RAM ROBOTICS AND MECHATRONICS

SOFTWARE FOR DIY AT HOME LAB FOR TELEMANIPULATION CONTROL

F.R. (Famke) van den Boom

BSC ASSIGNMENT

Committee:
dr. ir. D. Dresscher
dr. ir. E. Dertien

July, 2023

043RaM2023
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE
UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Abstract

Robotics avatars allow the user to carry out operations in a remote environment. As a result, the avatar is able to provide care in dangerous places, maintenance in hard-to-reach places and aid despite the distance. One of the technologies used in these avatars is telemanipulation. The goal of this graduation project is to develop a Do-It-Yourself (DIY) kit focussing on kinesthetics within telemanipulation. The client of this project is avatars.report, an online teaching environment with the goal of teaching people more about telemanipulation.

To develop the kit, research is done on the control architectures used within the kit. Moreover, information is collected on the design guidelines within educational teaching material. For if they are implemented correctly, users will be able to learn about kinesthetics within telemanipulation more effectively. Additionally, a stakeholder analysis is performed. With all the findings collected from the research, a requirements list for the DIY kit is constructed. This list serves as a foundation for the continuation of the research.

By creating a variety of iterations, different possibilities were explored as to how the listed requirements could be implemented within the DIY kit. Eventually, a design was chosen based on the theoretically conducted research. After combining the different ideated elements, a specified design was created. The DIY kit consists of a user interface and a hardware setup. With the interface, the user is able to select the architecture, control gain(s) and time delay within the hardware. When adjusting the values within the interface, the user is able to get instant feedback on the working of the control architecture. This feedback is both visually through the interface and physically through interaction with the hardware. The hardware consists of two motor setups which can be turned and continuously communicate with the interface.

The created design of the DIY kit has three main teaching goals, which are obtained. The first goal is that the user is able to see the effect of different control gains on the system. The second goal is that the user is able to see the effect of time delay on the system. The last goal is that the user is able to compare different control architectures.

Through the conduction of this research, guidelines for developing an educational DIY kit focussing on kinesthetics within telemanipulation have been formulated. The guidelines are based on both theoretical knowledge and practical experience. Although not all guidelines have been able to be fully implemented into the kit, it nevertheless provides guidance for creating such a kit. The mentioned guidelines include

1) Accurate implementation of the control architectures.
2) Formulation of requirement lists to determine the purpose of the kit.
3) Inclusion of educational guidelines to make the kit suitable for educational purposes.
4) Accurate parameter tuning to show different kinds of behaviour of the control architectures.

Acknowledgement

This graduation project was not possible without the help of several people. First of all, I would like to thank my supervisor Douwe Dresscher for his guidance and enthusiasm. His critical questions and open mind led me to make my own decisions and justify them. Next to that, I would like to thank my family and friends for their support in the last half year. In particular, I would like to thank Laura Schep for her encouragement and help. Lastly, I would like to thank my study advisor Amaal Shamari. Her guidance let me obtain my potential both personally and academically.

Table of content

List of Figures

List of Tables

Chapter 1 - Introduction

Robots are becoming increasingly present in today's society. From automatic vacuum cleaners to robotic surgery, robots are everywhere. One particular kind of robot is the robotic avatar. Robot avatars are robots that users can control from a distance. As a result, the user can be at places that would normally not be possible. The robots can provide care in dangerous places, maintenance in hard-to-reach places and aid despite the distance. (Ackerman, 2023)

One technology used in robotic avatars is referred to as telemanipulation. Melchiorri (2003) describes telemanipulation as 'the capability of a human being to carry out operations in a remote environment by means of a proper robotic system'. Avatars.report is an online teaching platform with the goal of learning people more about telemanipulation (Avatar Robotics, n.d.). In order to do so, the platform offers pen casts, assignments and Do-It-Yourself (DIY) kits. This graduation project aims to develop the software for one of these DIY kits. The DIY kits are physical setups where users can experiment with the topics covered in the pen casts. The DIY kit covered in this bachelor graduation project focussed on kinesthetic telemanipulation. Within kinesthetic telemanipulation, the user feels the feedback of the forces directly in their muscles and bones.

In order to develop this DIY kit, one main research question is constructed:

"What are the most important guidelines when developing software for an educational DIY kit focussing on kinesthetic telemanipulation?".

To be able to answer this question, multiple sub-questions are defined:

- *"How are the four different controller architectures constructed?"*
- *"Which aspect(s) of telemanipulation should be highlighted in the product?"*
- *"What are the most important concerns when designing software for education?"*
- *"How can the control gain range within the controller be determined?"*

This report follows the guidelines from 'A design process for Creative Technology' by Mader & Eggink (2014). This method suggests three steps in the design process; the ideation, the specification and the realisation phase. All steps within the process are cyclic, thus allowing for multiple feedback loops. As new knowledge is obtained and new questions arise along the way, it is exceedingly relevant to return to previous stages and create product iterations.

Chapter 2 - Background

2.1 Control Architectures

Within this graduation project, four different control architectures will be implemented. In this chapter, the general setup will be discussed. Next to that, the goal of the controller will be explained and the structure of four different architectures will be discussed. The information discussed in this chapter is based on lectures on avatars.report (Avatar Robotics, n.d.).

2.1.1 General Setup

Before diving into the specific architectures, a general view of the system is given. The setup consists of two robot arms; one master arm interacting with the human and one slave arm interacting with the environment. In order to let these two systems interact with each other, the controller exchanges velocity and force from the master to the slave and vice versa. A general view of the system can be found in Figure 1.

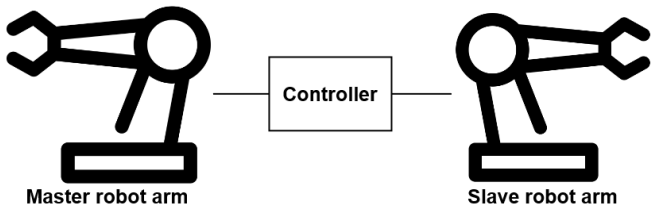

Figure 1: General setup of the system

Within each robot arm, three powerports can be defined; the interaction port, the robot dynamics port and the controller port. Each of these powerports is a power-conjugated couple consisting of a velocity and a force. In the report, the velocity and forces are referred to with a subscript. In the subscript, the first letter refers to either the human (h) or the robot (r) side and the second letter refers to the powerport, either interaction (i), dynamics (d) or controller (c). In Figure 2, a bond graph of the system is presented. For information about bond graphs, the book 'Introduction to physical systems modelling with bond graphs' by Jan B Broenink can be consulted (Broenink, 2000).

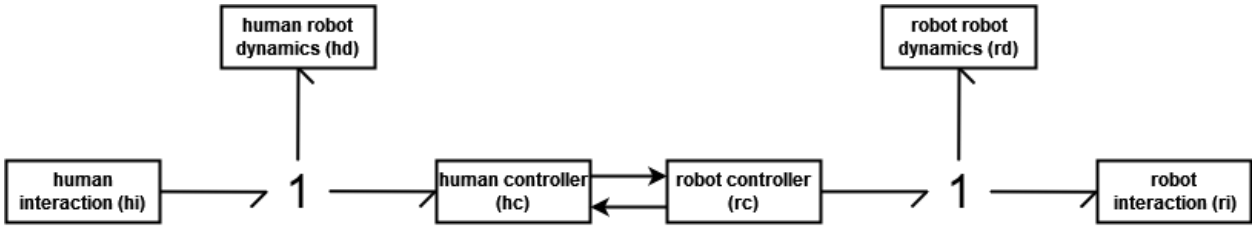

Figure 2: Bond graph of the system

In the bond graph, one can identify two one-junctions meaning that the powerports have a common velocity and a sum of forces. The according formulas for the forces are $F_{hc} + F_{hd} = F_{hi}$ (on the human side) and $F_{ri} + F_{rd} = F_{rc}$ (on the robot side). The following equality can be found for the velocity: $v_{hc} = v_{hd} = v_{hi}$ (human side) and $v_{rc} = v_{rd} = v_{ri}$ (robot side). For the system, the assumption is made that the velocity can be interchanged by the position as the two are time-related. Of course, this interchanged can only happen if all velocities are changed into positions or vice versa.

2.1.2 Challenges

The goal of the control system is to achieve perfect transparency. In the case of the system presented in 2.1.1, this means two things. First, the velocity of the interaction at the human side (v_{hi}) should be equal to the velocity of the interaction at the robot side (v_{ri}). Second, the force of the interaction on the human side (F_{hi}) should be equal to the force of the interaction on the robot side (F_{ri}). However, when trying to achieve this, two main challenges arise: dynamics and time delay.

2.1.2.1 Device Dynamics

The first challenge when trying to achieve perfect transparency is device dynamics. When looking at the controller, the aim is to have a good energetic connection. In order to obtain this, the following connection is assembled: The velocity of the controller on the human side is equal to the velocity of the controller on the robot side, and the force of the controller on the human side is equal to the force of the controller at the robot side. In short: $v_{hc} = v_{rc}$ and $F_{hc} = F_{rc}$. When looking at the sum of forces discussed in 2.1.1, the following can be derived: $F_{hi} = F_{hc} + F_{hd} \rightarrow F_{hi} = F_{rc} + F_{hd} \rightarrow F_{hi} = F_{rd} + F_{ri} + F_{hd} \neq F_{ri}$. One can see that $F_{hi} \neq F_{ri}$, but instead $F_{hi} = F_{rd} + F_{ri} + F_{hd}$, meaning that perfect transparency is not achieved and that device dynamics (F_{rd}, F_{hd}) play an important role. A countermeasure to reduce device dynamics is dynamics compensation.

2.1.2.2 Time Delay

The second challenge when trying to achieve perfect transparency is time delay. As there is a time delay within the communication channel, the position on the robot side is never the same as the position on the human side. Instead, the position on one end of the communication channel is the delayed position of the other end of the communication channel. This same principle applies to the forces within the system, making it impossible to reach perfect transparency. Due to this time delay, additional energy is generated within the system, causing instability.

2.1.3 Position-Computed Force Architecture

The position-computed force architecture is based on the controller representing a (software) spring. As the robot is a mass, one cannot set the velocity directly but only set the force. To tackle this problem, the controller is represented as a spring. With a spring, the velocity can be set. A realisation of the position-computer force architecture can be seen in Figure 3.

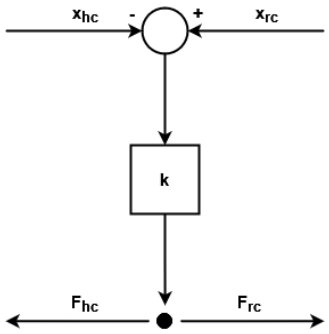

Figure 3: *Block diagram of the position-computed force architecture*

The architecture takes the difference between the position of the human side of the controller and the position of the robot side of the controller. These positions represent the position of the human interaction and the position of the robot interaction, respectively. In order to derive the control force, the position difference is multiplied by a spring constant k. The equation for this is F = k*x. In other words, the larger k, the more force it takes to expand the spring. Having a larger value for k is the characteristic preferred in the (software) spring, as the difference in velocity will be minimal.

2.1.4 Position-Measured Force Architecture

The position-measured force architecture is quite similar to the position-computed force architecture. However, instead of computing the force of the human controller, the force of the human controller is set to the measured force of the robot interaction. Doing this has one big advantage: the influence force of the robot dynamics on the robot side is removed. The realisation of the position-measured force architecture can be seen in Figure 4.

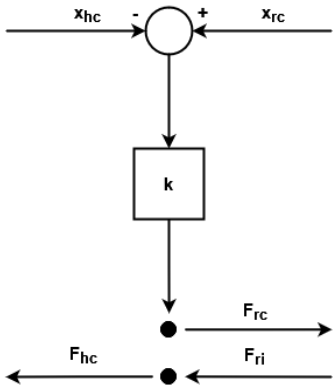

Figure 4: *Block diagram of the position-measured force architecture*

Looking at the block diagram, one can see that $F_{hc} = F_{ri}$. Using this and the equation of the sum of forces, $F_{hi} = F_{hc} + F_{hd}$ (discussed in 2.1.1), one finds $F_{hi} = F_{ri} + F_{hd}$. One can also derive the F_{hi} within the position-computed force architecture to confirm that the robot dynamics are indeed removed. In this architecture, there holds $F_{hc} = F_{rc}$, in combination with the sum of forces ($F_{rc} = F_{ri} + F_{rd}$), one will obtain F_{hc}

= F_{rc} = F_{ri} + F_{rd}. When combining this equation with the sum of force on the human side, F_{hi} = F_{hc} + F_{hd}, one will obtain F_{hi} = F_{ri} +F_{rd} + F_{hd}. As can be seen, the robot dynamics are included in the position-computed force architecture. This confirms the advantage of the position-measured force architecture as one can see that F_{rd} is not included in the equation for the position-measured force architecture, improving transparency.

The transparency is also improved when looking at the (software) spring in the controller. Due to the removal of the spring interaction on the human side, the spring is no longer part of the communication over the communication line. Instead, the spring is now local and only part of the equations on the robot side. As a result, the spring will no longer cause an increase in energy due to time delay. Therefore it is possible to increase the spring constant and thus the transparency.

2.1.5 Position-Position Architecture

The position-position architecture is designed based on feedback control. With feedback control, the goal is to let the feedback signal and the setpoint be equal. To reach this goal, a feedback control loop is used. An example of such a control can be seen in Figure 5.

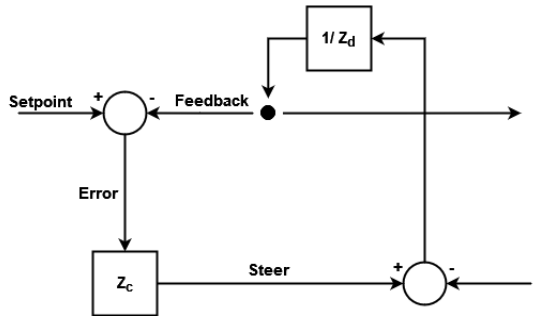

Figure 5: *Basic design of a feedback control loop*

In Figure 5, one can identify the four different signals; the setpoint, the feedback, the error and the steer signal. In feedback control, the value being controlled, the feedback signal, is measured and compared with a target value, the setpoint. The difference between these two signals is called the error signal. This error signal is transferred into a steer signal by multiplication of a certain Z_c. Z_c resembles the control impedance. This steer signal manipulates an input to the system to minimise this error allowing the feedback signal and the setpoint to be equal. (Kim et al., 2009)

In the position-position architecture, two of these feedback control loops are presented; one on the human side and one on the robot side. A realisation of the position-position architecture can be seen in Figure 6. Both the feedback loops take the difference between the position of the human controller and the position of the robot controller. These differences are then multiplied by either Z_{hc} or Z_{rc}

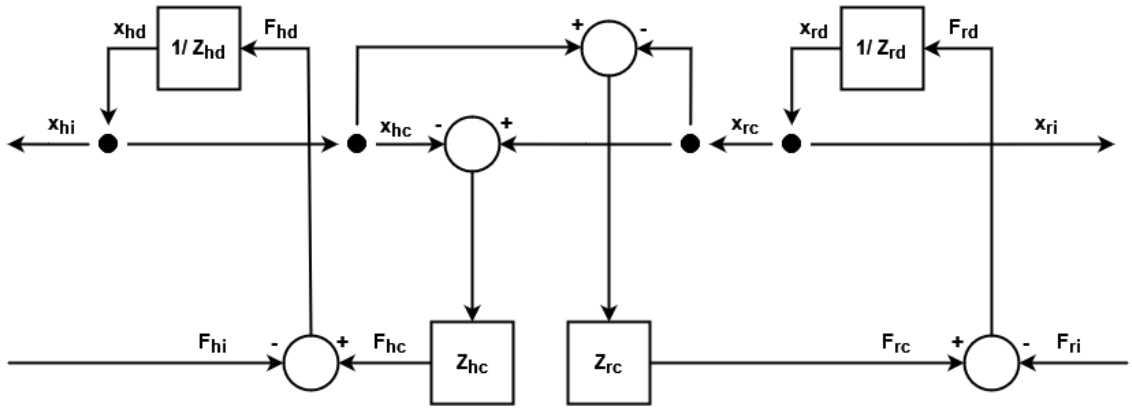

Figure 6: *Block diagram of the position-position architecture*

When looking at the goal of perfect transparency, one can also say something about the values of Z_{hc} and Z_{rc}. The goal of the feedback controller is to make the setpoint and the feedback signal equal. At the control loop at the robot side, this means that x_{rc} should be equal to x_{hc}. From the block diagram, the following equations can be derived: $F_{rc} = Z_{rc} * (x_{hc} - x_{rc})$ and $x_{rc,} = 1/Z_{rd} * (F_{rc} - F_{ri})$. When combining these equations, one will get $x_{rc,} = 1/Z_{rd} * (Z_{rc} * (x_{hc} - x_{rc}) - F_{ri})$. This equation can be rewritten as $x_{rc} = \frac{Z_{rc}}{Z_{rd}+Z_{rc}} x_{hc} - \frac{1}{Z_{rd}+Z_{rc}} F_{ri}$. In order to reach the control goal ($x_{rc} = x_{hc}$), Z_{rc} should go to infinity. This same derivation can be done for the control loop on the human side, and one will find that Z_{hc} should go to infinity.

One can also say something about the relation between Z_{hc} and Z_{rc}. Looking at the block diagram one can derive equations for both F_{hi} and F_{ri}: $F_{hi} = F_{hc} - F_{hd}$ and $F_{ri} = F_{rc} - F_{rd}$. These equations can be rewritten into $F_{hi} = Z_{hc} * (x_{ri} - x_{hi}) - F_{hd}$ and $F_{ri} = Z_{rc} * (x_{hi} - x_{ri}) - F_{rd}$. When looking at the equation for the robot interaction force formula, the following relation can be seen: $(x_{hi} - x_{ri}) = (F_{ri} + F_{rd}) / Z_{rc}$. Combing this equation with the equation for the human interaction force will give the following relation: $F_{hi} = -Z_{hc}/Z_{rc} * F_{ri} - Z_{hc}/Z_{rc} * F_{rd} - F_{hd}$. When $Z_{hc} = Z_{rc} \rightarrow \infty$ and $F_{rd} = F_{hd} = 0$, perfect transparency is reached.

2.1.6 4-Channel Architecture

The 4-channel architecture is based on feed-forward control. Feed-forward control can be applied to the system's velocity and force. A realisation of the 4-channel architecture can be found in Figure 7.

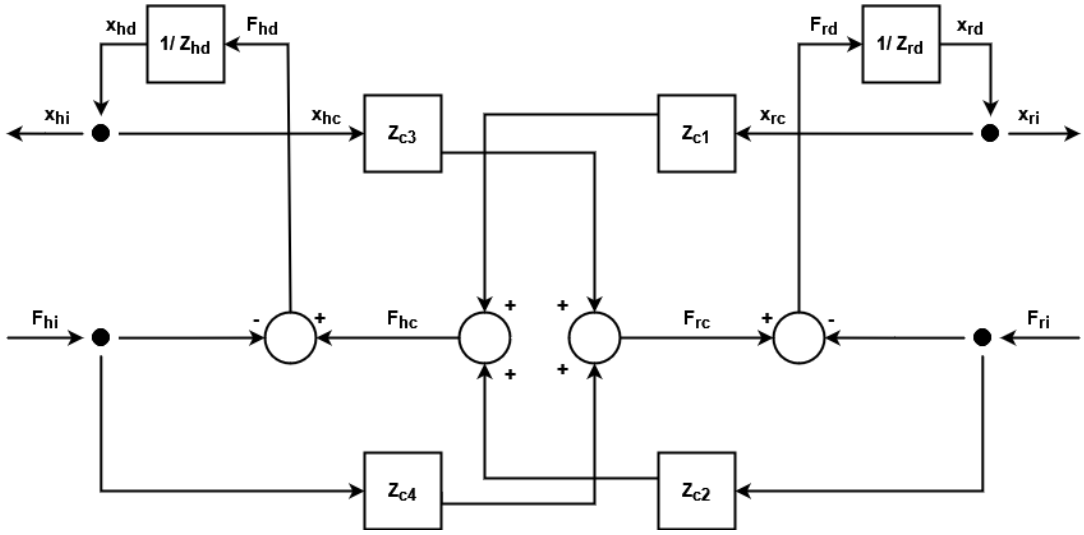

Figure 7: *Block diagram of 4-channel architecture*

Multiple equations can be derived when looking at the robot side of the controller. First off, the velocity of the robot interaction, $v_{ri} = 1/ Z_{rd,} * (F_{rc} - F_{ri})$. Second, the force of the robot controller, $F_{rc} = Z_{c3} * v_{hi} + Z_{c4} * F_{hi}$. Combining these equations will give: $v_{ri} = 1/ Z_{rd,} * (Z_{c3} * v_{hi} + Z_{c4} * F_{hi} - F_{ri})$. Also, multiple equations can be derived on the human side of the controller. First off, the force of the human interaction: $F_{hi} = F_{hc} - F_{hd}$. Second, the force of the human controller: $F_{hc} = Z_{c1} * v_{rc} + Z_{c2} * F_{ri}$. Combining these equations will give $F_{hi} = Z_{c1} * v_{rc} + Z_{c2} * F_{ri} - F_{hd}$.

One is also able to say something about the values for all four of the control gain. As discussed in 2.1.2 the goal is to achieve perfect transparency: $F_{hi} = F_{ri}$ and $v_{hi} = v_{ri}$. Looking at $v_{ri} = 1/ Z_{rd,} * (Z_{c3} * v_{hi} + Z_{c4} * F_{hi} - F_{ri})$, one can see that perfect transparency is achieved when $Z_{c4} = 1$ and $Z_{c3} = Z_{rd}$. The same can be done by looking at $F_{hi} = Z_{c1} * v_{rc} + Z_{c2} * F_{ri} - F_{hd}$. Perfect transparency is achieved when $Z_{c1} = Z_{hd}$ and $Z_{c2} = 1$.

2.1.7 Summary of Control Architectures

The setup used in the DIY kit consists of two robot arms connected by a controller. The goal of this controller is to reach perfect transparency. This goal is reached when $F_{hi} = F_{ri}$ and $x_{hi} = x_{ri}$. The two main challenges when reaching perfect transparency are time delay and device dynamics. In the DIY kit, four different control architectures are implemented: position-computed force control, position-measured force control, position-position control and the 4-channel architecture. Each of these controllers makes use of its own control gains. The optimal control gains to achieve perfect transparency can be found in Table 1.

Control architecture	**Optimal control gain(s)**
Position-computed force control	$k \rightarrow \infty$

Position-measured force control	$k \rightarrow \infty$
Position-position control	$Z_{rc} \rightarrow \infty$ $Z_{hc} \rightarrow \infty$ $Z_{rc} \approx Z_{hc}$
4-Channel architecture	$Z_{c1} = Z_{hd}$ $Z_{c2} = 1$ $Z_{c3} = Z_{rd}$ $Z_{c4} = 1$

Table 1: *Optimal control gains for each control architecture*

Chapter 3 - Analysis

3.1 Requirements Setting

In order to make a suitable end product, requirements have to be identified. These requirements help to make choices in the design process and validate the product at the end. In this chapter, the requirements list for the software part of the DIY kit is constructed using the MoSCoW-method developed by Clegg & Barker (1994). The development of requirements is split into four sections to cover all aspects of the project. First, a stakeholder analysis is performed. Second, the scope of the DIY kit is discussed. Third, the input-output compatibility is analysed. Last, the education aspects are researched. Combining all the elements provides the requirements list at the end of the subsection.

3.1.1. MoSCoW-Method

The requirements listed in this chapter are composed according to the MoSCoW-method. The MoSCoW-method prioritises requirements in 4 categories; Must Have, Should Have, Could Have and Will Not Have. The requirements in Must Have are essential for the finalisation of the project. The project is only safe, legal, effective and complete with these requirements. Going one step down, requirements in the Should Have category can be found. These requirements are optional for the product. Nevertheless, they are still important for finishing the product. The requirements in the Could Have category have a smaller impact on the product. These requirements are most of the time desirable, yet omittable. The last category is Will Not Have. These requirements have no priority to the current project. However, they can be implemented in future work. (Brush, 2023);(MoSCoW Prioritization, 2022)

3.1.2 Stakeholders Analysis

A stakeholder analysis is essential as it identifies the different parties and their needs in the project. Considering these needs, one can create the best solution for all stakeholders. In this section, first, the stakeholders are identified. Afterwards, the stakeholders are classified using the power-interest matrix developed by Mendelow (1981).

3.1.2.1 Stakeholder Identification

The first step in a stakeholder analysis is to identify all the stakeholders. In this graduation project, four stakeholders are presented; avatars.report, the end-users of the project, the collaborator and the University of Twente.

First off, the client in the graduation project, avatars.report. avatars.report is an online teaching platform with the goal of informing people about the working of robots. The platform informs its users in three ways; using pre-recorded lectures (pen casts), exercises and do-it-yourself (DIY) kits. The user can buy points in the webshop to access these information sources. By spending the points, the users can unlock

the subjects they would like to learn, giving the user control of their learning process. The goal of avatars.report in the project is to have functional, cheap and accessible DIY kits.

The second stakeholder is the target group of the project. avatars.report targets their teaching environment at people interested in learning more about kinetics in robotics with some basic prior technical knowledge. The environment is currently only used by Robotics and Interaction Technology students at the University of Twente. However, the education tool is also aimed at people working in the technology sector. The user's goal when using the avatars.report platform is to learn more about kinetics through interactive and intuitive DIY kits.

The third stakeholder is the collaborator in the project, Frank Bosman. The project's scope is to focus on the software and control aspects of the DIY kit; Frank focuses on the hardware of this project. His goal in the project is software that is compatible with the developed hardware. As one end project is delivered, close cooperation is needed.

The last stakeholder is the University of Twente. The university is the stakeholder that provides and gives feedback on the project. Moreover, they are responsible for the assessment of the result. Their goal is to let the project's development process comply with the graduation guidelines set by the Creative Technology program.

3.1.2.2 Power-Interest Matrix

Now that the stakeholders have been identified, a more insightful discussion of the stakeholders can be done using the power-interest matrix. As the name suggests, the power-interest matrix gives information on each stakeholder's power and interest. Stakeholders with much power should be satisfied, whereas people with high interests should be informed (Every, 2022). The power-interest matrix with the stakeholders discussed in 3.1.2.1 can be found in Figure 8.

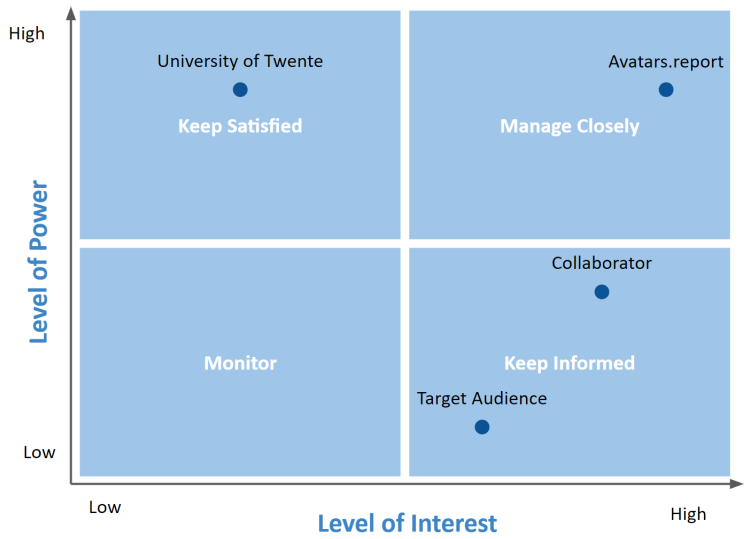

Figure 8: *Power-interest matrix of the stakeholders presented in the project*

Figure 8 shows that the collaborator and the target audience fall within the 'Keep Informed' section. These two stakeholders should be informed regularly as they are usually helpful in considering the product details (Every, 2022). avatars.report is the stakeholder in the 'Manage Closely' section. The stakeholder will likely impact decision-making and the project's success (Every, 2022). As a result, close consideration is needed to manage their expectations. The last stakeholder, the University of Twente, falls within the 'Keep Satisfied' section. Where the university has no direct interest in the project, they do have much influence. To bring the project to a successful end, the university should be kept in the loop.

3.1.2.3 Stakeholder Requirements

Using the MoSCoW-method discussed in 3.1.1, seven stakeholder requirements can be formulated. These requirements are grouped per stakeholder.

As stated in 3.1.2.1, the goal of the client, avatars.report, is to teach users about kinesthetic telemanipulation. Therefore the kit must cover this topic (req. 1) and should be effective in teaching it (req. 2). Next to that the client aims for the DIY kit to be easy to use, thus it should be intuitive (req. 3). Lastly, the client aims for keeping the DIY kits as cheap as possible. Hence, the designed kit should be affordable (req. 4).

avatars.report & Target group:

Req. 1: The DIY kit must cover kinesthetic telemanipulation.
- This requirement is further elaborated in section 3.1.3.

Req. 2: The DIY kit should be an educational tool.
- This requirement is further elaborated in section 3.1.5.

Req. 3: The DIY kit should be intuitive.

Req. 4: The DIY should be affordable.

From the stakeholder identification, another set of requirements is formulated. Close cooperation is needed to make the project work accordingly. To achieve this, the hardware and software should be compatible (req. 7). Next to that, it is a must for both the hardware and software to be finished in time (req. 5 and req. 6).

Collaborator Frank:

Req. 5: The hardware of the DIY kit must be done in time.

Req. 6: The software of the DIY kit must be done in time.

Req. 7: The hardware and software in the kit must be compatible.
- This requirement is further elaborated on in section 3.1.4

From the University of Twente, the end-product must adhere to the guidelines of the study's Graduation Project (req. 8).

University of Twente:
Req 8: The kit's development must adhere to the guidelines of a Creative Technology Graduation Project set by the University of Twente.

3.1.3 The Scope of the DIY Kit

Considering requirement 1 in the previous subsection, barriers regarding the topics discussed in the DIY kit should be set. In the video material provided by avatars.report, multiple different topics are discussed. Over three videos, avatars.report discusses four different architectures: position-computed-force control, position-measured-force control, position-position control and the 4-channel architecture. In discussion with the client, avatars.report, the goal is to cover all these four architectures within the kit. However, more than just the architectures are presented in the kit. As avatars.report wants to give the user insight into the more detailed working of the architecture, the control gains in the controllers are also covered. Users can change the control gain and feel the different responses of the controller. Another topic covered in the kit is dynamic compensation. As discussed in 2.1.7, dynamic compensation aims to make an even more accurate controller by compensating the robot dynamics. The dynamic compensation can the user again feel when using the kit. The last topic covered in the kit is time delay. Time delay impacts the energy generated in the system. The more significant the time delay, the more energy is generated in the system, leading to an unstable system.

3.1.3.1. Scope Requirements

Now that the scope of the kit is more elaborated on, requirement 1, 'The DIY kit must cover kinesthetic within telemanipulation,' can be extended. This is done by setting the following sub-requirements.
Req. 1a: The kit must cover the four architectures; position-computed-force control, position-measured-force control, position-position control and the 4-channel architecture.
Req. 1b: The user must be able to adjust the controller gains within the kit.
Req. 1c: The kit should cover dynamics compensation.
Req. 1d: The kit should cover time delay, with the option for the user to adjust the time delay.

Requirements 1a en 1b are included in the Must category, as the goal of the DIY kit is not accomplished without these requirements. Meanwhile, requirements 1b and 1c are an extension to the kit's goal and are therefore included in the Should category.

3.1.4 Input-Output Compatibility

As described in requirement 7, input-output compatibility has to be considered. In 2.1, four control architectures are mentioned, each with inputs and outputs. An overview of these inputs and outputs per architecture can be found in Table 2.

Architecture	**Input**	**Output**

Position-computed force	Position of human Position of robot	Force of human controller Force of robot controller
Position-measured force	Position of human Position of robot Force of the robot interaction	Force of human controller Force of robot controller
Position-position	Position of the human Position of the robot	Force of human controller Force of robot controller
4-channel	Force of human interaction Force of robot interaction Position of the human Position of the robot	Force of human controller Force of robot controller

Table 2: *Overview of inputs and outputs per control architecture*

3.1.4.1 Input-Output Requirements

For the hardware and software to be compatible, two sub-requirements are formulated:
Req. 7a: The hardware and software must be able to handle a force output.
Req. 7b: The hardware and software must be able to handle a position and force input.

3.1.5 Educational Aspects

The DIY kit focuses on learning for the user. As a result, the kit should be designed according to educational guidelines. Gagné et al. (2004) presented nine universal steps of instruction in their book 'Principles of instructional design' . The University of Illinois Springfield adapted these steps to develop web-based education (Elements of Instruction, n.d.). The Northern Illinois University complements this by presenting methods to accomplish these outcomes (Gagne's nine events of Instruction: Center for Innovative Teaching and Learning, n.d.). A summary of this theory is given below.

The first step is the attention of the user. The material should be appealing and engaging in grasping the user's attention. Methods for accomplishing this include but are not limited to colour, animations and sounds. The second step is to set objectives for the user. It is essential to set goals at the beginning of the material. Doing this will help the user to focus on the relevant information, giving them control of their learning process. Methods for accomplishing this included describing the required performances and course objectives. The third step focuses on recalling prior learning. By linking previous knowledge to new material, users learn more quickly. Methods for accomplishing this are asking questions about prior knowledge and relating prior information. The fourth step is about presenting the content. There are multiple ways of presenting information to the user. It is essential to match the information style to the learning objectives. Methods for accomplishing this are presenting materials in multiple ways and incorporating active learning methods. The fifth step is providing guidance to the user. This step will help the user in learning how to learn. Techniques for accomplishing this include providing real-world applications and using examples. The sixth step focuses on eliciting involvement. By letting users work on

the material, they process the information more quickly. Methods for accomplishing this include practical exercises and assessment opportunities. The seventh step is providing feedback. By giving feedback, learning gaps are identified. Techniques for achieving this include confirmatory feedback (informing the user that they accomplished the right thing) and analytic feedback (giving suggestions to the user to improve their learning). The eighth step is assessing performance. With this step, the goal is to check if the user understood the learning material. Methods for accomplishing this included tests and quizzes. The last step is offering enrichment of the learning material. With this step, users learn more than just the covered material. Methods for accomplishing this include promoting deep learning and incorporating other learning content. An overview of the different steps and their accomplishing methods can be found in Table 3.

Step	**Goal**	**Methods for accomplishing the goal**
1	Gain attention	- Stimulating using novelty and surprise - Use colour, animation and sounds
2	Set objectives	- Describe course objectives - Describe criteria for standard performance
3	Recall prior learning	- Relate prior knowledge to the topic - Let the user incorporate prior knowledge - Ask questions about prior knowledge
4	Present content	- Present the learning material in multiple ways - Incorporate active learning method
5	Provide guidance	- Use examples and non-examples - Provide real-world application
6	Elicit performance	- Include practical exercises - Give assessment opportunities
7	Provide feedback	- Give suggestions and information - Inform the user about their process
8	Assess performance	- Implement a variety of assessments
9	Offer Enrichment	- Incorporate other learning content - Promote deep learning

Table 3: *Gagne's nine universal steps of instruction*

As the DIY kit is only a part of the educational material provided by avatars.report, it is not possible to feature all nine steps in the kit. Nevertheless, the design of the DIY material should keep in mind these guidelines. For the kit, the following steps are included; step 1 (Gain attention), step 2 (Set objectives), step 3 (Recall prior learning), step 4 (Present content), step 5 (Provide guidance), step 6 (Elicit performance) and step 7 (Provide feedback).

In the DIY kit, it is crucial to gain the user's attention (step 1). First, the users are more likely to buy the DIY kit as an income for avatars.report. Additionally, an appealing kit will engage the user to keep on learning via the kit. For the software, this means making an engaging and appealing interface for the user. The goal to set the objectives (step 2) is included in the documentation of the software. In the documentation, the expectations and goals of the kit are described, giving the user a clear idea of what they can expect and what they need to do with the kit. Providing guidance (step 5) is also included in this documentation. In the documentation, multiple parameter combinations are given as examples. These examples include both (relative) optimal configurations of the controllers and also non-examples, configurations of the controllers in which the architectures do not work optimally. Step 4 (Present content) and step 6 (Elicit performance) are presented in the DIY kit. The kit is a different way of learning than the other materials provided by avatars.report. It gives the user an interactive, real-world setup with which the user can play. Step 3 (Recall prior learning) is also incorporated. As the content discussed in the pencasts is referenced within the DIY kit, users can link back to their prior knowledge and incorporate it. The last step that is included, step 7 (Providing feedback), is presented within the user interface. The user gets feedback on the used parameters within the control architecture. This feedback can be felt physically within the setup and seen visually on the interface.

3.1.5.1 Educational Requirements

The second requirement can be enhanced using six of Gagne's nine universal steps of instruction, as discussed in the previous section. The second requirement states, 'The DIY kit should be an educational tool' using the following five sub-requirements, this requirement can be specified:
Req. 2a: The user interface should be engaging and appealing.
Req. 2b: The kit's documentation should include the kit's expectations and goals.
Req. 2c: The documentation of the kit should include optimal and not optimal configurations of the controller
Req. 2d: The kit should be an interactive, real-world setup
Req. 2e: The user interface should give feedback to the user regarding the selected parameter configuration.

3.1.6 Requirements List

To conclude, requirements are essential as they set guidelines regarding the development and design of the product. In this subsection, multiple aspects of the kit were analysed, and requirements were drafted. A general list of all requirements can be found below.

Requirement list for the DIY kit:
Req. 1: The DIY kit must cover kinesthetic within telemanipulation.

- Req. 1a: The kit must cover the four architectures; position-computed-force control, position-measured- force control, position-position control and the 4-channel architecture.
- Req. 1b: The user must be able to adjust the controller gains within the kit.
- Req. 1c: The kit should cover dynamics compensation.

- Req. 1d: The kit should cover time delay, with the option for the user to adjust the time delay.

Req. 2: The DIY kit should be an educational tool
- Req. 2a: The user interface should be engaging and appealing.
- Req. 2b: The kit's documentation should include the kit's expectations and goals.
- Req. 2c: The documentation of the kit should include optimal and not optimal configurations of the controller
- Req. 2d: The kit should be an interactive, real-world setup
- Req. 2e: The user interface should give feedback to the user regarding the selected parameter configuration.

Req. 3: The DIY kit should be intuitive.

Req. 4: The DIY should be affordable.

Req. 5: The hardware of the DIY kit must be done in time.

Req. 6: The software of the DIY kit must be done in time.

Req. 7: The hardware and software in the kit must be compatible.
- Req. 7a: The hardware and software must be able to handle a force output.
- Req. 7b: The hardware and software must be able to handle a position and force input.

Req 8: The kit's development must adhere to the guidelines of a Creative Technology Graduation Project set by the University of Twente.

3.2 Method Setting

After the requirements list is drafted, information regarding the methods is selected. In this subsection, research is done on which methods exist for reaching the requirements.

3.2.1 Microcontroller

Multiple sensors and actuators are used to measure and actuate the inputs and outputs, as discussed in 3.1.4. Detailed information about these sensors and actuators can be found in the bachelor thesis of Frank Bosman. The sensor and actuators are connected to a microcontroller to process the inputs and send the outputs. Research is done to choose a suitable microcontroller for the DIY kit. Due to time constraints, only two microcontrollers are being considered for the kit: the Arduino Uno and the ESP32.

The Arduino Uno is a microcontroller developed by Arduino. The Arduino board has 14 digital input/output pins and six digital pins. The board can be programmed using the Arduino IDE (integrated development environment). Some standard libraries are already available within the IDE, and more can be installed, making it an easy-to-use and user-friendly programming environment. The microcontroller is priced at €24.00 via the original Arduino store. Last, the Arduino Uno has 32 KB of flash memory. (Arduino Uno REV3, n.d.);(Tan,2023)

The other option is an ESP32, a microcontroller developed by Espressif Systems. The ESP32 is a 32-bit microcontroller with a 2.4 GHz Wi-Fi and Bluetooth chip. The ESP32 has no IDE but can be connected and programmed via the Arduino IDE using an add-on. The ESP32 has up to 32 digital and 12 analog pins and

is priced at €9.50. Last, the ESP32 has 512 kB of RAM. (ESP32 WIFI and Bluetooth board - CP2102, n.d.);(Tan,2023)

In the Specification phase, the final microcontroller will be chosen.

3.2.2 Programming Language

The microcontroller is connected to a computer to show the user interface. This interface lets the user, among others, select different control architectures and adjust the control gains. For making this interface, multiple programming languages and IDEs are considered. Due to time constraints, the languages that are being considered include Python and Processing.

Processing is a visual programming language designed for easy prototyping and rapid visualisations. The language is based on Java and is aimed at beginners and expert programmers. It has its own IDE and is used repeatedly within the Creative Technology program. (Gelal, 2015)

Python is programming released in 1991 and used for, amongst others, creating web applications, creating workflows, connecting to database systems and making rapid prototypes. It has a simple syntax and can be used on all different platforms. (Python introduction, n.d.)

In the Specification phase, the final programming language will be chosen.

3.2.3 Parameter Tuning

To provide an out-of-the-box working DIY kit, the parameters in the control architectures should be tuned. The tuning is done using two different methods; system characterisation and trial and error. System characterisation is applied to the control architecture with feed-forward control, the 4-channel architecture. Next to that, system characterisation is used for dynamics compensation. Within system characterisation, the goal is to make a model of the system and find the value of the corresponding parameter by experiments. This method is executed similarly to lab 1 of Modelling & Contol (Dresscher, 2023). The parameter within the position-computed-force control and position-measured-force architecture are found using trial and error. From the theory discussed in 2.1.3 and 2.1.4, the control gain should be as large as possible. Nevertheless, the system's behaviour will be unstable with too large control gains. Trial and error are applied to find the line between the most accurate controller and a stable system. Trial-and-error is also used to find the parameter within the position-position architecture. As feedback control is used within the architecture, finding the parameters using system characterisation is impossible as the environment is hard to model. As an alternative, trial-and-error is used.

Chapter 4 - Ideation

In this chapter, ideation is done on the user interface of the DIY kit. The ideation is done on the five main elements of the interface; style, structure, user input, feedback and block diagrams. Final decisions regarding the user interface are made in Chapter 5 - Specification.

4.1 Elements of the Interface

4.1.1 Style

In order to make a coherent interface, an overall style is needed. As described in requirement 2a, the interface, and therefore also the interface style, should be engaging and appealing. Two options are taken into consideration for the style used in the interface. The first option is to use the style guide of avatars.report. This style is also used on their website and can be found in Figure 9 (Avatar Robotics, n.d.). This option aims to make the DIY kit coherent with the other educational aspects. The second option is to design a new style guide for this specific kit. Ideation is done on different colour schemes that can suit the DIY kit. A visualisation of the different colour schemes can be found in Figure 10.

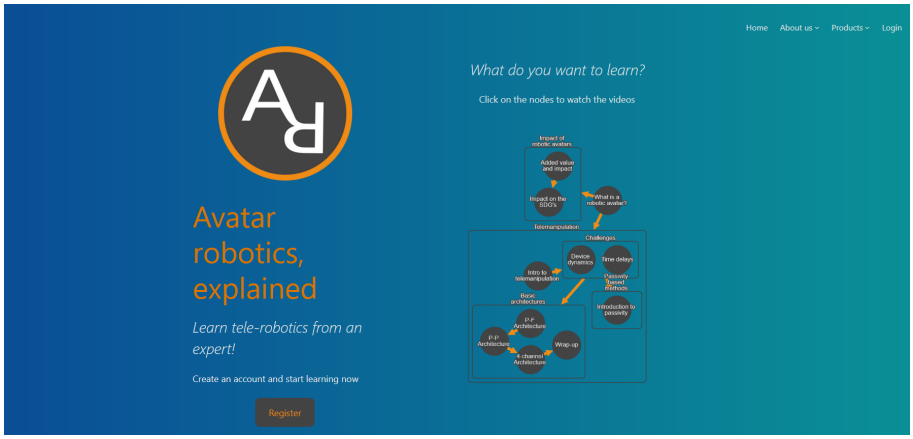

Figure 9: *The website style of avatars.report*

Figure 10: *Visualisation of different colour schemes for user interface*

4.1.2 Structure of the Interface

The second part of the ideation on the interface focuses on its structure. Two ideas for the structure are generated. A visualisation of both ideas can be found in Figure 11. Both designs begin with a start screen for the user. The idea behind the start screen is to give the user a clean, not intimidating place to begin. This provides the user with time to acclimate and lets them continue when they want. When pressing start, the first design will guide them to the general interface page. Here the user can both select the architecture they would like to test and select the according parameters. The main difference with the second design is that the architecture and parameter selection does not happen on the same page. Instead, the second design lets the user first select the architecture, after which they see a new screen. In this screen, the user is able to set the parameters.

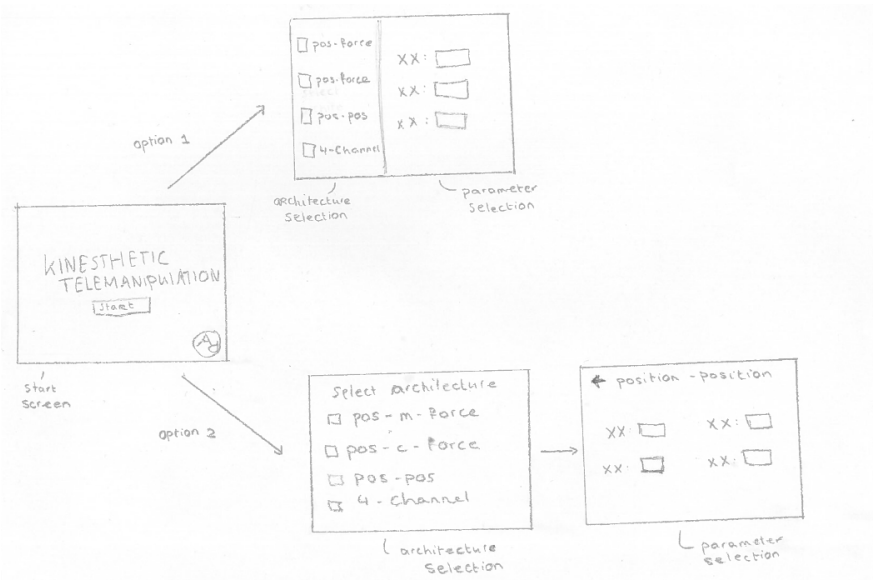

Figure 11: *Two options for the structure of the user interface*

4.1.3 User Input

As described in requirement 1b, the user should be able to adjust the controller gains. Ideation is done on how the user can interact with these control gains. The first option is to use a text box where the user can type in the desired control gain. This allows for exact input. Nevertheless, it can be harder to program in some programming languages. The second option is to use plus and minus buttons to adjust the gain. Although the input can only be changed in incremental steps, it does give appealing buttons to engage the user. The last option is to use a slider. This option makes it harder for the user to select a precise input value. Nevertheless, the slider does give engagement to the user. Visualisations of the three ideas can be found in Figure 12.

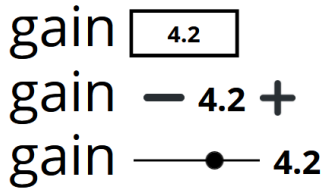

Figure 12: *Visualisation of the three options for the user input*

4.1.4 Feedback to the User

As described in requirement 2e, the user should receive feedback on the selected parameter configuration. Three options are taken into consideration for giving this feedback. The first option is to provide feedback using a (coloured) slider. The position on the slider indicates to the user how optimal the parameter configuration is regarding perfect transparency. For clarity, the slider has coloured parts. The second option is to use a percentage. The higher the rate, the more optimal the parameter

configuration. The last option is to use a circular meter with coloured parts for clarity. A visualisation can be found in Figure 13.

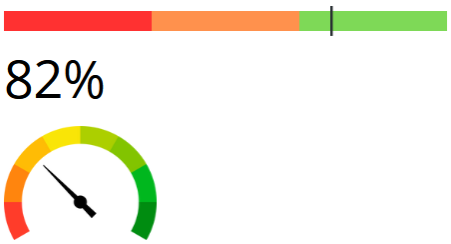

Figure 13: *Visualisations of the three options for user feedback*

Next to the slider itself, ideation is done on the inclusion of multiple sliders. A visualisation can be found in Figure 14. With two sliders, the user is able to get feedback on the physical and theoretical behaviour of the parameter configuration. This feature is especially interesting with the position-computed force and the position-measured force architecture as these controllers have different practical behaviour for higher control gains as discussed 2.1.3 and 2.1.4.

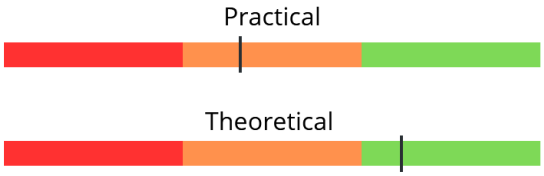

Figure 14: *Visualisation of two feedback bars focussing on physical and theoretical behaviour*

4.1.5 Block Diagrams

Last, ideation is done on whether to include block diagrams of each architecture in the interface. A visualisation of a block diagram can be found in Figure 15. By having the diagrams, the goal is to give the user a small recap on the working of all the architectures. Next to that, it makes it each to refer to certain gains. However, the block diagram might also be large and intimidating for the user.

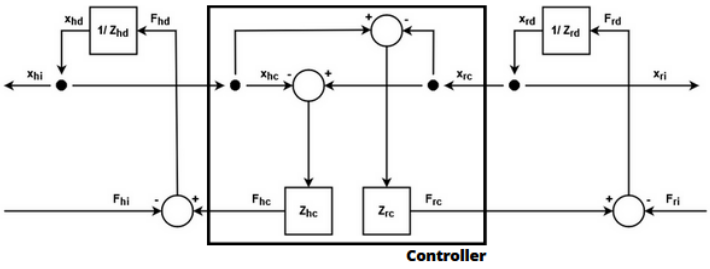

Figure 15: *Visualisation of a bond diagram in the user interface*

Chapter 5 - Specification

In this chapter, final decisions are made regarding the kit's design. The findings are split into two sections. First, the technical specifications are discussed. Second, the interface specifications are discussed.

5.1 Technical Decisions

5.1.1 Microcontroller

The microcontroller controls and receives the inputs and outputs from the sensors and actuators in the kit. As discussed in 3.2.2, two microcontrollers are being considered for this project; the Arduino Uno and the ESP32. In this section, a final decision is made regarding the microcontroller.

The ESP32 and the Arduino Uno are similar microcontrollers, yet both have advantages and disadvantages. The ESP32 has a built-in Wi-Fi and Bluetooth chip, whereas the Arduino needs a shield to access Bluetooth and Wi-Fi. Moreover, the ESP32 is much cheaper than the Arduino Uno; the microcontrollers cost €9.50 and €24.00, respectively. The last advantage of the ESP32 is double the amount of memory compared to the Arduino. The Arduino has, in turn, the benefit of an easy-to-use and user-friendly IDE with many standard and downloadable libraries. Although the ESP32 can be programmed using the Arduino IDE, it needs an add-on. Moreover, Arduino is much more frequently used in DIY projects and teaching materials, giving Arduino the 'recognisable' advantage. Lastly, interfacing with the Arduino is easier than on the ESP32.

	Arduino Uno	ESP32
Low Price		
Use in DIY-kits		
Own IDE		
Compatible with Processing		

Table 4: *Decision table regarding the choice for the microcontroller*

A decision table is made to decide on the microcontroller and can be found in Table 4. In the table, green indicates that the aspect is included, orange indicates that the aspect is partially included and red indicates the aspect is not or limitary present. The choice is made to go for the Arduino Uno as the microcontroller.

As formulated in requirement three, the DIY kit should be intuitive. As the Arduino Uno has its own popular IDE and is widely used in all kinds of (DIY) projects, the assumption is made that the Arduino is indeed more intuitive to use than the ESP32. Thus despite being more pricey than the ESP32, the Arduino wins on the usability aspect. Last, the DIY kit does not need the Bluetooth and Wi-Fi elements the ESP32 offers. The choice is made to go for the Arduino Uno.

5.1.2 Programming Language

As discussed in 3.2.3., two programming languages are being considered to develop the user interface; Python and Processing. In this section, a final decision is made regarding the microcontroller.

Python has the advantage of having an easy-to-use and readable syntax. Next, it is a known language and is regularly taught in education. Where Processing is less known, it is used as the primary language within the bachelor Creative Technology. Next to that, it is targeted at visual applications and prototyping.

	Python	Processing
Frequently used within Creative Technology		
Easy executable		
Easy to use IDE		
Interfacing with Arduino		

Table 5: *Decision table regarding the choice for the programming language*

For the interface, the choice is made to use Processing. Again, a decision table can be found in Table 5. Processing is used a lot in Creative Technology, and therefore the most experience in development is presented within this language. Using a familiar language will contribute to requirement six, the software of the DIY kit should be done in time. Moreover, Processing makes it easy for the user to run an executable program, and the IDE is easy to use. This aligns with requirement three, the DIY kit should be intuitive. Last, Processing allows for easier interfacing with Arduino contributing to requirement seven, the hardware and software in the kit must be compatible.

5.2 User Interface

5.2.1 Rapid Prototyping

After the Ideation done on the user interface in Chapter 4, rapid prototyping is performed to create different configurations of user interfaces. The configurations can be found in Figure 16.

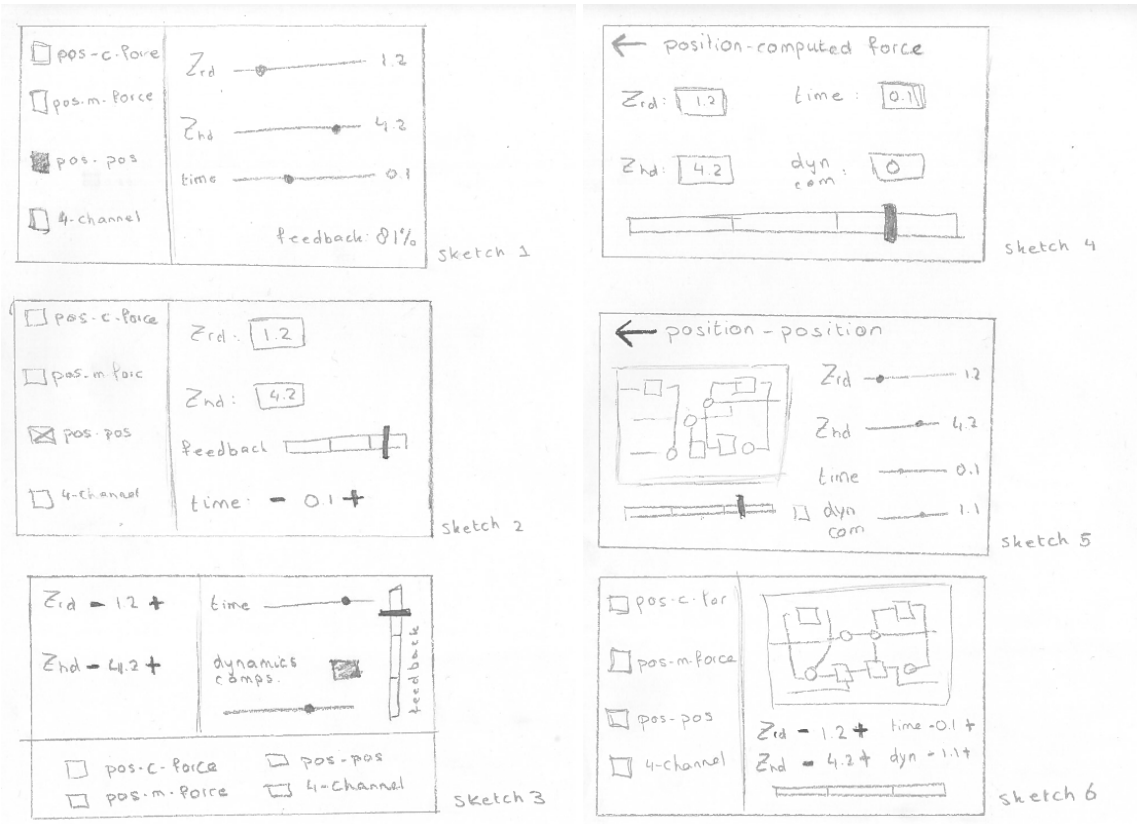

Figure 16: *Six configurations generated during rapid prototyping*

The configurations are analysed and discussed with the client, leading to the following finding. Regarding the feedback to the user, a meter (either linear or circular) is preferable. As the feedback is more of an indication than realistic advice, a number will be too precise. Regarding user interaction, a slider is the preferred option. First off, the slider is engaging and appealing to the user. This is in line with requirement 2a, the user interface should be engaging and appealing. Second, the slider has a specified minimum and maximum concerning input selection. Block diagrams in the interface are handy as they remind the user of which gains are used where in the architecture. This decision also contributes to requirement three, the DIY kit should be intuitive. The use of block diagrams also gives some direction regarding the structure of the interface. As some block diagrams can be relatively big, more room on the screen is needed. This room will be presented by only including the parameter selection on the screen. However, before making a final decision regarding structure, both options are worked-out into more precise prototypes.

5.2.2 Worked-out Prototypes

To make a final decision regarding structure, two worked-out prototypes are made. These prototypes can be found in Figure 17. In order to choose with regard to the interface's structure, research is done on the user's working memory. According to Benyon (2020), the working memory of a human is approximately 3 or 4 items. Comparing this research with the two prototypes, the choice is made to split the architecture and parameter selection, choosing the bottom design. This way, the user has to perceive less input and, therefore, have a better user experience. Although the bottom prototype still has over 4 interaction items, the choice is made to change the user interface the same as the other architectures have fewer control gains and therefore comply with the rule.

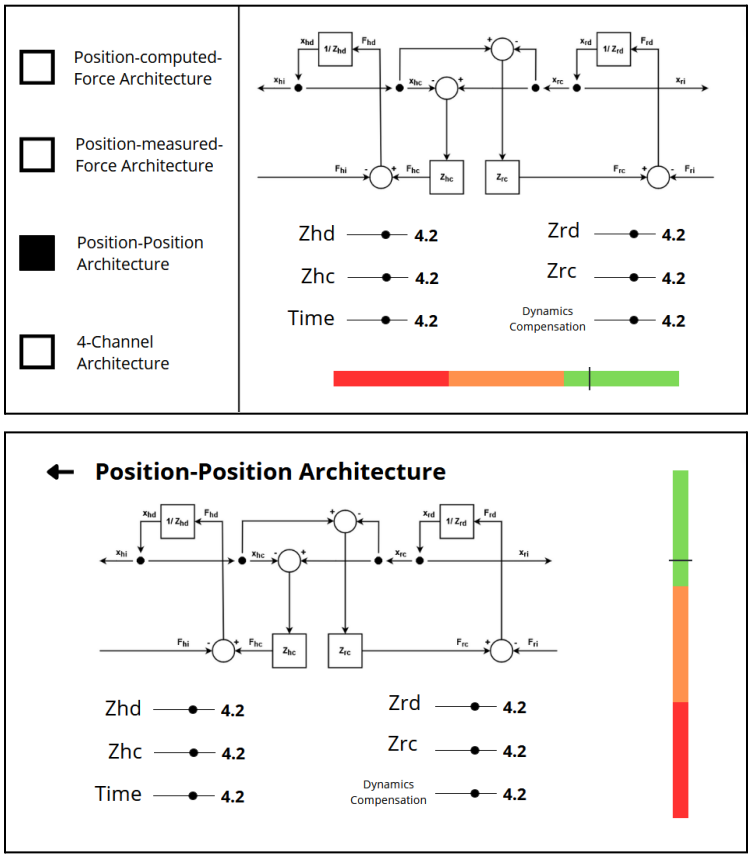

Figure 17: *Two more worked-out prototypes during the second prototype iteration*

5.2.3 Style

Last, a decision is made regarding the style used in the user interface. To make this decision, research is done on how humans react to specific colours. The study focuses on the five popular colours; red, blue, green, orange and yellow. Each colour's symbolisation, effect, and positive and negative associations were researched (Olesen, n.d.). These findings are summarised in Table 6.

Colour	Symbolizes	Effects	Positive Associations	Negative Associations
Red	Action Strength Energy Passion	Attention Motivates Stimulates Cautions	Sexuality Courage Desire Confidence	Anger Danger Revenge Aggression
Blue	Security Trust Loyalty Responsible	Protects Calms Relaxes Supports	Confidence Peace Honesty Reliability	Conservative Passive Depressed Predictable
Green	Harmony Safety Growth Health	Revitales Balances Relaxes Encourages	Generosity Hope Prosperity Luck	Judgemental Envy Materialism Inexperience
Orange	Youth Emotion Optimism Enthusiasm	Encourages Uplifts Stimulates Communicate	Spontaneity Creativity Warmth Positivity	Exhibitionism Superficial Impatient Domination
Yellow	Happiness Optimism Positivity Intellect	Clarifies Inspires Amuses Energises	Creativity Perception Mentality Warmth	Cowardice Deception Egotism Caution

Table 6: *Five popular colours and their symbolisation, effect, positive associations and negative associations*

For the user interface in the kit, the goal is to make an engaging and appealing interface as described in requirement 2a. Nevertheless, more can be said about the reactions the kit should deliver. The kit focuses on learning and should therefore inspire the user. The choice is made to use the colour yellow for the kit. Yellow is associated with intellect and positivity. Next to that, it has the effect of being inspiring and clarifying precisely the goal of an educational kit.

Chapter 6 - Realisation

6.1 Realisation user interface

6.1.1 Screens User Interface

Within the user interface, different screens are presented to the user. An overview of these screens can be found in Figure 18. The user interface starts with the screen at the top left, the home screen. As described in the Specification, the screen is styled in yellow to evoke intellect and optimism in the user. When pressing enter, the user will be guided to the next screen. In this interaction screen, the user can select a control architecture using their keyboard. Again, the same style is used as in the home screen to evoke the same emotions and make the whole interface coherent. The user can always return to the last screen by pressing the backspace on their keyboard. The screens at the left and right bottom are the parameter selection interface. When selecting a control architecture in the last screen, the user will be guided to the corresponding parameter selection. In this screen, three different aspects are presented. First, there is the bond diagram. The bond diagram makes it easier for the user to recall the specifics of the architecture. In the block diagram, the controller is outlined for extra clarity. Second, there is the input with which the user can interact. Two interactions are possible; adjust the control gain(s) and change the time delay. Third, there is the feedback bar. The feedback bar gives the user an indication of how optimal the chosen parameter configuration is. The kit has three main learning goals. First, the user is able to see the effect of different control gains on the system. Second, the user is able to see the effect of time delay on the system. Third, the user is able to compare different control architectures.

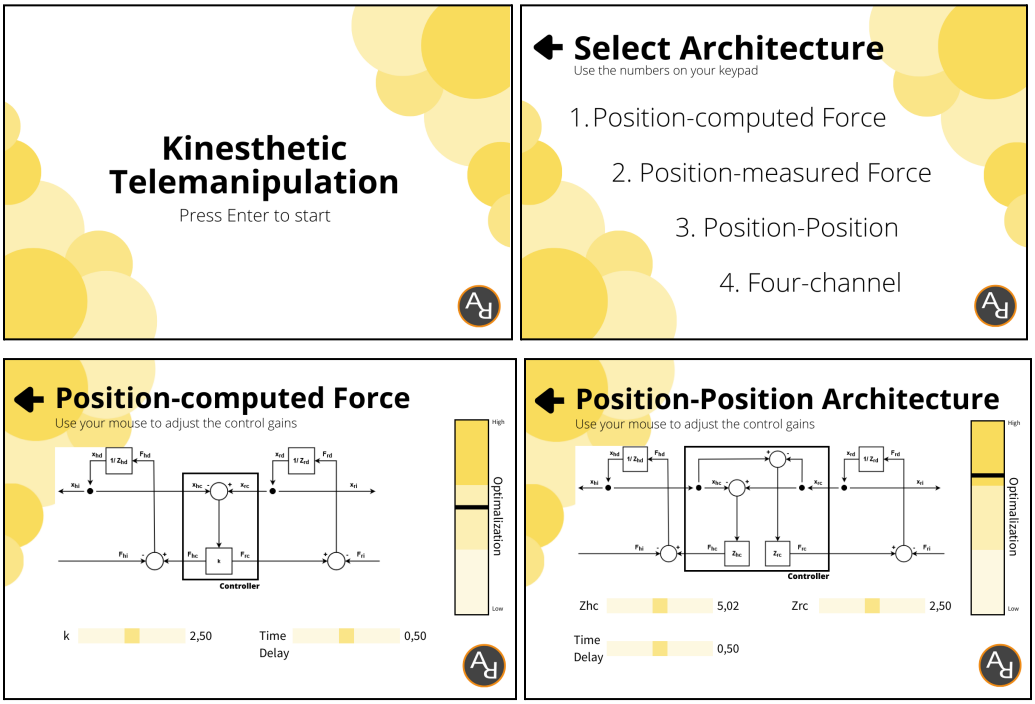

Figure 18: *An overview of all screens presented in the user interface.*

6.1.2 Slider

The sliders give the user control of the control gains and time delay present in the system. The basic functionality of the slider is constructed using the scrollbar example of Processing itself (Scrollbar / examples, n.d.) Additional features were added to give the slider its full functionality. The first feature of the slider is moving to the user's mouse position. The speed of this movement can be controlled by the speed attributed within the Slider class. Second, the slider changes colour when the user hovers over the slider. A visualisation of this feature can be found in Figure 19. Third, the slider can calculate its optimisation factor. This feature is discussed in more detail in 6.1.3. Last, the slider sends the value to the Arduino. The slider is styled in the same way as the other part of the user interface to make it coherent.

Figure 19: *Top slider is a visualisation of the slider without the mouse hovering over it, and the bottom slider is a visualisation of the slider with the mouse hovering over it.*

6.1.3 Feedback Bar

The feedback bar gives the user feedback on how optimal the chosen parameter configuration is. A visualisation of the feedback bar can be found in Figure 20. The bar is styled to match the style of the colour palette of the whole user interface. This choice is in contrast to the selected feedback bar in the specification. However, the specified red-orange-green feedback bar did not blend well with the other parts of the user interface. To still indicate the feedback, the feedback still contains three different coloured subsections. Additionally, the labels “high” and “low” were added.

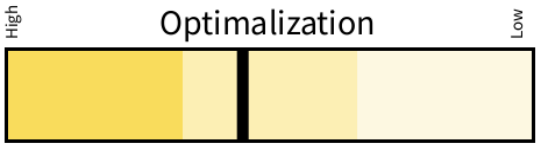

Figure 20*: Visualisation of the feedback bar in the user interface.*

To give feedback regarding the optimisation, a calculation has to be made. First, the optimisation factor of each control gain slider is calculated. The code for this calculation can be found in Figure 21.


```
float getOptimalization () {
      float maxError = max(optimalValue-valueMin, abs(optimalValue-valueMax));
      float optimalization = abs(optimalValue-valueSlider) / maxError;
      return optimalization;
 }
```

Figure 21: *Code to calculate the optimisation of a parameter configuration*

To calculate the optimisation factor, first, the maximal error is calculated. To do this, the minimal and maximum values of the slider are compared to the optimal value. Afterwards, the slider's value is compared to the optimal value and divided by this maximum error. This calculation gives the optimisation value. The optimisation factor will be between 0 (optimal) and 1 (least optimal). To calculate the total optimisation, all optimisation factors are summed and divided by the amount of factors. This will lead to a total optimisation between 0 (optimal) and 1 (least optimal). This value is plotted on the feedback bar.

6.1.4 Processing Communication to Arduino

Processing communicates with the microcontroller via serial communication. This communication includes sending the control gain(s), sending the time delay and selecting the right architecture. To keep communication minimal, messages are only sent when values are changed.

Figure 22: *General overview of the message's different fields and sizes*

Each message sent through serial communication can be divided into three fields; the header, the data field and the end field. A general overview of a message can be found in Figure 22. The header contains information about what kind of message is being sent. An overview of all headers being used and their corresponding message can be found in Table 7. The data field contains the data that is being sent to the microcontroller. Lastly, the end field indicates to the microcontroller that the message is done. For all messages, this end field contains a semicolon. The messages are filtered on semicolons and not end-line characters, as Windows, Unix, and Mac all consider different end characters (Malviya, n.d).

Header	Corresponding type of message
a	Architecture selection
b	Time delay
c	k
d	Z_{hc}
e	Z_{rc}

Table 7: *Overview of all headers used within the serial communication and the corresponding type of message*

The message is sent in a String format towards the Arduino. Before the message can be sent, the value sent in the data field is rounded to two decimals before the dot and three decimals after the comma. Next to that, every comma in the data field is replaced with a dot. As Processing uses the decimal separator according to the local language setting, the program will only work for some users.

6.1.5 Class diagram

A class diagram of the Processing program is constructed to show the working of the interface. The diagram describes all the classes and their most important functions and attributes. The diagram can be found in Figure 23.

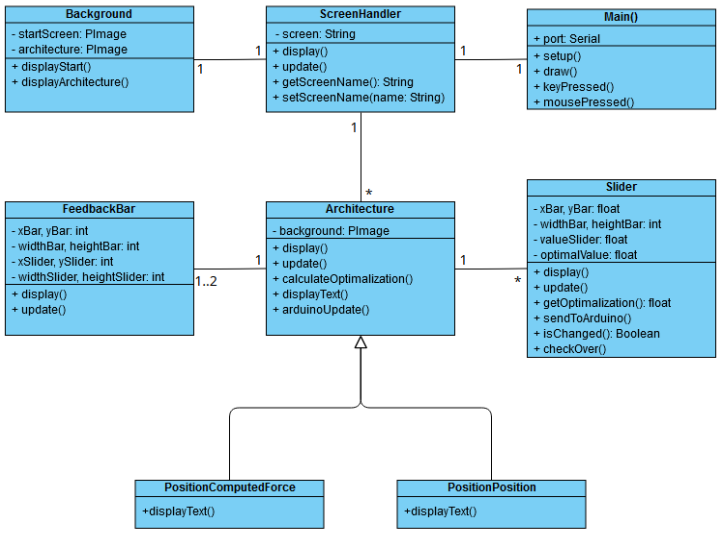

Figure 23: *The class diagram of the Processing program of the user interface*

The main() is the main class of Processing. Within the main class, the user input is being handled, and the program is being run. Next to that, the port for serial communication is initialised. The ScreenHandler class shows the right screen at the right time. In order to do so, the current screen is kept track of with the screen variable. The main can update the screen variable when user input is presented.

The ScreenHandler has two options for screens to show, either screens in the class Background or screens in the class Architecture. In the Background class, screens with no interactive element are presented. In the DIY kit, these are the start screen and the screen for selecting the architecture. As no interaction is needed, these screens are simple PImages made in other design programs.

The screens for the architectures are displayed using the Architecture class. The Architecture class is the superclass for all the architectures aiming for easy extensibility of the program. For a new architecture, only the placement of the slider(s), the slider labels and the feedback bar have to be defined, and the new architecture can be used.

The Architecture class uses two other classes; the Slider class and the FeedbackBar class. As the name suggests, the Slider is responsible for all functions related to the slider. Next to drawing the slider, the class is responsible for calculating the optimisation value, as discussed in 6.1.3, and sending the slider's value towards the Arduino, as discussed in 6.1.4. The FeedbackBar class has two main functions, updating and displaying the feedback bar. The update is based on the optimisation factor, as discussed in 6.1.3.

6.2 Technical Realisation

6.2.1 Setup

As the hardware for the DIY kit is not finished in time, another setup is built to test the control architectures. This setup is comparable to the hardware of the DIY kit, but there is one crucial difference; the setup does not include a force sensor. As a result, the software focuses on the position-computed force architecture and the position-position architecture, as the two other architectures can not be realised.

The setup consists of two separate motor setups, two power sources and one Arduino Mega. The motor setups consist, in their turn, of an interface PCB, an encoder, a motor, a transmission, and a flywheel. Table 8 gives more detailed information on all the components present. In order to control the two motor setups, two Arduino shields are used. These shields are specifically designed for these motor setups and are put on a breadboard to connect them to the Arduino Mega.

Component	**Specifications**
Interface PCB	Current controller, schematic can be found in Appendix A

Encoder	US Digital E6-400-197-IE-S-D-D-B
Motor	Bühler Motor 1.13.049.401
Transmission	Toothed belt drive connecting gear at the motor side (14 teeth) with gear at flywheel side (72 teeth)
Flywheel	Disk of 200g with a diameter of 11cm
Arduino shield	Custom shield with two opamps, schematic can be found in Appendix B
Arduino	Arduino Mega 2560
Power sources	Delta Elektronica D 030-1 (12V, 1A) Delta Elektronica D 015-1.5 (12V, 1A)

Table 8: *Specification of all components presented in the setup*

In Figure 24, an overview of the wiring in the setup is given. As for the shields, the pins that normally do not have a reference name, the choice is made to reference them as the pin they are supposed to connect to on the Arduino.

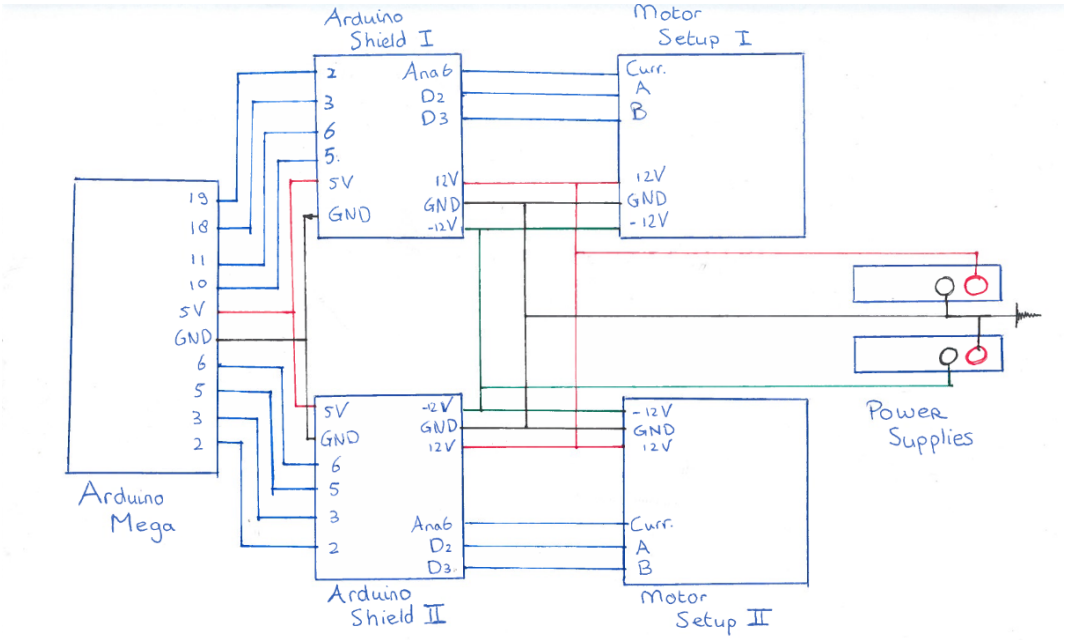

Figure 24: *Fritzing model of the setup to test the control architectures*

6.2.2 Microcontroller Steps

The microcontroller is responsible for the timing of the system, the reading of the encoders, the calculation of the control architectures, the output to the motor and the data handling from Processing.

The steps for this process can be found in Figure 25. All of these steps are discussed in more detail in this chapter. The full code of the Arduino can be found in Appendix C.

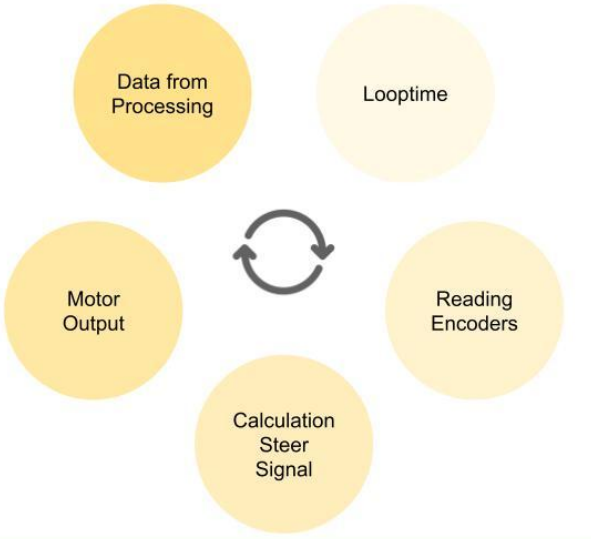

Figure 25: *Different steps taken by the microcontroller*

6.2.3 Looptime

To let the controller work accurately, the loop time needs to be monitored. The code responsible for the timing of the controller checks two things. First, the control loop waits until the loop time has passed before running the control code. This is done by comparing the current time with the control loop's last run. In the code, this is described as the trigger time. Second, the control loop checks if the desired loop frequency can be reached. If the program takes longer than the expected loop time, the microcontroller will register this and notice the user blinking the built-in led on the Arduino. With this feature, the maximum loop time of the program is evaluated in 7.2. Currently, the program runs on 500Hz.

6.2.4 Reading Encoders

The encoders on the motor setup are read using two interrupt pins on the Arduino board. In the Specification, the choice was made for an Arduino Uno board. Nevertheless, as the Uno has only two interrupt pins and two motor setups have to be connected, another way of reading the encoders has to be found. The choice was made to use an Arduino Mega as a substitute. The Mega has four interrupt pins, which allows the controllers to be tested quickly. The change of microcontroller does not influence the kit itself, as separate hardware and code are made.

To easily read the encoder, the Encoder.h library is used (Encoder, n.d). This library makes it possible to read the position on the encoders without specifying all the pin interrupts. In order to use the encoder positions in the control architecture, the values are converted to a position in radians. This conversion is done using the pulses per revolution of the encoder and the revolution of the encoder.

6.2.5 Calculation Steer Signal

The calculation for the steer signal depends on the control architecture. To show the similarity between the Arduino code and the block diagram of the control architecture, the two are visualised next to each other for each control architecture.

6.2.5.1 Position-Computed Force Architecture

The comparison between the block diagram and code for the Arduino can be found in Figure 26. In step one, the encoders are read and converted to a position in radians, as discussed in 6.2.4. In step two, the values are compared, and the error value is calculated. In the next step, a small amount of damping is added to the system. This step is not included in the diagram but is needed to increase stability within the system. Otherwise, the system will already be unstable for minimal control gains. As a result, the user cannot see the difference between different control gains. In the last step, the steer value is calculated using control gain k. This signal is converted and sent to the motor setup, more on this in 6.2.6.

```
//Step 1:
encoderPos1 = encoder1.read();
phi1 = encoderPos1 / (4*63.66);
encoderPos2 = -encoder2.read();
phi2 = encoderPos2 / (4*63.66);

//Step 2:
 error1 = phi2 - phi1;

//Step 3
rate1 = -2 * pi * fh * dedt1;
total1 = total1 + rate1 * (1 / loopfrequency);
dedt1 = 2 * pi * fh * error1 + total1;

//Step 4
tau1 = error1 * Kp + dedt1 * Kd;
tau2 = -tau1;
```

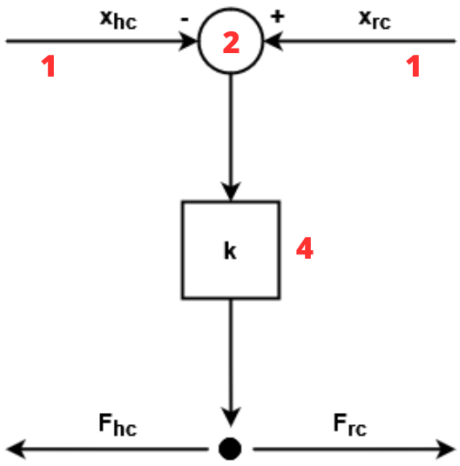

Figure 26: *Calculation of the steer signal for the position-computed force architecture*

6.2.5.2 Position-Position Architecture

The computation of the steer signal for the position-position architecture is similar to the computation of the steer signal in the position-computed force architecture. A visualisation of this calculation can be found in Figure 27. In step one, the encoders are read and converted to a position in radians, as discussed in 6.2.4. The error is calculated in step two, but different from the position-computed force architecture, two error signals are presented. For both these signals, the same amount of damping for the same reason is added as with the previous architecture. In the last step, the steer is calculated by multiplication with the control impedance. Currently, the impedance is one single gain in the user interface. However, it would be interesting to consider adding different gains to this impedance for future iterations.

```
//Step 1:
encoderPos1 = encoder1.read();
phi1 = encoderPos1 / (4*63.66);
encoderPos2 = -encoder2.read();
phi2 = encoderPos2 / (4*63.66);

//Step 2:
 error1 = phi2 - phi1;
error2 = phi1 - phi2;

//Step 3
rate1 = -2 * pi * fh * dedt1;
total1 = total1 + rate1 * (1 / loopfrequency);
dedt1 = 2 * pi * fh * error1 + total1;
dedt2 = 2 * pi * fh * error2 + total1;

//Step 4
tau1 = error1 * Kp + dedt1 * Kd;
tau2 = error2 * Zrc + dedt2 * Kd;
```

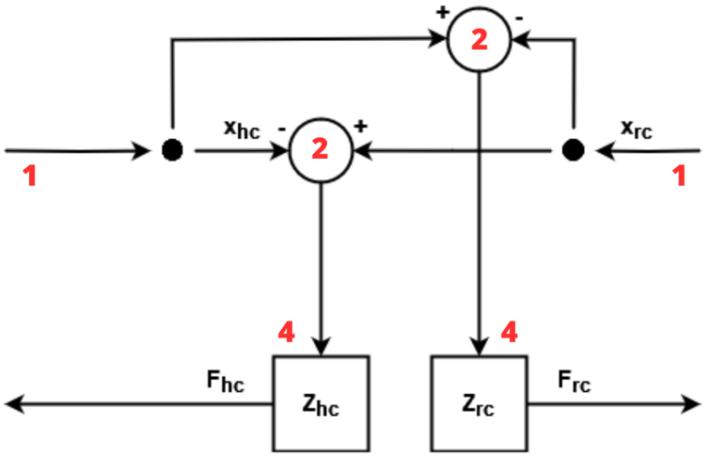

Figure 27: *Calculation of the steer signal for the position position architecture*

6.2.6 Motor Output

After calculating the steer signal value, a PWM signal is sent to the motor to let the setup move. To calculate the PWM signal, several conversions have to be done. First, the steer value is divided by the motor constant to obtain the desired current. Second, the current is limited between -1A and 1A to protect the motor setup. Third, there is compensation for the gain of 2 and the offset of -5V in the Arduino shields. Last, the output is mapped from 0V-5V to a PWM value between 0-255. This final signal is sent to the motor setup.

6.2.7 Data from Processing

Data is sent from Processing to change the control gain(s), time delay and control architecture in the setup. As discussed in 6.1.4, the messages sent from Processing contain three fields: a header, a data field and an end field. The microcontroller checks the serial communication to obtain these messages. This serial communication check is separate from the control of the architecture. The controller will keep storing messages in a buffer until it comes across a semicolon, an end field. If this is the case, the message will be dissected. First, the buffer is trimmed using the Arduino trim() function to get rid of, among others, newlines and carriage returns. Second, the header is checked, and the semicolon is removed. Using this header, the right parameter is selected. Last, the data field is transformed into a number and the according parameter is changed. After these steps, the buffer is cleared, and the process can start again. The choice is made to let the buffer be of the type String to have access to standardised functions. The code for receiving messages can be found in Figure 28.

```
void processingInput() {
  if (Serial.available() > 0) {
    char input = (char)Serial.read();
    if (input != ';') {
      buffer += input;
    } else {
      buffer.trim();
      char architecture = buffer.charAt(0);
      switch (architecture) {
        case 'a':  // select architecture
          selectArchitecture = buffer.substring(1).toDouble();
          break;
        case 'b':  //time delay
           loopfrequency = buffer.substring(1).toDouble();
          break;
        case 'c':  //kp
          Kp = buffer.substring(1).toDouble();
          break;
        case 'd':  //Zhc
          Zhc = buffer.substring(1).toDouble();
          break;
        case 'e':  //Zrc
          Zrc = buffer.substring(1).toDouble();
          break;
        default:
          break;
      }
      buffer = "";
    }
  }
}
```

Figure 28: *Code for reading the serial communication sent by Processing on the Arduino*

6.2.8 Control Gain Limits

The limits of the sliders for the control gains need to be determined to complete the user interface. As discussed in 3.2.3, the gains are determined using trial and error. These results can be found in Table 9. The minimum gain is for all of the controller gains equal to zero. The optimal gain is the point just before the controller gets unstable. The maximum gain is the point at which the instability of the controller remains equal overall.

Control Gain	Minimum	Optimal	Maximum
k	0.000	0.033	0.080
Z_{rc}	0.000	0.079	0.300
Z_{hc}	0.000	0.079	0.300

Table 9: *Overview of all the control gains in the kit and their minimum, maximum and optimal*

Chapter 7 - Evaluation

7.1 Requirements Evaluation

To evaluate the DIY kit, the requirements formulated in 3.1.6 are reflected. The outcomes of this evaluation can be seen in Table 10. Green indicates that the requirement is reached, orange indicates that the requirement is partially reached, and red indicates that the requirement is not reached.

Number	Requirement	Reached
1a	The kit must cover the four architectures; position-computed-force control, position-measured- force control, position-position control and the 4-channel architecture.	
1b	The user must be able to adjust the controller gains within the kit.	
1c	The kit should cover dynamics compensation.	
1d	The kit should cover time delay, with the option for the user to adjust the time delay.	
2a	The user interface should be enhancing and appealing.	
2b	The kit's documentation should include the kit's expectations and goals.	
2c	The documentation of the kit should include optimal and not optimal configurations of the controller	
2d	The kit should be an interactive, real-world set-up	
2e	The user interface should give feedback to the user regarding the selected parameter configuration.	
3	The DIY kit should be intuitive.	
4	The DIY should be affordable.	
5	The hardware of the DIY kit must be done in time.	
6	The software of the DIY kit must be done in time.	
7a	The hardware and software must be able to handle a force output.	
7b	The hardware and software must be able to handle a position and force input.	

8	The kit’s development must adhere to the guidelines of a Creative Technology Graduation Project set by the University of Twente.	

Table 10: *Evaluation of the user requirements*

The first requirement focuses on the content of the DIY kit. The original goal was to include four architectures and their control gain(s), time delay and dynamics compensation in the kit. Nevertheless, it was not feasible to reach all these goals. Due to a delay in working hardware, a set-up without a force sensor had to be used. As a result, it was impossible to realise the position-measured force and 4-channel architecture. Next to that, dynamics compensation is eventually not part of the DIY kit due to time limitations. Despite these setbacks, the two other architectures (the position-computed force and the position-position architecture) were implemented correctly. Moreover, the user can adjust the control gain(s) and the time delay.

The second requirement focuses on the educational part of the DIY kit. One thing that is immediately clear is that the kit does not contain (external) documentation. Instead, the choice was made to focus on the physical components of the kit. However, this does not mean that none of the requirements regarding the documentation are met. Requirement 2c regarding showing optimal and not optimal configurations is partially met due to the feedback bar in the interface. As the bar gives feedback on the chosen parameter configuration, the user is still shown optimal and not optimal configurations.

Regarding requirement 2a, the user interface should be enhancing and appealing, more research needs to be done in order to validate this requirement. At the moment, the kit is made with theoretical knowledge only. In order to obtain real-life validation, user testing is essential. The same holds for requirement 3, the DIY kit should be intuitive. Nevertheless, the product user tests are not within this research's scope.

Requirements 5, 6 and 7 are all related to the system's hardware and software. Although the hardware was not done in time, it was possible to use separate hardware. As stated before, this hardware was only able to handle position input. This explains the orange indication for requirements 5 and 7b.

7.2 Loop Frequency Evaluation

In the Specification, the choice was made to use an Arduino Uno as the microcontroller in the project. This choice was slightly altered to using an Arduino Mega, as explained in 6.2.4. Nevertheless, one point of concern when using the Arduino was its computational speed. In order to evaluate the choice, a test regarding the loop frequency is done.

In the microcontroller, a feature is added to let the built-in LED blink whenever the loop frequency can not be reached. In this evaluation, the microcontroller is tested on multiple different loop frequencies while the Processing program sends serial communication and the set-up is being interacted with. This

way, the most computations have to be done, with, as result, the most accurate reflection on the loop frequency.

After several tests with both the position-computed force and the position-position architecture, it is found that the loop frequency can reach a maximum of 1000Hz. With higher frequencies, the program cannot keep up with the computations, and the LED will start blinking.

7.3 Architecture Evaluation

As stated in 2.1.2, the goal of the control architectures is to reach perfect transparency. This is achieved when the position and the force on the human side of the controller are equal to the position and force on the robot side of the controller, respectively. Therefore, the position and forces should be evaluated to evaluate the architecture's working. Nevertheless, only evaluating the positions in the current setup is possible.

To evaluate the positions of both control architectures, a test was conducted. The test consists of first turning the wheel on the human side, one full turn clockwise and one full turn counterclockwise. Afterwards, this same procedure was repeated on the robot side. This is done for both architectures and with different control gains. Meanwhile, the microcontroller is connected with Serial communication to Excel and using Data Streamer, the position of both the human and robot controller is saved.

7.3.1 Position-Computed Force

First, a look is taken at the position-computed force architecture. The architecture is tested at seven different control gains; k=0.00, k= 0.02, k=0.05, k=0.07, k=0.10, k=0.12 and k=0.14. In Figure 29, the system's response can be seen for turning the wheel on the human side and the robot side with k=0.00. The response of the system is minimal, as no movement of the slave can be seen when executing the test.

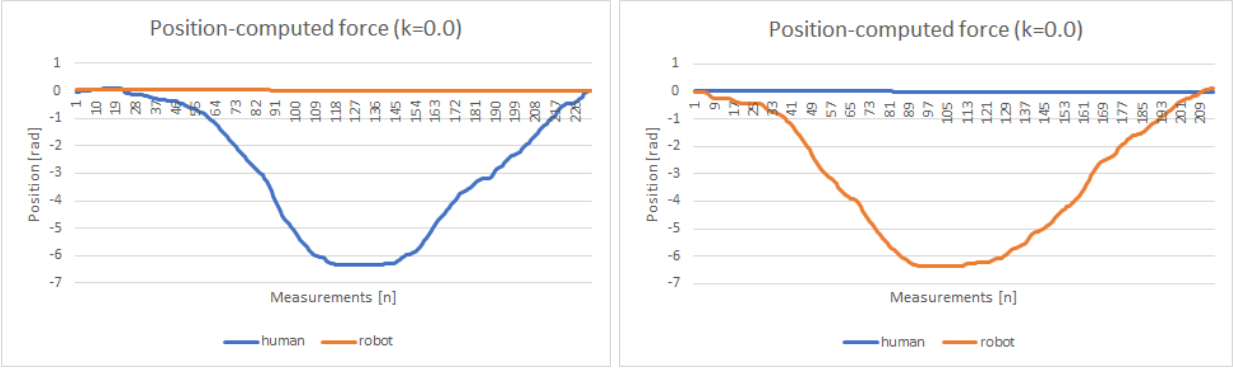

Figure 29: *The response of the system at k=0.00 in the position-computed force architecture*

Even when a slight amount of gain, the system's response increases, and output can be detected on both sides of the system. In Figure 30, the response of the system with k=0.02 is visualised.

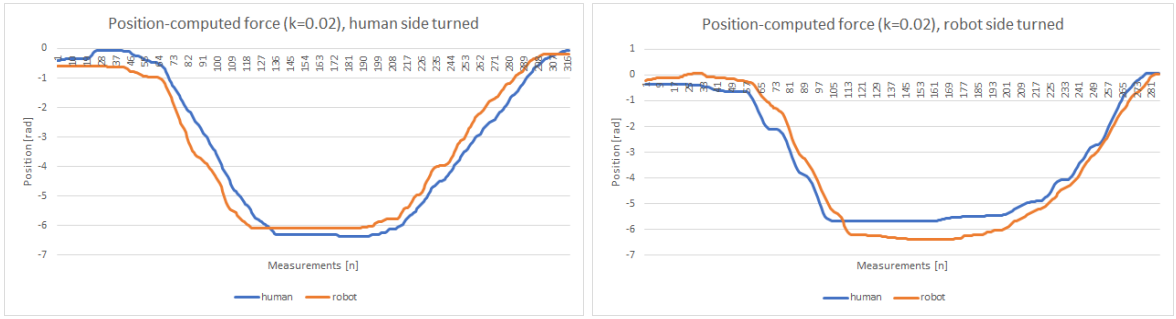

Figure 30: *The response of the system at k=0.02 in the position-computed force architecture*

This response becomes even more accurate when the control gain increases further. In Figure 31, the response of the system with k=0.10. In the right graph, one can see an inconsistency in the response of the human side to the robot side. These inconsistencies are detected more frequently at the higher gain. An elaboration on the phenomenon is given in Chapter 9. Additionally, one can see that the system is already becoming unstable.

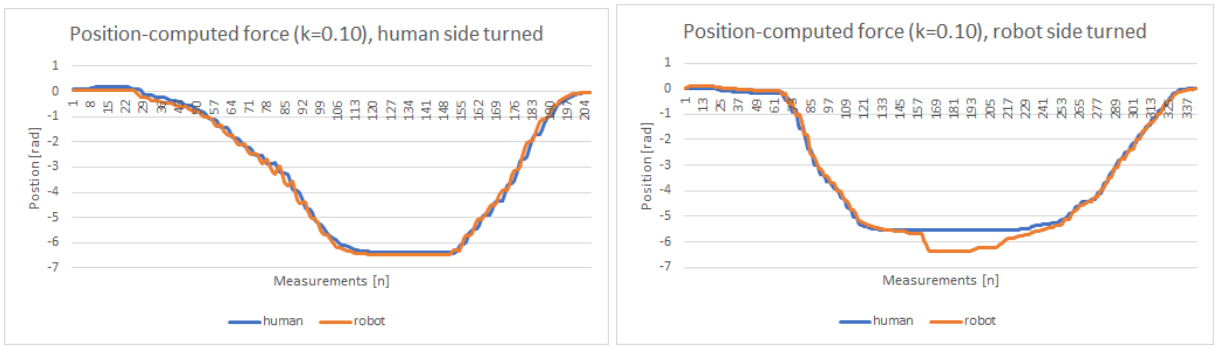

Figure 31: *The response of the system at k=0.10 in the position-computed force architecture*

Increasing the control gain even more, and the system becomes unstable, as can be seen in Figure 32. In the graphs, the response of the system can be seen for a control gain of k=0.14.

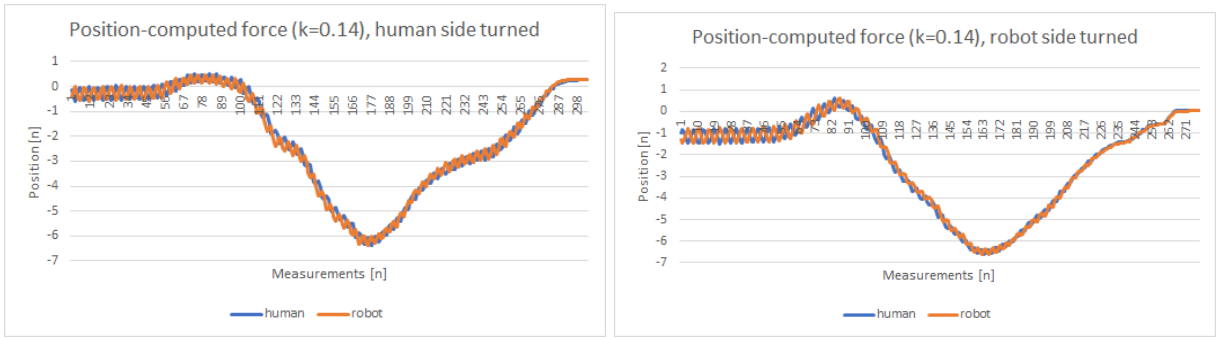

Figure 32: *The response of the system at k=0.12 in the position-computed force architecture*

In order to evaluate the chosen control values in 6.2.8, a comparison of the system's response for different control gains has to be made. For this comparison, the absolute value of each measurement is calculated. Afterwards, these absolute values are averaged and put in a table found in Table 11.

k	Average error turn on human side	Average error turn on robot side
0.00	2.873	2.576
0.02	0.459	0.512
0.05	0.210	0.173
0.07	0.168	0.311
0.10	0.134	0.215
0.12	0.154	0.185
0.14	0.261	0.236

Table 11: *Average error for different control gains in the position-computed force architecture*

As discussed in 6.2.8, the chosen control values for the position-computed force architecture are minimal gain = 0.0, optimal gain = 0.033 and maximum gain = 0.08. Several observations can be made by comparing these values with the findings in Table 11. First, the minimal gain is accurate and shows a system that is little to no responsive. Second, the optimal gain should have been higher. According to the table, the optimal control gain for the position computed force architecture lies somewhere between k=0.10 and k=0.14. Last, the maximum control gain could have been higher. At k=0.08, the system is indeed already unstable. Nevertheless, this effect could be even more noticeable by implementing a higher control gain maximum.

7.3.2 Position-Position Architecture

The same evaluation is done for the position-position architecture. This architecture is evaluated for Zhc = Zrc = 0.01, Zhc = Zrc = 0.05, Zhc = Zrc = 0.1, Zhc = Zrc = 0.15, Zhc = Zrc = 0.20 and Zhc = Zrc = 0.25. The choice is made to let both control impedances be equal to each other, as this is one of the requirements for perfect transparency in the position-position architecture (2.1.5). In Figure 33, the system's response can be seen for Zhc = Zrc = 0.01. One can see that the response when turning the human side of the system is presented, yet not optimal.

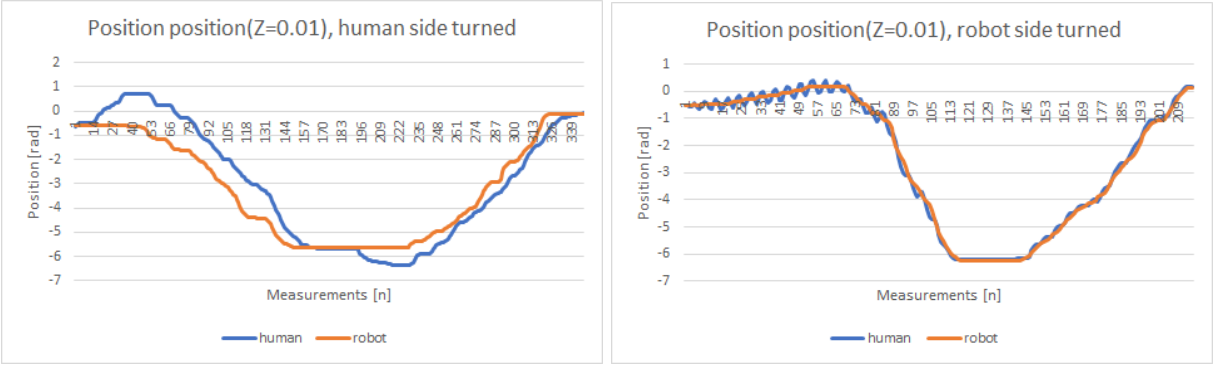

Figure 33: *The response of the system at Z=0.01 in the position-position architecture*

When increasing the control impedance on both sides of the controller, the response of the system becomes more accurate. The response of the system for $Z_{hc} = Z_{rc} = 0.1$ can be seen in Figure 34.

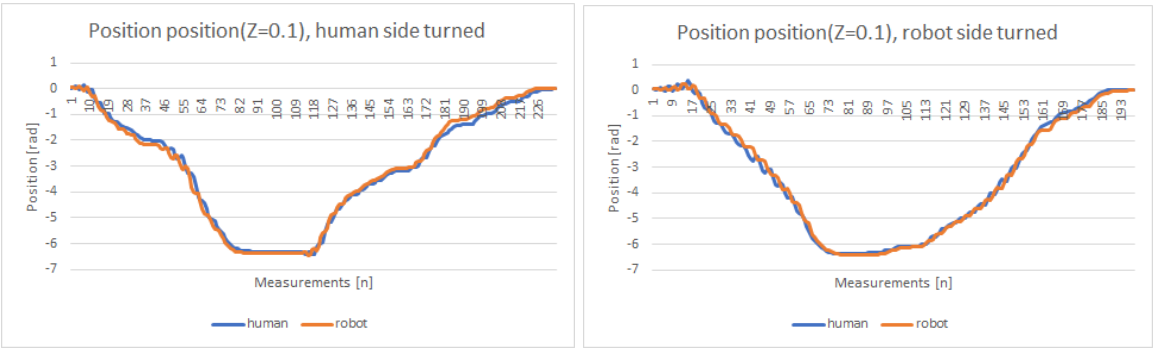

Figure 34: *The response of the system at Z=0.10 in the position-position architecture*

Similiar to the position-computed force, the system becomes unstable for larger control gain. This can be seen in Figure 35 for $Z_{hc} = Z_{rc} = 0.1$.

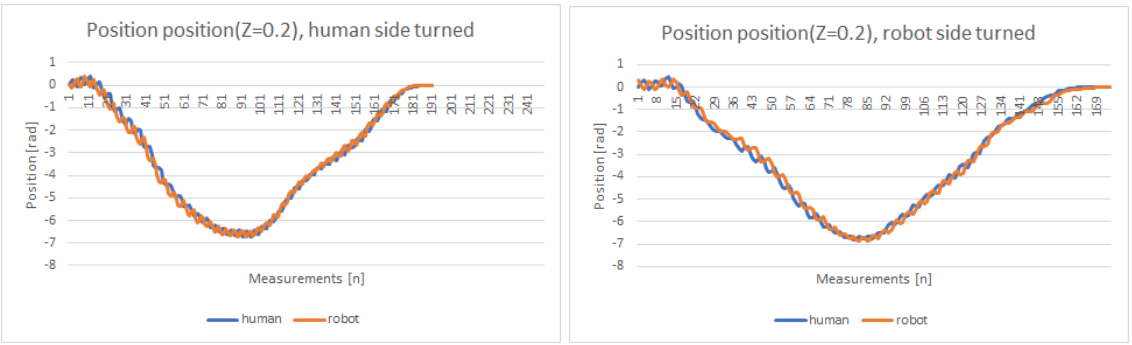

Figure 35: *The response of the system at Z=0.20 in the position-position architecture*

In order to evaluate the chosen control values in 6.2.8, a comparison of the system's response for different control gains has to be made. This comparison is the same as the comparison of the force-computed force architecture. The results for the average error can be in Table 12.

Z_{hc} , Z_{rc}	Average error turn on human side	Average error turn on robot side
0.01	0.693	0.119
0.05	0.174	0.128
0.10	0.136	0.123
0.15	0.231	0.184
0.20	0.207	0.194
0.25	0.343	0.867

Table 12: *Average error for different control impedances in the position-position architecture*

As discussed in 6.2.8, the chosen control values for the position-computed force architecture are, minimal gain = 0.0, optimal gain = 0.079 and maximum gain = 0.30. The following findings are found when comparing these values with the findings in Table 12. The minimum gain should be increased to at least 0.1. For lower control gains, the system did not allow movement, as a result, the evaluating test could not be performed. According to Table 12, the optimal control value lies around an impedance of 0.1. This impedance is quite similar to the estimated optimal gain of 0.079. The maximum gain is too high. At this value, the system was too unstable to perform measurements. Therefore it is wise to change the maximum gain to 0.25.

Chapter 8 - Conclusion

The main research question for this graduation project is: *"What are the most important guidelines when developing software for an educational DIY kit focussing on kinesthetic telemanipulation?".* To answer this main question, the four sub-questions are answered.

The first subquestion is: *"How are the four different controller architectures constructed?".* The goal of every architecture is to reach perfect transparency. This phenomenon is achieved when the position and force on the human side of the controller are equal to the position and force on the robot side of the controller, respectively. Control architectures use control gains, impedances, and feedback loops to achieve this. The main challenges for perfect transparency include time delay and dynamics. The architectures are discussed in detail in Chapter 2 - Background.

The second subquestion is: *"Which aspect(s) of telemanipulation should be highlighted in the product?"*. With the use of stakeholder analysis (3.1.2) and discussion with the client (3.1.3), a requirement list is composed for the product. This requirement list is the foundation for developing the DIY kit and can be found 3.1.6. As a result, three main goals of the DIY kit are found in order to teach the user about kinesthetics in telemanipulation. First, the user is able to see the effect of different control gains on the system. Second, the user is able to see the effect of time delay on the system. Third, the user is able to compare different control architectures.

The third subquestion is: *"What are the most important concerns when designing software for education?".* During the research, little to no literature regarding designing software for education could be found. As a result, the study focussed on more general educational design guidelines, as discussed in 3.1.5. Nine guidelines for educational purposes were found, of which seven applied to the DIY kit. Six of those guidelines are present in the end product, with the most prominent ones being giving feedback and recalling prior learning. The feedback is delivered through the user interface and the kit itself and informs the user about the behaviour of the control architecture. Additionally, recalling prior knowledge is actively presented by including block diagrams in the user interface. As the user has seen these diagrams during the pen casts, it contributes to actively recalling the content. The guideline that is currently not implemented in the kit is setting objectives. Although the learning goals for the DIY kit are formulated, these are not explicitly stated within the kit.

The last subquestion is: *"How can the control gain range within the controller be determined?".* The range of each gain in the DIY kit allows the user to see three main kinds of behaviour in the system. The system can either be no to little responsive due to a low gain, work as expected due to an optimal gain, or be unstable due to a high gain. The range of these gains is determined by the mean of trial and error, as discussed in 3.2.3. These gains are then validated and adjusted by measuring the position of controller inputs and determining perfect transparency. These findings are discussed in 7.3. Comparing the findings with the chosen control gains, it can be said that few adjustments need to be made with regards boundaries and optimal values in the user interface.

These four subquestions help to answer the main question of this graduation project. The main research question is: *"What are the most important guidelines when developing software for an educational DIY kit focussing on kinesthetic telemanipulation?"*. These most important guidelines proceed from the subquestions. The first guideline is the accurate implementation of control architectures. The second guideline is the formulation of the requirements list with the goal of formulating the purpose of the kit. The third guideline is the inclusion of educational design guidelines to make the kit suitable for educational purposes. The last guideline is accurate parameter tuning to show the user different kinds of behaviour within the kit. When considering these guidelines while developing software for an educational DIY kit focussing on kinesthetic telemanipulation, one can create a kit based on theoretical research.

The kit is developed as part of the educational platform avatars.report. However, it is also interesting to take a look at the kit in larger content. As stated in the Introduction, robots are becoming increasingly popular. As a result, people who know about them are needed. This entails knowledge of maintaining, developing and improving current and future robotics systems. The DIY kit gives a practical touch to the theoretical concept of control theory. The kit is an interesting, understandable and easy way for people to deepen their interest in the robotics field, different from the standard teaching methods. It allows the user to obtain deeper knowledge by experimenting and feeling the setup in real life.

8.1 Future Work

There is still room for improvement in the current DIY kit. First and foremost, the hardware of the kit needs to be replaced. Currently, the kit works with alternative hardware instead of the hardware designed for the kit. With the new hardware, it is possible to implement the remaining two control architectures and add dynamics compensation to the kit. Second, the user interface should be subjected to user evaluation. The intended user experience can be validated and improved by evaluating the kit with the user. Third, documentation should be included to give the user an explanation about the kit and to convey the learning goals. Last, additional features can be added to enhance the impact of the user interface further. Multiple interesting ideas are generated but have yet to be implemented due to time delay. First, two feedback bars could be included in the user interface. This way, the user is able to get both feedback based on the theoretical and practical behaviour of the system. Additionally, the control impedances in the position-position architecture can be expanded, meaning that the user can adjust the individual gains that are part of the impedance. Furthermore, the optimal control parameters should be adjusted according to the findings in 7.3 to give the user more accurate feedback. Moreover, the control parameters should be tunend and evaluated more accurately to give even better feedback.

Chapter 9 - Discussion

At the beginning of this project, the goal was to make software for a DIY kit focussing on kinesthetic within telemanipulation. This software should consist of a user interface and the control architectures on a microcontroller. The project scope expanded when the originally outsourced hardware was unavailable in time. An alternative was found as the hardware is an essential part of the project for testing and validating purposes. This alternative resulted in creating an independent, separate hardware setup compatible with at least testing two of the four architectures. Although it fulfilled the testing purpose, it took away time from developing the original project. As a result, the end product is limited in its features and needs to be validated with user testing. Missing features include but are not limited to the inclusion of documentation of the kit, implementing the two remaining architectures, including dynamics compensation and implementing more elements within the user interface. In short, the outsourced hardware not being available in time firmly impacted this research's in-depth software development possibilities.

Additionally, a critical remark must be made in the evaluation of the kit. This regards evaluating the implementation of the control architecture, as discussed in 7.3 In this test, the architectures are evaluated by turning the wheel clockwise and counterclockwise on both the human and robot side of the system. However, this turning movement is rather subjective in both precision and timing. Regarding precision, a mark was added to the wheel better to monitor the start and end points. Nevertheless, the movement is still variable for each turn. The same holds for the timing of the movement. It was intended to turn the wheel concisely amongst all the measurements, but no assurance can be given. Especially for lower control gains, a faster movement can result in a decrease in perfect transparency. All in all, this has decreased the precision in evaluating the transparency within the kit.

During the evaluation, inconsistencies were detected in the system at higher control values. When turning the wheel on the robot side, the human side would not react. Instead, resistance on the robot side could be felt. This phenomenon is most likely due to a position-dependent friction component at the human side of the system. This friction is higher than the delivered motor force, and as a result, the system will not move.

Sources

Ackerman, E. (2023, April 25). Your robotic avatar is almost ready. IEEE Spectrum. https://spectrum.ieee.org/xprize-robot-avatar

Arduino Uno REV3. Arduino Official Store. (n.d.). https://store.arduino.cc/products/arduino-uno-rev3

Avatar Robotics. avatars.report. (n.d.). https://avatars.report/

Benyon, D. (2020). Designing user experience: A guide to HCI, UX and interaction design (4th ed.). Pearson.

Broenink, J.F. (2000). Introduction to Physical Systems Modelling with Bond Graphs.

Brush, K. (2023, March 29). MoSCoW method. Software Quality. https://www.techtarget.com/searchsoftwarequality/definition/MoSCoW-method

Clegg, D., & Barker, R. (1994). Case method fast-track: A rad approach. Oracle.

Dresscher, D. (2023). Lab 1: Control. https://docs.google.com/document/d/1E1aXh9RQpxk3lNkXr3Logsn30geMUTFXxl6uQctL7B0/edit

Elements of instruction. Elements of Instruction | University of Illinois Springfield. (n.d.). https://www.uis.edu/ion/resources/tutorials/pedagogy/elements-of-instruction

Encoder. Encoder - Arduino Reference. (n.d.). https://www.arduino.cc/reference/en/libraries/encoder/

ESP32 WIFI and Bluetooth board - CP2102. ESP32CP2102V1. (n.d.). https://www.tinytronics.nl/shop/en/development-boards/microcontroller-boards/with-wi-fi/esp32-wifi-and-bluetooth-board-cp2102

Every, P. (2022, November 29). Stakeholder management using the power interest matrix. Solitaire Consulting. https://www.solitaireconsulting.com/2020/07/stakeholder-management-using-the-power-interest-matrix/

Gagne's nine events of instruction: Center for Innovative Teaching and learning. Northern Illinois University. (n.d.).

https://www.niu.edu/citl/resources/guides/instructional-guide/gagnes-nine-events-of-instruction.shtml

Gagné, R., Keller, J. M., Wager, W. W., & Briggs, L. J. (2004). Principles of Instructional Design (5th ed.). Cengage Learning.

Gelal, O. (2015, November 27). Ultimate Guide to the processing language part I: The fundamentals: Toptal®. Toptal Engineering Blog. https://www.toptal.com/game/ultimate-guide-to-processing-the-fundamentals

Kim, E., Liu, B., Roehm, T., & Tout, S. (2009). Ch 11.1 Feedback control: What is it? When useful? When not? Common usage. Woolf, P., In Chemical Process Dynamics and Controls. Openmichigan. https://ia800701.us.archive.org/28/items/ChemicalProcessDynamicsAndControls/ChemicalProcessDynamicsAndControls.pdf

Mader, A., & Effrink, W. (2014). A DESIGN PROCESS FOR CREATIVE TECHNOLOGY. https://www.researchgate.net/publication/265755092_A_DESIGN_PROCESS_FOR_CREATIVE_TECHNOLOGY

Malviya, G. (n.d.). *EOL or end of line or newline ASCII character: Loginradius blog*. loginradius. https://www.loginradius.com/blog/engineering/eol-end-of-line-or-newline-characters/

Melchiorri, C. (2003). Robotic Telemanipulation Systems: An overview on control aspects. IFAC Proceedings Volumes, 36(17), 21–30. https://doi.org/10.1016/s1474-6670(17)33365-7

Moscow prioritization. What is MoSCoW Prioritization? | Overview of the MoSCoW Method. (2022, December 21). https://www.productplan.com/glossary/moscow-prioritization/

Olesen, J. (n.d.). The power and symbolism of colors (infographics). Color Meanings. https://www.color-meanings.com/

Python introduction. Introduction to Python. (n.d.). https://www.w3schools.com/python/python_intro.asp

Scrollbar / examples. Processing. (n.d.). https://processing.org/examples/scrollbar.html

Tan, C. (2023, January 4). Esp32 vs Arduino: The main differences. All3DP. https://all3dp.com/2/esp32-vs-arduino-differences/

Appendix

Appendix A: Interface PCB - Schematic

This appendix contains the schematic of the interface PCB.

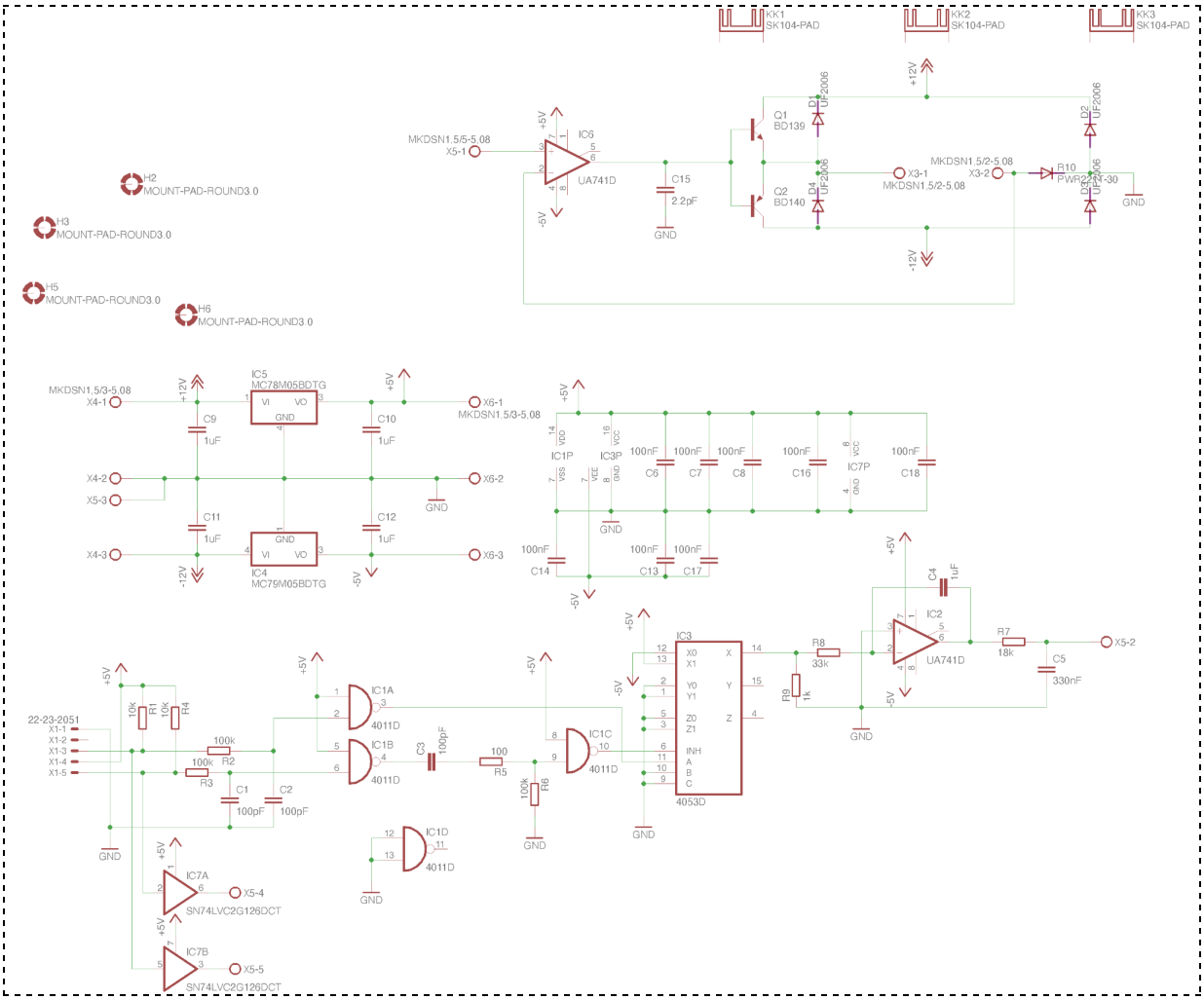

Figure A1: *Schematic of interface PCB presented on the motor set-up*

Appendix B: Custom Arduino Shield - Schematic

This appendix contains the schematic of the custom Arduino shield.

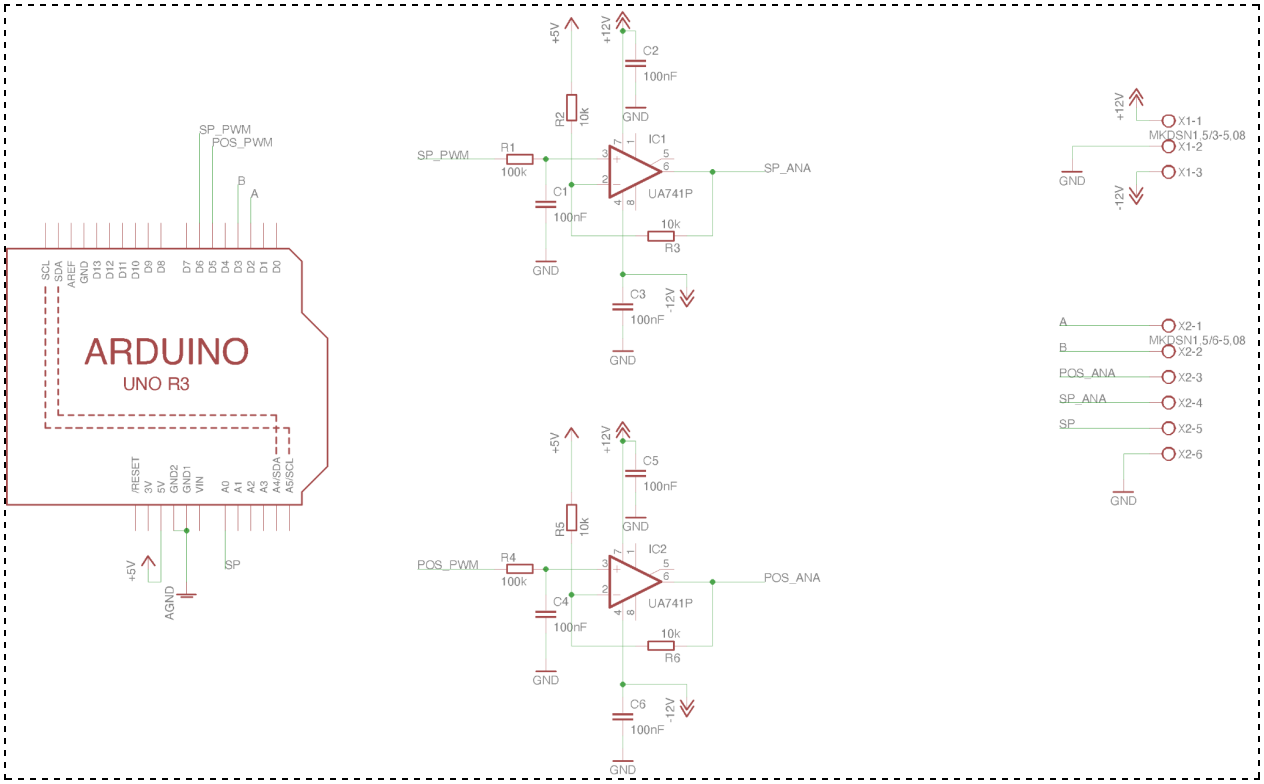

Figure A2: *Schematic of the Arduino shield used in the setup*

Appendix C: Arduino Code

This appendix contains the full Arduino Uno code.

```
#include <Encoder.h>

int outputPin1 = 6;   // PWM pin 6 as output pin
int outputPin2 = 11;  // PWM pin 6 as output pin

Encoder encoder1(2, 3);
Encoder encoder2(18, 19);

long encoderPos1 = 0;  //this variable stores our current value of encoder1
position.
long encoderPos2 = 0;  //this variable stores our current value of encoder2
position.

float loopTime;
unsigned long triggerTime;  // operating frequency of the PD controller

float Zhc, Zrc, Kp, Kd, Ki;  // variables for the constants
float Km = 0.028;            // motor constant is 2.8Ncm/A
float loopfrequency = 500;
float fh = 100;  //Hz
float pi = 3.14;

float phi1, phi2, i1, i2, error1, error2, dedt1, dedt2, output1, output2;
float rate1, rate2, total1, total2, outputPos;  // variables for angle,
setpoint and error
// variable to store previous error
float tau1 = 0;                                 // steer signal to send to
the motor
float tau2 = 0;                                 // steer signal to send to
the motor
int outputPWM1 = 0;                             // steersignal mapped from
0 to 255
int outputPWM2 = 0;

int selectArchitecture = 0;  //0 = position-computed force, 1 =
position-postition
String buffer;

void setup() {
```

```
  //------ insert constant values here!! ------//
  Zhc = 0.002;
  Zrc = 0.002;
  Kp = 0.02;    //0.02;//0.028; // Insert calculated value for kP here
  Kd = 0.0003;  //0.0001295;// // Insert calculated value for kD here
  Ki = 0;       // Change value to add integrator, if desired

  pinMode(outputPin1, OUTPUT);
  pinMode(outputPin2, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  processingInput();
  controlCode();
}

void processingInput() {
  if (Serial.available() > 0) {
    char input = (char)Serial.read();
    if (input != ';') {
      buffer += input;
    } else {
      buffer.trim();
      char architecture = buffer.charAt(0);
      switch (architecture) {
        case 'a':  // select architecture
          selectArchitecture = buffer.substring(1).toDouble();
          break;
        case 'b':  //time delay
          // loopfrequency = buffer.substring(1).toDouble();
          break;
        case 'c':  //kp
          Kp = buffer.substring(1).toDouble();
          break;
        case 'd':  //Zhc
          Zhc = buffer.substring(1).toDouble();
          break;
        case 'e':  //Zrc
          Zrc = buffer.substring(1).toDouble();
          break;
        default:
```

```
          break;
      }
      buffer = "";
    }
  }
}

void controlCode() {
  //------- calculate the current loop time and wait until the set loop
time is passed -----------//
  loopTime = millis() - triggerTime;         // the time since the previous
loop
  while (loopTime < 1000 / loopfrequency) {  // 500Hz is every 2ms
    loopTime = millis() - triggerTime;       // the time since the previous
loop
  }
  if (loopTime > 1000 / loopfrequency + 1) {  //+1 to have some tolerance
    Serial.print("Looptime (ms) = ");
    Serial.print(loopTime);
    Serial.print("!!!\n");
  }
  triggerTime = millis();  // reset triggerTime
  // END calculate the current loop time -------------//

  //------ read position from encoder ------//
  encoderPos1 = encoder1.read();
  encoderPos2 = -encoder2.read();

  phi1 = encoderPos1 / 63.66;  // position in radians = nr. of pulses
divided by number of pulses per radian (400 pulses per revolution)
  phi2 = encoderPos2 / 63.66;  // position in radians = nr. of pulses
divided by number of pulses per radian (400 pulses per revolution)

  //------ END read position from encoder ------//

  //------ calculation of the steer signal value / PD control ------//

  //Calculation of the error
  error1 = phi2 - phi1;

  rate1 = -2 * pi * fh * dedt1;
  total1 = total1 + rate1 * (1 / loopfrequency);
  dedt1 = 2 * pi * fh * error1 + total1;
```

```
//Calculation of the steer signal
tau1 = error1 * Kp + dedt1 * Kd;
if (selectArchitecture == 0) {  //position-computed force
  tau2 = -tau1;
}
if (selectArchitecture == 1) {  //position-position
  error2 = phi1 - phi2;

  rate2 = -2 * pi * fh * dedt2;
  total2 = total2 + rate2 * (1 / loopfrequency);
  dedt2 = 2 * pi * fh * error2 + total2;

  tau2 = error2 * Zrc + dedt2 * Kd;
}

//calculate the required current
i1 = tau1 / Km;
i2 = tau2 / Km;

//Limit the current output to 1A
if (i1 > 1) { i1 = 1; }  // max bound is 255
else if (i1 < -1) {
  i1 = -1;
}  // min bound is 0

if (i2 > 1) { i2 = 1; }  // max bound is 255
else if (i2 < -1) {
  i2 = -1;
}  // min bound is 0

//Compensate for 2 gain in opamp circuit and the -5 in the op-amp circuit
output1 = (i1 + 5) / 2;
output2 = (i2 + 5) / 2;

// map the output 0V..5V to 0...255 for the pwm pin
outputPWM1 = (51 * output1);
outputPWM2 = (51 * output2);

//------ END calculation of the steer signal value / PD control ------//

//------ write steervalue to PWM pin ------//
analogWrite(outputPin1, outputPWM1);  // analogwrite to PWM pin
```

```
  analogWrite(outputPin2, outputPWM2);  // analogwrite to PWM pin
}
```

Appendix D: Processing Code

This appendix contains the full Processing code of the user interface.

```
// sktech with inherentance and the two architectures with the serial
communication on

import processing.serial.*;

Serial port;
ScreenHandler screenHandler;
String val;
boolean firstContact = false;
boolean firstMousePress = false;

void setup() {
  size(1200, 800);
  screenHandler = new ScreenHandler();
  port = new Serial(this, Serial.list()[1],9600);

  // Print the available serial ports
  println("Available serial ports:");
  for (int i = 0; i<Serial.list ().length; i++) {
    print("[" + i + "] ");
    println(Serial.list()[i]);
  }
}

void draw() {
  screenHandler.update();
  screenHandler.display();

  if (firstMousePress) {
    firstMousePress = false;
  }
}

void keyPressed() {
  if (keyCode == ENTER && screenHandler.getScreenName() == "start") {
    screenHandler.setScreenName("architecture");
  }
  if (key == '3' && screenHandler.getScreenName()  == "architecture") {
    screenHandler.setScreenName("posPosArchitecture");
```

```
    port.write("a1;");
  }
  if (key == '1' && screenHandler.getScreenName()  == "architecture") {
    screenHandler.setScreenName("posComputedForce");
    port.write("a0;");
  }
  if (keyCode == BACKSPACE && screenHandler.getScreenName() ==
"architecture") {
    screenHandler.setScreenName("start");
  }
  if (keyCode == BACKSPACE && screenHandler.getScreenName() ==
"posPosArchitecture") {
    screenHandler.setScreenName("architecture");
  }
  if (keyCode == BACKSPACE && screenHandler.getScreenName() ==
"posComputedForce") {
    screenHandler.setScreenName("architecture");
  }
}

void mousePressed() {
  if (!firstMousePress) {
    firstMousePress = true;
  }
}
```

```
class Architecture {
  PImage background;
  ArrayList<Slider> sliders;
  FeedbackBar feedbackBar;

  Architecture() {
    sliders = new ArrayList<Slider>();
  }

  void display() {
    image (background, 0, 0);
    displayText();
    for (int i=0; i< sliders.size(); i++) {
      sliders.get(i).display();
    }
    feedbackBar.display();
```

```
  }

  void update() {
    for (int i=0; i< sliders.size(); i++) {
      sliders.get(i).update();
    }
    feedbackBar.update(calculateOptimalization());
    arduinoUpdate();
  }

  void displayText() {
  }

  float calculateOptimalization() {
    float optimalization = 0;
    for (int i=0; i< sliders.size()-1; i++) {
      optimalization += sliders.get(i).getOptimalization();
    }
    float optimalizationPercentage = map(optimalization, 0,
sliders.size()-1, 0, 1);
    return optimalizationPercentage;
  }

  void arduinoUpdate() {
    for (int i=0; i< sliders.size(); i++) {
      if (sliders.get(i).isChanged()) {
        sliders.get(i).sendToArduino();
      }
    }
  }
}
```

```
class Background {
  PImage startScreen;
  PImage architecture;

  Background(){
    startScreen = loadImage("StartScreen.png");
    architecture = loadImage("Architecture.png");
  }

  void displayStart(){
```

```
    image(startScreen,0,0);
  }

  void displayArchitecture(){
    image(architecture,0,0);
  }
}
```

```
class PositionComputedForce extends Architecture{

  PositionComputedForce() {
    super.background = loadImage("PositionComputedForce.png");
    super.sliders.add(new Slider(180, 650, 250, 35, 0.0, 0.08, 0.033, 7,
'c')); //k
    super.sliders.add(new Slider (680, 650, 250, 35, 0, 1, 0, 5, 'b'));
//time delay
    super.feedbackBar = new FeedbackBar(1060, 150, 75, 450);
  }

  void displayText() {
    fill(0);
    textSize(30);
    text("k", 145, 660);
    text("Time", 602, 660);
    text("Delay", 602, 700);
  }
}
```

```
class PositionPosition extends Architecture {

  PositionPosition() {
    super.background = loadImage("BackgroundPosPos.png");
    super.sliders.add(new Slider(200, 580, 250, 35, 0, 0.3, 0.079, 5,
'd')); //zhc
    super.sliders.add(new Slider(700, 580, 250, 35, 0, 0.3,0.079, 5, 'e'));
//zrc
    super.sliders.add(new Slider (200, 680, 250, 35, 0, 1, 0, 5, 'b'));
//time
    super.feedbackBar = new FeedbackBar(1060, 150, 75, 450);
  }
```

```
  void displayText() {
    fill(0);
    textSize(30);
    text("Zhc", 135, 590);
    text("Zrc", 635, 590);
    text("Time", 122, 670);
    text("Delay", 122, 710);
  }
}
```

```
class Slider {
  float xBar, yBar;
  int widthBar, heightBar;
  float spos, newspos;
  float sliderMinPos, sliderMaxPos;
  float valueSlider;
  float optimalValue;
  float valueMin, valueMax;
  char header;
  int speed;
  boolean over;
  boolean locked;
  float ratio;

  Slider (float x, float y, int w, int h, float min, float max, float
optimal, int s, char head) {
    widthBar = w;
    heightBar = h;
    xBar = x;
    yBar = y-heightBar/2;
    spos = xBar + widthBar/2 - heightBar/2;
    newspos = spos;
    sliderMinPos = xBar;
    sliderMaxPos = xBar + widthBar - heightBar;
    valueMin = min;
    valueMax = max;
    optimalValue = optimal;
    speed = s;
    header = head;
  }

  void update() {
```

```
    checkOver();
    if (firstMousePress && over) { //Makes sure only one slider can move
once mousepressed
      locked = true;
    }
    if (!mousePressed) {
      locked = false;
    }
    if (locked) {
      newspos = constrain(mouseX-heightBar/2, sliderMinPos, sliderMaxPos);
    }
    spos = spos + (newspos-spos)/speed;
    valueSlider = map(spos, sliderMinPos, sliderMaxPos, valueMin,
valueMax);
  }

  float constrain(float val, float minv, float maxv) {
    return min(max(val, minv), maxv);
  }

  void checkOver() {
    if (mouseX > xBar && mouseX < xBar+widthBar &&
      mouseY > yBar && mouseY < yBar+heightBar) {
      over =  true;
    } else {
      over = false;
    }
  }

  void display() {
    noStroke();
    fill(253, 248, 225);
    rect(xBar, yBar, widthBar, heightBar);
    if (over || locked) {
      fill(0, 0, 0);
    } else {
      fill(250, 229, 136);
    }
    rect(spos, yBar, heightBar, heightBar);
    fill(0);
    textSize(30);
    text(nf(valueSlider, 0, 3), xBar + widthBar +10, yBar+27);
  }
```

```
  float getOptimalization () {
    float maxError = max(optimalValue-valueMin,
abs(optimalValue-valueMax));
    float optimalization = abs(optimalValue-valueSlider) / maxError;
//value between 0 (best) and 1(worst)
    return optimalization;
  }

  float getValue() {
    return valueSlider;
  }

  boolean isChanged() {
    return (newspos > (spos+0.1) || newspos <(spos-0.1));
  }

  void sendToArduino(){
    String serialValue = header + nf(valueSlider,2,3) + ";";
    serialValue = serialValue.replaceAll(",",".");
    port.write(serialValue);
  }
}
```

```
class FeedbackBar {
  int xBar, yBar;
  int widthBar, heightBar;
  float xSlider, ySlider;
  int widthSlider, heightSlider;

  FeedbackBar(int x, int y, int w, int h) {
    xBar = x;
    yBar = y;
    widthBar = w;
    heightBar = h;
    xSlider = xBar;
    ySlider = yBar + heightBar/2;
    widthSlider = widthBar;
    heightSlider = 10;
  }

  void display() {
```

```
    fill(0); // black
    rect(xBar-3, yBar-3, widthBar+6, heightBar+6);
    fill(124, 255, 0); //green
    fill(249, 220, 92);
    rect(xBar, yBar, widthBar, heightBar/3);
    fill(255, 165, 0); //orange
    fill(252, 239, 180);
    rect(xBar, yBar+heightBar/3, widthBar, heightBar/3);
    fill(255, 0, 0); // red
    fill(253, 248, 225);
    rect(xBar, yBar+2*heightBar/3, widthBar, heightBar/3);
    fill(0);
    rect(xSlider, ySlider, widthSlider, heightSlider);
    pushMatrix();
    rotate(radians(90));
    textSize(30);
    text("Optimalization", yBar + heightBar/3-20, -xBar - widthBar- 15);
    popMatrix();
    textSize(15);
    text("High", xBar + widthBar +10 ,yBar +8);
    text("Low", xBar + widthBar +10 ,yBar+ heightBar -8);
  }

  void update(float percentage) {
    ySlider = yBar + percentage* (heightBar-heightSlider);
  }
}
```

```
class ScreenHandler {
  Background background;
  PositionPosition posPosArchitecture;
  PositionComputedForce posComputedForce;
  String screen;

  ScreenHandler() {
    background = new Background();
    posPosArchitecture = new PositionPosition();
    posComputedForce = new PositionComputedForce();
    screen = "start";
  }

  void display() {
```

```
    if (screen == "start") {
      background.displayStart();
    }
    if (screen == "architecture") {
      background.displayArchitecture();
    }
    if (screen == "posPosArchitecture") {
      posPosArchitecture.display();
    }
    if (screen == "posComputedForce") {
      posComputedForce.display();
    }
  }

  void update() {
    if (screen == "posPosArchitecture") {
      posPosArchitecture.update();
    }
    if (screen == "posComputedForce") {
      posComputedForce.update();
    }
  }

  String getScreenName () {
    return screen;
  }

  // This function sets screen to name.
  void setScreenName(String name) {
    screen = name;
  }
}
```