HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

DIPLOMARBEIT

FLAME - FLexible robot Arm for Mobile Expandable systems

Ausgeführt im Schuljahr 2021/22 von:

Software zur Ansteuerung des Roboterarms
Antonia Oberhauser-Tomasova — 5BHIF

Kommunikation zwischen dem Arduino und dem ODrive-System
Niklas Wieser — 5BHIF

Konstruktion sowie Fertigung des Roboterarms
Florian Zachs — 5BHMBA

Betreuer:

MMag. Dr. Michael Stifter
Dipl.-Ing. Martin Schubert

Wiener Neustadt, 4. April 2022

Abgabevermerk: Übernommen von:

Chapter 1

Eidestattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angegeben Quellen und Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Wiener Neustadt am 4. April 2022

Verfasser / Verfasserin:

Niklas Wieser

Florian Zachs

Antonia Oberhauser-Tomasova

Contents

Chapter 2

Acknowledgement

First, the authors would like to thank everybody for their support throughout this diploma thesis.
It required much effort and took much time. Special thanks go to our supervisors, MMag. Dr. Michael Stifter and Dipl.-Ing. Martin Schubert. They always had time and provided constructive and precise feedback.
The authors also want to thank everyone from the ODrive community who responded to the questions in the ODrive Forum and provided the right solutions when things did not go as planned.
Last but not least, biggest thanks to our families and friends who always believed in us and helped us maintain motivation throughout the year.

Chapter 3

Kurzfassung

Roboterarme gibt es bereits seit einigen Jahrzehnten und nehmen uns Menschen in diversen Branchen viel Arbeit ab. Industrielle Robotersysteme sind oft komplex und auch teuer. Besonders im Ausbildungs- und Hobbybereich gibt es noch einen erkennbaren Mangel an performanten und preiswerten Produkten, welche für Bildungszwecke eingesetzt werden können, um Schülerinnen und Schüler dadurch bestmöglich auf ihre berufliche Zukunft vorzubereiten. Das FLAME Projekt beschäftigt sich daher mit der Entwicklung eines Roboterarms, welcher genau diese Marktlücke schließen soll. Eines der größten Alleinstellungsmerkmale ist die Flexibilität, sowohl von der Software- als auch von der Hardwareperspektive. Darunter fällt die Möglichkeit, den Roboterarm stationär, sowie auch mobil auf einem Festo Robotino® zu betreiben. Ein weiteres Merkmal ist die Skalierbarkeit. Die Leistungsfähigkeit der verbauten Elektromotoren kann individuell angepasst werden und die Anzahl der verbauten Motoren ist ebenso variabel. Die Steuerung und daraus folgende Berechnungen finden zentral auf einem Rechner statt.
Im Laufe dieser Diplomarbeit wird die Verwendung von *ODrive* als Motorkontrollsystem und dessen Implentierung mit zentraler Steuerung, sowie mechanische und elektrische Designentscheidungen erläutert. Zusätzlich wird sich intensivst mit Inverser Kinematik beschäftigt und eine Schnittstelle zu möglichen graphischen Benutzeroberflächen ist auch gegeben. Am Ende wurde der Roboterarm zusammengebaut, auf seine Funktionsweise geprüft und kann damit für zukünftige Projekte verwendet werden.

Chapter 4

Abstract

Robotic arms have been around for several decades and have replaced many jobs in various industries. However, industrial robot systems are often very complex and also very expensive. Especially in the training and hobby area, there is still a noticeable lack of high-performance and inexpensive products that can be used for educational purposes and to prepare students as best as possible for their professional future.
Therefore, the FLAME project focuses on developing a robotic arm that is intended to fill this gap in the market. One of the most significant unique selling points is its flexibility, both from a software and hardware point, including the option of operating the robot arm in a stationary or mobile manner on a Festo Robotino®. Another selling point is scalability. The performance of the built-in electric motors can be individually adjusted, and the number of axes is also variable. All calculations for controlling the robot's axes are carried out on a centralized processing unit such as a dedicated PC, laptop or the Festo Robotino®.
This diploma thesis explains the use of *ODrive* as a motor control system and its implementation with a central control unit, and mechanical and electrical design decisions. In addition, inverse kinematics gets dealt with extensively, and an interface to future graphical user interfaces is also available. In the end, the robotic arm gets assembled and tested for functionality.

Chapter 5

Introduction

Author: Niklas Wieser

5.1 History of Robotics

Since the 1960s, industrial robots have been introduced in the automotive industry, particularly in countries such as the USA, Western Germany and Japan[1]. Back in the 70s, a lack of skilled workers led to increased development and production of hydraulic robots. Those were able to do welding under a very high load and helped ramp up production rates in the following years. Today, manufacturers like ABB, Fanuc, Kuka and many others are improving their products year after year at such a high rate that tasks that only could be done by humans are now slowly but surely taken over by robots.

5.2 Why robots are needed

Apart from vacuuming the floor or mowing the lawn, modern robots can alleviate strenuous tasks or complete too dangerous missions for humans to undertake. Some robots work in nearly any environment, lift heavy loads, handle toxic substances and do repetitive tasks. This has helped companies prevent many fatal accidents and save a lot of time and money.[3] In medical environments, robots are used to assist surgeons during intricate surgeries and help save countless lives because they can reach areas where human hands cannot allow enough accuracy.

[1] Unger-Leinhos, *Die Geschichte der Robotertechnik begann 1954*.
[2] Robotic Industries Association, *unimate_1-9518fc74.JPG*
[3] European Space Agency, *Robotics for Society*.
[4] Greg Nichols, *robotic_surgery-0.JPG*

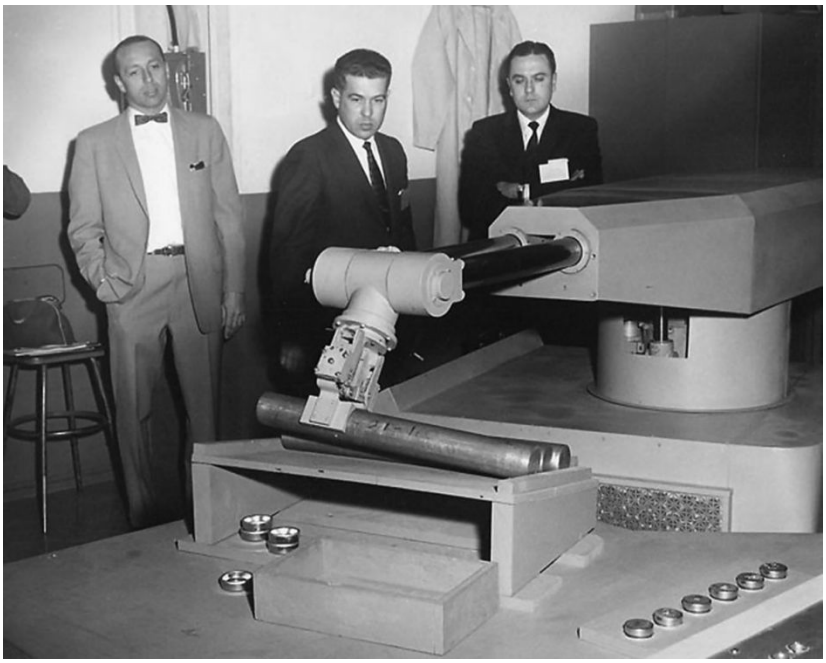


Figure 5.1: "Unimate", the first industrial robot. [2]

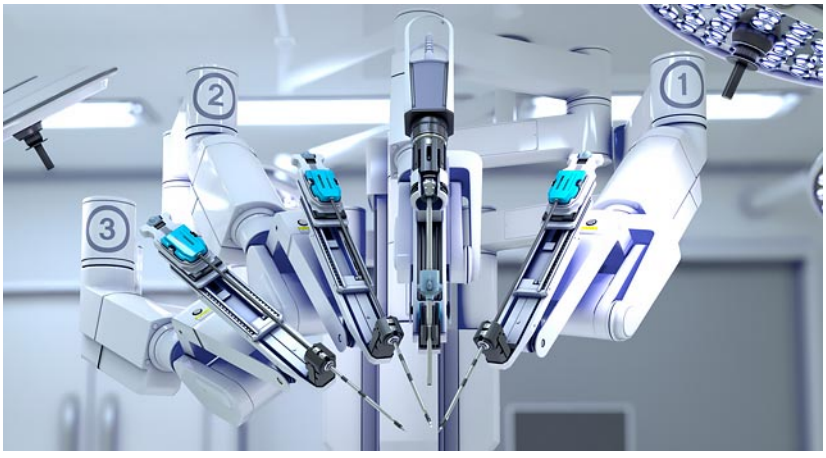


Figure 5.2: A surgical robot for hospitals [4]

Figure 5.3: Since 2012, Marsrover Curiosity is demonstrating the importance of robotics in space exploration. [6]

5.3 Robotics in education

The world is changing, and therefore education needs to adapt to the new challenges of the modern world. The interest in the educational use of robotics has increased massively over the last years, and schools all around the globe are trying to introduce a new way of learning - learning by doing[7].
Building and programming a robot in small steps and in a defined environment is very satisfying for students, as progress is always visible. Other important factors of learning by doing are having fun and working together as a team, which is usually the case when students get in touch with robots.

5.3.1 From Lego Mindstorms up to the Festo Robotino®

The Lego Mindstorms robots are based on research and ideas from the Lifelong Kindergarten group at the MIT Media Lab and are being used all over the world in elementary and secondary education and higher education. These kits from Lego are the perfect start in robotics as they provide everything students need to learn some basics of programming and to dive into the world of robotics[8].

[6] Joe Pappalardo, *54ca5599efee7_-_curiosity-0612-de.JPG*
[7] Dimitris Alimisis, Chronis Kynogos, *Constructionism and robotics in education*.
[8] Ibid.

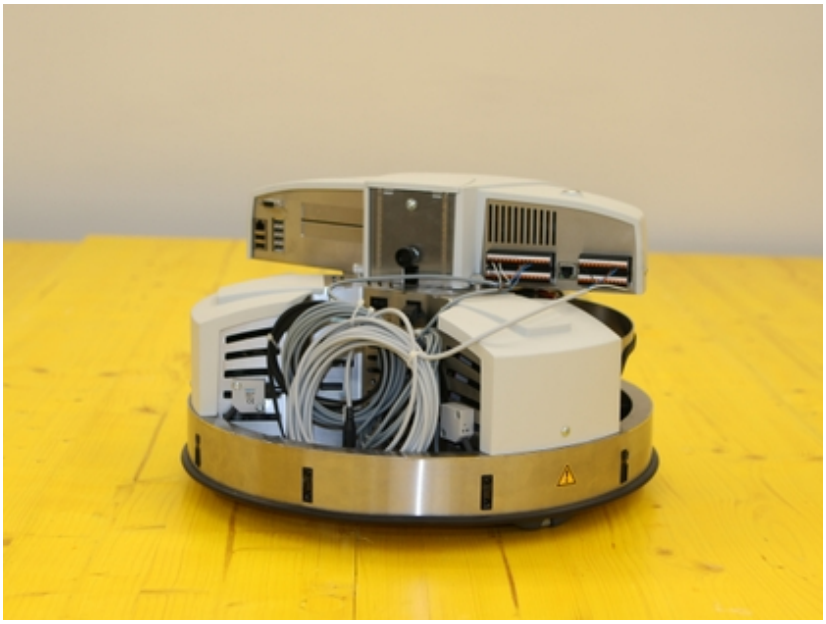


Figure 5.4: The Festo Robotino® is a ground based robot system designed for education and provides a perfect base for the robotic arm developed in the scope of this project. [11]

5.3.2 Competition is always part of the game

Participating in competitions is always the most exciting part of robotics. It begins with little contests within the school and rises to national or even international championships like the Global Conference on Educational Robotics - GCER - hosted by KIPR in Oklahoma, USA.[9] Unlike the GCER, more significant competitions often require the teams to use expensive manipulators on their robots. This diploma thesis is the first step towards an inexpensive, self-developed robotic arm to allow entry into such championships.

5.4 Goal of this diploma thesis

The robotics and mechanical engineering departments of the HTL Wiener Neustadt are constantly looking for ways to increase their competences in robotics and automation. For this reason, the decision was made to develop a robot arm to be used with the Festo Robotino®. The product is designed to be as modular and universal as possible so that others can modify parts or even develop their Add-ons like a custom gripper or special sensors. The goal is to develop a solid and scalable foundation for many more exciting diploma theses and SYP

[9] *KIPR Website*.
[11] Konstantin Lampalzer, *robotino.JPG (JPEG*

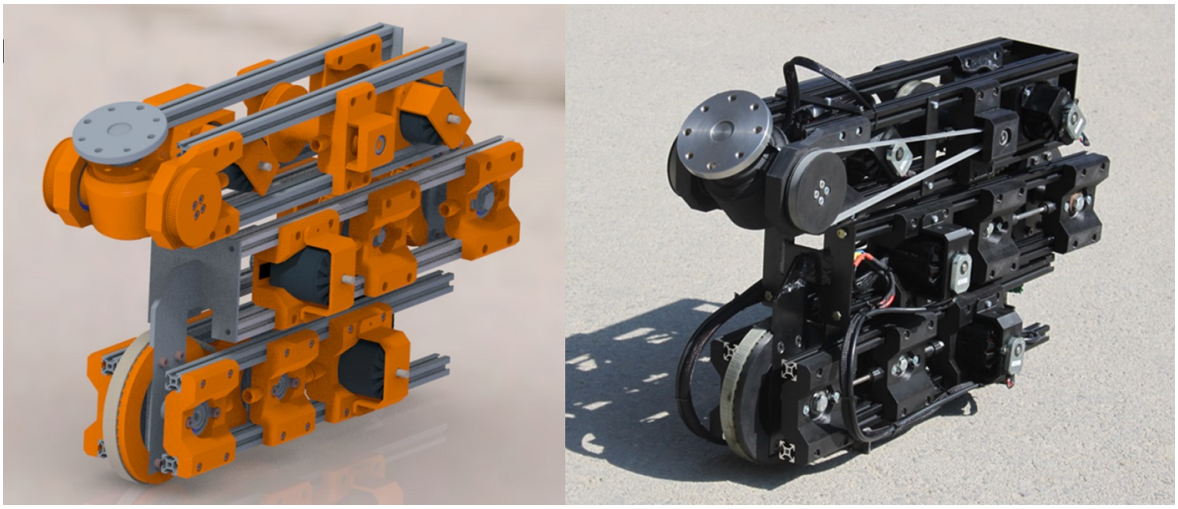


Figure 5.5: Side-by-side comparison of the 3D model and the final assembly

projects[12].

From a software point of view, a C++ library with a user-friendly API is developed, enabling future users to integrate the manipulator into larger projects easily. The main advantage of a C++ library is its performance and reliability. The API is designed to be as simple as possible to make it suitable for beginners and inexperienced programmers.

On the hardware side, technologies like 3D printing and metal turning are used for manufacturing the parts. The distance between the axes is variable in a specific range to make it easier to adapt the manipulator to the requirements. Brushless servo motors drive the robot. The batteries of the Festo Robotino® are used as the power supply, and additional batteries can be added if necessary.

As the entire product was designed from the ground up a lot of mechanical and electrical design challenges showed up, many of them are described in the following chapters. After all design challenges were solved and all decisions were made the product was manufactured and assembled. Everything went to plan, the final assembly can be seen in Figure 5.5.

[12]educational medium-sized software projects

Chapter 6

Project Management

Author: Niklas Wieser

Working on a larger project over a long period can get quite challenging for everyone involved. Keeping everything on track and working towards the targets requires some action. It was popular to use project management techniques where the entire project is entirely planned and then carried out. However, this only works well when everyone involved is very experienced and the entire project can be planned. This does not apply to this diploma thesis, as many technologies were explored while developing. For this reason, an agile project management technique was way more suited.

6.1 Agile Management

When working on pioneering projects, especially in software development, agile project management techniques became the standard. These methods take a new iterative approach with the principle of continual improvement in contrast to the traditional methods. This enables the project to adapt and change direction if problems arise with the current solution. These methods are generally based on the agile manifesto with the following statements at its core[1]:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over a following plan

Keeping these in mind helps to maximize the success of the project.

[1] Hoda et al., "Agile project management".

6.2 Trello board

Trello is a free-to-use collaboration tool that helps organise projects. It is a digital version of a Kanban[2] board. At the beginning of the project the board was filled with several tasks.

Figure 6.1: The board to monitor the progress of the project.

6.3 Meetings with the supervisors

The plan was to conduct a weekly meeting with the supervisors when the schedule allowed. These meetings took between 10 and 30 minutes and were used to discuss the status quo, to identify problems, to check on milestones, and to ensure continuous progress.

6.4 Working hours

[2]Epping, *Kanban für die Softwareentwicklung*.

Name	**Working hours**
Antonia Oberhauser-Tomasova	177
Niklas Wieser	179
Florian Zachs	211

Table 6.1: Working hours per person

Chapter 7

Technologies used

Authors: Niklas Wieser & Florian Zachs

This chapter focuses on existing technologies and systems used throughout this diploma thesis. There is no point in developing everything from scratch when others already did. Furthermore, by utilising existing technologies, everyone can focus on the actual workload, which maximises the chances of a successful project.

7.1 ODrive - High performance motor control

Author: Niklas Wieser

Any robotic arm needs a drive system. So the decision fell towards ODrive - High performance motor control as it enables accurately driving brushless motors for a reasonable price.

Stepper motors are ubiquitous in hobby robotics projects. They are likely to be used when working on minor robotics or automation projects today. Many DIY projects use them from 3D printers, CNC mills, pen plotters, and various other solutions. However, servo motors have taken over the industry.[1] There are some very versatile and inexpensive motors available at hobby shops, which are often used in hobby projects such as RC cars, planes or drones. They are very cost-effective, but unfortunately, there is usually no controller available for low-speed position control. The situation changed in 2017 with the rise of ODrive. It is a motor controller that can be used with any cheap hobby BLDC (brushless direct current) motor and a high-quality encoder. It turns it into a cheap, versatile, and capable servo drive.

The ODrive motor control board can handle almost any three-phase brushless DC motor

[1] Oskar Weigl, *Hobby Motors For Robotics*.

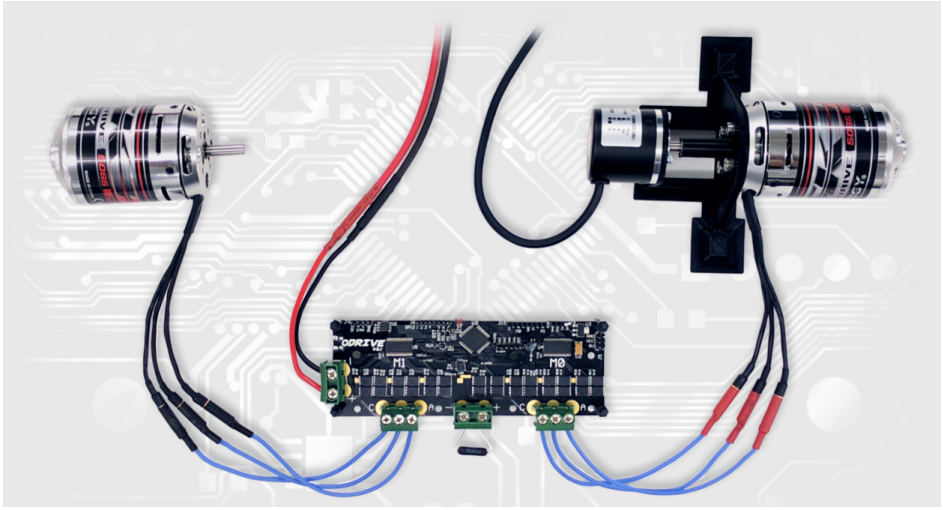


Figure 7.1: The ODrive control-board is the perfect option for hobby projects, which require high precision motor controlling. [3]

(BLDC). The motor parameters such as the coil resistances get configurated automatically. One ODrive board can operate two completely separate axes. The control loop is closed if a rotary encoder is connected and the system corresponds to a fully-fledged servo drive. The motors are free to be operated with a peak power output of over 2000 watts with proper cooling.

Another point worth mentioning is that the whole ODrive project is open source. The source code is freely available on GitHub and can be modified if necessary.

7.1.1 How to wire it up?

The first step is to select the correct board variant. There is a 24V and a 56V version available. Next, an appropriate power supply or a battery with a power output between 12V and the maximum voltage of the control board variant is necessary to power the system. At least one brushless motor and an encoder need to be connected. If the board is connected to a power supply, a power resistor is also required to prevent it from pumping excessive power into the power supply to achieve the desired deceleration torque. In some cases, this could break the power supply, or the overvoltage protection of the control-board might trip, and therefore all motors would be allowed to spin freely. To prevent this from happening, the resistor needs to be capable enough.[4]

[3] *ODrive-Website.PNG (source: docs.odriverobotics.com)*
[4] *ODrive Docs Wiring.*

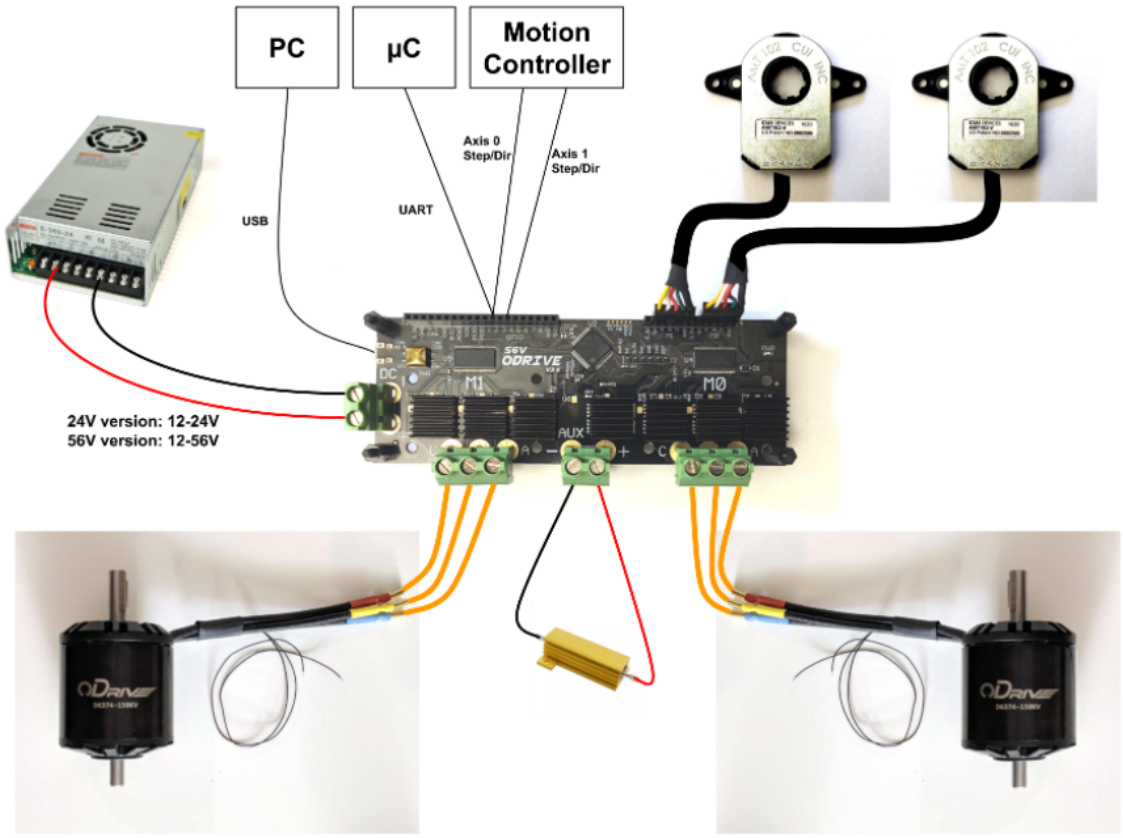


Figure 7.2: An example of correct wiring in step/dir mode. Two motors, their related encoders, a brake resistor and the 24V power supply. [6]

The three motor phases get connected to the large, three pole connector at the bottom of the control-board. Next, the corresponding encoder gets connected to the headers visible in the upper right corner of Figure 7.2.

Before powering it up, some time to check if everything is wired up correctly is appropriate. A motor could spin as soon as power is applied, so precautions are neccessary.

Unlike some other devices the control board does not receive power through the USB port. For testing firmware functionalities or taking a first look at ODrive Tool or the GUI, powering the board with 3.3V or 5V over jumper cables from another controller like an Arduino is possible. Powering a motor requires at least 12V.

[6] *ODrive Docs Wiring*

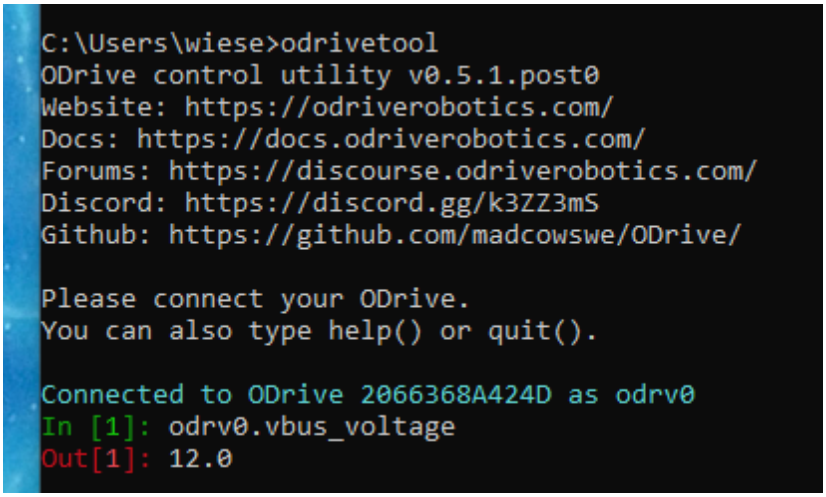


Figure 7.3: A quick demonstration of ODrive Tool. Typing "odrv0.<endpoint-path>" to read the current value from a specific endpoint.

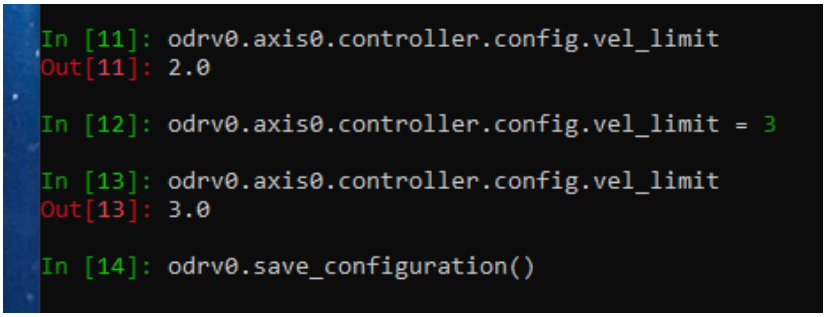


Figure 7.4: Reading and writing from an endpoint and calling a function with the ODrive Tool.

7.1.2 ODrive Tool

In the course of this diploma thesis, the term endpoint gets mentioned a lot. Within the context of ODrive, endpoints are readable and writeable properties and callable functions. More about endpoints can be read in Section 12.3.2.

The ODrive Tool is a command-line tool that provides an interactive shell to enable the user to configure the device manually. The ODrive Tool needs Python, so it needs to be installed beforehand. If Python is installed, the ODrive Tool can be installed by running the command "pip install –upgrade odrive" from the command-line. ODrive Tool is the perfect helper to explore all settings and configuration possibilities of ODrive.[7]

[7] *ODrive Docs Tool*.

Some features are:

- Any endpoint can be set with a single command.
- Automatic backup of all endpoint configurations to a JSON file.
- Automatic firmware update.
- Multiple ODrive boards can get configured simultaneously.
- A liveplotter that can be used for plotting parameters in real time.

7.1.3 ODrive GUI (beta)

The first beta version of the official ODrive GUI was released in August 2020. It uses the ODrive Tool as the backend. Therefore, the GUI inherits nearly all of the functionality from it. Several new features like a monitoring graph and an installation wizard make the GUI more user-friendly and more advanced than its command-line alternative. The application is an Electron app, which uses Vue for the frontend and accesses the installed ODrive Tool in the background for communication with the ODrive board. The GUI is still under development. It must be built from the source code, and documentation is still missing.

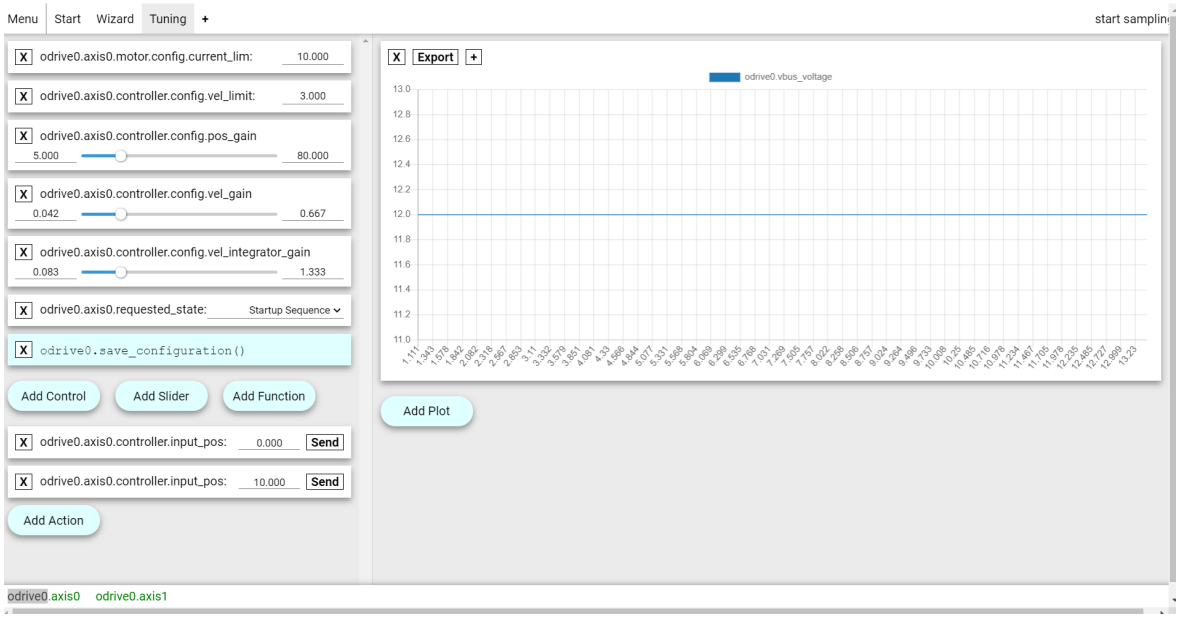


Figure 7.5: Frontend of the ODrive GUI with the list of selected endpoints on the left and a plot of specific endpoints over time on the right.

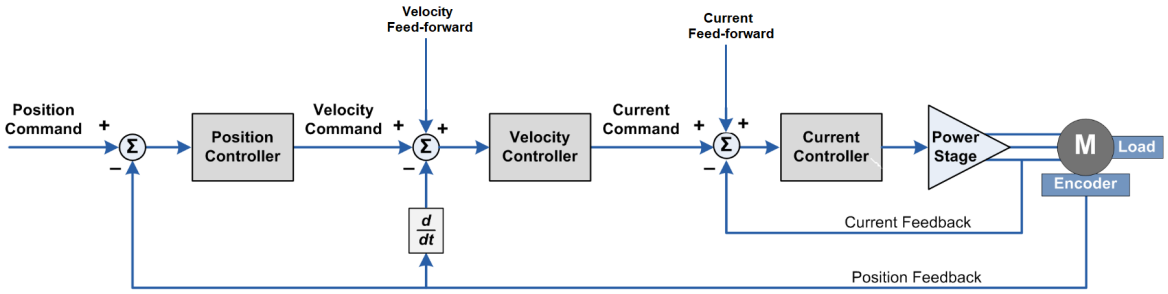


Figure 7.6: The control structure is staged into three loops. The user can specify the control mode. [10]

7.1.4 Control Structure

The control structure is designed as a cascaded position, velocity and current control loop. When running in position control mode, all three controllers are active, but when running in velocity control mode, the entire position controller on the left of Figure 7.6 is skipped. The same applies to the current control mode.

All control loops are based on a PID controller. This type of controller is a mathematical model which can be adapted to control a wide variety of systems.

In order to unlock the full potential of ODrive, tuning the controller is essential. It allows the controller to react to any interference or changes without getting unstable. Three values must be set correctly to control the motors most effectively and smoothly.[8]

- pos_gain → formerly known as k_P
- vel_gain → formerly known as k_I
- vel_integrator_gain → formerly known as k_D

All of them are located at <axis>.controller.config to tune them via odrivetool.

7.1.5 Homing and Endstops

Why are endstops even needed? The answer is quite simple. After startup, ODrive does not know how each axis is located. Therefore, it needs to move to an endstop on one side of

[8] *ODrive Control.*
[10] *ODrive-Control-Structure.PNG (source: docs.odriverobotics.com)*

the axis. Then, when the limit switch is reached, it stops and moves away by a predefined distance and sets its reference position there. This ensures that the reference position is always identical, no matter how the axis is positioned during startup. This process is called homing.

ODrive supports two endstops per axis. However, each endstop uses a GPIO pin, and there are only eight GPIO pins available per ODrive board, so not all features can be used simultaneously.[11]

In order to get the endstops to work, some configuration needs to be done.

- Set the GPIO pin number correctly with this command:

  <odrv>.<axis>.max_endstop.config.gpio_num = <1, 2, 3, 4, 5, 6, 7, 8>
  <odrv>.<axis>.min_endstop.config.gpio_num = <1, 2, 3, 4, 5, 6, 7, 8>

- Enable the endstop. If it is disabled homing will fail immediately.

  <odrv>.<axis>.max_endstop.config.enabled = <True, False>
  <odrv>.<axis>.min_endstop.config.enabled = <True, False>

- Set the offset from the endstop to the reference position. It is the number of turns from the endstop.

  <odrv>.<axis>.min_endstop.config.offset = <int>

7.1.6 Interfaces and Protocols

ODrive provides the opportunity to choose from a set of different interfaces and protocols.

ODrive supports:

- USB - Native, USB - ASCII
- UART - Native, UART - ASCII
- CAN
- RC PWM
- Analog Input

[11] *ODrive Endstops*.

ASCII Protocol

This protocol was developed to be human-readable and line-oriented. Therefore, not all parameters can be accessed via the ASCII protocol, but at least all parameters with float and integer types are supported. It is used by default on the UART interface when controlling it with an Arduino. For example, the following command can be used to set the position of a motor.[12]

Example: p 0 -2 0 0

- p for position mode
- 0 for motor number, can be 0 or 1
- -2 for the desired position, in [turns]
- 0 is the velocity feed-forward term, in [turns/s] (optional)
- 0 is the torque feed-forward term, in [Nm] (optional)

Note that the ODrive ASCII Protocol was made for simplicity and not for maximum throughput.

Native Protocol?

In order to achieve the goals and get the best out of the hardware, the ODrive Native Protocol is the way to go. Unfortunately, no official libraries implement communication between ODrive and other devices like PC or Arduino. Therefore, a custom solution is the key to success. Fortunately, relatively complete documentation is available.

The communication is packet-based and consists of two different types of packets.

- Request-Packet
- Response-Packet

When using UART, the packet gets wrapped by a stream wrapper. More about the implementation of the native protocol in chapter 12.

[12] *ODrive-ASCII-Prptocol*.

7.2 Arduino Mega 2560

Arduino is a family of microcontroller boards[14]. These boards have the same structure as PCs. Arduino is open-source and can be used for various use cases. One of the biggest advantages of Arduino is that it uses a simplified version of C and that it is easy to program, erase and reprogram with a USB cable at any time. After programming, an Arduino works while connected to a PC to debug or operate standalone.[15] These boards are extremely popular, and many people use them for their electronic projects. Compared to the ODrive boards (180€ per board), the Arduino is also relatively inexpensive as it retails around 35€, which helps keep the costs low.

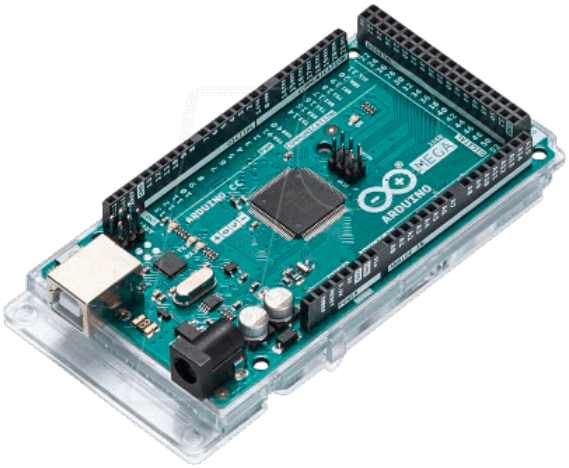


Figure 7.7: Arduino Mega 2560 [13]

The Arduino is the perfect choice for this project, as it builds the bridge between the ODrive boards and the central controller like a PC or a Festo Robotino®.

7.2.1 Ethernet Module

The communication between the Arduino and PC runs over UDP for several reasons, which are explained in section 13.6. Unfortunately the Arduino Mega 2560 which was choosen for this project has no Ethernet port by default. Therefore, an external Ethernet module is being used. The USR-ES1 W5500 is connected to the Arduino via SPI[16].

[13] *Arduino-Mega-2560.PNG (source: reichelt.at)*
[14] Schubert et al., "Using Arduino microcontroller boards to measure response latencies".
[15] Badamasi, "The working principle of an Arduino".
[16] Serial Peripheral Interface, a widely used interface for communication between a microcontroller and a peripheral device
[17] *Ethernet-Module.PNG (source: amazon.de)*

Figure 7.8: The USR-ES1 W5500 Ethernet module for Arduino using SPI [17]

7.3 Festo Robotino®

A Festo Robotino® is a mobile robot plattform with three omnidircetional drive units. The DC motors with optical shaft encoders enable the Robotino to reach speeds up to 10km/h.

7.3.1 Technical specification

The Robotino is a mobile battery-powered computer equipped with a dual-core Intel Atom CPU and 32GB of disk space. This enables the Robotino to use standard operating systems like any Linux distribution or even Microsoft Windows.[18] Interfaces like USB and Ethernet allow communication with other devices.

The Robotino has a diameter of 37cm and a height of 21cm. The total weight is around 20kg.[19]

By default, there is a specialized Linux operating system installed. It is divided into a regular Linux layer which provides standard user space and a RTLinux[20] layer. Also, the real time control can be accessed remotely from another PC using network communication. Festo provides an API which offers functions for accessing sensors and actuators.

[18] *Robotino OS*.
[19] Festo-Didactic, *Robotino - Mobile robot plattform for research and testing*.
[20] Real Time Linux
[21] *robotino-example.PNG (source: festo-didactic.com)*
[22] *festoRobotino.PNG (source: festo-didactic.com)*

Figure 7.9: The Robotino with a forklift. [21]

Figure 7.10: This image shows a number of interfaces the Festo Robotino® provides. [22]

7.3.2 Why the Robotino was chosen

The robot arm can be mounted on a mobile robotics platform, which gives the system another three degrees of freedom. The essential advantage is that there is no need for an external device to operate the robot arm, as the Robotino's internal PC can be used to control the FLAME robotic arm.

7.4 Creo Parametric

Author: Florian Zachs

7.4.1 What is Creo Parametric?

When designing a mechanical assembly with many moving parts, parametric modeling software is a must. This is where Creo Parametric comes into play.

Figure 7.11: Creo Parametric logo[23]

Creo is a parametric modeling software. Its main strength is the 3D modeling of complex mechanical assemblies, especially when many parts must interact with each other. Even assemblies with thousands of parts are no problem due to clean structuring with sub-assemblies. On the other hand, it also has a lot of features for mechanical drawings, generative design, simulation of mechanical stress (FEM), welding, piping, wiring, injection molding, animations, raytraced renderings and many more.

Creo is developed by the US-American developer studio PTC. They are also known for MathCAD, a calculation software which is widely used at the HTL Wiener Neustadt. For this diploma project, Creo Parametric 7 was chosen to be used with an educational license provided by the HTL Wiener Neustadt.

7.4.2 But what is parametric modeling?

Creo being parametric means that the entire modeling history is captured in a timeline and every feature or measurement can be parameterized and changed at any time. Figure 7.12 shows an example assembly in Creo Parametric.

The model tree can be seen on the left side of the screenshot, all features of the component are

[23] *PTC-Creo-Logo.PNG (source: Wikipedia)*

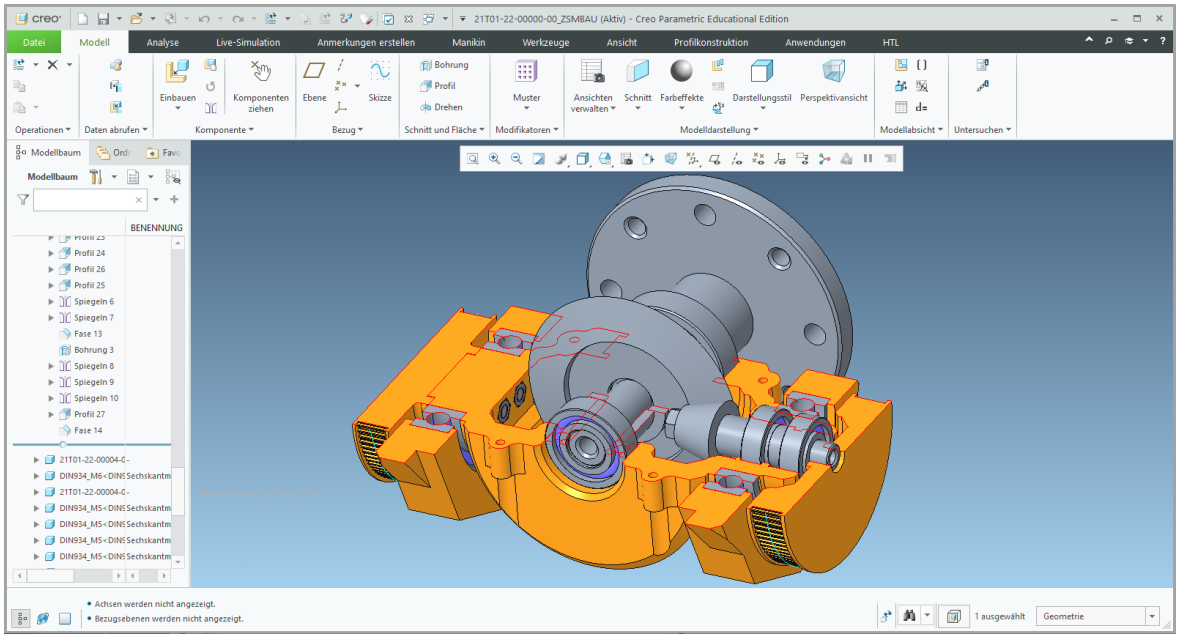


Figure 7.12: Example of an assembly in PTC's Creo Parametric 7

listed in hierarchical order. A feature can be any operation on the 3D model like extruding a two-dimensional sketch into three dimensions, cutting a pocket, making a chamfer or fillet, boolean operations between multiple bodies or even mirroring other features.

In the assembly workbench other components can be imported. The process of importing another component can also be thought of as adding a feature. This results in a clean structure with assemblies and an unlimited number of nested sub-assemblies, consisting of basic components. When components are built in into an assembly constraints can be assigned, which either lock them in place or allow them to move in certain directions. This way even movable mechanical assemblies can be fully simulated and animated. The possibilities are unlimited.

The crucial aspect of a parametric modeling software is that anything ever done to the component is stored in the feature definitions, listed in the model tree. Any change to the feature definitions simply updates them and automatically regenerates the 3D model.

Another important point is parameterization. Any feature can be defined by parameters which can then be set globally. This way a single parameter, which could describe a crucial common characteristic of the entire assembly, could be set and all parts could then be defined with a formula using this parameter. Changing the global parameter would then automatically update all parts.

Yet another way to achieve parameterization is by using skeletal sketches. This is especially useful for very large projects where hundreds or thousands of parts must interact with each

other. A skeletal sketch is a component which defines the most important dimensions with multiple two-dimensional sketches. It does the same job as a global parameter, but it can hold a lot more information. All other components can then reference key elements of this skeletal sketch, which results in a project which is thoroughly based on references. Changing the skeletal sketch would automatically update all other parts. No measurements are defined multiple times, only once in the master skeleton and all other components re-use them.

This method of parameterization is used heavily throughout this diploma thesis. A main skeletal sketch is available, which is referenced by some sub-skeletal sketches. They inherit the main skeletal sketch and add more sketches which are relevant for the specific assembly. Many crucial dimensions are defined in the skeletal sketches such as the length of the aluminium extrusions, the bearing size or the belt length. Any of these can be changed and the entire assembly and all parts adapt to the change.

7.4.3 Can popular software like Blender also be used?

In contrast to PTC Creo, the better-known modeling software Blender is not parametric. Blender is taken as an example for non-parametric modeling software because it is probably the most widely known nowadays. An example can be seen in Figure 7.13.

In non-parametric modeling software all operations act directly on the model and the features are not captured in a timeline. A 3D model is loaded and displayed and when a feature is applied, it manipulates the model immediately and only the new 3D model is kept in memory. Once the file is saved there is no record of the features which were applied to the model. This means that features cannot be changed once they are applied, only new features can be applied.

This type of modeling workflow is a good choice for simple 3D models where the aesthetics are prioritized. An example would be 3D models for video games, which is one of the main use cases for Blender. It would be a bad choice however for designing a complex mechanical assembly where many parts must interact with each other. When trying to design such a complex assembly in Blender it might seem to work at the beginning, but will fall apart very quickly, because as soon as a critical dimension changes due to a design change all parts must be changed. A very small change like only changing the diameter of a bolt may require up to ten or even more parts to be adapted manually. Doing this in a parametric modeling software takes only a few clicks.

For this reason Blender and similar non-parametric modeling software should stay with what they are best at and a proper parametric modeling software should be used for working on a complex mechanical assembly. Knowing Blender is not an excuse for acquiring proper software for the job.

[24] *Blender tutorial by "Imphenzia" (source: Youtube)*

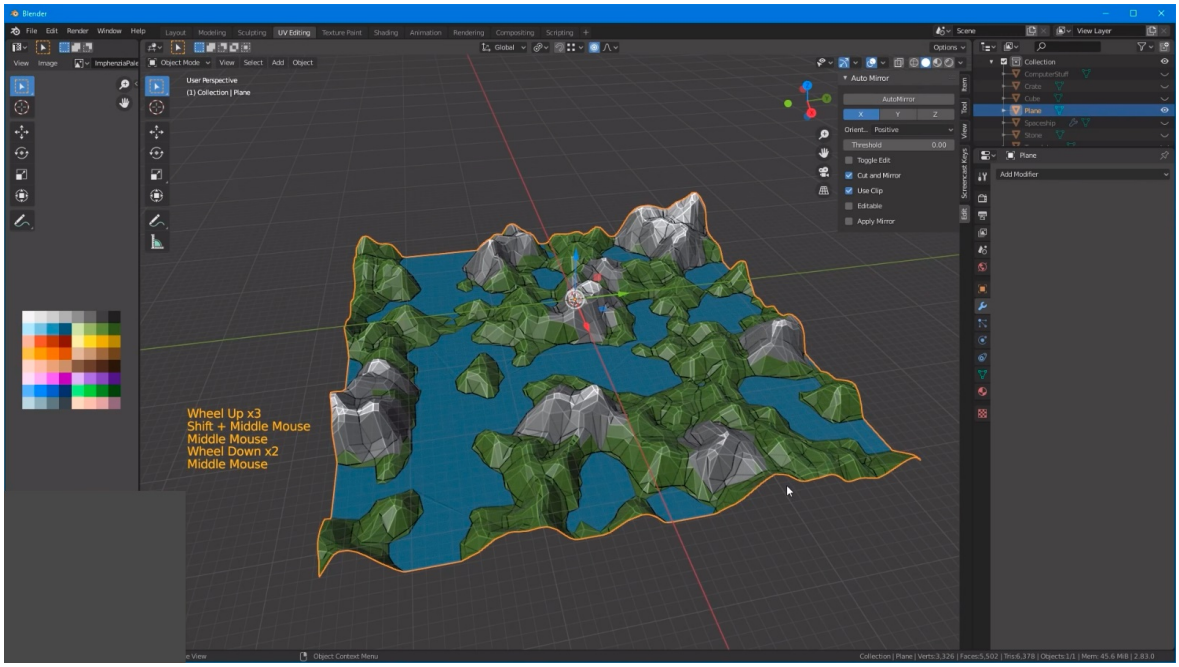


Figure 7.13: Example of Blender's user interface [24]

7.4.4 Why Creo Windchill is not used

When several engineers work on the same project it can get messy very quickly. It is Creo Windchill's mission to prevent this.

Creo Windchill is a PLM system, which stands for 'Project lifetime management'. It is used at the HTL Wiener Neustadt and in many parts of the industry. Its job is to manage various operations on a project, conducted by a large number of engineers concurrently. It is a lot more than only a version control system, however, which is why it is called 'Project lifetime management'. It controls everything, from the idea of a product, over development, manufacturing, distributing, to marketing and documentation. Everything is stored on a dedicated Windchill server, which can even be self-hosted. In the industry it would be hosted on a server belonging to the company's IT department, but for the HTL Wiener Neustadt and other educational institutions it is hosted by the HTL Mödling.

But how would Creo Windchill be used? The design suite of Creo Windchill is almost identical to that of Creo Parametric, so there is almost no visual difference while modelling. The main difference is that it does not operate in a local working directory, it rather operates in a remote server directory which all involved engineers have access to. A Creo project may consist of parts, assemblies, drawings, simulations, whatever is needed. All of those are referred to as components.

When using Creo Windchill, all components can be viewed by everyone, but any part must be

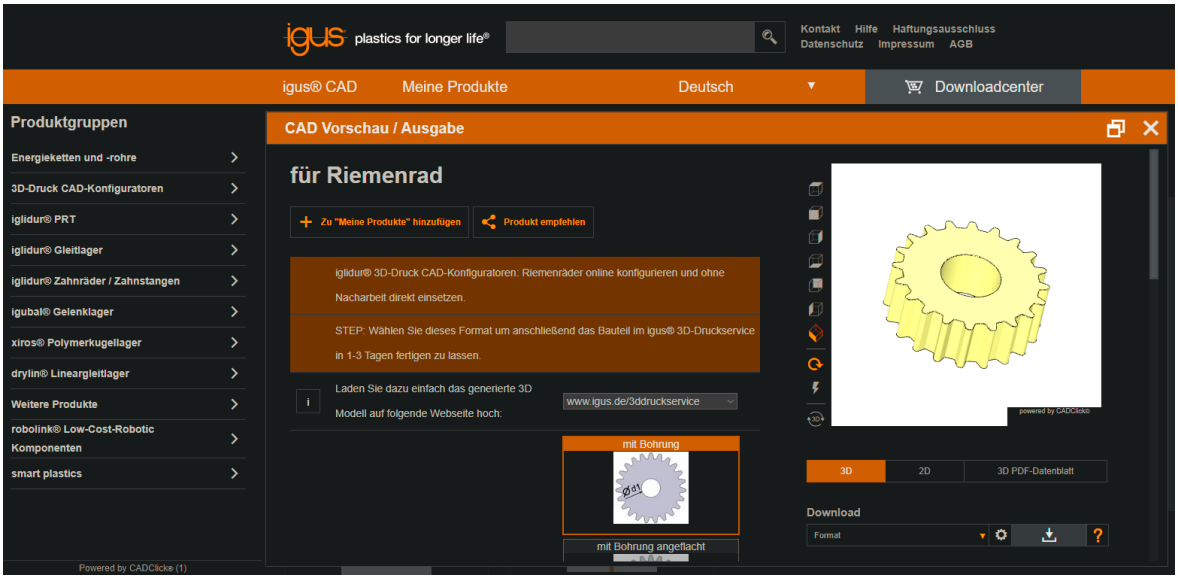


Figure 7.14: igus® online configurator for 3D prints [25]

"checked out" before modifying it. Checking out a component gives the engineer exclusive write access and prevents all other engineers from modifying it. When the engineer has finished his work, the component can be "checked in" again. At this point the access rights are returned to the server and can then be given to someone else. This workflow ensures that many engineers can work on a project at the same time and prevents them from overwriting each other's work.

The initial plan was to use Creo Windchill for collaboration in the FLAME project, but it is split into three main parts and all of the engineering work came down to one person. For this reason Creo Windchill was decided not to be used as it is not necessary and would only result in extra work.

7.5 igus®-CAD Online Configurator

The FLAME project requires 3D printed timing belt pulleys. For this job the online generator from igus® can be used as seen in Figure 7.14.

This online generator is available for free and mainly has features for 3D printing. It can be used to generate 3D models for timing belt pulleys, gears, linear bearings, gear racks and many more.

igus®-CAD was used for generating the contour of the 3D printed timing belt pulleys. The

[25] *igus online configurator for 3D printing*

belt's parameters were entered in the online form and a 3D model of a timing belt pulley was generated. However, the 3D model was not downloaded, instead a side view of it was exported as a DXF file, which was then imported into Creo.

Chapter 8

Mechanical and electrical design decisions

Author: Florian Zachs

8.1 The initial idea

The target of this diploma thesis is to plan, design, and manufacture a robotic arm that can be mounted on a mobile robotics platform. The mobile platform was chosen to be the Festo Robotino®.

The end product is designed to be used in various robotics competitions, which would enable robo4you - the initiator of this project - to take part in more significant competitions than what was possible in the past. More challenging competitions often require complex manipulators on mobile platforms which are very expensive. The goal of this project is to develop a multi-axis robotic arm for a comparatively low price.

This diploma project does not include the development of a manipulator, the scope of this project is limited to the robotic arm itself instead. The end goal of the FLAME project is a multi-axis robotic arm, which can be mounted on the Festo Robotino® and can be controlled easily and without much knowledge. The end effector of the arm is planned to be a universal adapter plate. This enables future users to attach any manipulator, only 3D printing knowledge is needed.

8.2 Drive system

To make such a robotic arm move, actuators are required for every axis. There are many different possibilities for controlling a robotic axis, but what the best option is depends entirely on the application. Some of the best options are outlined in the following sections.

8.2.1 Pneumatic actuators?

Pneumatic cylinders are a very simple way to get things moving. They are fast, simple, easy to control, and have a very long lifetime with little maintenance: The perfect solution for a system where the robot must only move back and forth in two positions and always does the same task.

The goal of the FLAME project is to create a flexible, modular, and scalable robotic arm, which does not really coincide with the benefits of pneumatic actuators. Since the position of the robot arm should be fully programmable by the end-user, the actuators must be able to move to any specific position within its range of motion.

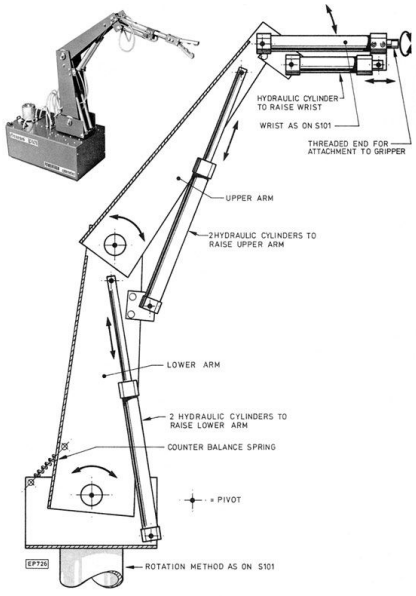


Figure 8.1:
Example of a robotic arm with linear pneumatic actuators[2]

This would be possible by using some kind of feedback for the current length of the cylinder and adjusting the air pressure accordingly, but this is nowhere precise enough if fast motion and accuracy in the millimeter range is required. This is mainly because it is powered by pressurized air, which is compressible. Additionally, due to the geometry of the joints their movement is limited to well below 180°.

As a result, the idea to use pneumatic cylinders as actuators was discarded.

8.2.2 What about hydraulics?

Hydraulic cylinders can also be used to move a robot and are very similar to pneumatic cylinders. The main difference is that the piston is moved by incompressible oil instead of pressurized air. This has many advantages and disadvantages.

[2] *Pneumatic-Robot-Arm.JPG (source: www.teamcassracing.top)*

The main advantages of hydraulics are that they can apply enormous forces and are very predictable. Therefore hydraulics are often used in industry when very high forces are required, for example for lifting workpieces or compressing wastage. Other advantages are that they can operate in hazardous environments and are very resistant to dirt.

However, a hydraulic pump is needed, a strong motor, a hydraulic oil reservoir, valves, cylinders and plumbings all over the place. Since this project is being developed on top of a mobile robotics platform and is battery-powered this would be unnecessary overhead.

Hydraulics can actually be used for accurate positioning, but they are comparatively slow and can only be used for static positioning: moving to a target position. They are not well-suited for controlling position and speed dynamically.

This makes it inapplicable for the FLAME project.

8.2.3 Why electric motors were chosen

Most actuators in robotics are electric. The main reason for that is that they are very flexible. Depending on the application they can be very large, powerful and fast, but on the other hand, they can also be very small and precise. Depending on the control electronics, the position and speed can be controlled very accurately in real-time. This makes it the perfect candidate for this diploma project.

Another important question to answer is how the actuator should be connected to the joint. It can either be connected directly via a transmission or with linkages and a spindle. Figure 8.2 shows an example. The electric motor is attached to a timing belt which drives a ball screw.

Figure 8.2: An electric motor with a ball screw[3]

In this case, the spindle would behave like an electric cylinder and would be similar to the pneumatic and hydraulic actuators discussed above. The benefit would be that only a very small transmission ratio is needed and the joint would be very strong. Additionally, depending on the spindle type the joint would not be back-drivable, which means that the robot would not collapse

[3] *Robot-Dog.PNG (source: James Bruton, YouTube)*

in a power loss.

Both benefits would be desirable for this project but due to the vastly increased range of motion and flexibility the decision was made not to use linkages to drive the joint. Instead, a transmission is used to achieve a large enough reduction ratio.

8.2.4 Stepper motors

There are two main competitors when it comes to position-controlled electric drives: Stepper- and servo drives.

They work on entirely different principles. A stepper motor is a special type of electric motor with 4 wires. In contrast to a conventional DC motor, it does not rotate continuously on its own.

Figure 8.3:
A standard NEMA 17 stepper motor[5]

A stepper motor usually has 4 wires, but it can also have more. When the correct combination of phases is powered, the rotor moves to a specific position and when the combination of powered phases is changed it moves to the next position: A so-called "step". By continuously switching the poles in the correct order continuous rotation can be achieved.

As a result, a stepper motor needs more complex drive electronics than a conventional DC motor, but it can be positioned precisely by sending a defined number of pulses. This makes it easy to position. A standard stepper motor usually has a resolution of 200 steps per revolution, higher resolutions can be achieved with micro-stepping.

The advantages of stepper motors are that they are easy to use, do not need to be configured, and have a very high stall torque. The main disadvantage, however, is that there is no position feedback. When the motor is overloaded it loses steps and is no longer at the position the controller thinks it is. There is no way to detect that, the controller must rely on the motor being where it was told to go.

This is not a problem for small, light-duty coordinate machines where the motors are not very likely to be overloaded. In the case of a large robotic arm which is supposed to lift heavy weights, this can be catastrophic and needs to be prevented.

Since the FLAME robot is very powerful and must operate in unknown environments without injuring people the position of the manipulator must always be known under all circumstances. This already eliminates the use of stepper drives.

[5] *NEMA17-StepperMotor.PNG*

8.2.5 Servo drives

Servo drives work on a completely different principle. The definition of a servo drive is that it operates in a closed loop. This means that the shaft position is constantly monitored and the motor power is corrected such that the difference between the actual and desired position converges to zero. The difference is that the actual and desired positions are all known at any time. This control method can not only be used for controlling the angle of a shaft, but also the speed and torque can be controlled easily. In theory, any physical quantity can be controlled in such a way.

The most important distinguishing feature of a servo drive is the position feedback. In theory, every type of electric motor can be used as a servo drive when pairing it with a position sensor and capable control electronics.

The drive electronics usually feature something similar to a PID-controller, which is a mathematical model of a control loop. Its task is to follow the deviation between the actual and desired position over time and provide the correct amount of motor power at any time. By tuning the parameters the servo's behavior can be adjusted to follow the setpoint as good as possible.

For many of the reasons outlined above, servo drives were chosen to be used in the FLAME project.

8.2.6 What servo drive to use

There are many different options out there when it comes to servo drives. The most noticable factor is the price point. There would be the option of using an industrial servo motor with a dedicated servo drive featuring an RS-232 port. This is what has been used in industry for decades, but it is not very flexible when it comes to implementing it in a completely self-made mobile robotic arm. This option was discarded because it is too expensive and not flexible enough.

The chosen drive system for this diploma project is the ODrive by ODrive Robotics Inc. The ODrive is a motion controller which can control two separate brushless DC motors with incremental encoders. The entire project is open source and the firmware of the controller can be modified as needed, it is released under the MIT license. The ODrive motor controller is described in detail in chapter 7.1.

A list of suggested motors is available from the ODrive website. The ODrive D6374 150kV brushless DC motors were chosen because they have the lowest kV rating of all listed motors. The rating of 150kV specifies the revolutions per minute per volt. For brushless DC motors, the current is load dependent and the rotational speed is directly proportional to the applied voltage. 150kV means that the motor spins at 150rpm at 1 volt and 1500rpm at 10 volts

and so on. The chosen motor type has a very low kV rating, which means it has a very high torque at a given speed. This is absolutely desirable for a robotic arm because the robot's axes do not need to move very fast but should be very strong. The remaining torque increase is achieved by a large belt reduction, discussed in the following chapters.

8.3 Axis configuration

When planning a robotic arm, an important question to answer is how many axes are needed. This entirely depends on the application.

In the case of the FLAME project, there is not a specific requirement for the degrees of freedom, but some assumptions can be made. Since the ODrive motion controller supports two axes per board, the number of axes should ideally be a multiple of two.

The robotic arm is going to be mounted on top of the Festo Robotino®, which is a mobile platform. Usually an industrial robotic arm has at least six axes, because that enables the end effector to be placed at any location in any rotation. This is not needed in this case because the arm is mounted on the Robotino which already controls three degrees of freedom and the robot arm's degrees of freedom add on top of that. As a result, having a six degrees-of-freedom robotic arm on a movable platform would be unnecessary.

On the other hand, having only two axes would result in a robotic arm with very limited movement which would be useless in most situations without a complex end effector.

For this reason, the number of axes was decided to be four. This gives the robotic arm enough degrees of freedom to be very useful in various situations when paired with the Festo Robotino®. On the other hand, it is also not unnecessarily complex, the ODrive boards are used to the best of their abilities and they are also not too difficult to control alongside each other.

8.3.1 How the joints are located

Figure 8.4 shows an Arduino Tinkerkit robot, which is a great example of the axis configuration desired for the FLAME project.

The robot in the image has five joints. The first joint is located below the robot, it rotates the entire assembly left and right. Then there are three joints in one plane and the fifth axis is the servo motor at the wrist, rotating the white manipulator. This is exactly the desired axis configuration, but without the first axis. The robot does not need to rotate left and right because it is going to be mounted on the Festo Robotino®, which can already turn on

[7] *Arduino-Tinkerkit-Robot.PNG*

Figure 8.4:
Arduino Tinkerkit Braccio-Robot as an example for the desired axis configuration[7]

its own.

8.4 Connecting the motors to the arms

Many important questions are already answered: ODrive servo motors are used, which are electric rotatory actuators, four of them to be precise. But how should the motors be connected to the arms?

Such an electric servo cannot be connected directly to the arm, mainly because it can not handle the enormous torque. A large transmission ratio is needed to increase the motor's torque enough that it can hold and move an entire arm. The needed transmission ratio is dependent on the expected load and the stall torque of the motor.

The motors chosen for this project are the ODrive D6374 150kV brushless DC motors. At peak power, they have a stall torque of around 3.85Nm, this is the torque at zero speed, when it is completely blocked. The stall torque was even confirmed to be accurate with an experiment. Due to it being a brushless DC motor, the torque is almost constant for all speeds.

For the first axis, a reduction ratio of around 1:25 was decided to be appropriate. As a result, the first arm can support a maximum torque of around 100Nm. The first axis has the greatest transmission ratio, the second, third and fourth axes have smaller and smaller reduction ratios, as they do not need as much torque.

To sum it up, the goal is to find a way to connect the motors to the arms with a reduction ratio of up to 1:25.

There are countless ways to achieve such reduction ratios, a few of them are outlined below.

8.4.1 Conventional spur gears

A gear reduction is the most obvious way to achieve a reduction ratio, it is perfect for transmitting very high torque. The biggest drawbacks, however, are that the reduction ratio is limited and, most importantly, the backlash.

One characteristic of spur gears is that there must be a certain amount of space between the teeth and the teeth of the other gear. Without it, they would not operate correctly and jam. This gap between the teeth is called backlash. This backlash leads to the other gear not being driven for a brief moment when the direction of rotation is changed. It can be thought of having an offset in one direction. This is not a problem for continuous transmissions like in a car but it is indeed a problem for positioning systems or robots where the position is critical.

The FLAME arm uses servo drives for precise positioning and any backlash in the drive system would compromise the achievable accuracy. While it is often not possible to achieve zero backlash, a smaller one is almost always better.

The main reason that conventional spur gears were not chosen is because the bearing configuration would be rather complex. Additionally, due to the size restriction, a two-stage reduction ratio of 1:5 each would have been needed to achieve the 1:25 overall. This would result in large and expensive gears, which is why conventional spur gears were not chosen.

8.4.2 Worm drives

A worm drive is a very special form of a gear reduction.

A worm drive consists of a worm gear and a worm screw. The worm screw can be thought of as a single or multi-start screw with a very course thread. The worm gear, on the other hand, is formed similar to a conventional spur gear, but the teeth are formed to mesh with the round worm screw.

Figure 8.5:
Example of a worm drive, without housing or bearings

Worm drives can achieve very high reduction ratios in a single stage and they can also take up very high torque. Additionally, they are not back-drivable. This means that force is transmitted from the input shaft to the output shaft and not the other way around. In other words, the output shaft is always locked in place, even when the input shaft is not held in place. This may be an advantage or disadvantage based on the application, but for a robotic arm, being not back-drivable is certainly an advantage. This would mean that all axes would always be locked in place, even at a power loss and the motors do not need holding torque when the robot is not moving.

Worm drives are not the best when it comes to backlash. Compared to other solutions they exhibit quite a lot of backlash and the entire gear box must be quite complex for the backlash to be adjustable.

Another disadvantage of worm drives is their incredibly low efficiency, as they dissipate a lot of heat when operating due to the high friction between the worm gear and the worm screw.

The decision was made not to use worm drives because they are very expensive and designing one from scratch would require very complicated housings and bearing locations. The disadvantages outweigh the advantages by far for the FLAME project.

8.4.3 Planetary gear sets

Another solution to achieve a large reduction ratio is to use a planetary gear set.

A planetary gear set works by having a ring gear on the outside, a sun gear in the middle and three or more planet gears inbetween. In Figure 8.6 the center shaft is the input shaft. When it rotates, the four planet gears rotate with it, but much slower because they are held between the sun and the fixed ring gear. All planet gears are held on the planet carrier pins attached to the planet carrier plate, not well visible in the image. The planet carrier plate is what is attached to the output shaft, coming out of the back of the gearbox.

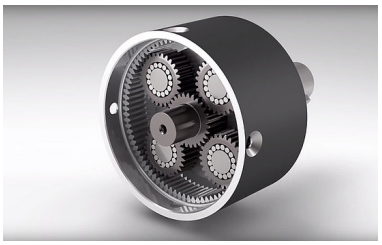


Figure 8.6:
Inner workings of a planetary gearbox

Such planetary gearboxes can achieve very high reduction ratios in a single stage, are very compact and durable and can easily be staged together to create a multi-stage reduction. These multi-stage planetary gear sets sound very complicated but are actually very straightforward to manufacture.

So straight-forward, in fact, that they are manufactured and used in all sizes, from miniature plastic DC motors up to cars, tractor wheel hubs, industrial robots, cranes, and even in aerospace applications. Who would have thought that even a standard LEGO® M-motor shown in Figure 8.7, the simplest, most fool-proof motor to think of, features a two-stage planetary gear reduction?

To sum it up, planetary gearboxes are a very simple, compact and elegant solution to achieve very high reduction ratios. Industrial planetary gearboxes are available pre-assembled as seen in Figure 8.8.

Unfortunately such industrial gearboxes are very expensive. As nice as it would be to use planetary gearboxes in the FLAME project, the cost is just too high.

Figure 8.8:
Example of an industrial planetary gearbox

8.4.4 Harmonic drives

Yet another way to achieve high transmission ratios is to use a so-called harmonic drive or strain wave gearing. This is a very special form of a transmission, similar to planetary gear sets. The working principle is fundamentally different though.

Figure 8.9 shows the working principle of a harmonic drive. It consists of a ring gear (blue),

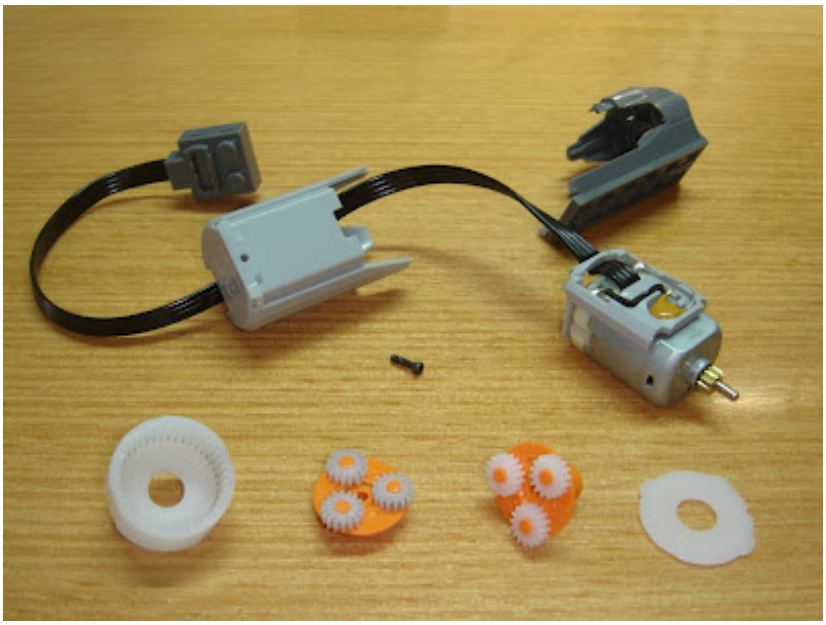


Figure 8.7: A standard LEGO® M-motor disassembled

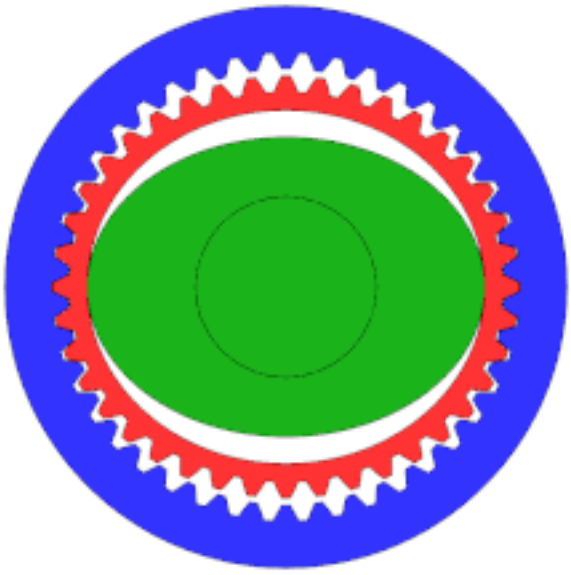


Figure 8.9: Working principle of a harmonic drive or strain wave gearing

Figure 8.10:
Example of an industrial harmonic drive or strain wave gearing. The ball bearing is slightly elliptical and deforms the spline gear so that the little teeth mesh on two opposing sides.

a flexible spline (red) and a wave generator (green). The ring gear is part of a sturdy housing of the entire gearbox. The flexible spline, on the other hand, is usually a bell-shaped metal gear with very thin walls. It is thin enough that it can be deformed without breaking. The wave generator (green) is the input shaft in this case. It rotates and continuously deforms the flexible spline elliptically. The spline has exactly two teeth less than the ring gear. Therefore, for every half-rotation of the input shaft, the spline moves by exactly one tooth, which is connected directly to the output shaft.

In practice the wave generator (green) is a slightly elliptical shape with a thin-walled bearing on it. The bearing is also deformed slightly, depending on the size of the teeth.

Harmonic drives are a fantastic way to achieve enormous reduction ratios in a single stage, even up to 1:500 would be no problem. Figure 8.10 shows a real example.

Such harmonic drives would be the ultimate solution for the FLAME robot, but unfortunately such industrial solutions are very expensive, probably the most expensive ones on this entire list of drive systems.

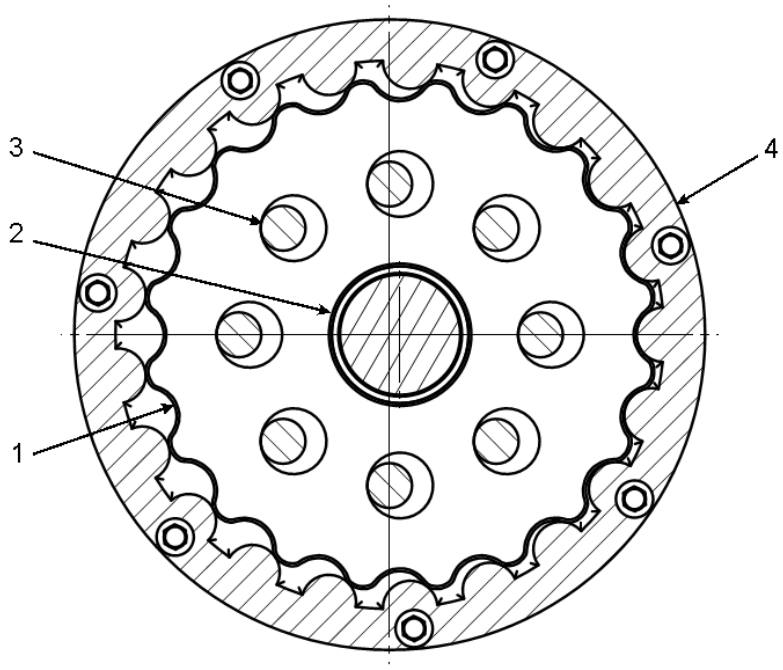


Figure 8.11: Working principle of a cycloidal drive

8.4.5 Cycloidal drives

The last type of inapplicable transmissions is a cycloidal gearbox. This is a type of gearbox very similar to a harmonic drive, shown in Figure 8.11.

The input shaft in the center has an eccentric bearing (2). The ring gear (4) is fixed and the inner gear (1) is free to move, mounted on the eccentric bearing and held by the drive pins (3). The main gear has exactly one tooth less than the ring gear. The eccentric gear fits perfectly into the ring gear and has contact on every tooth at the same time.

The eccentric input shaft in the center causes the gear to "wobble", which has the result that the gear rotates by exactly one tooth for every rotation of the input shaft. The rotation is picked up by the drive pins (3) which are connected to the output shaft. This results in a very high reduction ratio. Despite the high reduction ratio, it can even be back-drivable if the bearings are adequate. For this reason this type of transmission is very popular in robotics, because it is a very compact and reliable solution and even allows torque-control.

A cycloidal drive would be the ultimate transmission for a robotic arm besides a harmonic drive, but unfortunately they are just as expensive as harmonic drives, which is the reason they were not chosen for the FLAME project.

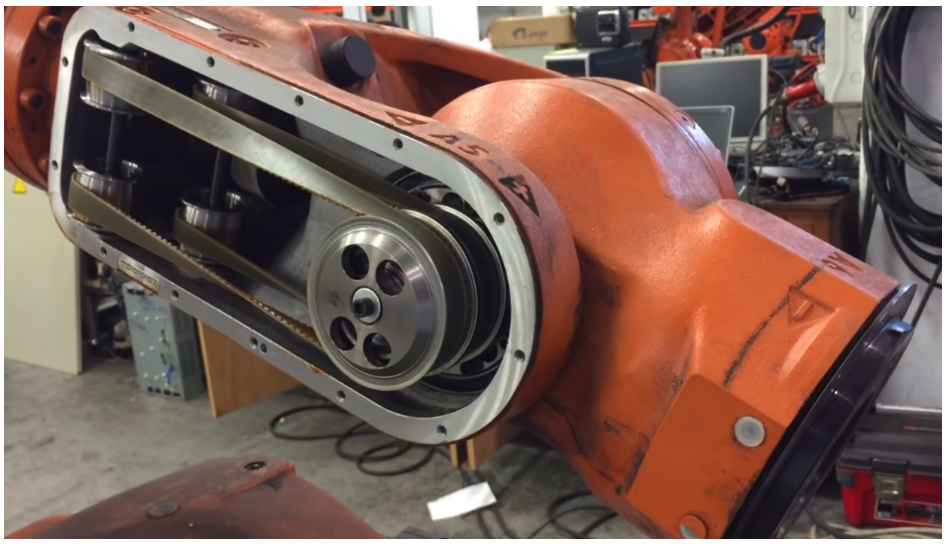


Figure 8.12: KUKA robot with timing belts[8]

8.4.6 Why timing belts were chosen

But after such an extensive list of failed candidates, what drive system is adequate for the FLAME robotic arm? Well, the decision fell on conventional timing belts.

Timing belts are flat belts with teeth on the inside. What makes them special is that by using their teeth, they conform to the shape of the pulleys, which is why they do not slip under normal circumstances. In contrast, a conventional V-belt always has slippage, even when tightened properly. This arises from the belt being tensioned differently around the perimeter of the pulley when it is under load. Timing belts do not have this characteristic because the teeth mesh with the pulley, therefore a timing belt always keeps two pulleys synchronized. This is why they are called timing belts or synchronous belts.

In the case of the FLAME robot, the belt would connect the mechanical arm to the motor. The position sensor is mounted on the back of the motor, which means that the belt must keep the arm and the motor rotation synchronized. Otherwise the arm and the motor would slowly get desynchronized and there would be no way of knowing the angle of the arm without additional sensors.

For this reason timing belts are used extensively in robotics, because the position sensors or encoders of industrial servos are almost always mounted on the back of the motor itself in a compact package. Any mechanical connection attached to a position-controlled servo must not have any slippage.

Figure 8.12 shows an example of timing belts in an industrial robot by KUKA, one of the industry leaders in robotics. This robot uses a combination of timing belts and cycloidal

[8] *Example of timing belts (KUKA)*

drives. The timing belt seen in the image is the first step of reducing the speed and increasing the torque of the high-speed servo motor. At the same time it does the job of rotating the movement by 90 degrees, the servo motor is in the back of the arm, in line with the arm itself. Behind the timing belt pulleys are cycloidal drives in the wrist joint, which provide the remaining gear reduction.

Compared to all other options on this list, timing belts are actually the cheapest. Additionally, they do not require a complicated bearing location and are easily replaceable in the future.

As said previously, a reduction ratio of around 1:25 is desired, this is achieved by a two-stage reduction with 1:5 each. The large pulleys are 3D printed, while the small pulleys are sourced from Mädler[9], a German company specialized on automation and power train technologies.

[9] *Mädler*

Chapter 9

The first prototype

Author: Florian Zachs

An important part of engineering is prototyping. The first prototype of the FLAME robot arm was partially modeled in Creo Parametric, but no parts of it were ever manufactured as it was superseded by the second prototype soon after. This chapter is about the engineering problems with the first prototype and how it led to the design of the second prototype.

9.1 Overall design and end of life

Figure 9.1 shows the state of the first prototype shortly before work was discontinued and the model of the second prototype was started. Only the first and second axes were modeled partially, the third and fourth axes and the end effector plate are missing completely. The initial idea was to use aluminium extrusions as the main structure and to connect them with thick steel plates or 3D printed parts. The motor for the first axis is located at the bottom and drives the right orange gear with a timing belt, which in turn drives the left orange gear with the second timing belt. It results in a two-stage belt reduction from the motor to the arm, the reduction ratio would have been 1:25.

All orange parts were planned to be 3D printed. The two shafts visible in the image would have been supported by ball bearings, which would have been mounted between two thick steel plates. All housings are indicated by the thin blue lines, but they were never modeled completely. Figure 9.2 shows the front and side view. The lines between the gears in the right image indicate the timing belts and how they were planned to be tensioned with ball bearings. All of this was completely redone in the second prototype.

The biggest problem with this design is that there is no obvious way to place the belt pulleys for the second axis. Because a two-stage belt reduction is needed, an intermediate axis is

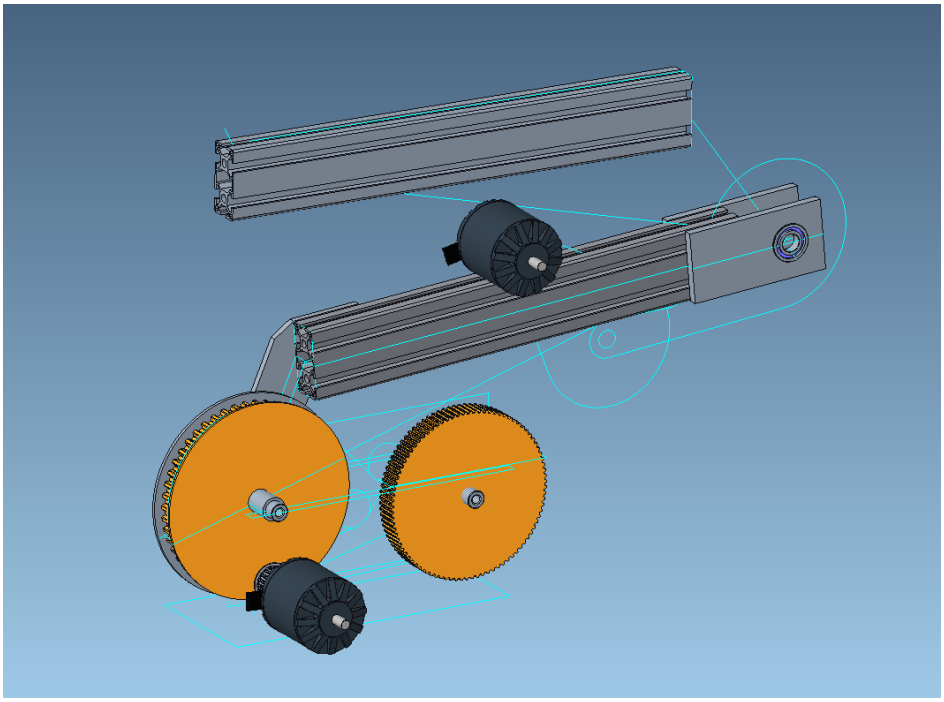


Figure 9.1: Final state of the first prototype, modeled in Creo Parametric 7

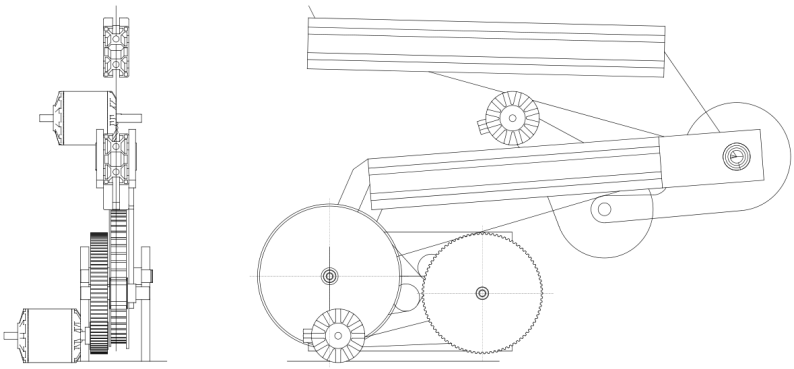


Figure 9.2: Front and side view of the first prototype

needed, which means two pulleys are needed on opposing sides of the aluminium extrusion. Additionally, the motor is quite long compared to the thickness of the extrusions. All in all it would be everything but compact and so this design was decided to be reworked from the ground up.

Chapter 10

The second prototype

Author: Florian Zachs

Designing the first prototype showed up many design problems, which were addressed in the second prototype. The second prototype was the first to be manufactured and assembled, it is the final design. Some minor parts like braces for stiffening the frame were manufactured without ever being modeled.

10.1 Skeletal sketches

The entire FLAME project consists of several sub-assemblies to make it easier to work with. Figure 10.1 shows the sub-assemblies in the model tree, seen on the left side of the image. The sub-assembly of the lower arm is selected and highlighted in red. The parts are divided in the sub-assemblies such that every sub-assembly represents one link of the arm. This means that the sub-assemblies can then be assembled again in another Creo document with dynamic constraints. This way the motion of the robot arm can be fully simulated, tested and animated all directly within Creo Parametric.

Figure 10.2 shows the master skeletal sketch. It defines the most significant dimensions of the robotic arm. The length of the aluminium extrusions can be seen, as well as the distance between the axes, the bearing diameter and the location of each bearing. It defines all dimensions where two sub-assemblies overlap.

Every sub-assembly also has a sub-skeletal sketch which inherits from the master skeletal sketch. It takes the master sketch and extends it by the critical dimensions where any two parts touch. The lower arm is taken as an example. The skeletal sketch of the lower arm sub-assembly can be seen in Figure 10.3. It takes the master skeletal sketch as a base and adds more sketches relevant for the lower arm, like the belt length, the bearing holders or

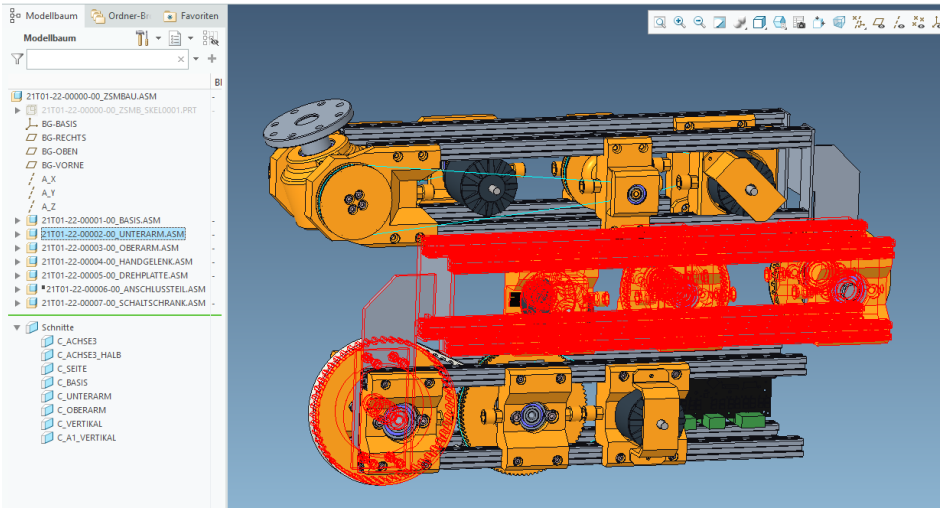


Figure 10.1: The sub-assemblies in the model tree and the final 3D model. The sub-assembly of the lower arm is selected and highlighted in red.

the shaft diameter. The lower arm's sub-assembly consists of simple parts and all of those reference the skeletal sketch. This results in a large Creo project with a tree hierarchy and everything inherits from the master skeletal sketch.

But why is this done? When designing a complex assembly, things have to change over time, this is nothing bad. But if the model is not structured correctly applying such a change can be catastrophic and might break things and require a lot of effort to get them right again. This problem is eliminated when using skeletal sketches correctly. Now, an important dimension like the distance between the bearings can be changed in the master skeletal sketch and all sub-assemblies and all parts adapt automatically, no extra work is needed. This way a lot of unnecessary work is eliminated and everything still fits together, no need to check every single part if it still fits. This is the most important reason a real parametric modeling software should be used, as discussed in chapter 7.4.

10.2 The drive system

As discussed in chapter 8, the arms are connected to the motors with timing belts. This section is about the design of the drive system and how the timing belts are implemented.

10.2.1 Timing belts

Figure 10.4 shows the drive system of the first axis in detail. The large black motor on the left has an aluminium timing belt pulley attached to it. The first timing belt is indicated by

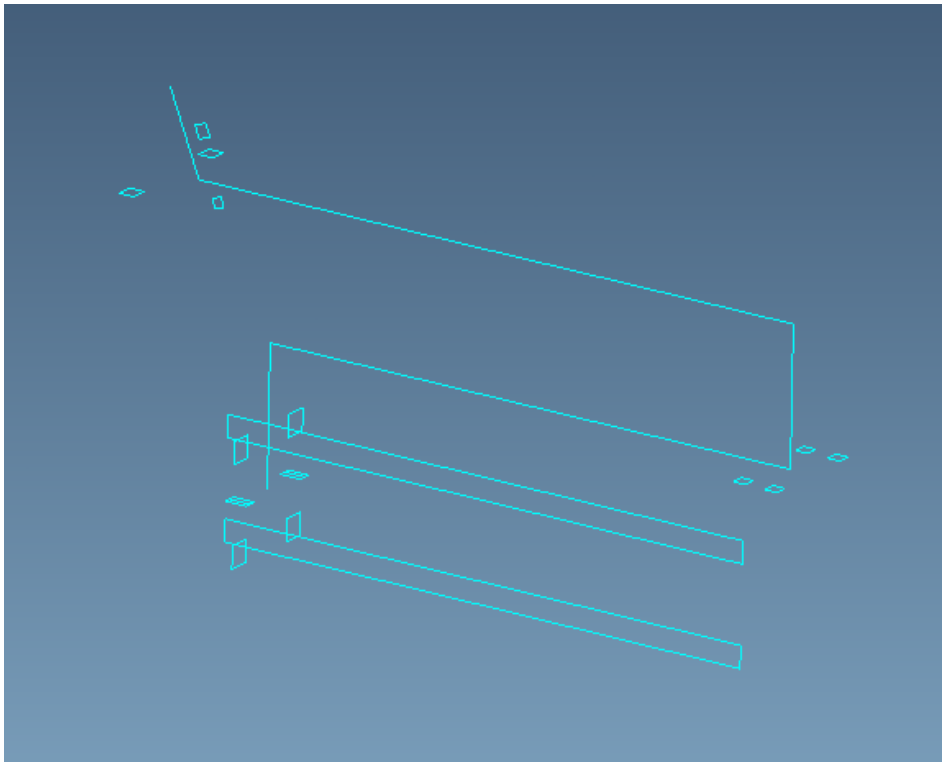


Figure 10.2: The master skeletal sketch of the robotic arm. It defines the most significant dimensions.

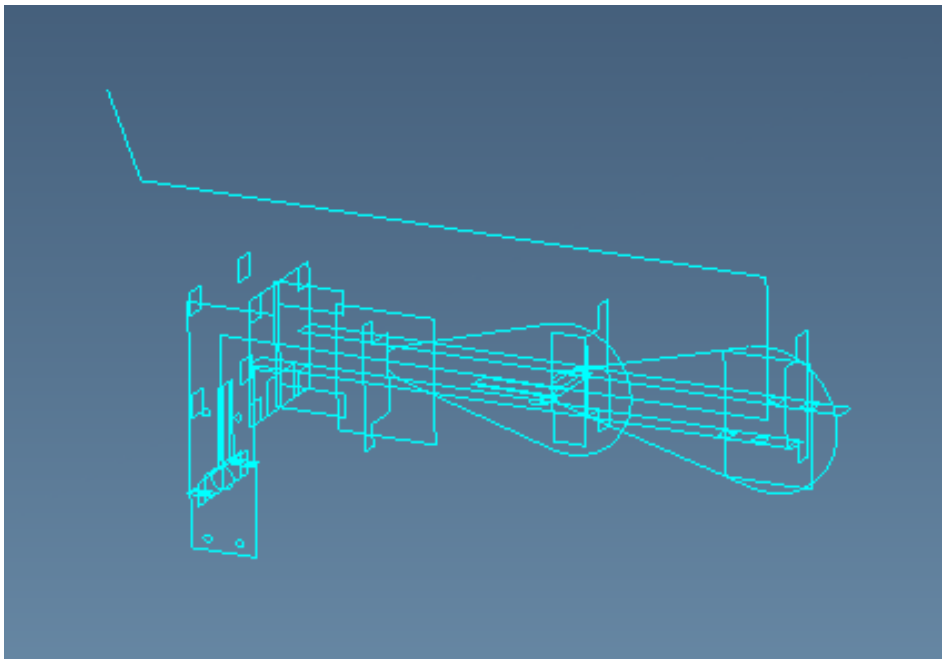


Figure 10.3: The sub-skeletal sketch of the lower arm. It references the master sketch and extends it, but only with the dimensions relevant for the lower arm sub-assembly.

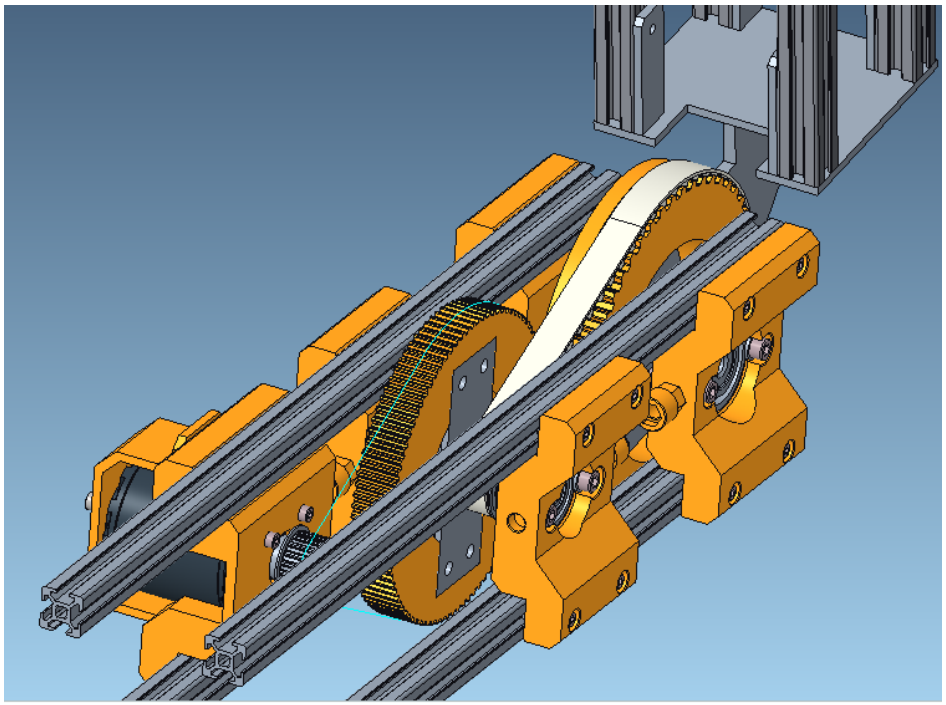


Figure 10.4: View of the drive system with timing belts

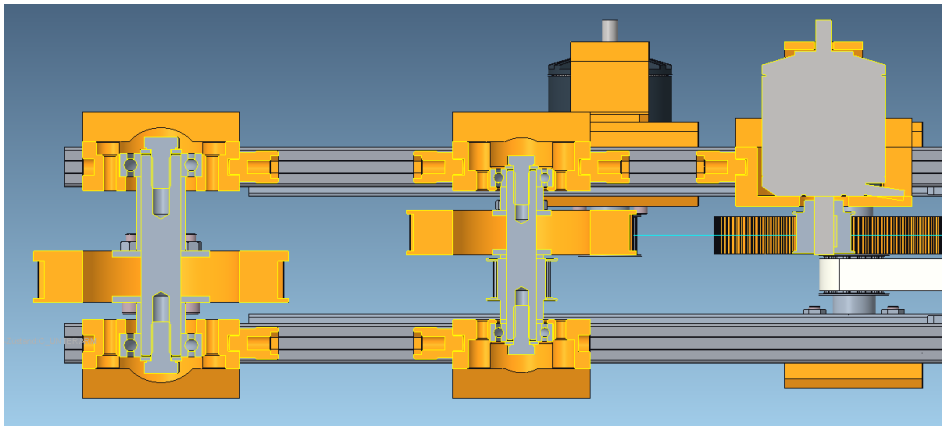


Figure 10.5: Cross-section of the second axis

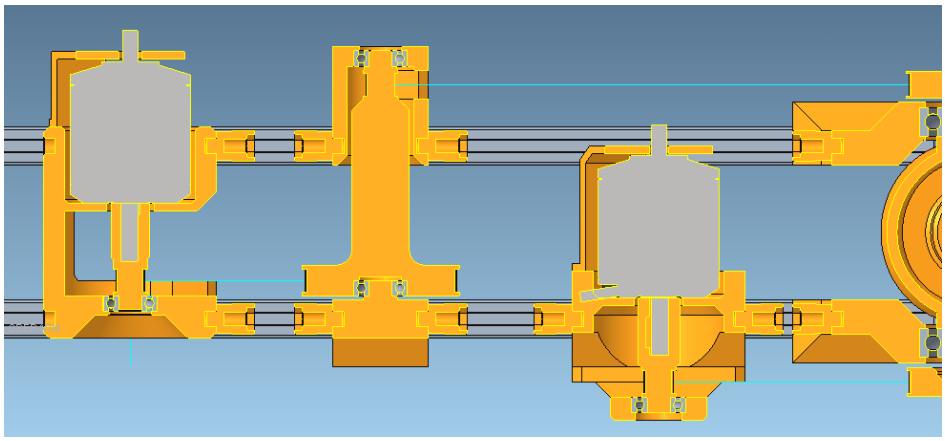


Figure 10.6: Cross-section of axis 3 and 4

the thin blue line, it was not modeled. The second timing belt is shown in white. This is how the reduction ratio of 1:25 is achieved, it is a two-stage reduction with 1:5 each.

Figure 10.5 shows the cross-section of the second axis. It shows how the 3D-printed gears are mounted. They are hollow and have steel plates on both sides. One of the two plates is welded to the shaft in the center, to allow torque transfer and increase sturdiness. All other parts on the shafts are clamped to the shaft with the two bolts on both ends of the shafts. This design is equivalent for axis 1 and 2. The belts for axis 1 and 2 are also equivalent, but the reduction ratio is only 1:16 in total, achieved by two 1:4 reductions.

Axis 3 and 4 are different, as seen in 10.6. Again, the belts are hinted by the thin blue lines. Axis 3 on the left side of the image also has a two-stage belt reduction, but in this case the entire intermediate shaft is 3D printed. The belt for axis 4 is on the bottom right, this axis only has one belt because it has additional gearing in the wrist, as discussed below.

10.3 The wrist joint

The requirement for the FLAME robotic arm was that it has a rotatable wrist. This was the hardest part of designing it. The final design of the wrist joint can be seen in Figure 10.7.

The third axis is defined such that is rotates the orange housing in the middle up and down. On the other hand, the fourth axis is the rotation of the adapter plate.

The ODrive motor is very large and is way too large to fit into the housing of the fourth axis. For this reason the motors for axis 3 and 4 both had to be mounted on the upper arm and both had to be transmitted to and into the wrist joint. Bevel gears seemed appropriate for this job. Gears were not used as the main power train system, mainly because of the backlash. It was decided that it is not a problem for the fourth axis, because it does not have a long arm on it which exaggerates the backlash, and it could even be adjusted by using shims behind the bevel gears. This way the distance between the gears can be adjusted very precisely until the backlash is acceptable.

The belt pulley on the upper left side of the image is for the third axis. When it is rotated, it rotates the entire assembly in the middle, which is held between the two large bearings on the left and right.

The belt pulley on the bottom right is for the fourth axis. It is mounted on a shaft, which drives the small bevel gear. The small bevel gear drives the large bevel gear, which is mounted on the main output shaft, which has the adapter plate attached to it.

This design results in a relatively simple way to move two interconnected axes with two stationary motors. It does have one disadvantage, however: Say the wrist must be tilted up or down. This is the movement of the third axis. If only the third motor is moved and the fourth is not, then the adapter plate will also turn, but it is not supposed to be when only the wrist is relocated. In other words, when moving the wrist up or down, the fourth motor has to do the exact same movements to keep the adapter plate in place.

This sounds complicated at first, but it is not in practice. It is as simple as adding the target position of axis 3 to the target position of axis 4 times a factor for the gear ratio. This is explained in more detail in Chapter 12.

10.4 Belt tensioning

As discussed in previous chapters, timing belts do not slip. However, that is only true as long as they are tensioned properly. For this reason, belt tensioners were installed.

This can be seen in Figure 10.8. The belt tensioners consist of M6 nuts, a threaded rod and the 3D printed connector parts. They have six flats around the perimeter and can be rotated

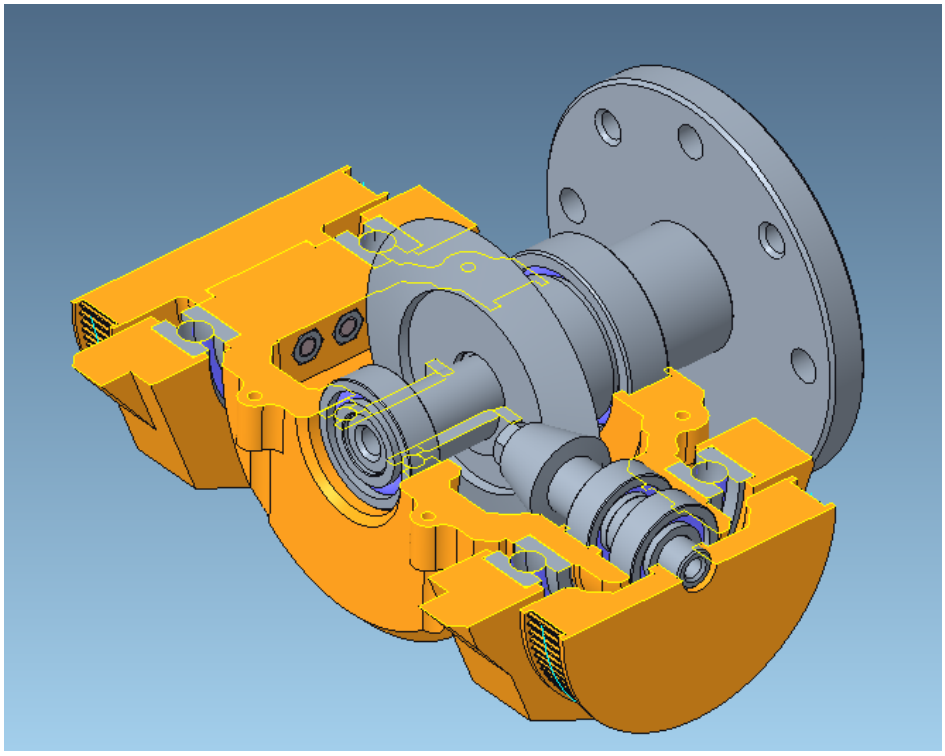


Figure 10.7: Cross-section of the wrist joint (axis 3 and 4)

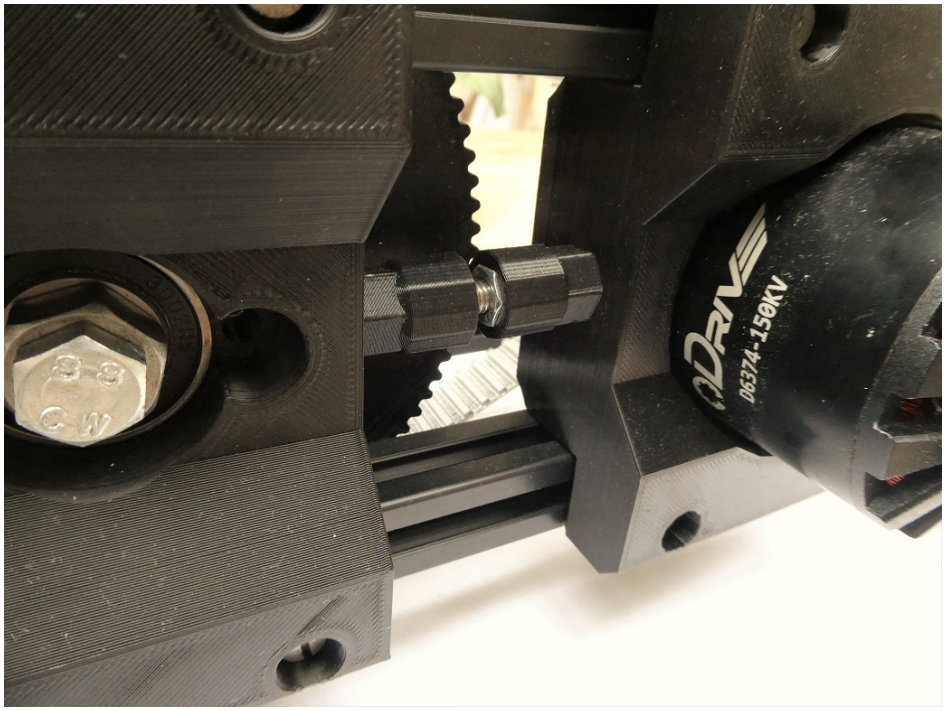


Figure 10.8: 3D printed belt tensioners

with a wrench, that way the distance between the various parts can be adjusted smoothly.

10.5 Limit switches

The ODrive motors use incremental encoders with index marks. That means that the ODrive controller always knows how much the motor has moved since it was started. However, it does not know where it is when starting up.

For this reason it needs to reference itself when starting up, which is done by utilizing limit switches. When starting up, the robot arm homes itself, which means that all axes move slowly towards their reference position until the limit switch of each axis is pressed. Then the system knows that the axis is exactly at the limit switch and moves back by a defined offset. Then position counter is reset back to zero and the system is homed. It can now start operating.

These limit switches are mounted as close to the arm's joints as possible to avoid unprecise results due to vibrations in the long arm. They are then wired to the ODrive, which does all the work of calculating homing offsets.

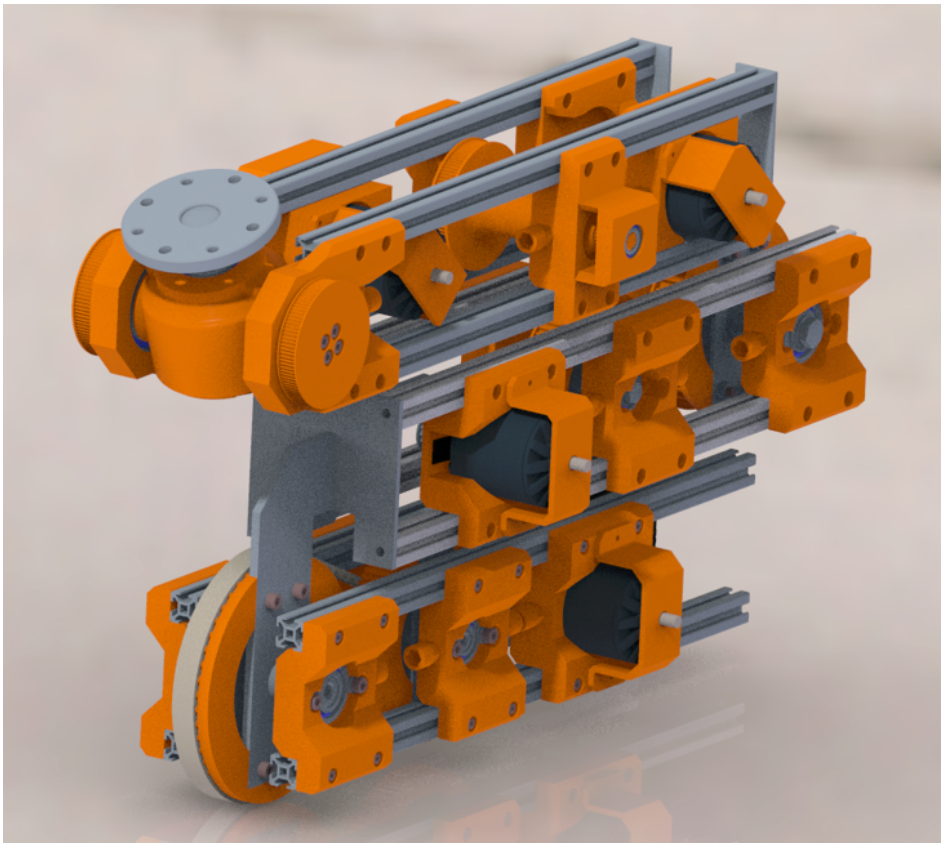


Figure 10.9: Raytraced render of the final 3D model

10.6 The final 3D model

After a lot of modeling in Creo, the 3D model was finally finished. Figure 10.9 shows the final version. The image was raytraced directly in Creo Parametric. Photos of the final assembly can be found in chapter 14.

Chapter 11

Kinematics of the robot arm

Author: Antonia Oberhauser

To achieve movement in certain directions, it is necessary to focus on the kinematics of the robot. Kinematics are a form of physics and mechanics which calculate the geometrically achievable motion of an organic or synthetic body. In other words: Kinematics can be considered as an analysis of different ways to achieve a pose of an organic or synthetic body.

11.1 TAP vs TCP

It is important to understand that there are the TAP [1] and the TCP [2] which refer to different positions on the robot. The TAP is the connecting point on the adapter plate and the TCP is the operating point from the attached tool. If the robot arm has a pen attached, for example, then the TAP would be the point, where the pen and the arm are connected and the TCP would be the tip of the pen. The kinematics from the FLAME robot arm only refers to the TAP because the kinematics need to be three-dimensional in order to get the offset to the TCP. It would be more complex to calculate with three dimensions, instead of two. However, if the third dimension would be neglected, the offset of the tool would only be half implemented and that is not the effort worth.

[1] Tool Attachment Point
[2] Tool Center Point

11.2 Different types of kinematic models

A robotic arm can be thought of as a chain of joints, while the joints can be linear or rotatory. To calculate movement of the robot arm, a kinematic model is necessary.
The models are termed forward and inverse kinematics. Usually, both are used in an industrial robotic arm. To allow calculation of the kinematic model with a variable number of joints, each joint has its own coordinate system. As a result, there are $n + 1$ local coordinate systems in the model, with n being the number of joints. Starting from the bottom of the robot arm, where it is fixed on the Festo Robotino®, to the top, where the effector is attached. There are multiple variations of kinematics. To determine which fits better to the task given, the differences are described in the subsections below.

11.2.1 Forward Kinematics

Forward Kinematics deals with the difficulty of finding the right vector function from the origin to the robot tooltip X (TAP), given a set of joint angles Q. Not only the position of the end effector needs to be determined, but also the orientation.

$$x = f(q) \tag{11.1}$$

The FLAME robot arm is supposed to move up, down, left and right. Since the arm has four axes, it is considered a 4 DOF[3] manipulator. It only moves in two dimensions because the first three axes are located at the same level, and the fourth axis is not included in the kinematics calculation. The goal is to find the right vector function for the end effector position. Given is the x and y coordinates of the end effector X which are in set of real values in two dimensions and the joint space of the robot Q, given by *q1* and *q2* in also a set of real values in two dimensions.

$$x = \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R} \tag{11.2}$$

$$q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \in \mathbb{R} \tag{11.3}$$

The forward kinematics function for a robot arm with two axes looks like this:

[3] Degree Of Freedom

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_1 cos(q_1) + l_2 cos(q_1 + q_2) \\ l_1 sin(q_1) + l_2 sin(q_1 + q_2) \end{bmatrix} \tag{11.4}$$

To complete the vector function, the orientation variable ψ needs to be included.

$$\begin{bmatrix} x \\ y \\ \psi \end{bmatrix} = \begin{bmatrix} l_1 cos(q_1) + l_2 cos(q_1 + q_2) \\ l_1 sin(q_1) + l_2 sin(q_1 + q_2) \\ q_1 + q_2 \end{bmatrix} \tag{11.5}$$

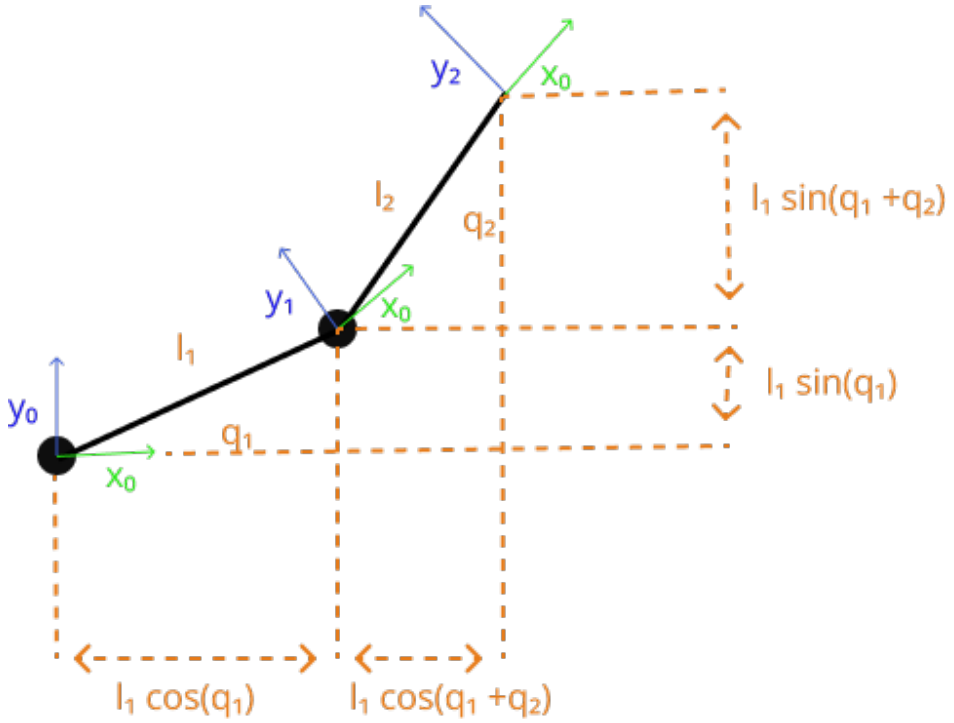


Figure 11.1: With this information it is possible to calculate the forward kinematics. [4]

11.2.2 Inverse Kinematics

Inverse Kinematics deals with the difficulty of finding the right set of joint angles Q of the robot while the desired end effector pose X is given. The mathematical function that needs to be solved is:

$$q = f^{-1}(x) \tag{11.6}$$

[4] Figure uses resources from *Figma* and *4.1 Forward Kinematics.*

In order to reach a specific target position in the robot's coordinate system, a set of joint angles Q needs to be found. There are two different approaches to find Q.

Closed-Form Solution:

As in the Equation 11.6 visible in order to solve Q, the desired pose X is given.

X is a vector function of Q which is equal to Equation 11.1. In contrast to this, the inverse kinematics can be used to solve q_1 and q_2. The closed-form solution[5] is a method to identify all possible solutions for Q. A disadvantage of this method is that the solutions depend on the robot and can not be used in general. To understand this method even better, the inverse kinematic of the planar manipulator in figure 11.1 is calculated with the closed-form solution.

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = f^{-1}(x) \tag{11.7}$$

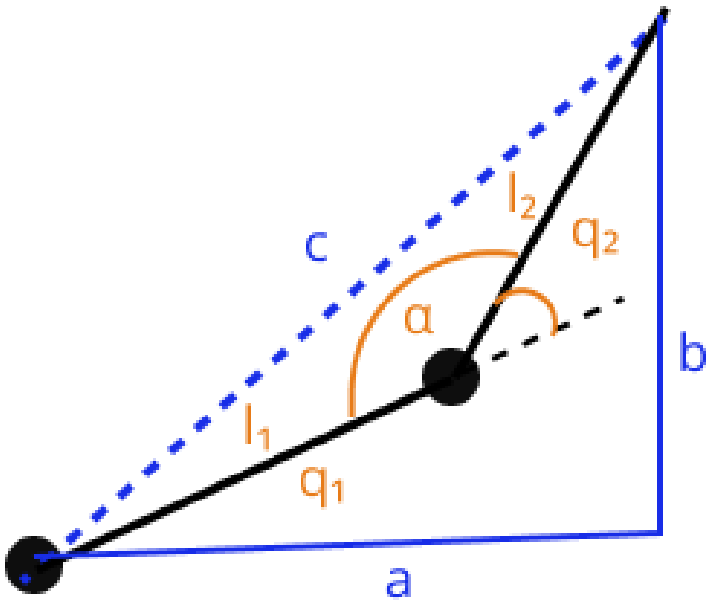


Figure 11.2: Visualization of the planar manipulator with all variables needed to solve q_2. l_1 and l_2 are the link lengths of the manipulator. a and b are the horizontal and vertical totals of the linked joints. [7]

[5] Bruno Siciliano, *Springer Handbook of Robotics*.
[7] Figure uses resources from *Figma* and *5.1 Inverse Kinematics*

To solve the Equation 11.7, a few mathematical basics are necessary. First of all the Pythagorean theorem (11.8) and the Law of Cosines (11.9) is needed to solve for the vector q_2:

$$c^2 = a^2 + b^2 \tag{11.8}$$

$$c^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos(\alpha) \tag{11.9}$$

As can be seen in Figure 11.2, angle α and q_2 must be equal to $\pi - \alpha$ since the two highlighted angles sum up to 180°.

$$q_2 = \pi + \alpha \tag{11.10}$$

To go on with Equation 11.10, both sides need to be in cosine functions, as seen below. Using trigonometry the expression $\cos(\pi)\cos(\alpha) + \sin(\pi)\sin(\alpha)$ is given. Since $\cos(\pi)$ should be -1 and $\sin(\pi)$ should be 0, it results in $\cos(\alpha)$. The result is equal to the negative of the Equation 11.9.

$$\begin{aligned}
\cos(q_2) &= \cos(\pi - \alpha) \\
&= \cos(\pi)\cos(\alpha) + \sin(\pi)\sin(\alpha) \\
&= -\cos(\alpha) \\
&= \frac{d^2 - l2_1 - l2_2}{2l_1 l_2}
\end{aligned}$$

Therefore, two possible solutions for q_2 exist, which are visualized in Equation 11.11.

$$q_2 = \left\{ \begin{array}{c} \pi - \cos^{-1}\left(\frac{l2_1 - l2_2 - x^2 - y^2}{2l_1 l_2}\right) \\ \cos^{-1}\left(\frac{x^2 + y^2 - l2_1 - l2_2}{2l_1 l_2}\right) \end{array} \right\} \tag{11.11}$$

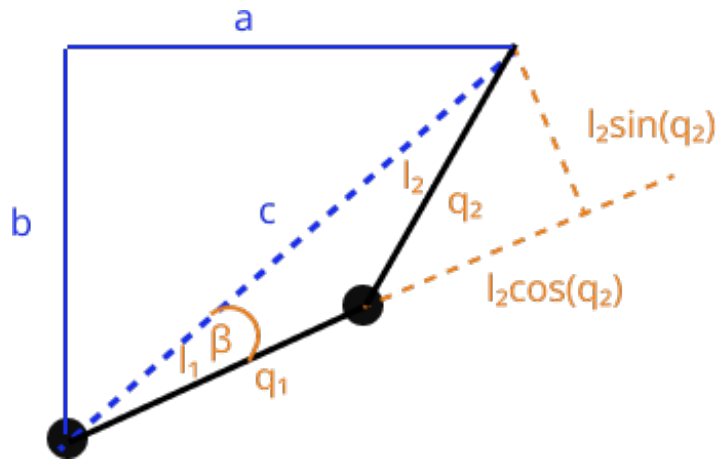


Figure 11.3: Visualization of the planar manipulator with all variables needed to solve q_1. [8]

Since q_2 is calculated, the next step is to solve q_1 with the tangent rule and the law of cosines. The tangent rule says that it is possible to find unknown parts of a triangle with either two angles and one side or two sides and one angle given.

$$\tan(q_1 + \beta) = \frac{b}{a}$$
$$q1 = tan^{-1}(\frac{b}{a} - \beta)$$

In order to solve q_1, the tangent rule needs to be used with the tangent of q_1 and β, which must equal the b over a, as shown in the Equations above. Since β is unknown, the inverse tangent needs to be used, as shown in the Equation 11.12.

$$\beta = \tan^{-1}(\frac{l_2 \sin(q_2)}{l_1 + l_2 \cos(q)_2}) \tag{11.12}$$

Since β can be calculated with Equation 11.12, it can be merged to Equation 11.13.

$$q_1 = \tan^{-1}(\frac{b}{a}) - \tan^{-1}(\frac{l_2 \sin(q_2}{l_1 + l_2 \cos(q_2)}) \tag{11.13}$$

[8] Figure uses resources from *Figma* and *5.1 Inverse Kinematics*.

Now that q_1 and q_2 are solved, the vector function q can be defined. However, since two solutions were calculated for q_2, there are two solutions for the function q as well.

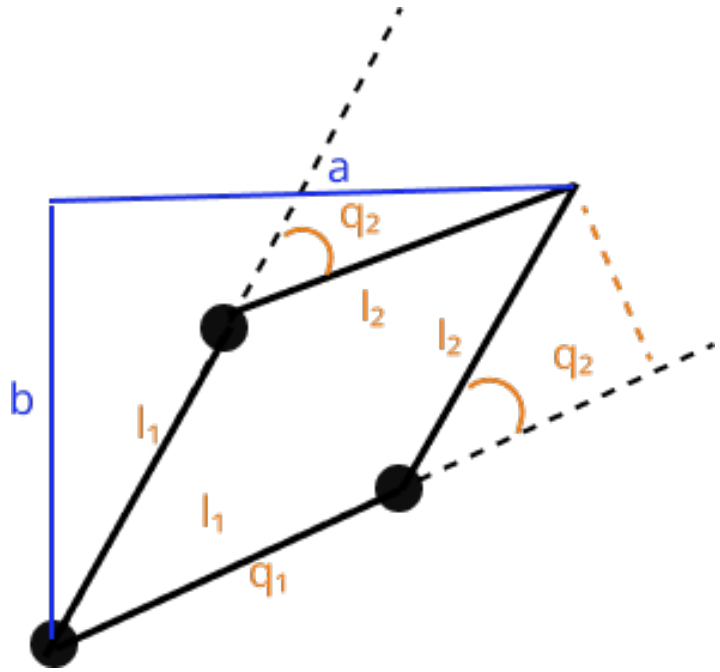


Figure 11.4: Visualization of the two-linked manipulator with two solutions. [9]

$$q = \begin{bmatrix} \tan^{-1}(\frac{b}{a}) - \tan^{-1}(\frac{l_2 \sin(q_2)}{l_1 + l_2 \cos(q_2)}) \\ \pi - \cos^{-1}(\frac{l1^2 - l2^2 - x^2 - y^2}{2 l_1 l_2}) \end{bmatrix} \tag{11.14}$$

or

$$q = \begin{bmatrix} \tan^{-1}(\frac{b}{a}) - \tan^{-1}(\frac{l_2 \sin(q_2)}{l_1 + l_2 \cos(q_2)}) \\ \cos^{-1}(\frac{a^2 + b^2 - l1^2 - l2^2}{2 l_1 l_2}) \end{bmatrix} \tag{11.15}$$

Mathematical Optimization:

With mathematical optimization, it is possible to minimize the pose error between the desired and actual end effector transform given the joint configuration Q. This method is often used for 3 DOF [10] or more manipulators. In contrast to the closed-form solution, the calculations

[9] Figure uses resources from *Figma* and *5.1 Inverse Kinematics*.
[10] degree of freedom

mostly are made by the computer and can be used to solve inverse kinematics for any generic robot arm.

$$\min_{q} ||x - f(q)|| \tag{11.16}$$

To solve this optimization problem, the Equation 11.16 is given. It has the objective function to minimize the robot tooltip pose error. The variable q under the *min* is the decision variable which will be modified to minimize the error.

$$q_{min} \leq q \leq q_{max} \tag{11.17}$$

Also given are the inequality constraints on the solution as in the Equation 11.17. The solutions should obey the joint limits.
To solve inverse kinematics with optimization, gradient descent is necessary. Gradient descent minimizes a scalar cost function, which describes the end effector pose gradient, by incrementing the state variable down the gradient of the function.

Starting with the desired end effector transform T_d, and the actual end effector transform T_k calculated from forward kinematics at this particular time k:

$$T_d = \begin{bmatrix} R_d & p_d \\ 0_{1\times 3} & 1 \end{bmatrix} T_k(q) = \begin{bmatrix} R_k & p_k \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{11.18}$$

T_d is the desired transform of the end effector in a homogeneous transformation matrix and T_k is the current transform with the configuration q. The difference in the desired and actual pose is given by Δx, which consists of the position error e_p and the orientation error e_0.

$$\Delta x = \begin{bmatrix} e_p \\ e_o \end{bmatrix}$$
$$e_p = p_d - p_k$$
$$e_o \leftarrow R_e = R_d R_k$$

The position error e_p is the actual position subtracted from the desired position. However, the orientation error e_o is expressed in the RPY[11] angles, extracted from the rotation error R_e.

[11] roll-pitch-yawn angle = attitude angles that are used to describe the orientation of a vehicle in three dimensional space *RPY*

$$
\begin{aligned}
x &= f(q) \\
\Delta x &= (\partial f / \partial q) \Delta q \\
\Delta q &= (\partial f / \partial q)^{-1} \Delta x
\end{aligned}
$$

As already mentioned, x is some vector function of q in the forward kinematic. Therefore, Δx is the partial derivative of f divided with partial derivative of q, multiplied by Δq. To solve Δq, the inverse of this partial derivative matrix is used.

$$
\begin{aligned}
q_{k+1} &= q_k + \alpha \Delta q_k \\
&= q_k + \alpha (\partial f / \partial q)^{-1} \Delta x_k
\end{aligned}
$$

Now, the joint configuration q can be incremented to cause the actual end effector pose to converge on the desired pose. Δ q is already given from previous equations and the Δx is the pose error. α is a one dimensional scalar and the matrix of partial derivatives $(\partial f / \partial q)^{-1}$ is $m \times n$ matrix, which maps the pose error Δx from Cartesian[12] space to joint space q. This process will be repeated until $||\Delta q|| \sim 0$.

The solution is depending on the initial guess q_0. To optimize the next configuration, the previous solution for q is used.

$$
\begin{aligned}
q_1 &= f^{-1}(x_1, q_0) \\
q_2 &= f^{-1}(x_2, q_1) \\
q_3 &= f^{-1}(x_3, q_2)
\end{aligned}
$$

The use of previous results of the inverse kinematic solution ensures congruous joint motion, but also solves the optimization faster.

11.3 Why would it be better to use inverse kinematic?

Inverse kinematics provide the ability to control the endpoint of the manipulator in a rectilinear motion. This requires several axes to move simultaneously. However, this method is

[12] coordinate system for three dimensional space

much more difficult than forward kinematics to implement because inverse kinematics require their own computational overhead.
Inverse kinematics cause the following problem:

11.3.1 Inverse kinematics problem

One of the first difficulties of inverse kinematics is to find the right vector of joint configurations which result the position of the TAP. It could be that there are infinite number of solutions for the joint variables or that there is none.

There are three ways in which inverse kinematics can be implemented in the robotic arm. The most difficult and time-consuming way would be to write a custom C++ library that deals with the mathematical implementation. Another possibility is to use ROS[13] modules. The ROS "Move It"[14] module has support for inverse kinematics, which can do all the work of calculating the inverse kinematics at runtime.

Another way to implement the inverse kinematics would be to use an industrial controller. Either a complete PLC [15] from Beckhoff[16] could be used, or the Beckhoff PLC software could run on a dedicated PC. As the Beckhoff TwinCAT software is free to use for testing purposes, it would be usable. Numerous kinematic models are available to use. Unfortunately, no kinematic model is available that would fit the FLAME robot arm precisely, so the six-axis-articulated model would need to be used. Two of the six axes do not exist and so would need to be locked in the kinematic model, which is probably possible but most likely not without contacting Beckhoff and receiving support. For this reason, the idea of using a Beckhoff industrial robot controller was discarded.

11.3.2 Custom C++ Library

It was determined that a custom C++ library is the best option and manageable when the user is expected to control the robot arm by specifying kartesian coordinates instead of four angles. It would be possible to implement the kinematics on the Arduino, but this would reduce the freedom of the user. Since the kinematics are implemented in the FLAME library, the end user can decide whether to use the inverse kinematics or to set each joint's angle directly.

[13] ROS - Robot Operating System *ROS*
[14] Move It Motion Planning Framework from ROS
[15] Programmable Logic Controller
[16] *Beckhoff*.

11.4 Implementation of the inverse kinematics

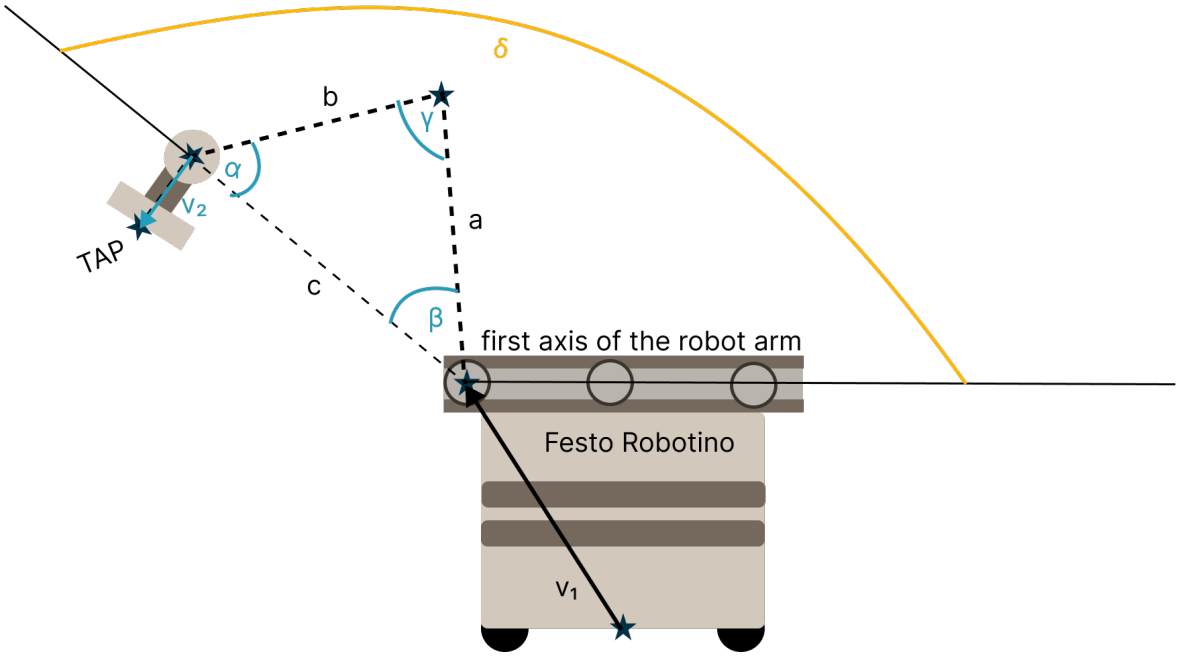


Figure 11.5: Visualization of the FLAME robot arm on the Festo Robotino®. [17]

To implement the inverse kinematics, the vector v_1, the coordinates of the TAP and the lengths of the axis a and b has to be given. The vector v_1 is a vector from the bottom center of the Festo Robotino® to the first axis. The lengths a and b are fixed because the length of the axis can not be changed. The vector v_2 is a vector from the third axis to the TAP. In order to calculate c the vectors v_1 and v_2 are used.

$$c^2 = a^2 + b^2 - 2 * ab * cos(\gamma) \tag{11.19}$$

The length between the positions of the first and third axis give the length of c. In the next step the angles α, β, and γ will be calculated. To determine the angles it is necessary to use the Equation 11.20 the inverse cosines of a, b and c.

$$\alpha = arccos(\frac{b^2 + c^2 - a^2}{2bc}) \tag{11.20}$$

[17] Figure uses resources from *Figma* and *5.1 Inverse Kinematics*.

$$\beta = arccos(\frac{a^2 + c^2 - b^2}{2ac})$$
$$\gamma = arccos(\frac{a^2 + b^2 - c^2}{2ab})$$

Theoretically, all important variables are now determined, and the robot arm should be in the desired position. However, these variables were only so easy to calculate because of a few simplifications. One of these simplifications is to have only one possible solution for the vector functions q, instead of two, which was described earlier and can be seen in Figure 11.4. In order to get only one solution, it was decided that the solution with the smaller angle for the first axis is always preferred. That is why the angle δ is the direction angle, which is the angle to the right horizontal. The second simplification is that the mathematical calculations work with the position of the TAP and not the actual TCP. Why this simplification was made, can be read in Subsection 11.1.

Chapter 12

Native Library

Author: Niklas Wieser

This chapter is focused on the development of the ODrive Native Library and its design decisions. It forms the backbone of communication between the ODrive boards and the Arduino. Questions like why it is necessary to develop a new communication library from scratch or how it is integrated into the robot arm system will be answered.

12.1 Major firmware issues

The first step after the ODrive hardware got delivered was a check for completeness. The delivery went well and nothing was missing or was damaged. So far, so good. The next step was to get familiar with all the features and build up the knowledge needed for developing. It did not take long to go for the more complicated way of writing a library for controlling multiple ODrive boards simultaneously with the native protocol and UART. Some irregularities occured during early attempts of sending native protocol packets from the Arduino to the ODrive board, shown in Figure 12.1. In just about 1 out of 50 requests a response was sent from the ODrive. It was clear that there was an issue somewhere in the firmware of the ODrive board.

Fortunately, the source code of the ODrive firmware is open-source and therefore available on Github. After much debugging, the cause of the problem was detected. However, since version v0.5.2, the implementation of the native protocol with UART has been refactored and it turned out to have major issues. The ODrive Forum brought up the idea of downgrading to firmware version v0.5.1 and this solved the issue. For this reason firmware version v0.5.1 must be used until the bug finally gets resolved in the future.

```
Request sent: w1 as AA 8 3D F5 0 1 80 4 0 40 9B E9 9
Duration time: 5
Request sent: w1 as AA 8 3D F6 0 1 80 4 0 40 9B 42 70
Duration time: 5
Request sent: w1 as AA 8 3D F7 0 1 80 4 0 40 9B CF 84
Duration time: 5
Request sent: w1 as AA 8 3D F8 0 1 80 4 0 40 9B B3 D3
Duration time: 5
```

Figure 12.1: A screenshot of the Arduino serial monitor running test code. No response from ODrive.

```
Request sent: w1 as AA 8 3D D3 0 1 80 4 0 40 9B 90 84
Response: AA 6 0 D3 80 0 0 40 41 8E 8F
12.00
Duration time: 8
Request sent: w1 as AA 8 3D D4 0 1 80 4 0 40 9B 76 E7
Response: AA 6 0 D4 80 0 0 40 41 B0 FD
12.00
Duration time: 8
Request sent: w1 as AA 8 3D D5 0 1 80 4 0 40 9B FB 13
Response: AA 6 0 D5 80 0 0 40 41 71 75
12.00
Duration time: 8
Request sent: w1 as AA 8 3D D6 0 1 80 4 0 40 9B 50 6A
Response: AA 6 0 D6 80 0 0 40 41 E 88
12.00
Duration time: 9
Request sent: w1 as AA 8 3D D7 0 1 80 4 0 40 9B DD 9E
Response: AA 6 0 D7 80 0 0 40 41 CF 0
12.00
Duration time: 8
```

Figure 12.2: A screenshot where the problem got solved. Every single request gets a response.

12.2 Integration

The primary purpose of the ODrive Native Library is to generate communication packets. It is written in ANSI C++ without the use of the C++ standard library, which enables it to be compiled on Arduino or any other system. Figure 12.3 gives a helpful overview of the entire software side of this diploma thesis and shows the separation into two parts. The ODrive Native Library between an Arduino and the ODrive boards and on the other hand, the so-called FLAME-Library, handles communication between the Arduino and a client PC using Ethernet and UDP. More about the FLAME-Library in chapter 13.6.

As mentioned in chapter 7.1.6 the decision fell towards the ODrive native communication protocol for several reasons. The problem is that very few people are using this protocol, and there is hardly any information available apart from the ODrive documentation itself. Some results of the ODrive Native Library are:

- The ability to outsource the calculation of motor positions to external devices and thereby creating an interface for communication. That simplifies the implementation of graphical user interfaces, which is not part of this diploma thesis but could be the objective of others in future.

- The update rate can get up to 100Hz which enables a smooth motion behavior at any speed and high accuracy.

- The possibility to use the UART protocol instead of USB for communication and as a result of that, increasing reliability.

12.2.1 Synchronous vs. asynchronous communication

To achieve a high update rate, communication between ODrive and Arduino needs to be asynchronous. However, testing synchronous communication in an early stage of development showed that the performance of sending and receiving packets is far from reaching the goals.

The asynchronous approach showed much better results than sending one packet, which holds the information for one axis of the robot arm and waits until the response arrives. Four packets containing the next target position for each axis are sent every ten milliseconds. In the meantime, the Arduino constantly listens to the serial ports and parses incoming responses. When looking at Figure 12.4, it gets evident why an asynchronous approach is superior.

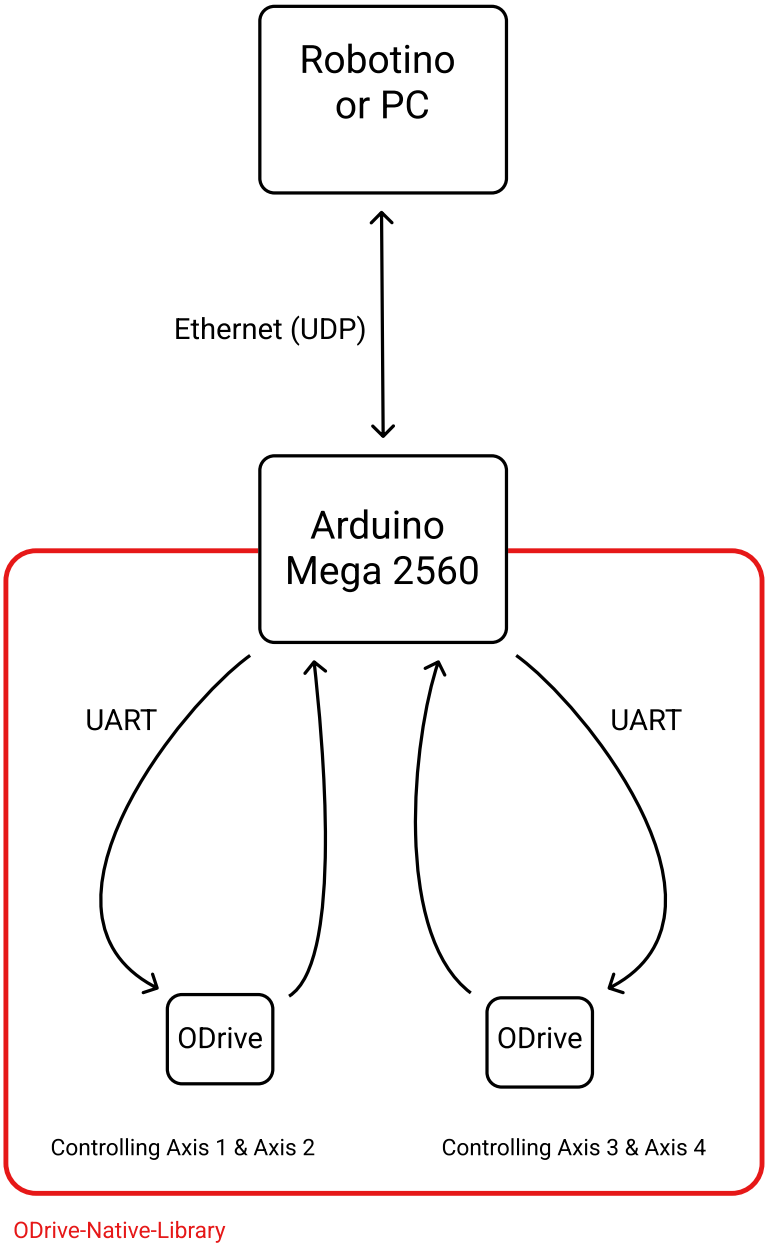


Figure 12.3: A sketch of the overall software-architecture and where the ODrive Native Library is integrated.

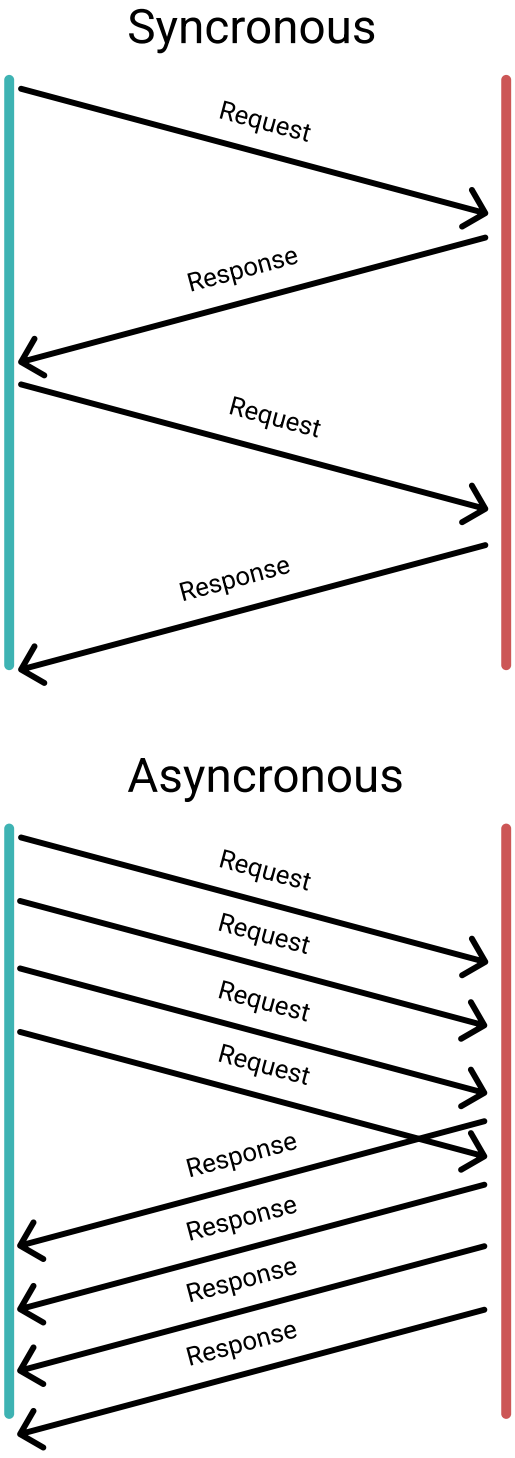


Figure 12.4: A visual representation of synchronous and asynchronous data transfer.

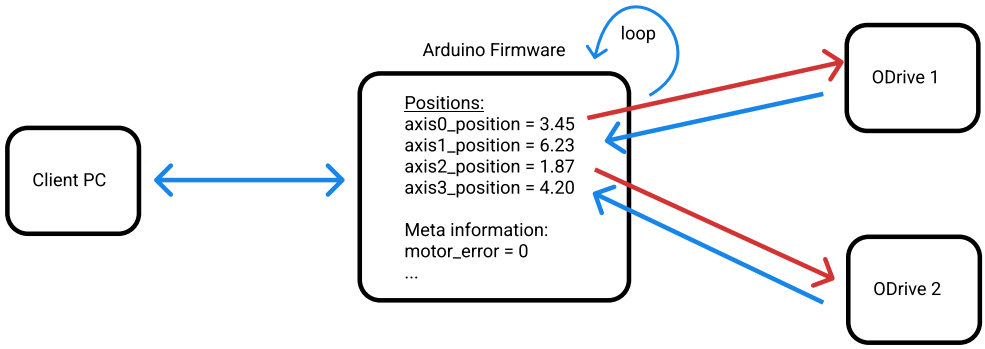


Figure 12.5: The working principle of the Arduino Firmware.

12.2.2 Arduino Firmware

This part of the FLAME software links everything together. The Arduino firmware utilizes the ODrive Native Library for sending and receiving data from the ODrive boards and, on the other hand, the FLAME-Library described in section 13.6, for communication with a client PC over Ethernet.

As shown in figure 12.5, it works by using several shared variables that act as a buffer. When a packet from the client PC arrives, it gets parsed by the FLAME-Library and, e.g. the value for the target position of the second axis gets written to the shared variable. The axis positions get updated at a specific rate, for example every ten milliseconds. This is where the ODrive Native Library does the work. The target positions from the shared variables get sent one by one to the ODrive boards. Data transfer goes in both directions, so the same principle applies to the other way round. There is no need to worry about race conditions as everything is single-threaded.

12.3 Implementation

The following pages focus on the inner structures of the ODrive Native Library. As well as the creation of and parsing packets for different endpoints, as well as error prevention and error handling.

12.3.1 The Update-Process

The purpose of the Update-Process is to implement a safety mode. One of the highest priorities during development is to ensure a certain level of security. The robot arm is quite heavy and could easily damage objects in its surroundings or hurt people.

When the machine detects a dangerous situation, all four axes stop to prevent unfavourable consequences for everyone involved. In 2 cases a full stop gets automatically applied:

Case 1: ODrive Errors

Safety behaviour of the robot arm in case any of the following errors occurs cannot be guaranteed. Therefore the update process constantly monitors all endpoints for errors, and if necessary, the motion gets interrupted. The person who operates the device then needs to evaluate the situation and fix the issue. In most cases, a calibration will fix it.

- global axis error
- motor error
- encoder error
- controller error

Case 2: Unrealistic Positions

The cause for a safety stop can also be unrealistic positions. This can happen when UDP control packets, described in section 13.6.1, get corrupted or completely lost during transmission. The plot demonstrates a situation when an emergency stop gets applied. The blue points represent target positions when the transmission is fine and the red points when not. Between time unit 2 and 6, some packets got lost, and the target position was not updated. This creates a problem when intact packets start to arrive again. The robot arm would stop between time unit 2 and 6 but then needs to drive from position 4 to 14 in just one time unit. This would be too dangerous. When the speed or acceleration values are too unrealistic and exceed a certain threshold the system enters a safety mode where positions get no longer accepted until the controller synchronizes the positions again and disables the safety mode.

12.3.2 Endpoints

As mentioned before in chapter 7, ODrive offers a wide variety of settings and allows the user to set everything up according to the given requirements. For firmware version v0.5.1, 547

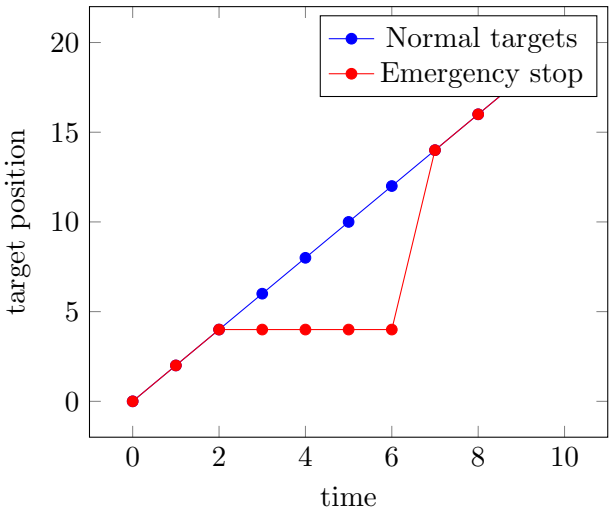


Figure 12.6: Demonstration of an emergency stop.

endpoints are available. ODrive is a very versatile motor control system and was designed for many different use cases. This is why there is such a large number of possible configurations and settings. For example, the user can set the current limit of the first motor to 10A or set a maximum number of revolutions per minute. Currently, there are three different types of endpoints:

- read only
- read and write
- calling a function

Not every endpoint has the same data type. Some require floating-point numbers, others require integers, and for one or the other, a boolean does the job. This increases complexity as the length of a packet varies according to the required data type. For example, the number of bytes for the payload starts at 1 byte for a boolean and ends at 8 bytes for an 8-byte integer. A list of all possible data types:

- boolean (1 byte)
- uint8_t (1 byte)
- uint16_t (2 bytes)
- int16_t (2 bytes)
- uint32_t (4 bytes)

- uint64_t (8 bytes)
- float (4 bytes)

The decision fell towards macros. For the Arduino, an endpoint is just a combination of a 16-bit integer number, the name of a supported datatype and a corresponding identifier. The compiler substitutes the string of tokens for each occurrence of the identifier in the code.

```
#define ENDPOINT_VBUS_VOLTAGE           1, float
#define ENDPOINT_SERIAL_NUMBER          4, uint64_t
#define ENDPOINT_HW_VERSION_MAJOR       5, uint8_t
#define ENDPOINT_BRAKE_RESISTOR_ARMED   12, bool
#define ENDPOINT_UPTIME                 14, uint32_t
```

Listing 12.1: Endpoint defines

12.3.3 Utility

In order to generate and parse ODrive-Native-Packets, some useful functions and a proper class structure is necessary.

ODrive Native Packets

There are two different types of packets[1]:

- Request Packet
- Response Packet

A request is a packet sent by the Arduino and a response is a packet sent by the ODrive board. When using UART, an additional Stream Wrapper gets added in both directions.

[1] *ODrive-Native-Protocol*.

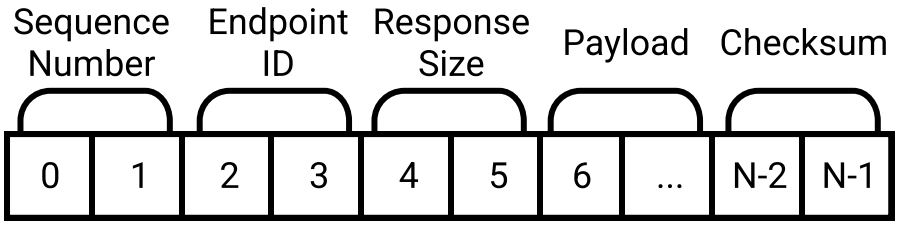


Figure 12.7: The structure of a Request Racket.

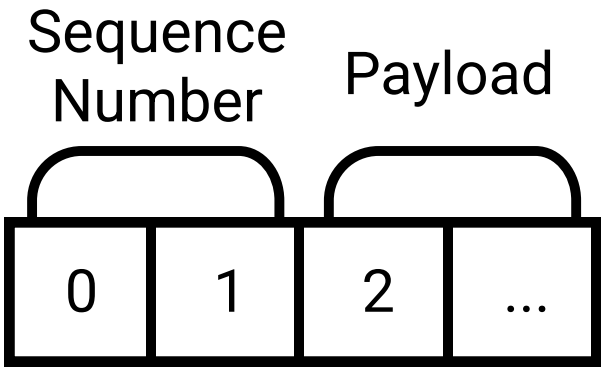


Figure 12.8: The structure of a Response Racket.

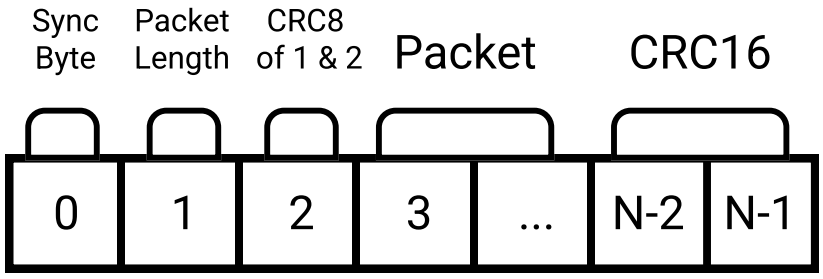


Figure 12.9: The structure of the Wrapper.

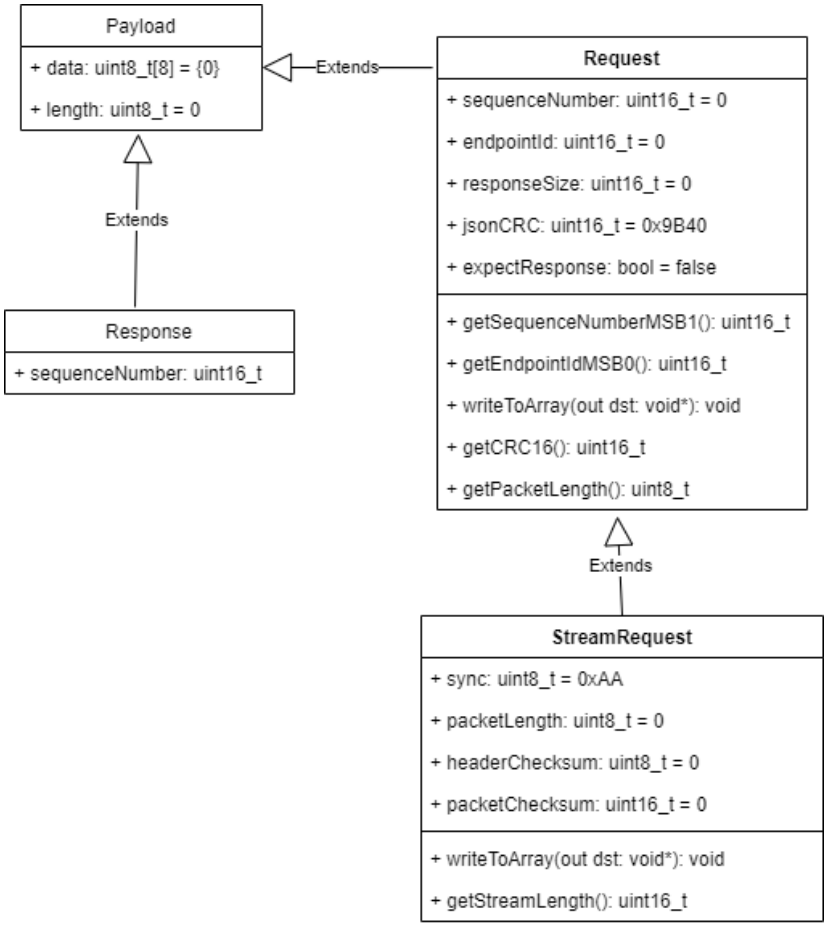


Figure 12.10: UML-Diagram of the class structure.

Packets

Figure 12.10 shows the class structure of the implementation of ODrive Native Packets, which are utilized by the utility functions explained above. These classes are heavily based on the packet format of the ODrive native protocol[2].

[2] *ODrive-Native-Protocol*.

Functions

The problem with ODrive Native Packets is that the overall length of the packet can vary between 0 and 8 bytes.

There are two big functionalities to implement:

- Generating and sending packets based on an endpoint and a value.
- Parsing incoming data into the data structure, shown in Figure 12.10 and checking the contents for correctness.

Sending: The most efficient solution is based un using templates. The template information represents an endpoint with an id and the value's data type. First, according to lines 5 and 6 in the listing below, the value gets written to an array. Next, a StreamRequest object containing meta information like the overall packet length, a sync byte, the sequence number, an endpoint id and CRC checksums gets created and is ready to be sent.

```
1  template <uint16_t endpointId, typename T>
2  StreamRequest GenerateStreamRequestWrite(ODrive* odrive, T value, bool
       forceResponse = false) {
3      Payload payload;
4
5      memcpy(payload.data, &value, sizeof(value));
6      payload.length = sizeof(value);
7
8      odrive->sequenceNumber = (odrive->sequenceNumber + 1) % 1024 + 1;
9      return StreamRequest(Request(odrive->sequenceNumber, endpointId, uint16_t(sizeof
           (value)), payload, access_t::WRITE, forceResponse));
10 }
```

Listing 12.2: Generating StreamRequest

The actual sending is done by another function. It takes the StreamRequest object, serializes it into an array, and sends it to one of the ODrive boards.

```
1  void sendStreamRequest(ODrive* odrive, const StreamRequest& stream) {
2      stream.writeToArray(outBuffer);
3      odrive->serialWrite(outBuffer, stream.getStreamLength());
4  }
```

Listing 12.3: Sending StreamRequest

Receiving: Receiving data from the ODrive board is split into 2 parts:

- Constantly listening to the data stream and parsing the data that arrives.
- When a response got successfully parsed, the payload data gets extracted and written to the corresponding shared variable.

The following pseudocode explains how incoming data gets parsed.

```
while bytesAvailable ≥ 1 do
    if parserState is HEADER then
        if bytesAvailable ≥ 3 then
            if nextByte is 0xAA then
                syncByte ← nextByte()
                length ← nextByte()
                CRC8 ← nextByte()
                if checkCRC8(syncByte, length) is CRC8 then
                    parserState ← PAYLOAD
                else
                    Discard all bytes until the next one is 0xAA
                end if
            else
                Discard all bytes until the next one is 0xAA
            end if
        else
            Less than 3 bytes in the Buffer, return and continue later
        end if
    else if parserState is PAYLOAD then
        if bytesAvailable ≥ length then
            Write the next n bytes to payloadBuffer
            parserState ← TRAILER
        else
            Not enough bytes available, return and continue later
        end if
    else if parserState is TRAILER then
        if bytesAvailable ≥ 2 then
            recieved ← next two available Bytes
            calculated ← CRC16 of payloadBuffer
            if recieved is calculated then
                Write the payload to the corresponding shared variable
            end if
            parserState ← HEADER
        end if
```

end if
end while

12.3.4 CRC

The ODrive Native Protocol utilizes CRC, a cyclic redundancy check. These are extra bytes that are added at the end of a packet. However, what is the purpose of these extra bytes? The answer is quite simple. CRC is an error detection technique that enables the packet receiver to verify whether this packet has been damaged. Following a simple example to demonstrate the purpose of checksums:

- 0x34 0x23 0x32, is a packet without a checksum
- 0x34 0x23 0x32 0x89, is a packet with a checksum

The transmitter appends a calculated value to the packet. In this case, the values were summed up, which is the easiest way to calculate a checksum. The receiver applies the same method as the transmitter and checks if the checksum is correct. Assume that the packet looks like this after transmission: 0x34 0x3 0x32 0x89 but $34 + 3 + 32 = 69$ and not 89. The packet gets discarded. Summing up is perfect for an example, but the chance that an error would not be detected is too high.[3]

CRC uses division, where the data from the packet gets interpreted as a long bitstream, which gets divided by another fixed binary number called generator polynomial. The actual CRC value is the remainder of the division. The actual implementation of CRC uses a shift register and XOR operations.[4]

Lookup Table

The most efficient way to calculate a CRC is by using a lookup table. This table needs to be calculated every time the generator polynomial changes. In the case of ODrive, these values are fixed, which makes the use of a hardcoded 8-bit integer array possible. For CRC8 this value is 0x37 and 0x3d65 for CRC16.[5] These values can be retrieved from the firmware of the ODrive board. The code listing below shows a so-called lookup table for CRC8, filled with pre-computed results for all 256 possible bytes. Starting with a fixed initial value, a bitwise XOR of 0x42 and the first byte from the input stream gets calculated. The result is the index of the next initial value. After the last byte is processed, the loop ends the final CRC gets returned. The principle of a CRC16 lookup table is similar to CRC8.

[3] Williams, *A painless guide to CRC error detection algorithms*.
[4] Molkenthin, *Understanding and implementing CRC calculation*.
[5] *ODrive-Native-Protocol*.

```
uint8_t CRC8(uint8_t* data, size_t len) {
    uint8_t crc = 0x42;
    static const uint8_t table[] = {
      0x00, 0x37, 0x6E, 0x59, 0xDC, 0xEB, 0xB2, 0x85, 0x8F, 0xB8, 0xE1, 0xD6, 0x53,
      0x64, 0x3D, 0x0A, 0x29, 0x1E, 0x47, 0x70, 0xF5, 0xC2, 0x9B, 0xAC, 0xA6, 0x91,
      0xC8, 0xFF, 0x7A, 0x4D, 0x14, 0x23, 0x52, 0x65, 0x3C, 0x0B, 0x8E, 0xB9, 0xE0,
      0xD7, 0xDD, 0xEA, 0xB3, 0x84, 0x01, 0x36, 0x6F, 0x58, 0x7B, 0x4C, 0x15, 0x22,
      0xA7, 0x90, 0xC9, 0xFE, 0xF4, 0xC3, 0x9A, 0xAD, 0x28, 0x1F, 0x46, 0x71, 0xA4,
      0x93, 0xCA, 0xFD, 0x78, 0x4F, 0x16, 0x21, 0x2B, 0x1C, 0x45, 0x72, 0xF7, 0xC0,
      0x99, 0xAE, 0x8D, 0xBA, 0xE3, 0xD4, 0x51, 0x66, 0x3F, 0x08, 0x02, 0x35, 0x6C,
      0x5B, 0xDE, 0xE9, 0xB0, 0x87, 0xF6, 0xC1, 0x98, 0xAF, 0x2A, 0x1D, 0x44, 0x73,
      0x79, 0x4E, 0x17, 0x20, 0xA5, 0x92, 0xCB, 0xFC, 0xDF, 0xE8, 0xB1, 0x86, 0x03,
      0x34, 0x6D, 0x5A, 0x50, 0x67, 0x3E, 0x09, 0x8C, 0xBB, 0xE2, 0xD5, 0x7F, 0x48,
      0x11, 0x26, 0xA3, 0x94, 0xCD, 0xFA, 0xF0, 0xC7, 0x9E, 0xA9, 0x2C, 0x1B, 0x42,
      0x75, 0x56, 0x61, 0x38, 0x0F, 0x8A, 0xBD, 0xE4, 0xD3, 0xD9, 0xEE, 0xB7, 0x80,
      0x05, 0x32, 0x6B, 0x5C, 0x2D, 0x1A, 0x43, 0x74, 0xF1, 0xC6, 0x9F, 0xA8, 0xA2,
      0x95, 0xCC, 0xFB, 0x7E, 0x49, 0x10, 0x27, 0x04, 0x33, 0x6A, 0x5D, 0xD8, 0xEF,
      0xB6, 0x81, 0x8B, 0xBC, 0xE5, 0xD2, 0x57, 0x60, 0x39, 0x0E, 0xDB, 0xEC, 0xB5,
      0x82, 0x07, 0x30, 0x69, 0x5E, 0x54, 0x63, 0x3A, 0x0D, 0x88, 0xBF, 0xE6, 0xD1,
      0xF2, 0xC5, 0x9C, 0xAB, 0x2E, 0x19, 0x40, 0x77, 0x7D, 0x4A, 0x13, 0x24, 0xA1,
      0x96, 0xCF, 0xF8, 0x89, 0xBE, 0xE7, 0xD0, 0x55, 0x62, 0x3B, 0x0C, 0x06, 0x31,
      0x68, 0x5F, 0xDA, 0xED, 0xB4, 0x83, 0xA0, 0x97, 0xCE, 0xF9, 0x7C, 0x4B, 0x12,
      0x25, 0x2F, 0x18, 0x41, 0x76, 0xF3, 0xC4, 0x9D, 0xAA
    };

    if (data == NULL)
        return 0xff;

    crc &= 0xff;
    while (len--)
        crc = table[crc ^ *data++];

    return crc;
}
```

Listing 12.4: Implementation of a CRC8 lookup table

Chapter 13

Libraries

Author: Antonia Oberhauser

Several libraries must be developed to establish communication with the robot arm. To give these libraries a certain structure, GitHub and submodules were used. These submodules allow to include other libraries from different repositories. Now every individual library can be in its own repository, in case it is needed in a separate project. To increase the user friendliness Premake is used as the build system.

13.1 Repositories

A repository is a place where all files from this project are saved. This repository can be private or public. If it is public then everybody in the world has access to the files and can save them locally. To be able to see the repository, a GitHub account is necessary.

13.2 Github

GitHub[1] is a United States-based global company that provides hosting for software development version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features.

[1] *GitHub*.

13.3 Submodules

As already mentioned Github submodules are used to structure the project. The project has an big amount of files, which are divided into three folders. The source files are in the source and include folders. There is also the required premake5.exe and premake5.lua files to build the project with premake. The third folder is called modules and links the repository of the FLAME library. This library in turn points to the protocol library called FLAME_Protocol and to the network library named NetLib. The NetLib also links to two external libraries termed asio and spdlog. This is how the directory tree from the project looks like:

```
User project
├── include
├── src
└── modules
    └── FLAME
        └── modules
            ├── FLAME_Protocol
            └── NetLib
                ├── asio
                └── spdlog
```

13.4 Premake

Premake[2] is a command line utility which reads a scripted definition of a software project and, most commonly, uses it to generate project files for tool sets like Visual Studio, Xcode, or GNU Make.

The portable Premake executable is part of the repository itself, therefore the library can always generate build files anywhere without ever installing Premake on the system. One of the greatest advantages of this build system is the user-friendliness. Since the end product should also be usable for students without any programming experience, the libraries were designed to be easy to use. The entire project can be set up by executing the platform specific generation script, which asks for the project name and sets up the build environment for the selected compiler.

[2] *Premake*

13.5 How is Premake used in the project?

This Project is available on the operating systems Windows and Linux and Mac OS. With the power of Premake only one command needs to be executed and the library starts compiling. On Windows it is possible to create a new project with a click on a batch file. Then a console opens asking for a project name. After entering a name a Visual Studio Project is created. On a Linux based operating system the user has to execute a shell script, which will ask for the project name and generates a build folder for the user.
An important aspect of the project is that the actual application and other files are statically linked. This means, that the executable file is not linked to shared libraries and does not need any third-party libraries loaded at runtime except the system libraries. This results in a universal, portable, self-contained executable which can be run on any instance of a given operating system. No additional dependencies need to be installed before running the compiled executable. On Windows, this means that the Visual Studio C+ runtime is required for compiling, but not for executing. The only downside of this approach is that the size of the executable increases drastically, but it is not a problem for a project of this size. The executable would realistically jump from 500kB to 5-10mB, which is still totally acceptable.

13.6 FLAME library

The FLAME library is required to send, receive and parse UDP packets from the user PC to the Arduino. In order to understand the functionality of the library, the different types of packets must first be defined. There are four types of packets declared in the FLAME_Protocol library.

13.6.1 TCP vs UDP

TCP

TCP is the shortening for Transmission Control Protocol. This protocol is the basic communication language of the internet. However, it can also be used as a communication protocol within a private network. As a connection-oriented protocol, it has three main tasks:

- Prevent data loss
- Split files and data streams
- Assign data packets to the appropriate applications

TCP/IP is based on two layers. The upper layer, TCP, separates a message or file into smaller packets. After these have been transmitted, they are also put together by a TCP layer at the receiver in such a way that these small packets ultimately result in the original message or file. The lower layer is the IP[3]. It ensures that every package arrives at the right destination. TCP/IP uses the client-server model. This means that a specific service is requested from a computer client. This service is provided by another computer on the network, which is the server.

UDP

UDP stands for User Datagram Protocol. It is a connectionless transport protocol. Although the UDP has to fulfill similar tasks as the TCP, it works – in contrast to the TCP – without a connection and insecurely. This means that as a sender, you never know whether the sent data packet has arrived, since no confirmation of receipt is sent. UDP is mainly used for DNS[4] queries, VPN[5] connections and audio and video streaming because it is faster due to its simplified way of working. UDP packets are transported directly to the application without being numbered. UDP cannot put the data packets together in the right order. Therefore, it is mainly used for applications where packet loss is not that bad, or that take care of the connection management themselves, or for those that only need to transport individual data packets without context.

UDP Packets

The robot arm is used in real-time, so latency is extremely concerning. Therefore, UDP packets are used. The following subsection will describe the packets which will be sent over UDP.

[3]Internet Protocol
[4]Domain Name System
[5]Virtual Private System

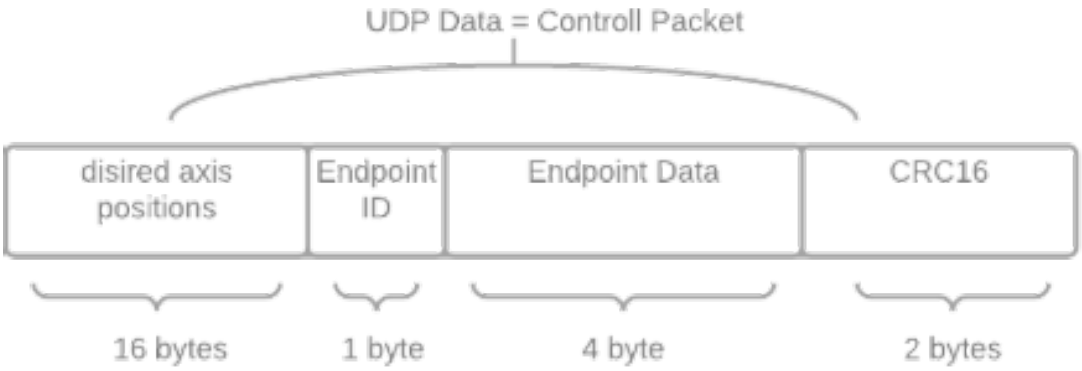


Figure 13.1: UDP Packet Example with Control Packet which will be sent via UDP.[6]

13.6.2 Discovery Packet

The discovery packet consists of one byte with the hexadecimal 0x42 as content. The user PC sends this packet via broadcast to find the right IPv4-address from the Arduino. The hexadecimal 0x42 was chosen to sent because it is unlikely to get exact this byte from a different packet other than the discovery packet.

```
1 #define FLAME_PROTOCOL_CONTROL_BYTE 0x42
2
3 void generatePacket(uint8_t* buffer);
4 bool parsePacket(uint8_t* buffer);
```

Listing 13.1: Discovery Packet structure

UDP Broadcast

A broadcast[7] is a multipoint connection which allows to send a data packets to every device in the same network at the same time. The sender In contrast to unicast, where only a single, known recipient is addressed, the sender does not have to explicitly specify the receiver, but a broadcast address or broadcast IP is used for this. The broadcast address represent the whole

[6] Figure uses resources from *DrawIO*
[7] *Broadcast*.

subnet where the host is located. This address can be determined by using the appropriate operating system commands from any host on the subnet.

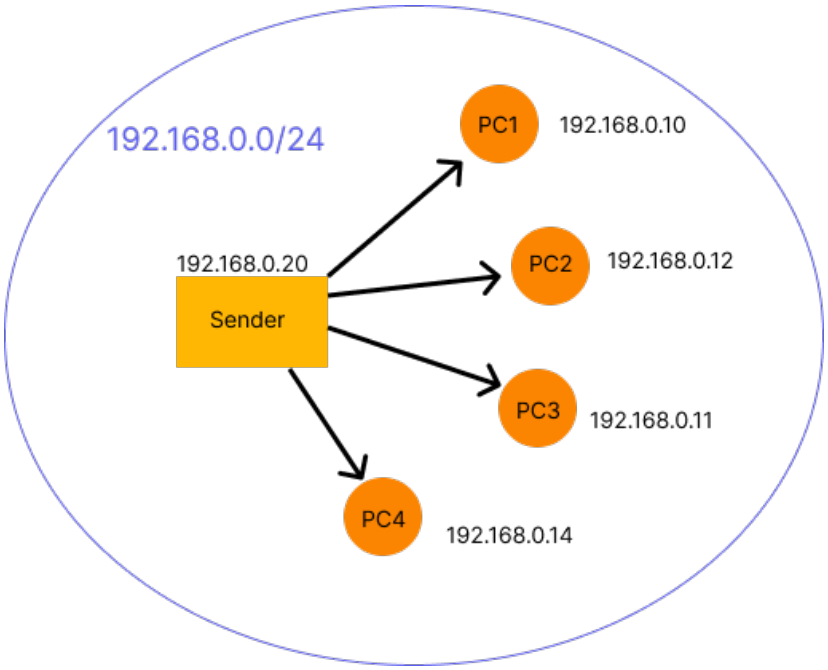


Figure 13.2: Example Broadcast - Sender sends data packet to every device which has a IPv4-address in the range 195.168.0.1 - 195.168.0.254.[9]

13.6.3 Discovery Response

If the Arduino recieves a discovery packet it respondes with a discovery response packet. It has a size of six bytes. The first four bytes contain the IPv4-address of the Arduino and the remaining bytes are the CRC16 checksum. Detailed information about CRC16 checksums can be read in the subsection 12.3.4.

```
1 struct DiscoveryResponse {
2     uint32_t ipAddress = 0;
3 };
4
5 void generatePacket(FLAME_Protocol::DiscoveryResponse* discoveryResponse, uint8_t*
      buffer);
6
7 bool parsePacket(FLAME_Protocol::DiscoveryResponse* discoveryResponse, uint8_t*
      buffer);
```

[9] Figure uses resources from *Figma*.

Listing 13.2: Discovery Response Packet structure

13.6.4 Control Packet

Now that the user PC has the real IPv4-address of the Arduino it sends the control packet. This packet has a size of 23 bytes. The first four bytes are the desired positions of the axis 1 to 4 in degrees from the reference mark. The fifth byte is the ID of a register element, which specifies a additional data, followed by four bytes, representing any data type. The register ID leads to an enum with important parameters, such as the torque of a motor. Important to say is that the endpoints in the odrive library are not the same as the registers in the Flame Library. The register ID can be cycled through all necessary endpoints, so that the desired positions are constantly updated and all other endpoints every n-th cycle.

```
1  struct ControlPacket {
2          float axis1 = 0;
3          float axis2 = 0;
4          float axis3 = 0;
5          float axis4 = 0;
6          uint8_t id = 0;
7          uint32_t additional = 0;
8      };
9
10
11 void generatePacket(FLAME_Protocol::ControlPacket* controlPacket, uint8_t* buffer);
12
13
14 bool parsePacket(FLAME_Protocol::ControlPacket* controlPacket, uint8_t* buffer);
```

Listing 13.3: Control Packet structure

13.6.5 Review Packet

As soon as the Arduino gets a control packet, it send this review packets in a continuous stream. The review packet has seven bytes. The first byte contains the register id, followed by four bytes with values with datatypes according to the register id. The last two byte are the CRC16 checksum. The register ID is cycled through all available endpoints, so each value is updated every *n-th* cycle.

```
1  struct ReviewPacket {
2          uint8_t id = 0;
```

```
        uint32_t data = 0;
    };


void generatePacket(FLAME_Protocol::ReviewPacket* reviewPacket, uint8_t* buffer);


bool parsePacket(FLAME_Protocol::ReviewPacket* reviewPacket, uint8_t* buffer);
```

Listing 13.4: Review Packet structure

13.6.6 Implementation

Now that the packets are defined, the actual functionality of the FLAME library on the user side can be described.
First, the user PC needs the broadcast addresses to send a discovery packet. To get these, it has to start a server with the port 22500 and loop though all interfaces to create strings with the broadcast addresses in it. 22500 was decided to be the port number, but any available port number could be used. After that, the user PC sends a discovery packet to all addresses and waits until it gets a discovery response from the Arduino.

```

void sendDiscovery() {
    //Get Interfaces
    auto interfaces = NetLib::GetNetworkInterfaces();

    //Create Discovery Packet
    uint8_t discoveryBuffer[FLAME_PROTOCOL_DISCOVERY_PACKET_LENGTH];
    FLAME_Protocol::generatePacket(discoveryBuffer);

    //Send Packet to Broadcast addresses
    for (auto& ifc : interfaces) {
        NetLib::SendUDP(NetLib::CreateBroadcastAddress(ifc), 22500, discoveryBuffer,
            FLAME_PROTOCOL_DISCOVERY_PACKET_LENGTH);
    }
}
```

Listing 13.5: Function for sending Discovery Packets

If a discovery response arrives, the PC saves the IPv4-address of the Arduino into an instance. The instance is a object with a control packet, a review packet, a discovery response packet and serveral other informations, like the receiver IPv4-address. One of the important elements in an instance is the packetbuffer with a maximal size of 23 bytes. The packetbuffer is used to overload the packets and variables in an instance. Now, control packets can be continuously

created and send to the Arduino. The control packets updates all register elements in every *n-th* cycle except the coordinates x, y and the rotation angles α and β. This elements will be updated in every cycle. The program needs to go through an enum with the registers and sends the first four elements with every packet. After that, the current register will be increased by one, so that, the next packet contains the next register value.

```
void sendControllPacket(FLAME::Instance* flame) {

    FLAME_Protocol::ControlPacket cp;

    cp.x = intToFloat(flame->registers[0]);
    cp.y = intToFloat(flame->registers[1]);
    cp.alpha = intToFloat(flame->registers[2]);
    cp.beta = intToFloat(flame->registers[3]);
    cp.id = flame->current_register;
    cp.additional = flame->registers[flame->current_register];

    if (flame->current_register < REG_NUM) {
        flame->current_register++;
    }
    else {
        flame->current_register = 4;
    }
    uint8_t buffer[23];

    generatePacket(&cp, buffer);

    while (!flameInstance.clientFound);

    NetLib::UDPClient uc(flameInstance.clientIP, 22500);

    uint64_t old = getMicros();
    uint64_t interval = 100000;
    while (true) {
        if (getMicros() >= old + interval) {
            old = getMicros();

            uc.send(buffer, 23);
        }
    }
}
```

Listing 13.6: Function for Control Packets

[10]Figure uses resources from *Flaticon* and *Figma*.

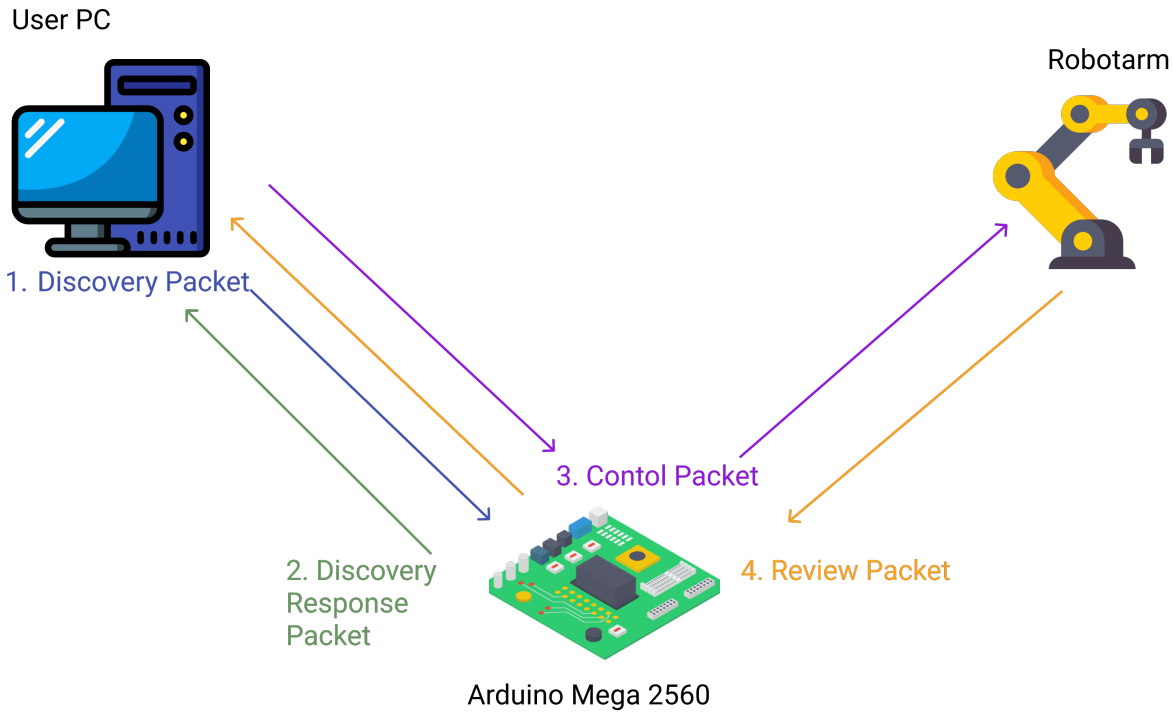


Figure 13.3: Visualization of how the packets are sent. [10]

Chapter 14

Final assembly

Author: Florian Zachs

14.1 What this chapter is about

This diploma thesis documents the entire process of developing the robotic arm. This chapter is about the final product, of which the development process and working principle was explained in the previous chapters.

The following chapter consists mainly of images, the goal is to show how the final assembly looks like in the real world.

14.2 The finished robotic arm

Figure 14.1 shows the finished robotic arm. It is not mounted on the Festo Robotino in the images, the process of mounting it is not documented in this diploma thesis.

As explained in previous chapters, most parts of it are 3D printed. In the Creo model all 3D printed parts were orange but for the real assembly all parts were printed with black PLA filament, because this is what was available at the time of printing.

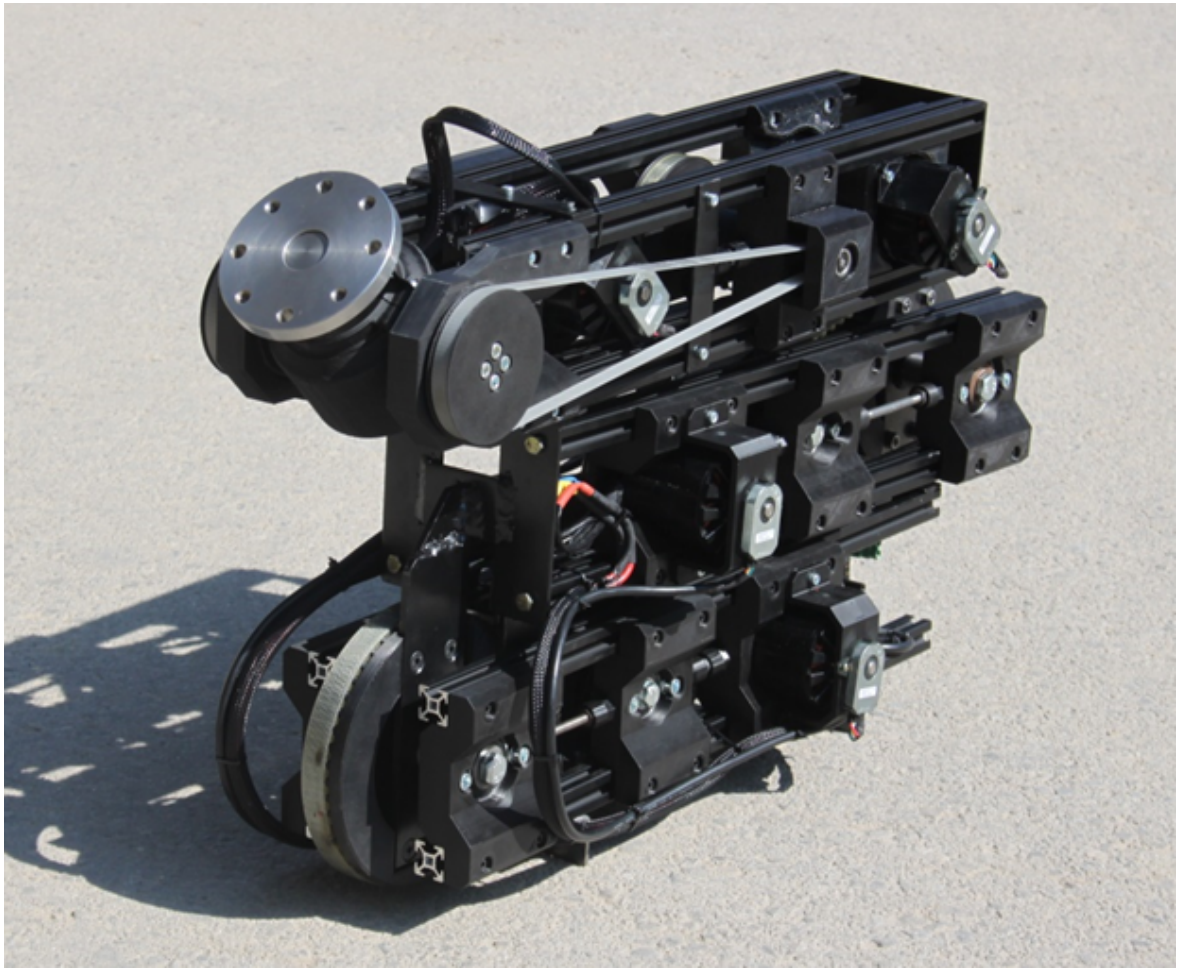


Figure 14.1: The finished robotic arm

14.3 The drive system

The drive system can be seen in Figure 14.2. As documented in previous chapters, timing belts are used to drive the axes. Figure 14.2 shows the two-stage belt reduction of axis 1, it has a reduction ratio of 1:25. All other axes have a reduction ratio of 1:16.

Figure 14.3 shows how the pulley of axis 3 sits in the bearing. In the center of the pulley a metal pin can be seen. This is because the pulley broke immediately when it was first assembled. It could not handle the shear forces from the belt tension, mainly because the 3D print had a very low quality and the settings were not correct for the PETG filament. It was printed again with PLA and the proper settings and to improve it further, a hole was modeled all the way through and a metal pin was driven through to prevent it from shearing off again. This was done for all small 3D printed pulleys and solved all problems. The pulleys

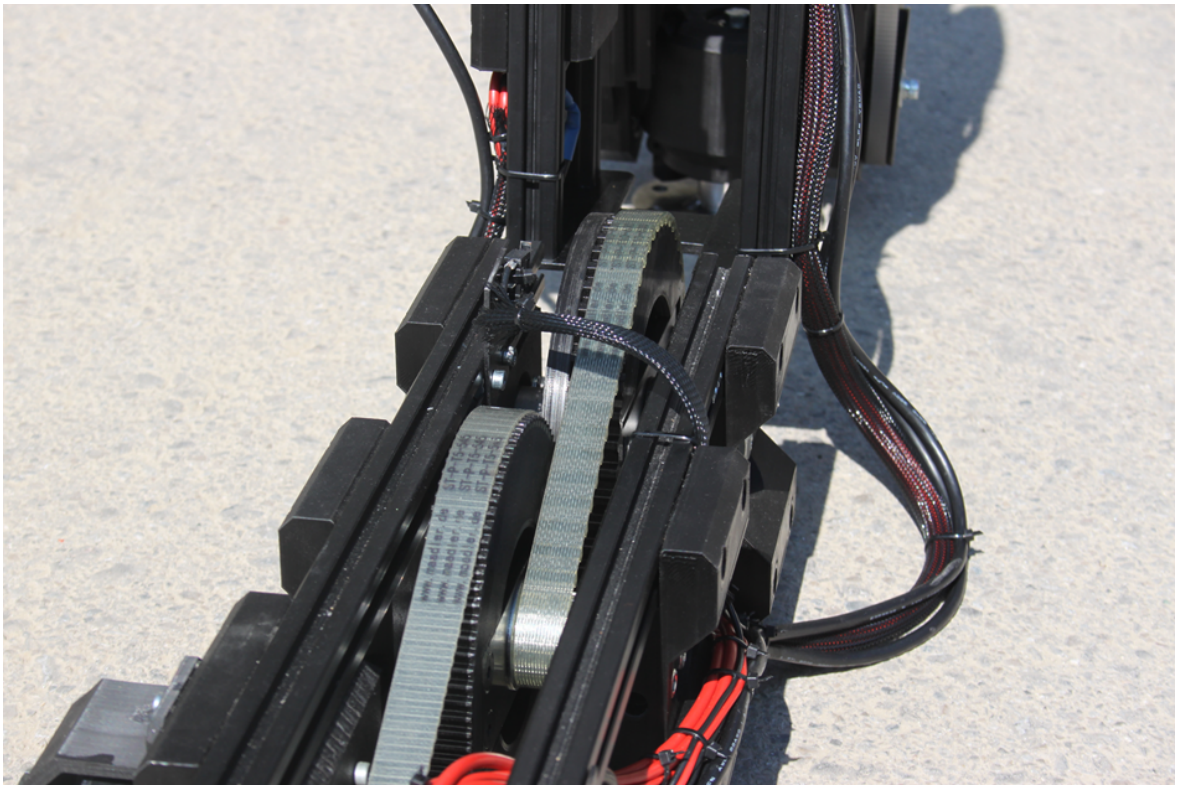


Figure 14.2: The drive system of the first axis

were printed with 60% infill. If they were to break again in the future, the pulleys could be printed with 100% infill, which means they would be solid plastic.

14.4 The wrist joint

The wrist joint is by far the most complex part of the entire robot arm. The universal adapter plate which represents the Tool Attachment Point (TAP) of the robot arm is held with two ball bearings as seen in Figure 14.4.

To allow the plate to be tilted and rotated around its own axis with two stationary motors, a gear box was developed. It holds the axis 4 and drives it with bevel gears.

As discussed in previous chapters, the large bevel gear was sourced from Mädler[1]. The entire stack consisting of the gear, two bearings, the washer and the bushing is clamped together by the M8 bolt. The gear is not secured against rotating any further, the friction created by the clamped stack is enough to keep it from rotating.

[1] *Mädler*

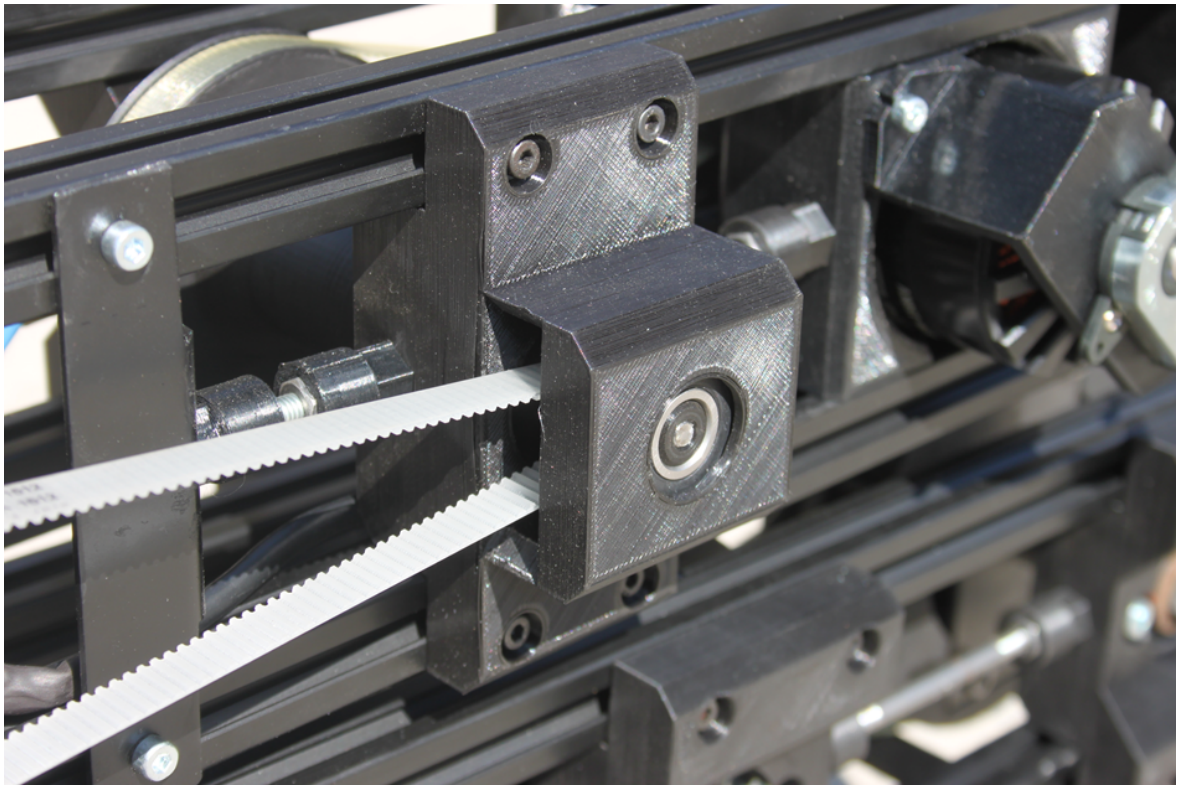


Figure 14.3: The timing belt and the pulley's ball bearing of axis 3

Exactly the same is the case for the small bevel gear, shown in Figure 14.5. The small bevel gear is clamped onto the shaft with an M6 nut, which pushes the gear against the inner ring of the small bearing. Friction is enough to keep it from rotating.

Both bevel gears are held in place in a 3D printed housing, shown in Figure 14.6. This allows the entire housing to be tilted up and down, while the adapter plate can still be rotated.

The housing is divided in two halves, so that it can be 3D printed and assembled. To do that, both gear stacks shown in Figure 14.6 are assembled separately and placed in one half of the housing. Then the second half is put on and the large bearings are slid on on the outside of the housing. This can be seen in Figure 14.7. The large bearings allow the entire housing to rotate up and down.

In Figure 14.7 the small drive shaft of the fourth axis can be seen, it is what drives the adapter plate's rotation. A 3D printed pulley is mounted on it (not visible in Figure 14.7). The torque is very low, so no additional protection against rotation is needed. The pulley is clamped against the inner ring of the small bearing with an M6 screw, the friction is enough to keep it from rotating.

The housing is rotated up and down with another 3D printed pulley, which is mounted on

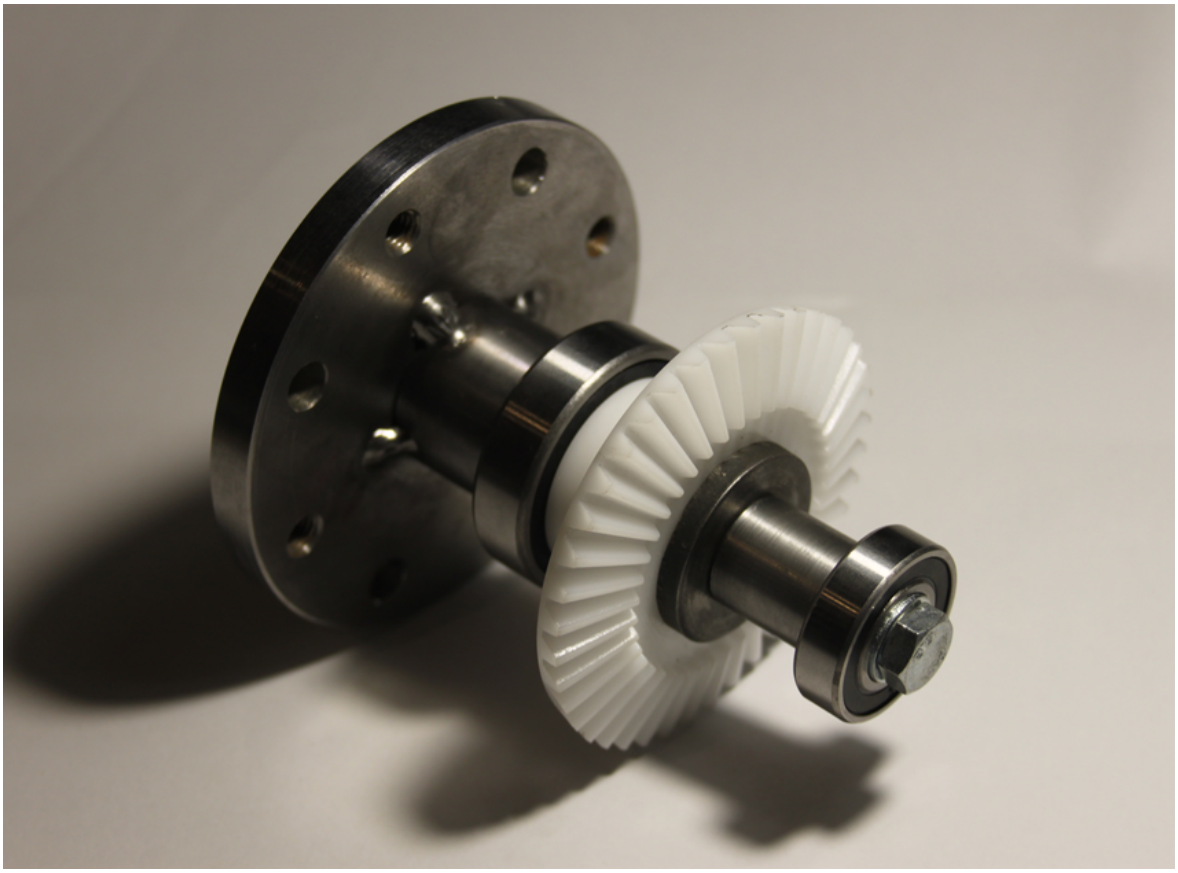


Figure 14.4: Universal adapter plate of the robot's wrist joint

the opposite side. This can be seen in Figure 14.8. It shows one half of the housing and the pulley. It is screwed onto the side of the two halves with four M5 bolts. The nuts are pushed into the 3D printed housing in hexagonal pockets. Once pushed in they do not slip out, which means the pulley can always be screwed on from the outside without holding nuts on the inside.

14.5 Encoders and belt tensioners

Figure 14.9 shows how the AMT CUI encoders are mounted on the ODrive motors. They cannot be mounted on the motor directly because the motors are outrunners. This means the stator is on the inside and the rotor is on the outside, which means the entire housing with the ODrive logo seen in Figure 14.9 rotates. Therefore a 3D printed bracket was made to mount the encoder on the back of the motor, while the motor's housing can still rotate freely.

Figure 14.5: The small bevel gear of the wrist joint

Figure 14.9 also shows the belt tensioners again, they consist of the threaded rod and the two nut holders, the belt can be tensioned by rotating the two nut holders against each other.

14.6 Limit switches

End stops are required on every axis because the ODrive encoders are incremental. Even though they have an index mark it is not enough to let the robot know at which angle the joint is rotated after startup. Because of that endstops must be used, which means the robot can home itself on startup. It moves each axis until the limit switch is pressed and then it knows exactly where the axis is located.

Figure 14.10 shows the limit switches of axes 3 and 4. The limit switch with the straight metal lever in the center of the assembly is for the third axis. It is pressed whenever the adapter plate rotates up far enough to press it with the outermost corner of the round plate.

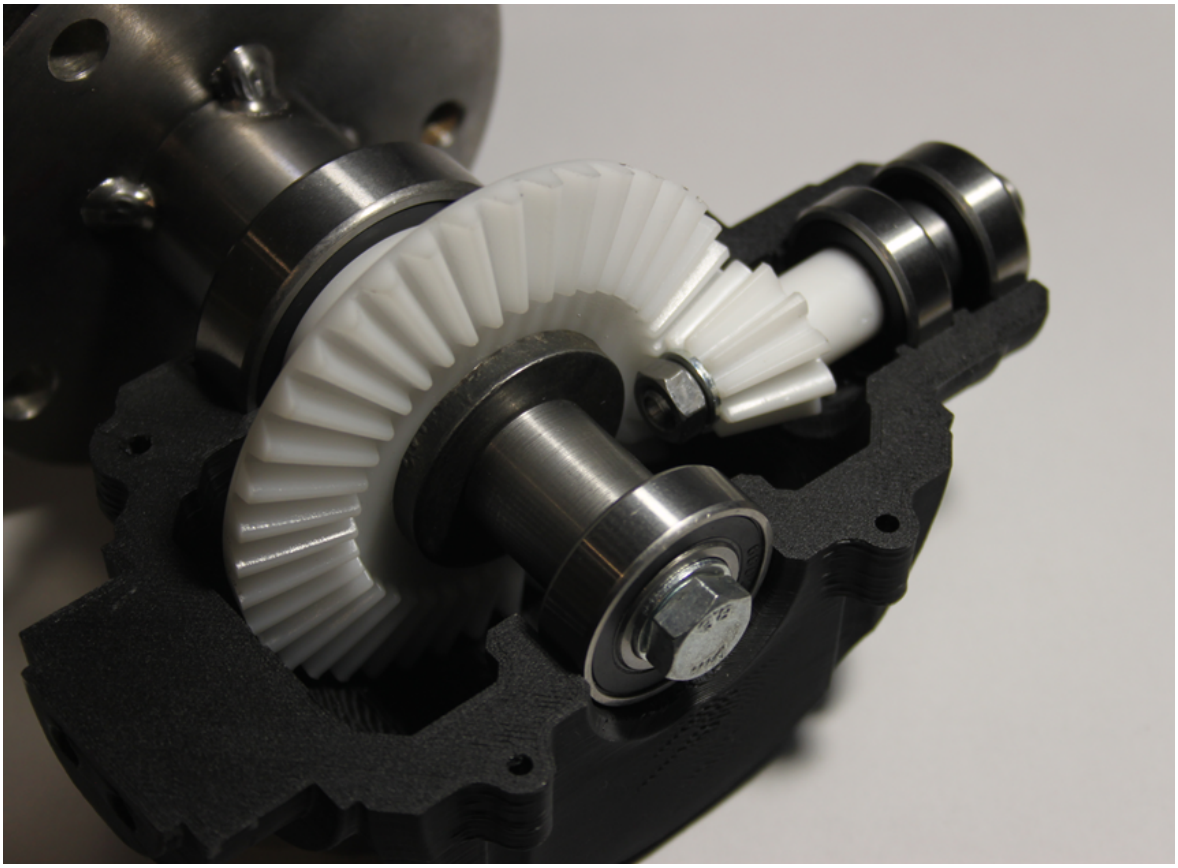


Figure 14.6: Inner workings of the wrist joint

Figure 14.11 shows the limit switch of axis 4. This limit switch is special. All other limit switches are made so that the axis can move to the limit switch for homing and then it can move within a limited range. This is different for the fourth axis.

The lever was bent so that the screw can move over it entirely from both sides. Homing is done so that the axis is rotated in one direction until the limit switch is reached. Then it rotates back to the initialization position and the limit switch is disabled. After that the axis can be moved indefinitely in both directions without restrictions and pressing the limit switch after homing does not do anything. Keeping the limit switch enabled in the ODrive would cause the ODrive to immediately halt on error when the limit switch is pressed after homing.

Additionally, the adapter plate can be seen in detail in Figure 14.10. It has eight holes, while four of them are 8mm through-holes and the other four are M8 threads. By using the four M8 threads anything can be attached to the adapter plate easily, the through holes allow the bolts to be inserted from the back in case the manipulator does not have space for bolt heads, but bolts inserted from the back are less accessible. Using bolts from the front in the

Figure 14.7: Assembled wrist joint with all bearings and shafts

M8 threads is the easiest way to mount manipulators.

14.7 Wiring

Figure 14.12 shows the wiring. The wires are bound to the frame in large loops so that the motion of the axis is not blocked. This is the case for all axes.

Without electronics this entire robot arm would be useless. The two ODrive boards can be seen in Figure 14.13. Very thick wires are used to allow maximum current for the motors and everything is held in place neatly with zip ties.

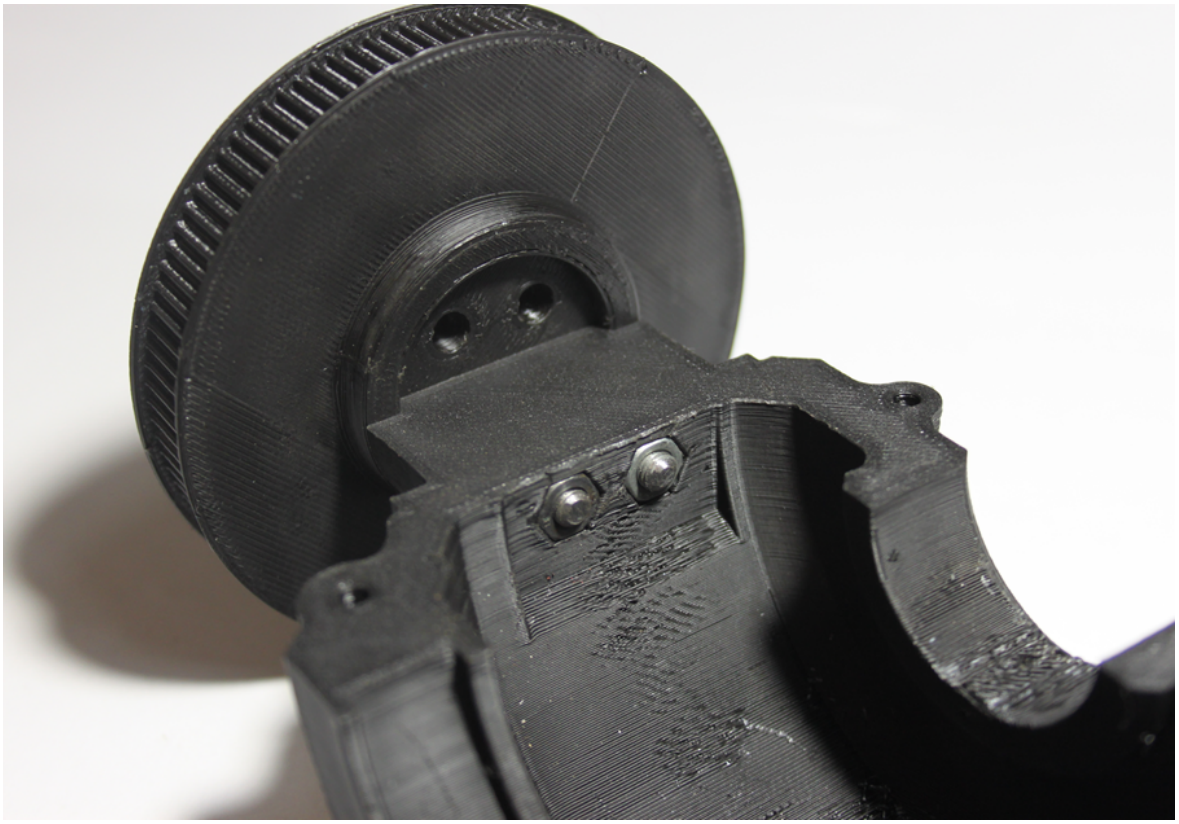


Figure 14.8: How the pulley is mounted with embedded nuts

14.8 Arm in motion

A robotic arm is obviously supposed to move. Figure 14.14 shows the extended arm. The wires simply bend to the side and do not touch anything where motion could be blocked.

As discussed above, the process of mounting the arm on the Festo Robotino® is not documented in this diploma thesis.

14.9 Side-by-side comparison

To end this chapter, a comparison between the 3D model from Creo and the real robot is shown. Figure 14.15 shows the Creo model on the left and the real robot on the right.

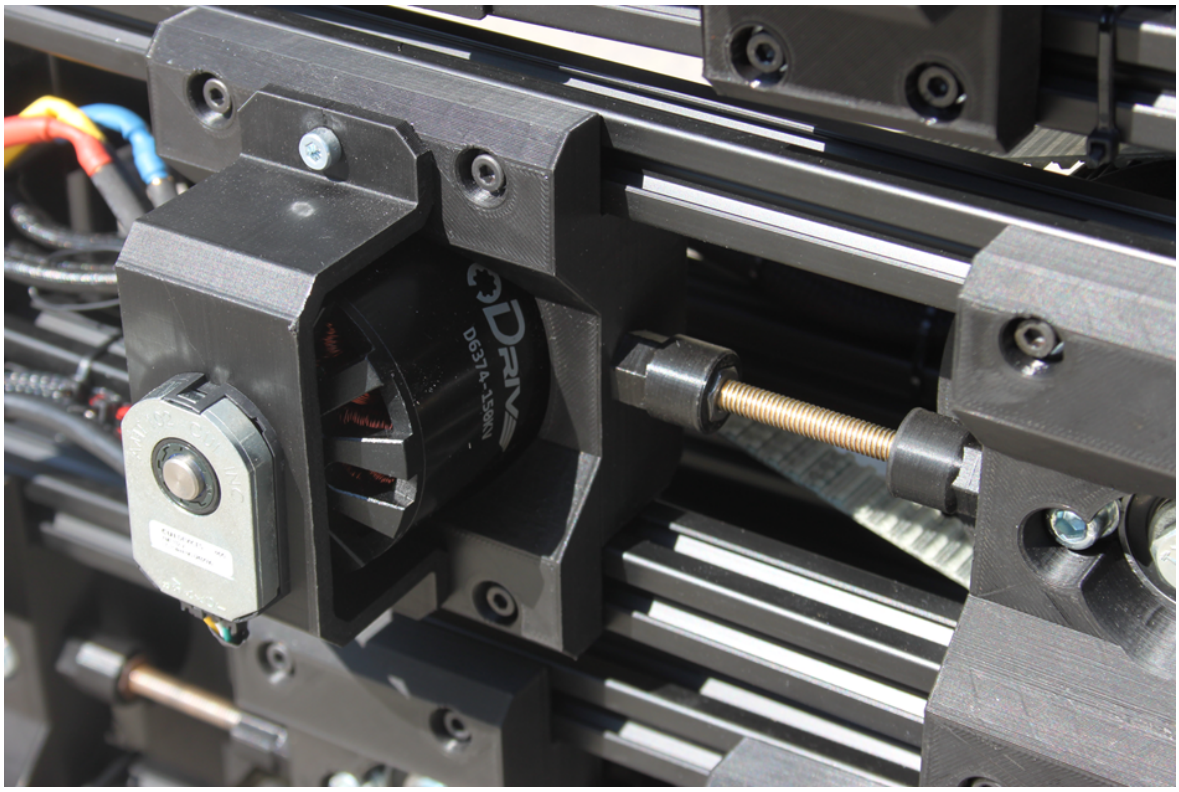


Figure 14.9: ODrive Encoder holders and the belt tensioners

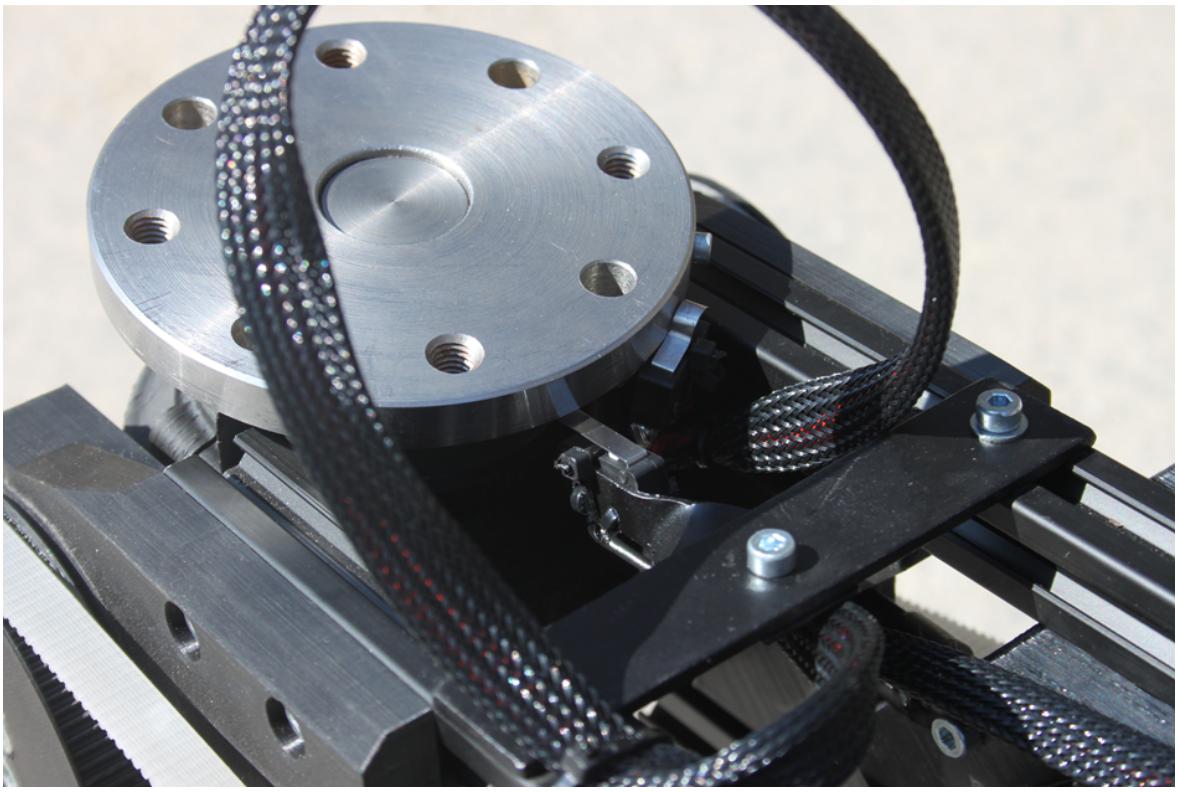


Figure 14.10: Limit switches of axes 3 and 4

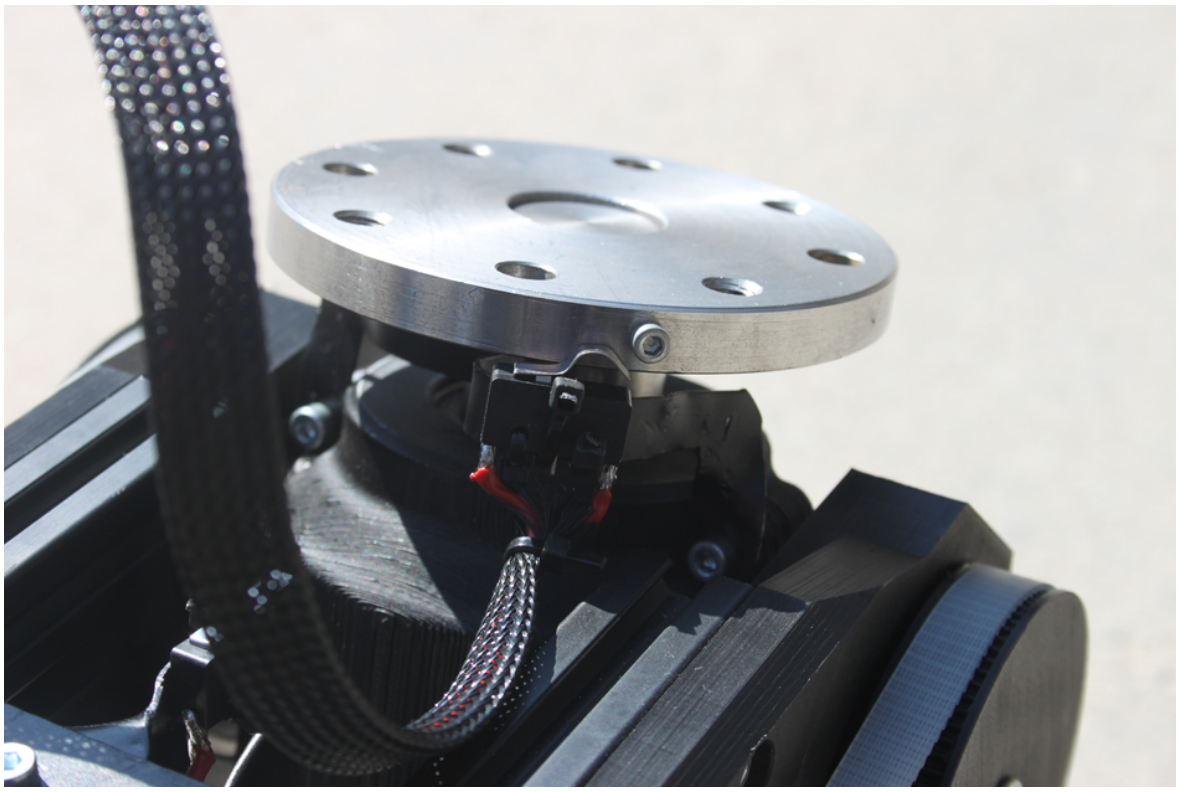


Figure 14.11: Limit switch of axis 4

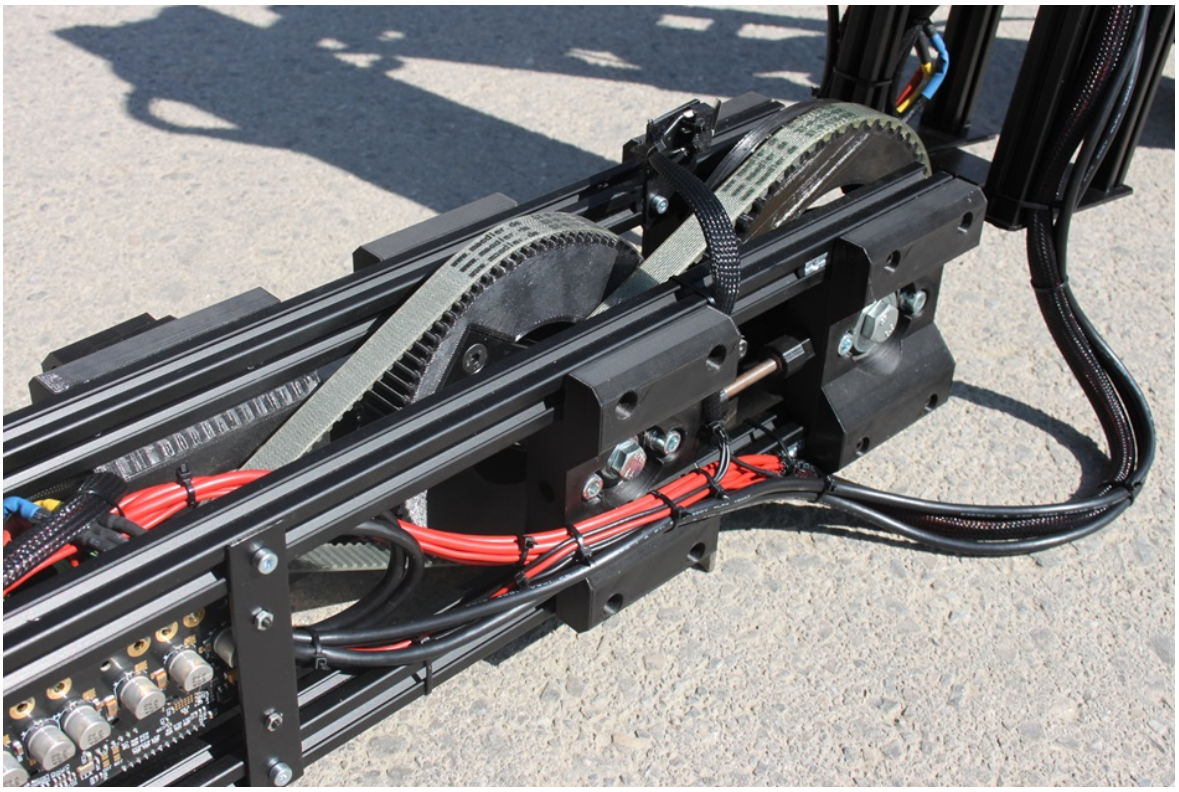


Figure 14.12: Wiring and timing belts of axis 1

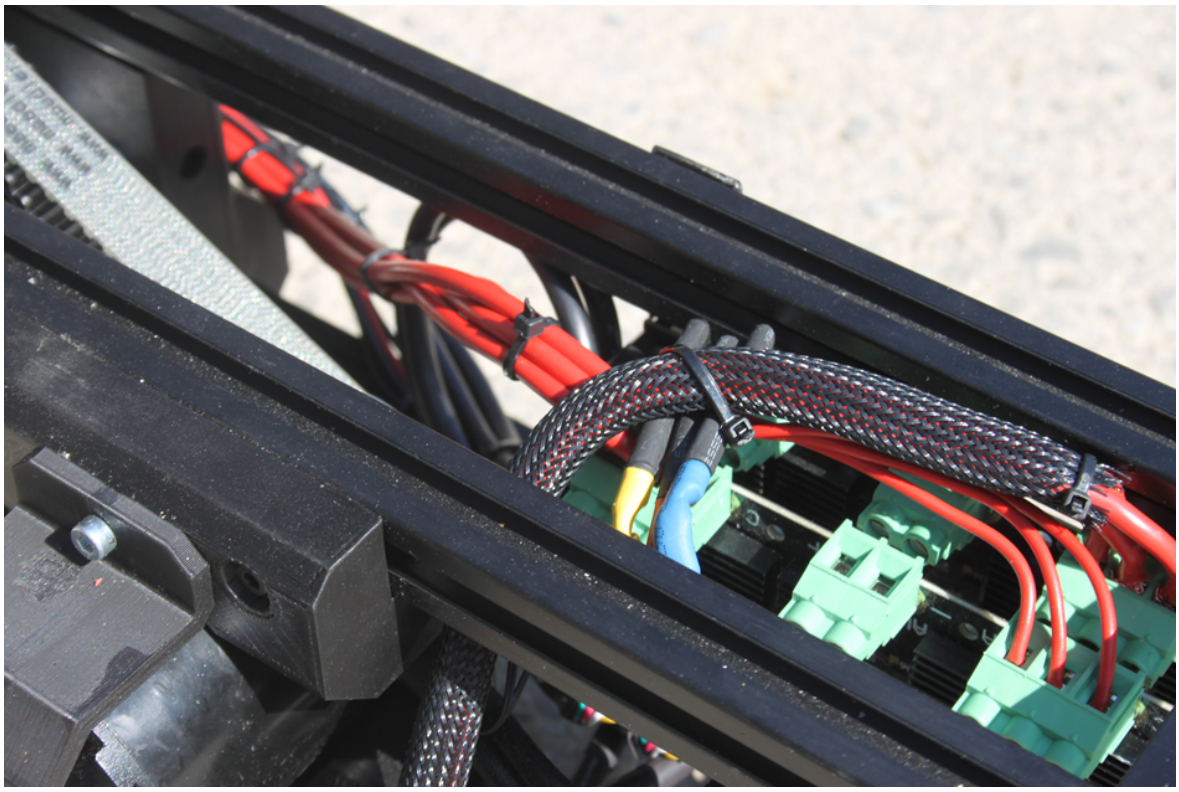


Figure 14.13: Electronics of the robot arm

Figure 14.14: The arm when it is partially extended

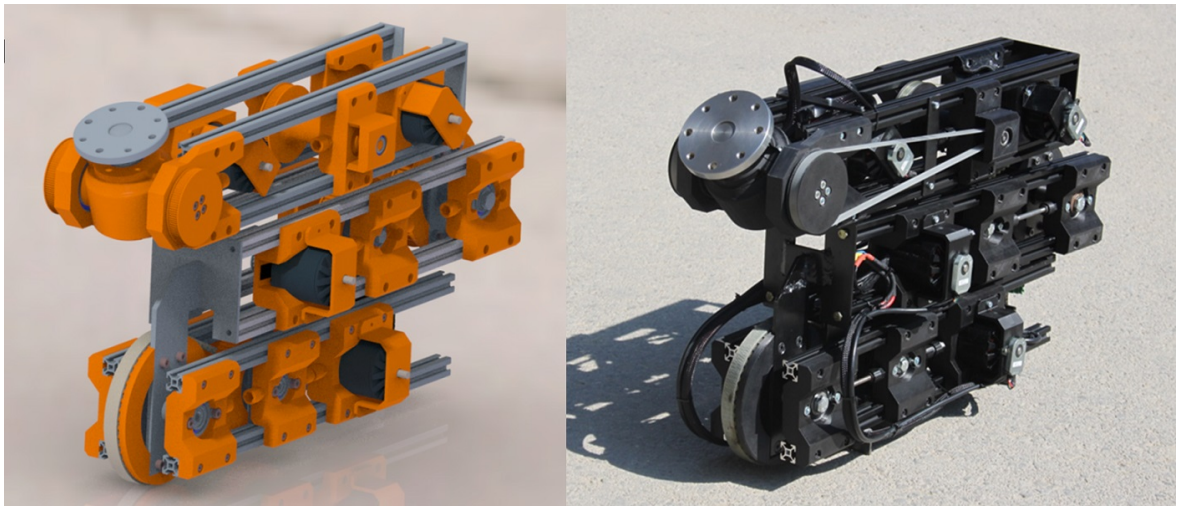


Figure 14.15: Side-by-side comparison of the 3D model and the final assembly

Chapter 15

Conclusion

As part of this diploma thesis, the flexible robot arm for mobile expandable systems was developed to fill a gap in the market and to provide a suitable solution for F-WuTS. Proving some initial concerns wrong, the collaboration between the mechanical engineering and IT departments went very well and confirmed that a diploma thesis across departments is not a problem. The possibility of cross-collaboration between the mechanical engineering and IT departments is kept in mind and might allow entirely new possibilities in the future.

15.1 Development

A large amount of time was spent on understanding ODrive and establishing errorless real-time communication between the components. It was trickier than expected as no comparable project had been done before, leaving the authors with no base to work on. In case others want to start a similar project, git repositories with all modules have been created to give them a jumpstart.

15.1.1 ODrive Native Library

The ODrive Native Library is responsible for sending and parsing ODrive Native Packets. The Arduino firmware utilizes this library to connect the ODrive boards with a PC or the Robotino. However, due to problems with specific ODrive firmware versions, the endpoint defines are fixed and would not work with newer versions. Automatically retrieving the endpoint definition from the ODrive board would solve this issue and may be implemented in the future.

15.1.2 Kinematics

The movement of the FLAME robotic arm depends on the kinematics. With several mathematical equations the movements of the axes are calculated. However, the inverse kinematics of the arm is simplified to a two-dimensional kinematics, because the calculations for three-dimensions would be more complex.

15.1.3 FLAME Library

The FLAME Library is responsible for the communication between the Arduino and the user PC. To control the robot arm a protocol with UDP packets is designed. The communication between the Arduino and the PC went smoothly.

15.1.4 Prototypes

In Creo Parametric, two prototypes of the robot arm were designed. Initially, it seemed like more prototypes would be needed and that more prototypes would also need to be built. The first prototype was modeled and then scrapped and the second Creo design was actually the final version. The second prototype was then manufactured and assembled and no additional changes were needed, surprisingly.

However, it showed that designing several prototypes is absolutely necessary and the first design is almost never the best.

15.1.5 Mechanical design

The entire robotic arm was modeled in Creo Parametric 7 from the ground up. There were no problems with Creo itself, everything went smoothly and it was a good choice to use Creo. It was also a very good decision not to use Creo Windchill, because in previous projects we noticed that Windchill causes a lot of hassle that simply does not exist when staying with the Creo Parametric offline version.

Another problem that showed up when exporting models from Creo for 3D printing was the resolution. The default exporting resolution is far too low to be usable for 3D printing. The model can be exported for 3D printing by clicking *Als Kopie speichern* and setting the filetype to *.3MF*, but the tick button *Export anpassen* must be ticked. Then a popup will appear and the first two values were set to 0, which corrects them to the lowest allowed value and the third value was enabled and set to 1mm. This results in usable 3MF files, but this must be done for every single model.

The second pitfall was the skeletal sketch. It was a very good choice to use master and sub-skeletal sketches, but it could have been easier. The entire project was set up such that the angle of each axis is defined in the master skeleton. This means that the angle of each arm can be changed in the master skeleton and the entire model adapts to that, which means that it can be modeled at any angle, but it also means that all sketches are quite complex because they need to work at any angle.

It would have been a lot easier if all axes were fixed and the entire arm was modeled in its collapsed state. When the motion must be animated, the entire model is assembled a second time with dynamic constraints anyways, so there would be one main unmovable assembly for creating the design and another dynamic assembly which is used for simulation and animation. Trying to do both at the same time is unnecessarily complex.

15.1.6 Manufacturing

Many parts were manufactured in the mechanical workshop of the HTL Wiener Neustadt, the drawings of all of these can be found in the appendix of this diploma thesis. Everything went smoothly and surprisingly all parts fitted perfectly without modification. However, it turned out that the mechanical workshop is not exactly the most efficient and took a lot longer than anticipated to manufacture all parts. Next time, at least four times as much time should be planned for manufacturing to still have freedom if something goes wrong. It was the main limiting factor for getting the project done in time.

15.1.7 Assembly

Assembly went well too, all of the modeling work was absolutely worth it. However, the limit switches were not included in the 3D model, instead they were added afterwards with hand-bent sheet metal. As a result, they are not an integral part of the arm and might cause problems in the future. We should have put in the extra effort of integrating them properly with 3D printed brackets.

15.2 Outlook

The authors hope that this diploma thesis inspires future students to dive into the world of robotics. Moreover, FLAME provides a perfect starting point for future projects like implementing a graphical user interface that could utilize more sophisticated Inverse Kinematics or developing new end effectors like a pneumatic gripper or a gripper with image recognition. We hope that any future users or developers have a great time while working with the FLAME robot.

Index

Authors Index

Literature Index

Bibliography

4.1 Forward Kinematics. URL: https://youtu.be/EzNAs2w1cS0 (visited on 03/16/2022).

5.1 Inverse Kinematics. URL: https://youtu.be/RH3iAmMsolo (visited on 03/16/2022).

Arduino-Mega-2560.PNG (source: reichelt.at). URL: https://cdn-reichelt.de/bilder/web/artikel_ws/A300/ARDUINO_MEGA_01_NEU.jpg (visited on 01/26/2022).

Arduino-Tinkerkit-Robot.PNG. URL: https://at.rs-online.com/web/p/shields-fur-arduino/1113738 (visited on 12/09/2021).

Badamasi, Yusuf Abdullahi. "The working principle of an Arduino". In: *2014 11th international conference on electronics, computer and computation (ICECCO)*. IEEE. 2014, pp. 1–4.

Beckhoff. URL: https://infosys.beckhoff.com/index.php?content=../content/1031/tcplccontrol/html/tcplcctrl_intro.htm&id= (visited on 03/26/2022).

Blender tutorial by "Imphenzia" (source: Youtube). URL: https://youtu.be/1jHUY3qoBu8 (visited on 02/21/2022).

Broadcast. URL: https://www.placetel.de/ratgeber/broadcast.

Bruno Siciliano, Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag, 2008.

Dimitris Alimisis, Chronis Kynogos. *Constructionism and robotics in education*. URL: https://roboesl.eu/wp-content/uploads/2017/08/chapter_1.pdf (visited on 09/23/2021).

DrawIO. URL: https://app.diagrams.net (visited on 03/26/2022).

Epping, Thomas. *Kanban für die Softwareentwicklung*. Springer-Verlag, 2011.

Ethernet-Module.PNG (source: amazon.de). URL: https://m.media-amazon.com/images/I/41PSTOzIQ0L._AC_SX450_.jpg (visited on 01/26/2022).

European Space Agency. *Robotics for Society*. URL: https://business.esa.int/funding/invitation-to-tender/robotics-for-society (visited on 09/18/2021).

Example of timing belts (KUKA). URL: https://www.youtube.com/watch?v=HXJOnWBbcwM (visited on 01/16/2022).

Festo-Didactic. *Robotino - Mobile robot plattform for research and testing*. Last accessed 26 January 2022. 2022. URL: https://www.festo-didactic.com/ov3/media/customers/1100/robotino_brochure_en_56940_2013_10_monitor.pdf.

festoRobotino.PNG (source: festo-didactic.com). en. URL: https://www.festo-didactic.com/ov3/media/customers/1100/robotino_brochure_en_56940_2013_10_monitor.pdf (visited on 02/02/2022).

Figma. URL: https://www.figma.com (visited on 03/26/2022).

Flaticon. URL: https://www.flaticon.com (visited on 03/26/2022).

GitHub. URL: https://github.com (visited on 03/26/2022).
Greg Nichols. *robotic_surgery-0.JPG*. URL: https://www.zdnet.com/a/hub/i/2021/07/22/b9692a93-addc-4f82-b029-cd9d9a68831b/robotic-surgery-0.jpg (visited on 09/18/2021).
Hoda, Rashina, James Noble, and Stuart Marshall. "Agile project management". In: *New Zealand computer science research student conference*. Vol. 6. 2008, pp. 218–221.
igus online configurator for 3D printing. URL: https://www.igus-cad.com/default.aspx (visited on 02/21/2022).
Joe Pappalardo. *54ca5599efee7_-_curiosity-0612-de.JPG*. URL: https://hips.hearstapps.com/pop.h-cdn.co/assets/cm/15/05/54ca5599efee7_-_curiosity-0612-de.jpg?crop=1xw:1.0xh;center,top&resize=980:* (visited on 09/18/2021).
KIPR Website. en. 2021. URL: https://www.kipr.org/gcer (visited on 09/23/2021).
Konstantin Lampalzer. *robotino.JPG (JPEG*. URL: https://robo4you.at/static/ebfc70c7c6085b3f9bfd67457/robotino.jpg (visited on 10/07/2021).
Mädler. URL: https://www.maedler.de (visited on 01/16/2022).
Molkenthin, Bastian. *Understanding and implementing CRC calculation*. Feb. 2015. URL: http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html.
NEMA17-StepperMotor.PNG. URL: https://www.ato.com/nema-17-bipolar-stepper-motor-2-8v-1-68a-2-phase-4-wires (visited on 12/05/2021).
ODrive Control. en. URL: https://docs.odriverobotics.com/control (visited on 11/04/2021).
ODrive Docs Tool. en. URL: https://docs.odriverobotics.com/odrivetool (visited on 10/21/2021).
ODrive Docs Wiring. en. URL: https://docs.odriverobotics.com/#downloading-and-installing-tools (visited on 10/21/2021).
ODrive Endstops. en. URL: https://docs.odriverobotics.com/endstops (visited on 11/04/2021).
ODrive-ASCII-Prptocol. en. URL: https://docs.odriverobotics.com/ascii-protocol (visited on 11/04/2021).
ODrive-Control-Structure.PNG (source: docs.odriverobotics.com). URL: https://docs.odriverobotics.com/controller_with_ff.png (visited on 11/04/2021).
ODrive-Native-Protocol. en. URL: https://docs.odriverobotics.com/native-protocol (visited on 01/17/2022).
ODrive-Website.PNG (source: docs.odriverobotics.com). en. URL: https://odriverobotics.com/ (visited on 10/16/2021).
Oskar Weigl. *Hobby Motors For Robotics*. en. URL: https://odriverobotics.com/ (visited on 10/16/2021).
Pneumatic-Robot-Arm.JPG (source: www.teamcassracing.top). URL: https://i.pinimg.com/originals/63/65/35/636535dd11529cdbdf03fc7dacc479ea.jpg (visited on 12/05/2021).
Premake. URL: https://premake.github.io/docs/What-Is-Premake/ (visited on 11/09/2021).
PTC-Creo-Logo.PNG (source: Wikipedia). URL: https://upload.wikimedia.org/wikipedia/commons/thumb/d/df/PTC_Creo_logo.svg/1200px-PTC_Creo_logo.svg.png (visited on 11/02/2021).

Robot-Dog.PNG (source: James Bruton, YouTube). URL: https://youtu.be/SgJcj2Gd0Rc (visited on 12/05/2021).

Robotic Industries Association. *unimate_1-9518fc74.JPG*. URL: https://www.ke-next.de/assets/images/4/unimate_1-9518fc74.jpg (visited on 09/18/2021).

Robotino OS. en. URL: https://wiki.openrobotino.org/index.php?title=Robotino_OS (visited on 02/02/2022).

robotino-example.PNG (source: festo-didactic.com). en. URL: https://www.festo-didactic.com/ov3/media/customers/1100/robotino_brochure_en_56940_2013_10_monitor.pdf (visited on 02/02/2022).

ROS. URL: https://www.ros.org (visited on 03/30/2022).

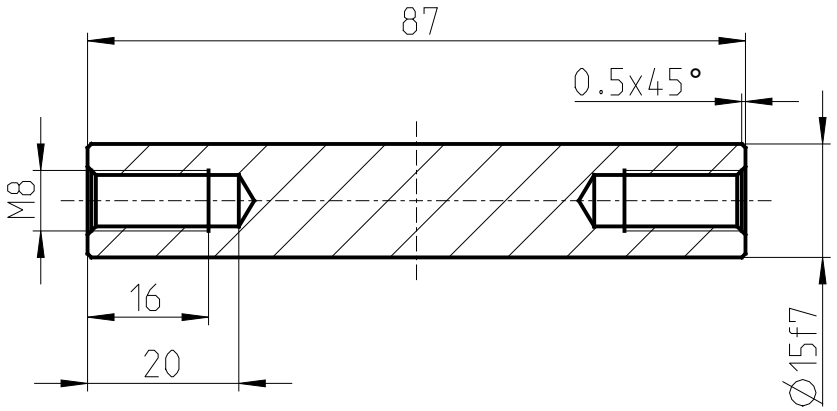
RPY. URL: https://de.wikipedia.org/wiki/Roll-Nick-Gier-Winkel (visited on 03/26/2022).

Schubert, Thomas W, Alessandro D'Ausilio, and Rosario Canto. "Using Arduino microcontroller boards to measure response latencies". In: *Behavior research methods* 45.4 (2013), pp. 1332–1346.

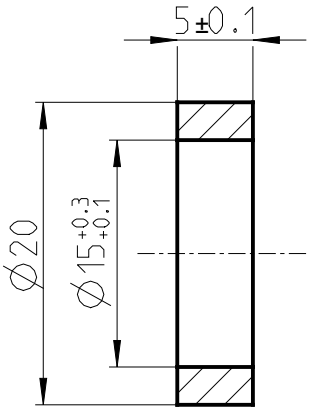
Unger-Leinhos, Angela. *Die Geschichte der Robotertechnik begann 1954*. URL: https://www.ke-next.de/robotik/die-geschichte-der-robotertechnik-begann-1954-108.html (visited on 09/18/2021).

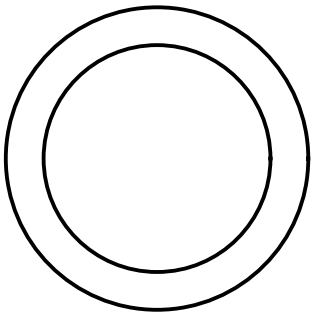
Williams, Ross N. *A painless guide to CRC error detection algorithms*. Aug. 1993. URL: http://www.ross.net/crc/download/crc_v3.txt.

Appendix



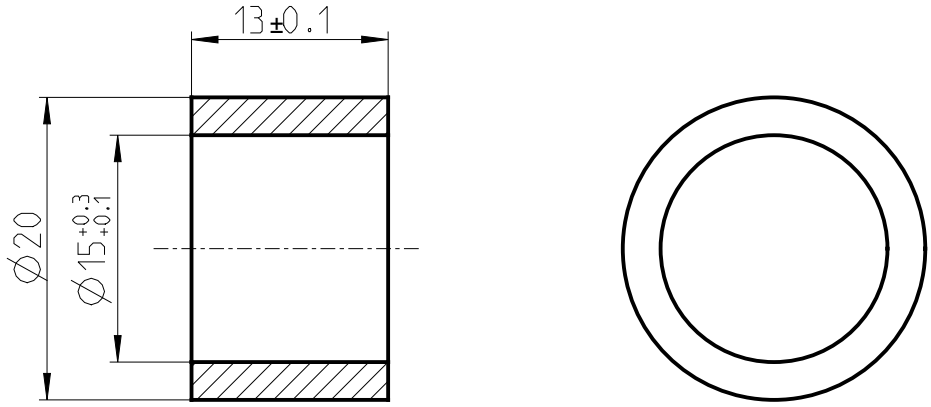
Dok. Art / Doc. Type -	Masse / Mass 0.11 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 04.12.2021	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00001-01	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title ZW WELLE_A1 -		Tolerances DIN ISO 2768-mk - - -





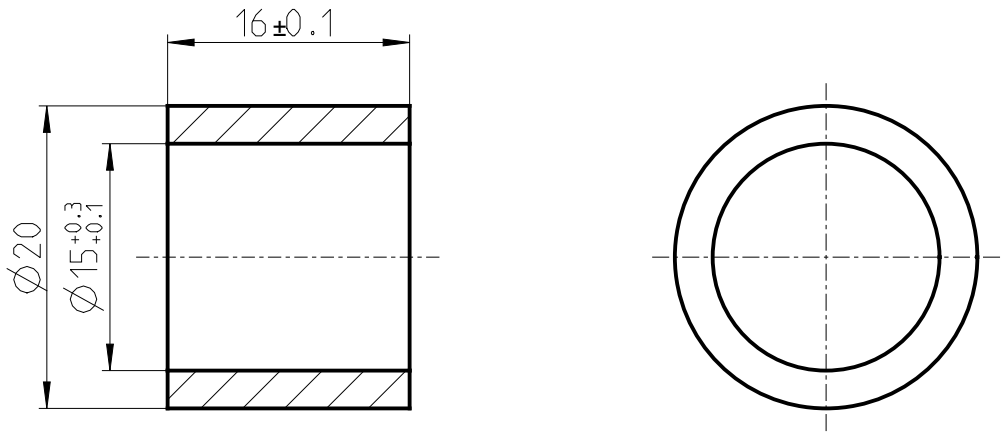
Alle Kanten 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.01 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 04.12.2021	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 2:1	Sach Nr. / Part No. 21T01-22-00001-10	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title A1_ZW_DISTANZH2 -		Tolerances DIN ISO 2768-mk - - -



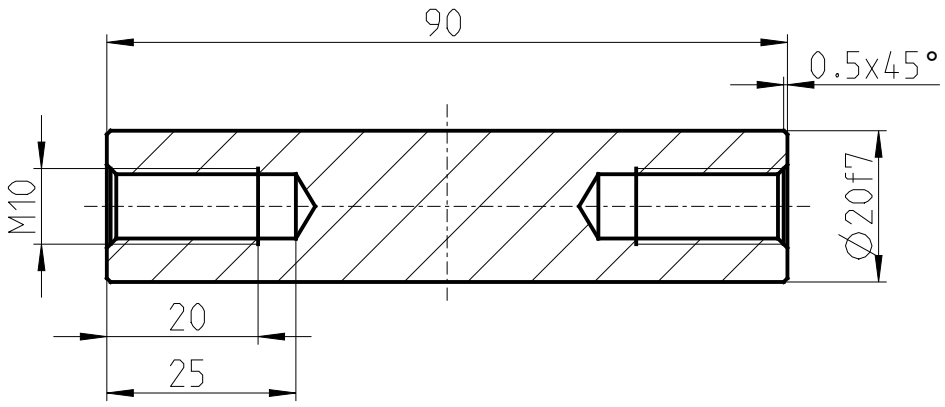
Alle Kanten 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.01 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 04.12.2021	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 2:1	Sach Nr. / Part No. 21T01-22-00001-11	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title A1_ZW_DISTANZH3 -		Tolerances DIN ISO 2768-mk - - -

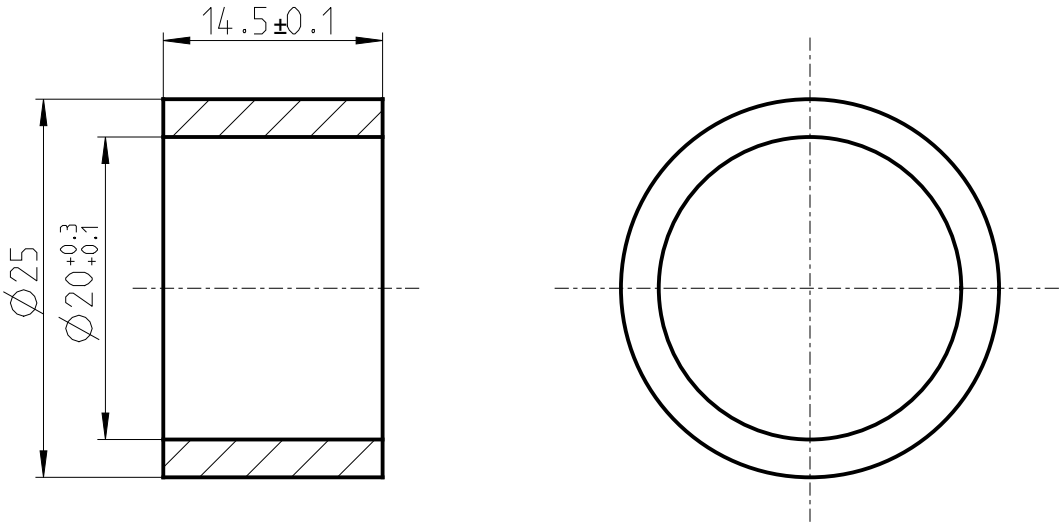


Alle Kanten 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.02 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 04.12.2021	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 2:1	Sach Nr. / Part No. 21T01-22-00001-09	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title A1_ZW_DISTANZH1 -		Tolerances DIN ISO 2768-mk - - -

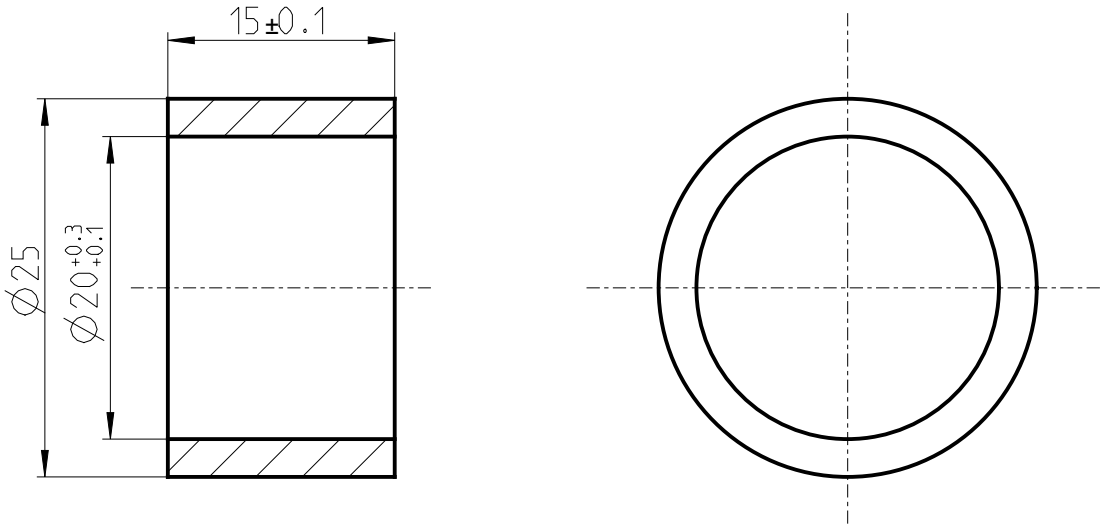


Dok. Art / Doc. Type -	Masse / Mass 0.20 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 04.12.2021	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00002-03	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title HAUPTWELLE_A1 -		Tolerances DIN ISO 2768-mk - - -



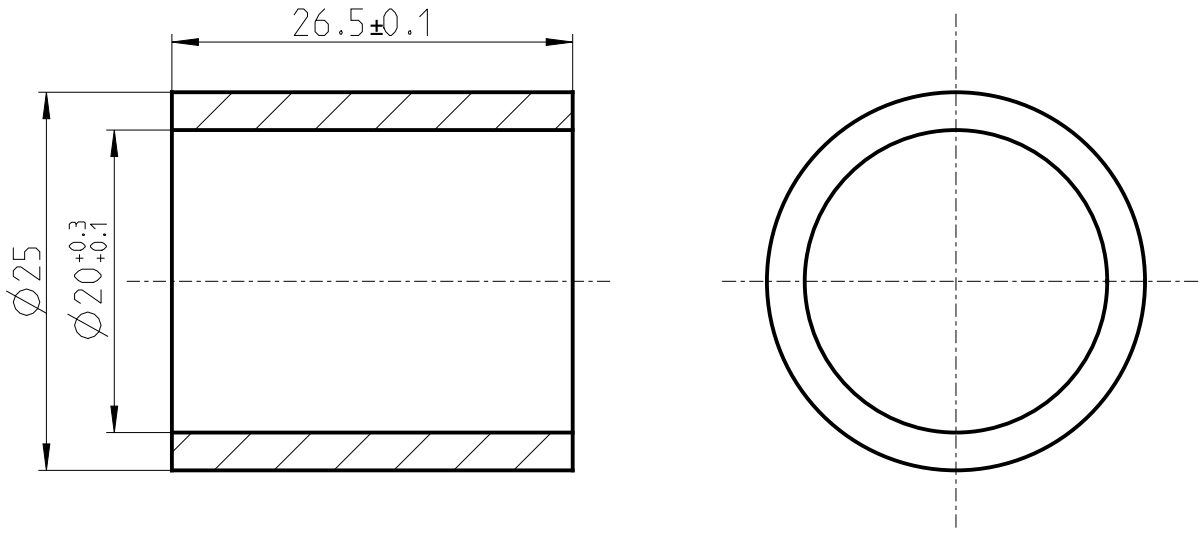
Alle Kanten 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.02 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 04.12.2021	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 2:1	Sach Nr. / Part No. 21T01-22-00002-08	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title A1_DISTANZH2 -		Tolerances DIN ISO 2768-mk - - -



Alle Kanten 0.5x45° angefast

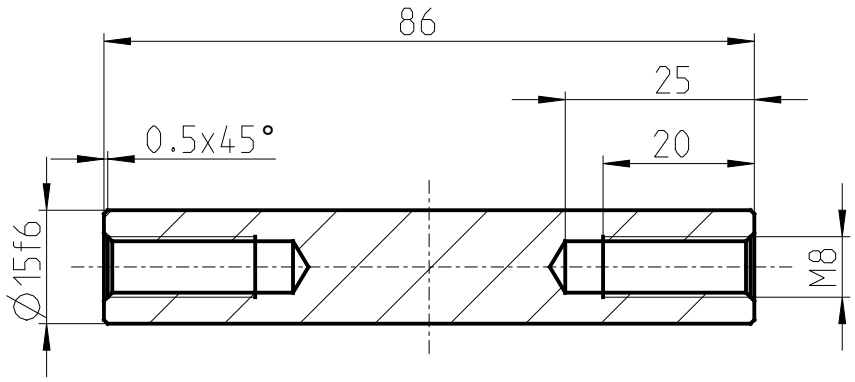
Dok. Art / Doc. Type -	Masse / Mass 0.02 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 04.12.2021	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 2:1	Sach Nr. / Part No. 21T01-22-00002-09	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title A1_DISTANZH3 -		Tolerances DIN ISO 2768-mk - - -



Alle Kanten 0.5x45° angefast

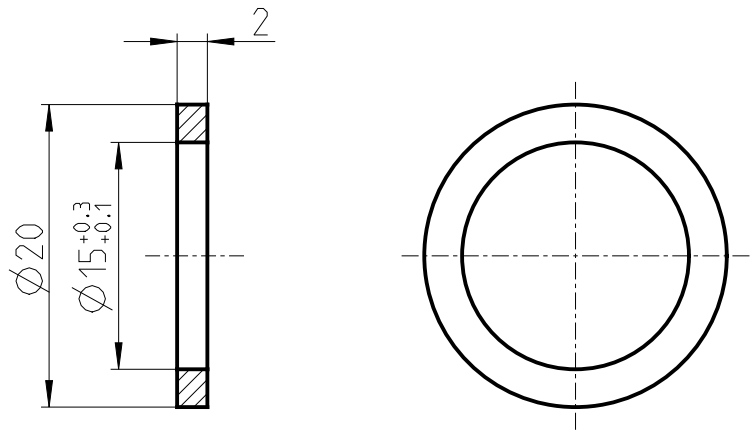
Dok. Art / Doc. Type -	Masse / Mass 0.04 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 04.12.2021	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 2:1	Sach Nr. / Part No. 21T01-22-00002-07	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title A1_DISTANZH1 -		Tolerances DIN ISO 2768-mk - - -

A-A



Dok. Art / Doc. Type -	Masse / Mass 0.10 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00002-21	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title ZwWelle Achse 2 -		Tolerances DIN ISO 2768-mk - - -

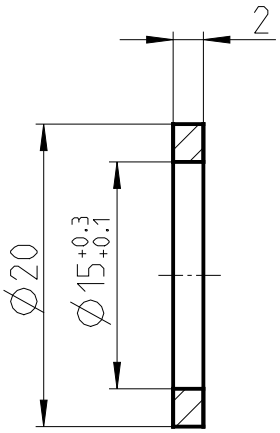
A-A
2:1

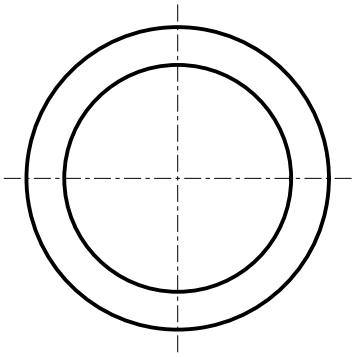


Alle Kanten mit 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.00 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00002-20	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title Distanzring 3 Achse 2 -		Tolerances DIN ISO 2768-mk - - -

A – A
2:1

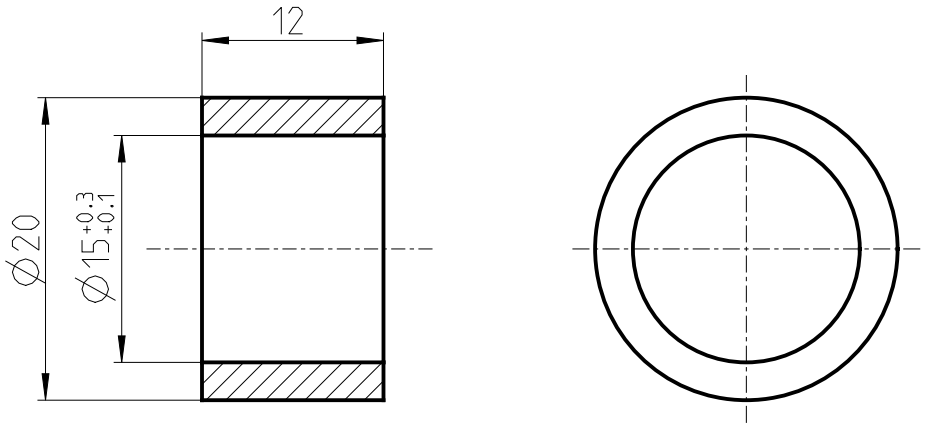




Alle Kanten mit 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.00 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00002-23	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title Distanzring 2 Achse 2 -		Tolerances DIN ISO 2768-mk - - -

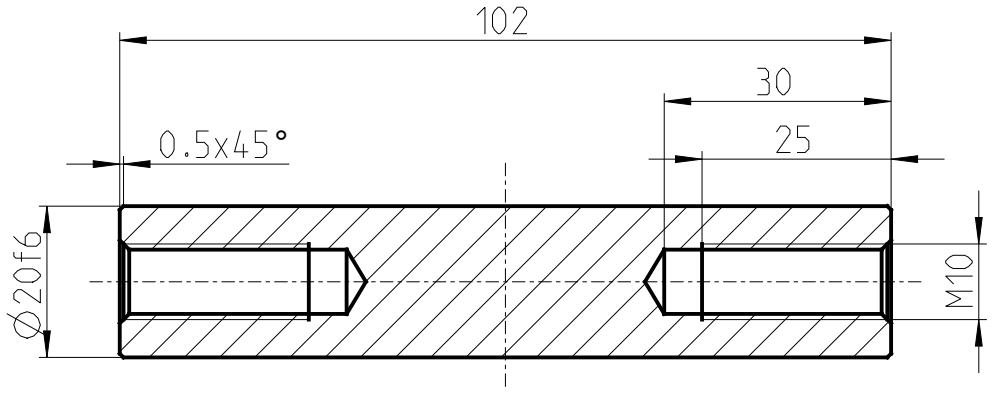
A – A
2:1



Alle Kanten mit 0.5x45° angefast

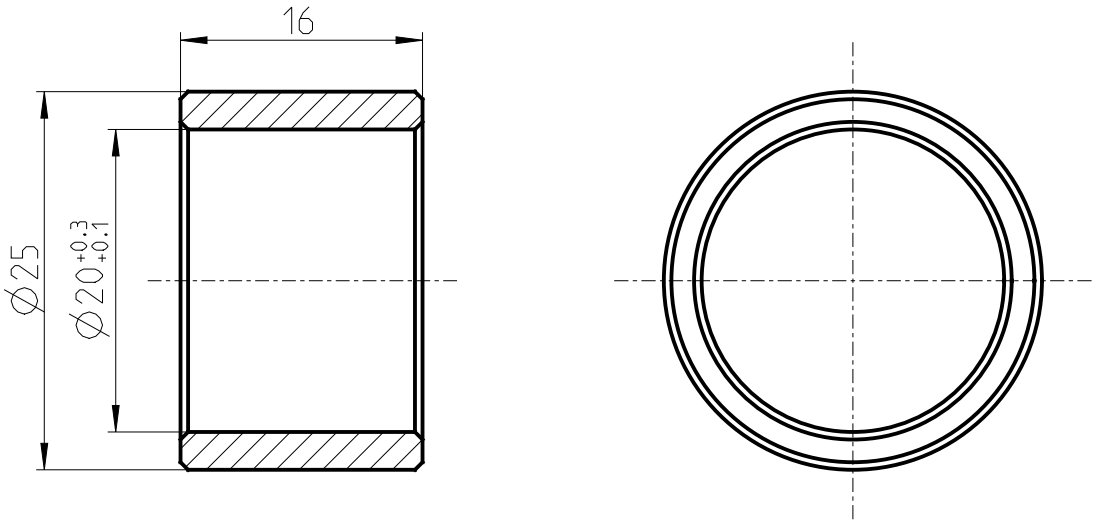
Dok. Art / Doc. Type -	Masse / Mass 0.01 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00002-22	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title DistanzH 1 Achse 2 -		Tolerances DIN ISO 2768-mk - - -

A-A



Dok. Art / Doc. Type -	Masse / Mass 0.22 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00003-06	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title Hauptwelle Achse 2 -		Tolerances DIN ISO 2768-mk - - -

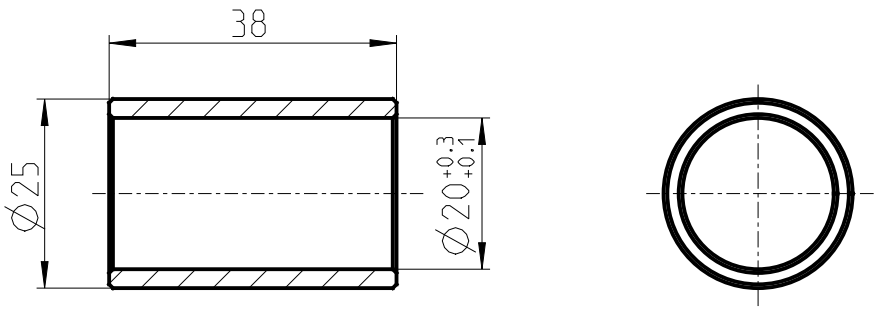
A-A
2:1



Alle Kanten mit 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.02 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00003-08	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title DistanzH 2 Achse 2 -		Tolerances DIN ISO 2768-mk - - -

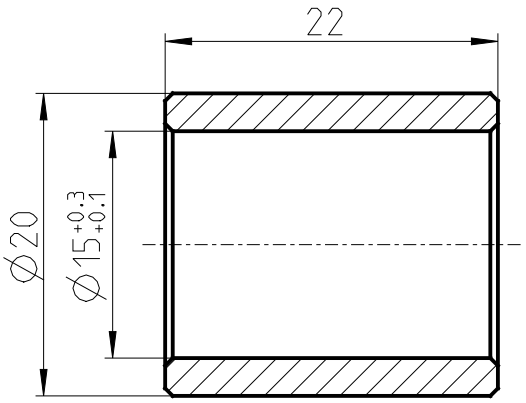
A-A



Alle Kanten mit 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.05 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00003-07	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title DistanzH 1 gr Achse 2 -		Tolerances DIN ISO 2768-mk - - -

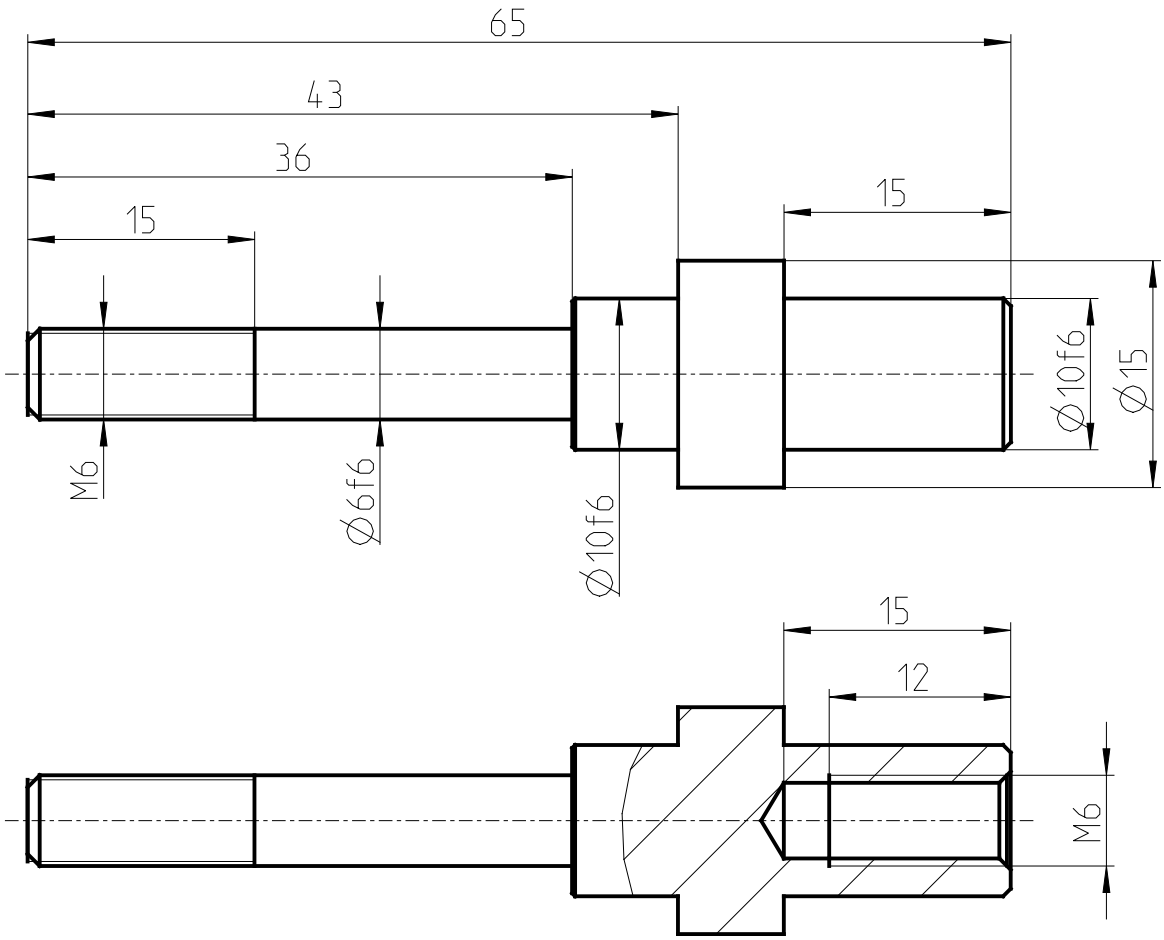
A-A
2:1



Alle Kanten 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.02 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00005-04	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title DistanzH Achse 4 -		Tolerances DIN ISO 2768-mk - - -

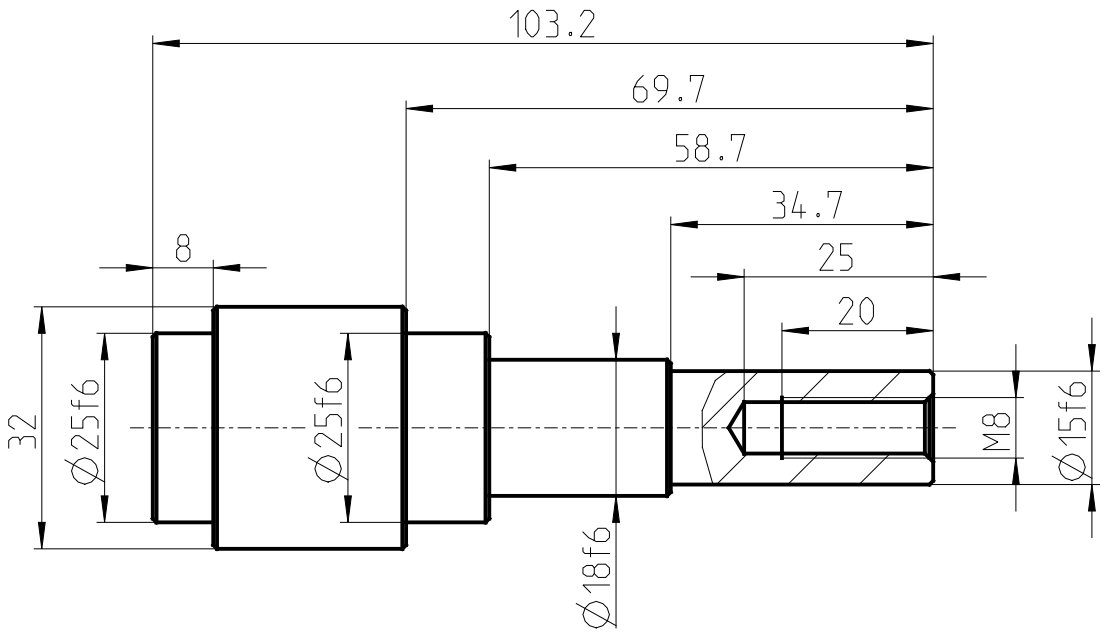
M=2:1



Alle Außenkanten 0.5x45° angefast

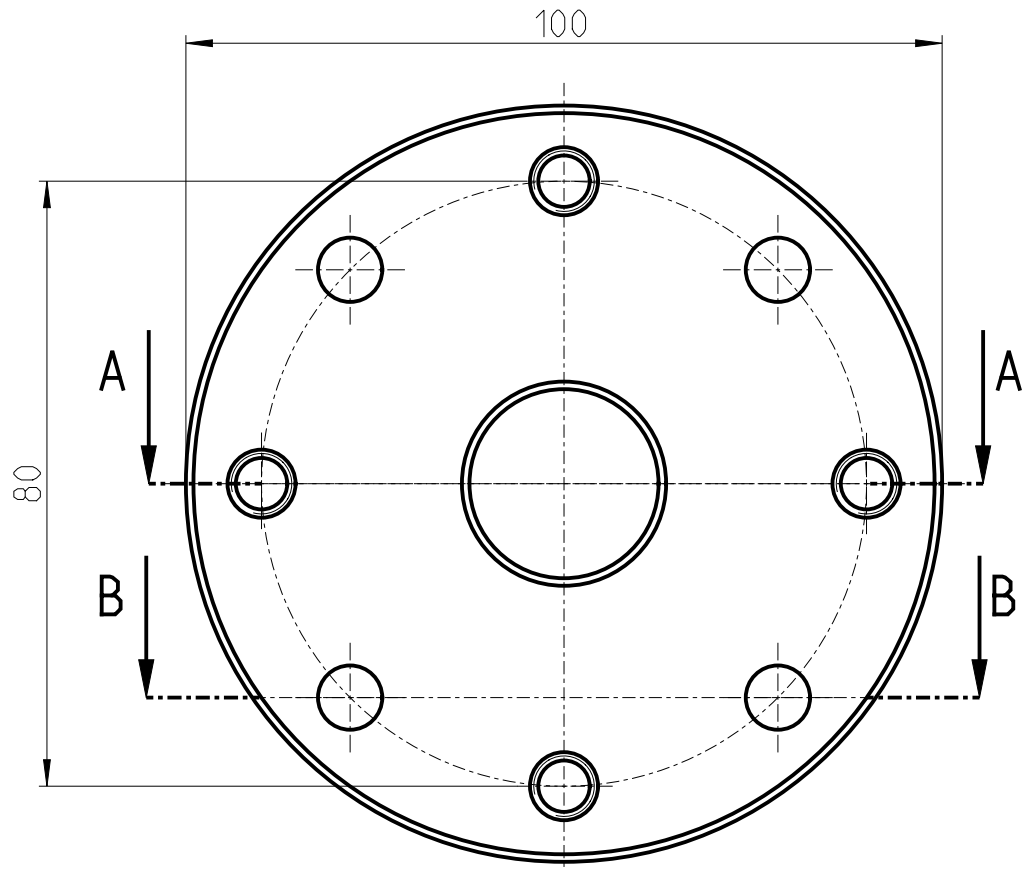
Dok. Art / Doc. Type -	Masse / Mass 0.03 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 2:1	Sach Nr. / Part No. 21T01-22-00004-02	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title Ritzelwelle -		Tolerances DIN ISO 2768-mk - - -

A-A

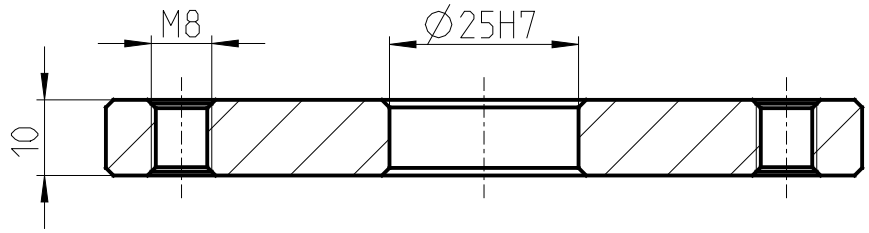


Alle Außenkanten 0.5x45° angefast

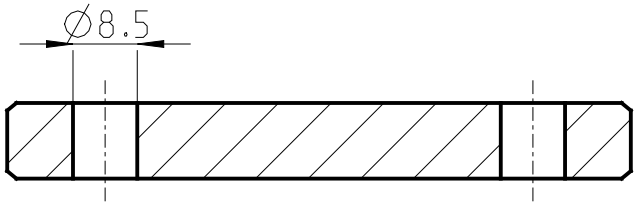
Dok. Art / Doc. Type -	Masse / Mass 0.32 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00005-01	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title Hauptwelle Achse 4 -		Tolerances DIN ISO 2768-mk - - -



A-A



B-B



Alle Kanten 0.5x45° angefast

Dok. Art / Doc. Type -	Masse / Mass 0.54 kg	Genehmigt / Approved -	Halbzeug/Wrought Prod. - Werkst./Mat. - --E295	
Eigentümer / Owner HTL	Datum / Date 26.01.2022	Ersteller / Designer Florian Zachs	Klasse / Class 5BHMBA	Bewertung / Grade
ARGE 3D-CAD www.3d-cad.at	Format A4	Maßstab / Scale 1:1	Sach Nr. / Part No. 21T01-22-00005-02	
htl bildung mit zukunft	Blatt / Sheet 1 / 1	Benennung / Title Adapterplatte Achse 4 -		Tolerances DIN ISO 2768-mk - - -