NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

INSTITUTT FOR MASKINTEKNIKK OG PRODUKSJON

Development of a Robot Based Rehabilitation Apparatus for Whiplash Injured Patients

Master Thesis

Tobias Fløtre
Halvor Romundstad

Supervisors: Knut Einar Aasland & Amund Skavhaug

June 2021

NTNU

Preface

Development of a Robot Based Rehabilitation Apparatus for Whiplash Injured Patients has been a master's thesis collaboration between Tobias Fløtre and Halvor Romundstad during the spring of 2021.

This is the 7th master thesis that has been conducted on the matter, and it has been written for the Department of Mechanical and Industrial Engineering(MTP) at NTNU. The project began in 2014 by a request from Firda Physical Medicine Center to investigate the possibility of developing a new and improved apparatus for rehabilitation of whiplash injured patients.

We would like to thank the instigator at Firda Physical Medicine Center, Morten Leirgul, for sharing his insight in the workings of the human neck and the motivation behind this project. We would also like to thank our supervisors Amund Skavhaug and Knut Einar Aasland for providing an interesting topic and helpful tips and guidance along the way.

Summary

This thesis is part of an ongoing projects in Collaboration with Firda Fysikalsk-Medisinske senter in Sandane, Sogn og Fjordane. The long-term goal of the project is to develop a training apparatus that can be used in the rehabilitation of patients who have suffered a whiplash injury. Our project is a continuation of the work performed by Ove Baardsgaard and Kia Brekke who developed a head mount, which have later disappeared. They also started the development of a graphical user interface.

The focus of this master thesis has been two main parts. Developing a graphical user interface and creating a new head mount.

A new head mount consisting of a helmet, an inflatable element and a magnetic connection has been developed. The design was focused on making a universal helmet that could fit any patient. However, it turned out that the use of a universal helmet resulted in problems when it comes to keeping the head still, as the air in the inflatable element had too much space to move around. The magnetic connection part of the head mount was not only used to connect the helmet to the robot, but also as an important safety feature in case something happens to the robot.

A new graphical user interface have been created that is separated from the robot control program. The reasons behind this was discussed in specialization project, but in essence it was done to improve the user experience and to improve performance in relation to real-time constraints. The new user interface have functionality for 3D visualization of exercises, both in real-time and a preview for existing exercises. Functionality for creating training programs have been added, and a database have been implemented to store the exercises and programs for each individual patient. A login system was added, and network communication between the graphical user interface and the robot control program.

Contents

List of Figures

1. Introduction

This master's thesis is part of an ongoing cooperation with Firda Physical Medical Centre at Sandane, Norway. Morten Leirgul is the instigator. The end goal of the project create a suitable rehabilitation apparatus for patients with whiplash injuries. This thesis is a continuation of 6 previous master's theses and a specialization project of the authors, Tobias and Halvor. In the specialization project the main objective was to get to know how the current control system worked and to start the development of a new graphical user interface.

1.1 Background and motivation

Whiplash is an common injury worldwide and it is related to accidents where the head and neck gets thrown rapidly forwards and backwards, like in a car crash. Dizziness, nausea and headache are common sufferings from people that have suffered a whiplash injury. Firda Fysmed are currently treating whiplash injured patients with an apparatus called Multi Cervical Unit (MCU) in combination with the physiotherapist holding their hand against the patients head while the patient pushes in different directions. Figure 1.1 shows how the MCU looks.

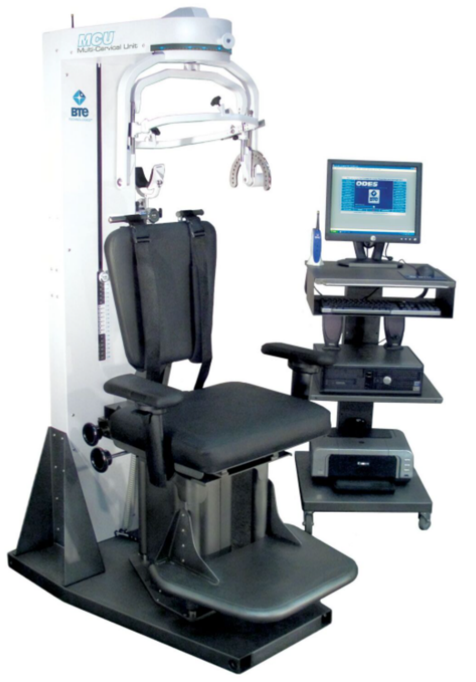


Figure 1.1: The Multi Cervical Unit. [1]

The MCU is an apparatus where the patient can perform three different motions. Rotating their head, move their head forwards and backwards or moving their heads side to side. Combining multiple movements simultaneously is not possible. These movements around one static axis does not resemble a natural neck movement, as a natural neck movement would result in the axes translating from their original positions.

Having a graphical user interface to monitor exercises while they are performed can be very helpful, not only for the physiotherapist, but also for the patient as it can allow them to view their movement while they are doing it. The graphical user interface should also have the option to create user profiles for both patient and physiotherapist. The profiles for each patient can be used to save unique training programs designed for the individual patient.

The patients treated with the MCU at Firda Fysmed are able to obtain good results, however, Firda Fysmed does not recommend acquiring an MCU to other clinics due to its limited possibilities. The end goal of this ongoing project is to develop an apparatus which can outperform the MCU when it comes to rehabilitating whiplash injured patients.

Many previous students have contributed to get the rehabilitation apparatus to the current state. From building a chair structure which can hold the robotic arm safely, to creating a head mount to attach the patient to the robot and starting the development of a graphical user interface and a compliant control system for the robot. Unfortunately, all the parts that made up the head mount, except from one, are no longer available. During the specialization project, wanted posters was put up on campus to make an attempt to recover the missing parts. No one replied to the posters, and as a result from this, a new head mount will have to be developed.

Initially, the plan was to attempt to develop a graphical user interface, a new head mount, and a compliant control system for the robot. It was quickly realised that developing all these parts would be too comprehensive, so some prioritization had to be made. Aleksander Lillienskiold at SINTEF revealed in a teams-meeting that they had been working with Panda robots there, specifically on making them as compliant as possible, which would be very beneficial for this project. However, it was very hard to get in touch with the correct people at SINTEF to plan a meeting or visit to see what they had done, due to the current pandemic. It is believed that it will be easier to establish a better dialogue with SINTEF when the pandemic is more manageable. Due to this, it was decided to prioritize the development of the new head mount and the graphical user interface, and not continue to develop a compliant control system for the robot.

1.2 Problem description

This has been an ongoing project since 2014. The long term goal is to develop a training apparatus for whiplash patients which fills the demands of a minimum viable product (MVP). There are two main areas that have been the focus of this thesis: Creating a secure head mount and continuing the development of the graphical user interface. Both will be further described below. Another important aspect in this thesis has been to make it as easy as possible for the next students to continue the work. The main goals regarding the creation of a new head mount are:

- Designing a helmet that can be worn by any patient regardless of head size and shape.
- Producing the helmet in a material that is rigid enough to maintain the precision of the robot when performing a training exercise.
- Develop an inflatable element that ensures that the helmet is comfortable to wear, while still keeping the head in the exact same spot.
- The inflatable element should also ensure that the helmet can fit anyone by adjusting the amount of air in the inflatable.
- Creating a connection to connect the helmet with the robot. This connection should include a safety feature to protect the patient from unexpected, and potentially dangerous, movements from the robot.

The main goals regarding the development of the graphical user interface are:

- Build a new graphical interface that is separated from the rest of the control loop in order to ensure that the real-time constraints of the robot control loop is not affected by the user interface.

- Establish network communication between the robot control program and the new graphical user interface
- Set up a database to store exercises and training programs used in the treatment of whiplash
- Continue the development of the functionality that are required for a minimum viable product
- Set up a software platform that is easy to build further on in upcoming projects.

1.2.1 Limitations

There are some limitations in this project which will be explained here:

- There will be no more work done in regards to developing a compliant control system for the robot, as it is believed that it will be easier for a group to continue the development with valuable insight from SINTEF when the pandemic gets under control
- No user tests performed by outsiders to test the functionality of the head mount will be done, to maintain infection control.
- No user tests performed by outsiders to check how intuitive the graphical user interface will be done, to maintain infection control.

1.3 Disposition of this thesis

This section will give a quick description on the disposition of the thesis.

The introduction will present the background and motivation for this thesis, and give a problem description based on that. The next chapter, theory, will explain relevant theory to better understand what is happening in the neck during whiplash and why it is a problem. It will also present relevant control theory to better understand the functionality of the control system. After that, the previous work chapter will explain the most important findings from previously written thesis' as well as an explanation of how the control system currently works from the author's own project report. The next chapter, development, will describe what has been done during this project, and the current implementation chapter will present the current state of the system. Discussion and further work will discuss the implemented solutions in relation to the problem description, and provide suggestions on how to further develop this project. Lastly, the conclusion chapter will quickly summarise what has been done.

2. Theory

This chapter will describe relevant theory to better understand why there is a need for a rehabilitation apparatus for whiplash injured patient. It will also cover some control theory so it will be easier to understand the control system.

2.1 Neck and whiplash

This section on the neck and whiplash is mostly based on the authors own project report. It will describe how the neck works and what a whiplash injury is.

2.1.1 The neck

The neck is one of the most important parts of the body, and is responsible for supporting and moving the head. The most important parts of the neck are cervical vertebrae, ligaments and muscles. The cervical vertebrae are the seven upmost parts of the spine. They are named C1-C7, where C1 is the top one. These vertebrae are shaped in a way that allows for movement in 6 degrees of freedom. However, it is the contraction of the many different muscles that makes the head move. The ligaments can be seen as elastic bands that can stretch and retract within their own elastic range without problems. However if a ligament is stretched beyond its elastic range, it will be permanently extended.[4]

Out of the seven vertebrae, C3-C7 are very similar to the rest of the spine. The vertebrae are stacked on top of each other and helps with keeping us upright. They also allow for some movement and rotation. C1 and C2, also known as ”Atlas” and ”Axis” respectively, are a bit different. They are shaped in a way that allows for much bigger rotation of the head. C1 is shaped like a ring and sits on top of a small bone from C2. How C1 and C2 move in relation to each other accounts for about 50% of the entire area where the neck manage to move in rotation and extension/flexion.

Figure 2.1 shows the three usual motions used to describe neck movement, which is also the the movements that can be done with the MCU. The figure is from a NASA article providing data on the necks range, which is important for the robot to handle. Table 2.1 shows the range of the female neck in the movements shown in Figure 2.1. The results of the female

Table 2.1: Neck range

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

neck is chosen as females usually have a bigger range than males, and the rehabilitation apparatus has to work for both genders.

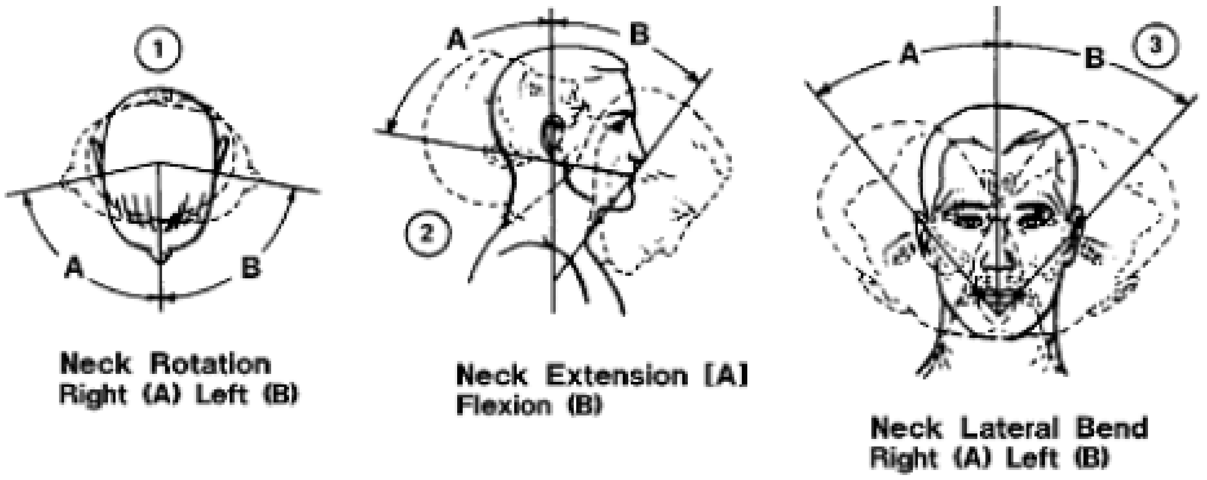


Figure 2.1: Neck motions

2.1.2 Whiplash

A whiplash injury is described as a neck injury caused by a forceful, rapid back-and-forth movement of the neck, just like cracking a whip [5]. Figure 2.2 shows the sequence of events which can result in a whiplash injury. A similar movement by the head and neck in any other direction can also be described as whiplash. The most common cause of whiplash is rear-end car accidents, but it can also be a result from a sports accident, physical abuse or other trauma.

During a whiplash, the neck is exposed to such quick accelerations and big movements that multiple parts of the neck can be damaged. If the ligaments are stretched out of their elastic range, it can cause permanent damage. Muscles, nerves and joints can also get damaged. What parts of the neck that are injured can be very hard to determine, resulting in difficulties when trying to decide the extent of the injury. This can lead to discussion of whether an injury actually exist, which can result in problems when getting insurance to cover costs.

The difficulty in deciding the injury can also present problems when it comes to rehabilitation. Imaging such as x-ray or MRI can only identify the most severe cases such as fractures, displacement of vertebrae or damages to the spinal cord. Training the neck after a whiplash

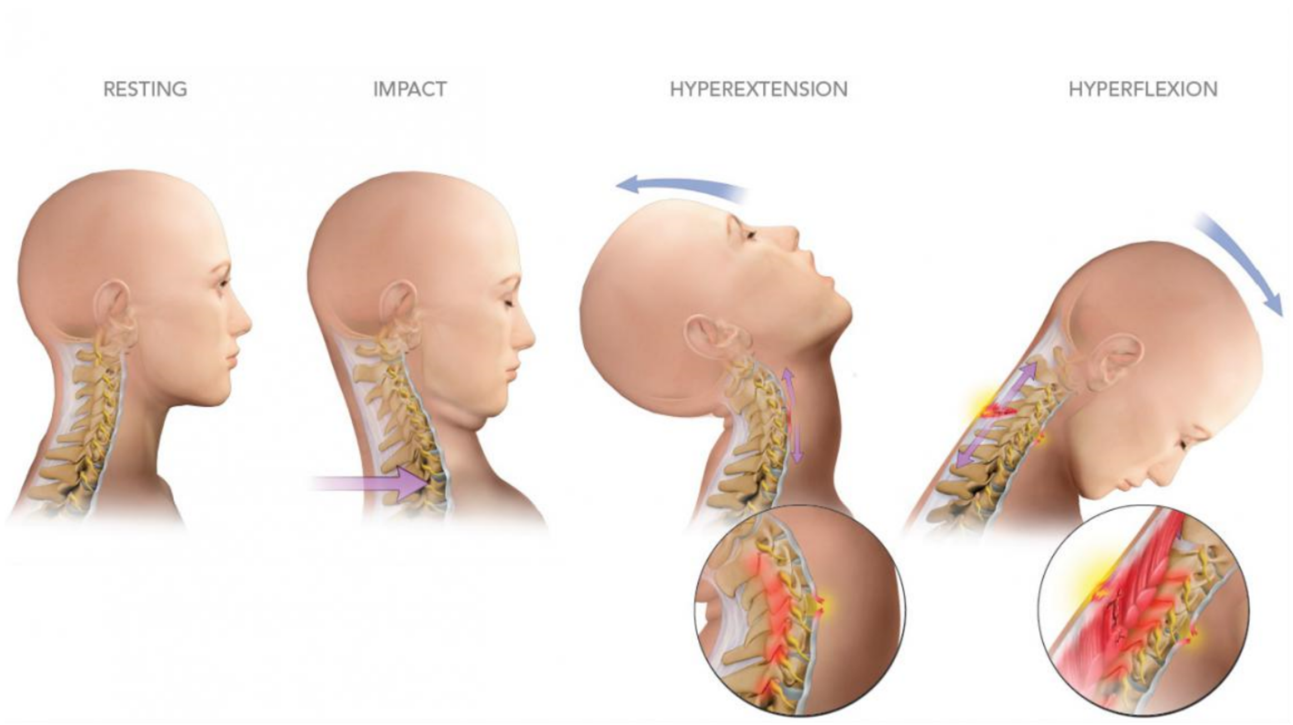


Figure 2.2: Sequence of events during whiplash

is disputed, even though there has been a lot of research yielding positive results. An article from New South Wales State Insurance Regulatory Authority reports such good evidence of training the neck having a positive effect that it should be the recommended treatment in most cases [6].

Firda fysmed has specialized in whiplash injuries and they agree that training is the best method for rehabilitating the neck. Their focus is to train the muscles around the injured area, which has gotten great results. 97% of the patients treated at Firda fysmed reported an improvement of their situation in a survey [7]. This survey also report that the degree of disability of the patients is reduced. Despite these great results, Firda fysmed believes that training with an optimized apparatus will yield even better results. Even though it is hard to clinically prove improvement by x-ray or MRI, improvements can be seen by increased strength and mobility. The most important thing is that the treatment can help improve the quality of life of the patient.

2.2 Control theory

This section on control theory is mostly based on the authors own project report. It will describe relevant control theory.

2.2.1 Admittance controller

An admittance controller is a controller based on the mass-spring-damper system shown in Figure 2.3, which means that the robot is supposed to replicate the behavior of a mass connected to a spring and a damper. Equation 2.1 shows the resulting expression for the desired acceleration, $\vec{a}$.

$$\vec{a} = \frac{1}{m(\vec{F_{ext}} - k\vec{r} - c\vec{v})} \tag{2.1}$$

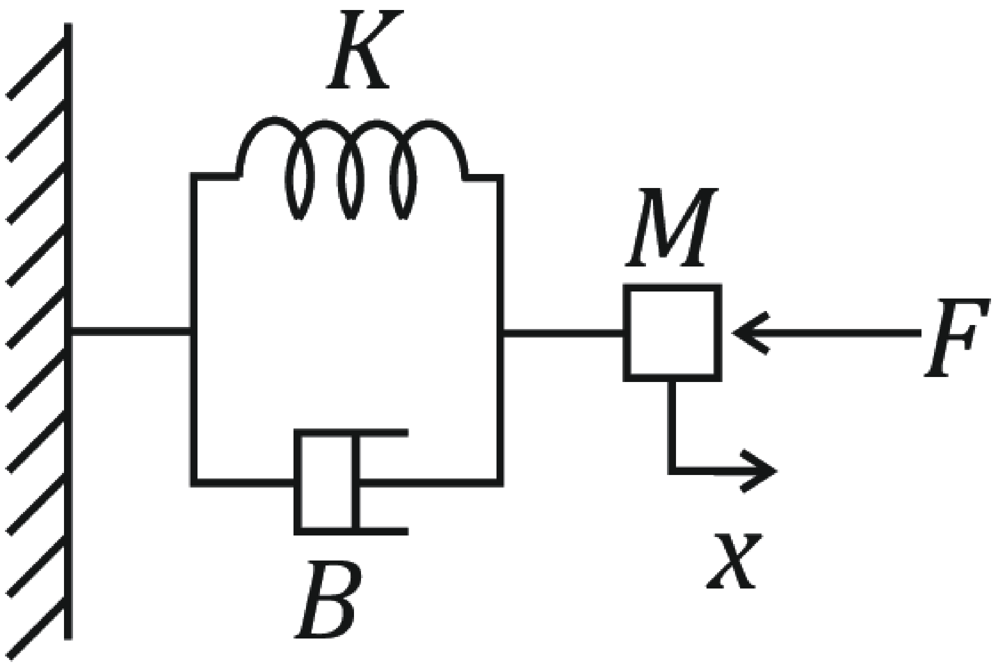


Figure 2.3: Mass-spring-damper system by Pertuz et al [2]

$c\vec{v}$ is the force applied by the damper, $k\vec{r}$ is the force applied by the spring, m is a virtual mass and $\vec{F_{ext}}$ is the external force estimated at the flange.

The spring in this system would try to move the robot to its original position, an undesirable behavior for this projects purpose. The spring force was therefore removed from the expression for the controller (Equation 2.1), which resulted in the following expression for the controller:

$$\vec{a} = \frac{1}{m(\vec{F_{ext}} - c\vec{v})} \tag{2.2}$$

2.2.2 PID controller

A PID controller is a great method for controlling a system around a set-point. A PID controller uses the difference between the desired set-point and the current state of the system, also called an error, as an input. The output from the controller tries do reduce this error so that the system state is in accordance with the desired set-point. There are three terms that decide the output from the controller, the proportional term P, the integral term I, and the derivative term D. The block diagram for a PID controller is shown in Figure 2.4.

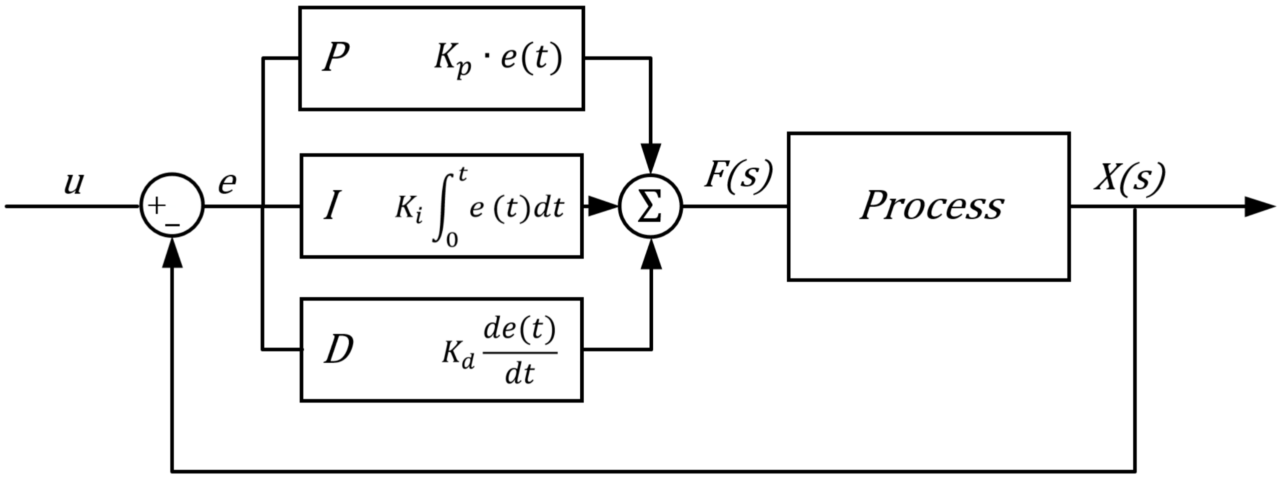


Figure 2.4: Block diagram for a PID controller [3]

P term

The proportional term is simply used to produce an output proportional to the error. A big proportional gain, K_p will give a big change in the output relative to the error, but can make the system unstable if tuned too high. On the contrary, a small proportional gain will give the controller less responsiveness and less sensitivity, but may not be able too reach the set-point if tuned too low.

I term

The integral term is used to produce an output based on the accumulated error over time. The integral value of the error is multiplied by an integral gain, K_i. This can remove the steady-state error in a system, but too big of an integral gain can cause the system to overshoot the set-point which increases the time before the system reaches the set-point, and may result in an unstable system.

D term

The derivative term uses the rate of change of the error and multiply this by an derivative gain, K_d. This term is used to predict the system behavior and will help reduce the time before the system reaches the set-point, and also improve the stability of the system. However, the derivative term is often not used as it is very sensitive to disturbances.

2.3 Terms

Object Oriented Programming

Object oriented programming is a form of programming that revolves around objects. An object is an instance of a class. A class contains methods, which are functions that define the behaviour of the object, and fields, which are variables that define the state of the object. A class is a ”blueprint” that defines the possibilities for what state and behaviour an object can have. Java is a programming language that was built around Object Oriented programming. All java programs are written in classes, so you cannot use Procedural programming in Java. C++ is an extension of C that introduces, among other things, classes which allows Object Oriented programming.

Head mount

Head mount is in this thesis used as a collective term for the helmet, the inflatable element and the magnetic connection.

3. Previous work

This has been an ongoing project from 2014, resulting in quite a few master theses written. This chapter will explain the most significant findings from previous thesis', mainly Baardsgaard and Brekke (2019) and Brattgjerd and Festøy (2018), as well as the authors own project report.

3.1 Robot Control

This entire section on robot control is the same as the corresponding section in the authors own project report. It is included since the control system for the robot have not been developed since the specialization project, and it is more practical for students continuing the development of the control system to have all relevant information in one single document.

The robot loop Thread runs a 1kHz control loop that receives force measurements from the robot, and calculates robot movement. The controls are implemented using libfranka, which is a library developed by Franka Emika for use with the panda robot in the Franka Control Interface (FCI).

The robot control loop is the part of the system where real-time constraints are most important. According to the FCI documentation the constraints are defined as the following:

> It must be guaranteed that the sum of the following time measurements is less than 1ms:
>
> - Round trip time between the workstation PC and FCI
> - Execution time of your motion generator or control loop.
> - Time needed by the robot to process your data and step the internal controller

Baardsgaard and Brekke started the work with developing a compliant control system for the robotic arm, but there is still some work to be done when it comes to making a compliant robotic arm, i.e lowering the resistance when moving the robot so that it can easily be moved in any direction. A big part of this specialization project has been investigating the previous work and understanding the different concepts.

3.1.1 System description

The overall target is to make a control system that makes the robot compliant to the point where the user can move the end-effector with a range from almost no resistance to resistance decided by the physiotherapist. The solution by Baardsgaard and Brekke can be divided into the following three parts:

- Getting the data from the internal sensors in the robot and processing it.
- Using a force controller which converts the force input data to a desired cartesian velocity.
- Converting the cartesian velocities to joint velocities, and controlling the Panda to carry out these velocities.

3.1.2 Getting sensor data

With the use of libfranka, obtaining the different sensor data is very simple. Most values can be taken from `robot_state`. Performing matrix and vector calculations is more easily done with the Eigen library[8], which is used when there is a need to map the values from `robot_state` into eigenvectors or eigen matrices. `franka::jointVelocities` was the interface chosen for the velocity controller, which results in the cartesian velocities not being obtainable from `robot_state`. This means that the cartesian velocities have to be calculated using the Jacobian and joint velocities with the following equation:

$$\vec{v} = J\vec{q} \tag{3.1}$$

where $\vec{v}$ is a 1 $\times$ 6 vector of cartesian velocities, J is the 6 $\times$ 7 Jacobian, and $\vec{q}$ is a 1 $\times$ 7 vector of joint velocities. External force on the flange can be obtained from `robot_state.k_f_ext _hat_k`. There is no torque sensor at the flange so `robot_state.k_f_ext_hat_k` uses the torques from the torque sensors in the joints to estimate a force at the flange. `K_F_ext_hat` is a 1 $\times$ 6 vector containing torques around the X-, Y- and Z-axis as well as forces in the X, Y and Z directions. This force signal is corrected for gravity, inertia, coriolis and centrifugal effects. There is also a low-pass filter, with a cutoff at 10 Hz, included to reduce noise. The filter is a function from libfranka; `franka::LowpassFilter`. This results in a smooth signal.

3.1.3 Velocity controller

Libfranka contains multiple controllers for different methods of controlling the robot. You can control either velocity or position, in task space or joint space. The force controller, which will be described later, gives the desired velocity/acceleration in task space, so a task space (cartesian) velocity controller was needed. Libfranka has two existing interfaces for this kind

of control, one which uses cartesian velocities, and on that uses joint velocities. Both of these where explored, and using the joint velocities controller was decided because it yielded great velocity control and the velocity, acceleration and jerk of all joints were automatically limited. The cartesian velocities interface does not limit these parameters automatically, and to make sure the joint space limits were satisfied, you had to set the cartesian limits much lower than recommended by Franka, which constrains the potential of the robot.
When using the `franka::jointVelocities` interface the robot is controlled using joint velocities, and since a patient perform exercise motions in cartesian space, there is a need to convert the cartesian velocities to joint velocities. This can be done by using 3.2:

$$\vec{\dot{q}} = J^{+}\vec{v} \tag{3.2}$$

where J^{+} is the pseudo-inverse Jacobian.

3.1.4 Force controller

One of the most important objectives in this project is to be able to move the end-effector with as little resistance as possible. The controller developed by Baardsgaard and Brekke [9] did not include the option to rotate the end-effector. Different control strategies to reduce the resistance with a constant orientation of the end-effector were explored.

The easiest solution is a feedback control where the force input at the flange is fed back to the motors by Newtons 2nd law, $F = ma$, where F is the estimated force at the flange, a is the desired acceleration or the Panda and m is a virtual mass. The virtual mass should be very low to reduce the inertia, however when m approaches zero, a goes towards infinity. This results in rapid movements by the robot from small force inputs at the flange. Baardsgaard and Brekke [9] also reports that the robot moved with increasing oscillations. In this type of project, where a patient is wearing a helmet connected to the robot, such movements can be dangerous. A feedback controller based on Newtons 2nd law will therefore not be optimal in this project. Other controllers had to be explored. The most common controllers are an admittance controller, a PID controller, or a controller based an a dynamic model of the system. All of these controllers were explored.

3.1.5 Admittance controller

The theory behind an admittance controller is described in Section 2.2.1. To test the controller you had to assign a value to the virtual mass, m. With m=10kg a stable motion with a slow response was reported. To quicken the response, m can be reduced, but at some point the robot starts oscillating because the response was overcompensating. c can be introduced to dampen these oscillations. Increasing c while lowering m while still having a stable motion is how the controller is tuned. However, no combination of these values yielded acceptable results. The robot still required a considerable force to move.

3.1.6 PID controller

The theory behind a PID controller is described in Section 2.2.2. The PID controller tested by Baardsgaard and Brekke [9] used the difference between zero, which is the desired force, and the force applied to the robot, F_{ext}, as the input to the controller. The resulting output from the controller is a signal to control the desired velocity of the robot. When tuning the PID controller for the robot, it started oscillating when K_p was increased. The D term could be used to dampen these oscillations, however, a loud unpleasant sound came from the motors when increasing K_d. This sound is caused by small, rapid velocity changes in the motors, which happen because of noise disturbing the signal. Filtering was tried to reduce the noise, but the signal could not be filtered enough to get a satisfactory control output from using the PID controller.

3.1.7 Dynamic model

A dynamic model of the system can be used to make a control system based on the physical properties of a model with the desired behavior. For example, the previously mentioned admittance controller is based on a dynamic model of a mass-spring-damper system. To make a dynamic model to base the control system on one have to specify the desired behavior of the robot. The desired behavior of the Panda would be to make it move with as little resistance as possible, while being stable and not moving if there is no force input. Baardsgaard and Brekke [9] mentions the behavior of an object moving on a surface with friction to be similar to the desired behavior of the Panda, as it will be easily movable and will stop if not pushed.

Making a dynamic model of the desired object is done by combining the equations for the inertia of the mass (Equation 3.3), the Coulomb friction with static friction (Equation 3.4) and the kinetic friction (Equation 3.5).

$$\vec{F_m} = m\vec{a} \tag{3.3}$$

$$\vec{F_{fs}} = -\vec{F_{ext}} \tag{3.4}$$

$$\vec{F_{fk}} = -\mu m g \hat{v} \tag{3.5}$$

Equation 3.6 shows the resulting dynamic equation for the model. $\vec{F_f}$ is either the kinetic or the static friction. The kinetic friction only applies when the force applied, $\vec{F_{ext}}$ is bigger than $\vec{F_{fk}}$, or if the object is still moving. Similarly, the static friction only applies when $\vec{F_{ext}}$ is less than $\vec{F_{fk}}$, or if the object is standing still.

$$\vec{F_{ext}} = m\vec{a} + \vec{F_f} \tag{3.6}$$

To get commands to control the robot from Equation 3.6, it has to be rewritten to Equation 3.7, which gives the desired acceleration.

$$\vec{a_{desired}} = \frac{1}{m}(\vec{F_{ext}} + \vec{F_f}) = \frac{1}{m}\sum \vec{F} \tag{3.7}$$

From this equation and Equation 3.4, static friction would result in $\vec{a_{desired}} = 0$, which could mean that robot is moving with a constant velocity. This is solved by the fact that $\vec{F_{fk}}$ is used as $\vec{F_f}$ when the velocity is nonzero, and the kinetic friction will slow the robot down until it stops.

To optimize the controller from the dynamic model, the virtual mass m, and the friction coefficient μ had to be tuned. Tuning these parameters is done by starting with a large m and no μ, then increase μ while decreasing m. Oscillations occurred when m was decreased, but increasing μ was an effective way to dampen these oscillations. The values yielding the best results were m=2.0 kg and μ=0.01. The dynamic model based force controller gave the least amount of resistance when moving the robot of the mentioned controllers.

3.1.8 Adding weight

The end goal is to use the robot as an exercise apparatus, which means that one should be able to ”add weight” as is done in normal exercises at the gym. To make the Panda different from the MCU, it is important to avoid having to use force to hold the head still in the middle of, or after a movement. The MCU will try to ”pull” the head back to the original position.

The easiest way to add resistance only during movement is to add a friction force with the desired resistant force, resulting in a new expression for the force controller being Equation 3.8, where $\vec{F_w}$ is the wanted friction force. With this added friction, the robot will only resist when an exercise movement is performed, and no force will be required from the patient too keep their head in place after a movement is finished, or if they stop in the middle of a movement. There is also no desire to have added resistance while moving the head back to the original position. To avoid this, the completed percentage of the motion is calculated. When 99% of the motion is completed, the resistance will be set to zero until the patient is back in the starting position, where the desired resistance is added again.

$$\vec{a_{desired}} = \frac{1}{m}(\vec{F_{ext}} + \vec{F_f} + \vec{F_w}) \tag{3.8}$$

3.1.9 Joint impedance

The stiffness of the robot is defined by the impedance of the joints, and need to be defined before starting the robot loop. Baardsgaard and Brekke [9] reports that setting a low joint impedance dampen oscillations caused by the force controller. After some experimenting with the impedance parameters, the stability of the controller increased. However, lower impedance reduces the accuracy of the robot, but can help the robot feel more safe and human for the patient.

3.1.10 Limiting joint positions

The robot will stop and give an error if the joints move outside limits described in Franka docs [10]. This is solved by Baardsgaard and Brekke [9] by resisting movement in the joints when approaching these limits. To resist movement when approaching joint limits, a PID controller is used.

The desired velocities from the force controller are used to calculate the desired joint positions, and then the desired joint positions are clamped to their respective limits. The difference between desired position and clamped position is fed into the PID controller as an error. The controller will try to minimize this error and will add the output to the desired positions as a correction.

The PID controller need to be tuned, Baardsgaard and Brekke [9] reports that K_p=0.001 and K_d=0.00002 gives a firm stop without being to harsh.

3.1.11 Recording a path

To use the Panda as an exercise apparatus, the patient needs to be able to repeat the desired motion from the physiotherapist multiple times. This is done by recording the motion while the physiotherapist guides the patient through the desired motion without resistance from the robot. After a motion is recorded, resistance can be added so the exercise can be performed.

A motion is recorded by obtaining the Cartesian positions of the end-effector from `robot_state.O_T_EE`. The positions are pushed to a vector in `robot_data.tracking_data`, which is shared so it can be read from another thread. A shared thread is required because the time it takes to save the data might exceed the 1ms time limit for the robot loop.

The data is saved to a .csv file where each row contains a set of X, Y and Z coordinates. Multiple sets of these coordinates is called a path. The reason .csv was chosen is because it is easy to write to with simple delimiters. It is also great if the points needs to be visualized for debugging using spreadsheet software.

The robot loop runs at 1kHz, which means that data is sampled 1000 times each second leading to an extremely detailed path and a large amount of data. This seems excessive so Baardsgaard and Brekke [9] decided a sampling rate of 100 times per second would be sufficient, which resulted in a smooth path.

3.1.12 Following a path

After a path is recorded, there is also a need for the robot to replicate this path. Baardsgaard and Brekke [9] tested multiple ways to achieve this which will be described in this section.

The first idea is based on taking the Cartesian velocity output from the force controller and manipulating it towards the same direction as the direction which should be constrained. This is done by using the dot product between the Cartesian velocity and a unit vector in the chosen direction. With this method, the robot only moved along the chosen vector, however this method does not account for the position of the robot which means that the robot only follows the direction of the path, not the path itself. This can result in an offset from the desired position.

As the results from the previously mentioned method of following a single vector was not satisfactory, something else had to be tested. This approach is based on using the many points used to create the path, specifically the points that are closest to the position of the end-effector. A vector can be created from the points closest and 2nd closest to the end-effector. Constraining the velocity to this vector should in theory update the constrained direction which makes the robot follow the path. This approach was relatively successful, however the robot struggled in the corners of the path. Offset can also cause problems as there is such a small distance between time steps, which can lead to the robot skipping between vectors resulting in movement in another direction than the desired one.

The two previous methods have been based on velocity control. To better the robots ability to follow a path, Baardsgaard and Brekke [9] thought of a new approach based on position control. In this approach, the desired velocity from the force controller is integrated to find the desired position. After that, the distance between the desired position and the closest point on each line segment on the path could be found. A line segment is a line between two of the points in the desired path. To figure out which point on a line segment is the closest, one can project the desired position onto the line. If the projected point is on the line then there is no error and the function can just return the projected point, if not, the function should return the closest point on the line. The new desired position was the calculated point that was the closest from all the line segments.

A PID controller was used to control the robot to the desired position. The error fed into the PID controller was the difference between the desired position, and the output from the controller is a ”correction” which is added to the desired velocity. This will help the robot steer to the path. Baardsgaard and Brekke [9] reports that after some tuning of the PID controller, the robot is able to perfectly follow a path. Together with the force controller, this method made the robot easy to move through a recorded path, but if K_p in the D term is too high the motors can emit an audible sound.

3.2 Head mount

The head mount serves as the connection between the patient and the robot. It is important that the head mount is rigid to minimize slack when performing exercises, but it should also feel safe and comfortable for the patient to wear. The head mount should also include a safety measure in case of unexpected movements by the robot. Previous projects have investigated several solutions for a head mount, and concluded that a helmet with inflatable pillows and a magnetic connection to the robot seemed the best solution.

3.2.1 Mounting mechanism for helmet

Baardsgaard and Brekke (2019) stated the following as the requirements for a mounting mechanism for the helmet:

- 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.

Out of these requirements, the safety requirement is the most important. Complex mechanical solutions and electric solutions were quickly deemed unfavorable as the mechanism should not be able to fail. In stead, a solution using magnets was investigated by Baardsgaard and Brekke.

The mechanism was developed by rapid prototyping, which resulted in a two part mechanism. The parts are 3d-printed with 16 $10mm \times 2mm$ neodymium disc magnets on each part. The part on the helmet is spherical with the corresponding part on the flange of the robot being concave. The helmet part of the mechanism is no longer available, so a new one that will fit the part mounted on the robot will have to be made.

3.2.2 Helmet

The helmet concept developed by Brattgjerd and Festøy (2018) used a Etto Twister alpine helmet. This helmet is quite big, and Baardsgaard and Brekke (2019) discovered that the distance from flange to head could be reduced significantly by redesigning the helmet.

The outer shell of the alpine helmet is the only essential part of that concept, which means that it is possible to recreate the outer shell with another material, such as the PLA used in 3d-printing. Baardsgaard and Brekke (2019) designed a new shell consisting consists of multiple parts glued together, however, it might require some adjustments to make sure it can fit any human head. The alpine helmet had some padding between the outer shell and the inflatable bag. With the 3d-printed shell, the lack of padding leads to a larger distance between the head and the outer shell. The added space between outer shell and the patients head increases the looseness as the air can move around inside the inflatable. This should be considered when making a new inflatable element.

3.2.3 Inflatable element

The helmet should be able to fit all human heads, however there needs to be some way to make some adjustments according to head size. Brattgjerd and Festøy (2018) suggested using an inflatable element for this purpose, and started the design for it. All prototypes were made of 0.7mm thick PVC from an inflatable mattress, with a thin velour cover. A normal iron was used to melt the sides together and create an airtight seal, and a bulb pump was used to inflate them.

The prototypes they made and their properties will be summarised here. Areas marked with blue is the pressure from the inflatable element on the head, whilst red marked areas is where the head presses on the helmet.

Prototype #1: One large inflatable element

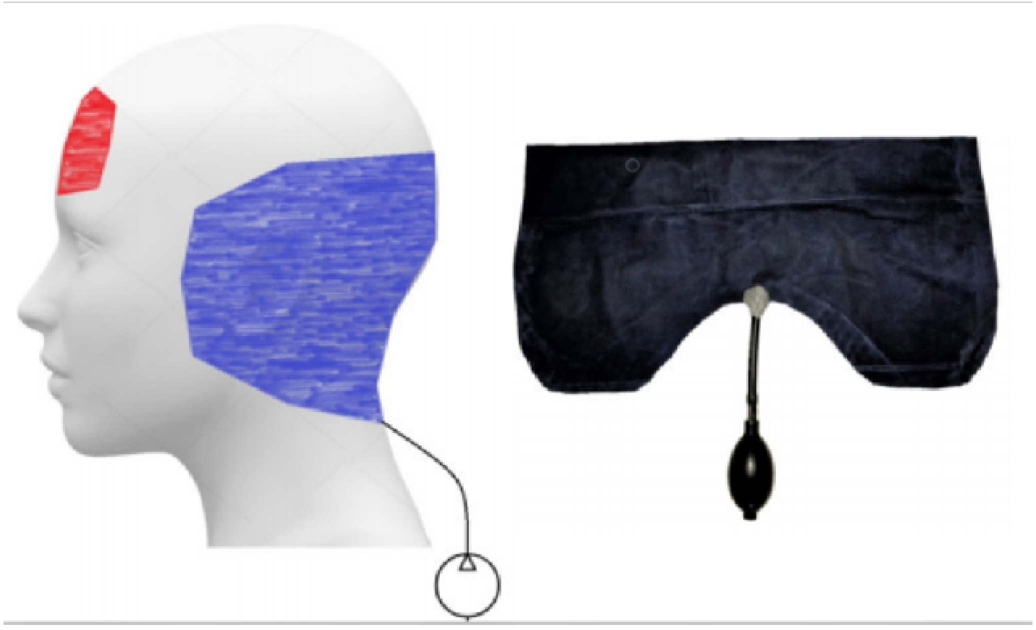


Figure 3.1: Pressure zones and design of one large inflatable element

This design consist of one large inflatable chamber with the bulb pump placed in the middle as shown in figure 3.1. It is designed to cover the entire back of the head and ears of the patient, as shown by the blue marks in figure 3.1. When inflated, the forehead of the patient will start to push against the helmet as shown by the red markings in figure 3.1.

Overall, this design was not great. The positives of this design was that the thin layer of velour made for a very comfortable air pillow, which was sturdy when correctly inflated. However, the placement of the bulb pump made it very inaccessible as the patient might have to lean forward when inflating the pillow which can be uncomfortable, especially for whiplash injured patients. Another problem was that the size of the pillow made for some obstruction when performing backwards motions. Ear fatigue was also a problem after about 10 minutes.

Prototype #2: Inflatable element with two airpockets

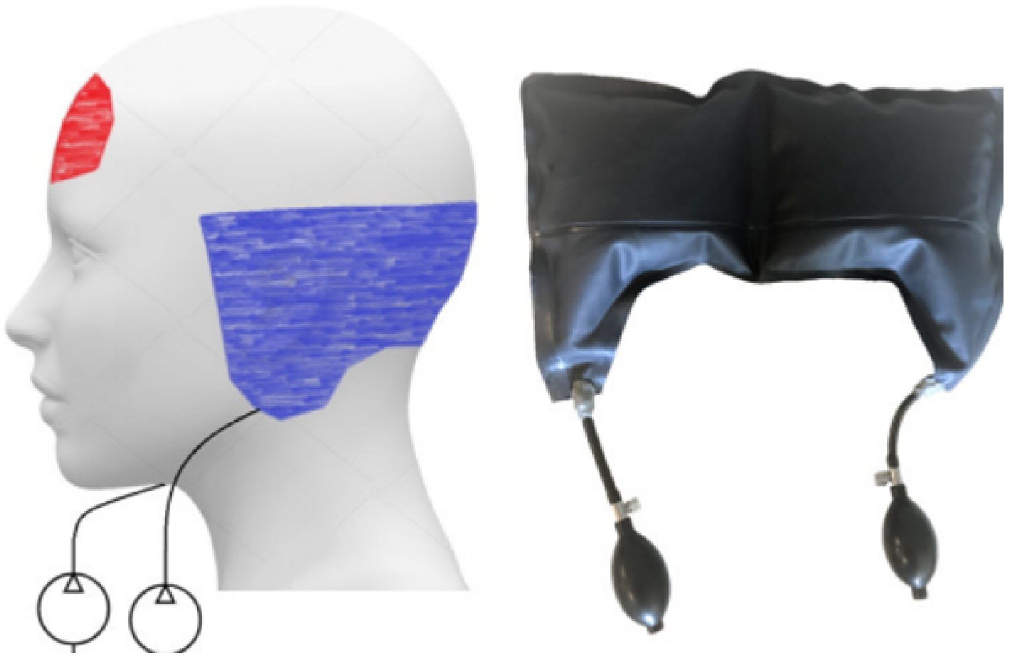


Figure 3.2: Pressure zones and design of the inflatable element with two airpockets

This design consists of two inflatable chambers with a separate pump for each part. The pumps are now placed on the tip of each chamber, making them more accessible. Prototype #2 is also designed to cover the ears and the back of the head. The weld in the middle is to keep the air from moving side to side when moving the head. Figure 3.2 shows that the pressure applied by prototype #2 is mostly the same as prototype #1.

As the design resembles the previous design, a pretty similar result was expected. Changing the position for the bulb pump was the only positive change. The weld to keep the air from moving side to side had no noticeable effect compared to one big chamber. Having to separate chambers just let to problems getting the same pressure in both chambers. The problems with the size were still present, as prototype #2 also restricted movement when performing backwards motions, with ear fatigue still being an issue.

Prototype #3: Inflatable element with ear slot

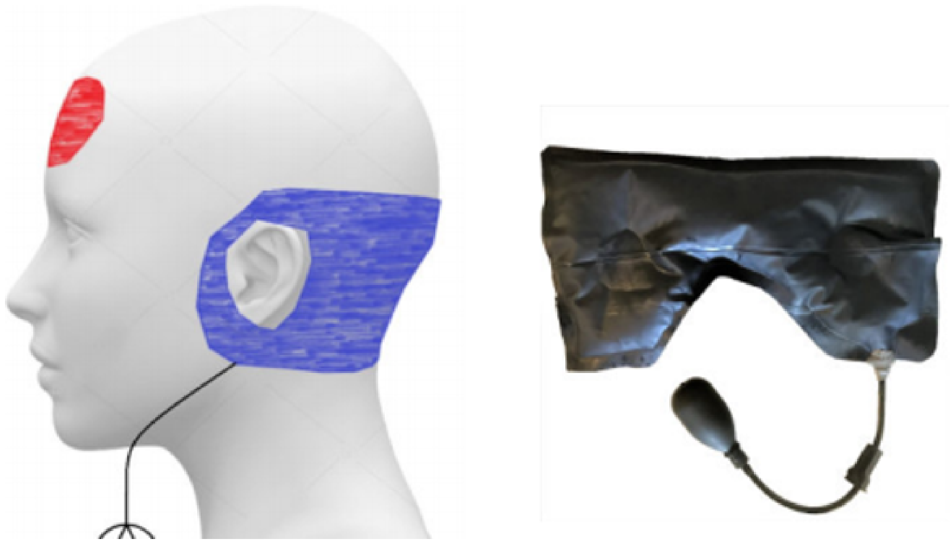


Figure 3.3: Pressure zones and design of the inflatable with ear slots

This prototype is designed to avoid ear fatigue, and the size was also reduced to avoid the problems when performing backwards motions as reported from prototypes #1 and #2. The pump is also located on the left side for easier accessibility. Even with the reduced size, figure 3.3 shows that the pressure zones are similar to the previous designs, except from the pressure on the ears.

The slot for the ear worked as intended. There was no reported ear fatigue after 20 minutes of use. An added benefit of having ear slots was that hearing improved, and the overall comfort was increased. Reducing the size also avoided the problems related to backwards motions being restricted. However, having less material around the temple reduced the performance when performing rotational and sideways motions.

Prototype #4: Inflatable element with forehead pillow

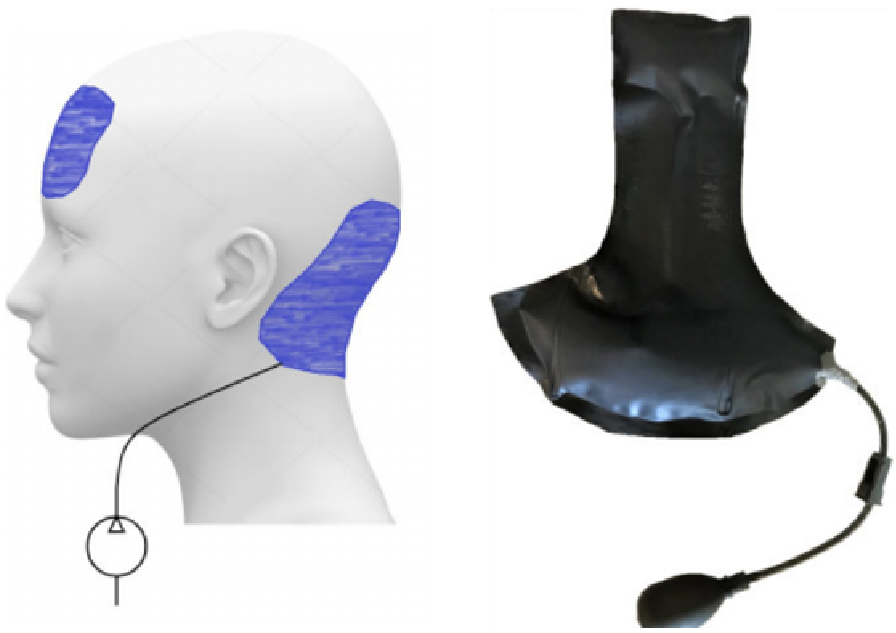


Figure 3.4: Pressure zones and design of the inflatable element with a forehead pillow

This prototype is mainly designed to test the effect of applying the same pressure to the forehead and the back of the head. Figure 2.4 shows the design with a thin air canal going from back to front to even the pressure. There are no ear slots in this design as it was designed to test forward and backward motions.

With the main purpose of this design being to test if a forehead pillow would increase performance in forward and backward motions, it unsurprisingly performed horrible in regards to sideways and rotational movement. However, having a forehead pillow resulted in the best design regarding forward and backward motions. No fatigue was noticed with this design.

Summary

From these prototypes, a solution including a forehead pillow and slots for the ears is optimal. It is important that the chamber pushing on the back of the head is not too big. The forehead pillow should also be wider so it covers the temple, increasing performance in sideways and rotational movements. The tube leading air into the inflatable should be located on either side for easier access.

4. Development

This chapter will explain the process of developing and creating the three main parts of the head mount, which is the helmet, the inflatable element, and the magnetic connection. It will also explain and describe the development of the graphical user interface.

4.1 Head mount

The head mount concept developed by Baardsgaard and Brekke (2019), and to some degree Brattgjerd and Festøy (2018), is no longer available. From the report of Baardsgaard and Brekke (2019) it is concluded that their concept, with some minor tweaks, should be satisfactory according to the requirements. This section will explain the process of creating the new head mount.

4.1.1 Helmet

Design

The helmet should be big enough to fit any human head without being so big that slack can be an issue. A bigger helmet could also be intimidating for a patient getting treatment for the first time. This is not a protective helmet like a bike helmet, so there is no requirement for the helmet to withstand any sort of knock or crash. The main objective of the helmet is to keep a stable connection between the patients head and the robotic arm.

It is also important to note that there is no universal head shape. According to a research project on head shapes conducted by the Hohenstein Institute [11], people have different head shapes ranging from extremely round to extremely oval as shown in 4.1. This figure shows the five shapes identified by the research project, with the difference between each shape being 0.4cm in length and 0.8cm in width.

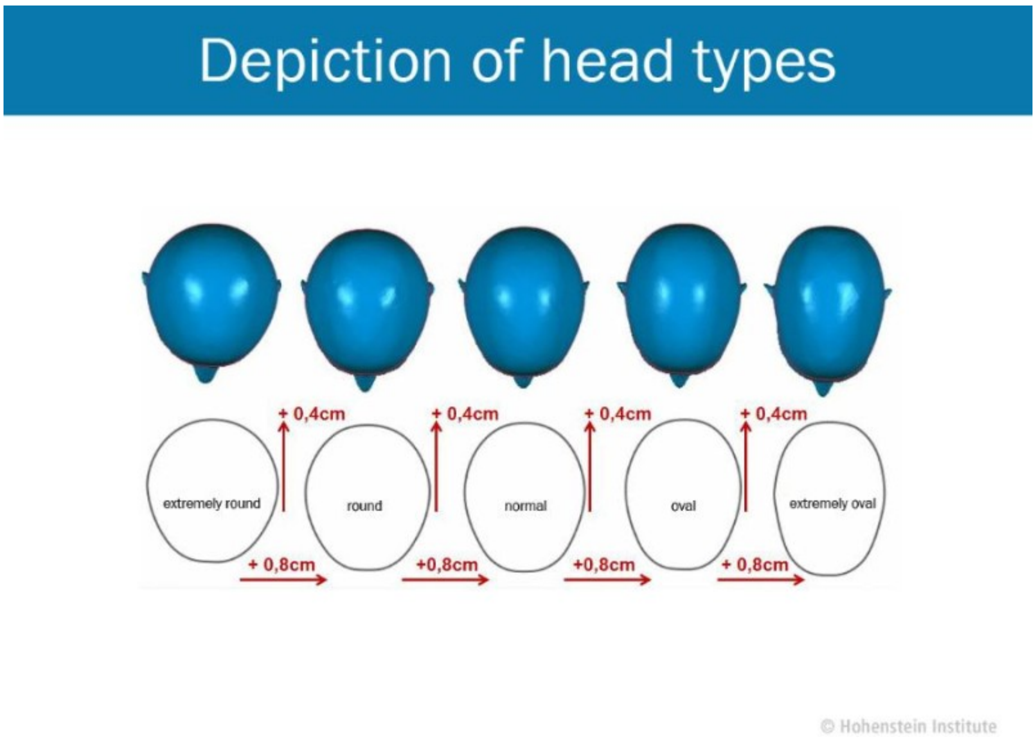


Figure 4.1: The five head types identified by the Hohenstein Institute

Wikipedia can be used to find the most important measurements on the average head. A male head is generally larger than a female head, so the male measurements will be the base. The breadth and length of the head is the most relevant for this design. Head breadth is the maximum breadth of the head, measured from ear to ear as shown by 1 in 4.2. Wikipedia reports that 99% of male heads fall within a breadth of 16.5cm [12]. Head length is the length from the middle of the eyebrows to the back of the head as shown by 13 in 4.3. Wikipedia reports that 99% of male heads fall within a length of 21.3cm[12].

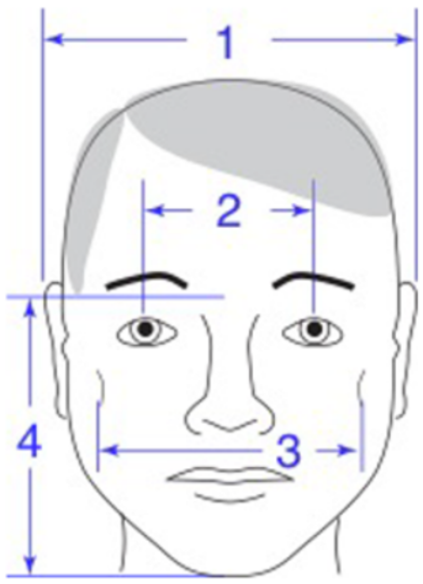


Figure 4.2: Head measurement illustration. 1 is the head breadth

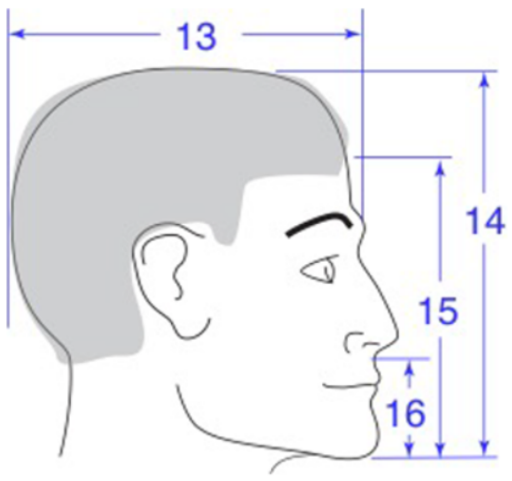


Figure 4.3: Head measurement illustration. 13 is the head length

The helmet was designed using Siemens NX CAD software. Figures 4.4 to 4.7 shows the helmet design. The length and breadth dimensions are a bit bigger than the previously mentioned 99th percentile measurements to ensure that there is room for the inflatable element. From the figures one can also see that there is made a slot for the ears to ensure ear comfort. The exact measurements on the helmet are:

Length: 23.3cm

Breadth including the ear slots: 21.4cm

Breadth not including ear slots: 20.0cm

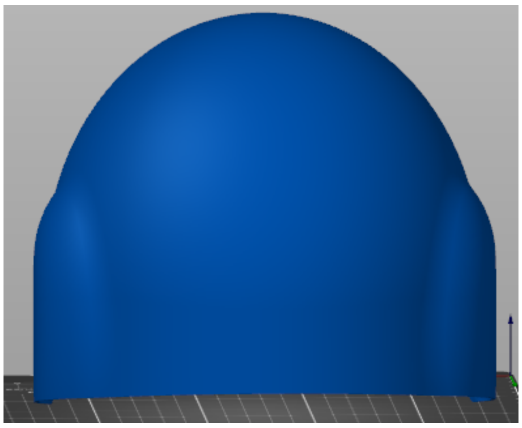


Figure 4.4: Helmet design seen from behind

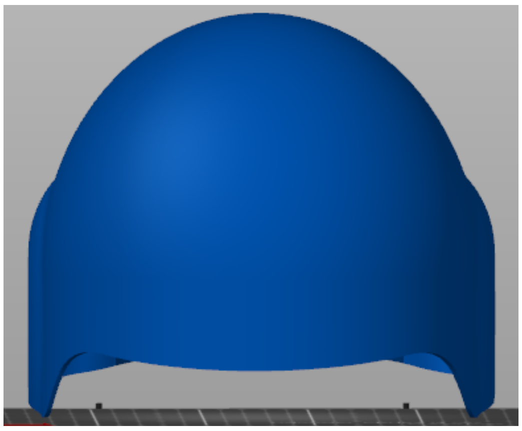


Figure 4.5: Helmet design seen from the front

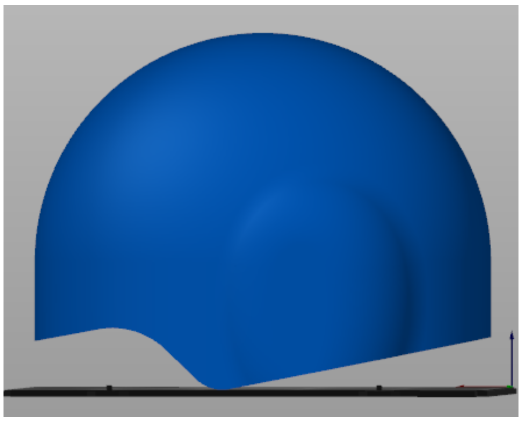


Figure 4.6: Helmet design seen from the left

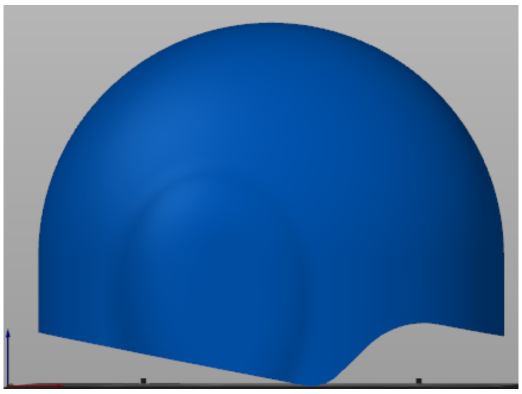


Figure 4.7: Helmet design seen from the right

Production

The helmet was produced using the Prusa i3 MK3 3D-printer available on campus. Since these printers are quite small, the helmet had to be split into smaller parts that could be printed and then glued together. The parts were glued together using regular super glue and held together for approximately 5 minutes for the glue to solidify.

All the print files are available at Github: `https://github.com/halvorrom/hjelm`. If there is a need to print another helmet on a later occasion, remember to check if you are using the same printer or the Prusa i3 MK3S. The smallest parts takes about 6 hours each to print whilst the biggest part takes almost 10 hours.

Figures 4.8 to 4.11 shows the helmet with the parts glued together.

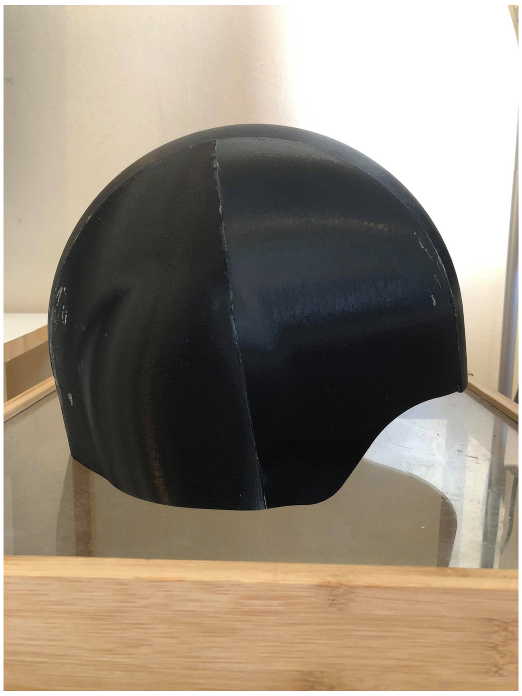


Figure 4.8: Finished helmet seen from the right

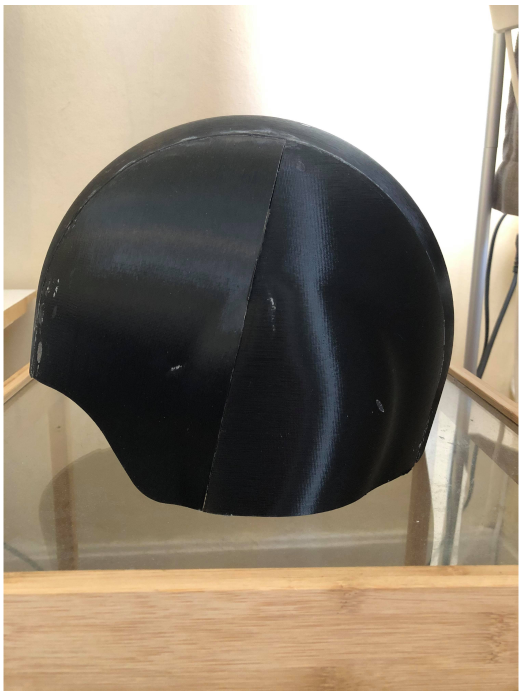


Figure 4.9: Finished helmet seen from the left

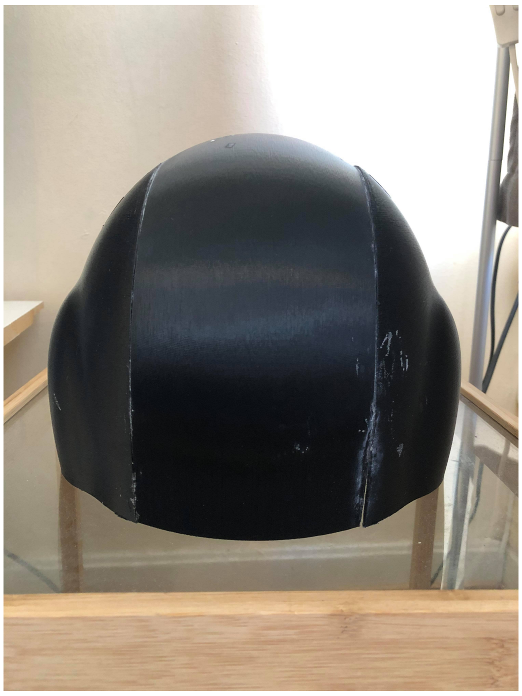


Figure 4.10: Finished helmet seen from behind

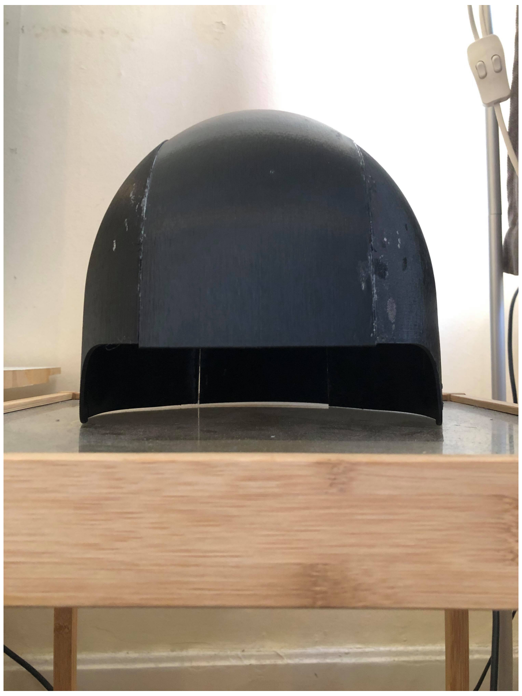


Figure 4.11: Finished helmet seen from the front

4.1.2 Inflatable element

Design

The inflatable element is required to be able to fit the helmet on any patient regardless of size and shape of the head. The previous work (REF TIL KAPITTEL) has given some idea of the ideal shape, however, no measurements or dimensions have been given so some experimenting is required. Figure 4.12 shows a rough sketch of the desired shape of the inflatable. The shape is designed to cover the forehead and temple as well as the back of the head. The circular white parts in the sketch is to illustrate where the ear slots should be.

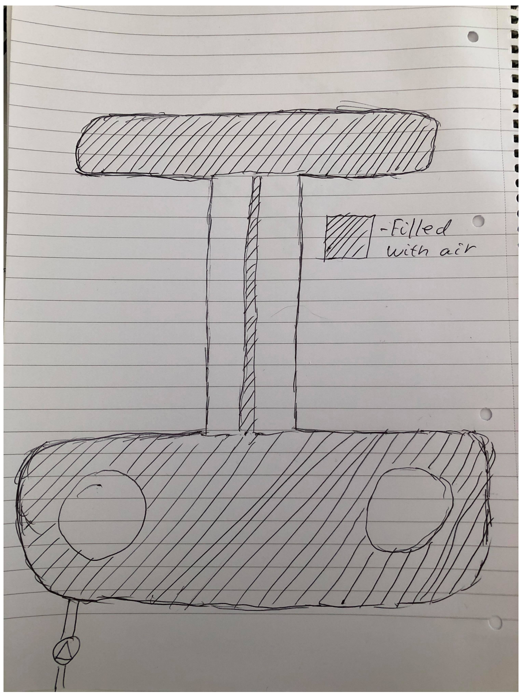


Figure 4.12: Rough sketch of inflatable element

To get a better picture of what size and shape the inflatable needs to be, a template to see how the sketch would actually fit a real head was made. The template is made by cutting the desired shape from a plastic bag. Figure 4.13 shows the shape of the template and figure 4.14 shows how the template fits on Halvors head.

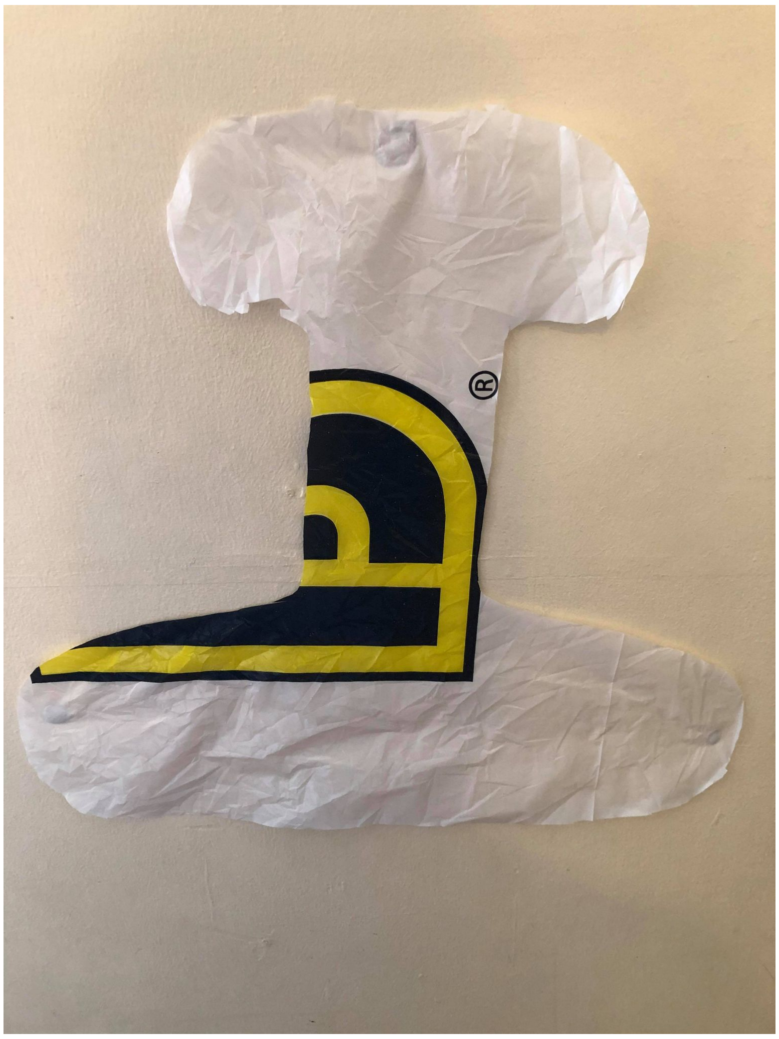


Figure 4.13: The first template seen from above

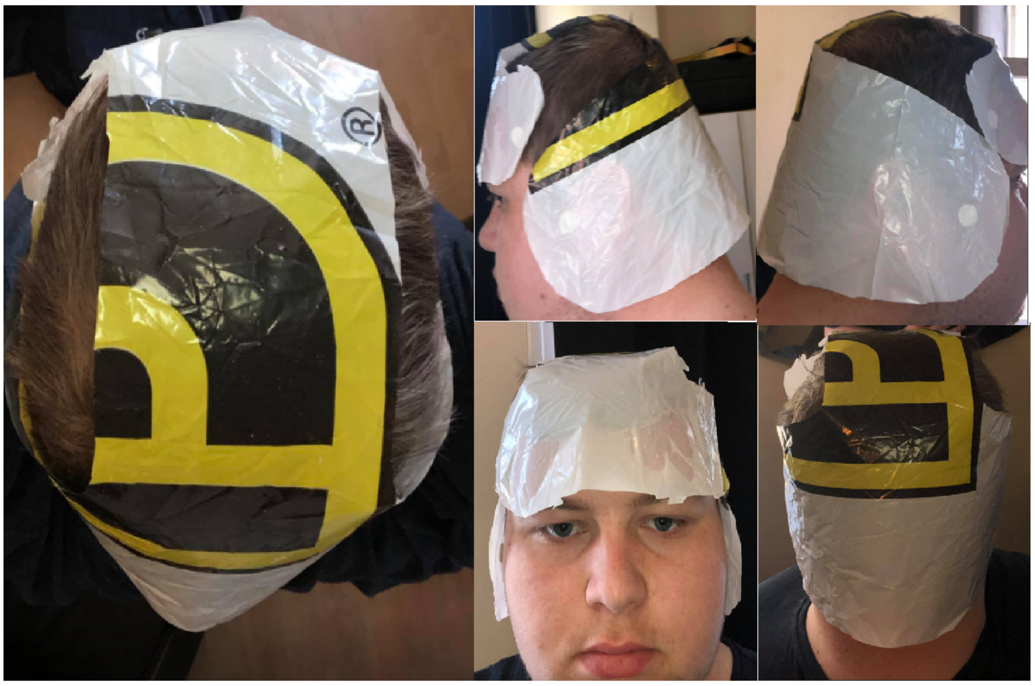


Figure 4.14: The first template on Halvors head from different angles

At first sight the template might look a little big, especially near the neck. The middle part going from front to back also looks wider than it needs to be. This is not a big concern as making the inflatable smaller can be easily done at a later stage by just melting more of the tarp together. The template is very useful to make sure that the pieces cut from the available tarp are big enough. Pieces that are too small will just end up being waste.

Production

The inflatable is made out of two pieces of a tarp which are melted together in the desired shape using a household iron. To avoid ruining the iron and the ironing board with melted plastic, sheets of baking paper was put between the board and tarp, and the tarp and iron. Some testing was required to figure out how much heat and pressure was necessary to melt the pieces properly together. Using the highest setting on the iron while applying medium pressure resulted in a great weld.

The object of the inflatable is to keep the head completely still within the helmet. If the inflatable only relies on being squeezed between the helmet and the head to stay in place, there is a good chance of it moving around inside the helmet. Using glue or some other permanent solution is unfavourable as it would make it significantly harder to make changes to the inflatable, or to replace it if it gets damaged.

To avoid the inflatable sliding around inside the helmet, multiple strips of velcro was used. The velcro strips ensures that the inflatable stays in place, but also allows for easy connection/disconnection of the inflatable. This is also great if several inflatables are made, and there is a need to quickly change between them.

4.1.3 Magnetic connection

The robot is theoretically able to provide forces that can be very damaging to the human neck, especially if there is a pre-existing injury which is very probable for anyone using this robot. Connecting the helmet directly to the robot using screws is in this case not feasible. There needs to be a connection that can be connected/disconnected regardless of the surrounding conditions. Any mechanic connection or a connection dependent on electricity is therefore not feasible. The easiest and most obvious solution is to use magnets.

The idea is to have one part containing magnets that can be fastened to the robot with screws, and one part containing magnets that can be connected to the helmet. For magnets to attract each other they need to have opposite polarity, so it is important to flip the magnets accordingly.

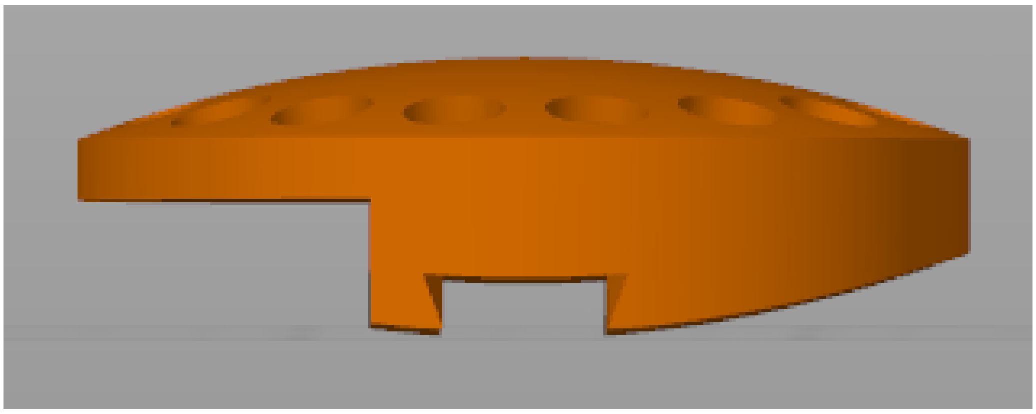


Figure 4.15: The helmet part of the magnetic connection seen from the front. The opening at the bottom is for the slider, and the opening to the left is where the wedge is pushed in.

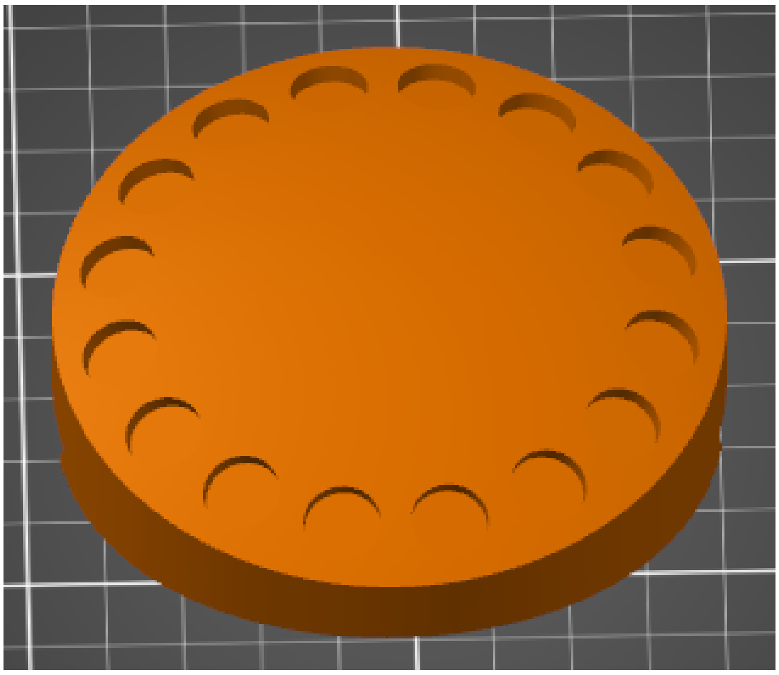


Figure 4.16: The helmet part of the magnetic connection seen from above. The holes are for the magnets.

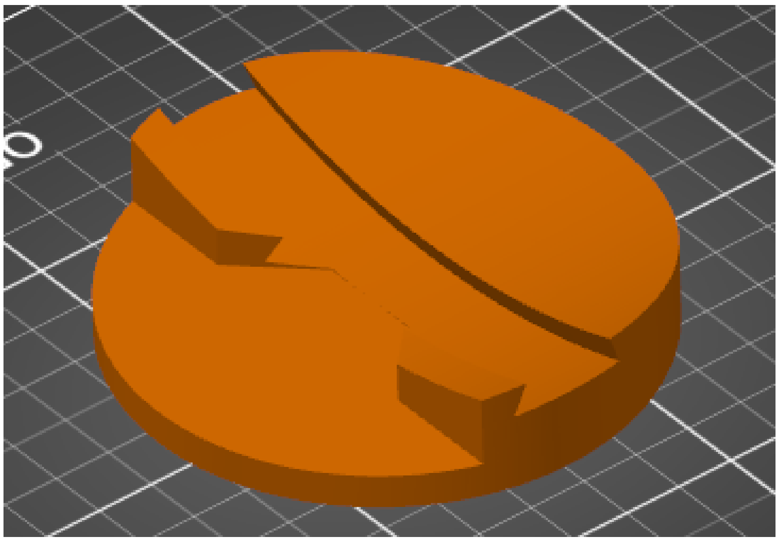


Figure 4.17: The helmet part of the magnetic connection seen from below. The wedge is pushed in from the bottom left side.

Connecting the magnetic connection to the helmet

The part with magnets that is not connected to the robot needs a way to be attached to the helmet. A solution that allows for some adjustment on where the part with magnets is attached to the helmet is ideal. An idea is to have a slider on top of the helmet that can be attached to the part with magnets. The part with magnets and the slider will be attached to each other using a device resembling a wedge which is pushed in from the bottom left side when seen as 4.17. This slider will allow the physical therapist to tilt the helmet according to what is best for each individual patient.

3d-printing

The parts for the magnetic connection, the slider and the wedge are all 3d-printed. The wedge is a very simple part without small, very detailed areas, or any overhangs. This makes for a straightforward print and figure 4.18 shows how the finished print turned out.

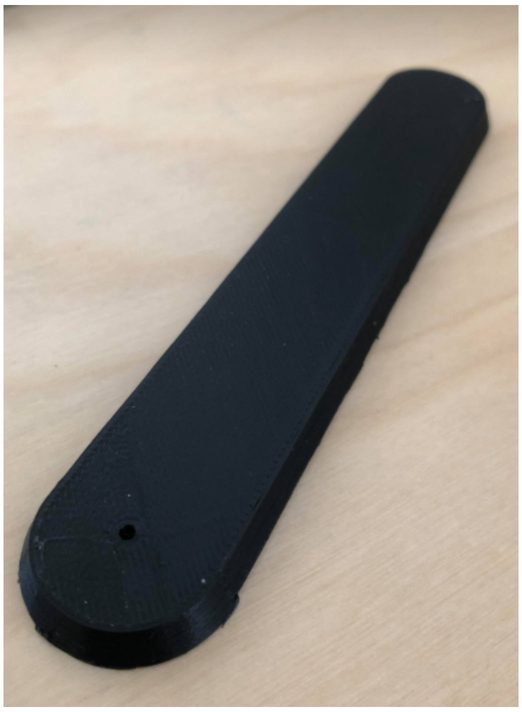


Figure 4.18: The wedge used to lock the helmet to the magnetic connection

The slider is a bit more complicated as it is bent to fit the top of the helmet, as well as having some overhang to fit the slot seen at the bottom in figure 4.15. To make the print less complicated and avoid using support structures, the slider was split down the middle and printed as shown in figure 4.19.

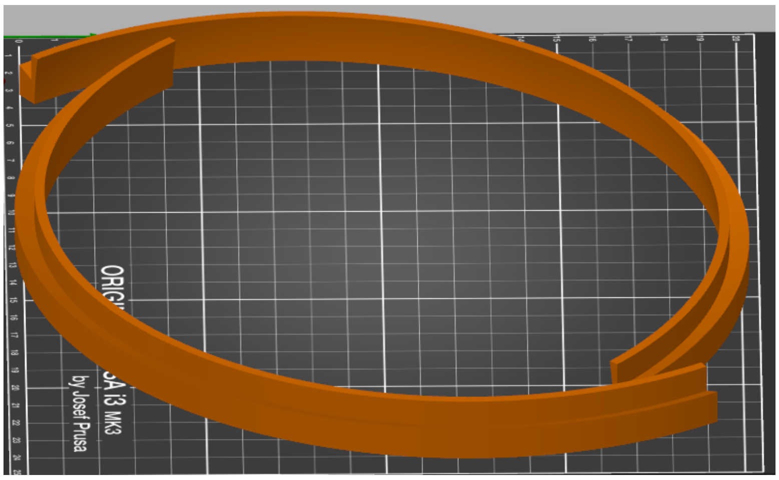


Figure 4.19: The parts that make up the slider

This eliminates the use of support structures as well as giving a nice, flat surface area to use as the base of the print. The two pieces were later glued together with super glue. The part shown by figures 4.15 and 4.16 is the most difficult part to print. It has no obvious side to use as a base since the top part is convex with holes, and the bottom part consists of multiple elements. The part was first printed with the bottom side as the base, using support structures where it was needed. Support structures are usually easy to remove, but that was not the case with this print. The main problem was the leftover support structure inside the slot for the slider. Even after several hours of picking tiny pieces of support out with pliers, the slider would not fit. When the slider finally was able to fit, it got stuck which resulted in the part breaking while trying to pull the slider out. These struggles led to the idea of just splitting the part horizontally, and glue the pieces together. Figure 4.20 shows where the part was split.

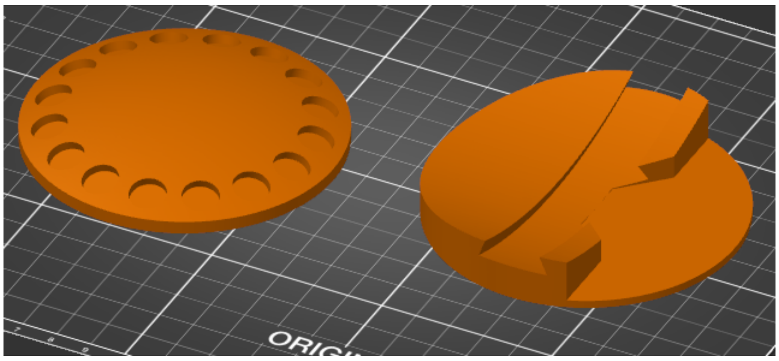


Figure 4.20: The helmet part of the magnetic connection split horizontally

However, when printing these pieces there was some problems with the 3d printer. The flat area of each part stuck to the printer plate in stead of the rest of the part. Figure 4.21 shows how the parts looked. Printing multiple parts simultaneously on one printer have not been an issue previously, but to simplify the printing process the parts were split into two entirely separate print files.

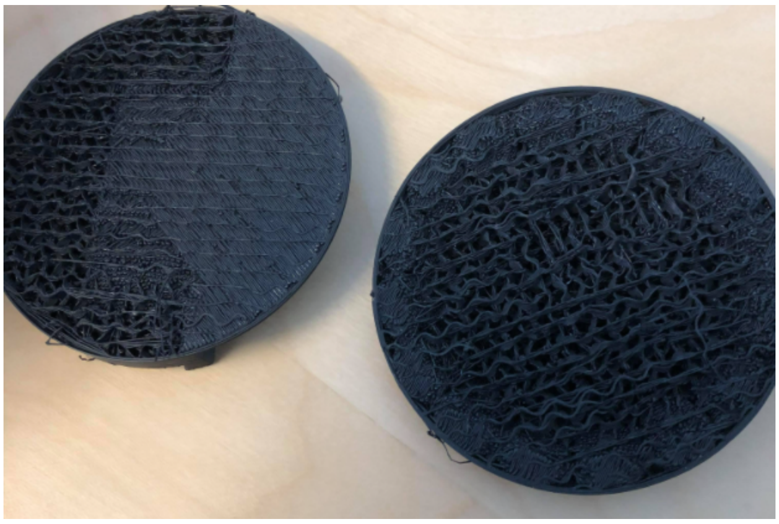


Figure 4.21: Failed print of the helmet part of the magnetic connection

4.2 Graphical User Interface

For the development of the Graphical User interface a JavaFX project was chosen. The main reason it was chosen is bacause of the cross platform capabilities. JavaFX runs on the Java Virtual Machine which is supported by all sort of operating systems and

To write the code for the graphical user interface the Eclipse IDE was chosen. Eclipse is an integrated Development Environment (IDE) designed for programming Java. IDEs are software that helps your programming by providing features like code editing, automation of building and running code, and debugging tools to help keep your code clean and correct. Eclipse was chosen because it provides Java specific features like automatic method generation, class inheritance etc. It also has support for git and Scenebuilder. cenebuilder is a graphical tool for JavaFX programming. It allows generation of fxml files by placing components in position using the SceneBuilder Interface.

4.2.1 JavaFX

JavaFX is an open-source platform for creating graphical user Interfaces based on java. It is based around fxml files which is their own version of Extensible Markup Language (XML). This fxml defines how the window should look, and is connected to a controller. The controller implements the desired logic in the user interface. Below is a a brief introduction to how a javafx program works. First a few definitions:

Stage
"The JavaFX Stage class is the top level JavaFX container. The primary Stage is constructed by the platform. Additional Stage objects may be constructed by the application." **Scene**
"The JavaFX Scene class is the container for all content in a scene graph. The background of the scene is filled as specified by the fill property."

JavaFX Application
In a JavaFX application, the foundation is the applications class. All JavaFX applications extends or inherits from this class. In the application class the main stage is created. The stage is basically the window of the application. In the window (or stage) as single scene is displayed at a time.

Passing Information Between Scenes

Since each scene is defined by its own fxml file with separate controllers, there is no shared information between the different scenes. When shared information is required at a scenes witch, this information must be passed from the current scene to the next. In the listing below is a method that switches the scene to the start page. This is the method that is called when a user logs out. No information is passed in this example.

Listing 4.1: Scene switch without passing information

```
1  public void changeSceneStartPage(ActionEvent event) throws IOException {
2      Parent parent = FXMLLoader.load(getClass().getResource("startPage.fxml"));
3      Scene scene = new Scene(parent);
4  
5      Stage window = (Stage)((Node)event.getSource()).getScene().getWindow();
6      window.setScene(scene);
7      window.show();
8  }
```

When information needs to be passed between two scenes, the controller of the next scene needs to be accessed before the scene switch takes place. IN the listing below is an example of a method that changes the scene to the user page from the login page. This is the method that is called when a user logs in to the system.

Listing 4.2: Scene Switch with information passing.

```
9  public void changeSceneUserPage(ActionEvent event, String user) throws IOException {
10     FXMLLoader loader = new FXMLLoader();
11     loader.setLocation(getClass().getResource("userPage.fxml"));
12     Parent parent = loader.load();
13 
14     Scene scene = new Scene(parent);
15 
16     UserPageController controller = loader.getController();
17     controller.initializeUser(user);
18     Stage window = (Stage)((Node)event.getSource()).getScene().getWindow();
19     window.setScene(scene);
20     window.show();
21 }
```

When you compare the two methods it should be noted that line 2 in listing 4.1 is equivalent to the lines 10-12 in listing 4.2. The reason this line is expanded when information needs to be passed is to give access to the loader object. From the loader object the controller for the new stage can be fetched. This allows any method from the new controller to be executed with parameters stored in the old controller object.

4.2.2 3D visualization

Data flow

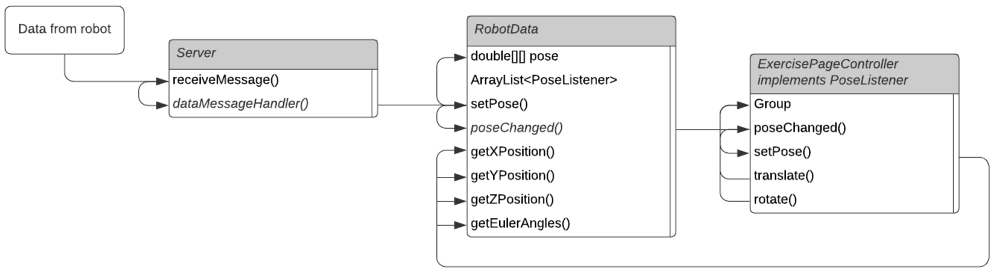


Figure 4.22: Data flow and method calls for the 3D visualization on the exercise page

The figure above shows a visualization of the information flow and method calls used for the visualization of the motion of the robot on the run exercise page of the GUI. Starting from the position data generated in the robot control loop, this data is received by a static TCP server which is instantiated by the Application class of the user interface. This server is static so that it can be accessed by all the other classes without being declared and initialized for every single class. The server contains a RobotData object, which is a class created to store the current state of the robot, as well as some methods for manipulating the data. When the server receives a message containing robotData it calls the dataMessageHandler method, which again calls the appropriate methods for updating the current state of the RobotData object. More details on how the server operates will be discussed in section 4.2.4.

To ensure that the 3D visualization is updated with the new data, the Observer pattern was implemented. The observer pattern is a technique in object oriented programming to ensure consistency between different objects. The reason the observer pattern was chosen is that it is a simple and efficient event driven implementation that eliminates the need for loops to poll the data for changes. In the observer pattern one object is declared as Observable, and then other objects can register as Observers. When the observable object changes state, the Observers are notified of the change and can call methods to handle the change of state to ensure internal and external consistency. The observable object needs to implement methods for adding and removing observers, and methods for notifying the observers whenever the state changes. The observer must implement methods for handling the change of state. In the example shown in the figure PoseListener is an interface for classes to observe the pose variable of the RobotData class. This interface forces the classes to implement a poseChanged method that handles the change of state. The RobotData object maintains a list of observers to notify whenever the pose changes by calling the poseChanged method.

When the poseChanged method is called from the controller, the controller object calls the setPose method. This is the method that updates the 3D view with the new data. The setPose method fetches the pose of the robot from the RobotData class and generates the transforms that are to be applied to the group. The group is a container node where graphical models can be added. When a transform is applied to the group, it is applied to all the models contained within. In the next section the mathematics of the transforms will be discussed further.

Transformation mathematics

In this section the mathematics behind the transformations used in the 3D visualization will be discussed. The RobotState class from libfranka dictates the possible values that can be read from the robot. In order to obtain the pose of the robot there are 3 basic options.

1. Reading the joint variables directly
2. Reading the calculated pose in the base frame
3. Reading the calculated pose in the end-effector frame

Reading the joint variables directly is the only way to get a full understanding of how the robot arm is configured. With 7 degrees of freedom there are theoretically an infinite number of possible configurations to achieve most poses (Exceptions are singular configurations). However when it comes to 3D visualization the only interesting parameter is the pose of the end-effector pose. Option 2 was therefore chosen for this application, because it gives all the necessary position data with respect to a stationary frame with the least amount of necessary pre-processing.

The pose is represented as an array of 16 doubles that correlates to a homogeneous transformation matrix. A homogeneous transformation matrix is a 4×4 matrix of the form

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$$

where R is a 3×3 Rotation matrix representing that orientation, and p is a vector representing the position of the end-effector frame.

While this representation is not a minimal one, it was still decided to transfer the whole transformation matrix over the socket server. The matrix requires twice as much data to be transferred, while it is a 4×4 matrix, the bottom row does not change for any valid transformation matrix. But the processing needed to transfer twice as much data comes cheap compared to the amount of processing that would need to be performed to get a minimal representation and back again to the transformation matrix. Because the Homogeneous is very useful for the mathematical operations performed to generate the transforms in the visual representation in the GUI.

The reason it is useful for this application is because of some of the properties of the rotation matrix:

1. Basic rotations can be combined into a single rotation matrix using matrix multiplication
2. The rotation matrix is orthogonal. This means that the inverse of the rotation matrix R^{-1}, which corresponds to a rotation in the opposite direction about the same axis, is equal to the transpose of the matrix. $R^{-1} = R^T$

Using these properties, finding the transformation matrix which corresponds to the transformation from one pose to another becomes very easy to calculate. Starting with two homogeneous transformation matrices T_{01} and T_{02} which represent the two poses by defining the transformation from the base frame (or the 0-frame) to the frame 1 and 2 respectively.

$$T_{01} = \begin{bmatrix} R_{01} & p_1 \\ 0 & 1 \end{bmatrix} \quad T_{02} = \begin{bmatrix} R_{02} & p_2 \\ 0 & 1 \end{bmatrix}$$

To find the transformation from pose 1 to pose 2, the problem can be decoupled into a translational transform and a rotational transform. Generally for rotation matrices we have that

$$R_{ab}R_{bc} = R_{ac}$$

so to find the rotational transformation between two points both given in the base frame

$$R_{12} = R_{10}R_{02} = R_{01}^{-1}R_{02}$$

To find the translational transform all you need to do is get the distance between the origins of the frames along each of the three axes. This along with the previous statement that $R^{-1} = R^T$ gives the following expression for the transform between two poses given in the same base frame

$$T_{12} = \begin{bmatrix} R_{01}^T R_{02} & p_2 - p_1 \\ 0 & 1 \end{bmatrix}$$

This result gives a fast method to calculate the transformations that are necessary to display the previews of the exercises as animations in the graphical user interface.

Implementing the 3D graphics

The implementation of the 3D graphics starts with the subScene. A subScene is an object that can be placed within a scene that is specifically designed for application where there is a mix of 2D and 3D elements within a user interface. The subScene is basically a container that allows a part of the screen to be used as a separate scene. The subScene is declared in the FXML file with dimensions and an fxid, which is an injectionable identifier that can be used to reference FXML elements from the controller. To this subscene a camera and a model is added. These are all classes that are part of the JavaFX library [13]. The model is initially placed in the middle of the subscene with the camera facing it directly.

During the development process this was first tested by using a square box as a representation of the head because it was easier to implement. In order to get a 3D model of a head into the subscene, a model and a modelImporter is required. Interactive mesh have developed great open-source libraries for importing 3D models into JavaFX [14]. These libraries provides loaders for a variety of file formats used for 3D modelling. The model used during testing and development of the 3D graphics were provided with a free license from the site turbosquid.com [15]. Being a free model it was not the greatest quality, but it was suitable for testing at this stage of the development. The model was downloaded and run through blender [16], a 3D modelling software. This was done to generate a .mtl file with the textures required to load the model.

Listing 4.3: The initialize method for the run Exercise page. Lines not related to the model view have been removed from this listing.

```
    @FXML
    public void initialize() {
        ObjModelImporter modelImporter = new ObjModelImporter();
        modelImporter.read("src/devomed/male_head.obj");
        Node[] objMesh = (Node[]) modelImporter.getImport();
        modelImporter.close();
        modelGroup = new Group(objMesh);

        camera = new PerspectiveCamera();
        camera.translateXProperty().set(-subscene.getWidth()/2);
        camera.translateYProperty().set(-subscene.getHeight()/2-10);
        camera.translateZProperty().set(1100);

        subscene.setRoot(modelGroup);
        subscene.setFill(Color.SILVER);
        subscene.setCamera(camera);

    }
```

Listing 4.3 shows the initialize method for the run Exercise class, similar code is in the initializer for all the pages that contains the 3D view. This method uses the object Model loader to import the object files into a mesh, the mesh is added to a group which is added to the subscene. A group is a container in JavaFX that can contain models, and is useful because then the transformation can be applied to the group rather than to every single individual node in the mesh. A perspective camera is also added to the scene. The position of the camera is selected so that the head is in view under all normal operation of the rehabilitation apparatus. The subscene is then instantiated from the group and displayed on the screen.

Calculating the transformations

With the subscene in place, all that is necessary to animate the model is to apply transformations to the model. The input from the server is a homogenous transformation matrix, and that matrix needs to be converted to a set of transformations that will result in the same movement. Any composite rotation can be represented by 3 coordinates [17]. In this implementation the ZYX euler angles was chosen as the representation, as that is a common representation used in robotics. ZYX euler angles represents a composite rotation by first rotating about the body-fixed Z-axis, then the body-fixed Y-axis, and finally a rotation about the body-fixed X-axis. A formula for calculating the angles around the individual axes needed to achieve the composite rotation can be found by multiplying the elementary rotation matrices and comparing them to the desired rotation. A method to compute the euler angles from the transformation matrix was implemented as shown in listing 4.4.

Listing 4.4: The computation of the euler angles from the rotation matrix

```
    protected double[] getEulerAngles(double[][] pose) {
        double[] eulerAngles = new double[3];
        eulerAngles[0] = getYaw(pose);
        eulerAngles[1] = getPitch(pose);
        eulerAngles[2] = getRoll(pose);
        return eulerAngles;
    }

    protected double getRoll(double[][] pose) {
        return Math.atan2(pose[2][1], pose[2][2]) * (180.0/Math.PI);
    }

    protected double getPitch(double[][] pose) {
        return Math.asin(pose[2][0]) * (180.0/Math.PI);
    }

    protected double getYaw(double[][] pose) {
        return Math.atan2(pose[1][0], pose[0][0]) * (180.0/Math.PI);
    }
```

Roll, pitch and yaw are the terms used for the rotation angles about the x-, y-, and z-axes respectively. The homogenous transformation matrix can now be represented as a translation in 3 dimensions along with a rotation that is represented as 3 rotations about the elementary axes. To apply these transformations to the model the set Pose method is created

Listing 4.5: The setPose method

```
    public void setPose(double x, double y, double z, double[]
       eulerAngles, Group group) {
        group.getTransforms().clear();
        translate(x*100, y*100, z*100, group);
        rotate(eulerAngles[0], Rotate.Z_AXIS, group);
        rotate(eulerAngles[1], Rotate.Y_AXIS, group);
        rotate(eulerAngles[2], Rotate.X_AXIS, group);
    }
```

Listing 4.5 shows the implementation of the setPose method, which applies the transformations to the group. This method is called by the poseListener so it is called whenever the pose variable in the robotData class changes. For the exercise previews the same method is used, except instead of an imput matrix from the server, the setPose method iterates over the Path.

4.2.3 Database

SQLite

As discussed in the project thesis, SQLite was chosen as the tool for setting up the database required for the implementation of the Graphical User Interface. SQLite has easy to use libraries for both C++ and Java. Using SQLite in the project is as simple as downloading the SQLite jar file and including it in the eclipse classpath for the project. In addition to the jar, DB browser for SQlite was also used. This is a graphical tool for manipulating the database file. This software allows setting up the database structure with tables and keys in a simple graphical interface.

The database is stored in a single database file on disk. SQLite allows multiple threads or processes to access this database file simultaneously, but access is automatically locked when a thread is making changes to the database by using mutexes. This means SQLite is both thread-safe and parallel-safe, allowing both single process and multi-process concurrency.

Database structure

The database is structured into tables, each containing a primary key, some data fields, and foreign keys. In this context keys are unique integers that acts as an identifier for a specific entry in a table. Using keys means that you can store different objects that have the same state while still being able to identify them when necessary. The keys are generated by a random number generator that produces an integer with 8 digits. The key generator is implemented as a static method in the database class, and all other classes call this method in the constructor when they are created for the first time. On subsequent constructions the ID numbers are loaded from the database and passed to the constructor so that the keys remain unchanged in new sessions. The keys are classified as unique in the database configuration, so an exception will be thrown in the unlikely event that a generated key already exists in the table.

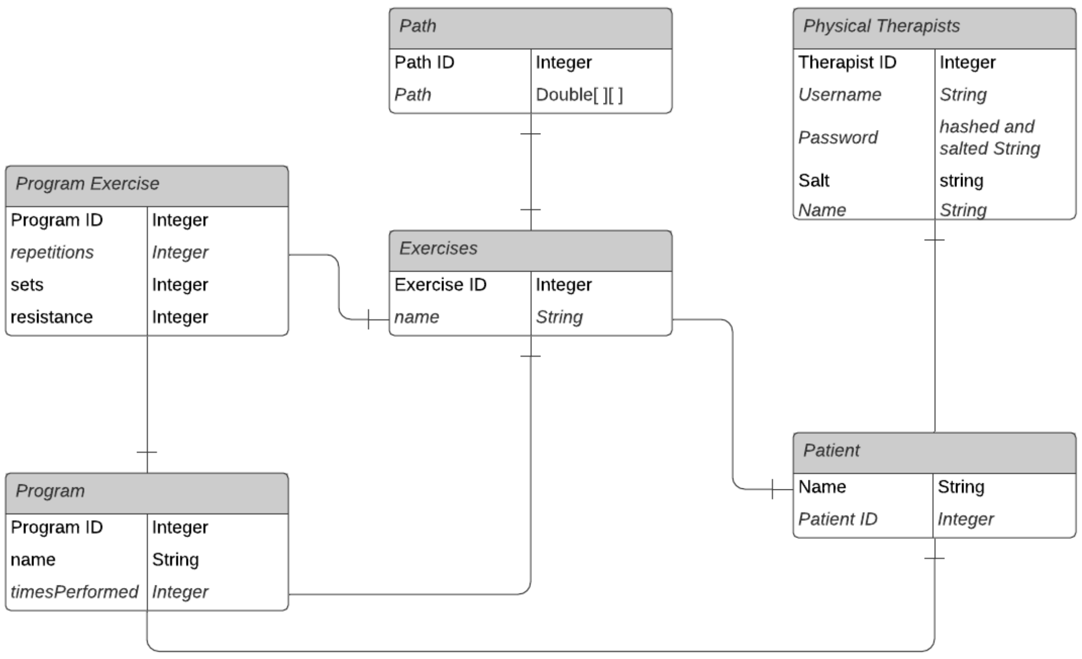


Figure 4.23: Database ER Diagram

The structure of the tables in the database is illustrated in the entity relationship diagram shown in figure 4.23. In this diagram each block represents a table and their columns. The lines between the blocks represent a foreign key present in the table. For example the programs table contains 5 columns: the primary key, a program name, a program counter that keeps track of how many times the program has been performed, and 2 foreign keys which reference a specific entry in the Exercise and Patient tables.

Reading and writing to the database

Upon a successful login, the databases needs to be loaded into objects that can be accessed through the GUI. To implement this a database class was created to contain the methods necessary for accessing the database and loading all the necessary data. Because the whole database file is locked for writing when a change is made, it is also important to always close the connection after a change is made. Because of this the basic structure of reading and writing to the database is as follows:

1. First the constructor is called to create a database object and establish a connection to the database file.
2. The Query or Update is executed
3. The destructor is called to close the connection and destroy the object.

To execute the querys and updates methods were created in the database class that takes a string as input and converts it to an SQL command, these methods are used in other classes to save the object state so that it can be retrieved later. The execute update method is used when an existing database entry is changed, and the execute Query method is used when a database entry is fetched, removed or added. These methods are both part of the SQLite JDBC driver. A set of methods is also created to load all the objects, starting with the Therapist who is logging in. These method create a chain that results in a therapist object where all the relevant contained objects are loaded, all the way down the chain. The primary method for loading the data is shown in listing 4.6.

Listing 4.6: The primary method for loading the data from the database upon a successful login

```java
    public Therapist getTherapist(int userID) {
        Therapist therapist = null;
        try {
            ResultSet rs = executeQuery("SELECT * FROM therapists
               WHERE id="+Integer.toString(userID));
            if (rs.next()) {
                therapist = new Therapist(rs.getInt("id"), rs.
                   getString("username"), rs.getString("password"),rs.
                   getString("name"));
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        addPatients(therapist);
        return therapist;
    }
```

This method generates a query that searches the therapist table for a user with the given user ID input. The query returns a result set that is used as input when the therapist constructor is called to create the therapist object with the state defined in the database. When the therapist object is created the next method is called, which searches for all the patients with the therapist ID registered as a foreign key in the table entry.

Listing 4.7: The next method in the loading chain, which populates the List of patients in the therapist class.

```java
    public void addPatients(Therapist therapist) {
        try {
            ResultSet rs = executeQuery("SELECT * FROM patients WHERE
               therapist="+Integer.toString(therapist.getUserID()));
            while (rs.next()) {
                Patient patient = new Patient(rs.getInt("id"), rs.
                   getString("name"), rs.getString("info"), therapist)
                   ;
                therapist.addPatient(patient);
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        for (Patient patient : therapist.getPatients()) {
            addExercises(patient);
        }
    }
```

Listing 4.7 shows the next method in the loading chain. Similarly to the previous method it executes a query that returns all the patients registered to the therapist, and then calls the next part of the chain which loads the registered exercises for all the patients. The chain then continues by iterating over the exercises to add the paths, programs are added to the patient, and program exercises are added to the programs. By using this loading chain all thte relevant information from the database is loaded by a single method call in the login page controller.

The full code for the database class can be found in appendix **??**.

Personal Information

After a conversation with Morten it was decided that there is no need to store any personal information in the rehabilitation apparatus database. Firda Fysmed already has a database where they store the relevant information about their patients. Because they run a full physical therapy center, not all patients will be using the robotic rehabilitation apparatus. It will therefore be impractical to replace their current system for patient information with the rehabilitation apparatus database. Any patient information will therefore either be redundant, or information is stored in separate locations for different injuries which introduces an unnecessary complication to their operation. This also simplifies the security requirements for the database. No sensitive personal information is stored, and the names are the only data that are considered personal information.

Password security

The next issue to be faced is that the database needs to store usernames and passwords. That in and of itself is not a difficult task but it raises some security concerns that are important to manage. Even if you do not care that someone gets access to the system, keeping passwords hidden is vital because according to a survey done by google in cooperation with Harris Poll in 2019 [18], 65 % of users reuse passwords on multiple accounts. Getting access to a database where passwords are stored in plain text can then be used to get access to other accounts with more value and importance. It was therefore decided to implement some security measures to prevent anyone from getting access to the passwords in the database.

One possibility is to encrypt the passwords before they are stored in the database, however there are several problems with this approach. Most important of these issues is that if using encryption, the password needs to be decrypted to verify login information. That means you need encryption keys which needs to be stored somewhere, and that key needs to be stored securely. Essentially just moving the goalpost.

Another possibility is to use a hashing algorithm. A hashing algorithm is an algorithm that turns a password into a seemingly random string of characters, much like encryption. Hashes however are very difficult to reverse even if you know all the parameters of the hashing algorithm. To verify that the inputted login information is correct, the input is run through the same hashing algorithm, and then the results are compared. The same input string will always produce the same output, but otherwise the outputs are unrelated. For example if you give the inputs ”1234” and ”12345” into the same hashing algorithms the hashes would be completely different, and there would be no way to analyze the hash to find out that the inputs were similar.

A common way to try to defeat hashing algorithm is to produce dictionary tables or ”rainbow tables” as they are commonly called. These tables allow you to search for a hash, and find the plaintext if the password exists as one of the precomputed items in the table. To combat this hashing algorithms often use salt. Cryptographic salt is another string that is added to the password before the hash is computed. This renders rainbow tables extremely inefficient as they would need to compute a separate table for every combination of password and salt. Since the salts only task is to prevent dictionary attacks, it is not necessary to keep the salt secret, it can be stored in plaintext along with the password

Hashing the password was decided to be the best option for this application. BCrypt is a hashing algorithm developed by Niels Provos and David Mazières in 1999 [19]. An open-source implementation of this algorithm created by Damien miller was chosen to encrypt the passwords in the database. BCrypt was chosen because it is a slow hashing function. By slow it is meant that the hashing in itself requires a significant amount of computing, additional computing time can also be added in the constructor so that the algortihm can adapt to faster computers. Usually having slow hashing algorithms is a problem for servers as they need to perform large amounts of hashing to verify all the user logins, but for this application a slow hashing algortihm is ideal because the server only needs to handle a single login at the start of each session, and the heavy computation is an extra layer of security that prevents brute force attacks.

The Bcrypt class was downloaded from his website [20], and included as part of the project. With that class in the project, the hashing is simply a method call with the string as input, and it generates the secure hash string that can be stored in the database. Salt is also generated using a method from that class, and the salt is stored alongside the password in the database so that the same salt can be used to verify the credentials.

4.2.4 Networking

One of the major goals of this project is to move the User Interface to a different computer. As discussed in the specialization project the plan is to achieve this by transferring data over network sockets.

C++ Library

The first step in this process was to decide on a C++ library to use in the robot control side of the application. While in Java there is the java.net package in the standard module for programming with sockets, there is no counterpart in the C++ standard library. Several options were considered and it was decided to use asio. Asio was chosen for several reasons:

1. It is part of the boost library. That means that asio has been reviewed and accepted by the boost organization, which includes members of the C++ standards committee. Boost has a trackrecord of producing excellent libraries, some of which are now part of the C++ standard library.
2. Asio comes highly reccommended by developers .
3. The library is easy to use and the documentation is excellent
4. The library is header-only, which means that no building or linking is necessary.

Establishing Connection

The next step was to establish a network connection between the robot controller and the user interface. A new thread was created in the robot control program called the communication thread. The implementation of this thread was done similarly to the existing threads, a new iteration was added to the thread spawning loop in the main method. This iteration created a new POSIX-thread where the static robot data struct was passed as an argument so that this thread has access to the data. Making the connection is very simple with Asio. Listing 4.8 shows a minimal example of connecting to a socket and sending messages.

Listing 4.8: A simple program that sends data over a socket until the robot shuts dow.

```
1   asio::error_code ec;
2   asio::io_service service;
3   asio::ip::tcp::endpoint endpoint(asio::ip::address::from_string(
        server_ip, ec), 8080);
4   asio::ip::tcp::socket socket(service);
5   socket.connect(endpoint, ec);
6   while (!robot_data->shutdown) {
7       socket.send(asio::buffer(linearization(robot_data)));
8       usleep(300000);
9   }
```

The first five lines in listing 4.8 is the code to create and connect a socket to a server. An endpoint is created with an ip address and a port number. In this example the ip address is defined by the string server_ip, which is set to the ip address of the computer running the GUI on the local network. The port is set to 8080. Which port number is used does not matter as long as it there is no conflict with other processes running on the machine. A TCP socket object is then created and connected to the server endpoint. If there is no response to the server, the object ec will contain an error message.

Lines 6-9 shows how to send a string over the socket. The while loop keeps running until the robot shuts down, and for every loop it sends a string that is returned by the linearization function. After the message is sent, the thread sleeps for 300 ms.

Message Handling

On the server side, the received messages needs to be decoded and processed. In the GUI application a server class was created. this class contains a robot data object and a socket with an input reader. In Java, sockets are part of the Java.net package that is part of the core module. A message handler was written based on a simple protocol that checks the first two bytes of the message to determine what type of message is being sent, and then calls the appropriate method to process the message. The message Handler is shown in listing 4.9.

Listing 4.9: The message handler method in the Server class

```java
    public void receiveMessage() throws IOException {
        String message = in.readLine();
        if (message == null) {
            return;
        }
        String[] splitString = message.split(":");
        int messageType = Integer.parseInt(splitString[0]);
        String content = splitString[1];
        switch(messageType) {
        case 1:
            dataMessageHandler(content);
            break;
        case 2:
            System.out.println(content);
            break;
        }
    }
```

The ”in” object in listing 4.9 is of the BufferedReader class. This object converts the incoming data to a string. The socket receives a stream of bytes which is converted to a character stream with an InputStreamReader, which is then the input to the BufferedReader. The buffered reader gets a line of text from the input stream. This way we are guaranteed to always get an entire message, as the reader will only return the string when it finds a new line symbol ”\n”. The message is then checked for null before it is processed. The string is then split into two parts: the content of the message, and the message type identifier. Based on the message type identifier the handler for that type of message is called.

In the method showed in listing 4.9 there are only 2 types of messages implemented: the data message, and a simple printer that displays the string in the terminal. This method is however easily expandable to include more message type handlers by adding more cases to the switch-case statement. This solution can be extended to handle up to 10 different message types. If more are required then that can be achieved by simply changing the data type of the messageType object to a character. That change will increase the maximum number of cases to over 250, which far exceeds the requirements for this project.

The data

Listing 4.10: The data message handler method in the Server class

```java
public void dataMessageHandler(String message) {
    message = message.replace(",", ".");
    String[] data = message.split(";");
    if(data.length == 16) {
        double[][] pose = new double[4][4];
        int column = 0;
        int row = 0;
        for (int i=0; i<16; i++) {
            pose[row][column] = Double.parseDouble(data[i]);
            row += 1;
            if (row == 4) {
                row = 0;
                column +=1;
            }
        }
        robotData.setPose(pose);
    }
}
```

The pose of the robot is represented in the robotData object as 4x4 array of doubles. The pose data sent over the socket is represented as a string with values separated by a semi-colon. The data message handler shown in listing 4.10 shows the implementation of the method that takes the string input received on the socket and updates the values in the robotData object. This is done by declaring and initializing a new array, splitting the input string at the semicolons, and iterating over the atring array in a for loop to populated the new array. The new array is then passed to setPose method of the robotData class. As discussed in section 4.2.2, from here the listeners are notified of the poseChange and updates the graphical user interface accordingly.

Threading

The next step was to implement a way for the network communication to happen in both direction simultaneously. This was done by adding additional threads, so that for each side of the communication there is one thread that deals with incoming messages, and one thhread that deals with sending information. In the GUI the write Thread is not an own thread per se, but rather methods that are called from the main thread when necessary.

In the robot control program and additional thread was created to handle incoming messages. To avoid any and all problems assosciated with accessing sockets from different threads, a separate socket was created for the read thread that communicates on a different port.

Listing 4.11: The read thread in the robot control program

```cpp
void* communication_read_thread(void* arg) {
    shared_robot_data *robot_data = (shared_robot_data *)arg;

    asio::error_code ec;
    asio::io_service service;
    asio::ip::tcp::endpoint endpoint(asio::ip::address::from_string(
       server_ip, ec), 8081);
    asio::ip::tcp::socket socket(service);
    socket.connect(endpoint, ec);

    if (!ec) {
        std::cout << "Read Connected!" << std::endl;
    } else {
        std::cout << "Failed to connect to address:\n" << ec.message()
            << std::endl;
    }

    while (!robot_data->shutdown) {
        asio::streambuf read_buffer;
        asio::read_until(socket, read_buffer, '\n');
        receiveMessageHandler(robot_data, bufferToString(read_buffer))
           ;
    }
    socket.close();
    return NULL;
}
```

Listing 4.11 shows the read thread of the robot control program. This thread loops for as long as the program is running and reads one line at a time from the input stream and stores it in the read buffer. The method asio::read_until will return an input stream when it reaches a certain character. By setting the parameter to the new line symbol "\n", this methods works much like the readLine method described in section 4.2.4. The input stream is converted to a string and passed to the receive message handler that processes the data.

The message handler currently only supports start and stop tracking commands for creating new exercise paths, but can easily be expanded to handle different message types.

On the server side, a thread was created to read the messages that is continuously sent from the robot control program. In Java threads are created by creating a class that implements the runnable interface. This interface requires you to overwrite the run method that will execute in the thread.

Listing 4.12: The run method from the server read runnable

```
void* communication_read_thread(void* arg) {
    @Override
    public void run(){
        while (!isStopRequested()) {
            try {
                App.server.receiveMessage();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

5. Current Implementation

This chapter will describe the current state of the project. It will explain the functionality and design of the GUI and the different parts of the head mount.

5.1 Graphical User Interface

In order to launch the application, the GUI to launch first to bind the server socket. When the server socket have been bound, the robot control program can be launched which will create a client socket that connects to the server. When the client has connected to the server, the GUI will launch to the start page shown in figure 5.1.

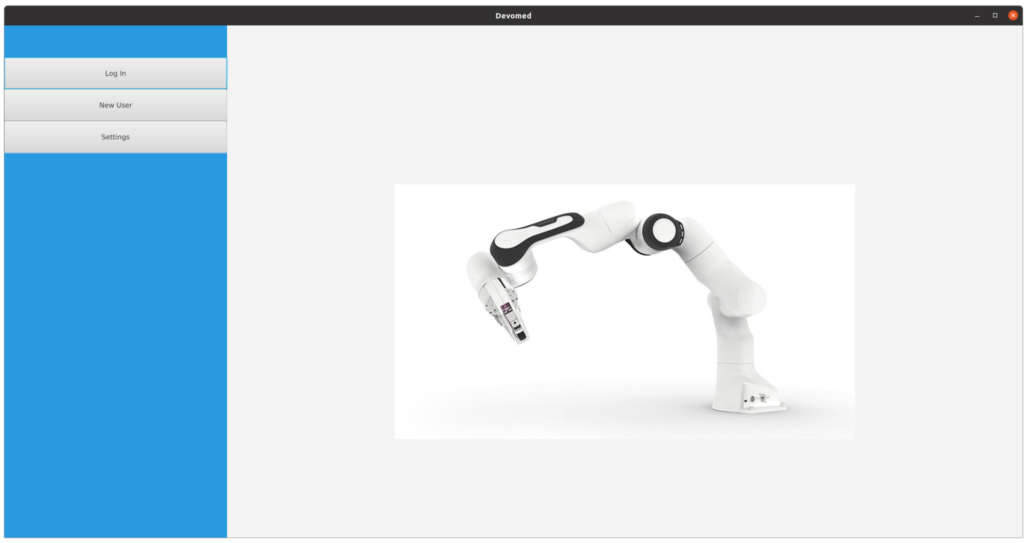


Figure 5.1: The start page of the GUI

5.1.1 Log-in Page

Progress from the start page requires the user to log in. For development purposes it is also possible to create a new user without logging in to the system first, this feature should be removed in further development. The login screen is shown in figure 5.2. It consists simply of two text fields to input a username and password. When the log in button is pressed the controller calls a method that fetches the strings from the text fields, a username and a password candidate. The method then queries the database to check if there exists a user with that username. If the user exists then the password candidate is hashed with the salt stored along the password hash in the database. The hashes are then compared to validate the login credentials. If the username or password is entered incorrectly there is a label that will read ”Login failed” if the hashes do not match.

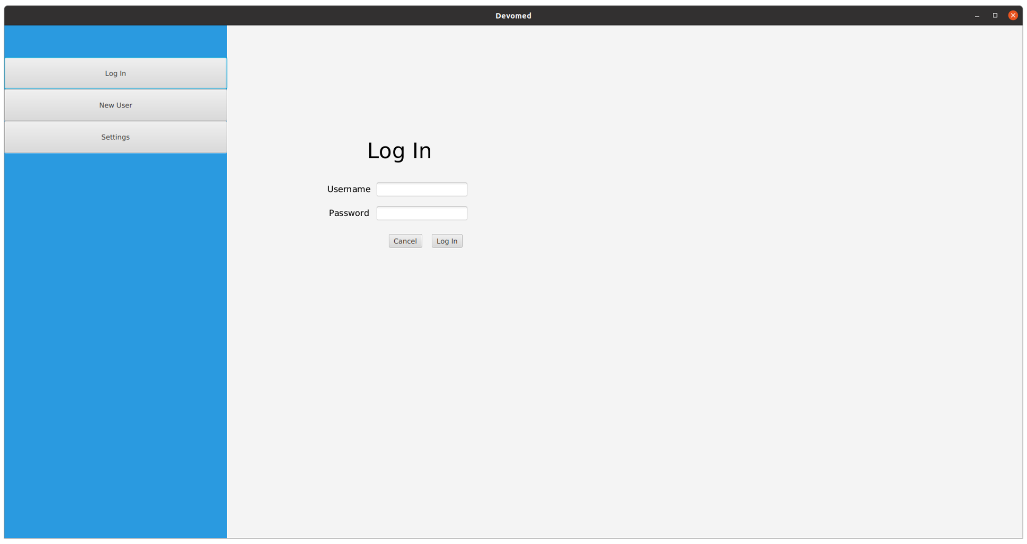
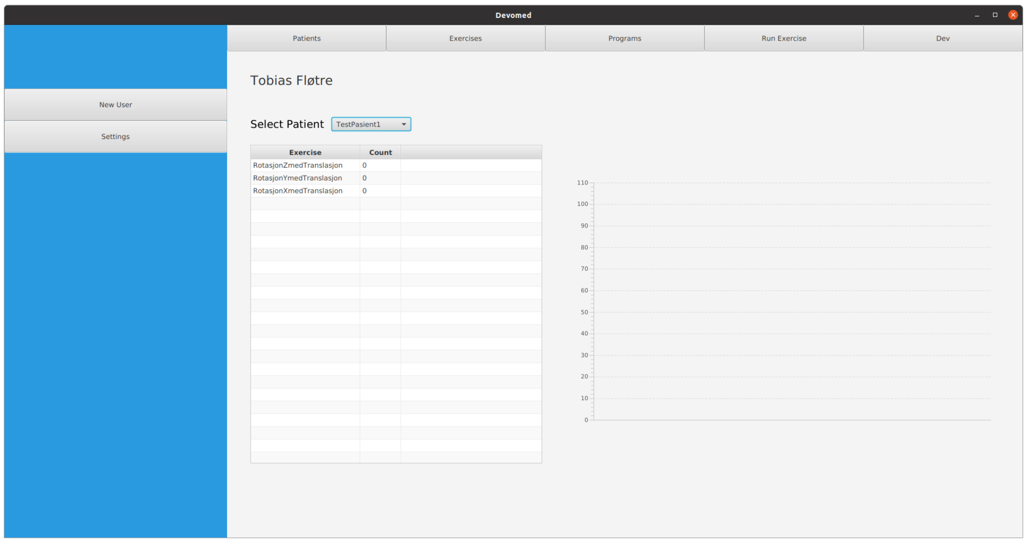


Figure 5.2: The program page of the GUI

5.1.2 Patient Page

Upon a successful login, the user is taken to the patient page. During the initialization of this page the username is passed to the database which calls a series of methods to load all the patients who have this user registered as their therapist, with all the exercises and programs that belongs to the respective patients. This information is then passed along for all the scene transitions until the user logs out. The patient page consists of a choice box, table view and line chart. The choice box allows the user to select the current patients from a list of registered patients. The choiceBox is populated with all the registered patients during the initialization of the scene. A listener is implemented to monitor the selection as it happens, and this listener calls a method to populate the tableView with all the exercises the patient has registered. The line chart is currently not functional, but the purpose of it is to display the patients performance and progression on the selected exercise. Based on this graph the therapist should be able to determine how much force the patient is able to produce in the direction of motion for the exercise.

The Patient page is shown in figure 5.3. The buttons at the top functions as tabs, allowing the user to navigate to the other pages of the GUI. The only limitation is that in order to switch to a different tab a patient must be selected first. If an attempt to switch tabs without selecting a patient is made there is a label next to choice box that will read ”select patient”.



Exercise	Count
RotasjonZmedTranslasjon	0
RotasjonYmedTranslasjon	0
RotasjonXmedTranslasjon	0

Figure 5.3: The patient page of the GUI

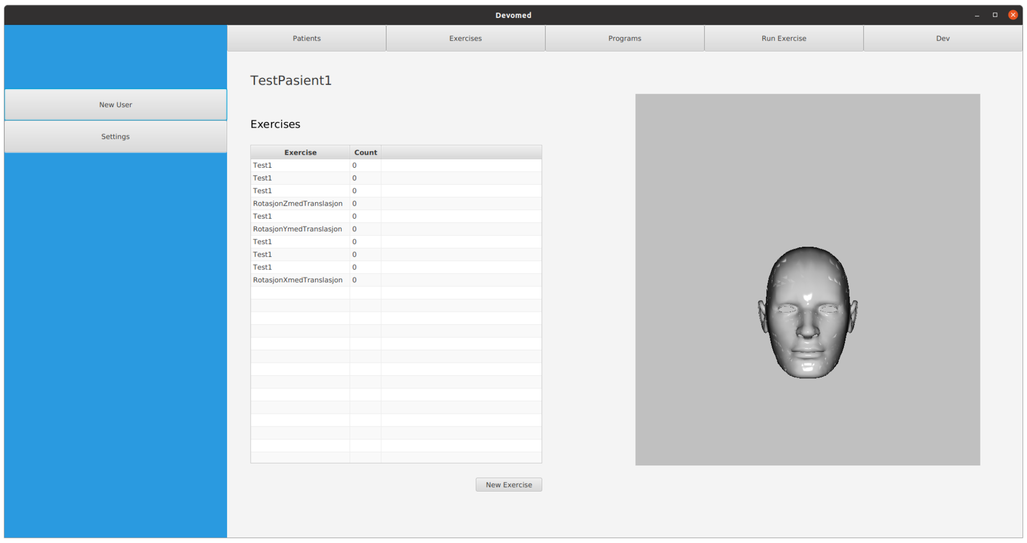
5.1.3 Exercise Page

Figure 5.4 shows the exercise Page of the GUI. The purpose of the exercise page is to display a preview of the exercise before performing it. The exercise page consists of a tableView and a subScene with 3D graphics. In the initialization of this page user and selected patient is passed from the previous page, and the tableView is populated with the list of the current patients exercises. In the subScene a 3D model of a head is loaded and placed in the middle of the subScene along with a perspective camera. A listener is added to the tableView that that detects when an exercise has been clicked in the table.

Whenever an exercise is selected, the selected exercise is fetched from the tableView and the path of that exercise is passed as a parameter to spawn a thread that deals with the animation of the exercise. This animation will play back and forth until a stop request is made, which can occur three different ways:

1. The stage (window) is closed.
2. The scene is changed.
3. The exercise is deselected.

In case of deselection the animation of the new selection will immediately start. There is currently no way to deselect an exercise without selecting another. Below the exercise table there is a button that will take you to the page for creating new Exercises.



TestPasient1

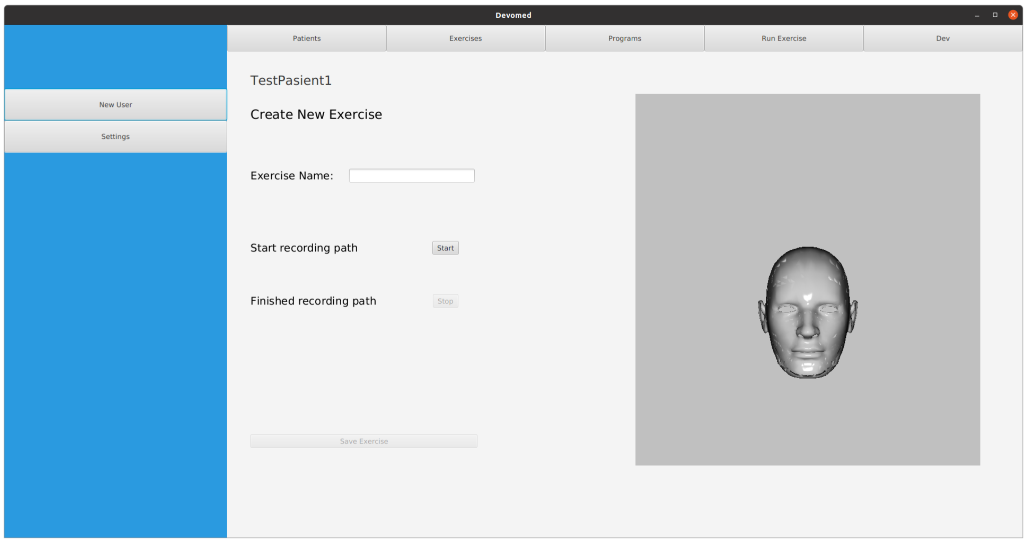
Exercises

Exercise	Count
Test1	0
Test1	0
Test1	0
RotasjonZmedTranslasjon	0
Test1	0
RotasjonYmedTranslasjon	0
Test1	0
Test1	0
Test1	0
RotasjonXmedTranslasjon	0

Figure 5.4: The exercise page of the GUI

5.1.4 New Exercise Page

Figure 5.5 shows the new Exercise page of the GUI. This page allows the user to create new exercises by tracking the movement of the robot and generating a path. When the start button is pressed, a message is sent to robot control program over the network socket that starts the position tracking in the robot control program. This tracking generates a .csv file with the positions of the robot during the tracking period which can be used to create a Path for the new exercise. The file transfer of the logged positions to the GUI system is not yet implemented so creation of exercises is not yet functional.



TestPasient1

Create New Exercise

Exercise Name:

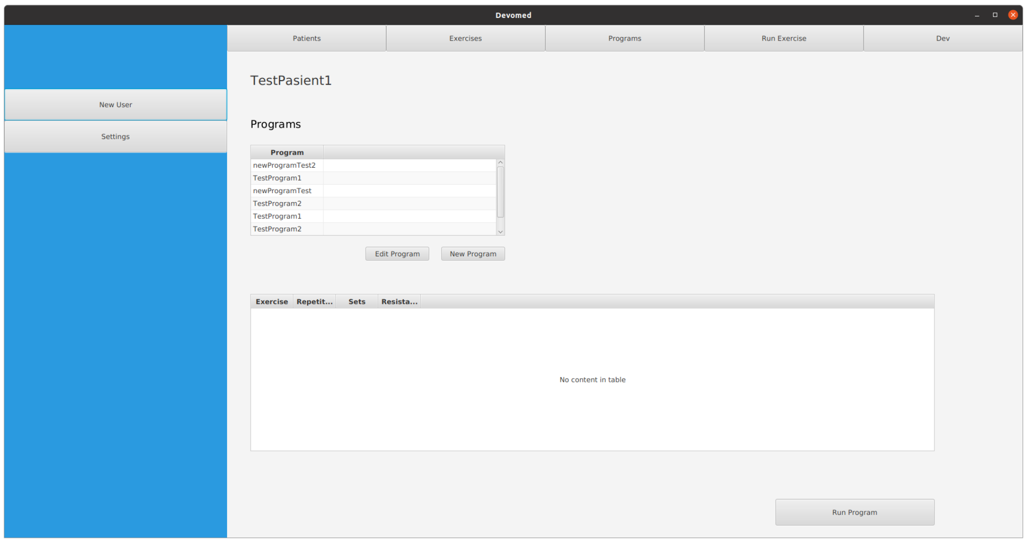
Start recording path

Finished recording path

Figure 5.5: The page for creating new exercises in the GUI

5.1.5 Program Page

The program page shown in figure 5.6. This page allows the user to see a list of the different programs associated with a patient, and what exercises each program contains. The page consists of two tableViews. In the initialization the top tableView is populated with all the programs that belongs to the selected patient. When a program is selected, the listener method populates the bottom tables with the exercises associated with the program. Instead of using the exercises class, the programs contains a class called ProgramExercise. This class contains an exercise as well as some additional information about how the exercises are to be performed. This includes how many repetitions to perform each exercise, how many times each exercise should be repeated, and what resistance the exercise should be performed at. This allows for some functionality around running exercises that automatically increases repetitions or resistance goals each time the exercise is run, while the core exercise object remains unchanged.



Program
newProgramTest2
TestProgram1
newProgramTest
TestProgram2
TestProgram1
TestProgram2

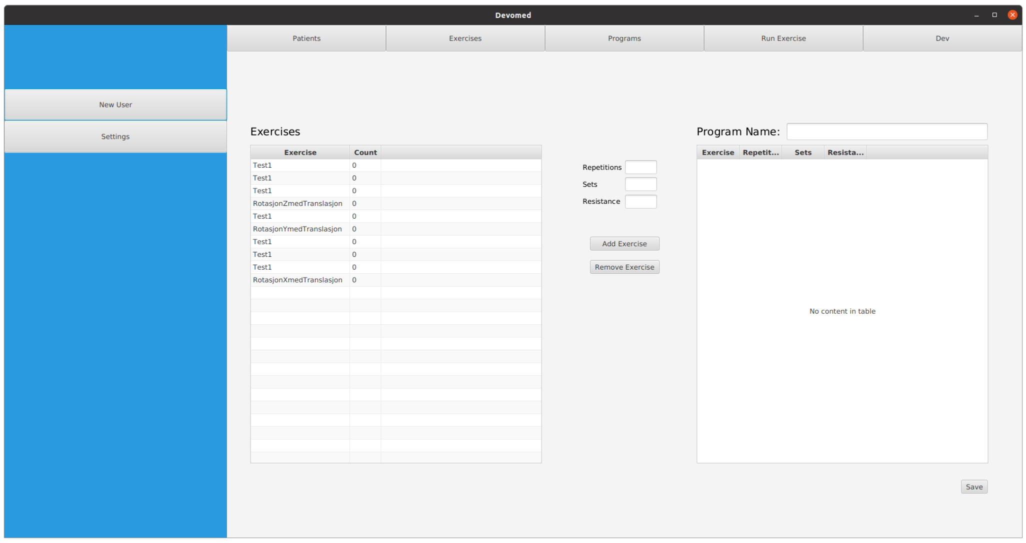
Exercise	Repetit...	Sets	Resista...

No content in table

Figure 5.6: The program page of the GUI

5.1.6 New Program Page

The two buttons on the program page labeled ”Edit Program” and ”New Program” will both bring you to the new Program Page shown in figure 5.7. The difference between the two buttons is that when you click the edit exercise button, the selected program is loaded into the program tableView on the right, while if you click the ”New Program” button, the program tableView will be empty upon initialization. This page allows you to create a new training program from the exercises that are registered to the patient. In the initialization of the page, all Exercises registered to the selected patient will be loaded into the table on the left. An exercise is added to the program by selecting the desired exercise from the table, and filling out the three cells with the desired number of repetitions and sets, and the resistance the exercise is to be performed at. By selecting an exercise from the program table on the right, exercises are removed from the program when the ”Remove Exercise” button is pressed. When the save button is clicked a new program object is created with all the selected exercises, and then the user is returned to the program Page.



Exercise	Count
Test1	0
Test1	0
Test1	0
RotasjonZmedTranslasjon	0
Test1	0
RotasjonYmedTranslasjon	0
Test1	0
Test1	0
Test1	0
RotasjonXmedTranslasjon	0

Exercise	Repetit...	Sets	Resista...

No content in table

Figure 5.7: The page for creating and editing programs in the GUI

5.1.7 Run Exercise Page

Figure 5.8 shows the run Exercise Page. This page contains a subScene similar to the one on the exercise page as well as a slider. On this page the subScene displays the current position of the robot arm with data received on the server Socket. The purpose of this page is to display information on the screen that is useful during the performance of the exercise. This can be both information that is useful to the patient, and information that is useful to the therapist. The page also contains a slider to adjust the resistance of the robot.

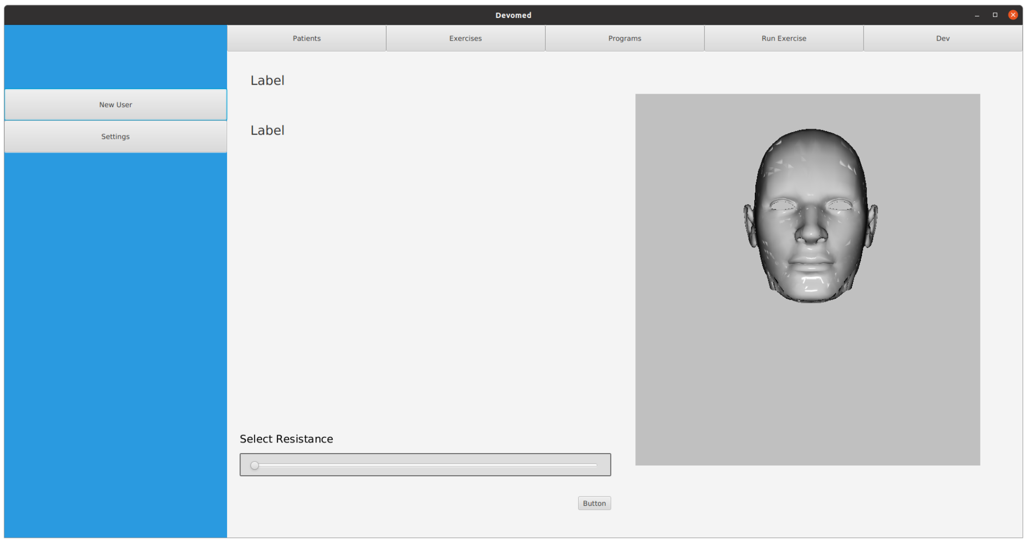


Figure 5.8: The Run Exercise page of the GUI

5.2 Head mount

The head mount consists of a helmet, an inflatable element and the magnetic connection to connect the helmet to the robot. Figure 5.9 shows Halvor using the head mount.

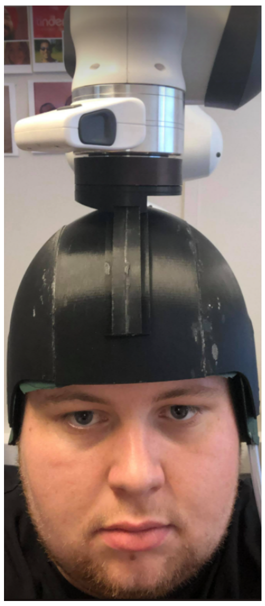


Figure 5.9: The head mount mounted on Halvors head

5.2.1 Helmet

The helmet is made out of five 3d-printed parts which are glued together, with a slider glued on the top as seen in figure 5.10.

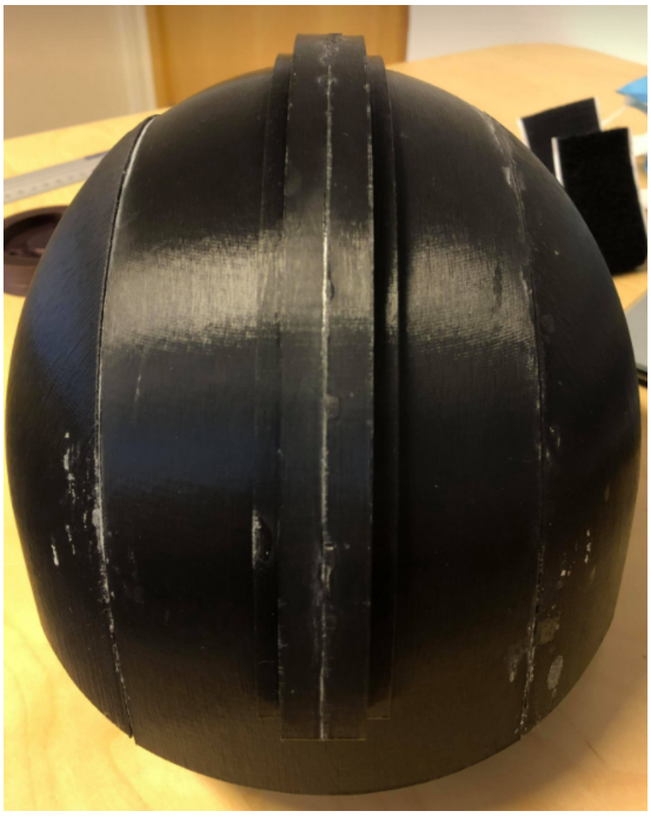


Figure 5.10: The helmet with the slider glued on top

It covers the forehead, the temple, and the back of the head which are the most important areas when performing exercises. The helmet shell is only about 2mm thick, resulting in a lightweight device, only weighing about 0.25kg . Inside the helmet there is 5 strips of velcro to attach the inflatable element. Figure 5.11 shows the placement of the strips.

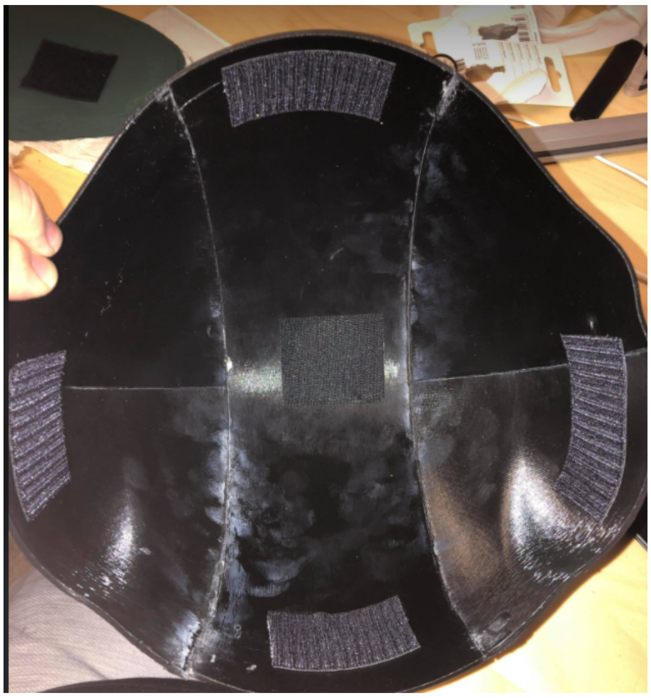


Figure 5.11: Placement of velcro strips inside the helmet

To attach the helmet to the robot, the slider is put in the bottom slot on the part with the magnets and locked in place using the wedge as shown in figure 5.12.

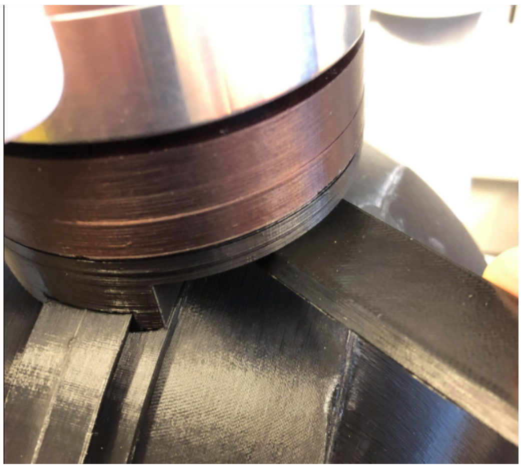


Figure 5.12: The slider on the helmet locked in place with the wedge.

5.2.2 Inflatable element

The inflatable element is made out of a plastic tarp which is cut in the desired shape with the sides melted together to create an airtight seal. A bulb pump is used to inflate it through a thin tube which is glued between the two parts of plastic tarp, creating an airtight seal. Figure 5.13 shows the tube going in to the inflatable.

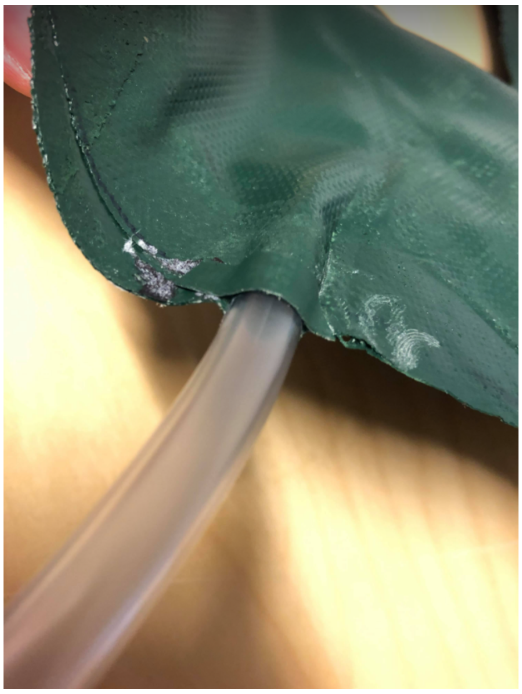


Figure 5.13: The tube used to inflate the inflatable element.

The place where the tube goes in and the inflatable itself are both completely airtight, however, some air will leak through the bulb pump if the inflatable is completely filled up as it is in figure 5.14.

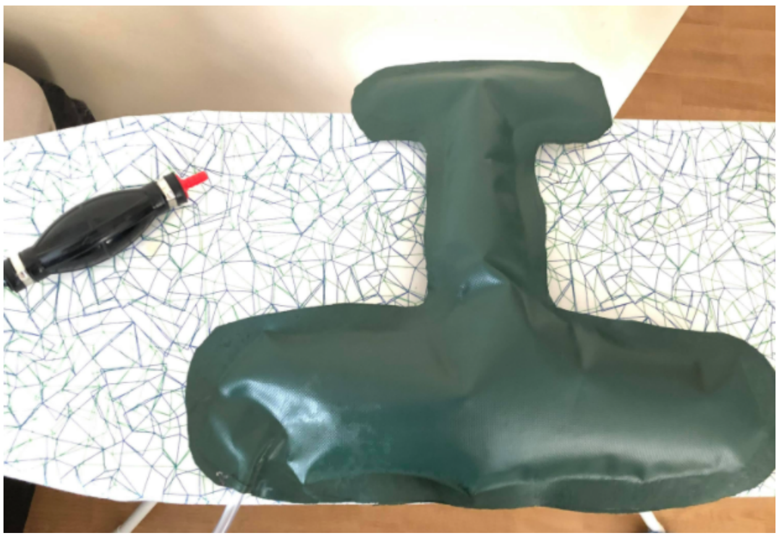


Figure 5.14: A fully inflated inflatable element.

5.2.3 Magnetic connection

The magnetic connection consists of two main parts. One part which is screwed directly onto the robot and one part which can be connected to the helmet. Each part is equipped with 16 $10mm \times 2mm$ neodymium disc magnets with a reported pulling force of 0.9kg each. Figure 5.15 and 5.16 shows both sides of the parts of the magnetic connection separately and figure 5.17 shows how it looks when they are connected to the robot.

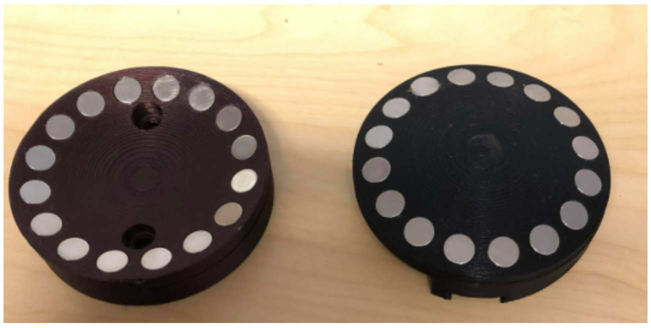


Figure 5.15: The sides of the magnetic connection with magnets.

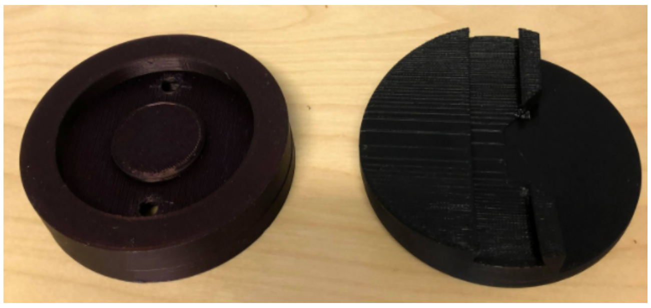


Figure 5.16: The sides of the magnetic connection that will be connected to the helmet and the robot.

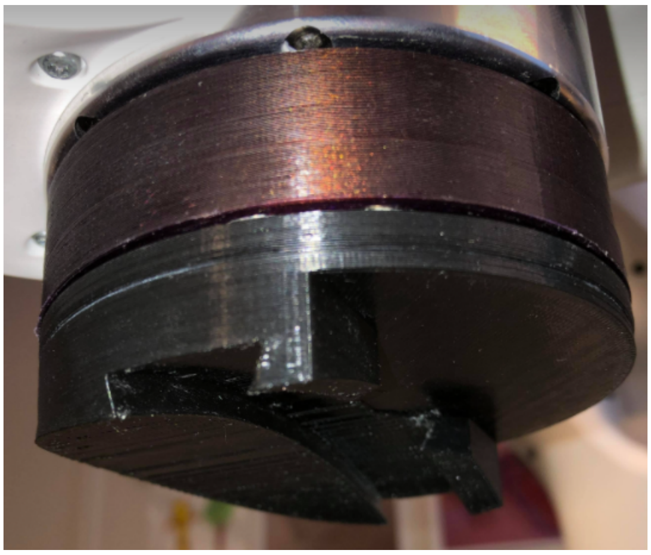


Figure 5.17: How the magnetic connection looks while connected to the robot.

6. Setup Guide

As this thesis is part of an ongoing project, the work done here will likely be continued and built upon further over the next years. This chapter will be a guide for these students to get the system up and running, and how the tools and software were set up during the work done in this thesis. The goal is that the students who are going to build further on the work done in this thesis can easily get the system running on their own machines, and that they will not have to spend a lot of time troubleshooting simple issues that causes compilation or other errors.

6.0.1 Graphical User Interface

As mentioned in chapter ??, the Graphical User Interface is built in JavaFX using the eclipse platform with scenebuilder to generate fxml and sqlite for the database platform. This section will describe how to get this software to run on the other system. The general steps will be applicable to any Operating system, while the detailed instructions are for Ubuntu 20.04, which was the OS used during the work described in this thesis. They might also be applicable to other linux/ubuntu distributions, but this can not be guaranteed.

Quick reference guide for getting this up and running

1. Install Eclipse
2. Install plugins
3. Clone git repository
4. Add the required JARs to the build path
5. Add the required arguments to the run configurations
6. Install scenebuilder

Install Eclipse

Eclipse is a free IDE available for windows, mac and linux. To install it simply download the installer from the eclipse downloads site [21] and follow the installer.

Install plugins

The only necessary plugin is E(fx)clipse, which is a great tool when working with JavaFX. E(fx)clipse can be installed directly from the Eclipse IDE through the eclipse marketplace. Help → Eclipse Marketplace → Search for any plugins you want and click install.

Clone git repository.

The project is available on github at github.com/tobflo14/devomed. To retrieve the code you can use the following terminal command:

```
git clone https://github.com/tobflo14/devomed
```

The project can then be imported in Eclipse. This requires you to hagve git installed on your system. If you dont have git, install with

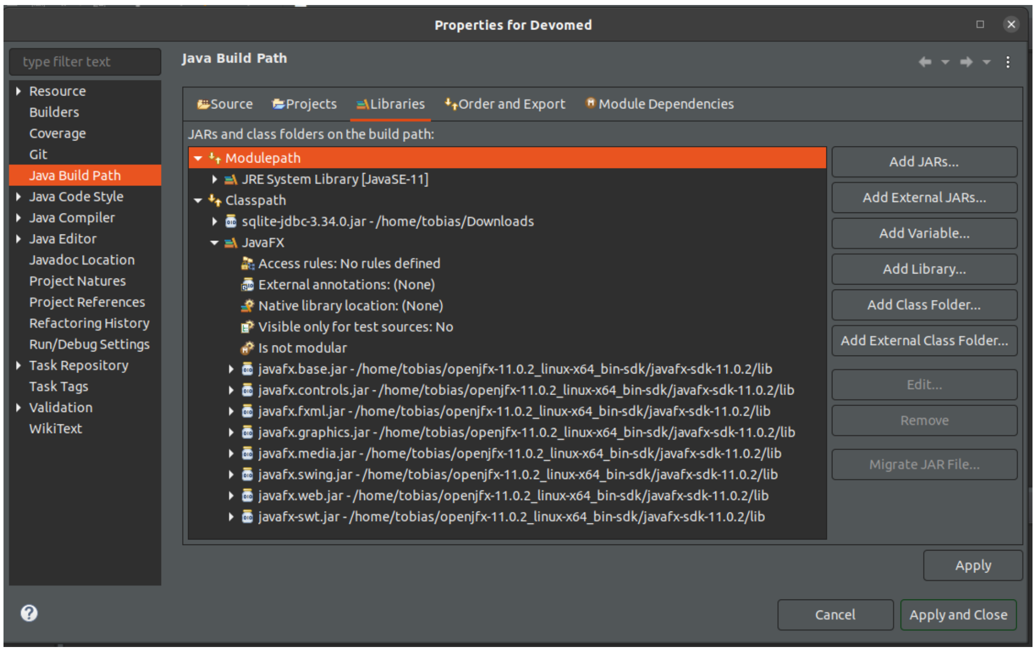
```
sudo apt-get install git.
```

Add the required JARs to the build path

As of JDK 11, JavaFX is no longer part of the core module of the java development kit. Instead it is a standalone module. To use it you therefore need to add JavaFX to the build path of your project. You also need to add the SQLite JDBC driver to the build path to access the SQLite database platform.

To add JavaFX to your build path, first you need to download the jar files from the JavaFX website [22]. The latest long-term support version JavaFX 11 was used in the development of the GUI. When the jar files are downloaded and extracted, go into the eclipse IDE and import the git repository as a project. Right click the project and select Properties → Java Build Path. Highlight the classpath and select add Library. Create a new user library named JavaFX and click add external JAR, then add all the jar files from the downloaded folder. Add this newly created library to your classpath

Similarly you need to download the jar file for the jdbc driver. The jar is available at maven repositories [23]. and add the external jar to your classpath. When this is completed your build path should look like figure 6.1.



JARs and class folders on the build path:

```
Modulepath
  JRE System Library [JavaSE-11]
Classpath
  sqlite-jdbc-3.34.0.jar - /home/tobias/Downloads
  JavaFX
    Access rules: No rules defined
    External annotations: (None)
    Native library location: (None)
    Visible only for test sources: No
    Is not modular
    javafx.base.jar - /home/tobias/openjfx-11.0.2_linux-x64_bin-sdk/javafx-sdk-11.0.2/lib
    javafx.controls.jar - /home/tobias/openjfx-11.0.2_linux-x64_bin-sdk/javafx-sdk-11.0.2/lib
    javafx.fxml.jar - /home/tobias/openjfx-11.0.2_linux-x64_bin-sdk/javafx-sdk-11.0.2/lib
    javafx.graphics.jar - /home/tobias/openjfx-11.0.2_linux-x64_bin-sdk/javafx-sdk-11.0.2/lib
    javafx.media.jar - /home/tobias/openjfx-11.0.2_linux-x64_bin-sdk/javafx-sdk-11.0.2/lib
    javafx.swing.jar - /home/tobias/openjfx-11.0.2_linux-x64_bin-sdk/javafx-sdk-11.0.2/lib
    javafx.web.jar - /home/tobias/openjfx-11.0.2_linux-x64_bin-sdk/javafx-sdk-11.0.2/lib
    javafx-swt.jar - /home/tobias/openjfx-11.0.2_linux-x64_bin-sdk/javafx-sdk-11.0.2/lib
```

Figure 6.1: The Java Build Path necessary to run a JavaFX application with SQlite.

Add the required arguments to the run configurations

The final step before being able to run a JavaFX application is to add the arguments to the run configuration of the project. Open the app class file in the src folder and click on run. You should then get an error saying “Error: JavaFX runtime components are missing, and are required to run this application”. This is because you need to provide the path of the javafx library to the java runtime. Go into run configurations by clicking the arrow next to the run button and select “app” (the name of the class with the main method of the GUI) under Java application. Select the arguments tab and in the VM arguments you need to add the following line:

```
--module-path \path to javafx jar" --add-modules javafx.controls,javafx.fxml
```

The path to the javafx jar will vary based on the operating system and where you downloaded them, in ubuntu you can find the path listed as parent folder of the properties of the jar files.

Install Scenebuilder

Download scenebuilder from the gluon website [24], and follow the installer. To get this to work from inside Eclipse, you need to link eclipse to the scenebuilder executable. Do this by clicking window → Preferences → JavaFX and provide the path to the scenebuilder executable.

6.1 Robot Control

1. Ensure your hardware and operating system meets the requirements
2. Install required packages
3. Install libfranka
4. Download git repository
5. Build and compile
6. Hardware setup

Ensure your hardware and operating system meets the requirements

The franka control interface has some requirements that the system must meet in order to interact with the panda robot. For the hardware the main requirements are the networking card and cables. According to Franka Emika the minimum requirements for the network card is 100BASE-TX, and the cable needs to be a “high performance networking cable”. A cat5e cable was tested and was within the requirements. Using a cable that is as short as possible gave better results. Once libfranka is installed and the hardware connected, the network configuration can be tested using the communication test in the libfrank examples. Running the test will tell you if the current setup is sufficient to operate the robot. There are also requirements to the operating system. Most importantly the linux kernel being used must be patched with a preempt_rt patch to be a real-time kernel. Instructions for how to do this can be found in the documentation for the Franka Control Interface [25].

Install Required Packages

To install the packages required type the following commands in the terminal:

```
sudo apt-get update
Sudo apt-get install libeigen3-dev
Sudo apt-get install libpoco-dev
Sudo apt-get install freeglut-dev
Sudo apt-get install libgtkmm-3.0-dev
sudo apt-get install libboost-all-dev
```

Installing libfranka

Instructions for installing libfranka can also be found in the FCI documentation [25]. Building libfranka from source can be done with a few commands in the terminal. Navigate to where you want to install libfranka and type in the following commands:

```
sudo apt install build-essential cmake git libpoco-dev libeigen3-dev
git clone --recursive https://github.com/frankaemika/libfranka
cd libfranka
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
cmake --build .
```

These commands will build and compile the newest version of the libfranka library. Finally in order for cmake to be able to locate the package

```
sudo make install
```

Download git repository

The project is available on github at github.com/tobflo14/devomedRobot. To retrieve the code simply type into the terminal:

```
git clone https://github.com/tobflo14/devomedRobot
```

Build and compile

Navigate to devomedRobot folder and open a terminal there. To compile the program use the following commands:

```
mkdir build
cd build
cmake build ..
make
```

Hardware setup

Connect the robot as shown in the image.

7. Discussion and further work

This chapter will discuss the current implementation in relation to the problem description and what further work should be done.

7.1 Problem description

7.2 User interface

A brief demonstration of the user interface is shown in a video linked in appendix A. Because of the decision to create the GUI in javaFX for cross-platform compatibility, the user interface had to be rebuilt almost from scratch. The basic layout of the user interface was preserved as much as possible while implementing some new functionality.

When it comes to the appearance of the interface, no styling was done to the fxml pages. That means buttons, tables, text and textfield all appear with their default styling and fonts. To make the GUI more visually appealing css styling can be added to the pages. For the purpose of creating an MVP this was considered a low priority task. Implementing a style sheet with a graphical profile should be considered for future work on this project.

7.2.1 JavaFX

JavaFX proved to be an excellent tool to create a user interface. The object oriented nature of java and javaFX leads to code that is easy to understand and easy to build further upon. Great emphasis was made during the development to make the code as intuitive and readable as possible. Variable names were chosen to be very descriptive of what the variable represents, and methods were given long names that describes precisely what the behaviour of the method is. We believe that the code written is an excellent basis for further development of the project.

7.2.2 Network Communication

The network communication implementation described in section 4.2.4 works well for the functionality that has been developed. As it stands that functionality includes updating the position data of the robot in real-time as it moves, and starting and stopping position tracking for the generation of exercises. In accordance with the goals set in the problem description, the network programming was developed with easy expansion in mind. We believe that further development of networking functionality will work seamlessly with the basis that was developed during this project.

7.2.3 3D Visualization

3D visualization is used for two different purposes in the current implementation: Previewing the exercises, and showing a real-time visualization of the movement. The previews seems to function as expected, and is able to follow any path that has been tested. One feature that could be implemented is support for speed in the animation. The current implementation follows the path with equal time between each point, so if the exercise requires varying speed in different section of the path this would not be visible in the preview. This could be implemented by either adding a speed variable to the exercise that would relate to the time between points in the animation.

There are also some problems with the head model itself. Morten commented that the graphics needs to look natural and realistic to not be a distraction during the performance of the exercise but rather be a helpful tool to aid in the recovery of the patient. He suggested that the graphics need to be displayed from the shoulders so that the patient could better understand how the movement is done. This functionality would require a much more complex model than the one currently being used. To animate the movements of the neck the model needs to be separated into a skeleton that defines how the movements are performed, and a skin that defines how the head looks at any possible configuration of the skeleton. Another challenge with this feature is that peoples heads are different. Defining this movement accurately would require more information than just the pose of the robot. Unless you know the position of the shoulders and size of the neck this could potentially lead to some awkward or impossible neck angles in the visualization. One idea is to take measurements when creating the patient profile that alters how the visualization is done. This along with a way to lock and reproduce the position of the torso in the chair can be used to define the motion of the neck from the measure motion of the robot.

7.2.4 Database

SQLite was an excellent tool to create a database for this purpose. It is easy to work with, fast and easy to manage with the graphical DB browser tool. The main drawback of SQLite is that it locks the entire database file when it is being used by a thread. Since the database is only used locally this has not been an issue. A backup system needs to be developed to ensure the reliability of the system. With all the exercise data stored in one place there is currently a risk of losing everything if the database file is corrupted or lost. Some form of cloud based backup could be used, or the database file could be backed up on the computer running the robot controller.

7.3 Head mount

Figure 7.1 displays the entire head mount with the robot. The purpose of the head mount is to ensure that the patient can perform the correct exercise motions while connected to the robot. It should feel safe and comfortable to wear, while still keeping the head in place. The different parts of the head mount will be discussed below with suggestions for any improvements that may be advantageous.

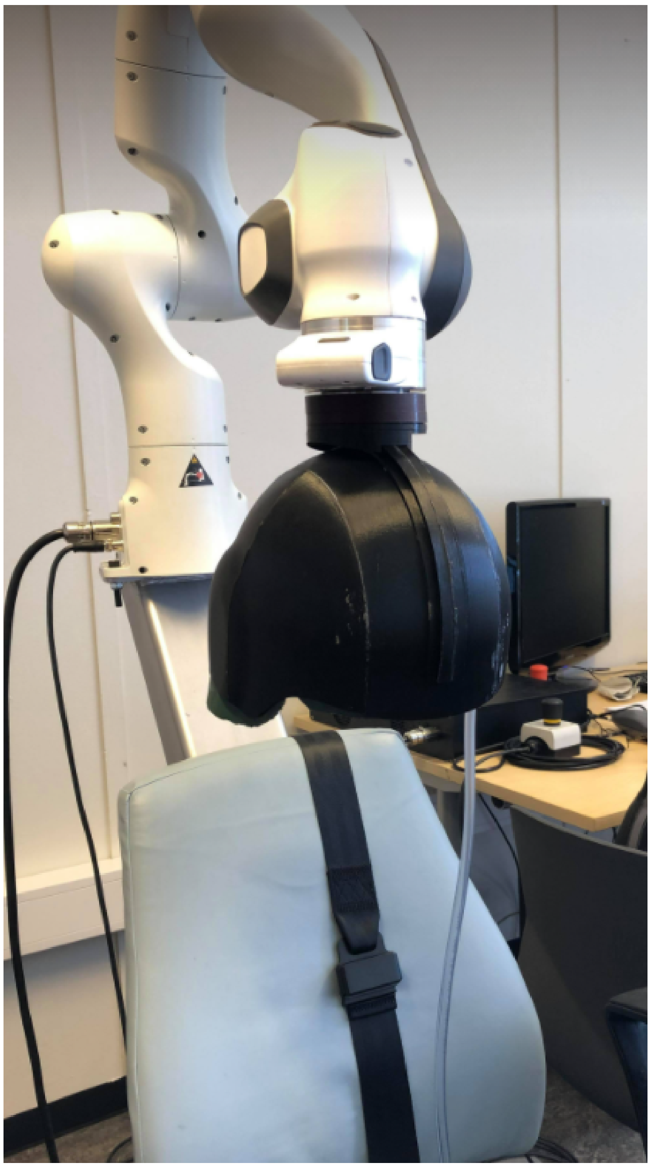


Figure 7.1: The head mount attached to the robot.

7.3.1 Helmet

The main objective of the helmet is to provide a rigid way of securing the head to the robot. It has dimensions to fit any human head, and to ensure that people with smaller heads also can use the helmet without being too loose, it is equipped with an inflatable element that can be filled accordingly. This provides a rigid connection to the robot, but it is worth noting that testing was done with Halvors head, which is quite big. A smaller head will leave more room for the air to move around inside the helmet, which can result in some unwanted slack. Air is compressible so creating a universal helmet that can fit anyone, and still be rigid enough to keep the head in place can probably not be done with this type of solution. Solutions that can rely on something other than air, for example foam that hardens, to keep the head in the same place inside the helmet should be investigated. However, solutions that require a unique design for each individual patient will be very expensive.

The helmet shell is rigid, however, no tests to discover how much force the helmet can withstand without breaking have been done. The reason for not testing this is to avoid actually breaking the helmet. Exercises performed by whiplash injured patients will most likely not expose the helmet to forces that can result in breaking it. The weakest areas of the helmet are the places where parts are glued together. If a 3d-printer big enough to print the entire helmet in one piece becomes available for students in the future, it would be advantageous to print it in one piece. Other than that, the current helmet serves its purpose, but making the same helmet in different sizes can be a good idea when it is time to incorporate this apparatus as an approved method of rehabilitation.

7.3.2 Inflatable element

The main objective of the inflatable element is to ensure that any patient can use the helmet, regardless of head size, by inflating the element. It will also make the helmet more comfortable to wear. Figure 5.14 shows the inflatable element fully inflated. Some aspects of the inflatable element can be regarded as a success, whilst some aspects require some further work. The shape and size of the inflatable element allows for patients to fit the helmet regardless of head size, and the slots for the ears helps to avoid ear fatigue when wearing the helmet for longer periods. However, as mentioned above, air is compressible and will move around when moving the head, resulting in slack. Wearing the helmet for longer periods is uncomfortable because of the plastic material used to create the inflatable element. It becomes very clammy and sweaty after only wearing it for a couple of minutes. If a way to use air to keep the head still inside the helmet is discovered, further work on this project should definitely explore different materials to use for the inflatable element.

The inflatable itself and the seal around the tube going in is completely airtight, however, a small amount of air can be noticed leaking out of the bulb pump when it is fully inflated. The bulb pump not being able to hold the air in also means that it is not able to pump more air in when the pressure reaches a certain point. Air was leaking out so slowly that it was still possible to test the functionality of the inflatable by blowing it up with the mouth. Students continuing this project should look for a more suited bulb pump.

7.3.3 Magnetic connection

The objective of the magnetic connection is to provide a way to connect the helmet to the robot, with the added ability to quickly disconnect the helmet as a safety feature in case something unforeseen happens to the robot. It is important that the magnetic connection is strong enough to not interfere when a patient is performing an exercise, but loose enough to trigger if a potential dangerous force is emitted from the robot. The bottom part of the magnetic connection is also equipped with a slider system to attach the helmet which will allow the physical therapist to tilt the helmet forwards or backwards depending on the needs of the patient.

The magnetic connection in itself works great, the pull of the magnets results in a stable connection to the robot, without being so strong that it is impossible to disconnect from the robot by moving the head. The pull from the magnets is strongest along the axis going straight through the connection, but it only requires a slight tilt from this axis to significantly reduce the force required to disconnect from the robot. The required force to disconnect from the robot exactly along its axis feels a little too much. However, it is very unlikely that a patient will be able to stay exactly along this axis if the robot should suddenly move so this is not regarded as an issue.

A problem that is an issue is that the wedge, which is used to lock the slider on the helmet together with the magnetic connection, can very easily unlock itself. This results in the helmet just sliding back and forth on the connection. A suggestion on how to fix this is to drill two holes in the bottom part of the magnetic connection as shown by the white dots in figure 7.2, and subsequently drill multiple holes in the slider with the same space between them as the holes in the magnetic connection.

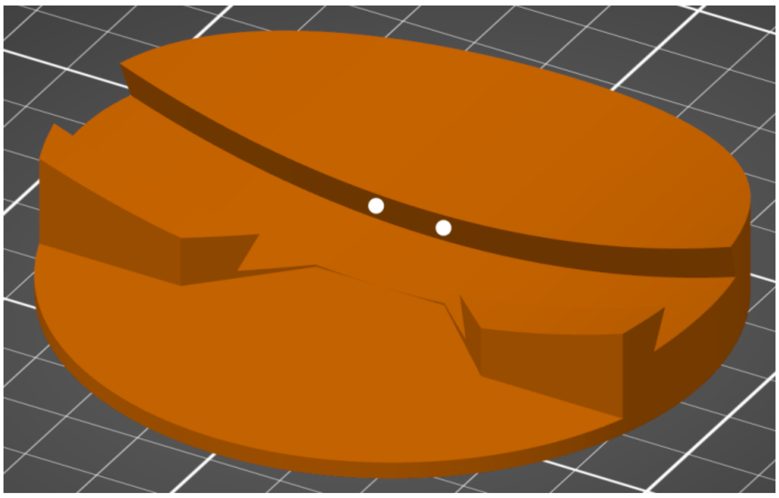


Figure 7.2: Suggested spots to drill holes for the improved locking system for the slider.

Then, instead of a wedge, design a plug-like device that can be pushed in, locking the slider in its place. This will reduce the precision of where the slider can be locked to the connection, but making enough holes should still result in satisfactory precision. Another option can be that the development of the robot control result in the slider not being needed. In that case, the possibility of attaching the magnets directly to the helmet should be explored, possibly requiring a redesign of the helmet.

8. Conclusion

8.1 Head mount

The head mount that has been created consists of three main parts, a helmet, an inflatable element and a magnetic connection. The purpose of the head mount is to allow patients to interact with robot safely. The helmet and the inflatable element are both designed to allow any patient, regardless of head size, to be able to use the head mount. Creating a universal helmet is not the optimal solution as the slack from air moving around results in the head mount being nowhere near precise enough to perform rehabilitation exercises with the required precision. It may be that the helmet can still be used, if another method of adjusting the inside of the helmet is discovered.

The magnetic connection provides a rigid connection to the robot, while still being able to disconnect with a reasonable amount of force. It provides safety in a situation where the patient is vulnerable.

8.2 Graphical user interface

Bibliography

[1] BTE Technologies. Multi cervical unit. Available at https://www.btetechnologies.com/rehabilitation/mcu/, 2021.

[2] S. A. Pertuz. Mass-spring-damper system. Available at https://www.researchgate.net/figure/Mass-Damper-Spring-system-representing-an-impedance-controller-behaviour-On-this-system_fig1_324363084, 2018.

[3] Dewesoft. Pid-control. Available at https://training.dewesoft.com/online/course/pid-control, 2020.

[4] Thomas Erik L. Gælok and Michelle Strand. Utvikling av maskin for opptrening av nakkeslengskade. Master's thesis, NTNU, 2017.

[5] Mayoclinic. Whiplash. Available at https://www.mayoclinic.org/diseases-conditions/whiplash/symptoms-causes/syc-20378921#:~:text=Whiplash%20is%20a%20neck%20injury,traumas%2C%20such%20as%20a%20fall, 2020.

[6] State Insurance Regulatory Authority. Guidelines for the management of acute whiplash associated disorders for health professionals. Available at https://www.sira.nsw.gov.au/resources-library/motor-accident-resources/publications/for-professionals/whiplash-resources/SIRA08104-Whiplash-Guidelines-1117-396479.pdf, 2014.

[7] TNS Gallup. Firda fys-med senter brukerundersøkelse. Available at https://www.firdafysmed.no/brukerundersokelse.pdf, 2013.

[8] Benoît Jacob and Gaël Guennebaud. Eigen-library. Available at http://eigen.tuxfamily.org/index.php?title=Main_Page, 2020.

[9] Ove Baardsgaard and Kia Brekke. Development of robot based rehabilitation apparatus for whiplash injured patients. Master's thesis, NTNU, 2019.

[10] Franka Emika GmbH. Franka control interface documentation. Available at https://frankaemika.github.io/docs/ (09/12/2020).

[11] Hohenstein Institute. Hohenstein conducts 3d head study to help helmet manufacturers. Available at `https://www.innovationintextiles.com/hohenstein-conducts-3d-head-study-to-help-helmet-manufacturers/`, 2015.

[12] Human head. Available at `https://en.wikipedia.org/wiki/Human_head`, 2021.

[13] Oracle. Javafx documentation. Available at `https://openjfx.io/javadoc/16/`, 2021.

[14] InteractiveMesh. Javafx 3d model importers. Available at `http://www.interactivemesh.org/models/jfx3dimporter.html`, 2021.

[15] Mad Mouse Design. Male head 3d object. Available at `https://www.turbosquid.com/3d-models/male-head-obj/346686`, 2007.

[16] Blender. Blender, the free and open source 3d creation suite. Available at `hhttps://www.blender.org/`, 2021.

[17] K. M. Lynch and F. C. Park. *Modern Robotics - Mechanics, Planning, and Control*. Cambridge University Press, Cambridge UK, 1st edition, 2017.

[18] Google & Harris Poll. Online security survey. Available at `https://services.google.com/fh/files/blogs/google_security_infographic.pdf`, 2019.

[19] Niels Provos and David Mazières. A future-adaptable password scheme. 1999.

[20] Damien Miller. Java implementation of bcrypt. Available at `https://www.mindrot.org/projects/jBCrypt/`, 2021.

[21] Eclipse. Downloads. Available at `https://www.eclipse.org/downloads/`, 2021.

[22] Gluon. Javafx. Available at `https://gluonhq.com/products/javafx/`, 2021.

[23] Xerial. Sqlite jdbc driver. Available at `https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc`, 2021.

[24] Gluon. Scenebuilder. Available at `https://gluonhq.com/products/scene-builder/`, 2021.

[25] Franka Emika. Fci documentation, installation linux. Available at `https://frankaemika.github.io/docs/installation_linux.html`, 2021.

Appendices

A. Demonstration

A video showing a short demonstration of the current functionality of the new user Interface can be found on the link below. The video has been uploaded to youtube as an unregistered video, meaning that it is only reachable by the link below. This video is also included as digital appendix.

`https://www.youtube.com/watch?v=DcUI61rZRzU&ab_channel=TobiasFl%C3%B8tre`

B. Database Code

```java
    package devomed;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Random;

public class SqliteDB {
    Connection c = null;
    Statement stmt = null;

    public SqliteDB () {
        try {
            Class.forName("org.sqlite.JDBC");
            c = DriverManager.getConnection("jdbc:sqlite:devomedDB");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }

    }

    public void closeConnection() {
        try {
            c.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public void executeUpdate (String query) {
        try {
            this.stmt = c.createStatement();
            stmt.executeUpdate(query);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public ResultSet executeQuery (String query) {
        try {
            this.stmt = c.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            return rs;
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
            return null;
        }
    }

    public ArrayList<Therapist> getTherapists() {
        ArrayList<Therapist> therapists = new ArrayList<Therapist>();
        try {
            ResultSet rs = executeQuery("SELECT * FROM therapists");
            while (rs.next()) {
                int userID = rs.getInt("id");
                String username = rs.getString("username");
                String password = rs.getString("password");
                String name = rs.getString("name");
                therapists.add(new Therapist(userID, username,
                   password, name));
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return therapists;
    }

    public String getPassword (String username) {
        String password = "";
        try {
            ResultSet rs = executeQuery("SELECT password FROM
               therapists WHERE username=\'"+username + "\'");
            password = rs.getString("password");

        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
            return password;
    }

    public Therapist getTherapist(int userID) {
        Therapist therapist = null;
        try {
            ResultSet rs = executeQuery("SELECT * FROM therapists
               WHERE id="+Integer.toString(userID));
            if (rs.next()) {
                therapist = new Therapist(rs.getInt("id"), rs.
                   getString("username"), rs.getString("password"),rs.
                   getString("name"));
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        addPatients(therapist);
        return therapist;
    }

    public Therapist getTherapist(String username) {
        Therapist therapist = null;
        try {
            ResultSet rs = executeQuery("SELECT * FROM therapists
               WHERE username=\'"+username + "\'");
            if (rs.next()) {
                therapist = getTherapist(rs.getInt("id"));
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return therapist;
    }

    public Patient getPatient(int userID) {
        Patient patient = null;
        try {
            ResultSet rs = executeQuery("SELECT * FROM patients WHERE
               id="+Integer.toString(userID));
            if (rs.next()) {
```

```java
                patient = new Patient(rs.getInt("id"), rs.getString("name"), rs.getString("info"), this.getTherapist(rs.getInt("therapist")));
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return patient;
    }

    public Patient getPatient(String username) {
        Patient patient = null;
        try {
            ResultSet rs = executeQuery("SELECT * FROM patients WHERE username=\'"+username+"\'");
            if (rs.next()) {
                patient = new Patient(rs.getInt("id"), rs.getString("name"), rs.getString("info"), this.getTherapist(rs.getInt("therapist")));
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        return patient;
    }

    public User getUser(String username) {
        if (this.getPatient(username) != null) {
            return this.getPatient(username);
        }
        else if (this.getTherapist(username) != null) {
            return this.getTherapist(username);
        }
        else {
            return null;
        }
    }

    public User getUser(int userID) {
        if (this.getPatient(userID) != null) {
            return this.getPatient(userID);
        }
        else if (this.getTherapist(userID) != null) {
            return this.getTherapist(userID);
        }
        else {
            return null;
        }
    }

    public static int generateID() {
        int minimum = (int) Math.pow(10, 7);
        int maximum = (int) Math.pow(10, 8) - 1;
        Random random = new Random();
        return minimum + random.nextInt((maximum - minimum) + 1);
    }

    public void addPatients(Therapist therapist) {
        try {
            ResultSet rs = executeQuery("SELECT * FROM patients WHERE therapist="+Integer.toString(therapist.getUserID()));
            while (rs.next()) {
                Patient patient = new Patient(rs.getInt("id"), rs.getString("name"), rs.getString("info"), therapist);
                therapist.addPatient(patient);
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        for (Patient patient : therapist.getPatients()) {
            addExercises(patient);
        }
    }

    public void addExercises(Patient patient) {
        try {
            ResultSet rs = executeQuery("SELECT * FROM exercises WHERE patient="+Integer.toString(patient.getUserID()));
            while (rs.next()) {
                Exercise exercise = new Exercise(rs.getInt("id"), patient, rs.getString("name"), rs.getInt("count"));
                patient.addExercise(exercise);
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        for (Exercise exercise : patient.getExercises()) {
            addPath(exercise);
        }
        addPrograms(patient);
    }

    public void addPath(Exercise exercise) {
        try {
            ResultSet rs = executeQuery("SELECT * FROM paths WHERE
               exercise="+Integer.toString(exercise.getExerciseID()));
            while (rs.next()) {
                Path path = new Path(rs.getInt("id"), exercise);
                path.setPoints(rs.getString("points"));
                exercise.setPath(path);
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

public void addPrograms(Patient patient) {
try {
ResultSet rs = executeQuery("SELECT * FROM programs WHERE patient="+
   Integer.toString(patient.getUserID()));
while (rs.next()) {
Program program = new Program(rs.getInt("timesPerformed"), rs.
   getString("name"), patient, rs.getInt("id"));
ResultSet rs2 = executeQuery("SELECT * FROM programExercises WHERE
   program="+Integer.toString(rs.getInt("id")));
while (rs2.next()) {
ProgramExercise programExercise = new ProgramExercise(rs2.getInt("id")
   ,
  patient.getExercise(rs2.getInt("exercise")),
  rs2.getInt("repetitions"),
  rs2.getInt("sets"),
  rs2.getInt("resistance"));
    program.addExercise(programExercise);
}
patient.addProgram(program);
}
} catch (Exception e) {
System.out.println("Error came from addProgram()");
System.out.println("Error: " + e.getMessage());
}
}
}
```

C. Github Repositories

Here the github repositories will be provided that contains all the code for the project. The code is split into two repositories, one for the robot control side, and one for the GUI side.

C.1 GUI

`https://github.com/tobflo14/devomed`

C.2 Robot Control

`https://github.com/tobflo14/devomedRobot`

D. Specialization Project

NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

INSTITUTT FOR MASKINTEKNIKK OG PRODUKSJON

Development of a Robot Based Rehabilitation Apparatus for Whiplash Injured Patients

Specialization Project

Tobias Fløtre
Halvor Romundstad

Supervisors: Knut Einar Aasland & Amund Skavhaug

December 2020

NTNU

Preface

This project has been a collaboration between Tobias Fløtre and Halvor Romundstad as a specialization project during the fall of 2020.

We would like to thank Ove Baardsgaard for invaluable assistance in the startup phase of the project, and Morten Leirgul for sharing his insight in the workings of the human Neck and the motivation behind this project. We also want thank our Supervisors Knut Einar Aasland and Amund Skavhaug for providing an interesting topic and helpful tips and guidance along the way.

Summary

This project is part of an ongoing series of projects in Collaboration with Firda Fysikalsk-Medisinske senter in Sandane, Sogn og Fjordane. The long-term goal of the project is to develop a training apparatus that is to be used in the rehabilitation of patients who have suffered a whiplash injury. Our project is a continuation of the work performed by Ove Baardsgaard and Kia Brekke who started the development of a platform based on the Panda robot arm by Franka Emika GmbH.

The focus of this specialization project has been to familiarize ourselves with the current concept, the theory behind it, and the Panda robot arm, and to further develop the solutions for compliant robot control and the Graphical User Interface.

For the robot control a new method for compliant robot control that includes orientation was tested. The results were inconclusive, but some improvements were observed, and more investigation is needed to conclude if it is a viable option.

For the Graphical User Interface several improvements have been suggested, and implementations for the unimplemented requirements have been proposed. The most important of these proposed changes is to separate the GUI from the control loop to improve the robot loops Real-Time performance.

Contents

List of Figures

1. Introduction

This project is part of an ongoing cooperation with Firda Physical Medical Centre (Firda Fysmed) at Sandane in Norway, where physiotherapist Morten Leirgul is the instigator. The goal of the project is to create a rehabilitation apparatus for patients with whiplash injuries. This specialization project is a continuation of 6 previous master's theses, with Baardsgaard and Brekke (2019) [3] being the latest. In their thesis they investigated if the robotic manipulator, Panda, by Franka Emika GmbH could be used as main motion platform for a whiplash rehabilitation apparatus, and started the work to develop a compliant robot control system. They also started the development of a graphical user interface.

1.1 Background and motivation

Whiplash is an common injury worldwide and it is related to accidents where the head and neck gets thrown rapidly forwards and backwards, like in a car crash. Patients suffering from whiplash often experience symptoms like dizziness, nausea or headache. The current treatment methods of whiplash patients at Firda Fysmed is a combination of the patient pushing their forehead against the physical therapists hand in different directions and using an exercise apparatus called the Multi Cervical Unit (MCU).

The MCU is an apparatus where the patient can either rotate their head, move their head forwards and backwards or side to side. It is not possible to combine these movements. These type of movements around one static axis is not representative of a natural neck movement, as a natural neck movement would result in the axes translating from their original positions.

When performing the training exercises, it is desirable to have a graphical user interface where the exercise can be monitored. In addition to monitoring the exercise, the user interface should include the option to create two types of user profiles. There should be one option to create a profile for the physiotherapist, and another option where the physiotherapist can create a profile for each patient, as each patient will require a unique training program.

The patients treated with the MCU at Firda Fysmed are able to obtain good results, however, the limited possibilities with the MCU results in Firda Fysmed not recommending it to other clinics. The goal of this ongoing project is to develop an apparatus which can compete with the MCU in either price, functionality or both.

1.2 Problem description

There has been a lot of previous work in this project where the long term goal is to develop a training apparatus for whiplash patients which fills the demands of a minimum viable product (MVP). The supervisors has stated the importance of not rushing to try to find a solution, but instead take the time to learn and fully understand how the current concept works and what the previous groups have done. The end-goal is a MVP in the summer. The main goals in this project will be:

- Learning how the current system works and understand the theory behind it.
- Continuing the development of the robot control system, the main focus being on rotation.
- Continuing the development of the GUI so that it meets the demands of a MVP.

2. Theory

2.1 Panda by Franka Emika

This section will give a quick introduction on why the Panda was chosen for this project and its specifications.

The Panda is a robotic manipulator made by Franka Emika. It is made up by 7 joints linked to each other as shown in Figure 2.1. This allows for 7 DOF, which is the required number of DOF as stated by Baardsgaard and Brekke [3]. 6 DOF should be enough in theory, however the joints can turn in different planes according to the pose resulting in the loss of one or more DOF. If the robot has <5 DOF, it will reach a kinematic singularity. A kinematic singularity means that not all movements are possible anymore, no matter how the joints move. With the 7 DOF the Panda has, this should not be a problem.

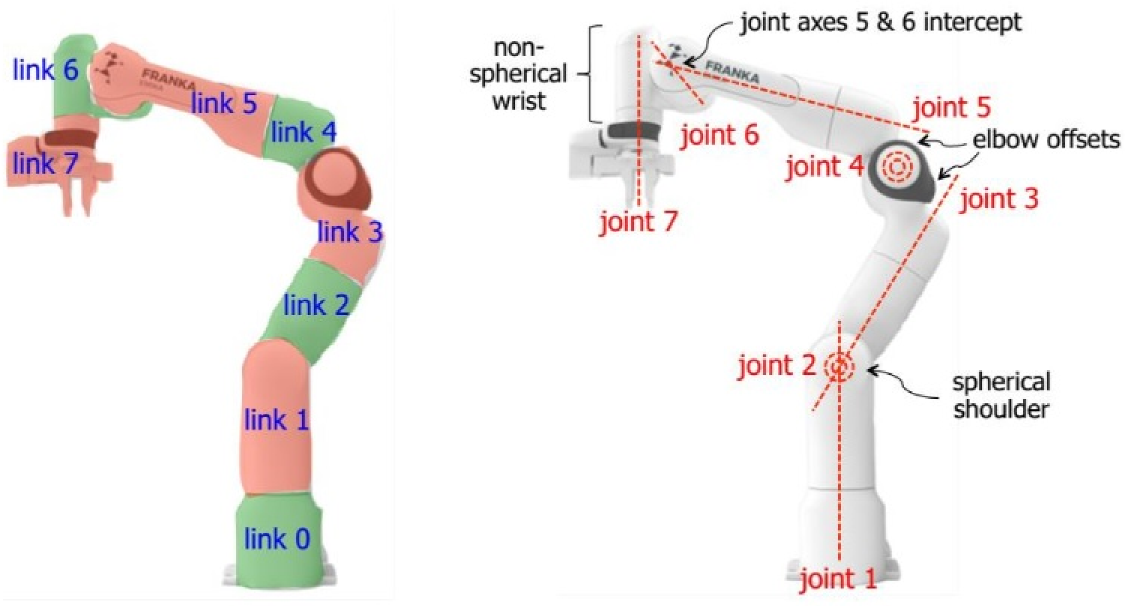


Figure 2.1: The joints and links of the Panda, The figure shows a gripper connected to the end-effector, but in this project it will be replaced by a helmet.

The robot should also be able to record a guided movement and repeat that movement. Morten Leirgul at Firda fysmed stated that the robot should not deviate more than a millimeter when repeating the recorded movement. Franka Emika states that the Panda has a repeatability of +/- 0.1mm [4] which should be sufficient.

The Panda also has torque sensors in each joint, a 3kg payload and 855mm maximum reach. At the time the Panda was acquired, there was no other robotic manipulator with the Panda's repeatability, 7 DOF and torque sensors available in the same price range. This resulted in the Panda being the natural choice.

2.2 Neck and whiplash

This section will describe how the neck works and what a whiplash injury is.

2.2.1 The neck

The neck is one of the most important parts of the body, and is responsible for supporting and moving the head. The most important parts of the neck are cervical vertebrae, ligaments and muscles. The cervical vertebrae are the seven upmost parts of the spine. They are named C1-C7, where C1 is the top one. These vertebrae are shaped in a way that allows for movement in 6 degrees of freedom. However, it is the contraction of the many different muscles that makes the head move. The ligaments can be seen as elastic bands that can stretch and retract within their own elastic range without problems. However if a ligament is stretched beyond its elastic range, it will be permanently extended.[5]

Out of the seven vertebrae, C3-C7 are very similar to the rest of the spine. The vertebrae are stacked on top of each other and helps with keeping us upright. They also allow for some movement and rotation. C1 and C2, also known as ”Atlas” and ”Axis” respectively, are a bit different. They are shaped in a way that allows for much bigger rotation of the head. C1 is shaped like a ring and sits on top of a small bone from C2. How C1 and C2 move in relation to each other accounts for about 50% of the entire area where the neck manage to move in rotation and extension/flexion.

Figure 2.2 shows the three usual motions used to describe neck movement, which is also the the movements that can be done with the MCU. The figure is from a NASA article providing data on the necks range, which is important for the robot to handle. Table 2.1 shows the range of the female neck in the movements shown in Figure 2.2. The results of the female neck is chosen as females usually have a bit larger range than males, and the robot has to work for both genders.

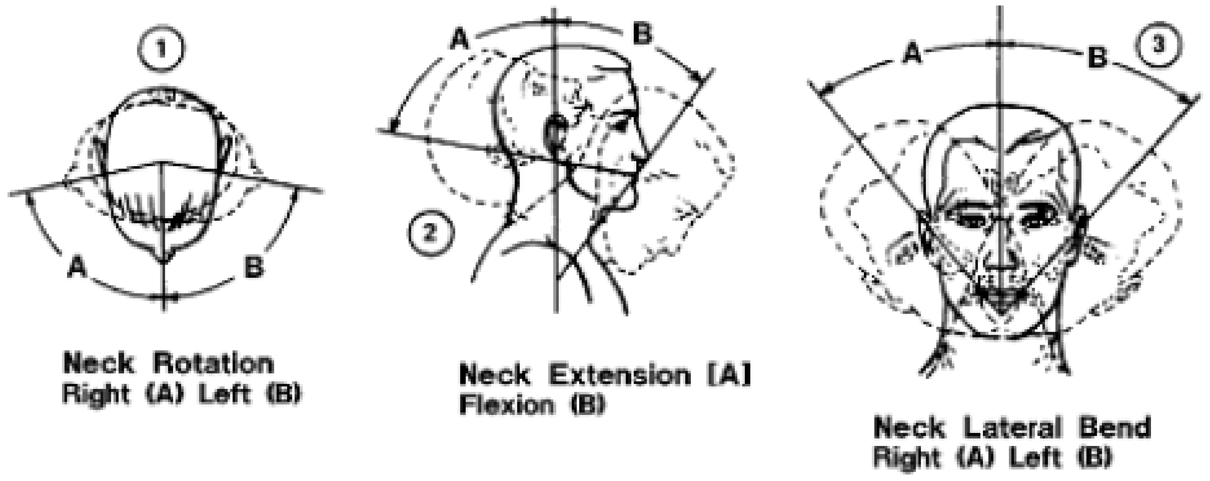


Figure 2.2: Neck motions

Table 2.1: Neck range

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

2.2.2 Whiplash

Whiplash is described as a neck injury caused by a forceful, rapid back-and-forth movement of the neck, like cracking a whip [6]. Figure 2.3 shows the sequence of events which can result in a whiplash injury. A similar movement in another direction can also be described as whiplash. The most common cause of whiplash is rear-end car accidents, but it can also be a result from a sports accident, physical abuse or other trauma.

During a whiplash, the neck is exposed to such quick accelerations and big movements that multiple parts of the neck can take damage. The ligaments can be stretched out of their elastic range causing permanent damage, or muscles, nerves and joints can get damaged. What parts of the neck that are injured can be hard to determine, leading to difficulties when deciding the extent of the injury. This can lead to discussion of whether an injury actually exist, which can result in problems when getting insurance to cover costs.

The difficulty in deciding the injury can also present problems when it comes to rehabilitation. Imaging such as x-ray or MRI can only identify the most severe cases such as fractures, displacement of vertebrae or damages to the spinal cord. Training the neck after a whiplash is disputed, even though there has been a lot of research yielding positive results. An article from New South Wales State Insurance Regulatory Authority reports such good evidence of training the neck having a positive effect that it should be the recommended treatment in most cases [7].

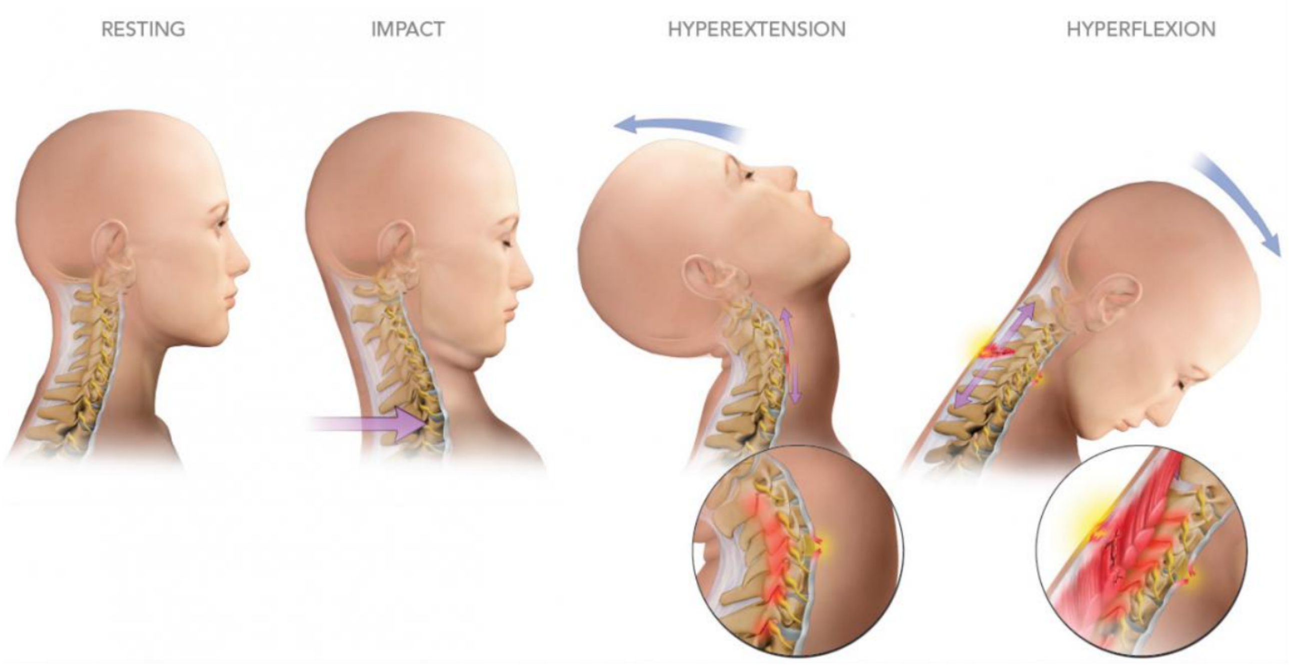


Figure 2.3: Sequence of events during whiplash

Firda fysmed has specialized in whiplash injuries and they agree that training is the best method for rehabilitating the neck. Their focus is to train the muscles around the injured area, which has gotten great results. 97% of the patients treated at Firda fysmed reported an improvement of their situation in a survey [8]. This survey also report that the degree of disability of the patients is reduced. Despite these great results, Firda fysmed believes that training with an optimized apparatus will yield even better results. Even though it is hard to clinically prove improvement by x-ray or MRI, improvements can be seen by increased strength and mobility. The most important thing is that the treatment can help improve the quality of life of the patient.

2.3 Control theory

Baardsgaard and Brekke [3] started the work with controlling the robot to the desired behavior. This section will describe relevant control theory.

2.3.1 Admittance controller

An admittance controller is a controller based on the mass-spring-damper system shown in Figure 2.4, which means that the robot is supposed to replicate the behavior of a mass connected to a spring and a damper. Equation 2.1 shows the resulting expression for the desired acceleration, $\vec{a}$.

$$\vec{a} = \frac{1}{m(\vec{F_{ext}} - k\vec{r} - c\vec{v})} \tag{2.1}$$

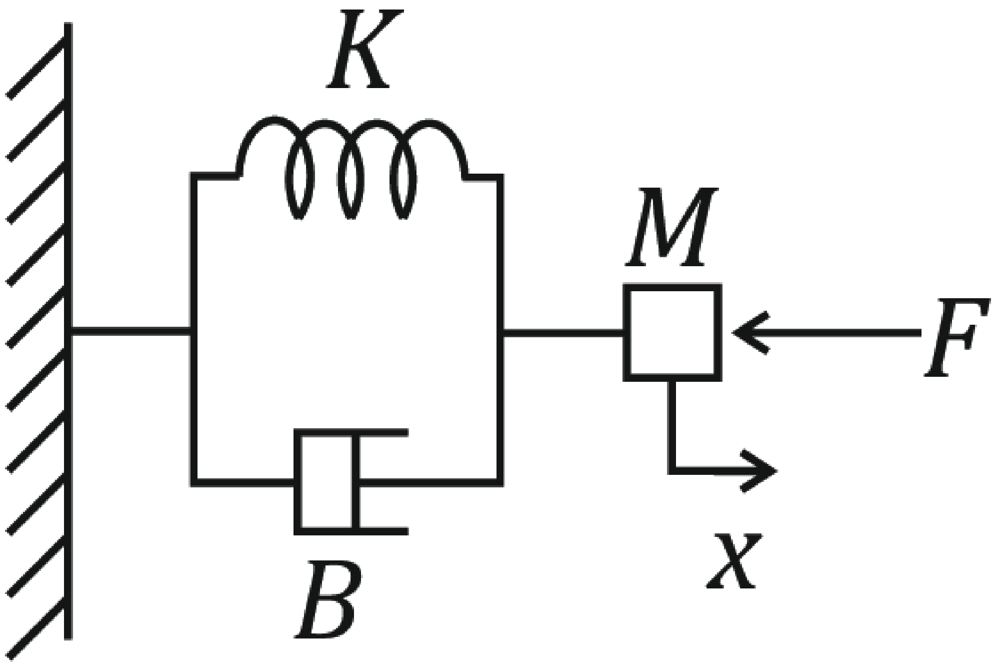


Figure 2.4: Mass-spring-damper system by Pertuz et al [1]

$c\vec{v}$ is the force applied by the damper, $k\vec{r}$ is the force applied by the spring, m is a virtual mass and $\vec{F_{ext}}$ is the external force estimated at the flange.

The spring in this system would try to move the robot to its original position, an undesirable behavior for this projects purpose. The spring force was therefore removed from the expression for the controller (Equation 2.1), which resulted in the following expression for the controller:

$$\vec{a} = \frac{1}{m(\vec{F_{ext}} - c\vec{v})} \tag{2.2}$$

2.3.2 PID controller

A PID controller is a great method for controlling a system around a set-point. A PID controller uses the difference between the desired set-point and the current state of the system, also called an error, as an input. The output from the controller tries do reduce this error so that the system state is in accordance with the desired set-point. There are three terms that decide the output from the controller, the proportional term P, the integral term I, and the derivative term D. The block diagram for a PID controller is shown in Figure 2.5.

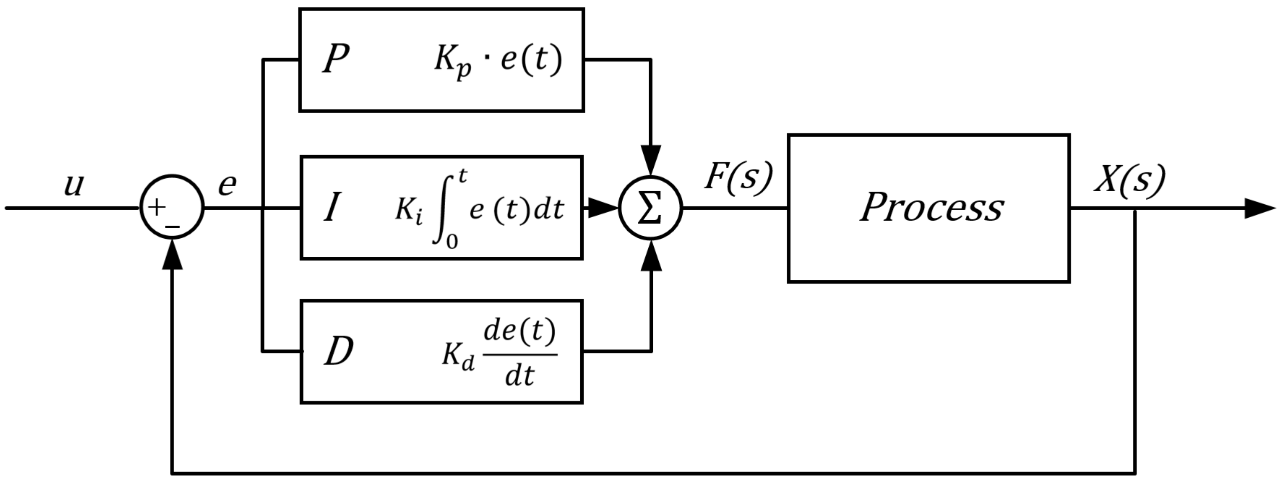


Figure 2.5: Block diagram for a PID controller [2]

P term

The proportional term is simply used to produce an output proportional to the error. A big proportional gain, K_p will give a big change in the output relative to the error, but can make the system unstable if tuned too high. On the contrary, a small proportional gain will give the controller less responsiveness and less sensitivity, but may not be able too reach the set-point if tuned too low.

I term

The integral term is used to produce an output based on the accumulated error over time. The integral value of the error is multiplied by an integral gain, K_i. This can remove the steady-state error in a system, but too big of an integral gain can cause the system to overshoot the set-point which increases the time before the system reaches the set-point, and may result in an unstable system.

D term

The derivative term uses the rate of change of the error and multiply this by an derivative gain, K_d. This term is used to predict the system behavior and will help reduce the time before the system reaches the set-point, and also improve the stability of the system. However, the derivative term is often not used as it is very sensitive to disturbances.

2.4 Terms

Preemption means to interrupt a task running on a processor to allow a higher priority task to run instead. The interrupted task may resume running at the point where it was interrupted when the higher priority task is complete.

TCP/IP is also known as the Internet protocol suite is a set of communication protocols used to transmit data over the internet.

Serialization is the process of converting objects or data structures to a stream of bytes.

Sockets are network nodes that acts as end-points in network communication (sender and receiver).

Shoulder, Elbow and wrists are terminology for describing the joints of the robot arm by comparing it to the human arm. Joints 1, 2 and 3 are combined referred to as the shoulder joints, joint 4 is the elbow, and joints 5, 6 and 7 make up the wrist of the robot.

3. Previous work

This has been an ongoing project since 2014, and the latest master thesis by Baardsgaard and Brekke (2019) [3] investigated the possibility of using the robotic arm Panda by Franka Emika GmbH as proposed by Brattgjerd and Festøy (2018) [9]. They also started developing a graphical user interface and a control system for the Panda.

3.1 Physical Components

The physical components of the apparatus are the seating mechanism, the robot, and the helmet.

The seating was developed by Brattgjerd and Festøy. It consists of a chair with armrests, mounted on a heavy steel platform. The height of the seat can be adjusted with buttons on the armrest, to accommodate patients of different heights.

The helmet is the interface between the patient and the robot. The helmet concept was developed by Brattgjerd and Festøy, where they used inflatable airbags inside the helmet to secure it properly. Baardsgaard and Brekke improved upon the concept by making the helmet one-size-fits-all, and made a prototype which is now missing.

3.2 Software

3.2.1 Software Architecture

In the main function a global struct with robot data is declared, and then threads are created to handle the different core components of the software. The threads that are currently working are the **GUI Thread**, which creates the graphical user interface and the **Robot Loop Thread**, which handles the motion of the robot. A simple diagram of this is shown in figure 3.1.

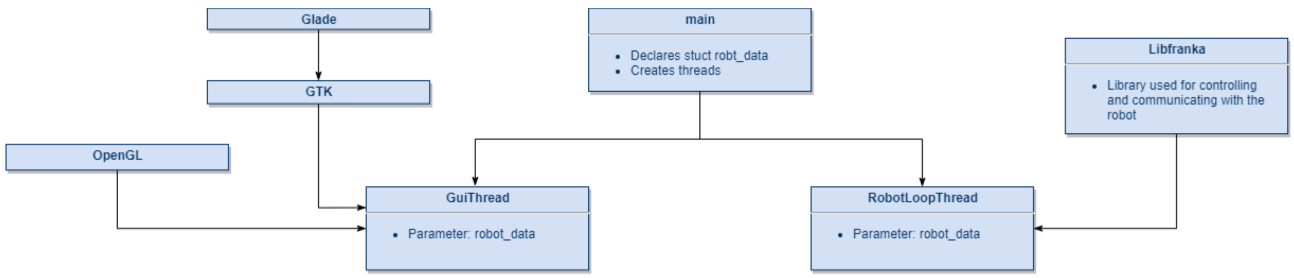


Figure 3.1: Diagram showing how the threads in the system interacts.

3.2.2 Graphical User Interface

Baardsgaard and Brekke started the development of a Graphical User Interface (GUI). Their solution is based on GTK and OpenGL, and is developed using Glade. Current features include user selection, creating and running exercise programs, and visualizing the exercises using 2D and 3D plots. There are also some development features for adjusting parameters in the Robot controller. None of the features are working as intended at the moment.
According to Baardsgaard and Brekke the most crucial implementations that needs to be made 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

3.3 Robot Control

The robot loop Thread runs a 1kHz control loop that receives force measurements from the robot, and calculates robot movement. The controls are implemented using libfranka, which is a library developed by Franka Emika for use with the panda robot in the Franka Control Interface (FCI).

The robot control loop is the part of the system where real-time constraints are most important. According to the FCI documentation the constraints are defined as the following:

It must be guaranteed that the sum of the following time measurements is less than 1ms:

- Round trip time between the workstation PC and FCI
- Execution time of your motion generator or control loop.
- Time needed by the robot to process your data and step the internal controller

Baardsgaard and Brekke started the work with developing a compliant control system for the robotic arm, but there is still some work to be done when it comes to making a compliant robotic arm, i.e lowering the resistance when moving the robot so that it can easily be moved in any direction. A big part of this specialization project has been investigating the previous work and understanding the different concepts.

3.3.1 System description

The overall target is to make a control system that makes the robot compliant to the point where the user can move the end-effector with a range from almost no resistance to resistance decided by the physiotherapist. The solution by Baardsgaard and Brekke can be divided into the following three parts:

- Getting the data from the internal sensors in the robot and processing it.
- Using a force controller which converts the force input data to a desired cartesian velocity.
- Converting the cartesian velocities to joint velocities, and controlling the Panda to carry out these velocities.

3.3.2 Getting sensor data

With the use of libfranka, obtaining the different sensor data is very simple. Most values can be taken from `robot_state`. Performing matrix and vector calculations is more easily done with the Eigen library[10], which is used when there is a need to map the values from `robot_state` into eigenvectors or eigen matrices. `franka::jointVelocities` was the interface chosen for the velocity controller, which results in the cartesian velocities not being obtainable from `robot_state`. This means that the cartesian velocities have to be calculated using the Jacobian and joint velocities with the following equation:

$$\vec{v} = J\vec{q} \tag{3.1}$$

where $\vec{v}$ is a 1 $\times$ 6 vector of cartesian velocities, J is the 6 $\times$ 7 Jacobian, and $\vec{q}$ is a 1 $\times$ 7 vector of joint velocities. External force on the flange can be obtained from `robot_state.k_f_ext_hat_k`. There is no torque sensor at the flange so `robot_state.k_f_ext_hat_k` uses the torques from the torque sensors in the joints to estimate a force at the flange. `K_F_ext_hat` is a 1×6 vector containing torques around the X-, Y- and Z-axis as well as forces in the X, Y and Z directions. This force signal is corrected for gravity, inertia, coriolis and centrifugal effects. There is also a low-pass filter, with a cutoff at 10 Hz, included to reduce noise. The filter is a function from libfranka; `franka::LowpassFilter`. This results in a smooth signal.

3.3.3 Velocity controller

Libfranka contains multiple controllers for different methods of controlling the robot. You can control either velocity or position, in task space or joint space. The force controller, which will be described later, gives the desired velocity/acceleration in task space, so a task space (cartesian) velocity controller was needed. Libfranka has two existing interfaces for this kind of control, one which uses cartesian velocities, and on that uses joint velocities. Both of these where explored, and using the joint velocities controller was decided because it yielded great velocity control and the velocity, acceleration and jerk of all joints were automatically limited. The cartesian velocities interface does not limit these parameters automatically, and to make sure the joint space limits were satisfied, you had to set the cartesian limits much lower than recommended by Franka, which constrains the potential of the robot.
When using the `franka::jointVelocities` interface the robot is controlled using joint velocities, and since a patient perform exercise motions in cartesian space, there is a need to convert the cartesian velocities to joint velocities. This can be done by using 3.2:

$$\vec{\dot{q}} = J^{+}\vec{v} \tag{3.2}$$

where J^{+} is the pseudo-inverse Jacobian.

3.3.4 Force controller

One of the most important objectives in this project is to be able to move the end-effector with as little resistance as possible. The controller developed by Baardsgaard and Brekke [3] did not include the option to rotate the end-effector. Different control strategies to reduce the resistance with a constant orientation of the end-effector were explored.

The easiest solution is a feedback control where the force input at the flange is fed back to the motors by Newtons 2nd law, $F = ma$, where F is the estimated force at the flange, a is the desired acceleration or the Panda and m is a virtual mass. The virtual mass should be very low to reduce the inertia, however when m approaches zero, a goes towards infinity. This results in rapid movements by the robot from small force inputs at the flange. Baardsgaard and Brekke [3] also reports that the robot moved with increasing oscillations. In this type of project, where a patient is wearing a helmet connected to the robot, such movements can be dangerous. A feedback controller based on Newtons 2nd law will therefore not be optimal in this project. Other controllers had to be explored. The most common controllers are an admittance controller, a PID controller, or a controller based an a dynamic model of the system. All of these controllers were explored.

3.3.5 Admittance controller

The theory behind an admittance controller is described in Section 2.3.1. To test the controller you had to assign a value to the virtual mass, m. With m=10kg a stable motion with a slow response was reported. To quicken the response, m can be reduced, but at some point the robot starts oscillating because the response was overcompensating. c can be introduced to dampen these oscillations. Increasing c while lowering m while still having a stable motion is how the controller is tuned. However, no combination of these values yielded acceptable results. The robot still required a considerable force to move.

3.3.6 PID controller

The theory behind a PID controller is described in Section 2.3.2. The PID controller tested by Baardsgaard and Brekke [3] used the difference between zero, which is the desired force, and the force applied to the robot, F_{ext}, as the input to the controller. The resulting output from the controller is a signal to control the desired velocity of the robot. When tuning the PID controller for the robot, it started oscillating when K_p was increased. The D term could be used to dampen these oscillations, however, a loud unpleasant sound came from the motors when increasing K_d. This sound is caused by small, rapid velocity changes in the motors, which happen because of noise disturbing the signal. Filtering was tried to reduce the noise, but the signal could not be filtered enough to get a satisfactory control output from using the PID controller.

3.3.7 Dynamic model

A dynamic model of the system can be used to make a control system based on the physical properties of a model with the desired behavior. For example, the previously mentioned admittance controller is based on a dynamic model of a mass-spring-damper system. To make a dynamic model to base the control system on one have to specify the desired behavior of the robot. The desired behavior of the Panda would be to make it move with as little resistance as possible, while being stable and not moving if there is no force input. Baardsgaard and Brekke [3] mentions the behavior of an object moving on a surface with friction to be similar to the desired behavior of the Panda, as it will be easily movable and will stop if not pushed.

Making a dynamic model of the desired object is done by combining the equations for the inertia of the mass (Equation 3.3), the Coulomb friction with static friction (Equation 3.4) and the kinetic friction (Equation 3.5).

$$\vec{F_m} = m\vec{a} \tag{3.3}$$

$$\vec{F_{fs}} = -\vec{F_{ext}} \tag{3.4}$$

$$\vec{F_{fk}} = -\mu m g \hat{v} \tag{3.5}$$

Equation 3.6 shows the resulting dynamic equation for the model. $\vec{F_f}$ is either the kinetic or the static friction. The kinetic friction only applies when the force applied, $\vec{F_{ext}}$ is bigger than $\vec{F_{fk}}$, or if the object is still moving. Similarly, the static friction only applies when $\vec{F_{ext}}$ is less than $\vec{F_{fk}}$, or if the object is standing still.

$$\vec{F_{ext}} = m\vec{a} + \vec{F_f} \tag{3.6}$$

To get commands to control the robot from Equation 3.6, it has to be rewritten to Equation 3.7, which gives the desired acceleration.

$$\vec{a_{desired}} = \frac{1}{m}(\vec{F_{ext}} + \vec{F_f}) = \frac{1}{m}\sum\vec{F} \tag{3.7}$$

From this equation and Equation 3.4, static friction would result in $\vec{a_{desired}} = 0$, which could mean that robot is moving with a constant velocity. This is solved by the fact that $\vec{F_{fk}}$ is used as $\vec{F_f}$ when the velocity is nonzero, and the kinetic friction will slow the robot down until it stops.

To optimize the controller from the dynamic model, the virtual mass m, and the friction coefficient μ had to be tuned. Tuning these parameters is done by starting with a large m and no μ, then increase μ while decreasing m. Oscillations occurred when m was decreased, but increasing μ was an effective way to dampen these oscillations. The values yielding the best results were m=2.0 kg and μ=0.01. The dynamic model based force controller gave the least amount of resistance when moving the robot of the mentioned controllers.

3.3.8 Adding weight

The end goal is to use the robot as an exercise apparatus, which means that one should be able to "add weight" as is done in normal exercises at the gym. To make the Panda different from the MCU, it is important to avoid having to use force to hold the head still in the middle of, or after a movement. The MCU will try to "pull" the head back to the original position.

The easiest way to add resistance only during movement is to add a friction force with the desired resistant force, resulting in a new expression for the force controller being Equation 3.8, where $\vec{F_w}$ is the wanted friction force. With this added friction, the robot will only resist when an exercise movement is performed, and no force will be required from the patient too keep their head in place after a movement is finished, or if they stop in the middle of a movement. There is also no desire to have added resistance while moving the head back to the original position. To avoid this, the completed percentage of the motion is calculated. When 99% of the motion is completed, the resistance will be set to zero until the patient is back in the starting position, where the desired resistance is added again.

$$a_{\vec{desired}} = \frac{1}{m}(F_{\vec{ext}} + \vec{F_f} + \vec{F_w}) \tag{3.8}$$

3.3.9 Joint impedance

The stiffness of the robot is defined by the impedance of the joints, and need to be defined before starting the robot loop. Baardsgaard and Brekke [3] reports that setting a low joint impedance dampen oscillations caused by the force controller. After some experimenting with the impedance parameters, the stability of the controller increased. However, lower impedance reduces the accuracy of the robot, but can help the robot feel more safe and human for the patient.

3.3.10 Limiting joint positions

The robot will stop and give an error if the joints move outside limits described in Franka docs [11]. This is solved by Baardsgaard and Brekke [3] by resisting movement in the joints when approaching these limits. To resist movement when approaching joint limits, a PID controller is used.

The desired velocities from the force controller are used to calculate the desired joint positions, and then the desired joint positions are clamped to their respective limits. The difference between desired position and clamped position is fed into the PID controller as an error. The controller will try to minimize this error and will add the output to the desired positions as a correction.

The PID controller need to be tuned, Baardsgaard and Brekke [3] reports that K_p=0.001 and K_d=0.00002 gives a firm stop without being to harsh.

3.3.11 Recording a path

To use the Panda as an exercise apparatus, the patient needs to be able to repeat the desired motion from the physiotherapist multiple times. This is done by recording the motion while the physiotherapist guides the patient through the desired motion without resistance from the robot. After a motion is recorded, resistance can be added so the exercise can be performed.

A motion is recorded by obtaining the Cartesian positions of the end-effector from `robot_state.O_T_EE`. The positions are pushed to a vector in `robot_data.tracking_data`, which is shared so it can be read from another thread. A shared thread is required because the time it takes to save the data might exceed the 1ms time limit for the robot loop.

The data is saved to a .csv file where each row contains a set of X, Y and Z coordinates. Multiple sets of these coordinates is called a path. The reason .csv was chosen is because it is easy to write to with simple delimiters. It is also great if the points needs to be visualized for debugging using spreadsheet software.

The robot loop runs at 1kHz, which means that data is sampled 1000 times each second leading to an extremely detailed path and a large amount of data. This seems excessive so Baardsgaard and Brekke [3] decided a sampling rate of 100 times per second would be sufficient, which resulted in a smooth path.

3.3.12 Following a path

After a path is recorded, there is also a need for the robot to replicate this path. Baardsgaard and Brekke [3] tested multiple ways to achieve this which will be described in this section.

The first idea is based on taking the Cartesian velocity output from the force controller and manipulating it towards the same direction as the direction which should be constrained. This is done by using the dot product between the Cartesian velocity and a unit vector in the chosen direction. With this method, the robot only moved along the chosen vector, however this method does not account for the position of the robot which means that the robot only follows the direction of the path, not the path itself. This can result in an offset from the desired position.

As the results from the previously mentioned method of following a single vector was not satisfactory, something else had to be tested. This approach is based on using the many points used to create the path, specifically the points that are closest to the position of the end-effector. A vector can be created from the points closest and 2nd closest to the end-effector. Constraining the velocity to this vector should in theory update the constrained direction which makes the robot follow the path. This approach was relatively successful, however the robot struggled in the corners of the path. Offset can also cause problems as there is such a small distance between time steps, which can lead to the robot skipping between vectors resulting in movement in another direction than the desired one.

The two previous methods have been based on velocity control. To better the robots ability to follow a path, Baardsgaard and Brekke [3] thought of a new approach based on position control. In this approach, the desired velocity from the force controller is integrated to find the desired position. After that, the distance between the desired position and the closest point on each line segment on the path could be found. A line segment is a line between two of the points in the desired path. To figure out which point on a line segment is the closest, one can project the desired position onto the line. If the projected point is on the line then there is no error and the function can just return the projected point, if not, the function should return the closest point on the line. The new desired position was the calculated point that was the closest from all the line segments.

A PID controller was used to control the robot to the desired position. The error fed into the PID controller was the difference between the desired position, and the output from the controller is a ”correction” which is added to the desired velocity. This will help the robot steer to the path. Baardsgaard and Brekke [3] reports that after some tuning of the PID controller, the robot is able to perfectly follow a path. Together with the force controller, this method made the robot easy to move through a recorded path, but if K_p in the D term is too high the motors can emit an audible sound.

4. Development

4.1 Robot Control

4.1.1 Compliant robot system

One of the most important tasks to be done in order to achieve an MVP is a compliant control system for the robot arm. The compliant control implementation developed by Baardsgaard and Brekke currently only supports compliant position control. To achieve an MVP this must be expanded to also account for rotational motion. Baardsgaard and Brekke attempted to solve this problem using a force controller by switching out mass for inertia in their dynamic model [3]. The result they got from this solution was that there was a lot more resistance in the rotations compared to the result they had achieved when focusing only on the position. In this section we have looked at some possibilities on how to implement rotation into the compliant robot system.

4.1.2 Torque control

Based on a suggestion from the user Alessia on the Franka Community Forums, a potential solution based on torque control was tested. She had used this solution for a similar application and that reportedly worked well for both rotational and positional compliance. The procedure is described below.

1. Control the robot with zero torque plus the coriolis vector
2. Manually rotate each joint, while logging the data for joint torque and joint velocity during the movement
3. Use linear regression to estimate a negative damping factor for each joint as the relation between torque and velocity. Put these values in a 7x7 Diagonal Matrix D
4. Add the term $-cD\dot{\theta}$ to the torque controller. c is a tuning factor that needs to be found experimentally.

Testing the torque control

Due to a series of technical problems in the startup of this project (detailed in Appendix A), the time spent testing the robot were cut short. To investigate whether this method is worth looking into further, a quick test was performed where the method described in section 4.1.2 was followed. After the matrix D was estimated, the control law with damping factors was compared to a pure torque control where the control callback was simply return the 0 vector. The external torque was fetched from the robot state class in libfranka along with the joint velocity.

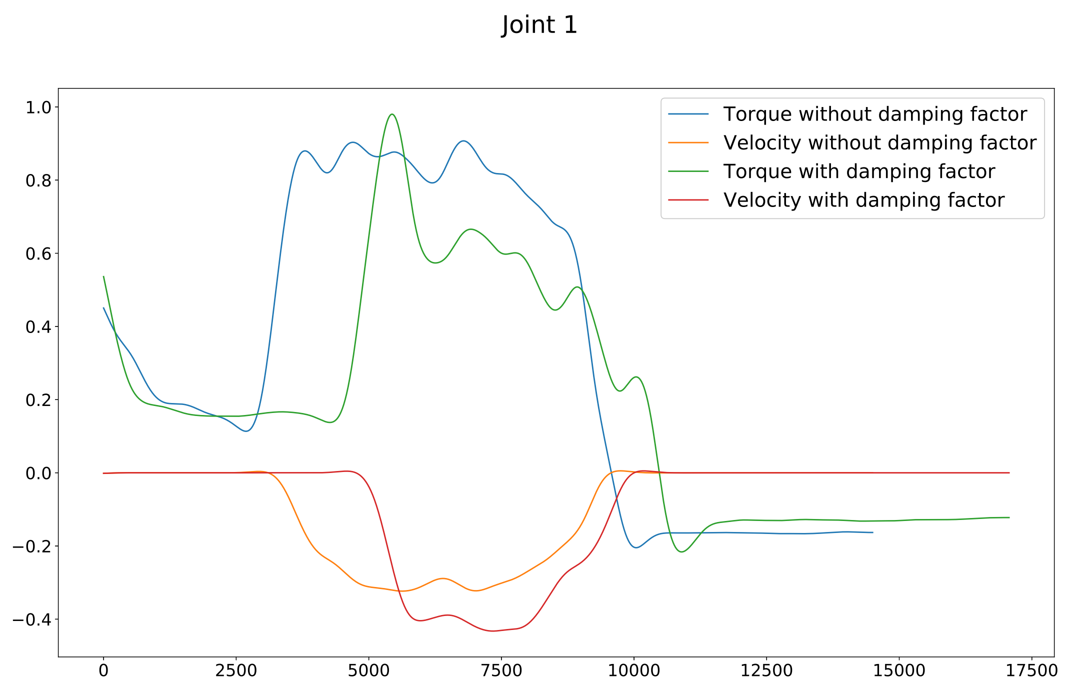


Figure 4.1: A plot comparing the external torque measured at joint 1 to the joint velocity with and without the estimated damping factor.

Figure 4.1 shows the resulting data for joint 1. This plot contains data from two different movements so the time does not quite match for all of them, but it is still a good illustration of the general trend for the joints where this method was effective. It shows that after the damping factor was added to the control law, the static resistance remained unchanged, but after the motion was started the velocity could be maintained at a higher rate with a lower applied torque. This indicated that the resistance in the joints had been reduced.

The results were not consistent for all the joints. For some joints a greater velocity was achieved by applying a lower torque, and for others the data was inconclusive. By feeling the joints the difference was very noticeable however. After the damping factors were applied, moving the robot felt much lighter both in rotation and translation, but the wrist joints still felt stiffer so the difference in resistance between rotation and translation remained significant.

All the data can be found in appendix B.

As Baardsgaard and Brekke pointed out in their specialization project, there is a lot of noise in the measurement data. The torque measurements are the worst in this aspect, but the noise is also significant in the velocity measurement. All the plots needed to be filtered before the data could be visually inspected. For the torque measurements, libfrankas robot state class already contains a filtered torque measurement, but for the plots used in this section the data was also passed through a low pass filter with a cutoff frequency of 1 Hz.

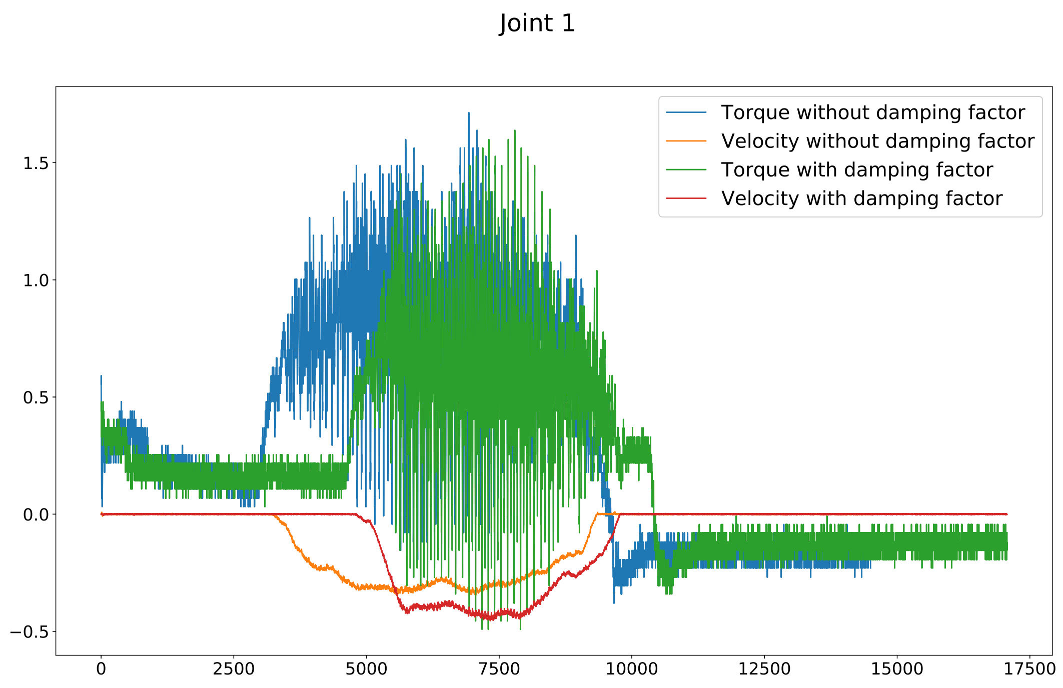


Figure 4.2: The same plot as in figure 4.1, but without filtering.

Figure 4.2 shows the unfiltered data for joint 1. All this noise might be the reason why the results are so inconsistent. The damping factors were particularly difficult to estimate because of all the data noise. With better data this might become a good method to eliminate the kinetic friction in the joints. But as the control law is based on the joint velocities it has little effect when the joints are stationary. More investigation is required to find out if the resistance is reduced further by improving the data collection, and if the model can be expanded to eliminate the static friction of the joints.

4.2 Graphical User Interface

4.2.1 The Panda as a Real-Time System

A real-time system can be defined as a computer system that needs to react to external events within a defined time-frame. [12] The Panda robot by Franka Emika is an example of such a real-time system. The control loop that is used in libfranka to control the robot runs as a 1 kHz real-time loop. That means that the code written to control the robot has to run exactly once every millisecond. Most of that time is spent on the internal software of the robot, like network communication, movement of the joint etc. The result is that the time left over for the robot control loop is approximately 300 µs according to the FCI documentation. [11].

As opposed to a hard Real-Time system, where a missed deadline will lead to a total system failure, the Panda can be considered a firm Real-Time system. Firm Real-Time means that the system can tolerate occasional missed deadlines, but the results of all computations that are not completed within the deadline is worthless. In the torque control interface of libfranka missed deadlines are handled by repeating the torques from the last successfully delivered package. This will repeat until 20 consecutive deadlines is missed, in which case the robot will shut down with a communication constraints violation error.

With real-time constraints in the system, it is very important to have consistent computation times. Without consistency there is no way to evaluate whether the code meets the requirements or not. Evaluation of real-time systems are done using worst-case execution times. It does not matter if 99% of execution meets their deadline, or that the average execution time is on time. You must be able to guarantee that the task will always meet its deadline, regardless of other external factors. This is especially important for this application as we are dealing with patients who have neck injuries that can be very vulnerable to sudden accelerations. When the robot shuts down with error, the joints are physically locked with locking bars to protect the robot from falling. A shutdown therefore involves a very sudden stop of motion. If 20 consecutive deadlines are missed in the middle of an exercise, this sudden stop may cause an already serious injury to be worsened.

In order to achieve the consistency to be able to guarantee the execution time of the control loop, the panda uses a Linux Real-Time kernel. This is a Linux kernel that have had the CONFIG_PREEMPT_RT patch applied in order to turn it into a fully preemptible kernel. That means task running in user space can preempt kernel tasks. The result of this is an operating system that is Real-Time capable. The execution time of a task is only dependent on tasks with a higher priority, so by giving the control loop a high priority you end up with a predictable and consistent worst-case execution time.

4.2.2 Implementation of the GUI as a Real-Time program

The graphical user interface needs to communicate in real-time with the Panda Robot. The current implementation of this developed by Baardsgaard and Brekke is to have the graphical user interface running as separate threads in the same process running on the same machine. The robot data is stored in a global struct that all threads have access to, which is passed as an argument when the thread is created. This might be fine for simple control loops, but as the computations needed becomes more complicated during the development of the system, execution time might become an issue. The goal must therefore be to eliminate the user interface as a limiting factor for the robot control loop.

In the implementation created by Baardsgaard and Brekke the two parts of the system interfere with each other because they are running in the same process on the same system. That means that the control loop and the GUI are competing for computer resources, they share files and data, and communication between the threads requires mutexes or semaphores that can slow the system down.

4.2.3 Splitting the program

In the current system, the GUI and Robot Control loops are running as multiple threads of a single process running on the same system. This is not ideal as you have a system with real-time constraints sharing resources with lower priority tasks. Which may lead to delays in the control loop if the GUI threads get overwhelmed.

On the trip to Firda Fysikalsk-medisinsk center, a point came up that had not previously been discussed, and that was the desire to have the graphical user interface available to the patient during the exercises. This would allow the patient to switch between exercises without hassle. It would also allow the patient to see a preview of the exercise to get a reminder of what to do, and it allows the patient to see their movement in real-time on the screen.

The proposed solution to these problem is to separate the robot control loop from the rest of the system. This has several advantages

1. The performance of the GUI software has no impact on the performance of the robot control loop.
2. The Robot control loop will get more available CPU-time, which means more/better calculations can be made if execution time is the limiting factor.
3. The GUI can be used from more mobile devices, like a tablet or phone.
4. More flexibility in choice of language and tools.

Splitting the program means that the GUI and the robot control loop runs on completely different systems. As explained in section 4.2.2. This is good for the real-time performance of the system, but it will also aid in the development process. Working on the GUI will not occupy the computer used for the control loop, so more development tasks can be done in parallel. In addition, the current implementations have these systems so intertwined that it is not possible to test any part of the GUI without having the robot running and connected

4.2.4 Programming tools

What is meant by giving more flexibility in choice of language and tools is that the panda robot has quite specific requirements in which programming language, operating system, and libraries that must be used. Libfranka only supports C++ programming and is absolutely necessary to control the robot. The operating systems available is also limited as a Linux system patched for real-time capability is the only OS with official support. There is a version available for windows but it is listed by Franka Emika as experimental [11]. When all the components of the program that does not involve the robot control loop are moved to a different system, all constraints are removed from the Graphical User Interface implementation, This means that we are free to choose any and all tools that are suited to the task and/or the authors are already familiar with.

4.2.5 Data Transmission

Separating the components of the program also introduces some new challenges. Primarily the robot data needs to be transferred between the two programs. In order to have a Graphical User Interface on a machine that is separate from the machine running the robot control loop, the robot data must be transferred between the machines in some way. To develop this there are two primary questions that needs to be answered: How to transfer the data, which data needs to be transferred.

The best way to transfer the data seems to be over TCP/IP. A cabled transmission like USB just adds an unnecessary inconvenience by being tethered to the other machine, limiting range and if using a handheld device during exercises the cable could get in the way of performing the exercise. The benefits of using TCP/IP over other wireless alternatives such as Bluetooth is that the internet is more widely used, which means more useful resources for development. It also requires no additional hardware to implement, it is more secure when well configured and has unlimited range.

When it comes to what data needs to be transferred, the problem can be broken down into the separate functions of the GUI to determine what data needs to be transferred to achieve this functionality.

Visualization requires the GUI to know the position of the robot in real-time. Since the robot has 7 rotational axes, the position of the robot can be uniquely identified by a set of seven joint angles θ. These joint angles represents the rotational position of each joint. With these seven angles any representation of the end-effector position can easily be calculated. In libfranka θ is represented by an array of seven doubles. In C++ each double requires 8 bytes, so 56 bytes in total for each update.

Creating and Performing Exercises requires the transfer of a path from the GUI to the control loop. During creation of the exercise, the path is created using the position data of the robot. The path is represented as a set of positions the robot is allowed to move to. According to the Panda datasheet [13], the pose repeatability of the robot is 0.1mm, that means a resolution greater than that would never be necessary. Exactly what the resolution of the path has to be will be determined by testing once the functionality is implemented. As a worst case scenario we can consider a 50cm path with a resolution of 0.1mm. That would result in 5000 points which would require 280 kB data. Creation and loading of exercise is not a task that necessarily needs to be in real-time. Waiting for a second before starting the exercise would not be unacceptable or inconvenient for the user.

Control variables is used here as an umbrella term for the rest of the variables that needs to be shared between the control loop and the GUI. This includes some Boolean values describing which mode the robot is currently in, Boolean values that are used to indicate that the robot should start or shut down, and variables that changes the control loop itself. These latter values are needed in the GUI as a tool to aid with testing and tuning the controller. By allowing these values to be set from GUI instead of being hard coded, tuning can be done without shutdowns or recompilations, greatly improving the speed these test can be performed.

Figure 4.3 shows a diagram for how the network communication would work. Robot data is fetched from the Robot State class of libfranka in the control loop. The necessary data must then be serialized to a byte stream in order to pass it over the network. Network sockets would be created in the Robot Control process (server side), and the GUI process (client side). Both the server side and the client side can send and receive data. The difference between them lies in how they connect to each other. The server side binds its own machine specific IP address and process specific port number to the socket and listens for inbound connections from other processes. The client side needs to know the IP address and port number of the server, and then attempts to connect outbound. After the server accepts, the connection is established and both sockets can send and receive data until the socket is closed. The sockets needs to be a stream type socket. This means that all data is guaranteed to arrive, and that the order of the data is guaranteed to be received in the same order it was sent.

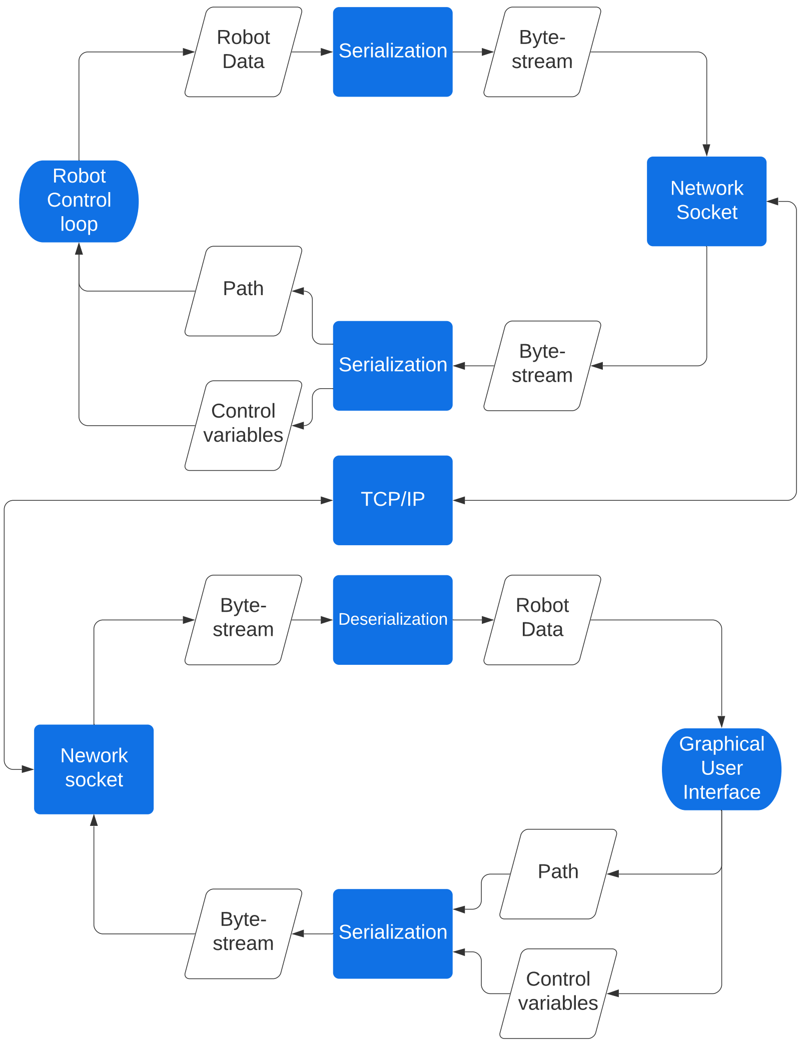


Figure 4.3: Flowchart showing the data flow for the network communication.

4.2.6 Database

One of the requirements for the graphical user interface is to have a database of patients with related exercises and training programs. For a relatively small application such as this, simply writing data to files in the system directory could be a viable solution. Alternatively there are a range of Database Management systems(DBMS) and libraries that could be implemented. These DBMS comes with features that are helpful when storing, editing, retrieving and managing the data in the database. Without using a DBMS all these features would have to be written in the GUI system, this would take a lot of time, and could result in a less reliable outcome. The popular DBMSs have been tested by millions of users and developers across the world and is a choice that saves time, troubles and eases implementation. Several options for a DBMS have been considered, and the best choice seems to be SQLite

SQLite is a lightweight relational database management system. One of the main benefits of SQLite is that it is embedded directly in the application system. Most DBMS uses a server-client architecture to manipulate and communicate with the database. SQLite stores the entire database as a single file on disk, and the application interacts directly with the database through the SQLite API. No extra server process is needed. SQLite is also self-contained, which means it can run pretty much anywhere because it does not require much assistance from the operating system or other libraries. Other benefits include a extensive and thorough documentation, it is very fast (According to the developers it is faster than direct file I/O in some cases [14]) and the code is tested to an aviation standard, the code is also open source. The library is written in C, but it has APIs available for most programming languages.

Database Structure

SQLite is a relational database manager. A relational database means that the databases consists of tables with data, and tables with the relation between the data points.[15] An example is that there is a table of exercises and a table of patients, that is the data we want to store in the database. Each of those tables have a unique key for identifying every entry in the table. Using these unique keys relations between the data can be expressed in many different ways without having duplicate or redundant data. For the GUI application a draft of the database structure have been created. Figure 4.4 shows an ERD of the proposed database structure. The boxes are tables in the database, and the lines between them represents the relationships. The crossing short line represents that a table contains exactly one of the other, while the ”crow’s foot” with a crossing line represents one or many. For example an exercise has exactly one patient, and a patient has one or many exercises. A line without markers represents a one-to-one relationship. With a structure like this patients can be created, exercises be created and stored, and it is possible to track a patients progress over time, which is the goal of the database.

Database security

Some entities in the database has security requirements. This section will identify these entities and discuss how this affects the implementation of the database.

Passwords is the one of these entities that has additional requirements for security. Passwords should never be stored in plain text in a database. That would mean anyone who managed to get access to the database would be able to read out the password of every user in the system. That could be a disgruntled employee, or a hacker who gets remote access, regardless we need to prevent them from being able to read password if they get access to database tables. The main method used to secure passwords is *hash* and *salt*. [16]

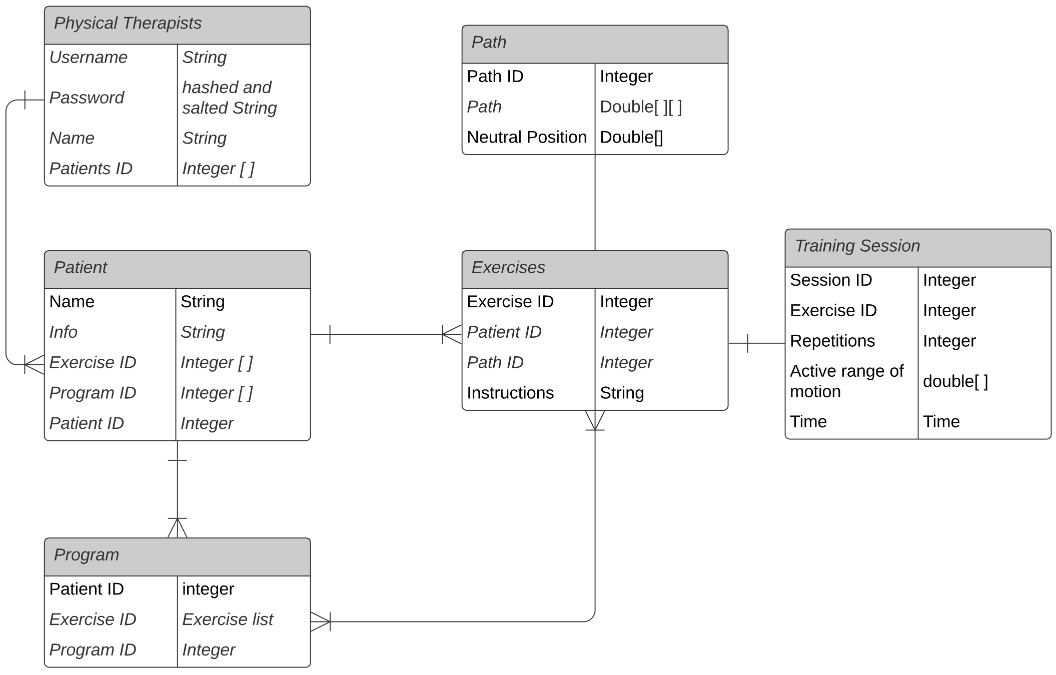


Figure 4.4: Entity Relationship Diagram showing the proposed structure of the database.

A hash is a cryptographic function used to convert the password into a string of random characters. The security in a hash comes from the fact that these functions are very easy to do one way, but incredibly hard to do in reverse.

When the password is created the string is fed into a hash function and the resulting string of random characters are stored in the database. When the user logs in, the typed password is then fed through the same function and the result is checked against the hash stored in the database. If the two strings match, then the system knows that the correct password was entered. If anyone gets access to these hash tables however, it gives them no information about what the password actually is. Only the user itself knows the actual password.

Salt is another level of security on top of the cryptography. One weakness with unsalted hash functions is that the same password generates the same hash every time. This can be exploited by attackers who gets access to the hash tables by comparing the hashes. If the same hashes appear multiple times in the table, then you know that these users have the same password. This would usually mean that password is weak to a dictionary attack, where a list of common passwords is tested to see if they produce the same hash. Another vulnerability is that some hashes have been cracked and distributed in tables known as ”rainbow tables” that can reverse engineer a password from a hash. Salting solves both of these problems by adding a random string at some point in the password before it is hashed. This string of ”salt” is then stored alongside the hash in the database, in plain text. When using salt in the hash, the same password no longer generates the same hash. Previously generated rainbow tables are also useless, as it is very unlikely they contain the exact combination of password and salt. These kind of tables would need to be specific to this application, and each password would need have an entry for every possible salt string. With this kind of cryptography in the login system, passwords should be sufficiently secured.

Sensitive Personal Information is another entity with security requirements. Processing of personal information is strictly regulated by Norwegian law. Any information that can be used to identify a person counts as personal information. In addition there is a list of specific types of information that is regarded as sensitive, and information about health is one of those. That means patient information in the database would fall under the sensitive personal information category which comes with additional requirements. The first requirement, which applies to any kind of personal information is that without grounds or sufficient legal basis, all processing of personal information is prohibited. There are several ways this legal basis can be satisfied for this application [17], the simplest one being consent. This requires the registered patient to give informed and expressed consent that is well documented and given through an active action. It is also a requirement that consent can be revoked as easily as it was given. This can be easily implemented as a step in the patient registration process. For sensitive information the requirements are even stricter. Processing of sensitive personal information is permitted if it is necessary in the process of performing health services, but strict confidentiality must be maintained[18]. This means that the operator must be qualified personnel with a duty of confidentiality, and the system must protect the confidentiality of the sensitive information in the database. Encryption is necessary here as well. Exactly how this should be implemented needs to be investigated further.

5. Conclusion and further work

This chapter will describe what has been done in relation to the problem description and what further work should be done.

5.1 Problem description

- Learning how the system delivered to us works and understand the theory behind it.
- Continuing the development of the robot control system, the main focus being on rotation.
- Continuing the development of the GUI so that it meets the demands of a MVP.

5.2 Current system

The first part of the project has been to understand how the system delivered to us worked. A lot of time has been spent reading the previous master theses and specialization projects to get a broad view of what, why and how things have been done. This has provided a great insight in the overall project and helped understand how to proceed with the project.

5.3 Robot control

The robot control system works pretty well without rotating the end-effector. However the result of the torque control test showed some improvement, but more investigation is required to determine whether this method can become a viable solution. If the robot can be compliant even with rotation, it should be great to use as an exercise apparatus.

Further work on the robot control should focus on making rotation easier. The torque control test showed some promising results in reducing the resistance, and further work should investigate if improving the data collection can reduce the resistance even more. One should also investigate whether the model can be expanded to eliminate the static friction in the joints.

Should the further investigation of the torque control test not amount to anything, acquiring a force-torque sensor should be considered. A force controller with an external force-torque sensor can help remove any force estimation issues.

5.4 User interface

The decision to split the user interface and the control system has allowed a number of different approaches to develop a GUI. Another benefit is that more computing power can be used on the robot control system.

For the user interface the next step in the process is begin implementation of the changes proposed in this project. Creating a separate GUI program that can run on a mobile device is the main priority. This also includes the communication software needed to wirelessly transfer the necessary information. This communication needs to be fast, secure and reliable to satisfy the requirements for the MVP. The GUI Implementation should be based on the design Implemented by Baardsgaard and Brekke, and the requirements identified by Gælok and Strand.

A database for storing information about patients and exercises also needs to be implemented. In this project we proposed using the Database Manager SQLite to build a simple Database stored as a single file on disk. Sufficient security measures must be applied to protect the users passwords and to comply with Norwegian regulations on processing of sensitive personal information. What personal information that needs to be stored in the database should be discussed with Firda Fysmed. A system for backing up the database should also be implemented.

Bibliography

[1] S. A. Pertuz. Mass-spring-damper system. Available at https://www.researchgate.net/figure/Mass-Damper-Spring-system-representing-an-impedance-controller-behaviour-On-this-system_fig1_324363084, 2018.

[2] Dewesoft. Pid-control. Available at https://training.dewesoft.com/online/course/pid-control, 2020.

[3] Ove Baardsgaard and Kia Brekke. Development of robot based rehabilitation apparatus for whiplash injured patients. Master's thesis, NTNU, 2019.

[4] Franka Emika GmbH. Panda by franka emika. Available at https://www.franka.de/technology, 2020.

[5] Thomas Erik L. Gælok and Michelle Strand. Utvikling av maskin for opptrening av nakkeslengskade. Master's thesis, NTNU, 2017.

[6] Mayoclinic. Whiplash. Available at https://www.mayoclinic.org/diseases-conditions/whiplash/symptoms-causes/syc-20378921#:~:text=Whiplash%20is%20a%20neck%20injury,traumas%2C%20such%20as%20a%20fall, 2020.

[7] State Insurance Regulatory Authority. Guidelines for the management of acute whiplash associated disorders for health professionals. Available at https://www.sira.nsw.gov.au/resources-library/motor-accident-resources/publications/for-professionals/whiplash-resources/SIRA08104-Whiplash-Guidelines-1117-396479.pdf, 2014.

[8] TNS Gallup. Firda fys-med senter brukerundersøkelse. Available at https://www.firdafysmed.no/brukerundersokelse.pdf, 2013.

[9] Stian K. Brattgjerd and Andrea Marie Festøy. Development of rehabilitation apparatus for whiplash patients. Master's thesis, NTNU, 2018.

[10] Benoît Jacob and Gaël Guennebaud. Eigen-library. Available at http://eigen.tuxfamily.org/index.php?title=Main_Page, 2020.

[11] Franka Emika GmbH. Franka control interface documentation. Available at https://frankaemika.github.io/docs/ (09/12/2020).

[12] A. Burns and A. Wellings. *Real-Time Systems and Programming languages*. Addison Wesley Longmain, Boston, Massachusetts, United States, 4th edition, 2009.

[13] Franka Emika GmbH. *Data Sheet Robot Arm & Controller*, April 2020.

[14] SQLite. Sqlite documentation. Available at https://sqlite.org (09/12/2020).

[15] R. Elmasri and S. B. Navathe. *Fundamentals of database systems*. Person Education, London, United Kingdom, 7th edition, 2016.

[16] Defuse Security. Salted password hashing. Available at https://crackstation.net/hashing-security.htm, 2019.

[17] Datatilsynet. Ha behandlinggrunnlag. Available at https://www.datatilsynet.no/rettigheter-og-plikter/virksomhetenes-plikter/behandlingsgrunnlag/, 2018.

[18] Personopplysningsloven. Lov om behandling av personopplysninger. Available at https://lovdata.no/dokument/NL/lov/2018-06-15-38/*#KAPITTEL_gdpr-4(13/12/2020), 2018.

Appendices

A. Startup problems

During this project we faced a lot of issues trying to get the equipment and the previously developed systems to work. Without a working robot system we were unable to perform tests, and progress on the robot control part of the project was limited. The reason these issues are addressed here in the appendix is that the solution to most of these problems were fairly trivial, but they were also very specific and hard to identify. A lot of time was spent trying to solve them so it is appropriate to address them here.

Network problems

* The first problem we faced after installing the robot system was network connectivity problems. The robot communicates with the Linux system through an Ethernet port, but connection tests showed that the connection was insufficient even though we used exactly the same setup as described in the setup guide from both Franka Emika and Baardsgaard and Brekke. We tried with a range of different Ethernet cables, both brand new ones and cables that was used in previous projects. Problems with packet loss and low success rate on transmission persisted on all of them. Finally two identical network cables were brought in were one of them worked, and the other did not. The exact problem was not identified, but when we found a cable that worked we never had that problem again

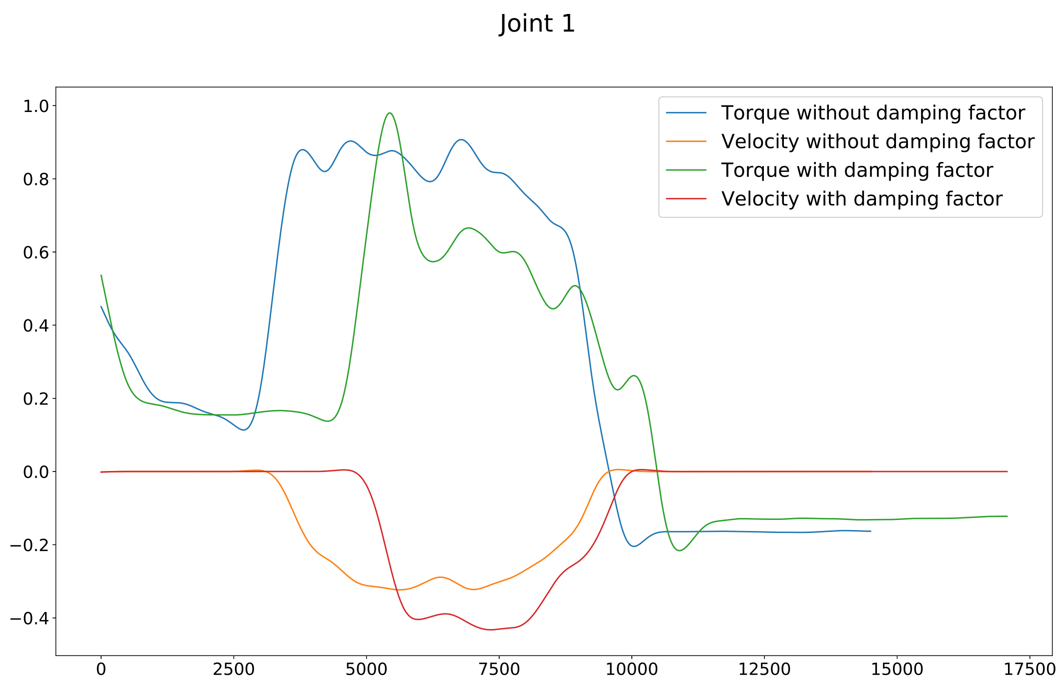
Running previously written code

When the network problems was resolved, the next problem that was encountered was getting the previously written system to run so we could test it out. We used the same PC that was used in the previous projects, so the code was already compiled and should be ready to go. It was also attempted to download the GitHub repository and recompile everything from scratch as described in the setup guide in Baardsgaard and Brekke. The code compiled and built all the executables that was expected, but when attempting to execute them, the program always terminated with a segmentation fault. A segmentation fault normally means there is an error in the code that causes the program to try to access memory that it is not permitted to access. Countless hours were therefore spent trying to find the bug, with debuggers and analysing core dumps. We also had several online meetings with Ove Baardsgaard who wrote the code, but he could not get the code running either. Amund Skavhaug helped us with the problem and recommended a structured approach to finding the error, where we analysed the system piece by piece, reinstalled libraries and tested that everything was built correctly by executing minimal code examples. The solutions turned out to be that the code only executes when it is executed from the integrated terminal in visual studio code.

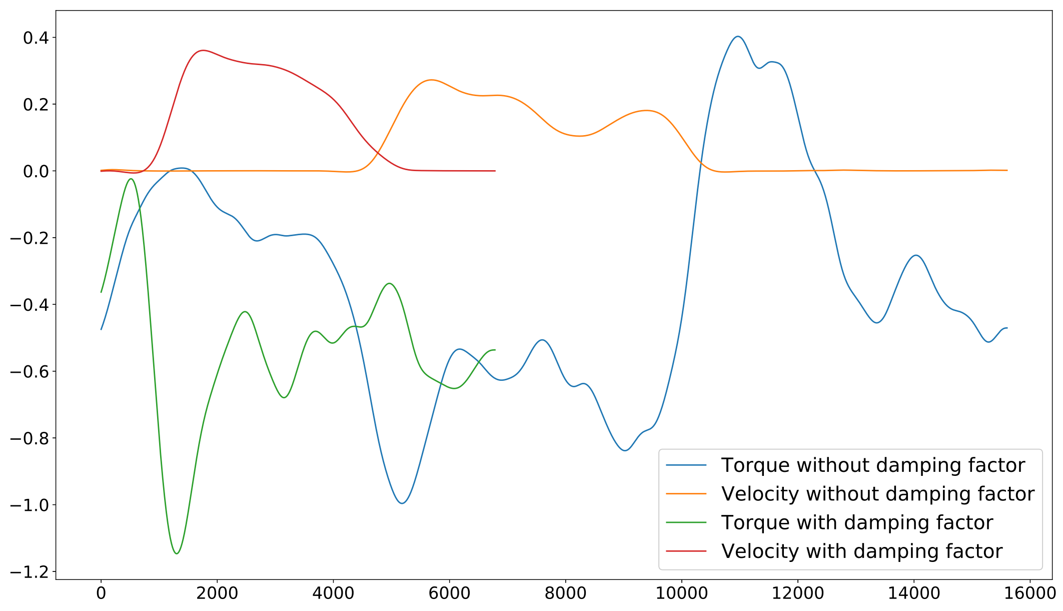
Updating the robot

When the code was working, the next step was to update the robot to the newest firmware version. This also took a while because we needed an invitation from someone with a franka world account registered to the robot. After we managed to get access to the account we upgraded the robot, and another problem occured. After the robot was updated the locking mechanism that is meant to ensure the robot does not fall when unpowered refused to unlock. Log files and information about our problem were sent to Franka Emika Support. When they had completed their analysis they found out to solve this I needed to go into the control panel, change the end-effector setting to the wrong configuration, and then switch it back. This somehow solved the problem. The final issue we faced regarding the update was getting the updated library to work with the new firmware version. When uninstalling the previous version of libfranka, some CMake files and shared object files from the old build were not deleted. This caused CMake to attempt to use the old libfranka build, which was not compatible with the new firmware. Once that was settled the system was finally up and running.

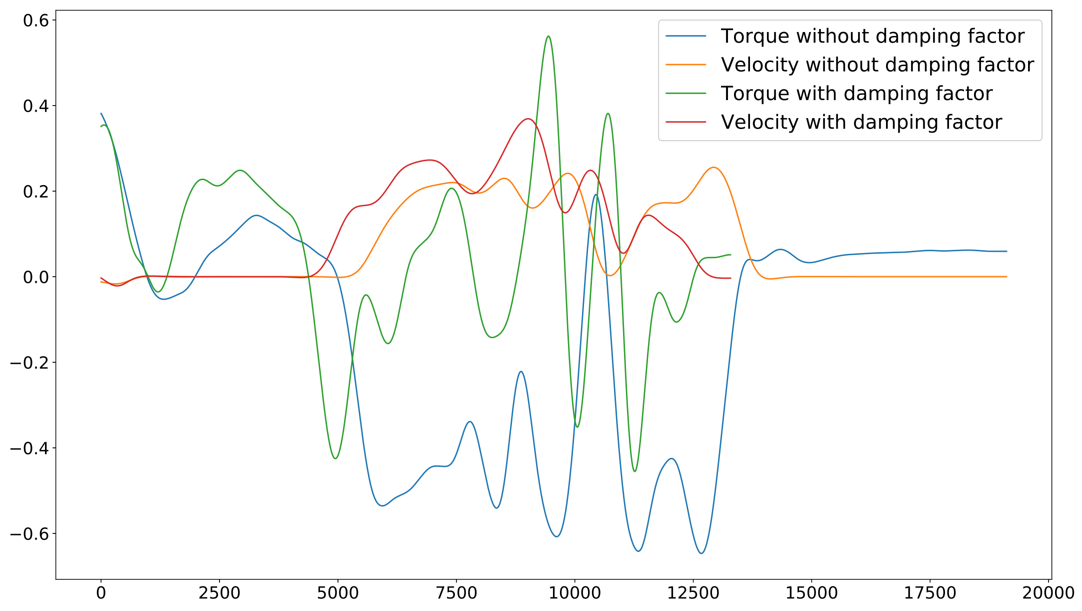
B. All plots from the zero torque test



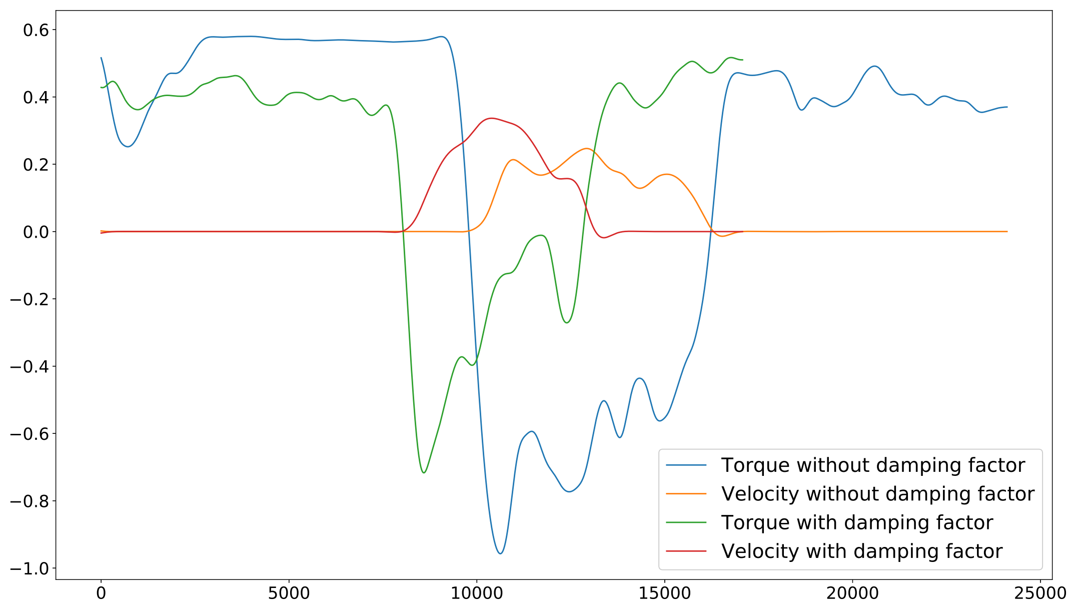
Joint 2



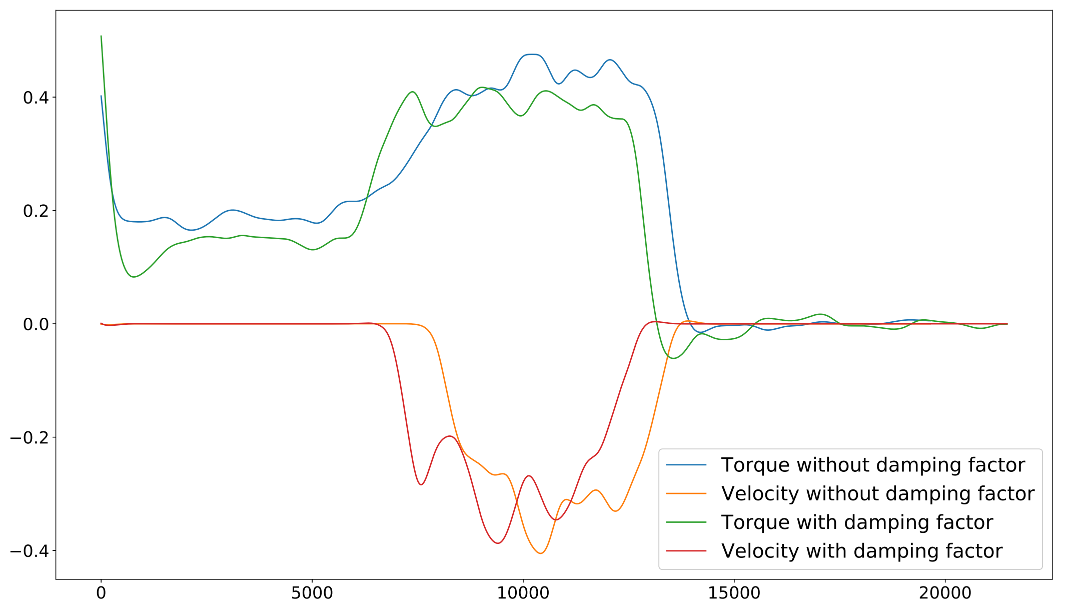
Joint 3

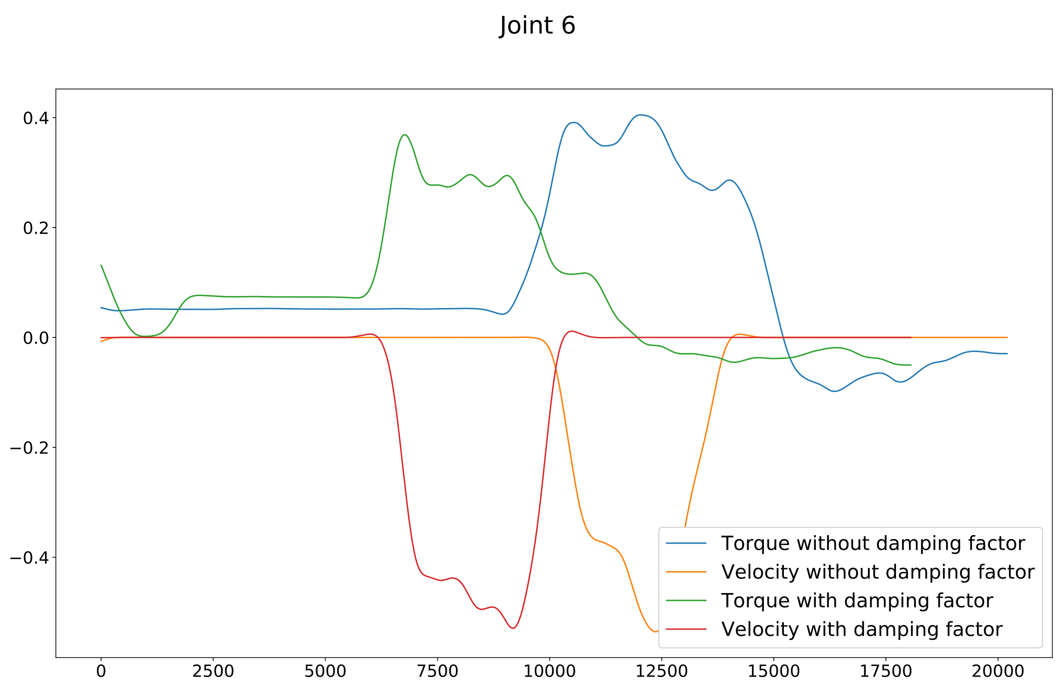


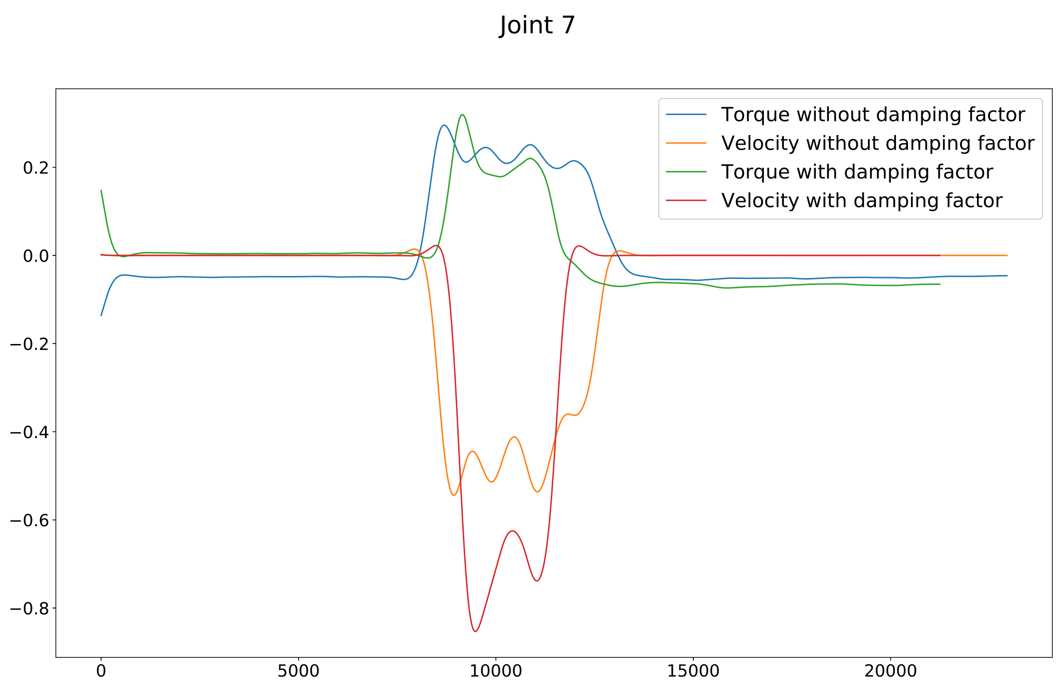
Joint 4



Joint 5




Joint 6
0.4
0.2
0.0
−0.2
−0.4
0
2500
5000
7500
10000
12500
15000
17500
20000
Torque without damping factor
Velocity without damping factor
Torque with damping factor
Velocity with damping factor


Joint 7
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
5000
10000
15000
20000
Torque without damping factor
Velocity without damping factor
Torque with damping factor
Velocity with damping factor

C. Risk Assessment

RISIKOANALYSE (alternativ til bruk av RiskManager)

Enhet/Institutt:	Institutt for maskinteknikk og produksjon	**Dato opprettet:**	30.11.2020
Ansvarlig linjeleder (navn):	Torgeir Welo	**Sist revidert:**	15.12.2020
Ansvarlig for aktiviteten som risikovurderes (navn):	Tobias Fløtre		
Deltakere (navn):	Halvor Romundstad		

Beskrivelse av den aktuelle aktiviteten, området mv.:

Risikoanalysen omfatter utvikling av et robotbasert treningsapparat for behandling av pasienter med nakkeskade I forbondelse med Spesialiseringsprosjekt utført høsten 2020

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)			
Testing av robotarm	*Uønskede bevegelser av roboten treffer mennesker*	*Oppsettet inneholder to ulike nødstoppbrytere, en som stopper roboten gjennom programvare, og en som kutter strømtilførsel.*	*2*	*3*				*6*	*Ha alltid nødstoppbryteren tilgejengelig ved oppstart. Ingen oppholder seg I arbeidsområdet til roboten ved oppstart.*	3
Testing av robotarm	*Uønskede bevegelser treffer objekter I nærheten*	*Oppsettet inneholder to ulike nødstoppbrytere, en som stopper roboten gjennom programvare, og en som kutter strømtilførsel.*	3		3			*9*	*Flytt roboten ut fra veggen slik at krasj ikke er mulig.*	3

E. Risk Assessment

RISIKOANALYSE (alternativ til bruk av RiskManager)

Enhet/Institutt:	Institutt for maskinteknikk og produksjon	**Dato opprettet:**	30.11.2020
Ansvarlig linjeleder (navn):	Torgeir Welo	**Sist revidert:**	15.12.2020
Ansvarlig for aktiviteten som risikovurderes (navn):	Tobias Fløtre		
Deltakere (navn):	Halvor Romundstad		

Beskrivelse av den aktuelle aktiviteten, området mv.:

Risikoanalysen omfatter utvikling av et robotbasert treningsapparat for behandling av pasienter med nakkeskade I forbondelse med Spesialiseringsprosjekt utført høsten 2020

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)			
Testing av robotarm	*Uønskede bevegelser av roboten treffer mennesker*	*Oppsettet inneholder to ulike nødstoppbrytere, en som stopper roboten gjennom programvare, og en som kutter strømtilførsel.*	*2*	*3*				*6*	*Ha alltid nødstoppbryteren tilgejengelig ved oppstart. Ingen oppholder seg I arbeidsområdet til roboten ved oppstart.*	3
Testing av robotarm	*Uønskede bevegelser treffer objekter I nærheten*	*Oppsettet inneholder to ulike nødstoppbrytere, en som stopper roboten gjennom programvare, og en som kutter strømtilførsel.*	3		3			*9*	*Flytt roboten ut fra veggen slik at krasj ikke er mulig.*	3