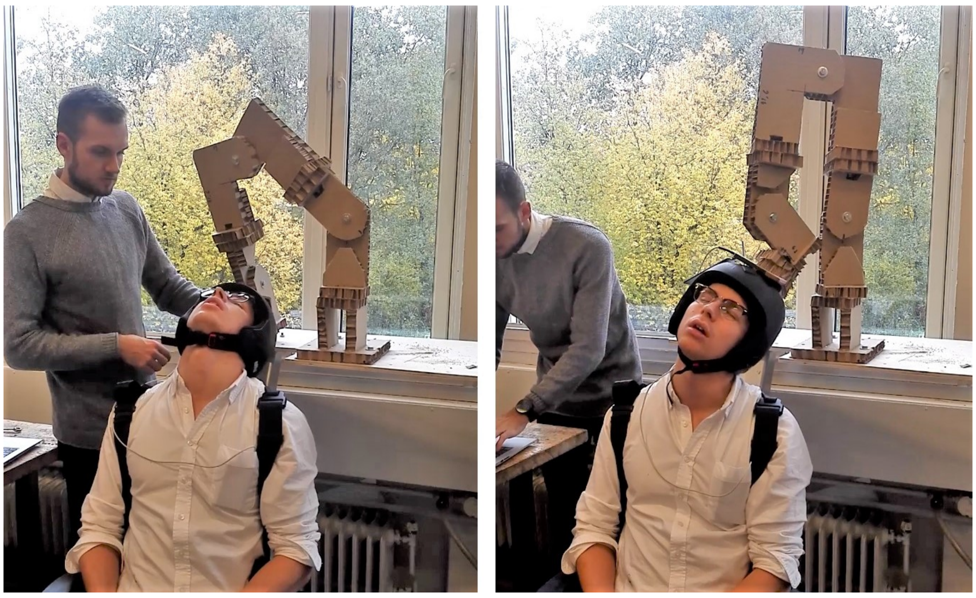


Figure 3.5: Two of the extreme positions of the head tested with the model attached in position 1

The results from these tests were plotted in a diagram (Figure 3.6). It is clear that none of the positions were able to fully cover the desired motion space, though a compromise between position 1 and 2 could yield sufficient results.

During the interview with Aleksander Lillienskiold, another idea of positioning the robot was brought up. By placing it much lower, joint 5 could move ”over” joint 1 when performing the backward bending, instead of colliding with it. Since testing this in practice would need ma-

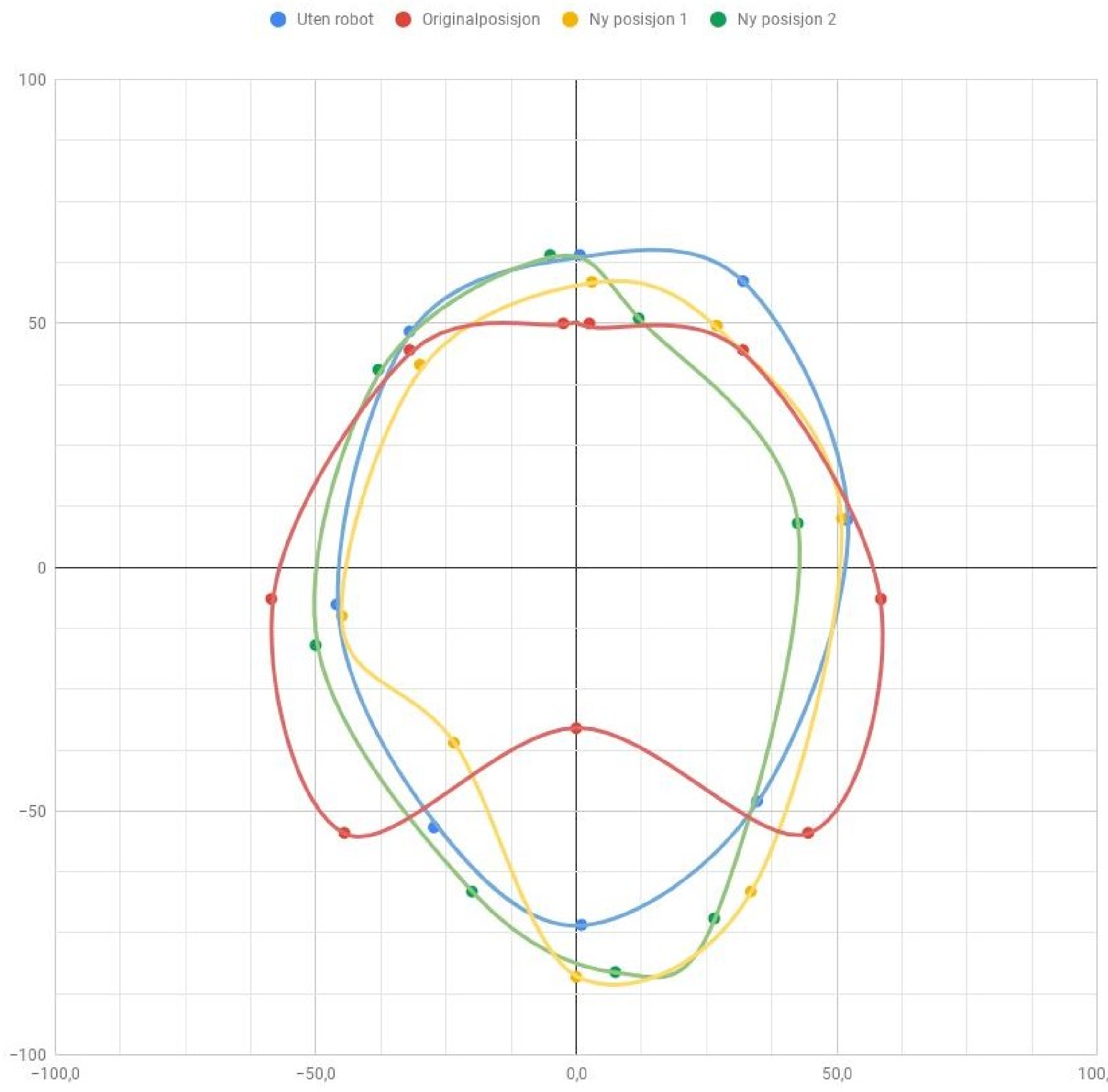
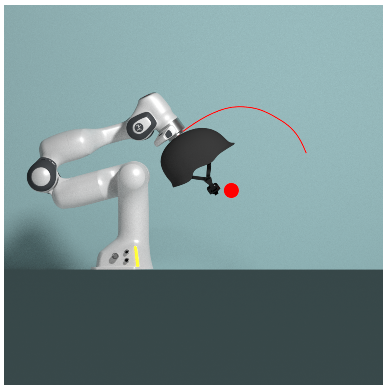
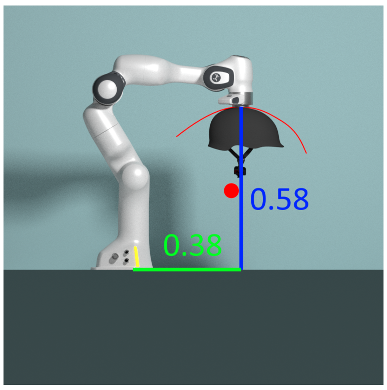


Figure 3.6: Plot of the results from the position test with smooth interpolated lines to illustrate the motion space. X-axis: rotation about y-axis [°]. Y-axis: rotation about x-axis [°].

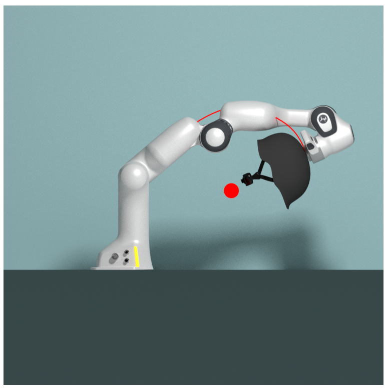
jor changes of the chair mount, we tried simulating it using Blender 3D (Blender Foundation, 2018), an animation program we were familiar with. The simulation is not completely accurate, but it is a quick and easy tool to use. The tracker paths from Brattgjerd and Festøy (2018) performing forward and backward bending were transferred into Blender. The robot was simulated to follow the path using Blender's internal IK solver, much like how the Panda works. After testing and refining the position of the path, we felt like we found the only position where the robot actually would manage the complete motion. As seen in Figure 3.7, the robot is very close to its limits in both ends of the path.



(a) Backward bend



(b) Initial position with measurements [m]



(c) Forward bend

Figure 3.7: Suggested position of the robot to manage extreme positions. Path from Tracker is shown by red line.

3.2 Inflatable element for head/neck attachment

From the work of Brattgjerd and Festøy (2018), a promising solution for the head/neck attachment was made. Using an alpine helmet with an inflatable element and a hand pump, they had a solution which was comfortable, sturdy and universal. There were still some improvements left to address, and since the actual prototype was now unattainable, we had to develop a new prototype. As there were no drawings of the prototype with some scale or measurements in the previous thesis, we needed to develop a new prototype.

To figure out a suitable shape for the inflatable element we used a technique well known in the costume making industry using aluminium foil and tape to make a template of the head (KamuiCosplay, 2017). From the experience noted by Brattgjerd and Festøy (2018) we marked out a design on the template we thought was suitable, with the element covering both the forehead, back of the head and avoiding the ears, as seen in Figure 3.8. The template was cut out and transferred to a plastic sheet from an inflatable mattress made of soft polyvinyl chloride (PVC) as recommended by Brattgjerd and Festøy (2018). Since the element changes shape when inflated, we added a 10mm buffer around the template. We later discovered that this made the element too large, and could have been neglected.

Our first design made a good, even pressure around the head, but through testing it was apparent that the air was moving inside the element which made the force transfer very poor. Gælok and Strand (2017) suggested the use of multiple air pockets to avoid this issue. This was tested by Brattgjerd and Festøy (2018), which did not notice any improvement using two air pockets instead of one. We tested this by making a new weld, and connecting another tube as shown in Figure 3.9. A scaled outline of the element is shown in the appendix, Figure 4.1. The tubes were connected to the same pump, but needed to be clamped individually. We believe small electric solenoid valves could be a good solution to close the connections. Our opinion is that this reduced the slack significantly. Since this solution is more complex than having one air pocket, this needs to be further tested.

To seal the inflatable element we heated the front and the back sheet, welding it together,

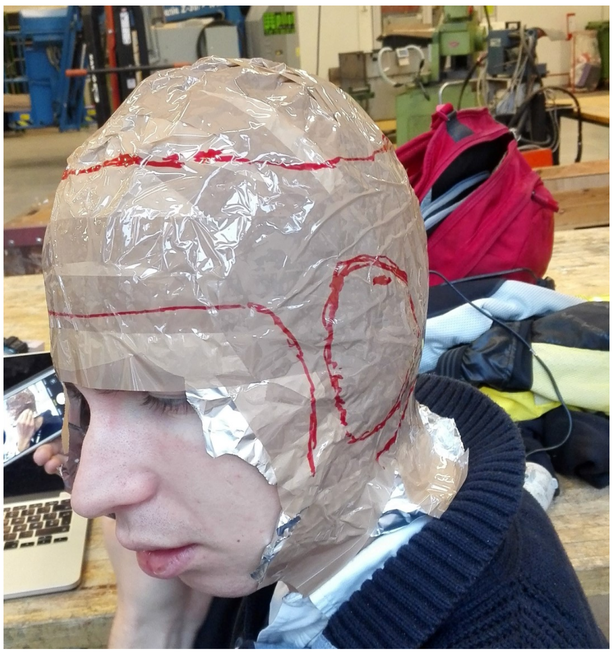


Figure 3.8: Template with markings for the boundary of the inflatable element

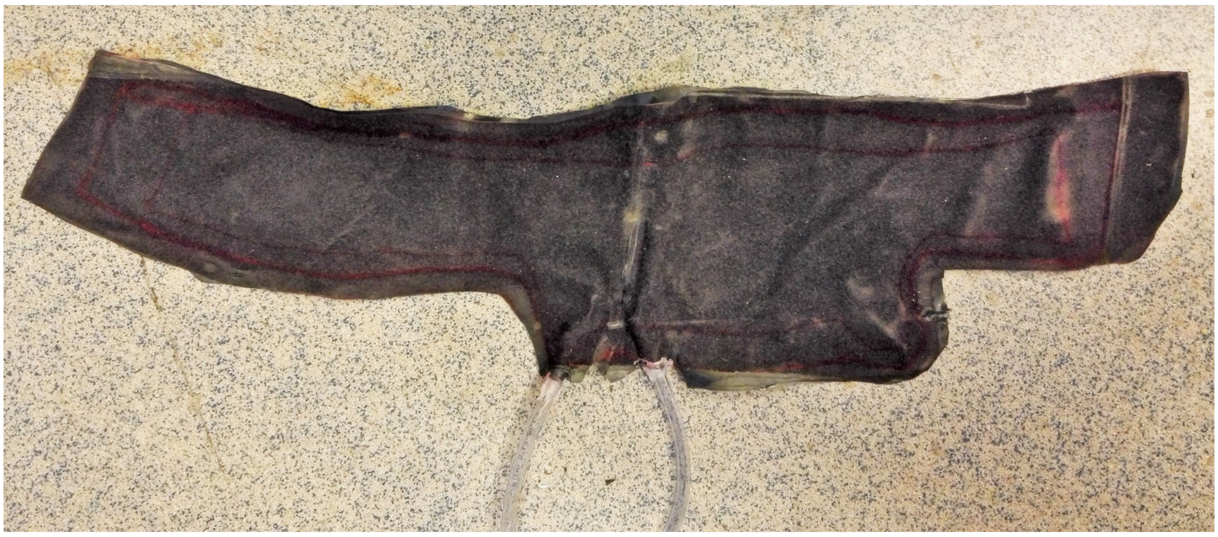


Figure 3.9: Inflatable element with two air pockets and two tubes.

using a household iron. This took some testing before it became airtight. Using the highest heat setting on the iron, medium pressure of about 10kg, for approximately 10 seconds made a good weld. Plastic tubes were hot glued into the inflatable element and connected to an air pump, but it was hard to create a good seal. The bag was inflated under water to locate any leaks.

Brattgjerd and Festøy (2018) used a bulb pump to inflate their elements, and mentions that ”an automatic inflation system seems excessive”. As we had an electric air pump from the inflatable mattress, we used that in our design. An electric air pump is not an expensive element and easy to implement, we think that it should be chosen to minimize the work required by the physiotherapist using the machine.

Brattgjerd and Festøy (2018) raised attention to the pressure zones on the forehead, temple, and back of the head. During testing, we saw that the alpine helmet did not cover the whole area of the forehead and the temple, which lead to lost support from the inflatable element.

3.3 Connection between motion platform and head/neck attachment

During testing with the connection mechanism made by Brattgjerd (2017) mounted to the Panda, we spotted some problems. To minimize the needed motion space, the flange of the Panda have to be as close to the axis of rotation in the neck as possible. In addition, the force on the flange decreases when the distance increases if the torque from the neck is constant. To reduce the resistance from the robot, the force needs to be as high as possible. The connection mechanism adds a height of 30mm, and the mounting bracket used to connect it to the helmet also adds 14mm. In addition, the helmet contains a padding with a thickness of 20mm. Including an estimate of the distance from the neck to the top of the head of 230mm (Neuhaus, 2014), and the last joint length of the robot of 107mm, the total lever for the neck to the robot is 401mm. By reducing the distance from the flange to the head to 5mm, the lever becomes 342mm, which means that the torque could be reduced by 15%.

Gælok and Strand (2017) stated a need for multiple helmets, and thereby a quick mounting connection to the helmet. The development of the inflatable elements have moved forward since then, and the need for multiple helmets is no longer pertinent. In addition to needing the flange to be mounted closer to the head, a permanent mount of the helmet should be considered. One disadvantage could be that it might be tricky for patients with disabilities in neck movement to put on the helmet easily while the robot is attached. Our suggestion is to postpone the development of this element, and continue when the Panda is ready to be tested with real patients.

3.4 Controlling the Panda

3.4.1 Set-up

The Panda required some set-up before we could use it and run custom C++ code on it. We followed the set-up guide in the manual that came with the robot, and used a computer with Linux Mint to communicate with the robot. To start controlling the robot, we used Desk (Franka Emika GmbH, 2018b), an easy to use programming interface made by Franka Emika for the Panda robot. It quickly became apparent that this interface was not sufficient for our purpose. To control the robot more advanced, libfranka (Franka Emika GmbH, 2017c) needed to be installed on the computer. This was done following the steps on the web page for libfranka (Franka Emika GmbH, 2017b).

Since we have mounted a helmet instead of the original gripper on the flange, some parameters in the robot needs to be changed. The robot compensates for gravity, and therefore needs to know the mass, rotational inertia, and center of mass of the helmet. These can be set in the Desk interface, under ”settings $\rightarrow$ end-effector”. We set these parameters by trial and error, but further work should try to measure these parameters to get the correct values.

To control the robot using libfranka, all that was needed was to open the terminal in Linux, browse to the build directory of libfranka and run an example code using a command like this:

```
examples/echo_robot_state 172.16.0.2
```

To run a new C++ file, we added a .cpp file to the examples folder. Then the `CMakeList.txt` needed to be edited, and the filename had to be added under `set(EXAMPLES`. If any libraries needed to be linked, for example ”Thread”(cplusplus.com, 2017) to enable multi-threading, the line `target_link_libraries(filename Threads::Threads)` had to be added. Then we ran the command `cmake --build .` to build the code, and ran the code using the same command as previously mentioned.

3.4.2 Libfranka library

A lot of time has been used to understand the functions of libfranka. They are documented online, but the documentation lacks detail and is not straightforward to understand. Therefore we will summarize our experience with the relevant functions of libfranka.

Libfranka contains four main interfaces for communicating with the robot (Franka Emika GmbH, 2017c):

- **Non-realtime commands.** These are for setting parameters of the robot. For example impedance values such as `setCartesianImpedance()` which makes the robot yield if someone hits it. Also, `setCollisionBehaviour()` sets the threshold for collision detection which makes the robot stop if someone hits it too hard.
- **Realtime commands.** This is where you can run a control loop which loops with a frequency of 1kHz, which makes you able to control the robot in real time.
- **Robot state.** This reads the current state of the robot, such as joint and flange position, forces or torques on the flange and all joints. This can be read with 1kHz frequency, so you can use it in the realtime control loop.
- **Model library.** Contains a model of the robot with forward kinematics of all joints, body and zero jacobian matrices of the joint, and also the centrifugal, coriolis, and gravity vector, and the inertia matrix.
- **Errors.** The robot will stop and send an error if you command it to do something out of its limits. The limits are well described on the web page of libfranka (Franka Emika GmbH, 2017d). This is very helpful to avoid the robot damaging itself, and thereby making it easy to test less robust code.

3.4.3 Realtime commands

There are two ways of controlling the robot in realtime. Either by controlling the torque in the joints, or by providing a motion (position or velocity).

Controlling the robot in realtime is done with `franka::Robot.control()`. The input is either torques, motions or both. The inputs needs to be defined as callback loops using libfranka classes. There are various classes, depending on how you want to control the robot.

`franka::CartesianPose` is one of the classes you can use to make a callback loop. It positions the robot's flange in the cartesian space relative to the base. To use it you need to return an array with 16 values, which corresponds to a 4x4 position matrix. A 4x4 matrix can represent both rotation and translation, in addition to multiple other transformations (Baker, 2017).

3.4.4 Writing output data to file

Writing and printing data to the console should be done with multi-threading to avoid delay in the 1kHz realtime loop. The example in libfranka called `joint_impedance_control.cpp` (Franka Emika GmbH, 2018a) contains a piece of code that shows how to print data from the robot in realtime using multi-threading. The example starts a thread using `std::thread` (cplusplus.com, 2017). To prevent different threads from accessing data at the same time, `std::mutex` is used.

3.4.5 Robot state

A useful function for getting the position of the flange relative to the base frame is `O_T_EE()`. It outputs a 4x4 position array like mentioned earlier. This is a useful base for moving the robot in the cartesian space. One can read the function into an array called pose: `std::array<double, 16> pose = robot_state.O_T_EE` And then by changing a value of the array, for example: `pose[14] += 0.1`, one can return the pose to `franka::CartesianPose` and move the flange in the z-direction 0.1m. However, this will not work as the control loop is run 1000 times per second, and moving 0.1m in 1/1000 of a second exceeds the acceleration and velocity limits of the robot. This could be solved with a PD controller.

To read the forces and torques acting on the flange, there are two main functions:

- `K_F_ext_hat` returns the forces and torques acting on the flange as seen from an axis cross on the tip of the end-effector. The output is an array with values `[Fx, Fy, Fz, Tx, Ty, Tz]` with units N and Nm.
- `O_F_ext_hat` returns the forces and torques acting on the flange as seen from an axis cross on the base of the robot. The output is an array as above.

The axis for the functions can be seen in Figure 3.10, which was determined after some testing. When the robot is in this position, and a force is acted in the Z-axis on the flange, `K_F_ext_hat()` will only read a value in Fz, while `O_F_ext_hat()` will read a value in both Fz and Tx, since the force is acting on a distance from the origin (see Figure 4.2 in the appendix).

Franka comments that there is no force/torque sensor in the flange itself, and the values of the

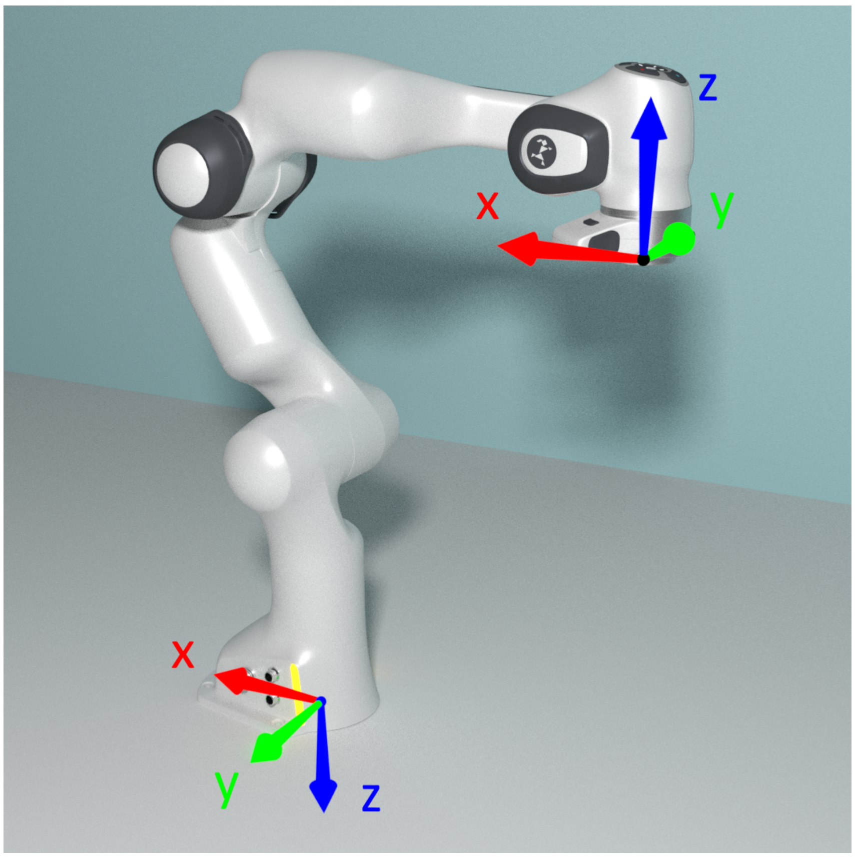


Figure 3.10: Axis for the force/torque outputs at the flange and base

two functions above are simply calculated from the measured torque in each joint (Cristian Kodura, Franka Emika, 2018). This can lead to some issues, by example if the joints are extended as shown in Figure 3.11. Since the equation for calculating force from torque is $F = T/d$, the force reaches infinity when the distance d approaches zero. In addition, this means that the calculated force would be less accurate in such positions. To avoid values approaching infinity, the output of `K_F_ext_hat()` and `O_F_ext_hat()` is set to only zeroes by libfranka when the joints are in such configurations.

3.4.6 How to increase the compliance

To be able to use the Panda as a motion platform for the whiplash rehabilitation apparatus, it was undoubtedly a need to increase the compliance of the robot. That means to reduce the required force needed to move the robot by hand. Libfranka provides an interface for controlling the torque of each joint, where gravity is compensated by the robot. By setting these to 0, it means that the joints senses the torque and tries to move accordingly to keep the torque levels at 0. This can easily be done by writing a controller, and sending zeroes to all joints with the `franka::Torques` class as seen in Listing 3.1.

By running this code, we tried rotating individual joints and could feel a small resistance. We decided to write the output of the torque sensor and the position in joint 0 to a file, and plot the results. As seen in Figure 3.12, the angular position of the joint is updated much later than we started applying torque. This could be caused by either a delay in the system, or that the

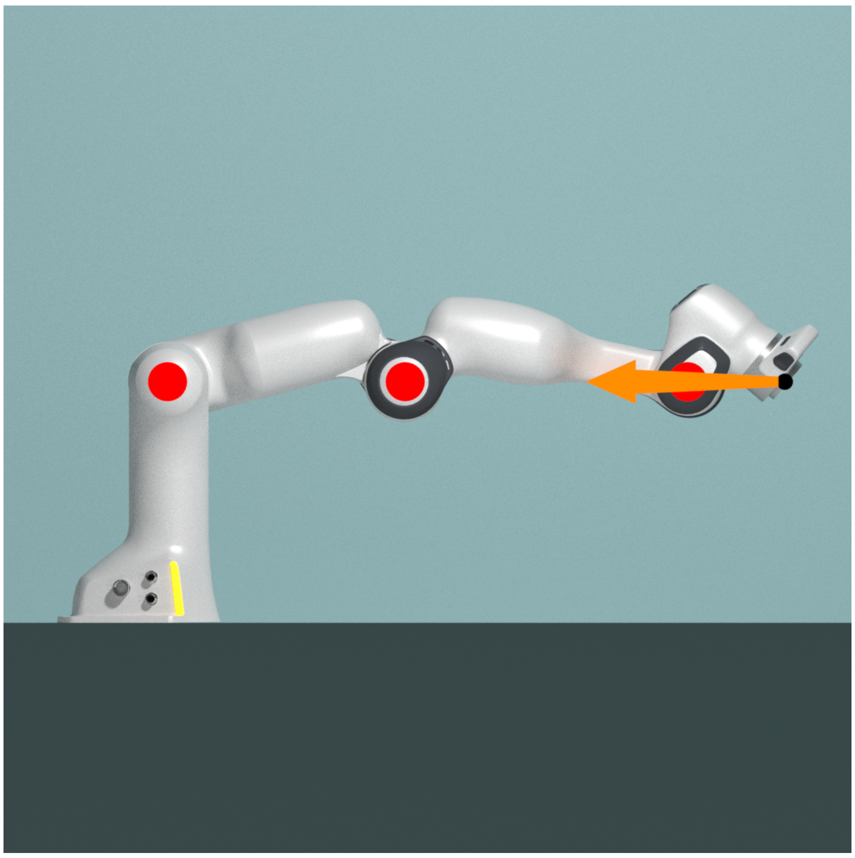


Figure 3.11: The Panda positioned with joints aligned with a force (orange arrow)

```
robot.control([&](const franka::RobotState&, franka::Duration)
    -> franka::Torques {
  return {{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}};
});
```

Listing 3.1: (Franka Emika GmbH, 2017c)

torque needs to surpass a certain threshold before the robot is commanded to move. As we don't have access to the source code of libfranka, we cannot see how the `franka::Torques` class works, or change any parameters. To be able to tweak these parameters, it may be necessary to write a custom controller which works like the `franka::Torques` does. Franka also confirms that this could be necessary Cristian Kodura, Franka Emika (2018). Our guess is that it reads the torque input from a joint, runs it through a PD controller to make it move in a controlled fashion, and sends the needed motion to the joint.

Another option to control the robot is to use the output of `K_F_ext_hat()`, which gives us force and torque in any cartesian direction at the flange. This force could then be sent through a PD controller as above, to a cartesian pose motion generator where we can input the desired next cartesian position. The cartesian pose motion generator in libfranka calculates the IK to move the flange of the robot to the desired position. The difference from the previous option is that this will more likely be equally compliant in all directions.

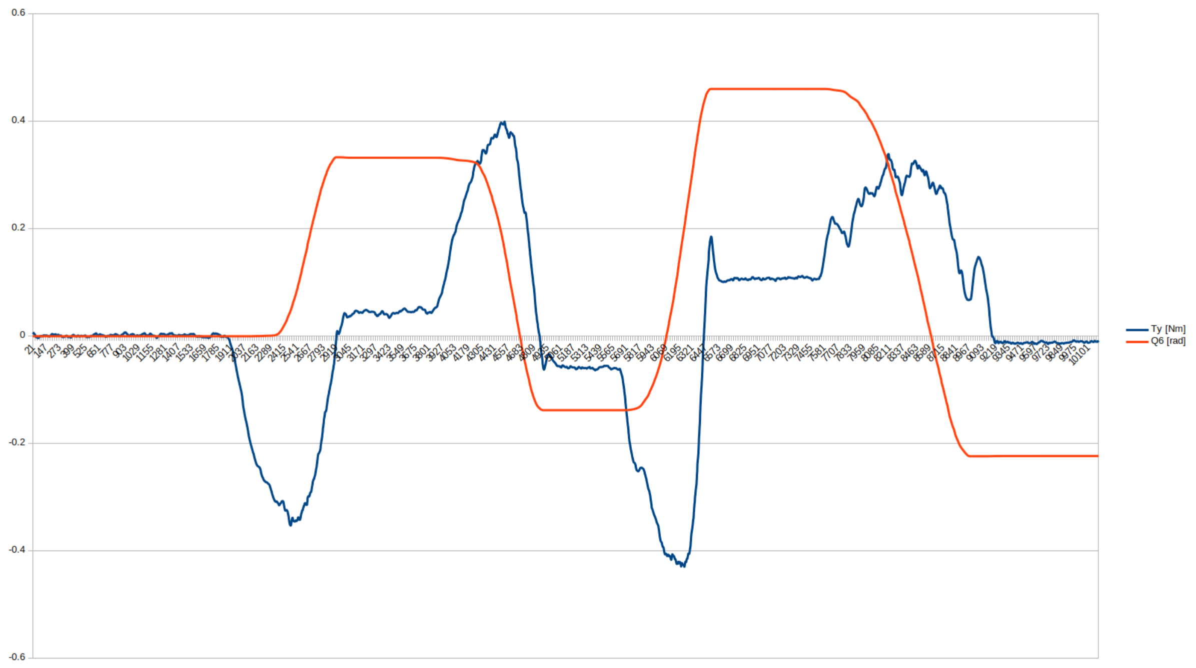


Figure 3.12: Plot of torque(blue line) and angular position(red line) in joint 6 while rotating the joint by hand

Franka states that the internal torque sensors are ”super sensitive” (Cristian Kodura, Franka Emika, 2018), but if tests show that the sensors are not accurate or sensitive enough, another option would be to mount an external force/torque sensor to the flange. The data from this sensor could be fed to the motion generator of libfranka. Eirik Njåstad stated that in this case, the robot could be steered with a feather, or as one of the providers force/torque sensors, ATI Industrial Automation states: ”Can resolve down to 0.149 gram-force” (ATI Industrial Automation, 2018).

3.4.7 Experimenting with mass-spring-damper system

With the suggested approach from the interview with Adam Leon Kleppe we wanted to make a mass-spring-damper system for the Panda. To understand how, we tested the example code from libfranka called `cartesian_impedance_control.cpp`. Impedance control is a means to control the mechanical impedance of a robotic manipulator. That is how the end-effector resists motion when subjected to force. This behavior can described by a set of mass-spring equations, further described in the previous work by Khan (2018). The code specifies the compliance parameters as translational stiffness, rotational stiffness and damping. Then it reads the initial position and define it as the equilibrium position of the system. By removing the error distance from the callback loop you will have a compliant control, rather than impedance with a fixed equilibrium point.

We ran the code while lowering the stiffness parameters, and tested the Panda. The code did produce the dynamic behaviour we wanted, but the end-effector was still resisting to much compared to the requirements Morten Leirgul stated. Eventually we removed all stiffness and damping values initialized in the code, still the Panda felt too stiff. We tried writing the matrices by hand, and could not see how any stiffness was still affecting the mass-spring-damper equations, the only value left in the callback for the torque control loop was from the `franka::Coriolis` function.

The experiments with the cartesian impedance control example from Libfranka did not bring us any closer to the behavior the whiplash rehabilitation apparatus required and left us confused. Later we realized that the all the values from the impedance control callback function is passed on to the Panda as joint-level torque commands. The Franka Emika github documentation of `franka::Torques` (Franka Emika GmbH, 2017a) is not rich, and thus we have had to reason about it from our own findings. There is a new torque instance for every time step by the impedance control callback function. We pass it an array with 7 doubles, each representing the desired torque in a joint. These joint-level torques are without gravity and friction, and the Panda moves to achieve the desired torque levels specified by the last call to Torques. `franka::Torques` relies on some sort of PD controller that we do not have access to, however we can test it.

To conclude our experiment with the `cartesian_impedance_control.cpp` we could say that the mass-spring-damper system is built on top of the compliant control system of `franka::Torques` and thus changing the stiffness and damping parameters has no effect on the the mechanical resistance experienced when running the Franka Emika with zero torques as seen in Listing 3.1.

Chapter 4

Future Work

During this semester, a lot of ideas have come up from our experimentation. We were not able to test all of these, so we have listed the ideas and experiences for future work.

4.1 Positioning of the Panda

The position of the Panda has been discussed, and still needs some further research. The things that needs to be addressed is: Covering the whole motion space, and need for individual adjustment.

Since there is large variations in human height, there may be need for individual adjustments to allow the robot to work in its optimal work space. The previous work by Brattgjerd and Festøy (2018) addressed this issue by making the seat height adjustable. We think it might be better to adjust the position of the robot instead of the seat, because there is much less weight involved in addition to being less intimidating for the patient. We also think that the robot should be able to position itself after the patient has been seated, to automate more of the treatment. The robot could drive linear actuators to position its base at the height where the arm is in its optimal position.

Our suggestion is to build a testing rig that can vary both the height of the robot, and the distance to the patient. Then test a large range of positions to find the most optimal placement. Afterwards test that the position with people of different heights to see if height adjustment is necessary. A more robust simulation software such as Visual Components (Visual Components, 2018) could also be used in this process.

4.2 Designing a test for the human neck

Whiplash injuries are debated and researchers do not agree about what causes the symptoms. Testing the neck performance might lead to interesting results. Firda Fysmed stated that a finished whiplash rehabilitation apparatus should also serve as a precise instrument to test the patient's performance and mobility for furthering the research on whiplash injuries. If the apparatus should serve this purpose, the measurements of the apparatus should be compared to a set of verifiable results.

We therefore recommend designing a testing rig designed for accurate testing of the neck performance. There could be performed controlled tests of the human neck, both in terms of mobility and performance. The test should be designed to limit the effect of other variables. From our experiments our own neck mobility produces varying results depending both the number of tests that day, and events in the days leading up to the test. With a certain number of test subjects and a highly specified test design, the results could be very useful for many applications.

4.3 Head/neck attachment

We believe that the current solution with an inflatable element inside a helmet is a promising solution for the head/neck attachment, and just needs further refinement. We have some suggestions for further work based on our experience.

The helmet could benefit of a redesign to cover the forehead and the temple better. In addition, the current helmet contains a thick Styrofoam padding which is not needed for this purpose. By making the helmet a simple plastic shell, it's also possible to reduce the distance from the flange to the head. Berg and Sunde (2016) described the concept of vacuum forming plastic as a method to create the helmet shell, which we believe can be a suitable method to create a new prototype.

The inflatable element needs a redesign. We believe that the current design with two elements is a good solution, but the new element needs to also cover the temple. It may be better to design the template from the helmet instead of a head, to make it fit better inside the helmet. As mentioned there is no need to accommodate for the thickness of the inflatable element when drawing out the design. To fasten the tubes better, one could make the inlet of the element like a short pipe where the tubes can be sealed in with pipe clamps.

4.4 Programming the robot

Programming the Panda is the next main step in this project. A lot of work is left to be done, and making the robot more compliant is the first step. One could start with trying to make the individual joints react to a lower torque input, and see if that solves the problem. If one finds out that the sensors are not sensitive enough, an external sensor should be attained. To fit Firda Fysmed's requirements the motion should be equal in all directions. If the Panda does not behave accordingly, one should try to control the robot with cartesian poses/velocities instead.

Succeeding the first problem, the challenge is to record the patient's movement, and then guide patient through said movement. Recording the data from the `O_T_EE()` function for example, leads to a large amount of data that represents an uneven trajectory. A simplification algorithm could be necessary to reduce the amount of data, and make a smoother trajectory. Then the robot needs a controller to appropriately guide the patient's head through same trajectory.

Bibliography

ATI Industrial Automation, 2018. F/t sensor: Nano17 titanium. https://www.ati-ia.com/products/ft/ft_models.aspx?id=Nano17+Titanium, Accessed: 2018-12-20.

Baker, M. J., 2017. Maths - using 4x4 matrix to represent rotation and translation. http://www.euclideanspace.com/maths/geometry/affine/matrix4x4/index.htm, Accessed: 2018-12-19.

Barnsley, L., Lord, S., Bogduk, N., 1994. Whiplash injury. Pain 58 (3), 283 – 307.
URL http://www.sciencedirect.com/science/article/pii/0304395994901236

Berg, O. J., Sunde, Ø. K., 2016. Utvikling av apparat for behandling av nakkeskadde. Master's thesis, NTNU.

Blender Foundation, 2018. Blender 3d. $https://www.blender.org/$, Accessed: 2018-12-19.

Brattgjerd, S. K., 2017. Development of a new machine for rehabilitation of whiplash patients. Master's thesis, NTNU.

Brattgjerd, S. K., Festøy, A. M., 2018. Development of rehabilitation apparatus for whiplash patients. Master's thesis, NTNU.

CHRobotics LLC, 2018. Understanding euler angles. http://www.chrobotics.com/library/understanding-euler-angles, Accessed: 2018-12-20.

Corke, P., 2018. Robotics toolbox. http://petercorke.com/wordpress/toolboxes/robotics-toolbox?fbclid=IwAR0NzIlbtFEuLGkBG8yUtxufTVcIDLbG4jnf5-_G2XwaJhF2JVk_h61mQB4, Accessed: 2018-12-20.

cplusplus.com, 2017. std::thread. http://www.cplusplus.com/reference/thread/thread/, Accessed: 2018-12-19.

Cristian Kodura, Franka Emika, 2018. private mail conversation.

Franka Emika GmbH, 2017a. franka::torques class reference. https://frankaemika.github.io/libfranka/classfranka_1_1Torques.html#a409e1c7e30054ccaba18a5590bb3b9d5, Accessed: 2018-12-20.

Franka Emika GmbH, 2017b. Installation instructions. https://frankaemika.github.io/docs/installation.html, Accessed: 2018-12-16.

Franka Emika GmbH, 2017c. libfranka. https://frankaemika.github.io/docs/libfranka.html, Accessed: 2018-12-17.

Franka Emika GmbH, 2017d. Robot and interface specifications. https://frankaemika.github.io/docs/control_parameters.html, Accessed: 2018-12-15.

Franka Emika GmbH, 2018a. joint_impedance_control.cpp. https://github.com/frankaemika/libfranka/blob/master/examples/joint_impedance_control.cpp, Accessed: 2018-12-19.

Franka Emika GmbH, 2018b. Panda. https://www.franka.de/panda, Accessed: 2018-12-19.

Gælok, T. E. L., Strand, M., 2017. Utvikling av maskin for opptrening av nakkeslengskadde. Master's thesis, NTNU.

Grøtte, H., 2018. A virtual security net and soft motion controller for a 7-dof redundant cooperative robotic manipulator. Master's thesis, NTNU.

KamuiCosplay, 2017. Full eva foam armor - medion erazer girl. https://www.youtube.com/watch?v=NovgPB4UAXU, Accessed: 2018-12-15.

Kelly, M., Saund, B., 2018. Pd controller. http://www.matthewpeterkelly.com/tutorials/pdControl/index.html, Accessed: 2018-12-19.

Khan, S., 2018. Development of a compliant control system for rehabilitation of whiplash injuries. Master's thesis, NTNU.

LFN, 2005. Nakkens anatomi. http://www.lfn.no/Pdf/HLRL_Kap12_NakkensAnatomi.pdf, Accessed: 2018-12-15.

Louw, A., 2018. 71 - whiplash injury: Treatment and rehabilitation. In: Giangarra, C. E., Manske, R. C. (Eds.), Clinical Orthopaedic Rehabilitation: a Team Approach (Fourth Edition), fourth edition Edition. Content Repository Only!, Philadelphia, pp. 479 – 486.e1.
URL http://www.sciencedirect.com/science/article/pii/B9780323393706000718

nakhon phagdeechat, 2018. Angle meter free! https://itunes.apple.com/us/app/angle-meter-free/id422843391.

NASA, 2000. 3 anthropometry and biomechanics. https://msis.jsc.nasa.gov/sections/section03.htm, Accessed: 2018-12-13.

Neuhaus, J., 2014. Dimensions of the human figure. https://www.jneuhaus.com/human.html, Accessed: 2018-12-20.

ROBOTIC IQ, 2017. Collaborative robots buyer's guide. https://blog.robotiq.com/collaborative-robot-ebook, Accessed: 2018-12-20.

Slåttsveen, K. B., Tolo, S. F., 2015. First development of new machine for rehabilitation of whiplash patients. Master's thesis, NTNU.

Visual Components, 2018. Visual components homepage. https://www.visualcomponents.com/ , Accessed: 2018-12-20.

Appendix

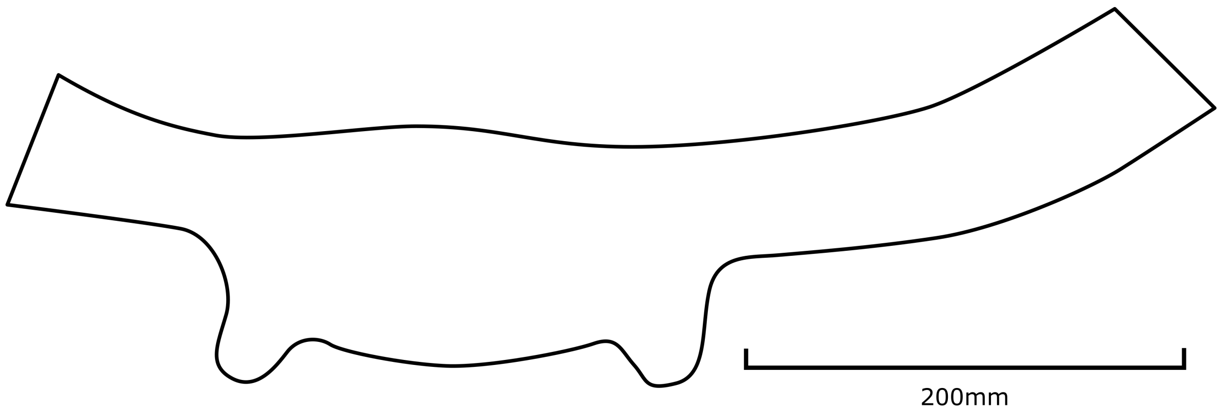


Figure 4.1: Outline of the inflatable element

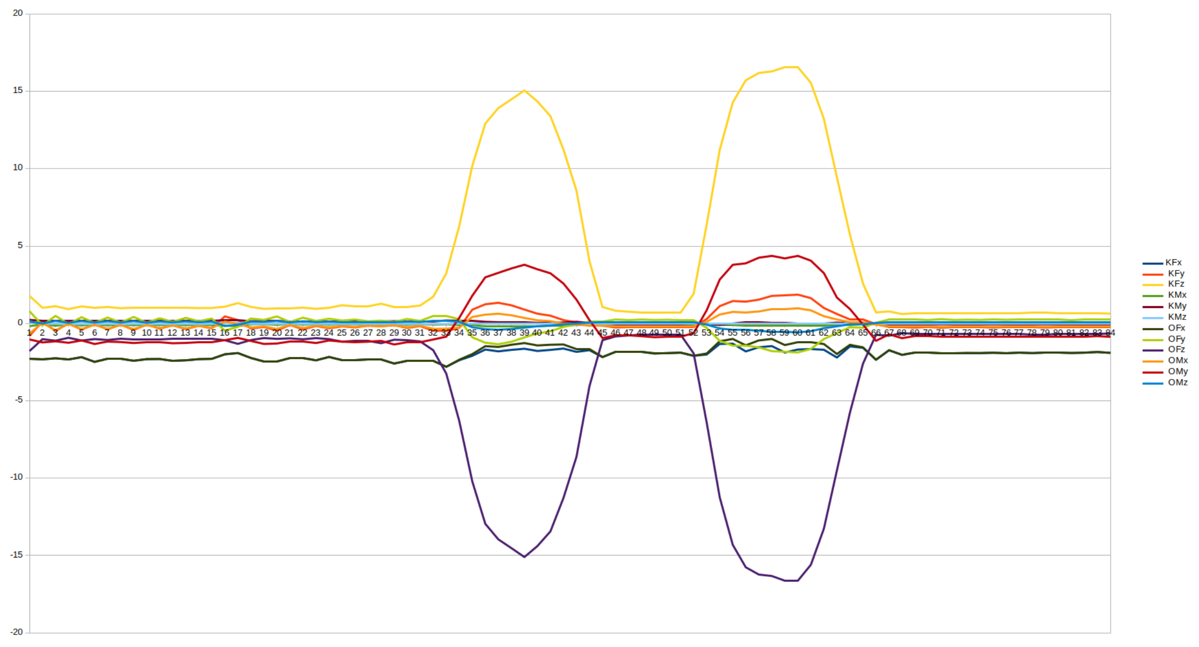


Figure 4.2: Plot of all forces and torques from the output of K_F_ext_hat and O_F_ext_hat, while pushing on the flange upwards two times.

x	Uten robot	Originalposisjon	Ny posisjon 1	Ny posisjon 2
0,7	64			
32,0	59			
52,0	10			
34,7	48			
1,0	73			
27,3	53			
46,0	8			
32,0	48			
2,5		50		
32		44,5		
58,5		6,5		
44,5		54,5		
0		33		
2,5		50		
32		44,5		
58,5		6,5		
44,5		54,5		
3			58,5	
27			49,5	
51			10	
33,5			66,5	
0			84	
23,5			36	
45			10	
30			41,5	
5				64
12				51
42,5				9
26,5				72
8				83
20				66,5
50				16
38				40,5

Table 4.1: Values from testing of the different positions of the Panda

Appendix C

Risk Assessment

RISIKOANALYSE (alternativ til bruk av RiskManager)

Enhet/Institutt:	MTP	**Dato opprettet:**	15.02.2019
Ansvarlig linjeleder (navn):	Agnes Digranes	**Sist revidert:**	15.02.2019
Ansvarlig for aktiviteten som risikovurderes (navn):	Ove Baardsgaard		
Deltakere (navn):	Kia Brekke		

Beskrivelse av den aktuelle aktiviteten, området mv.:

Utvikle en prototype på verksted av treningsapparat for rehabilitering av nakkeslengskadde, i forbindelse med masteroppgave våren 2019.

Aktivitet/arbeidsoppgave	**Mulig uønsket hendelse**	**Eksisterende risikoreduserende tiltak**	**Vurdering av sannsynlighet (S)**	**Vurdering av konsekvens (K)** *Vurder en konsevenskategori om gangen. Menneske skal alltid vurderes.*				**Risikoverdi (S x K)**	**Forslag til forebyggende og/eller korrigendene tiltak** *Prioriter tiltak som kan forhindre at hendelsen inntreffer (sannsynlighetsreduserende tiltak) foran skjerpet beredskap (konsekvensreduserende tiltak)*	**Restrisiko etter tiltak (S x K)**
			(1-5)	Menneske (1-5)	Øk/materiell (1-5)	Ytre miljø (1-5)	Omdømme (1-5)			
Saging	Øyeskade p.g.a. sprut av slipestøv/partikler	Vernebriller er alltid tilgjengelig i verkstedet	3	3				9	Før oppstart: Gjennomføre dokumentert HMS-opplæring med studentene (bruk av håndverktøy og pålagt verneutstyr)	3 (S = 1)
Sveising	Brannskade	Bruke vernehansker	3	3				9	Gjennomføre opplæring	3
Flytte tunge objekter	Klemskade, ryggskader	Kran/truck tilgjengelig	2	3				6	Bruke vernesko ved arbeid med tunge objekter	4
Håndtering av skarpe objekter	Kuttskader	Vernehansker er tilgjengelig	3	3				9	Avgrade skarpe kanter	3
Bruk av maskiner	Ødelagt utstyr	Personell tilgjengelig	2		4			8	Gjennomføre opplæring, spørre personell ved usikkerhet	4
Sveising	Gassforgiftning	Avtrekk tilgjengelig	3	3				9	Skru på avtrekk, gjennomføre opplæring	3
Styring av robot	Ukontrollerte bevegelser, slag mot personer	Nødstoppknapp lett tilgjengelig, stopp innprogrammert ved kontakt	2	3	3			6	Passe på stoppknapp ved bruk, ikke skru av sikkerhetsfunksjonene	3

5/6-19 Kia Brekke

5/6-19 Ove Baardsgaard

5/6-19 [illegible]

Ove Baardsgaard, Kia Brekke

Master's thesis 2019