

Trandem, Hans-Martin
Ytrøy, Vegard H.

Robotised Grinding in Confined Space

Bachelor's thesis in Mechanical Engineering
Supervisor: Mork, Ola Jon
Co-supervisor: Kleppe, Adam Leon
May 2023

Trandem, Hans-Martin
Ytrøy, Vegard H.

Robotised Grinding in Confined Space

Bachelor's thesis in Mechanical Engineering
Supervisor: Mork, Ola Jon
Co-supervisor: Kleppe, Adam Leon
May 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Ocean Operations and Civil Engineering





DEPARTMENT OF ICT AND NATURAL SCIENCES
&
DEPARTMENT OF OCEAN OPERATIONS AND CIVIL
ENGINEERING

IELEA2920 - BACHELOR THESIS AUTOMATION

MASA2900 - BACHELOR THESIS MACHINE

Robotised Grinding in Confined Space

Authors:

Hans-Martin Isentorp Trandem

Vegard Herland Ytrøy

Date: 22.05.2023

Foreword

This bachelor's thesis is an interdisciplinary project with the Department of ICT and Natural Sciences, and the Department of Ocean Operations and Civil Engineering. Vegard H. Ytrøy is an automation engineer student with a background as an automation technician, and Hans-Martin Trandem is a mechanical engineering student with a background as a CNC operator.

Our project began thanks to an opportunity that came up through Vegard's links with Effe. We picked this task because it was flexible and practical, letting us explore many solutions, with in one project.

We have looked into ways to achieve grinding in confined spaces. We would like to thank Effe for providing the case study and letting us use their office space, workshop, and robots to work on our thesis. Effe has been a great partner and has offered valuable insights into the problems we've faced.

We also want to thank Dan Bentzen and Frants H. Poulsen for their helpful criticism and support when we encountered obstacles. Their guidance has been much appreciated. A special thanks go to our supervisors, Ola Jon Mork and Adam Leon Kleppe, for their insights into the project and for helping us create this bachelor thesis. Their expertise and encouragement have played a crucial role in our work.

Lastly, we want to express our gratitude to our fellow students, with whom we've had the chance to discuss key issues and gain new insights for our thesis. Their contributions have enriched our work and made this experience all the more rewarding.

Vegard have been employed at Effe throughout this bachelor thesis. Vegard has not received any compensation for his work on this bachelor thesis.

Table of Contents

List of Figures	iii
List of Terms	v
1 Introduction	1
2 Theory	4
2.1 Mathematical equations	4
2.2 Mechanics	4
2.2.1 Statics	4
2.3 Product architecture	4
2.4 Computer Aided Manufacturing	5
2.5 Control Theory	5
2.5.1 PID	5
2.5.2 Kalman Filter	5
2.6 Robotics Theory	6
2.6.1 Kinematics	6
2.6.2 Singularity	6
2.6.3 Interpolation	7
2.7 Industry 4.0	7
3 Method	9
3.1 Hardware	9
3.1.1 Calculations	10
3.2 Concept 1	13
3.3 Concept 2	13
3.4 Mounting mechanism	14
3.5 Robot	15
3.5.1 Path generation and simulation	16
3.6 Test mounting of robot	17
3.7 NX CAM	19
3.7.1 Machine tool builder and Kinematic structure	20
3.7.2 Toolpath programming	21
3.8 Statics of the angle grinder	23
3.9 Tank centre	27

3.10	Refinement of concept 3	28
3.11	Clamping force	30
3.12	Automation and robot programming	31
3.12.1	First Version (URScript)	31
3.12.2	Second Version (URCap)	32
3.12.3	Third version (FT 300-S)	32
3.12.4	Fourth Version (C++)	33
3.12.5	Fifth Version (GUI using Qt)	34
3.12.6	Sixth Version (PID)	35
3.12.7	Seventh Version (Multiple Grinding Orientations)	36
3.12.8	Current system state	38
4	Results	41
4.1	Assembly	41
4.2	Regulation and Control Loop	42
5	Discussion	44
5.1	Jig	44
5.2	Simulation and path generation	45
5.3	Programming and robot control	46
5.3.1	Choice of Programming Language	47
5.3.2	Choice of Regulation and Control	47
5.4	System Architecture	48
6	Conclusion	49
	Bibliography	50
	Appendix	53
A	System code (C++)	53
B	Python and calculations code	53
C	Pre-project rapport (Forprosjektrapport)	53
D	Video	53
E	Machine drawings	53

List of Figures

1	Visualisation of a tank cutout process	1
2	Module diagram	2
3	Common support types	4
4	X, Y and Z components of the vector between point A and B.	7
5	The four industrial revolutions	8
6	Functional elements for the jig	9
7	Concept sketches	10
8	Siplefied sketch of the jigg with robot arm	10
9	Free body diagram of the robot jig	11
10	Main hub concept.	13
11	Concept 2	14
12	Different types of mounting solutions.	15
13	Grinding demo	16
14	Toolpaht conversion NX - RoboDK	17
15	Stiffening measures for the frame	18
16	Robot Control NX	20
17	NX machine tool builder	21
18	Variable contour	22
19	Drive method — Streamline	22
20	Load case for the angle grinder	24
21	FBD for the angle grinder.	26
22	Tank simulation	28
23	Version 3 of jig	28
24	Foot assembly	30
25	Terminal output from FT 300-S	32
26	Qt interface for designing GUI in Visual Studio	34
27	First working version of the GUI with a plotter	35
28	Setup for first grinding tests (safety guard removed for illustrative purposes)	36
29	Visualization of interpolation and interpolation with regulation vectors (only for illustrative purposes)	38
30	Schematic of the robot program	39
31	Current version of GUI	40
32	FT300-S measurements from UR force compensation	42

33	FT300-S measurements from custom force compensation with error, setpoint and PID values	43
34	Final version of jig	44

Special Terms

Acronyms:

UR	Universal Robotics
CAD	Computer Aided Design
CAM	Computer Aided Machining
MCS	Machine Coordinate System
TCP	Tool Center Point
RTDE	Real Time Data Exchange
hp	horse power
GUI	Graphical User Interface
R&D	Research and Development
DoF	Degrees of Freedom
FBD	Free Body Diagram
NC	Numerically Controlled
IDE	Integrated Development Environment

Glossaries:

XML-RPC	A Remote procedure calling using HTTP as the transport and XML as the encoding
Java	A high-level, class-based, object-oriented programming language, designed to have as few implementation dependencies as possible
Python	A high-level, general-purpose programming language, with emphasis on user readability
C++	A high-level, general-purpose programming language, with performance, efficiency, and flexibility as design highlights
Qt	Qt is a cross-platform software for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms
CSYS	NX specified coordinate system

Units

Gravity $g = 9.81m/s^2$

Horse power $1hp = 745.70W$

1 Introduction

In this bachelor thesis, we will look at how to achieve remote grinding with a robot in confined spaces. There is a video in the appendices that demonstrate some of the results from this thesis.

The goal of grinding is to remove material [36, 18]. When grinding, there are different methods of achieving it, but one of the most common is to use an angle grinder. An angle grinder is a handheld tool, either electric or pneumatic driven. Angle grinders are considered one of the most dangerous tools [25]. In the US there are more than 5000 documented cases each year of angle grinder injuries and accidents [4]. There are many hazardous aspects with angle grinding. Some hazards are; airborne particles, noise, moving parts and electrocution (if electric). Airborne particles may enter the lungs and cause scarring over time [25]. The grinding disc is also a major hazard, as the disc can shatter and explode, causing major damage to the operator [20].

If a remote grinding operation can be preformed, it will relive workers for other tasks, and improve work security for workers [6]. By removing the worker from a hazard zone, the need for safety personnel is eliminated, and more workers can be used for other tasks. Exposure time in confined spaces and the hazardous environment created while working are minimized by replacing the worker with a robot [11, 6]. Some development to robotising of such tasks has been done, but most systems are of a more predictable state or in a standardised setup.

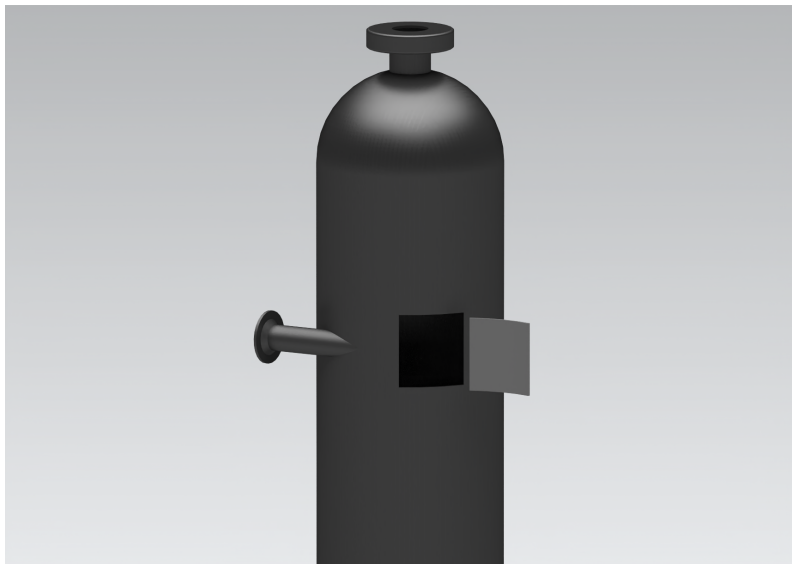


Figure 1: Visualisation of a tank cutout process

Effee, a company specialising in induction heating and robot welding repairs, is seeking to adapt its robotic welding technology for grinding purposes. While Effee currently possesses the capabilities to perform advanced welding operations with a robot, these operations still require an operator on-site for supervision and preparation. At present, Effee lacks a method for remotely preparing a surface for robotic welding. A solution enabling remote grinding and preparation would permit more advanced, remote operations. Research and development of this technology is a keen interest within the industry.

The specific case for our investigation is a process tank as shown in figure 1. Parts of the tank have pitting corrosion on its interior walls. Previously, repairs were carried out by cutting the damaged section out of the tank. Then a new section was prepared for this tank, then welded in place of the cut section. This operation, is not only dangerous but also lead to further costs and complications. Following the repair, the tank had to undergo a new round of pressure certification and commissioning before it could be reused. If repairs can be conducted without breaching the structure of the tank, this would eliminate the need for a full pressure test, thereby saving both time and money. These potential savings offer additional motivation for the development of an

alternate repair method.

One possible tool to facilitate these tasks could be a model predictive controller (MPC), as mentioned in “Model Predictive Force Control in Grinding based on a Lightweight Robot” [11]. However, implementing such a system would necessitate extensive modelling for each specific job. Consequently, substantial research is being invested into robotic grinding and polishing, particularly in developed countries where labour and cost efficiency are of high importance [40]. These advancements have led to the introduction of innovative products to the market. Nordbo Robotics, for instance, offers systems like the CraftMate, which is capable of grinding and polishing on most 3D surfaces [3].

Numerous papers have been written on the subject of estimating grinding forces. Many of these are based on a mathematical model that entails an estimation of the cutting geometry of the grinding wheel. Using an estimation of the grain geometry, a force can be calculated [14, 24]. These models are based on the use of a grinding machine, rather than an angle grinder. With a grinding machine, rigidity and accuracy are simpler to control.

However, when using an angle grinder, the system becomes significantly more dynamic. One study aimed at estimating grinding forces with an angle grinder demonstrated promising results, but it’s relatively complex as it employs machine learning and advanced sensor data processing [7]. Employing these techniques to estimate forces is beyond the scope of this thesis.

To achieve this task we will need to develop a way to mount the robot inside the tank, a way to map the surface that needs to be ground down, a way to generate the grinding paths, and a way to monitor and regulate the grinding process. The media causing the corrosion in the tank is an industrial coolant. The tank has an inner diameter of 1100 mm. The robot we will be using is a UR10e equipped with a Robotiq FT300-S provided by Effee. Effee also provided us with a Dynabrade 52633BK air driven angle grinder for robots.

The product architecture of the product is modular based. Figure 2 shows the key aspects of the development of this product. The product is divided into two main categories, software, and hardware.

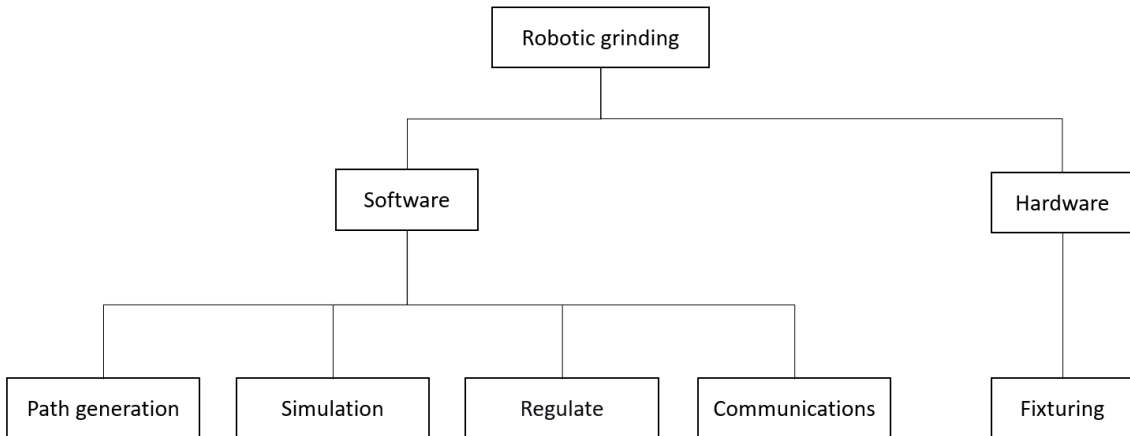


Figure 2: Module diagram

The work was divided based on the different experiences and fields of engineering. Hans-Martin worked on developing the hardware side of the project, the path generation, and the simulation. Vegard’s main field in this thesis is the regulation and the communication. Many of these fields overlap and requires cooperation. Design considerations had to be accounted for in all fields to achieve a good result. Figure 2 show the different tasks divided up into different modules. The integration between path generation and regulation was not fully formed in this paper, but functioning concepts are devised as standalone solutions to build on.

Thesis Outline

The structure of the thesis is as follows:

- **Chapter 1: Introduction**
 - An introduction to the thesis and the specific case.
- **Chapter 2: Theory**
 - Provides an overview of terms and theories used in this thesis to test and explore the possible solutions for the problems.
- **Chapter 3: Methods and development**
 - Discusses the hardware purposed for the task and the challenges with mounting the robot.
 - Design and testing of concepts for the robot jig main hub and mounting mechanisms, simulation of the system solutions.
 - Details the machine tool builder and kinematic structure, toolpath programming, and automation and robot programming, integration and development.
- **Chapter 4: Results**
 - Presents the results from the thesis.
- **Chapter 5: Discussion**
 - Discusses the findings, solutions and choices made.
 - Summarises the findings of the thesis.
 - Discusses the potential benefits of the proposed solution.
- **Chapter 6: Conclusion**
 - Summarises the discussion.
 - Suggests areas for future research.

Each chapter builds upon the previous one, leading to a comprehensive look in to remote grinding in confined spaces.

2 Theory

2.1 Mathematical equations

General equation for a circle

$$(x - x_0)^2 + (y - y_0)^2 - r_0^2 = 0 \quad (1)$$

2.2 Mechanics

Mechanics is the study of the interaction of forces, matter, and motion. The foundation that traditional mechanics is based on Newton's three laws of motion

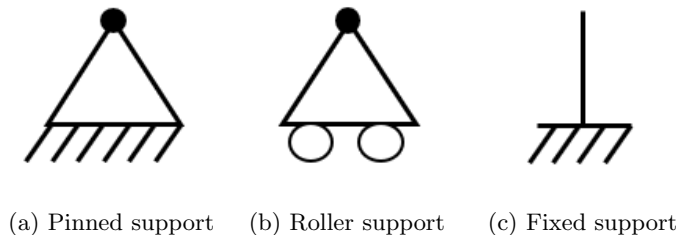
1. If an object is not acted upon by an external force, it will remain stationary or at a constant speed.
2. If an external force is acts upon an object, the object will accelerate.
3. If a force is applied, an equal but opposite force will occur.

2.2.1 Statics

Statics is a sub-discipline of mechanics that deals with the study of forces and their effects on bodies in equilibrium. Statics is a fundamental concept in engineering, particularly in the design and analysis of structures such as bridges, buildings, and mechanical components. It enables engineers to determine the distribution of forces within a system and ensure that the structure is stable and safe under the applied loads [2].

A statically determinate system is a type of structural system for which the equilibrium equations can be used to uniquely determine all the internal forces and reactions. In these systems, the number of unknown forces is equal to the number of independent equilibrium equations.[2].

In the theoretical analysis of statically determinate systems, an idealised model is drawn, to simplify calculations and provide a clearer representation of force vectors. This approach primarily focuses on beam sections, which bear the load in the system.



(a) Pinned support (b) Roller support (c) Fixed support

Figure 3: Common support types

2.3 Product architecture

Product architecture is a loosely defined term, but in essence is it a method to structure key elements of the functions of the product and help with early stages of research and development. Karl Ulrich describes in his paper *The role of product architecture in the manufacturing firm* [34] three key phases of product architecture.

(1) *Arrangement of functional elements*. What this means is to map out the different functions of the product, and different interactions between the functions. In this phase we are not interested in the physical “solutions”, only the functions of the product.

(2) *The mapping from functional elements to physical components.* The product architecture's second aspect involves mapping functional elements to physical components, which can include separable parts, sub-assemblies, software subroutines, or distinct regions of integrated circuits. A discrete physical product comprises one or more components. The relationship between functional elements and components can be one-to-one, many-to-one, or one-to-many.

(3) *The specification of the interfaces among interacting physical components.* Some parts will naturally interact with each other. It is practical to map these interactions and create contained solutions. Many connections can be standardised and grouped together. For example, it would be impractical to map every detail of a bolted connection.

2.4 Computer Aided Manufacturing

Computer Aided Manufacturing is a way of utilising data from CAD software to program NC-machines. By utilising CAM software there is a great potential to reduce programming time, ensure the programming is correct, and optimising the machine and the part. CAM software require a high level of detail in the data and model of the system, and an understanding of the underling principle of the manufacturing process [5, 19].

2.5 Control Theory

Control theory is a part of engineering and mathematics that studies how systems behave and how to control them. It's about using system knowledge and mathematical models to control and predict a system. Control theory is used in all areas of modern industry and production. The goal of control theory is to create controls that can keep a system working in a certain way, even when there are problems, uncertainties, or other outside factors that might disturb the system. By using control theory, engineers can build systems that work better, are more reliable, safer, and can work in different situations [1].

2.5.1 PID

The Proportional-Integral-Derivative (PID) controller, has become a vital tool in engineering and industrial settings, smoothly transitioning from the analogue era to the digital age and serving as a fundamental building block in various complex control systems. PID control is a feedback mechanism that regulates a system's output by adjusting its input, where each of the three components - Proportional (P), Integral (I), and Derivative (D) - play a critical role. The controller calculates an error signal, the difference between the desired set-point and the system's current output, which then guides the adjustments to the system's input to minimise the error and bring the system closer to the set point. With the Proportional component responding proportionally to the current error, the Integral component accounting for cumulative past errors, and the Derivative component considering the error's rate of change, PID controllers can be optimized to achieve the desired control behaviour for a variety of systems [15].

2.5.2 Kalman Filter

The Kalman Filter is a mathematical algorithm that is used to estimate the state of a system based on measurements. It is widely used in control systems, navigation, and other applications where accurate estimation of a system's state is required. The way the Kalman Filter works is to create a prediction of the system's state based on a mathematical model, and then correct this prediction based on measurements from the system. The filter takes readings from sensors, combines them with a model of the system, and produces an estimate of the current state of the system.

The Kalman Filter operates by using two main stages: prediction and update. In the prediction stage, the filter predicts the next state of the system based on the previous state and a mathematical

model of the system. The prediction also includes an estimate of the uncertainty associated with the prediction, based on the optimal Kalman Gain. In the update stage, the filter combines the predicted state with the actual measurements and produces an updated estimate of the state of the system. The updated estimate also includes an estimate of the uncertainty associated with the measurements [41, 16].

2.6 Robotics Theory

Robotics theory is a field that deals with the design, construction, and operation of robots. In recent years, collaborative robots or “cobots” have become increasingly popular due to their ability to work alongside humans in a shared workspace. Collaborative robots require careful consideration of their coordinate system to ensure safe and efficient interaction with human operators. The robot’s coordinate system is defined by its position and orientation, and it is essential to accurately model and control the robot’s movements within this system. [13]

A robot’s coordinate system is typically defined by the robot’s base, which serves as a reference point for all its movements. The robot’s end-effector, the tool or gripper that interacts with the environment, is also an important component of the coordinate system. To ensure safe collaboration with humans, the robot’s movements must be carefully planned and controlled within this coordinate system, taking into account factors such as the robot’s reach, speed, and force limits. Collaborative robotics development aims to ensure methods and algorithms that enable robots to work safely and effectively alongside humans in a shared workspace, while also striving for optimal performance and productivity [12].

2.6.1 Kinematics

Robot kinematics is a key aspect of collaborative robotics development, as it forms the foundation for controlling and programming robots to perform specific actions. By studying the motion of robotic systems using mathematical models and algorithms, researchers can optimise robot design and operation, making collaborative robots safer and more efficient. Understanding the kinematics of a robot is crucial for engineers, so they can determine its position, velocity, and acceleration at any given time. With advances in robot kinematics and collaborative robotics development, we can expect to see increased automation in a variety of industries, from manufacturing and assembly to healthcare and education.[37]

2.6.2 Singularity

In robotics, a singularity refers to a specific point in a robot’s workspace where its DoF are reduced, causing the robot’s movements to become unpredictable or stop altogether. The DoF of a robot refers to the number of ways it can move or rotate, and it is determined by the number of joints or axes the robot has.

When a robot’s TCP approaches or enters a singularity, the robot’s ability to move freely can be compromised, and its movements can become jerky or erratic. This is because at a singularity, two or more of the robot’s axes become aligned, resulting in a loss of independent control over those axes. [31]

For example, in a six-axis robot, a singularity occurs when two of its axes align, reducing the robot’s DoF from six to five. At this point, the robot’s movements may become unstable or unpredictable, potentially causing errors or collisions. In some cases, the robot may even come to a complete stop, requiring manual intervention to recover.

To avoid the negative effects of singularities, engineers must carefully plan and optimise the robot’s path and trajectories to avoid approaching or entering a singularity. This can involve using software tools to simulate and visualise the robot’s movements and identify potential singularities before programming the robot’s motion. By accounting for singularities and optimising the robot’s

movements, engineers can ensure that the robot operates safely and effectively in its workspace [38].

2.6.3 Interpolation

Linear interpolation is a widely used mathematical technique for estimating values between a discrete set of known data points. This method involves using linear polynomials to create new data points that lie within the range of the given data points. The linear interpolation method is especially useful in situations where we need to estimate the value of a function at a point that falls on a line drawn through two known data points. By using linear interpolation, we can construct an approximate curve that passes through the given data points. Making it easier to visualise and analyse the behaviour of the underlying function and estimate or predict linear trajectories. Moreover, linear interpolation is a straightforward and computationally efficient method that can be easily implemented using various programming languages and software tools. [22, 39]

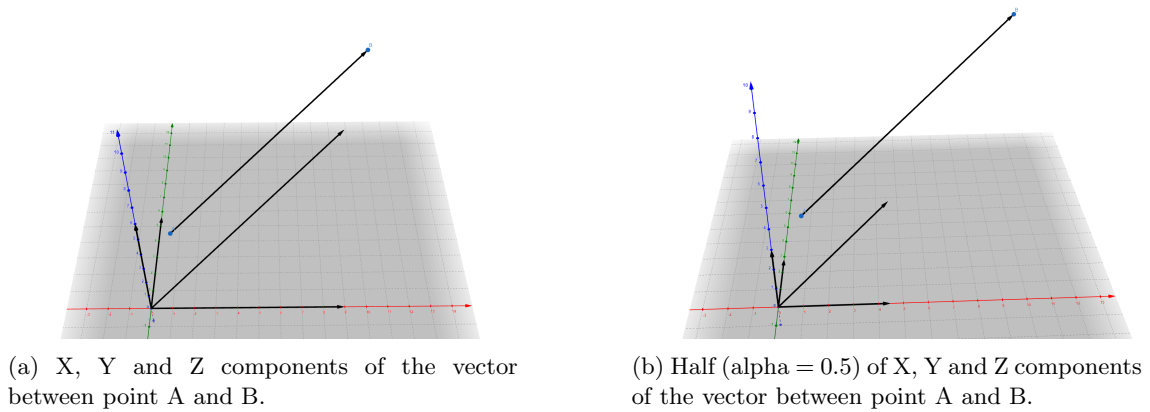


Figure 4: X, Y and Z components of the vector between point A and B.

Linear interpolation is a commonly used technique in robotics for controlling the movement of robotic arms and other mechanical devices. In this context, linear interpolation involves generating a sequence of intermediate positions between two given points to achieve a smooth and continuous motion trajectory. In addition to controlling the movement of robotic arms, linear interpolation is also used in other robotics applications, such as path planning, trajectory generation, and sensor data processing. Figure 4 illustrates how linear interpolation can be used to calculate the exact position between two points and break it down into the 3D vectors required to move the robot from its current position. This approach can be applied to each control loop within the robot, enabling the robot to move smoothly and predictably. [42]

2.7 Industry 4.0

The term “Industry 4.0” refers to the fourth industrial revolution, characterised by the combination of cyber-physical systems, the Internet of Things (IoT) and the Internet of Services (IoS) within the manufacturing sector [23]. This revolution aims to optimise the entire value chain through the integration of intelligent technologies, ultimately leading to enhanced productivity and efficiency.

Cyber-physical systems are a key aspect of Industry 4.0. A cyber-physical system exchanges information and trigger events automatically based on the data the systems collects. [23]. By enabling real-time data exchange and decision-making, cyber-physical systems offers the potential to create more flexible, adaptive, and efficient production processes.

Internet of Things (IoT) is another stage of industry 4.0. IoT is a network of interconnected device, machines, and objects that can communicate and exchange data without the need for human

intervention [23]. By embedding sensors, actuators, and other smart devices in manufacturing equipment, the IoT enables real-time monitoring and control of production processes, which can lead to improved efficiency and reduced waste.

Complementing the IoT is the Internet of Services (IoS), which focuses on providing value-added services to support the manufacturing process [23]. These services can range from cloud-based software solutions to analytics and data-driven decision making tools. The IoS enables manufacturers to optimise their processes, reduce costs, and enhance customer satisfaction by providing tailored services and products.

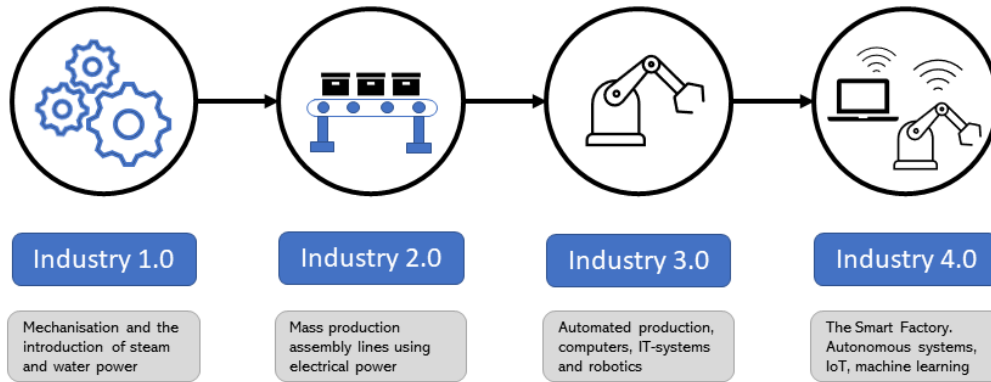


Figure 5: The four industrial revolutions

3 Method

3.1 Hardware

As the goal of this bachelor is to grind the inside of a process tank. One of the tasks set out by this bachelor is to find a solution on how to mount a UR10e from Universal Robotics inside the tank. Before the development of the jig started, we set up some design criteria and specifications, ref. table 1. This is so we can quantify the different aspects of the product and measure the point of the product.

Demanded	Specification	Value	Importance (1-5)
<i>Production</i>	- Easy to manufacture	No special methods required to manufacture it	3
<i>Economics</i>	- Easy to manufacture	One of product	3
<i>Durability</i>	- Easy to maintain	No special training required	4
	- None flammable	No flammable materials	4
	- Stiff	Stiff enough to support the robot	5
<i>Design</i>	- Practical	Minimal training required	3
	- Appearance	Functional	1
<i>User friendliness</i>	- Light weight	One person can lift it	2
	- Demountable	Assembled and disassembles in tank	5
	- Easy to clean	Removal of dust	3
	- Easy to mount	One person can mount it	4
	- Intuitive	Minimal training to use it	5
<i>Environment</i>	- Reusable	Demountable from the tank	4
	- Safe materials	Only metal	3
<i>Dimensions</i>	- Diameter mounted	1100mm	-
	- Dismantled	D=250mm L=500mm	-
	- Clamping range	100mm	-
	- Weight	Under 45kg	-

Table 1: Demand specification table

The fixturing of the project can be divided into three main modules, grinder to robot, jig to tank and robot to jig. We have further subdivided the blocks into further modules. Some modules are out of our control or has not been further develop as it is off the shelf products, or time constrains. This mainly apply to the fixturing of force torque sensor as this is a product design for the robot.

We mapped what we think are crucial functions of the jig in fig. 6. This diagram shows the interaction between the functions and the physical parts.

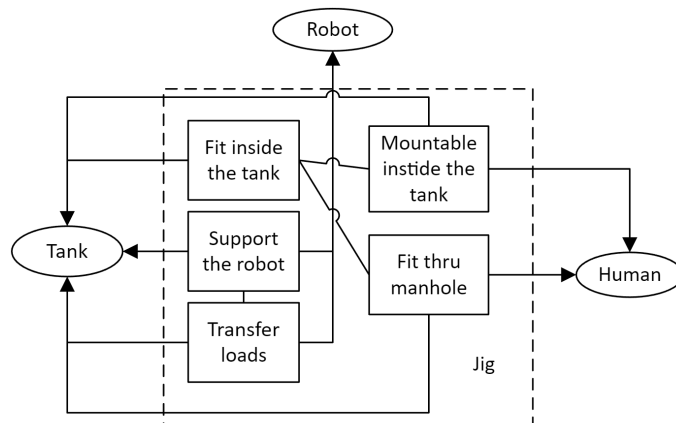
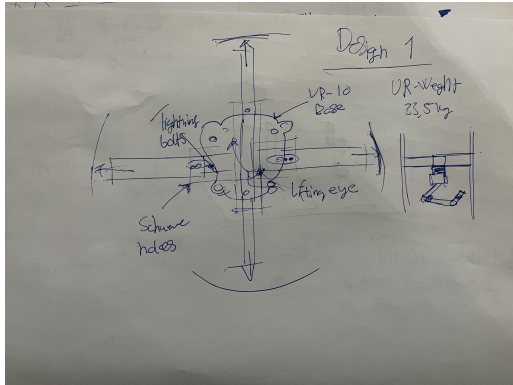


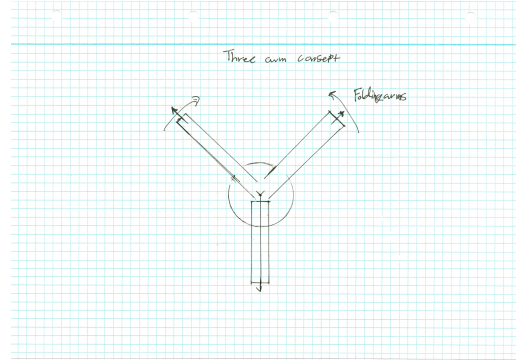
Figure 6: Functional elements for the jig

This jig is not going to be mass-produced, and does not need to follow any regulations. For now,

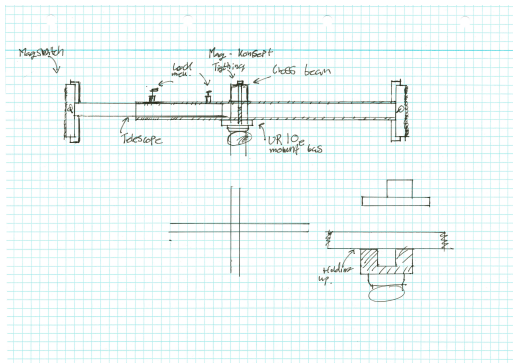
we will assume that only one unit is to be produced. The design process was composed of some rough sketches of the different concepts and then taken in to Siemens NX for modelling. The drawing seen in figure 7 shows the rough idea of the first concept. It composes of four arms and a round base in the centre that the robot mounts to. The idea is to mount the arms first and then insert the centre. This will help with the issue of lifting it in the tank. The way it will be connected to the wall can be done several ways. It was decided to look at the “body” of the jig first.



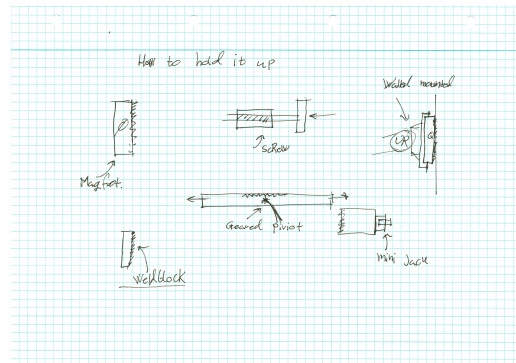
(a) Static for arm concept



(b) Three arm concept



(c) Telescope with magnets



(d) Force pads concept

Figure 7: Concept sketches

3.1.1 Calculations

Before the sketch was realised in NX, we did some rough estimations of what forces the jig would encounter. In these calculations, it's assumed that the system is static.

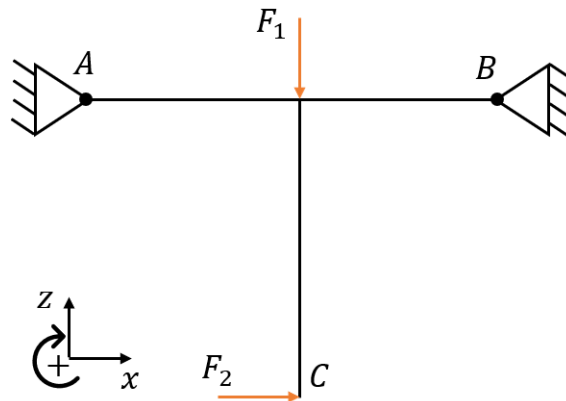


Figure 8: Siplefied sketch of the jig with robot arm

For the first concept, it is assumed that the beams are pinned to the wall, as shown in figure 8. F_1 represents the weight of the robot, and F_2 represents the theoretical max force that the robot can apply, for a more refined calculation this force needs to be more precise. But for the purpose of early design work, the force that is listed as max force is acceptable. The Distance AB is the total distance of the beam between the tank walls. This is the same as the tank inner diameter. The distance DC is the length of the UR10e fully extended. F_1 and F_2 are known forces, this makes this system statically determined. We can first solve the system graphically as shown in figure 9.

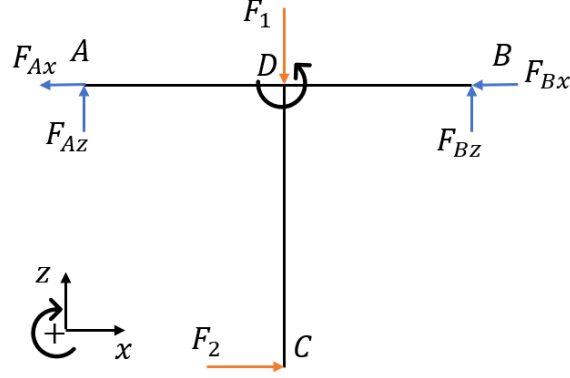


Figure 9: Free body diagram of the robot jig

$$\sum M = 0 \quad (2)$$

$$\sum M_A = 0$$

$$M_a = F_1 \cdot AD - F_2 \cdot DC + F_{Bz} \cdot AB$$

$$F_{Bz} = \frac{F_1 \cdot AD - F_2 \cdot DC}{AB} \quad (3)$$

$$\sum F = 0 \quad (4)$$

$$\sum F_z = 0$$

$$F_{Az} + F_{Bz} - F_1 = 0$$

$$F_{Az} = F_1 - F_{Bz} \quad (5)$$

$$F_{Ax} = F_{Bx} = \frac{F_2}{2} \quad (6)$$

The forces will then be.

$$F_1 = 33.5kg \approx 330N$$

$$F_2 = 80N$$

$$F_{Bz} = 70.5N$$

$$F_{Az} = 259.5N \quad (7)$$

$$F_{Ax} = F_{Bx} = 40N$$

In these calculations, we do not account for the weight of the jig, and that there is a cross-beam that would share some of the load. Doing the calculations this way it makes it so, we have a built-in safety factor in our design. These calculations are only a guideline for the concept design work. New, more accurate calculations should be done later.

3.2 Concept 1

The first concept is based on a cross type section, as shown in figure 10. The idea was to attach four square pipes to a central hub that the robot would mount to. This turned out to not be a good idea. The design would be too hard to mount inside the tank. This is because you would need to attach two pipe sections at once, and thread them in the central hub. Ideally, only one worker should be able to mount it alone. The square pipes that were chosen were also unnecessary big at 100x100mm. The design would also be difficult to manufacture. The main hub is one solid piece. If the part was to be 3D-printed, this design would be preferred. The part could be printed in one solid piece. This would reduce the part count significantly. But printing in plastic would make the part too weak, and 3D printing metals can be expensive. It would not make sense in this application.

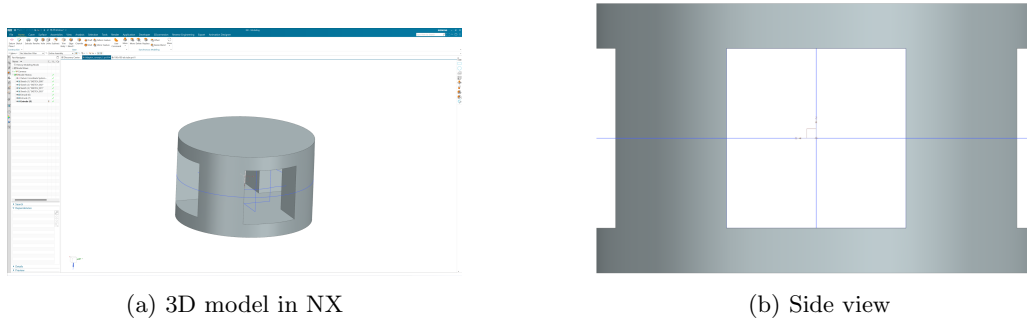


Figure 10: Main hub concept.

3.3 Concept 2

The second concept has the same principle as concept 1 with four beams and a centre hub, as shown in figure 11. A redesign of the main hub, was the main priority. Concept 2 composes of four main parts, one long square tube, two shorter tubes, the main hub, and a cover. In concept 2 the biggest design consideration was how to mount and assemble the jig. The jig is to be mounted inside the tank, and needs to be easy to mount. To accomplish this, the mounting sequence is important. When the jig is to be mounted in the tank, it is completely disassembled. The first part to be mounted in the tank is the long square pipe. By mounting this first, the main hub can be put on top and roughly aligned. After the main hub is aligned, the rest of the components can be mounted. The UR10e is to be mounted last. The robot is attached to the end cap of the jig, see figure 11c.

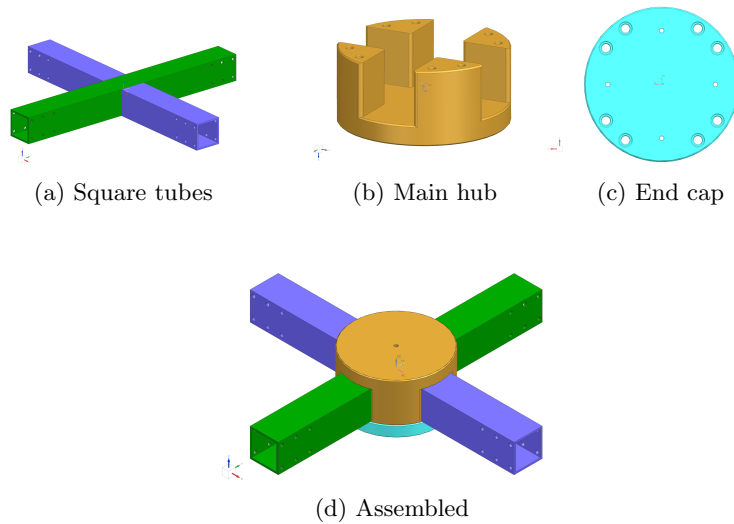


Figure 11: Concept 2

3.4 Mounting mechanism

The jig is to be mounted in the tank, to find the optimal solution to this, we looked at similar possible solutions. The different inspirations can be seen in figure 12. As Effe sees it, there are three different principles on how to mount the jig. Weld it into place, clamping it inside the tank using mechanical leverage, or to clamp it in place using magnets.

In our designs, we will try to avoid welding the jig in place. Although welding the jig in place would be the most secure, the jig would need to be welded in place by a worker, and cut out by a worker. Effe told us that it is possible to get permission to weld and grind a small job like this, but it is not ideal.

Using mechanical leverage is a solution. This way of mounting the jig puts the beams under tension and locks it in to place. This is the solution is worth investigating as it is simple mechanism that is easy to implement.

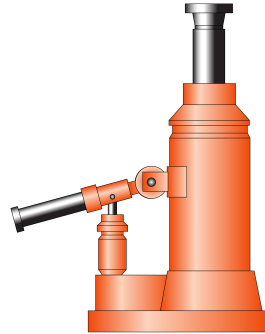
Magnets are a promising way of solving our issue. It is a common solution to mount different hardware on magnetic surfaces. Effe currently uses this type of magnets on many of their welding robots. The magnetic field can easily be turned on and off using a switch. These magnets are permanent, so there is no requirement of electricity. There are some concerns about what effect grinding dust will have on the magnet.



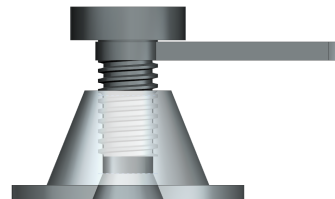
(a) Arm Hand Jack



(b) Permanent magnet



(c) Jack. Photo: [29]



(d) Screw Jack

Figure 12: Different types of mounting solutions.

3.5 Robot

The robot used for this task is a UR10e from Universal Robotics, it is a collaborative robot with safety features built in to prevent overload and collisions. It is contradicting to how the task asks the robot to be used, where an external load against a surface is needed to make the robot do the grinding operation. The robot has a max lifting capacity of 12.5 kg and a reach of 1300 mm, with a 6 joint configuration giving the robot 6 axis operation. The intended use of the robot was to have it preform the calculations and iterations to maintain a downward pressure to give the grinder the correct pressure towards the surface. The first test was to try to use a UR-script to create a control loop for the robot, as shown in figure 13. This proved the concept and showed that the robot was able to grind using the equipment provided.

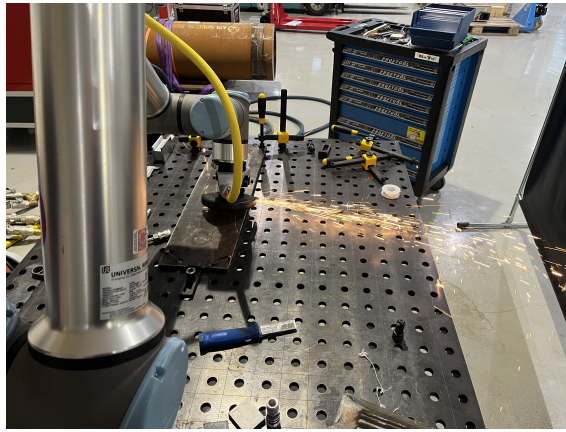


Figure 13: Grinding demo

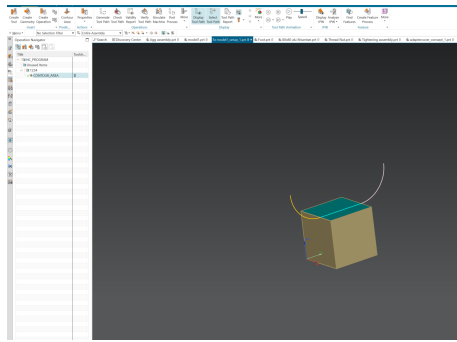
3.5.1 Path generation and simulation

To perform the grinding operation, we need to have the robot perform moves. These kinds of moves are called toolpaths. Ideally, we would like to experiment with many different patterns of toolpaths and other ways we might need to adjust the toolpath. Effe has some proprietary software that runs on UR for welding applications. We looked into using this, but we found some limitations and the tools were not what in the use-case for this application. The limitation we saw in the Effe welding path generation is that it does not currently have collision detection. On our part, it would have required a lot of modifying of the code. We decided in our case, it would make more sense to use CAM software to generate the toolpaths.

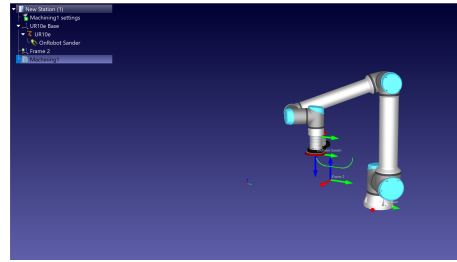
Our first idea on how we would solve the issue of generating toolpaths and simulate the robot was to use RoboDK. RoboDK is a software that provides a comprehensive library of robots and tools to simulate and program robots. Unfortunately, RoboDK does not provide tools to generate toolpaths on a geometry. To work around this, you can import 5-axis G-code into RoboDK and convert it to the appropriate programming language for the robot.

To generate the G-code we use Siemens NX. NX provides an advanced suite of tools that is used to program CNC machines. The reason we chose to use NX is because of the advanced capability of the software and the licence agreements NTNU has with Siemens. We also have some familiarity to the NX platform.

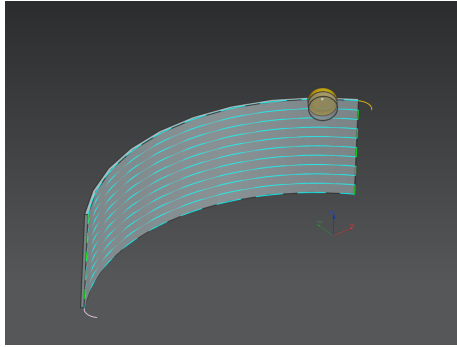
The first task was to test the workflow of NX and RoboDK. We started with a simple toolpath in NX and exported the G-code to RoboDK. With the simple toolpath this worked as intended. The simulation was correct, and the moves were as programmed in NX. But when we tried some more advanced tasks, we ran into problems. The first problem was it did not convert properly. This was probably because the MCS was in the centre of the pipe section, this made the code so the B-axis in the 5-axis CNC machine control, was the only part that changed. RoboDK interpreted this as only a joint move on the robot, and not a translation of the robot. We also found the process to be a bit cumbersome. We had all the files in NX and the only part that was needed to do in RoboDK was to create the UR-script. With some further investigation into NX we found it possible to do the whole process in NX. The different test can be seen in figure 14.



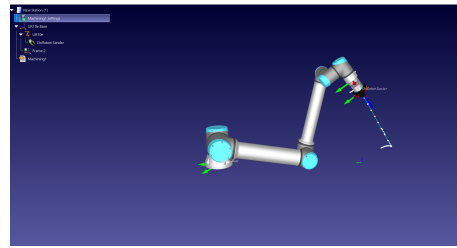
(a) Simple toolpath in NX



(b) Simple toolpath in RoboDK



(c) Advanced toolpath in NX



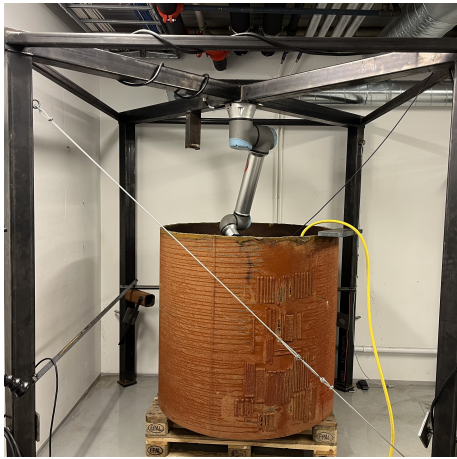
(d) Advanced toolpath in RoboDK

Figure 14: Toolpath conversion NX - RoboDK

3.6 Test mounting of robot

Early on in the project, we consulted with experts in the field, in particular Eirik Njåstad from Sintef. He suggested that we mount the robot upside down during testing. This is due to the significant differences in forces acting on the various sensors when the robot is inverted, primarily because of gravity. To facilitate this orientation change, Effee provided us with a frame they had used in some of their earlier projects. We were also given a scrap section of tubing to testing grinding on.

Initially, the frame was quite wobbly. Previously the frame was utilized for testing a welding operation, the rigidity of the frame was not crucial. However, during our grinding demonstration, it became apparent that the frame needed to be more rigid. The frame would wobble for almost a minute after the robot had stopped moving. We made some simple modifications to the frame to increase its stiffness. We added two wires diagonally as shown in figure 15a, a measure that drastically improved the frame's rigidity. Additionally, we welded two L-beams on the opposite side of the wires as shown in figure 15b, further enhancing the frame's stiffness.



(a) Wire



(b) L-beam

Figure 15: Stiffening measures for the frame

3.7 NX CAM

NX has a comprehensive CAM package. CAM is a way of programming CNC machines using computers instead of manually programming the machines. The way you program the machines is to define toolpaths on pre-determined geometry (usually 3D models). This makes it possible to simulate the tool and the machine's movements. NX allows you to program advanced toolpaths, these types of algorithms are highly sophisticated and hard to replicate. NX has a package called Robotic machining. This package includes tools to convert the traditional CNC-machine toolpaths to toolpaths that are suited for robotics.

What makes robot machining different from conventional CNC-machines, is that a 6-axis robot has eight solutions to the joint configuration. J3 and J5 refers to joint 3 and 5. OH refers to the base. The "+" and "-" signs refers to the different configuration of the joints [30].

- J3+ J5+ OH+
- J3+ J5+ OH-
- J3+ J5- OH-
- J3- J5+ OH+
- J3- J5+ OH-
- J3- J5- OH+
- J3- J5- OH-

Why this is important to have control over is because of collisions, and entanglement of the different wires and hoses. NX allows you to switch between the different joint configurations to see the different poses, as shown in figure 16. Since the space that is supposed to be ground only 1100 mm in diameter, this feature allows us to choose the correct joint configuration that takes the least space.

It is also necessary to define the tool orientation relative to the toolpath. When the grinding disk moves along the programmed path, it is desired that the position of J6 is managed. There are several options, but the most applicable for our task is tangent, and tangent zigzag. The tangent option means that the tool will orient itself tangent with the toolpath, but do not differentiate between going backwards or forwards. The tangent zigzag option orients the tool tangent, but also account for the direction the tool moves in.

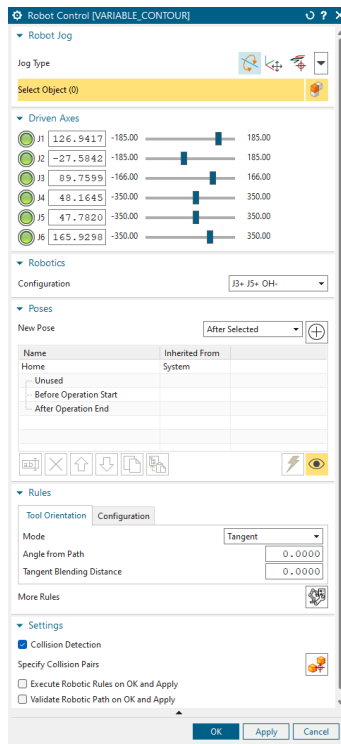


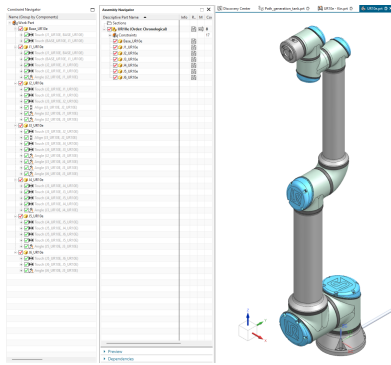
Figure 16: Robot Control NX

3.7.1 Machine tool builder and Kinematic structure

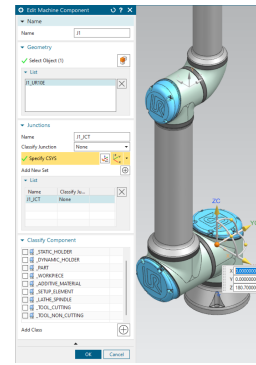
In order to simulate the robot in NX, we needed to assemble and define the kinematic structure of the UR10e. Siemens provides some pre-made robots with corresponding kinematics, but did not have the UR10e pre-built. However, they have a post-processor for URScript.

The first part of the process is to download the CAD files from UR's website. These files need to be simplified as the detail on the model is too high to be practical. It is important to orient the assembly and joints correctly in the coordinate system of the assembly. This is so that when the kinematic structure is built, it corresponds with the coordinate system of the robot. UR robots define the y-positive direction to be out of the cable on the robot's base. The rest of the joint positions are as shown in figure 17a.

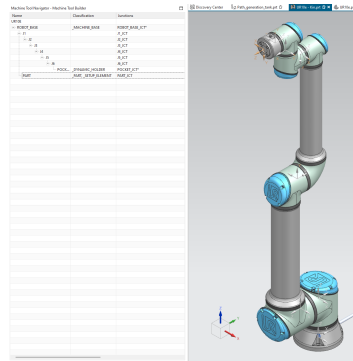
In order to make the kinematic model of the UR10e, NX needs the proper file structure in the directory. Siemens recommends copying the file structure of a pre-installed machine and modifying it for the new robot to be installed. When the file structure is set up, the kinematics can be defined. When defining the kinematics of the machine, there are two defining features: machine components as shown in figure 17b, and axis as shown in figure 17c. Defining the machine component is to define which solids go into the joint, and at this stage, you also define the centre of rotation on the joint. At this point, the orientation of the CSYS is not important, as long as one of the axis is oriented with the centre of rotation. When defining the axis as shown in figure 17d. The rotation is set, along with the initial position of the axis, the soft and hard limits on the different axes, the rotation speed, and the acceleration of the joint. This process has to be repeated six times for all the joints.



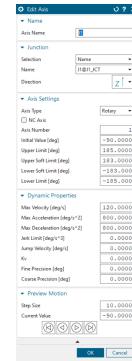
(a) Assembly of UR10e



(b) Defining machine component



(c) Kinematic structure



(d) Axis definition

Figure 17: NX machine tool builder

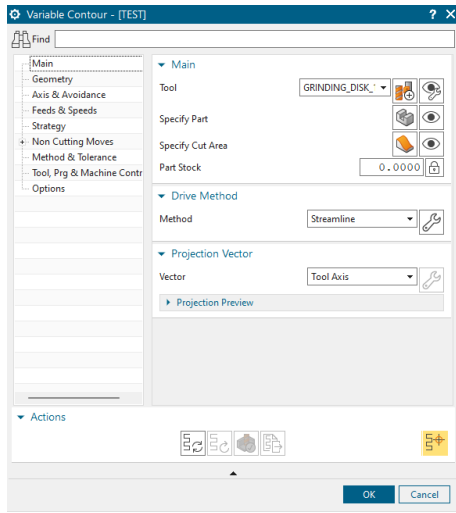
The last process of installing a new machine/robot in NX is to install the post processor. Siemens provides a wizard for installing the post processor. The post-processor is what converts the internal NX code to UR-script.

3.7.2 Toolpath programming

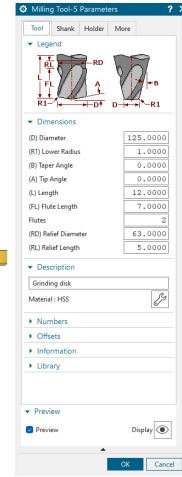
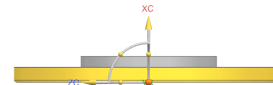
A toolpath refers to a predetermined path that a tool will follow along a workpiece, based on a specified coordinate system. Utilising software-generated toolpaths simplifies the creation of advanced toolpaths, effectively prevents collisions, and enables accurate simulation of the tool's motion, in contrast to the more labour-intensive and error-prone process of manually hard-coding the path.

The toolpath that is most suited for our application is the variable contour. The variable contour toolpath is a basic multi-axis toolpath that allows us to control the tool angle on a specified surface. The toolpath also allows us to set the specific pattern on the surface and the boundary conditions. The toolpath also determines the speed of the tool. We think this is going to be a crucial aspect of the grinding process. The toolpath also determines the lift and entry moves. We need to have control over this so the robot doesn't collide with the tank walls.

In the dialogue box, as shown in figure 18a, several distinct parameters must be defined for the variable contour toolpath. The first parameter to consider is the tool geometry, which simulates the appearance of the grinding disc. There are multiple dimensions that need to be specified, as illustrated in Figure 18b. These parameters generate a 3D representation that is displayed. The subsequent parameters to be defined involve the geometry and surface area intended for grinding. These features are determined based on the modelled geometry.



(a) Variable contour dialog box



(b) Tool def.

Figure 18: Variable contour

The drive method is responsible for determining how drive points, which dictate the toolpath, are distributed [21]. There are several different drive methods to choose from, but we found that the “Streamline” method works best for this application. The Streamline drive method automatically fills out the parameters if the selected geometry is valid. However, we needed to make some adjustments to the cut direction and the cut pattern.

By default, the tool would move along the sweep of the pipe section in a ZigZag pattern. A more optimal toolpath for this application would involve moving up the pipe. This can be achieved by specifying the cut direction. The number of steps can be easily increased or decreased by changing the number of step overs, as shown in Figure 19.

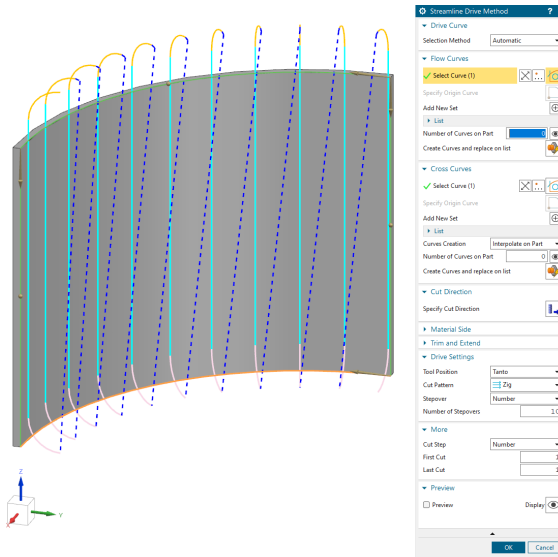


Figure 19: Drive method — Streamline

Variable control can check if the toolpath is colliding or gouging into the selected machining geometry and/or extra geometry. We utilized this by adding the tank walls as extra geometry. It will not detect collisions from the robot in the part, only the tool geometry. If a collision or gouging is detected, NX can either cut the area that is colliding, lift the toolpath over, or do nothing and warn the user. For this application, warning the user is enough.

The non-cutting moves are adjusted, so the robot does not collide with the tank wall. The lead-in of the tool is set to be parallel to the tool axis. This makes the entry of the tool gentle and in theory should help with the start of the cut.

The feeds and speeds are also defined in the variable control dialogue box. In theory, you can adjust the spindle speed, but since our angle grinder is air-powered, this value is set to an arbitrary value of 1. The feed of the grinding process needs to be tuned when the grinding test is performed.

The implementation of the code from NX, posed some challenges. When we executed the UR-script generated by NX, the robot exhibited a jittery motion. This was unacceptable, as such motion could potentially damage the robot. Initially, we believed that we were sending too many commands to the robot. Some of the earlier tests had over a thousand lines of code. When NX generates a toolpath, it sometimes segments a straight line into many sections. By decreasing the accuracy of the toolpath, we can reduce the number of sections. This approach provided some improvement, but the jittery motion persisted. After investigating the “moveL” command, we realized that the smoothing parameter was set to 0 [33]. This meant that the robot would move to the next position perfectly, requiring a full stop before each motion. The smoothing parameter represents a radius that the robot can pass through before moving on to the next point. After adding a value of 0.05 (it represents a value of 50 mm) the jitter motion was eliminated. When running the large codes now, there is no jitter.

3.8 Statics of the angle grinder

In order to correctly regulate the robot, we need to know how the forces of the angle grinder will affect the FT 300-S. We will assume that the system is statically determined, that there is only one point of contact, and that the angle grinder is fixed. The angle grinder is fixtured in the top highlighted in red shown in figure 20a. The contact point of the angle grinder is on the tip of the disk highlighted in red figure 20a. The angle grinder can be divided up in three planes; YZ, XZ and YX. The planes are divided up, so the coordinate system aligned with the coordinate system of the FT 300-S as shown in figures 20b, 20d, 20f.

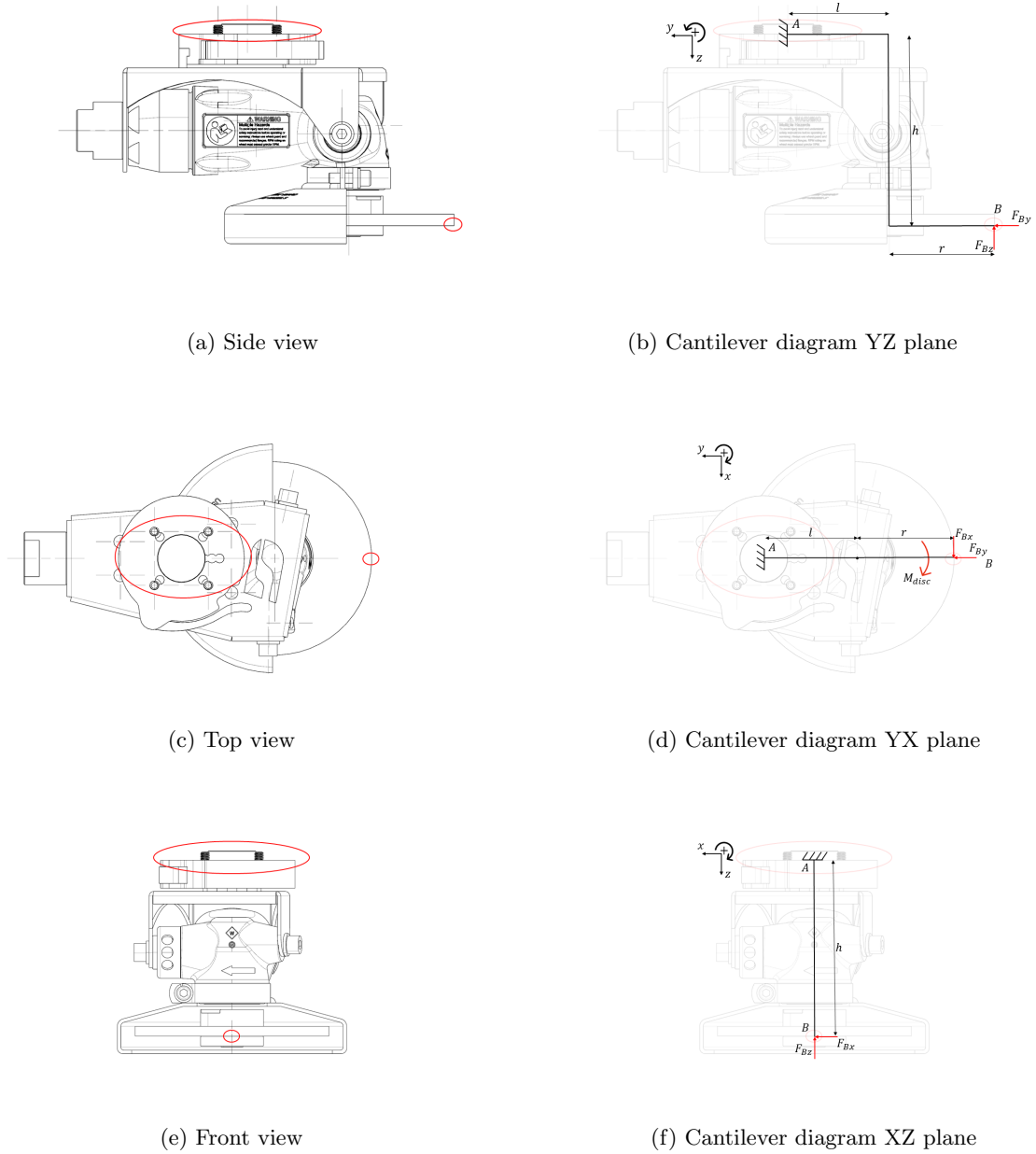


Figure 20: Load case for the angle grinder

The system composes of two forces and one moment. F_{Bz} and F_{By} occurs from the applied forces the robot exerts. The F_{Bx} force comes from the moment of the grinding disk. The friction of F_{Bz} and F_{By} will determine the actual force of F_{Bx} , as the wheel is not fixed and do not transfer the force perfectly. We can calculate the moment using equation 8, and solving for the moment. The angle grinder has 1.3 hp off effect and a max rpm of 12 000.

$$M_{disc} = \frac{P \cdot 60s}{2\pi n} \quad (8)$$

$$M_{disc} = \frac{969W \cdot 60s}{2\pi \cdot 12,000rpm}$$

$$M_{disc} = \underline{\underline{0.771Nm}} \quad (9)$$

The F_{disc} can then be calculated from the moment and the radius of the disk.

$$F_{disc} = \frac{M_{disc}}{r} = \frac{0.771Nm}{0.063m} = 12.2N$$

The frictional force that will oppose F_{disc} is calculated by combining the frictional force of F_{Bz} and F_{By} .

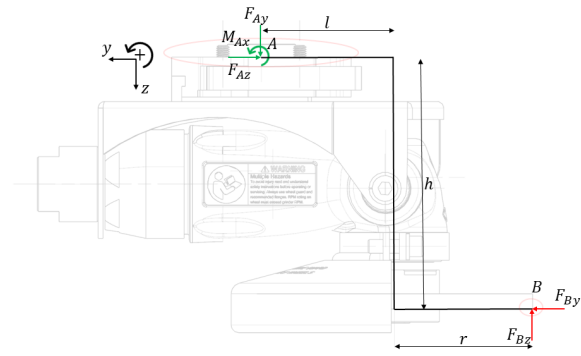
$$F_{friction} = F_{Bx} \cdot \mu + F_{By} \cdot \mu \quad (10)$$

The coefficient of friction in this case is unknown. In theory, we should be able to estimate this by reading of the sensor data and calculating the corresponding forces.

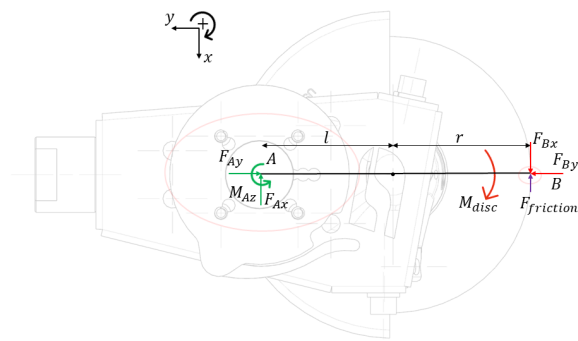
F_{Bx} can then be calculated.

$$F_{Bx} = F_{disc} - F_{friction} \quad (11)$$

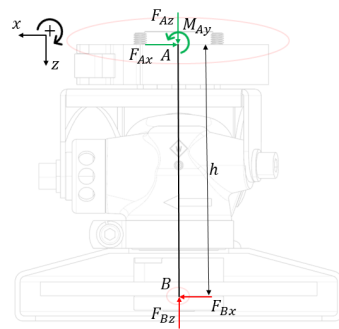
With all the forces calculated, we can set up FBD for all the planes, as shown in figure 21.



(a) FBD for YZ plane



(b) FBD for YX plane



(c) FBD for XZ plane

Figure 21: FBD for the angle grinder.

With this, we can create a matrix as shown in equation 12, that is needed to get the correct TCP forces that the FT-300S will measure.

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -h & (l+r) & -1 & 0 & 0 \\ -h & 0 & 0 & k3 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} F_{Bx} \\ F_{By} \\ F_{Bz} \\ M_{Bx} \\ M_{By} \\ M_{Bz} \end{bmatrix} = \begin{bmatrix} F_{Ax} \\ F_{Ay} \\ F_{Az} \\ M_{Ax} \\ M_{Ay} \\ M_{Az} \end{bmatrix} \quad (12)$$

3.9 Tank centre

When the UR10e is mounted in the tank, it will most likely not be in the centre. With the robot not in centre, the simulation in NX will not correspond with the real world. In the simulation, we see that it is close to collide with the wall. In order to make the simulation accurately, we need to know the exact position of the UR10e in the tank. This can be found by using an algorithm that takes in three points and calculates the position of the circle centre [27].

$$\begin{vmatrix} x^2 + y^2 & x & y & 1 \\ x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0 \quad (13)$$

$$(x^2 + y^2) \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} - x \begin{vmatrix} x_1^2 + y_1^2 & y_1 & 1 \\ x_2^2 + y_2^2 & y_2 & 1 \\ x_3^2 + y_3^2 & y_3 & 1 \end{vmatrix} + y \begin{vmatrix} x_1^2 + y_1^2 & x_1 & 1 \\ x_2^2 + y_2^2 & x_2 & 1 \\ x_3^2 + y_3^2 & x_3 & 1 \end{vmatrix} - \begin{vmatrix} x_1^2 + y_1^2 & x_1 & y_1 \\ x_2^2 + y_2^2 & x_2 & y_2 \\ x_3^2 + y_3^2 & x_3 & y_3 \end{vmatrix} = 0$$

Substituting circle.

$$r^2 - x \frac{D_{12}}{D_{11}} + y \frac{D_{13}}{D_{11}} - \frac{D_{14}}{D_{11}} = 0 \quad (14)$$

Expanding equation 1.

$$r^2 - 2xx_0 - 2yy_0 - r_0^2 = 0 \quad (15)$$

Combining equation 14 and 15 and solving for x_0, y_0 and r_0 .

$$x_0 = \frac{1}{2} \frac{D_{12}}{D_{11}} \quad (16)$$

$$y_0 = -\frac{1}{2} \frac{D_{13}}{D_{11}} \quad (17)$$

$$r_0^2 = x_0^2 + y_0^2 + \frac{D_{14}}{D_{11}} \quad (18)$$

To perform the required calculations, we utilize Python as our programming language. Upon calculating the circle, we found it beneficial to generate a visual representation of both the circle and the offset. While this visualisation is not essential to the function, it serves as a practical aid to better understand the process. The data is then exported to an Excel document and subsequently imported into NX. As the model undergoes parametric modifications, the simulation adapts the geometry to account for deviations in the robot's position from the centre of the tank.

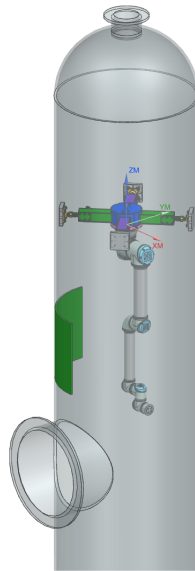


Figure 22: Tank simulation

Figure 22 displays the mock-up of the tank, jig, and robot. The green section shows the area for grinding. The error calculated in Python will adjust the model to the correct position. The code can be seen in appendix B.

The green area, which shows the region to be ground, can be adjusted to the correct position and size by modifying the parameters we have set. Currently, the parameters are altered in NX, but they can also be integrated into the Python script for a more streamlined process.

3.10 Refinement of concept 3

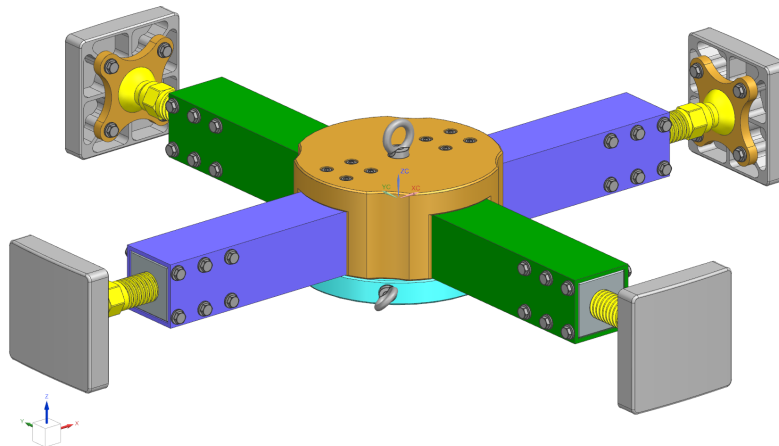


Figure 23: Version 3 of jig

The third version of the jig builds upon the previous two iterations. Version 3 is more detailed and further refined. A notable advancement in this design is the significant reduction in weight. Version 3 can be seen in figure 23.

Version two had a total weight of 49 kg (excluding screws and small supporting parts). Much of this weight stemmed from the feet of the jig, which are responsible for holding the jig in place. Each foot is situated at the end of a beam, resulting in four feet in total.

Each foot comprises three main components: the shoe, the bolt, and the nut. There is also some supporting hardware. The bolt and nut feature a M36x6 trapezoidal thread, which is preferred in applications involving substantial load transfers. The bolt is connected to the shoe via a ball joint connection, enabling the jig to align in place more easily, while also preventing bending moments from being transferred to the tank walls. The cavities on the foot and the ball have slightly different diameters, providing some wiggle room between the shoe and the bolt. In theory, preventing binding of the two parts.

The pre-optimized version of the foot weighed a total of 8.2 kg, with much of this weight originating from the bolt and nut. Both components are made of steel. To reduce the weight, one option could have been to replace the steel with a lighter material, such as aluminium. However, aluminium is unsuitable for this application, as it is soft and prone to smearing and thread damage. Consequently, we sought to optimize other parts of the foot first.

In previous versions, the shoe was designed as a single solid piece of aluminium, weighing 2.9 kg. This component was possible to optimize for weight reduction. The shoe's purpose is to create a large surface area for load distribution and creating friction between the wall and the jig. By reducing the size of the shoe and hollowing out much of the material, we achieved a substantial weight reduction, as shown in figure 24. The new weight of the shoe is 0.9 kg, which is over a third of the original weight, amounting to an 8 kg reduction in the jig's total weight.

The main hub also underwent design changes. The hub now supports hardware for the two beams, as illustrated in purple in Fig. 23. It also features cutouts on the side of the wall, not only reducing some weight but also enabling the baseplate of the jig, to be more easily lifted into place using two wires. The baseplate has two eye bolts to facilitate lifting.

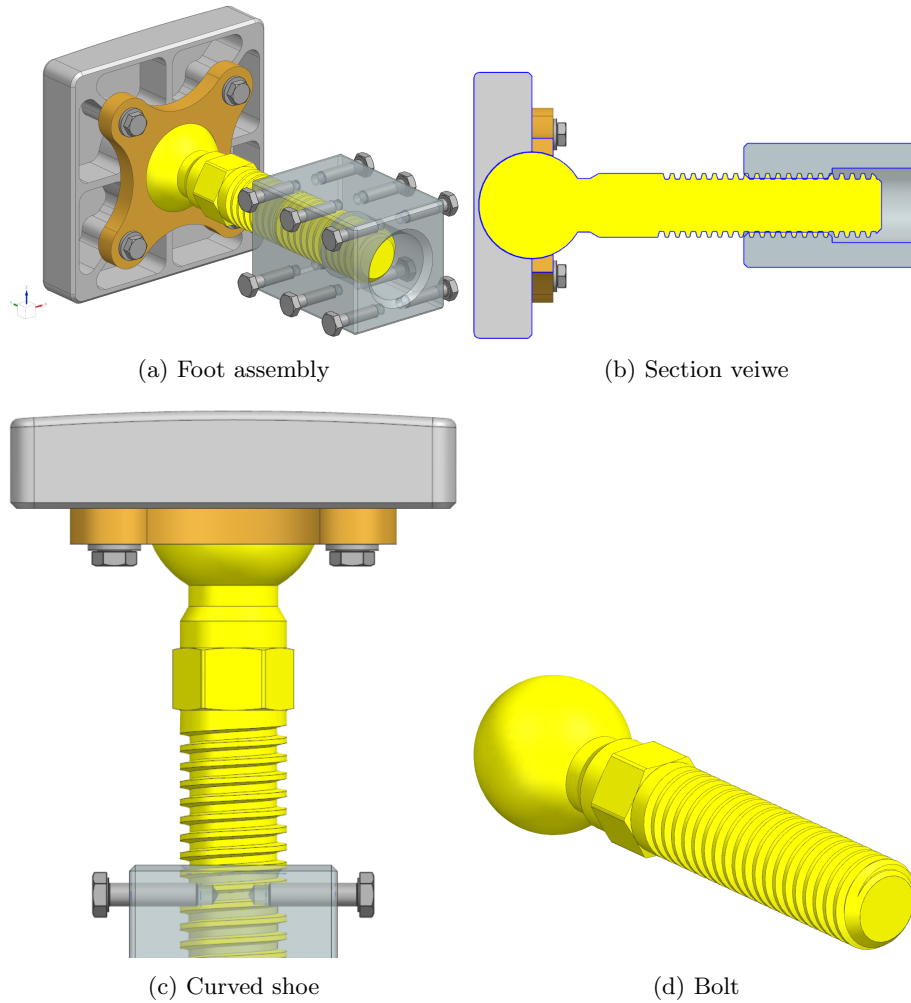


Figure 24: Foot assembly

The total weight of jig version 3 has been reduced from 49 kg to 43 kg. Although the overall weight reduction is not as significant as initially anticipated. But due to the addition of washers and bolts to the assembly, it still represents a considerable weight reduction.

3.11 Clamping force

In order for the jig to be mounted inside the tank, the feet need to be extended. In our concept, this is achieved by tightening the bolt on the foot assembly. There are two factors that determine how well the jig is mounted on the walls: how tight the bolt is screwed in, and the friction between the walls and the shoe.

In order to calculate how much torque the bolt needs to be tightened with, we must know how much force we need to apply, so the jig does not slide down. To calculate the required frictional force, we need a coefficient of friction. From the “Verksted Handbook,” we can find a table of coefficients of friction, and the coefficient for steel against bronze is 0.15 [9]. Although the shoe is made of aluminium, we assumed that the coefficient of friction would be about the same. But we would need to account for the uncertainty in our calculations. With this information, we can calculate the force of the bolt that must be exerted to hold the jig up. From the calculation in equation 7 we know that a theoretical max force the jig will experience is 259.5 N. Although this calculation is done in a system with only two feet, we still assume that this is still the case. This will give an inherent safety factor in our calculations. This force does not account for the weight of the jig that is 43 kg. Which we need to add in this calculation.

$$G = 259.5\text{N} + 222\text{N} = 481\text{N}$$

$$F = \frac{G}{\mu} = \frac{481\text{N}}{0.18} = 2675\text{N} \quad (19)$$

With this, we can calculate the moment that the bolt need to assert. The constant k is a thread friction constant that was provided by Effe.

$$M = F \cdot r \cdot k \quad (20)$$

$$M = 2675\text{N} \cdot 0.018\text{m} \cdot 0.2 = 9.63\text{Nm} \quad (21)$$

This is the minimum load required to hold the jig in place. There are other factors that tightening the bolt addresses. By tightening the bolt, we pre-tension the jig, which may help improve its rigidity. The bolt is also exposed to significant vibrations from the grinding operation. If the bolt is not pre-tensioned, it might loosen during the operation. In these calculations, there are no safety factors included, so it would not be wise to rely on them entirely.

3.12 Automation and robot programming

Learning new programming languages and setting up programming IDE's can be time-consuming, but it is a necessary step towards gaining a comprehensive understanding of the system. In addition, exploring the possibilities of the programming language in combination with the system can unlock new capabilities and enhance the efficiency of the system.

Before starting with the programming, it is important to get a complete look at the system to weigh different aspects and decide the most sensible way to progress. Understanding of the kinematics is necessary for accurate and effective robot programming. This involves understanding the geometric properties of the robot, such as joint angles and link lengths.

Additionally, understanding the different modes of operation and safety protocols, ensure the safety and smooth operation of the robot. By implementing sensors and utilizing feedback from the system. Can significantly enhance the robot's capabilities. A clear understanding of sensor functionality, limitations, and integration with the robot's control system is necessary for improved control and stability.

3.12.1 First Version (URScript)

The initial version of the robot was based on the UR force application, which allowed different scenarios to be created, including force application from a standstill or during motion. Although this function worked well for simple force tasks on simple geometries and surfaces, more advanced geometries and orientations of the robot gave issues. More testing and research into the functionalities may have given different results, but was not prioritised. A program was developed to operate the angle grinder, dragging it over a metal plate for a set distance. This was repeated in a loop, proving that it is possible to use a robot for grinding operations. This was done using the built-in force functions from UR.

However, this method also highlighted the lack of flexibility in the solution used, and drawbacks became apparent when using an angle grinder. Maintaining a constant pressure towards the surface was not as smooth as wanted, and the robot's force measurement resolution of 5N resulted in oscillations in movement and uneven grinding operations. In section 4.2 it can be seen in figure 32. Integration of custom paths and external sensors posed a challenge. This added complexity to further investigation. The focus was on controlling the robot using only URScript for the grinding operation.

3.12.2 Second Version (URCap)

URCaps is a Java-based plugin for UR robots, with integration into PolyScope which is the graphical interface. URCaps can be created to make custom programming solutions with integrated UI for easy use and integration of external hardware and new programs. The plan for this was to create a simple listen node in Java using an XML-RPC server, this would communicate with a python script for ease of programming. Partly to make it possible to run the entire configuration on the UR controller and make it possible to use a custom program to run the force regulation.

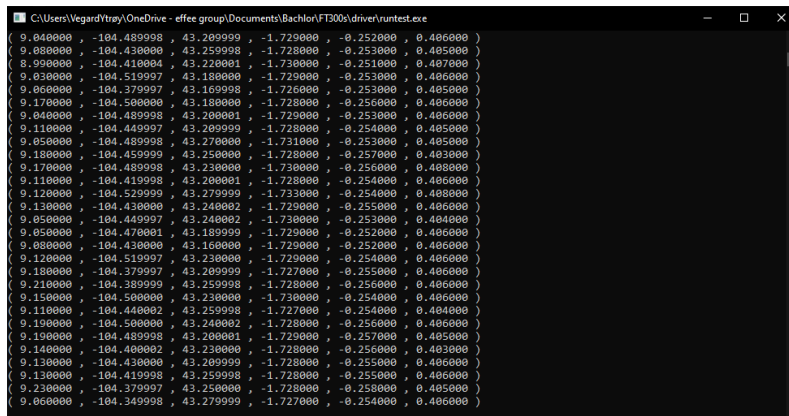
Using an XML-RPC server, a setup was made where float and integer values were sent from the robot controller to a python script running on the OS in the background. Code received a value from the robot then multiplied it before it was returned this was made to prove the functionality of the communication. When more advanced functions were implemented, issues were found. The robots run on a Linux 5.9 version, and therefore has some limitations in python versions and other supporting programs. When programming more advanced functionalities, one would have to create these. Because many of the libraries used for maths and functions are based on newer versions of python.

Issues with outdated python versions and needing to do extensive programming in Java to create the functionalities intended for the grinding operations. XML-RPC server combined with python was shown to be a slow way of sending data, especially when the UR10e is capable of a 500Hz refresh rate [28]. This version was scrapped and other ways of implementing a regulation loop on the robot were looked at.

3.12.3 Third version (FT 300-S)

An external F/T sensor was bought to replace the internal measurements of the robot, as aforementioned neither the precision nor accuracy of those measurements was good enough to be used in the regulation loop. The sensor bought was a Robotiq FT 300-S, it has a high resolution, accuracy, and refresh rate. This gives good measurements on the TCP of the robot and can be directly used in an external control loop, via the RS485 communication on the unit. The data is read directly on a computer using a RS485 to USB converter, and from there decoded in to raw values of the 3-axis and rotational axis around them. Raw data in the form of a vector X, Y, Z, Rx, Ry, Rz is used in the program.

The Robotiq FT 300-S had some open source software available online, this was written in C. There were also some applications for running and testing the sensor as is, but there was no way to use this in an external application. Therefore, a script running the C code was made, where a compiler was needed to create the application from the code. MinGW was installed and used as the C compiler for the code, this is a terminal based compiler. The output from the code supplied by Robotiq is a string displayed in a terminal window as seen in figure 25.



```
C:\Users\Vegard\OneDrive - efree group\Documents\Bachlor(FT300S)\driver\runtest.exe
( 9.040000 , -104.489998 , 43.209999 , -1.729000 , -0.252000 , 0.406000 )
( 9.080000 , -104.430000 , 43.259998 , -1.728000 , -0.253000 , 0.405000 )
( 8.990000 , -104.410004 , 43.220001 , -1.730000 , -0.251000 , 0.407000 )
( 9.030000 , -104.519997 , 43.180000 , -1.729000 , -0.253000 , 0.406000 )
( 9.060000 , -104.379997 , 43.169998 , -1.728000 , -0.252000 , 0.405000 )
( 9.170000 , -104.500000 , 43.180000 , -1.728000 , -0.256000 , 0.406000 )
( 9.040000 , -104.489998 , 43.200001 , -1.729000 , -0.253000 , 0.406000 )
( 9.110000 , -104.449997 , 43.209999 , -1.728000 , -0.254000 , 0.405000 )
( 9.050000 , -104.489998 , 43.270000 , -1.731000 , -0.253000 , 0.405000 )
( 9.180000 , -104.459999 , 43.250000 , -1.728000 , -0.257000 , 0.403000 )
( 9.170000 , -104.489998 , 43.230000 , -1.730000 , -0.256000 , 0.408000 )
( 9.110000 , -104.419998 , 43.200001 , -1.728000 , -0.254000 , 0.406000 )
( 9.120000 , -104.529999 , 43.279999 , -1.733000 , -0.254000 , 0.408000 )
( 9.130000 , -104.430000 , 43.240002 , -1.729000 , -0.255000 , 0.406000 )
( 9.050000 , -104.449997 , 43.240002 , -1.730000 , -0.253000 , 0.404000 )
( 9.050000 , -104.470001 , 43.189999 , -1.729000 , -0.252000 , 0.406000 )
( 9.080000 , -104.430000 , 43.160000 , -1.729000 , -0.252000 , 0.406000 )
( 9.120000 , -104.519997 , 43.230000 , -1.729000 , -0.254000 , 0.406000 )
( 9.180000 , -104.379997 , 43.209999 , -1.727000 , -0.255000 , 0.406000 )
( 9.210000 , -104.309999 , 43.259998 , -1.728000 , -0.256000 , 0.406000 )
( 9.150000 , -104.500000 , 43.220000 , -1.728000 , -0.254000 , 0.406000 )
( 9.110000 , -104.440002 , 43.259998 , -1.727000 , -0.254000 , 0.404000 )
( 9.190000 , -104.500000 , 43.240002 , -1.728000 , -0.256000 , 0.406000 )
( 9.190000 , -104.489998 , 43.200001 , -1.729000 , -0.257000 , 0.405000 )
( 9.140000 , -104.400002 , 43.230000 , -1.728000 , -0.256000 , 0.403000 )
( 9.130000 , -104.430000 , 43.209999 , -1.728000 , -0.255000 , 0.406000 )
( 9.130000 , -104.419998 , 43.259998 , -1.728000 , -0.255000 , 0.406000 )
( 9.230000 , -104.379997 , 43.250000 , -1.728000 , -0.258000 , 0.405000 )
( 9.060000 , -104.349998 , 43.279999 , -1.727000 , -0.254000 , 0.406000 )
```

Figure 25: Terminal output from FT 300-S

Python was the language of choice for storing, processing, and visualizing the data due to its simplicity and versatility. However, the code for reading the data was originally written in C, which presented a challenge in terms of integrating it with the Python code. Setting up a C compiler, particularly MinGW, was necessary to address this challenge.

Different communication platforms, such as TCP/IP, various bus or serial communication options, a shared terminal window, and a shared document, were explored to establish a connection between the C and Python scripts. However, none of these platforms were deemed suitable.

Ultimately, a library called Ctypes was employed, enabling Python to run C functions and code within the script. This library made it possible to run the code for receiving data and communicating with the sensor directly in Python, eliminating the need for a separate C script. Ctypes is a library that enables the running and manipulation of C code and data in Python, and it was used to read a .so file containing the C functions. A .so file is a dynamically shared library [32].

Once the data was accessible in Python, it was processed and visualized using Matplotlib. This is a popular Python library for creating static, animated, and interactive visualizations in Python. The data was graphed to give a clear understanding of the forces at work in the system. Figure 32 in section 4.2 is an example of this.

In order to implement the script from C in Python, it was necessary to have knowledge of both languages. However, C is not a language we have any previous knowledge of, therefore it required additional effort to learn the basics of the language before integrating it with Python.

3.12.4 Fourth Version (C++)

In order to begin the fourth version, the first step was to set up a suitable programming environment on the computer. Microsoft Visual Studio was chosen as the development platform, as it offers a comprehensive set of tools for C++ programming. Although we had some experience with programming in C++-like environments, such as Arduino, setting up Visual Studio was still a challenging task. It was necessary to download and install the software, configure the development environment, and set up the necessary libraries and plugins for the project.

Once the development environment was set up, the next step was to implement the C code used for communication with the Robotiq FT 300-S sensor into the C++ code for controlling the robot. Although the C code does not run simultaneously with the C++ code, the functions from the C code are available to run within the C++ code. In order to include the C code within the C++ code, it was necessary to declare it as C code within the C++ code using “extern ”C” ...” syntax [10]. This allowed the C++ code to recognise and use the C functions. The C code was then compiled along with the C++ code using the C++ compiler, rather than separately compiling the C code and linking it to the C++ code. By doing this, we were able to leverage the power and flexibility of C++ programming while still having access to the robust communication capabilities of the Robotiq sensor.

As the project progressed, we discovered that the robot’s built-in collision detection and fault tolerance settings were too restrictive for the project’s needs. In order to overcome this, we adjusted the settings to increase the tolerances before faults and collision detection would be triggered. This allowed the robot to move more freely and perform more complex tasks without being hindered by overly cautious safety protocols.

In addition to adjusting the robot’s settings, we spent a significant amount of time reading through the UR-RTDE library documentation and familiarising ourselves with the various functions available [35]. This library provides a range of useful functions for controlling and communicating with UR’s robots. A simple PID controller was utilised for controlling the Z-axis movement of the robot.

However, implementing the PID controller proved to be a challenging task, and we spent considerable time troubleshooting and testing the controller to tune its response. Despite our efforts, the controller did not work as intended. As a result, it was decided to start from scratch and create a new version of the controller. In this new version, an interface was made, a live data stream, and a live

tuning was given priority. The live data feed would be particularly helpful for tuning the controller in real-time and seeing how it reacts to different parameters.

To assist in the development process, several functions were created that could be used within the code, including `approxEqual`, `increaseInt`, `decreaseInt`, and `wait`. The `approxEqual` function checks if two numbers are within a range of each other, while the `increaseInt` and `decreaseInt` functions increase or decrease an integer by 1, respectively. Finally, the `wait` function checks if the last command was sent less than 2ms ago, and if it was, it waits for the remaining time before continuing. These functions helped streamline the development process and make the code more efficient.

3.12.5 Fifth Version (GUI using Qt)

The graphical interface was set up using Qt. The Qt environment needs to be configured to work with the desired programming language and development platform. Once configured, a new project can be created in the Qt environment. The Qt designer tool was then used to create the graphical user interface (GUI) for the application, by dragging and dropping various GUI widgets, and configuring their properties as seen in figure 26. Once the GUI design is complete, the application code needs to be written, connecting the various GUI elements to the application logic, typically using signal and slot connections. Using the Qt test framework the code was tested in Visual Studio and deployed, a version of the interface can be seen in figure 27.

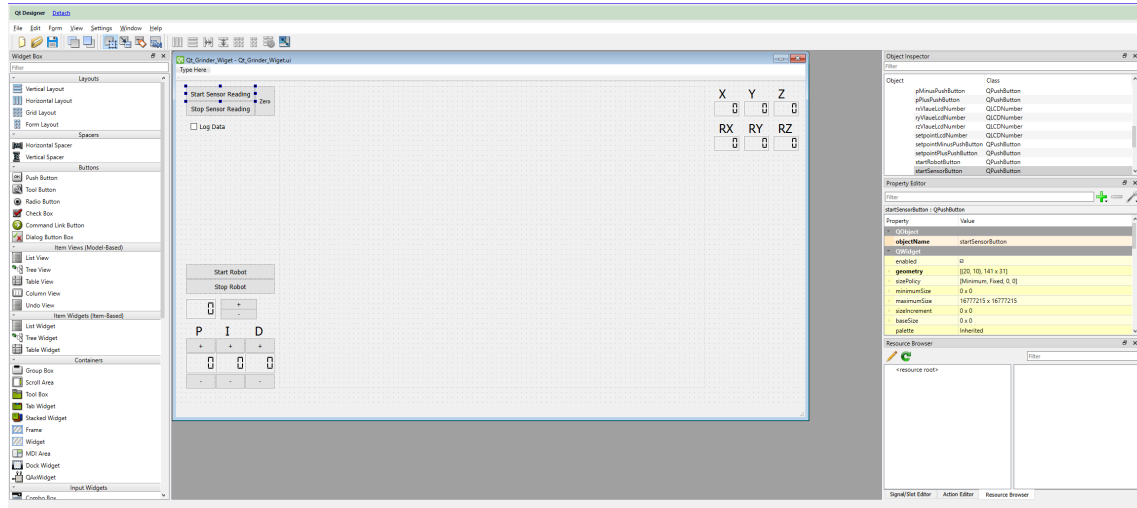


Figure 26: Qt interface for designing GUI in Visual Studio

The code from the original script was then transferred to the graphical interface. The code was cleaned up to improve its appearance and performance by restructuring it. Some processes were threaded to make them independent of a while loop. This helped to enhance the performance and stability of the code, as well as improving its overall user experience. Because some processes were now started in the background and the whole program didn't have to wait for certain parts to start. The cleaner code structure made it easier to modify, maintain and add new features to the application in the future.

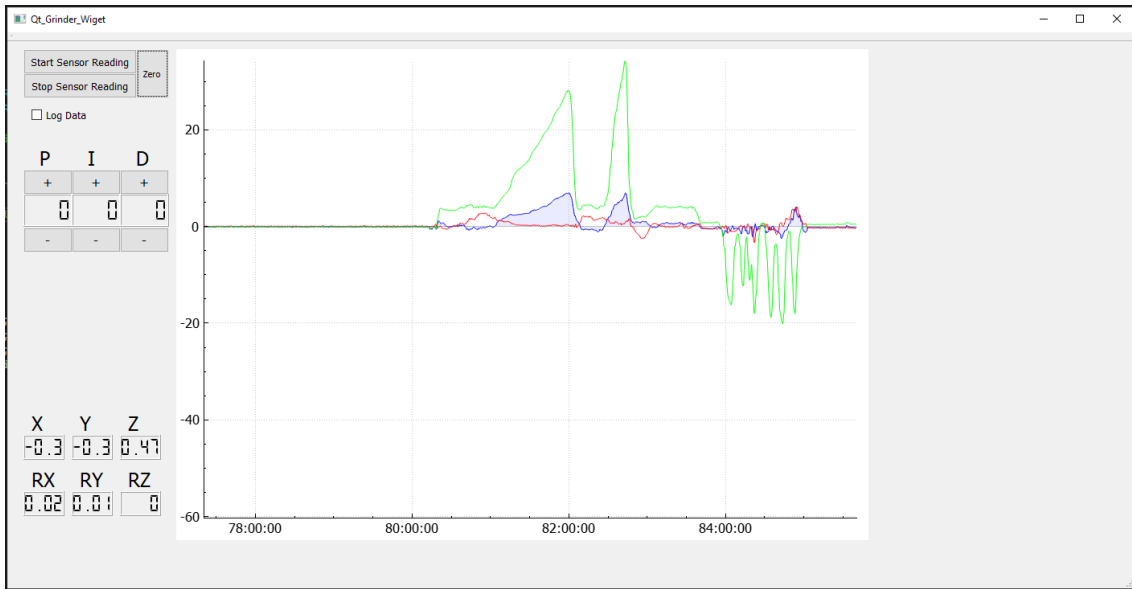


Figure 27: First working version of the GUI with a plotter

Furthermore, buttons and counters were created to provide easy control over PID parameters and visualize data from the F/T sensor. The GUI included live plots of the data stream, with the current version displaying graphs of the forces acting on the X, Y, and Z axis. To overcome issues with running the F/T sensor with the GUI, threading was implemented to ensure that the sensor data was consistently and reliably displayed. After the data is acquired from the sensor, it is put into a matrix calculation seen in equation 12. This converts the forces experienced by the sensor to actual forces applied on the TCP of the robot configuration.

The implementation of the new PID controller in Qt involved displaying the parameters and options for live tuning. While exploring more advanced control methods, such as the Kalman filter, it was decided to create the code in such a way that it could be implemented in the future. Although the Kalman filter was not implemented at the current stage, exploring this control method helped in improving the current control loop. It was noted that the noise of the system appeared to be Gaussian, which would suit the Kalman filter well and provide a more reliable data feed while the system is active. Some of the noise issues could be fixed by using a low-pass filter, but the current version works well, and it was deemed unnecessary at this point.

During testing and troubleshooting, numerous integration errors were encountered, with different aspects of the code not wanting to run simultaneously. This required extensive debugging and optimisation of the code to ensure that all aspects of the system worked together seamlessly, providing accurate and reliable control of the system. Despite these challenges, the final implementation provided a control system with a user-friendly interface for monitoring the F/T sensor and controlling the system's control parameters.

3.12.6 Sixth Version (PID)

The implementation and customization of the PID controller for use with the robot RTDE communication and Qt proved challenging. The controller was tuned and fixed manually in the beginning, with the system being restarted each time due to issues with live updating the PID parameters. The parameters were initiated once at the start of the program, which added extra complexity to the tuning process. Despite these challenges, a working force compensation system was developed. However, the response of the system was unstable due to the lack of proper tuning. Additionally, it proved difficult to fine-tune the system and improve its stability.

After encountering issues with live updating the PID parameters. An update function was added to the PID library [26]. However, an error was found in the `pid.Update()` function that resulted in

the incorrect output of an integer value of 0, rather than the desired floating-point number, due to incorrect division of integer numbers. This error was fixed, enabling the live updating of the PID parameters without further issues.

In addition, simple movement in the x and y axis was added to test the force application. To calculate the movement, a vector between two points was determined, and the desired speed to travel that distance was calculated. The robot's movement was then calculated for a 2 ms interval, which is the speed of the control loop. This was done manually for this version, for all new positions the calculations had to be done. The force compensation only worked on the robot Z-axis, so movement was limited to the XY plane.

To ensure proper calibration of the PID controller, the best parameters found at this point were determined to be $P = 0.5$, $I = 0.8$, and $D = 5.7$. The conversion used is direct from N to m and a down-scaling factor for P and I of $1/1000000$ and a D of $1/100000000$. These parameters were calibrated to provide the best stability and accuracy of the system in that configuration.



Figure 28: Setup for first grinding tests (safety guard removed for illustrative purposes)

The initial grinding test using the new system proved successful, as the robot maintained a stable force and effectively ground the designated area. Although some noise was present in the system, it produced a mostly stable output with minimal undesirable traits. However, the operation generated significant vibrations that could potentially lead to damage and problems within the robot and the overall system. As illustrated in figure 28, the forces are directly transferred to the robot arm, intensifying the strain on the system. See section 5.1 and 5.3.2 for a more comprehensive reasoning around the issue.

3.12.7 Seventh Version (Multiple Grinding Orientations)

The integration of the interpolation function into the robot's control system is a crucial factor in the development of its grinding capabilities. This function enables the robot to follow a path between two points. As illustrated in Figure 29a, the interpolation function calculates new positions for every requested step along a given vector, allowing the robot to move in 3D space. The step interval is determined by a speed calculation as seen in Equation 23 that takes the length of the vector between the starting and stopping position (equation 22) and calculates the necessary movement for each cycle of the robot controller, which runs at a frequency of 500Hz. These increments are added together to determine how far along the vector the robot is (equation 24). For a more detailed view on how these functions operate, see appendix A and the functions interpolate, calculateIncrements

and robotRegulation.

$$v = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \quad (22)$$

$$\alpha V = \begin{cases} \frac{v}{|v|} \cdot \frac{|speed \cdot \Delta t|}{1000 \cdot |v|} & \text{if } |v| > 10^{-6} \\ 0.0 & \text{otherwise} \end{cases} \quad (23)$$

$$V_{tot} = \sum_{n=1}^{\infty} \alpha V_n \quad (24)$$

The PID controller is responsible for regulating the robot's movement only for the grinding application. The controller takes the force applied on the TCP of the robot and calculates a vector with origo in the TCP and direction in the Z-axis of TCP. This vector is then transposed in to the base coordinate system, to give vectors in the X, Y, Z axis. These vectors are combined with the interpolated point that the robot is set to goto, thus determining the next location. The regulation essentially happens in a single axis, with the addition of time, giving a 2D system. The PID has no information on the robot's position or other external factors. It only takes into account the current and previous measurements of force in the Z-axis of the TCP.

$$\begin{aligned} R &= R_z(\alpha) R_y(\beta) R_x(\gamma) \\ &= \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{bmatrix} \\ &= \begin{bmatrix} \cos \alpha \cos \beta & \cos \alpha \sin \beta \sin \gamma - \sin \alpha \cos \gamma & \cos \alpha \sin \beta \cos \gamma + \sin \alpha \sin \gamma \\ \sin \alpha \cos \beta & \sin \alpha \sin \beta \sin \gamma + \cos \alpha \cos \gamma & \sin \alpha \sin \beta \cos \gamma - \cos \alpha \sin \gamma \\ -\sin \beta & \cos \beta \sin \gamma & \cos \beta \cos \gamma \end{bmatrix} \end{aligned} \quad (25)$$

$$V_{base} = [X \quad Y \quad Z] \quad (26)$$

$$V_{TCP} = [X_{TCP} \quad Y_{TCP} \quad Z_{TCP}] \quad (27)$$

$$\begin{aligned} \mathbf{T} &= \begin{bmatrix} R & V_{base}^T \\ 0 & 1.0 \end{bmatrix} \\ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} &= \begin{bmatrix} \cos \alpha \cos \beta & \cos \alpha \sin \beta \sin \gamma - \sin \alpha \cos \gamma & \cos \alpha \sin \beta \cos \gamma + \sin \alpha \sin \gamma & X \\ \sin \alpha \cos \beta & \sin \alpha \sin \beta \sin \gamma + \cos \alpha \cos \gamma & \sin \alpha \sin \beta \cos \gamma - \cos \alpha \sin \gamma & Y \\ -\sin \beta & \cos \beta \sin \gamma & \cos \beta \cos \gamma & Z \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \end{aligned} \quad (28)$$

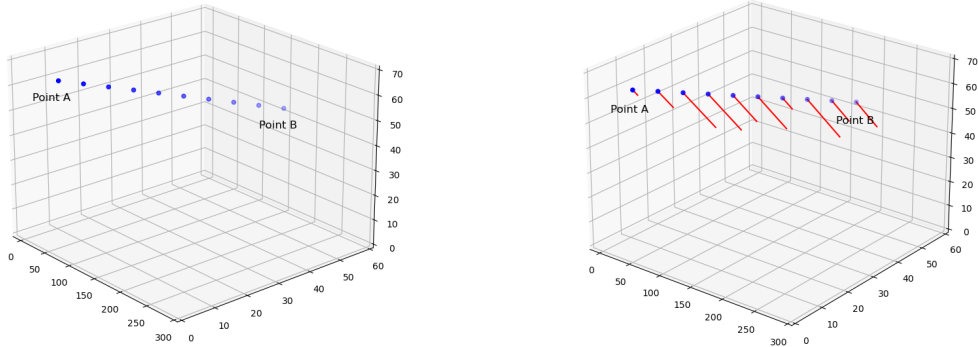
$$H_{TCP} = [V_{TCP} \quad 1.0] \quad (29)$$

$$H_{base} = \mathbf{T} \cdot \mathbf{H}_{TCP} \quad (30)$$

The PID controller returns a value that represents the amount by which the robot needs to move towards or away from the surface. This value is then used as the input for the Z_{TCP} variable in equation 27, which determines the desired movement in the TCP z-axis. In equation 26, the vector being added is 0.0,0.0,0.0, as it represents the starting point from the origin of the next interpolated point. If the current position were to be added, the resulting vector would be the old position plus the PID control vector. The rotational matrix in equation 25 is computed based on the Rx, Ry, and Rz values of the current position, which indicate the rotation of the axes from their original position. These values are denoted as γ , β , and α , respectively, and are fed into equation 25 to calculate the rotational matrix.

The vector V_{TCP} is converted into a homogeneous vector H_{TCP} by adding a scalar value of 1.0 at the end. The 4x4 pose transformation matrix \mathbf{T} can be constructed by adding the transposed vector V_{base} to the rotational matrix R , as shown in equation 28. Multiplying the homogeneous vector H_{TCP} from equation 29 with the pose transformation matrix produces the homogeneous

vector in base coordinates, which is shown in equation 30. Extracting the first three elements of H_{base} gives the transposed vector from the TCP coordinate system to the base coordinate system.



(a) Interpolation points from point A to B (b) Interpolation with compensation vectors

Figure 29: Visualization of interpolation and interpolation with regulation vectors (only for illustrative purposes)

Before sending the command to move the robot, data from various processes are combined to determine the correct target position. The PID calculation output is sent through the transpose calculation, and the subsequent point is extracted from the interpolation function. The position data from both functions are combined, resulting in a vector representing the robot’s next position. As illustrated in Figure 29b, the end of the red line indicates the new position for the robot’s TCP, with an offset from the interpolated point.

The two steps are performed separately to allow the regulator to function atop other paths. Instead of being the sole controlling entity of the robot, the regulator can theoretically be fed with poses along any given path. The regulator then adds its vector to the pose, resulting in the output displayed in Figure 29b.

3.12.8 Current system state

The system is now operational but not fully complete, with numerous bugs requiring fixing. The current graphical user interface (GUI) can be seen in Figure 31. This version functions, and has no issues. However, the background program has some issues. The main issues lie in communication between the program and external components, such as the FT 300-S and UR10e. Improving communication setup would likely resolve most of these issues. The GUI has been updated with an enhanced layout and new features, including start and stop controls for the robot and a functional “log data” button. The graph now also includes a legend for better information display. The program in its entirety can be seen as a schematic in Figure 30.

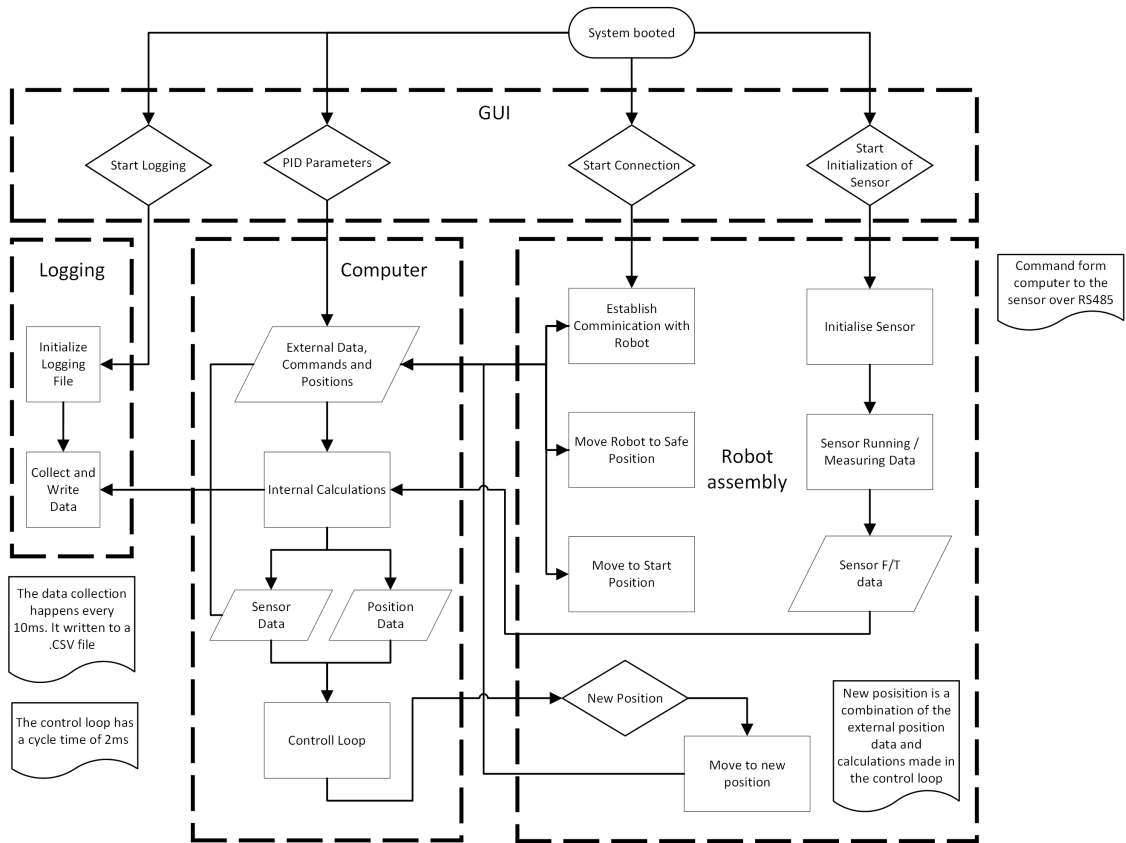


Figure 30: Schematic of the robot program

Minor improvements have been made to the PID, particularly in the internal variables and some calculations, to enhance reliability and speed. These modifications do not affect the calculations themselves, but rather how the program manages them. Logging functionalities have been added to the system, enabling information extraction during operation. The current setup provides details on position calculations, PID controller status, and force/torque sensor data.

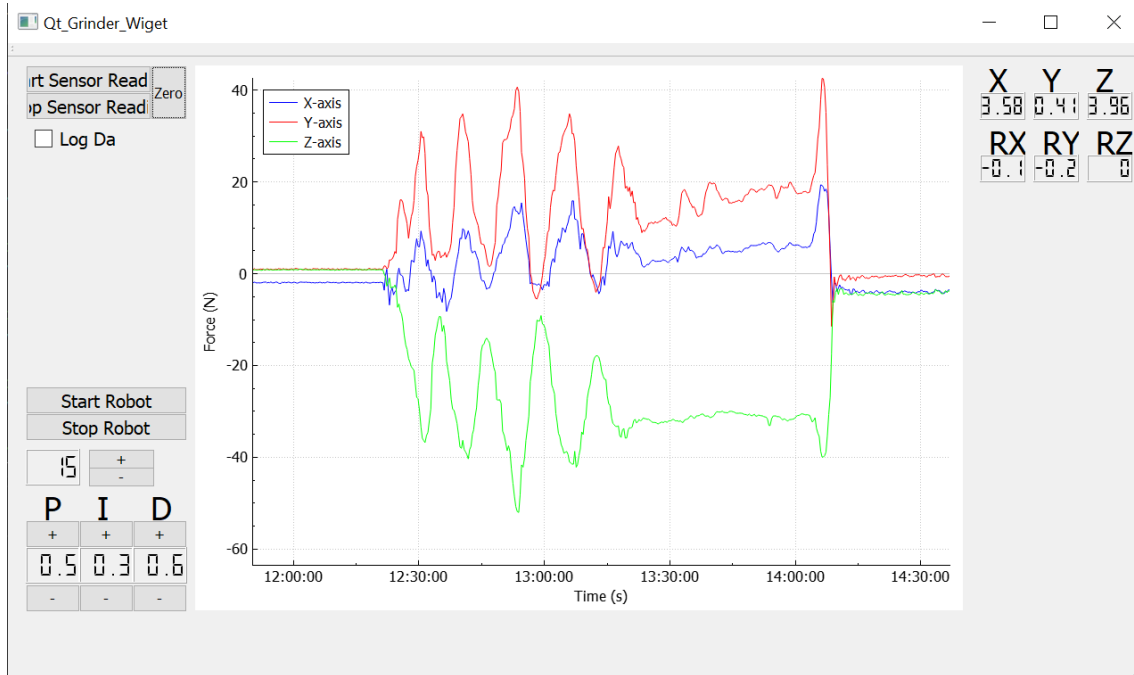


Figure 31: Current version of GUI

Challenges arose during the implementation of the logging function, primarily due to issues with directory naming for saving generated files. The logging function creates a unique file name, tagged with the exact date and time down to the millisecond, and saves it to a predetermined path. The problem stemmed from the presence of Norwegian letters, specifically “ø”, in the path name. Changing the path to one without these letters resolved the saving issue. Additionally, issues with misplaced data within the file were addressed and corrected.

The system in its current state has done some limited grinding tests, and produced data from these tests. The tests have been done in a semi-controlled environment, but some factors have not been controlled. Especially climatic variables, temperature, humidity and such has been seen as non-important to the experiments. But other variables such as the angle of the grinder, speed and location of the grinding passes which play a large role in the quality and success of the grinding has been under-prioritised. This is mainly due to a higher priority of functionality and not prioritising the metallurgical aspects of the operation.

4 Results

In the results section, we list our study's solutions and findings. We focus on how well these results answer our research questions and meet the project's goals in real-world situations. Some of the results given here are not the only way to solve the problems given, because it is a solution for the current state of the project. The results should to some extent be viewed as a single entity. This is because all decisions made through this project play on each other to provide a solution to the original problem presented in the introduction chapter 1. In appendices E we have included machine drawings of all the parts in the jig assembly, as well as assembly drawings. All code used and created during the project can be found in appendices A, B.

4.1 Assembly

The developed product architecture consists of software and hardware components for grinding the inside of a process tank. The jig, which is a critical component of the system, was designed based on set design criteria and specifications, and rough sketches of the concept were created. Calculations were performed to estimate the forces the jig would encounter, which provided a guideline for the concept design work.

Several design concepts were considered for the mounting jig, with Concept 3 being the final design choice due to its simplicity and ease of assembly inside the tank (see Figure 10). The final dimensions of the jig were height 150 mm, diameter of 1038 mm \pm 72 mm and a total weight of 43kg. The robot used for the task was a UR10e from Universal Robotics, which was able to perform the grinding operation using the equipment provided (see Figure 13). Toolpaths were generated using Siemens NX, with initial attempts to use RoboDK proving to be cumbersome for more advanced tasks (see Figure 14).

The initial three versions of the robot control solution did not yield any results that could be integrated into the current system. They only provided negative outcomes that guided our development decisions. The fourth version marked a significant shift, as it laid the foundation that the current version directly builds upon. In this version, we set up a C++ environment in Microsoft Visual Studio, incorporating C code into the C++ code to control the robot and communicate with the Robotiq FT 300-S sensor. Some adjustments were made to the robot's fault tolerance settings. Furthermore, we explored the UR-RTDE library for controlling the robot, and attempted to implement a PID controller, which didn't work as expected. Including creating several functions to streamline the development process.

Building on this, the fifth version focused on creating a GUI using Qt, with the code from the original script transferred and cleaned up for better performance. The GUI provided control over PID parameters and visualized F/T sensor data. The sixth version of the robot control system involved implementing and customising a PID controller for use with the robot RTDE communication and Qt. A working force compensation system, An Update function was added to the PID library. The PID controller was properly calibrated, with optimal parameters being $P = 0.5$, $I = 0.8$, and $D = 5.7$. The force compensation worked only on the robot's Z-axis, limiting movement to the XY plane.

The initial grinding test using the new system was successful, with the robot maintaining a stable force and effectively grinding the designated area. The seventh version of the robot control system incorporated an interpolation function, enabling the robot to follow a path between two points in 3D space by calculating new positions at each requested step along a given vector. The PID Z-axis vector is transposed into the base coordinate system to give vectors in the X, Y, Z base axes, which are combined with the interpolated point that the robot is set to go to, determining the next location. The two steps are performed separately to allow the regulator to function atop other paths. Resulting in the output displayed in Figure 29b.

4.2 Regulation and Control Loop

The results from the custom control loop is compared to a test done with UR’s own force solution, which uses the built force sensors. We base the results on the kinematic model of the robot and the torque sensors in the wrists of the robot. The first result seen in figure 32 show the forces applied to the sensor using the UR force application. The results were taken on a moving pass with the angle grinder on, grinding the inside of a metal cylinder. As seen in figure 32 on the green line “Force Z axis” between 10 and 20 seconds the force applied seem to oscillate. Data analysis show that the in the interval from 10 to 16 seconds the mean force applied in the z-axis is -33.05N , but that is with a standard deviation of 11.82N and maximum and minimum values of -4.34N and -55.82N .

This data shows that the operation isn’t stable and could potentially give uneven grinding results. More and repetitive testing need to be done to determine this, but the results shown here give a good indication of the results it could produce. One thing to note on these numbers are that they are not converted via the matrix equation 12, but for the force going directly on the z-axis can be seen with a static factor of -1. The results would therefore be the same with the exception of a different prefix.

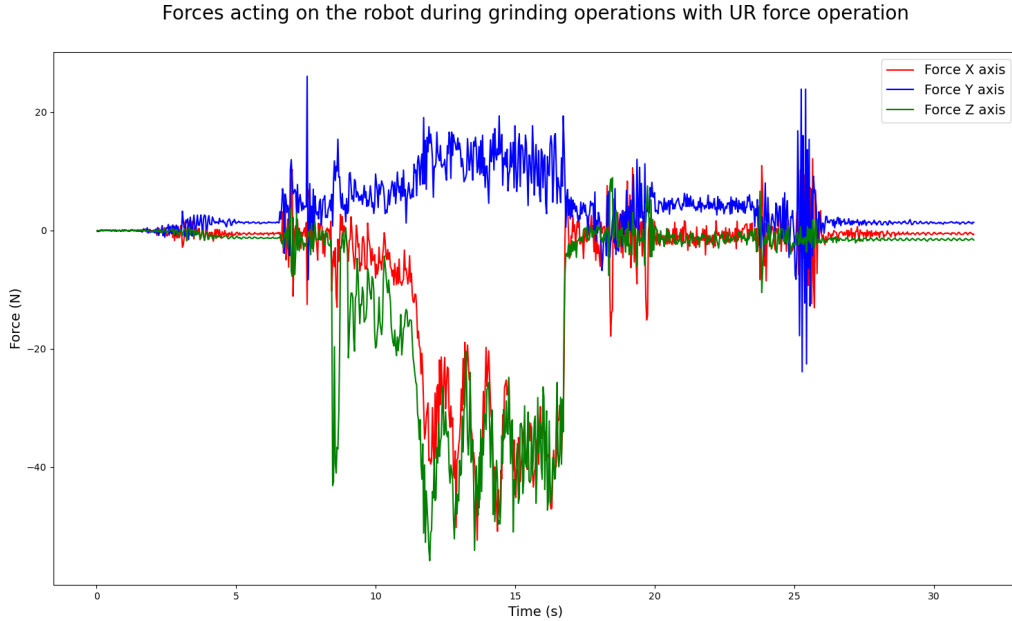


Figure 32: FT300-S measurements from UR force compensation

In a later version, a custom force application was designed, utilizing the FT300-S force/torque sensor mounted on the robot. This application is a custom PID controller that uses the current force applied to the sensor. Similar to the result above, the best section of the graph is extracted and analysed. The graph shown in figure 33 examines the part from 16:47:30 until the end of the data set. This section maintains a stable setpoint of 51N in the TCP z-axis, with the mean for this section being 51.094N . The standard deviation is 3.02N , and the variance is 9.12N . The max and minimum values recorded are 59.64 and 42.4N , respectively.

This custom application results in a system that is more stable and can provide a more constant force towards the surface than the UR system can. Notably, this system is also tune-able, and adjustments can be made while it’s running. As seen in the bottom graph in figure 33, the PID parameters are fine-tuned during operation. The system achieves optimal stability when the P-band is set at 5, I at 6, and the D-band at 10 for the controller operating in 3D interpolated space. More details on how these parameters are scaled can be found in section 3.12.6.

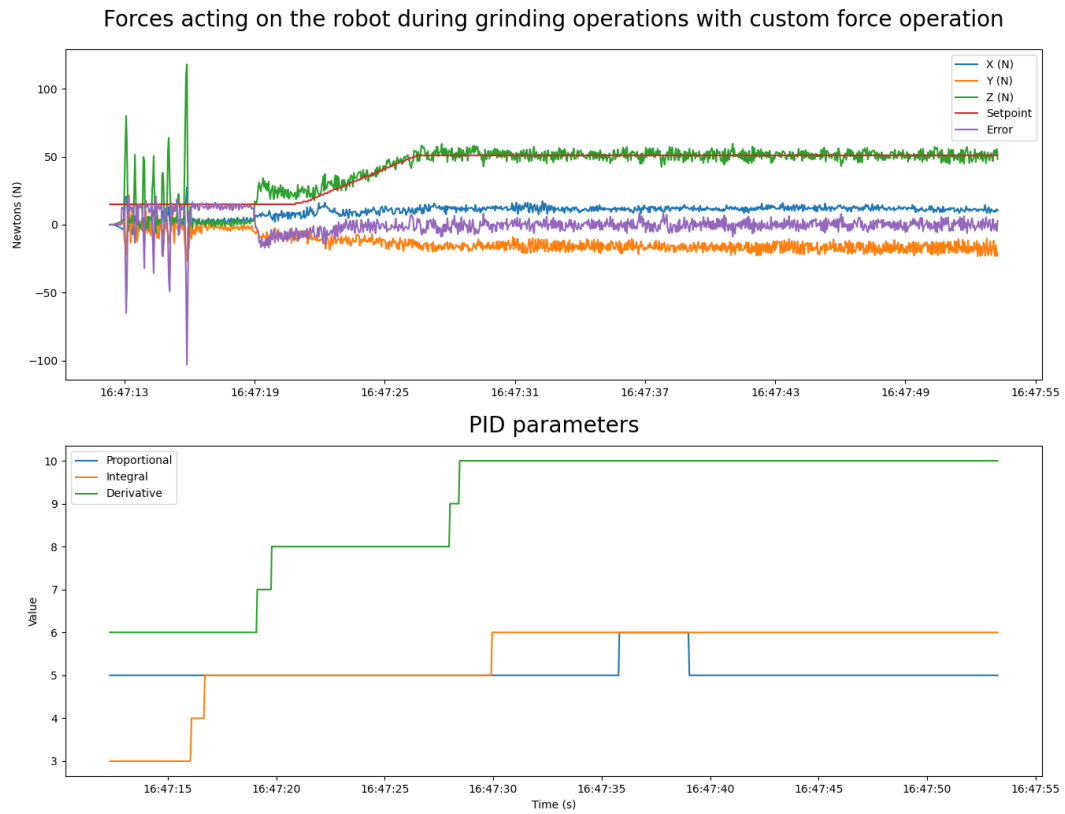


Figure 33: FT300-S measurements from custom force compensation with error, setpoint and PID values

The TCP z-axis is seen in figure 28, it's the axis going trough the entire assembly. Directly trough the FT300-S and the grinder assembly. The figure 20 shows all the leverage arms created when grinding with the angle grinder, it also shows where the center point of the z-axis.

5 Discussion

In this chapter of the bachelor thesis, we will discuss different parts of the thesis and the results. We will examine the different weakness and strength of our work, and evaluate what could have been done different or better.

5.1 Jig

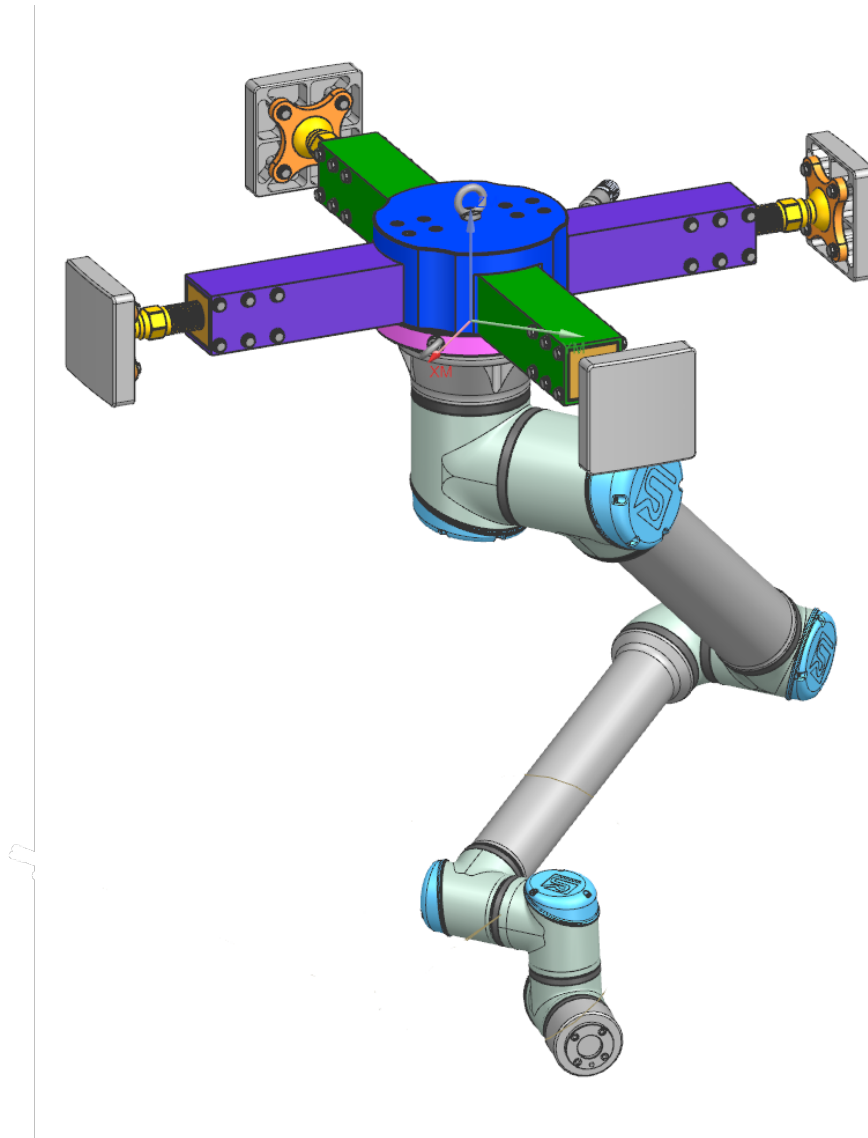


Figure 34: Final version of jig

In the early phases of the project, a feasible design emerged. The design we chose was one with four legs. It might have been a good idea to create small-scale models of the different concepts. This would allow us to test out the different mechanics of the jigs. Some of the reasons why we chose not to make physical models are the simplicity of the part. It has a narrow use application and will most likely only be made as a one-off. The design we produced has modularity built into it. As all the parts are bolted together, the different parts can be quickly changed out. If some part needs to be modified, and or changed. The rest of the jig can be used.

A crucial aspect of the jig is its rigidity. Early on in the project, some basic calculations were done

to assess how much load the jig would theoretically experience. There has not been a study done on the deflection of the jig. This would have given us an idea of how the jig might bend under operation. Ideally, a Finite Element Analysis would have been performed. However, due to time constraints and a lack of proper experience with finite element analysis, this has not been done. With a better understanding of the displacement of the jig and stress concentrations, further weight optimisations could be achieved. Although our target weight and dimension goal was reached as we have set in table 1, ideally it would be as light as possible, but as strong as possible. With the data from our grinding test, it shows that there are not a lot of forces involved. The dominant force is vibration.

Vibration is a major concern we see after the tests. The vibrations are so severe that they might cause damage to the robot. The vibration also has the potential to loosen the feet. Our calculations done on the feet in equation 21 only show the minimum required moment to hold the jig in place. The bolts would most likely need to be pre-tensioned. This would reduce the likelihood of the bolt loosening. A countermeasure that would help is to have a locknut. This nut would be tightened against the block of the foot, further pre-tensioning the bolt. To reduce vibrations from the robot.

A future solution could be creating a custom damper or cushion. This could help absorb and dissipate the vibrations produced during operation, reducing their impact on the system. However, an ultimate solution could be the implementation of an active pneumatic compensation unit. Such a unit would compensate for vibrations and other undesired movements in real-time, ensuring the stability and longevity of the system. Implementing such a solution would be more costly and complicated, but it would provide the best possible results in terms of stability and performance.

In the process of mounting the jig, ideally, we would like the jig to be mounted by one operator inside the tank. This might prove difficult as the tank is only, 1100 mm in diameter. If the operator were to lift it by hand, it could become too cramped. We added some eye bolts to make it possible to lift the parts up. If this is to be done, a lifting procedure is worth looking into. There is a pipe section on top of the tank that could be used for a lifting operation, see figure 1.

The design process could have been executed more systematically. Conducting a study on the users of the product and examining how they perform this task or similar tasks, might have provided valuable insights into the challenges and problems we could have addressed. However, we decided not to focus too much on this aspect of the design process, as the application is quite specific with a narrow use case.

During the early stages, Effe informed us that they might weld two beams together to make the solution as cost-effective as possible. Instead of delving deeper into user research, we chose to dedicate more time to integrating the entire process within the NX software.

5.2 Simulation and path generation

There are some clear benefits of using NX compared to multiple platforms. In the NX platform, we can design, analyse, and simulate seamlessly. This makes it possible to adjust models, simulations, and parameters without exporting the files. However, using NX in almost every aspect of the process has some drawbacks. NX has expensive licenses, and a cost analysis should be performed to see if the NX license would make it overall cheaper to use this platform. NX also has a steep learning curve, which might deter users from utilising it fully.

Setting up NX with the UR10e proved somewhat challenging, as this was our first experience using the machine tool builder module in NX. Implementing the machine tool builder was not straightforward, and defining the axes precisely, as required, was particularly difficult due to the limited information provided in UR's documentation.

Once the kinematic model was established, the file structure proved to be quite specific. In theory, if you wished to use a different robot, it should be simple to swap the model out, provided that the new robot is already built in NX. However, compared to RoboDK, replacing the robot in NX is notably more difficult. Siemens does provide some robot models and postprocessors, but not as many as RoboDK.

The major advantage of using NX over RoboDK for simulation is the comprehensive integration of all process aspects. In NX, we can create our models, program paths, and simulate the robot. We found that we only used RoboDK to convert g-code to UR-script, a task that NX is also capable of performing.

RoboDK doesn't provide tools to generate toolpaths on geometry, although it can execute 5-axis G-code. We encountered issues while running the G-code produced by NX. We suspect the issue lay in the method of toolpath generation in our initial tests, as shown in fig. 14a and 14b.

The first test with a 3-axis toolpath functioned as expected, with the robot's motion "sweeping" over the designated area. The UR-script produced from RoboDK was also correct. However, when programming a toolpath on a pipe section, we faced difficulties. In NX, the tool swept across the surface, but when the G-code was imported to RoboDK, the robot only tilted the TCP and proceeded in a straight line. This was likely due to the way the G-code was generated.

Upon reviewing the G-code, we noticed that only the rotation angle was programmed to alter. It appears that RoboDK interprets this as merely a rotation in the TCP, not a motion of the robot. We hypothesise that we could have bypassed this issue by altering the 5-axis postprocessor to one where the tool axis rotates, not the part. RoboDK also supports plugins for other CAM software, but unfortunately, not for NX. This plugin might have helped resolve this issue.

In conclusion, it seemed somewhat redundant to create the G-code only to import it into RoboDK. Siemens NX by default supports robotic machining, and we had access to the licenses to implement this. RoboDK does have the advantage of being able to mirror the robot's motion and position in real-time, a capability that NX lacks.

NX has proven to be a powerful tool for generating toolpaths. The process of programming the toolpath can be a bit overwhelming at first. The menu for the Variable Contour features an extensive list of submenus and parameters that need to be filled out. However, since our surface is relatively simple, many of the parameters can be left at their default settings.

We tested the motion of the toolpath on the robot, but we did not get the opportunity to use the NX-generated toolpath for grinding. The motion of the robot closely matched our simulation within NX. One challenge we encountered was how UR implements joint configurations. In NX, we can select the joint configuration, but this option is not available as a parameter in the UR script.

To work around this issue, NX employs a "moveJ" command defined by UR script [33], which instructs the robot on the angles the joints should move to. After the initial "moveJ" command is set, the subsequent movements should, in theory, follow the selected joint configuration. We experienced a few unexplained instances where the motion did not match the simulation, but these occurrences were rare.

5.3 Programming and robot control

In this section, we look at the various aspects of automation and robot programming, which are vital components of the solution presented in this paper. To ensure seamless and efficient robot operation, it is critical to employ suitable programming languages for the tasks presented. Moreover, utilizing GUI to present the system so the user can get feedback from the system.

Before starting to program, it's important to look at the system to understand its different parts and figure out the best way to move forward. Knowing how the robot's kinematic system works is key for accurate and successful programming. Overall, automation and robot programming encompass a broad range of tasks, from new programming languages and comprehending the kinematic system to implementing sensors and feedback systems.

5.3.1 Choice of Programming Language

During the initial testing and familiarising with the robot and programming languages, URscript was the first to be tested. This has a lot of predetermined functionalities delivered by UR, and functions great for simple and repeatable tasks. When the tasks get more complex we found the language difficult to customise, and got time consuming to set-up for more advanced applications. Especially if the same task is to be repeated, just with different scenarios or scenes.

These characteristics made the URscript undesirable for the exact task this thesis focus on. The creation of a UR cap was also explored, but fell under many of the same troubles as URscript, the difference here was that UR caps could make more advanced systems that would be customizable. But that also included more advanced programming in the form of Java, combined with the lack of knowledge and experience programming Java. The advanced programming needed to meet the requirements of the task would be too challenging with the time limit set. A UR cap would most likely produce the results wanted. Here the limit is be knowledge, time and resources.

We had a good understanding of the Python programming language and found a library that could communicate with the UR robots. However, using the Python interface had some difficulties, including problematic dependencies and issues installing the library correctly. Additionally, an external resource suggested that Python might be too slow for the application. As a result the same library was found for C++ and we chose to use it instead because of its high speed and our previous knowledge of the language. Although C++ presented initial challenges and a steep learning curve, it proved to be a reliable choice, offering numerous options for designing a solution and catering to various project requirements.

5.3.2 Choice of Regulation and Control

A PID controller was chosen as the regulator for this project because it's easy to work with. It covers a wide area of use cases and can be implemented easily even if there is a lack of computational power in the system. As seen in the section 4.2 the new control loop gives a significantly better result over the UR stock force compensation. This is most-likely because it's specially designed for this exact application. It has four times lower standard deviation than compare able systems, and the remaining deviation can be contributed to the nature of angle grinding. It has an unpredictable nature and causes an inherent "noise" on any system due to vibrations. These vibrations are also amplified when the grinder is in use, so much of the noise on the measurements can't be removed without physical change of the grinder itself. These vibrations will contribute to increased wear and tear and place stress on the system, exceeding the robot's rated tolerances. System degradation and damage, might not be immediate but over time can cause severe damage to the robot or other components.

But the vibrations can cause another issue, because if the system isn't tuned perfectly, the regulator might start to act on these changes. Causing oscillation and unwanted changes in the system that lead to instability. One solution that could solve the vibration problems from a regulation standpoint could be to implement a Kalman filter. This could be used to directly control the robot and replace the PID controller. It could also be used as an intermediate step to filter out most of the noise on the signal before sending it to the PID. The issue with Kalman filters is that the mathematical models employed often lack the precision needed to produce outputs suitable for their intended use cases. The accuracy in this case might not be needed, because grinding is an unstable and imprecise way of removing material.

The main issue with a Kalman filter is getting a good estimate of the two covariance values from the system. The covariance is a measurement from the process and observation noise. A future improvement could be to use artificial intelligence to estimate and calculate the controlling values within the filter [17]. Such a model could be self regulating and need less intervention from an operator to tune and control the system for every change happening. A multi sensor system would also be easier to implement with the use of a Kalman filter, a multi sensor array could also improve the quality of the grinding operation [8]. Using the AI to tune the filter values wouldn't introduce any computational delays to the process, either because the AI could be placed in the background

on an external resource.

5.4 System Architecture

The project's system architecture required decisions that brought both limits and opportunities for progress. The primary objectives have centred around achieving seamless integration of the various hardware components. In this section of the discussion, we will examine the system architecture with an emphasis on the interactions between NX, the jig, the robot (UR10e), force/torque sensor (FT300S) and C++ as the chosen programming language. We will elaborate on the role of the different software and hardware aspects and how they are integrated together, but also the shortcomings of the current system.

The current system begins with the operator taking three points from the inside of the tank with the UR10e. These points are then fed into an algorithm that calculates the position of the robot. The position of the robot is then updated in the NX simulation. In NX, the operator specifies the dimensions and position of the tank area that needs to be ground. After this, the toolpath can be generated and, if necessary, tweaked. When the toolpath is set up, a simulation of the robot motion is run to check for collisions and grounding motions. Once the simulated toolpath is verified, the code is post-processed to UR-script. The UR-script would be used as the path for the custom controller. This path provides the data for the interpolation function, which is the basis for the force regulations starting points. In the custom robot controller, we can control the feed and force of the robot on the tank surface. The path generation and the custom controller doesn't communicate or exchange data at this moment, this is the last step before a fully functioning prototype is developed.

The project has had a high focus on the integration of the different components within the project, but also make the individual components work as a standalone solution. The two main components that in its current state isn't exchangeable is the UR10e and the communication between them setup in C++. All other additions are created as individual blocks interacting with this main structure. This means that the force/torque sensor used can be easily replaced by another external sensor, or even the onboard force capabilities of the UR10e. The same goes for the software, the PID regulator within the project can be replaced with a new class, replacing the PID with e.g. a Kalman Filter. The last part would be to create a block that would make the custom controller read the paths generated in NX and regulate on top of them.

6 Conclusion

In conclusion, the findings demonstrate the effectiveness of the proposed methods and techniques. The observations and data generated through this thesis shows potential for significant improvements in the performance and capabilities of the system. Further research and development could yield even better results, potentially opening up new opportunities for the application of these technologies. The results of this study show there is room for more work in this area, and it also highlights the importance of our study in the bigger picture.

This bachelor's project aimed to find a solution for achieving remote grinding inside a tank with pitting corrosion. The tank's interior needs to be ground smooth before it can be re-welded. We found it very promising to use a 6-axis robot equipped with an angle grinder attachment. Some key challenges we encountered included mounting the robot, avoiding collisions, and managing force on the surface.

Our solution utilises NX software for design, path programming, and simulation, as well as a custom robot controller for surface force control. We discovered that using NX as a path programming and simulation software offers a promising solution. Moreover, our custom force controller demonstrates a four-fold improvement in force control compared to the built-in force control on the UR10e. We were not able to integrate the path generated from NX with the custom force torque system.

Bibliography

- [1] Karl Johan Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. v2.11b. Electronic edition. Princeton, NJ: Princeton University Press, 2008. ISBN: 978-0-691-13576-2. URL: http://www.cds.caltech.edu/~murray/books/AM08/pdf/am08-complete_28Sep12.pdf.
- [2] Kolbein Bell. *Konstruksjonsmekanikk: del 1 - Likevektslære*. no. Fagbokforlaget, 2014, pp. 5–7. ISBN: 9788245016857.
- [3] *CraftMate*. Website. Nordbo Robotics AS, 2023. URL: <https://www.nordbo-robotics.com/craftmate-platform>.
- [4] Michael G. Curran et al. ‘Angle grinder injuries in orthopedics: A case series and review of the literature’. In: *Trauma Case Reports* 32 (Apr. 2021), p. 100413. DOI: 10.1016/j.tcr.2021.100413. URL: <https://doi.org/10.1016%2Fj.tcr.2021.100413>.
- [5] *DAP – IT – Store norske leksikon*. 2021. URL: https://snl.no/DAP_-_IT (visited on 10/05/2023).
- [6] Yuxin Deng et al. ‘A Review of Robot Grinding and Polishing Force Control Mode’. In: *2022 IEEE International Conference on Mechatronics and Automation (ICMA)*. Guilin, Guangxi, China: IEEE Press, 2022, pp. 1413–1418. DOI: 10.1109/ICMA54519.2022.9855998. URL: <https://doi.org/10.1109/ICMA54519.2022.9855998>.
- [7] Matthias Dörr et al. ‘Prediction of Tool Forces in Manual Grinding Using Consumer-Grade Sensors and Machine Learning’. In: *Sensors* 21.21 (Oct. 2021), p. 7147. DOI: 10.3390/s21217147. URL: <https://doi.org/10.3390%2Fs21217147>.
- [8] Luca Guerrini et al. ‘A Survey on Multisensor Fusion and Consensus Filtering for Sensor Networks’. In: *Discrete Dynamics in Nature and Society* 2015 (2015), p. 683701. ISSN: 1026-0226. DOI: doi:10.1155/2015/683701. URL: <https://doi.org/10.1155/2015/683701>.
- [9] Hartvig Hartvigsen. *Verkstedhåndboka*. Gyldendal undervisning, 2006.
- [10] *How to mix C and C++, C++ FAQ*. 2023. URL: <https://isocpp.org/wiki/faq/mixing-c-and-cpp> (visited on 05/05/2023).
- [11] S. Husmann et al. ‘Model Predictive Force Control in Grinding based on a Lightweight Robot’. In: *IFAC-PapersOnLine* 52.13 (2019). 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019, pp. 1779–1784. DOI: <https://doi.org/10.1016/j.ifacol.2019.11.459>. URL: <https://www.sciencedirect.com/science/article/pii/S2405896319314405>.
- [12] INLEARC. *2. Industrial Robot Functionality and Coordinate Systems – Inlearc*. 2023. URL: <https://www.tthk.ee/inlearcs/industrial-robot-functionality-and-coordinate-systems/> (visited on 25/04/2023).
- [13] International Federation of Robotics. ‘Demystifying Collaborative Industrial Robots’. In: *International Federation of Robotics* (Dec. 2018). URL: https://web.archive.org/web/20190823143255/https://ifr.org/downloads/papers/IFR_Demystifying_Collaborative_Robots.pdf (visited on 23/08/2019).
- [14] Johnson. *Design of experiments based force modeling of the face grinding process*. 2008. URL: http://career.engin.umich.edu/wp-content/uploads/sites/51/2013/08/08_NAMRCJohnsonface_grinding_modeling.pdf.
- [15] Matthew Johnson. *PID Control Technology*. Ed. by Matthew A. Johnson and Hassan Moradi. London: Springer London, 2005. ISBN: 978-1-85233-702-5. DOI: doi:10.1007/1-84628-148-2.1. URL: https://doi.org/10.1007/1-84628-148-2_1.
- [16] Rudolph Emil Kalman. ‘A New Approach to Linear Filtering and Prediction Problems’. In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.
- [17] Seonguk Kim, Ivan Petrunin and Hee-Suk Shin. ‘A Review of Kalman Filter with Artificial Intelligence Techniques’. In: *2022 Integrated Communication, Navigation and Surveillance Conference (ICNS)*. Dulles, VA, USA: IEEE, 2022, pp. 1–12. DOI: doi:10.1109/ICNS54818.2022.9771520.

-
- [18] R.I. King and R.S. Hahn. *Handbook of Modern Grinding Technology*. Chapman and Hall Advanced Industrial Technology Series. Springer US, 2012. ISBN: 9781461319658. URL: <https://books.google.no/books?id=sJ8ACAAAQBAJ>.
- [19] B. Lundkvist and I. Øien. *Maskintegning*. Universitetsforlaget, 1993.
- [20] ILO Content Manager. *Grinding and Polishing*. 2011. URL: <https://www.iloencyclopaedia.org/part-xiii-12343/metal-processing-and-metal-working-industry/item/680-grinding-and-polishing> (visited on 27/04/2023).
- [21] *Manufacturing Milling*. 2023. URL: https://docs.sw.siemens.com/en-US/doc/209349590/PL20200605195244930.mfgmilling/contour_drive_overview (visited on 05/05/2023).
- [22] E. Meijering. ‘A Chronology of Interpolation: From Ancient Astronomy to Modern Signal and Image Processing’. In: *Proceedings of the IEEE* 90.3 (2002), pp. 319–342. DOI: doi:10.1109/5.993400. URL: <https://ieeexplore.ieee.org/document/993400>.
- [23] Mamad Mohamed. *Challenges and Benefits of Industry 4.0: An overview*. 2018. URL: http://www.ijom.com/article_2767_1ed3957e521af286722efb31b2772314.pdf (visited on 27/04/2023).
- [24] Jing Ni et al. ‘Establishment and Verification of the Cutting Grinding Force Model for the Disc Wheel Based on Piezoelectric Sensors’. In: *Sensors* 19.3 (Feb. 2019), p. 725. DOI: 10.3390/s19030725. URL: <https://doi.org/10.3390/s19030725>.
- [25] Jigneshkumar ParmarChetankumar. *Health risks associated with the grinding process*. 2020. URL: https://www.researchgate.net/publication/342246923_Health_risks_associated_with_the_grinding_process (visited on 27/04/2023).
- [26] *PID C++ implementation · GitHub*. 2019. URL: <https://gist.github.com/bradley219/5373998> (visited on 20/05/2023).
- [27] Stephen R. Schmitt. *Circle from 3 Points*. 2016. URL: https://web.archive.org/web/20161011113446/http://www.abecedarical.com/zenosamples/zs_circle3pts.html (visited on 25/04/2023).
- [28] *Real-Time Data Exchange (RTDE) Guide - 22229*. 2019. URL: <https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/> (visited on 20/05/2023).
- [29] *Red car jack isolated on white — Stock vector — Colourbox*. 2023. URL: <https://www.colourbox.com/vector/red-car-jack-isolated-on-white-vector-4387155> (visited on 15/05/2023).
- [30] *Robotic Machining*. 2023. URL: https://docs.sw.siemens.com/en-US/doc/209349590/PL20200605195244930.robotic_machining/xid1058668 (visited on 08/05/2023).
- [31] Universal Robots. *Universal Robots - What is a singularity?* 2022. URL: <https://www.universal-robots.com/articles/ur/application-installation/what-is-a-singularity/> (visited on 25/04/2023).
- [32] *SO File (What It Is How to Open One)*. 2022. URL: <https://www.lifewire.com/so-file-4150722> (visited on 20/05/2023).
- [33] *The URScript Programming Language for e-Series*. 2021. URL: https://s3-eu-west-1.amazonaws.com/ur-support-site/115824/scriptManual_SW5.11.pdf (visited on 01/05/2023).
- [34] Karl Ulrich. *The role of product architecture in the manufacturing firm*. 1995, pp. 419–440.
- [35] *Universal Robots RTDE C++ Interface — ur_rtdc1.5.6documentation*. 2023. URL: https://sdurobotics.gitlab.io/ur_rtdc/ (visited on 05/05/2023).
- [36] Rowe W. Brian. *Principles of Modern Grinding Technology - W. Brian Rowe - Google Bøker*. 2014. URL: https://books.google.no/books?hl=no&lr=&id=FR13AgAAQBAJ&oi=fnd&pg=PP1&dq=Grinding&ots=J-zhqE0Lv4&sig=UUZ1B19VEielb92SKgaGdHbnWac&redir_esc=y#v=onepage&q=Grinding&f=false (visited on 16/05/2023).
- [37] Kenneth Waldron and James Schmiedeler. ‘Kinematics’. In: *Springer Handbook of Robotics*. Springer Berlin Heidelberg, 2008, pp. 9–33. DOI: 10.1007/978-3-540-30301-5_2. URL: https://doi.org/10.1007/978-3-540-30301-5_2.
- [38] Hao Wang et al. ‘Singular Configuration Analysis and Singularity Avoidance with Application in an Intelligent Robotic Manipulator’. In: *Sensors* 22.3 (2022), p. 1239. DOI: doi:10.3390/s22031239.
-

-
- [39] Wikipedia contributors. *Linear Interpolation—Wikipedia, The Free Encyclopedia*. [Online; accessed 27-April-2023]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Linear_interpolation&oldid=1105095045.
- [40] Xiaohui Xie and Lining Sun. ‘Force control based robotic grinding system and application’. In: *2016 12th World Congress on Intelligent Control and Automation (WCICA)*. 2016, pp. 2552–2555. DOI: 10.1109/WCICA.2016.7578828.
- [41] P. Zarchan and H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. Progress in astronautics and aeronautics. American Institute of Aeronautics and Astronautics, 2000. ISBN: 9781563472541. URL: <https://books.google.no/books?id=AQxRAAAAMAAJ>.
- [42] Zhang Zhiyong et al. ‘Picking Robot Arm Trajectory Planning Method’. In: *Sensors and Transducers* 162 (Jan. 2014), pp. 11–20.

Appendix

A System code (C++)

See "vedlegg.zip" under folder "Qt_Grinder_Widget/Qt_Grinder_Widget"

B Python and calculations code

For circle center code see "vedlegg.zip" under folder "Python/" For code on graphing and calculations see "vedlegg.zip" under folder "Python/Data analysis"

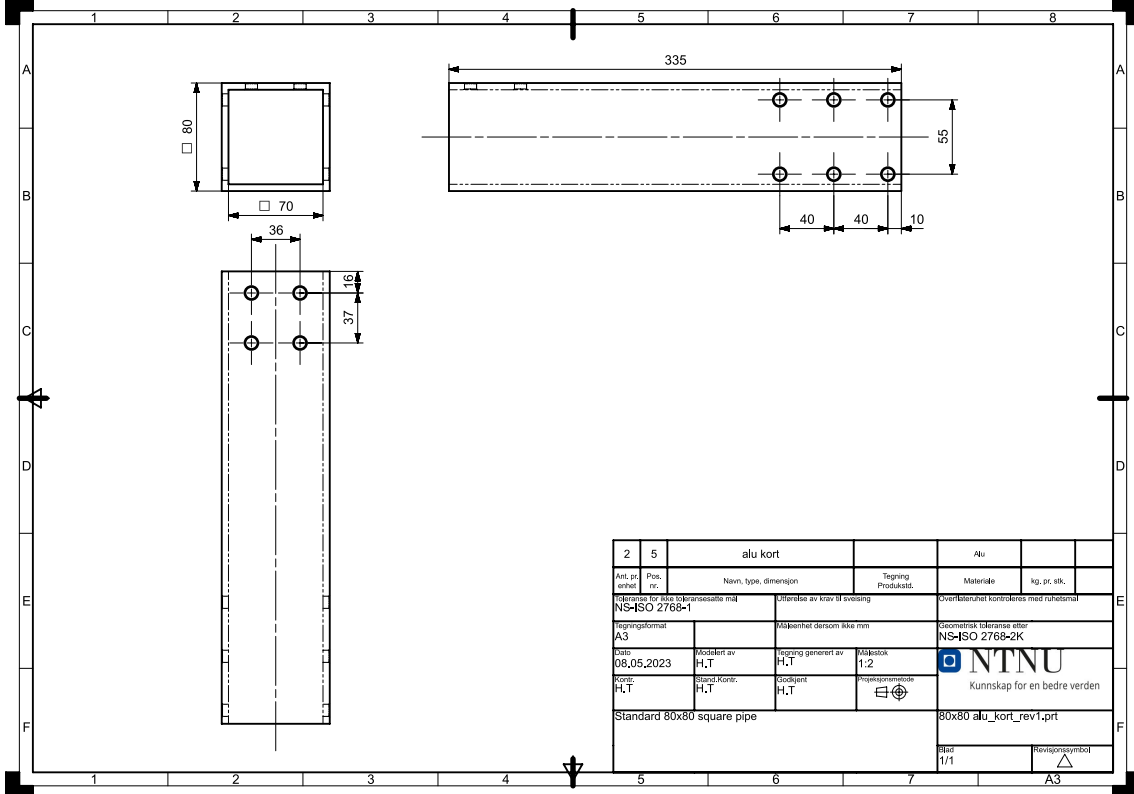
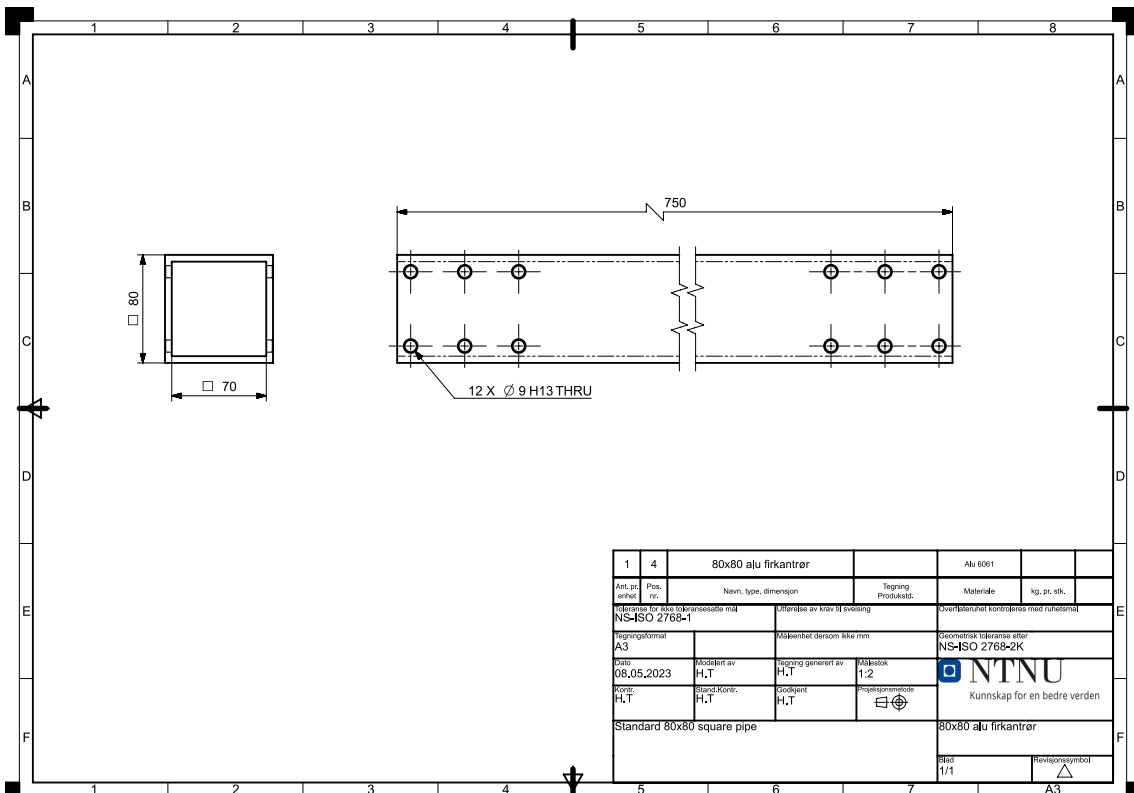
C Pre-project rapport (Forprosjektrapport)

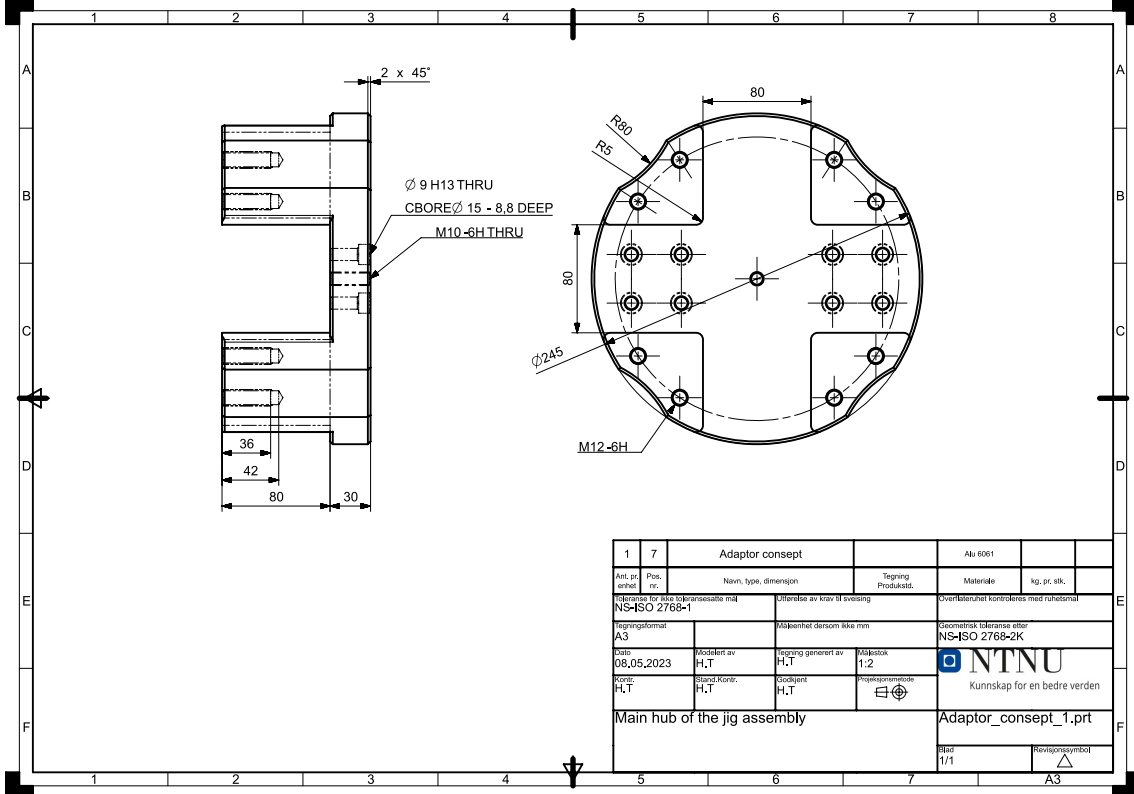
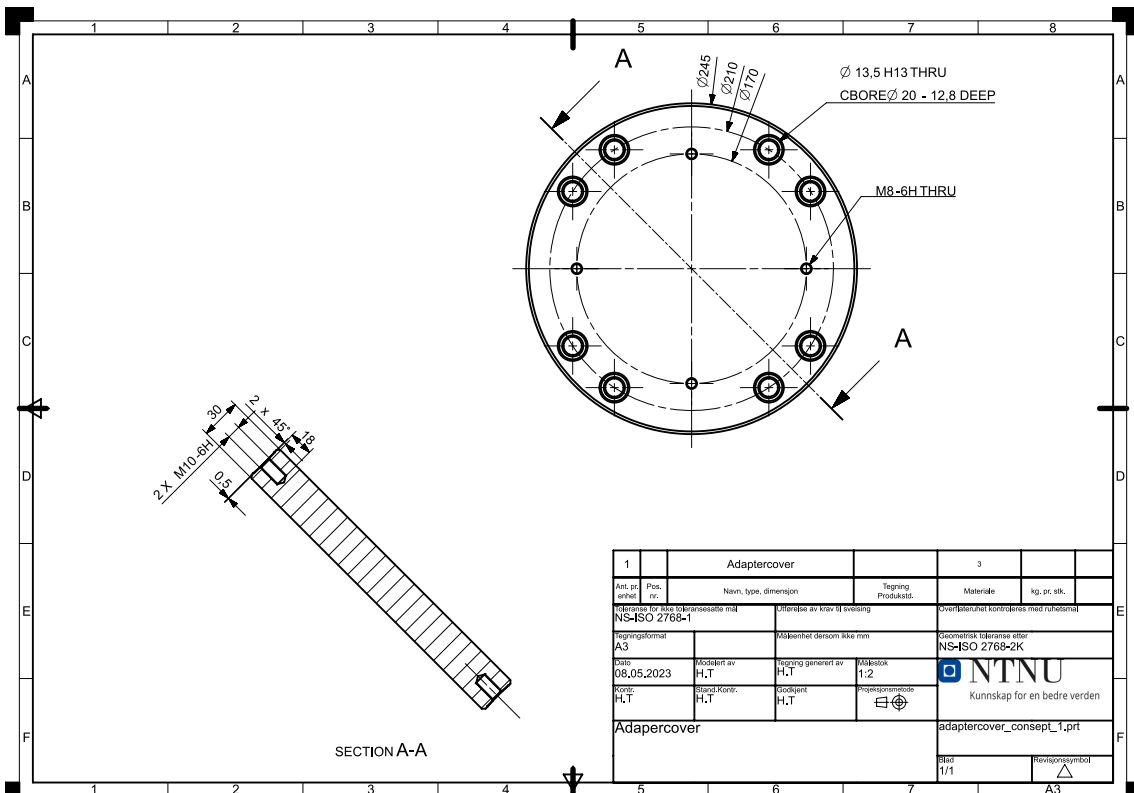
See "vedlegg.zip" under folder "Forprosjektrapport/"

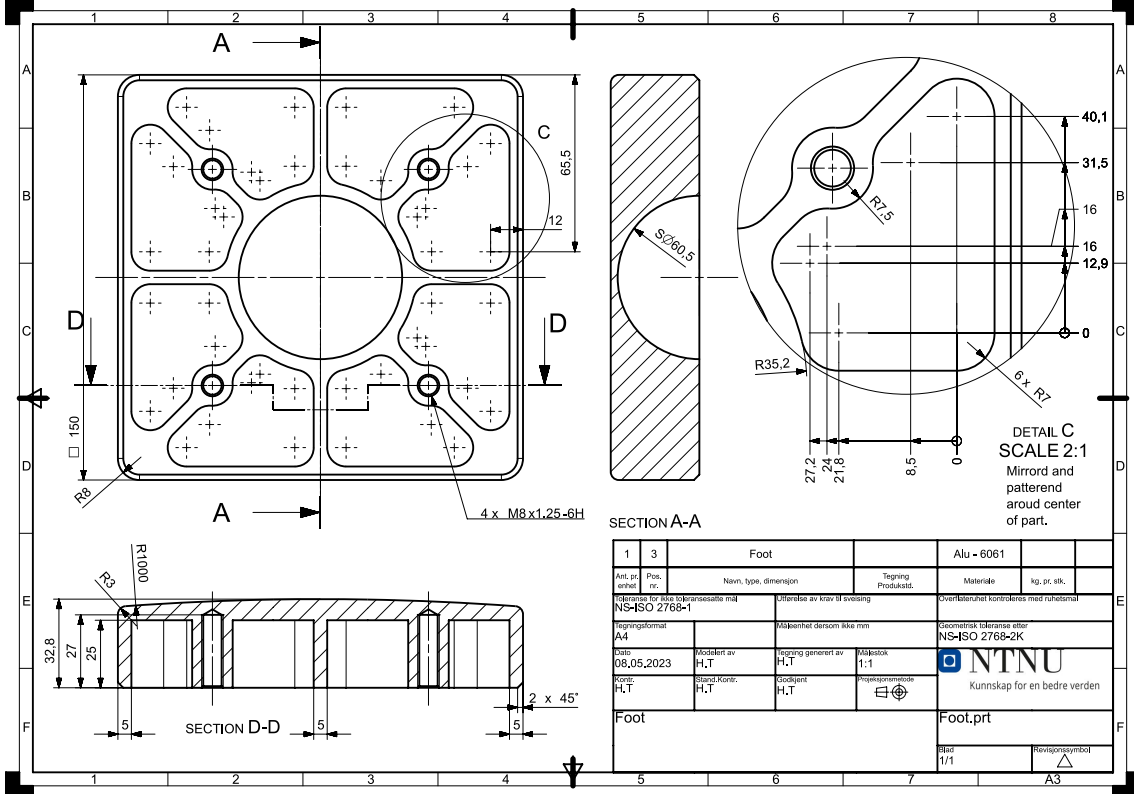
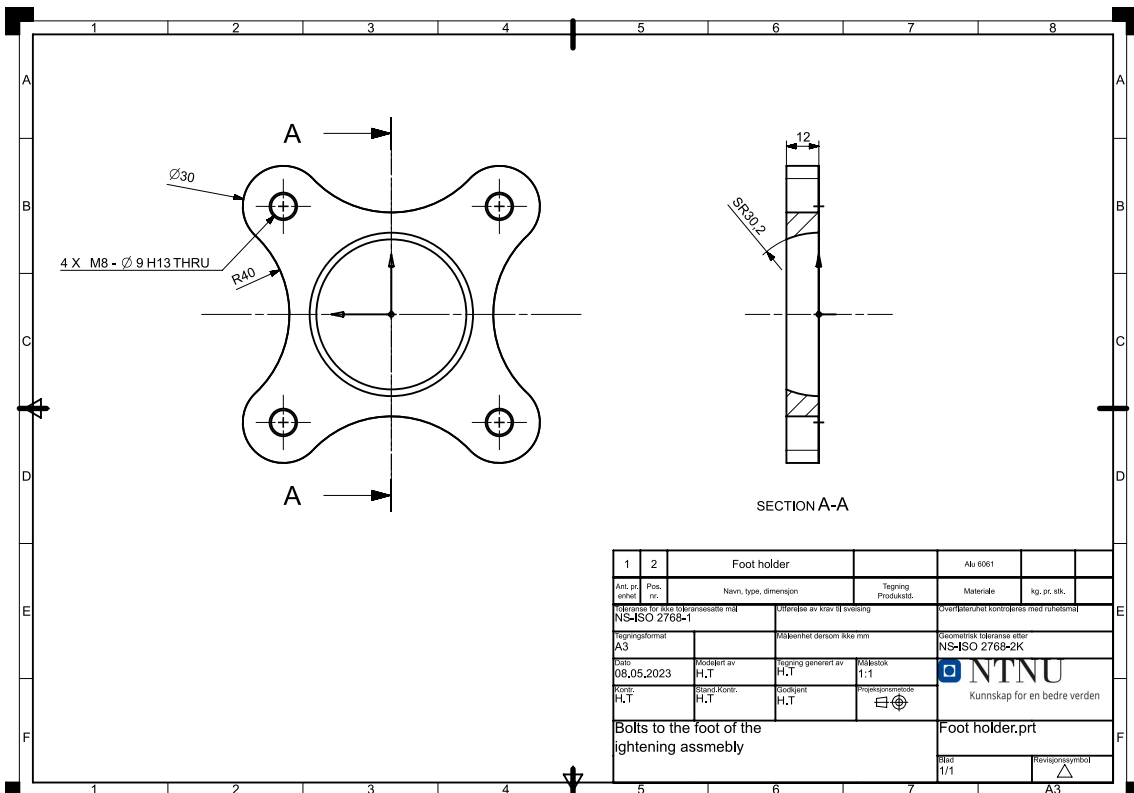
D Video

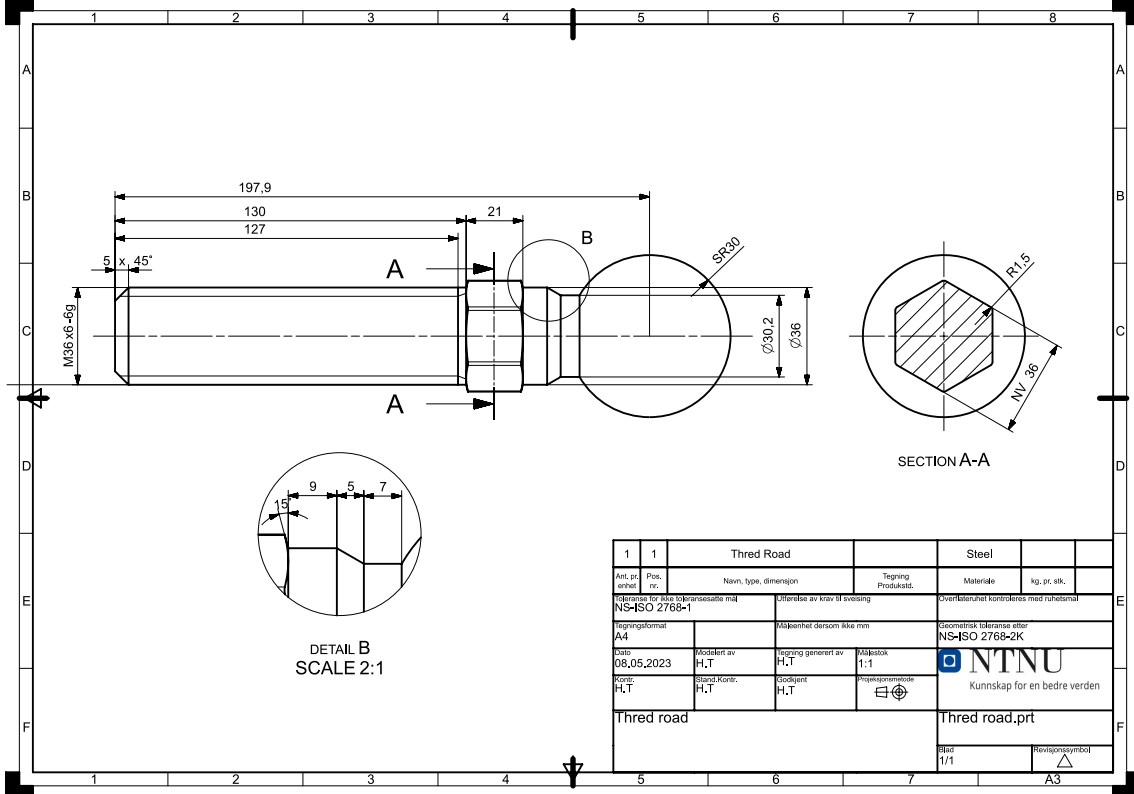
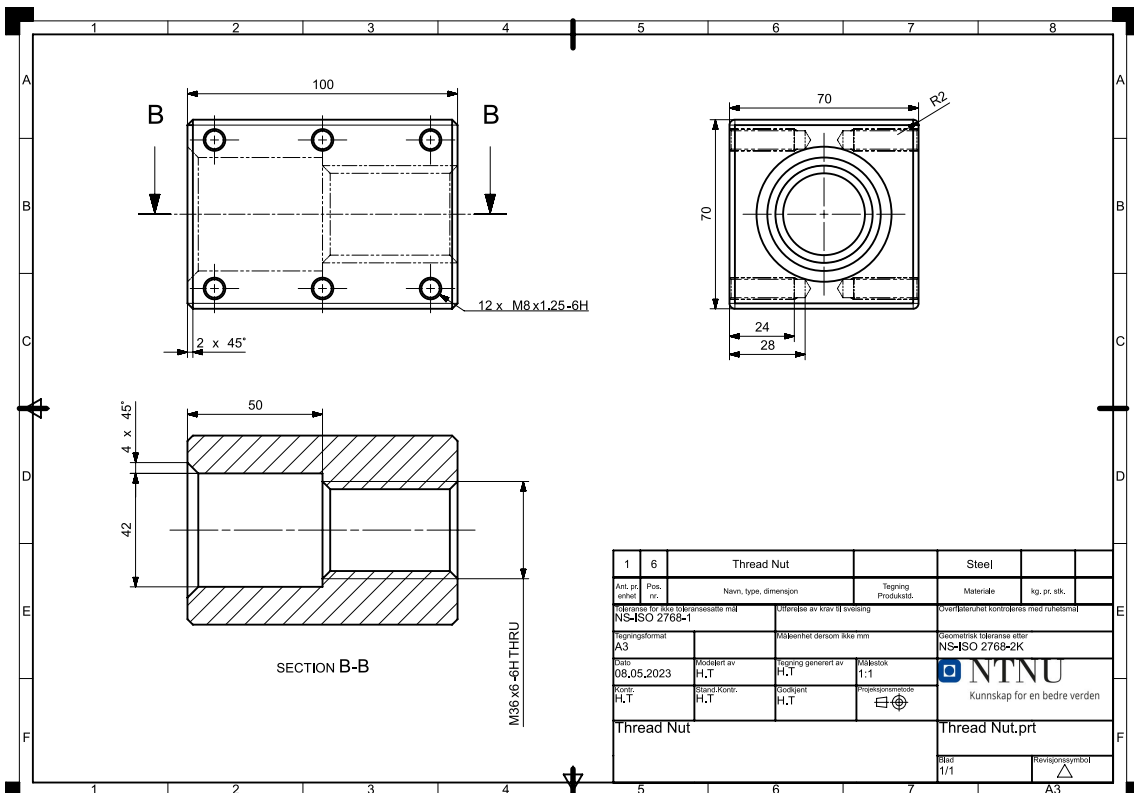
See "vedlegg.zip" file "video"

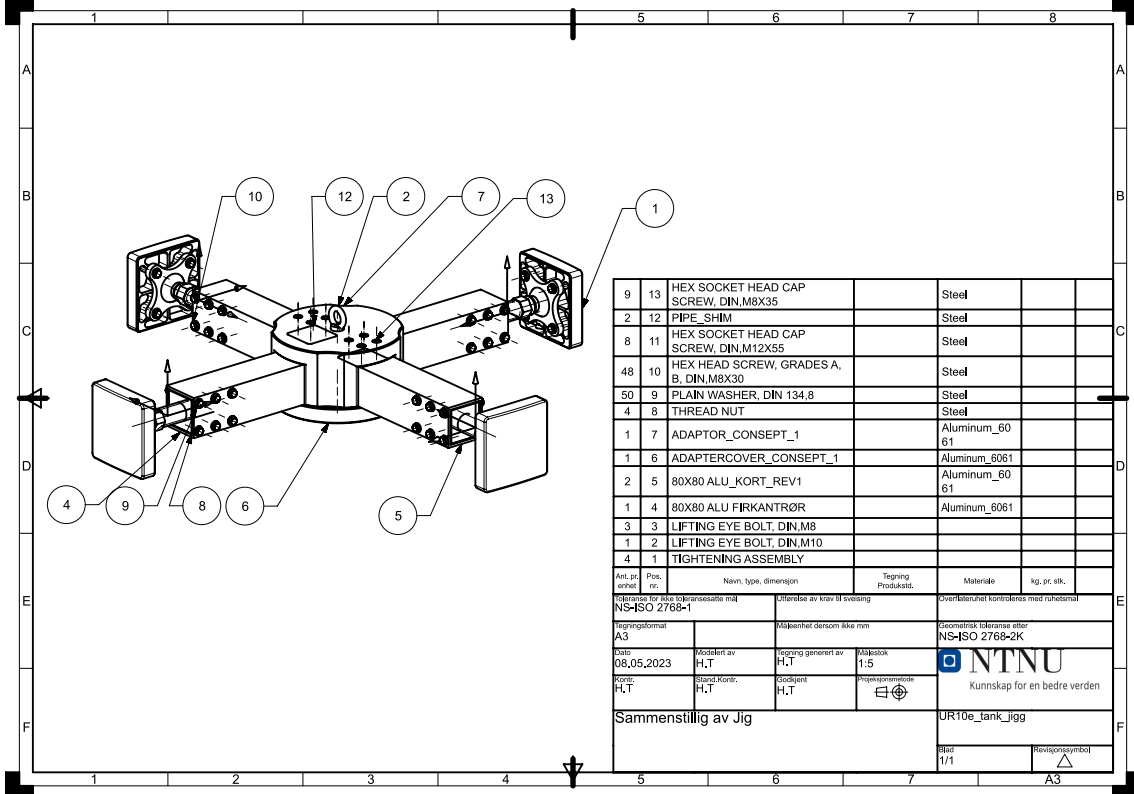
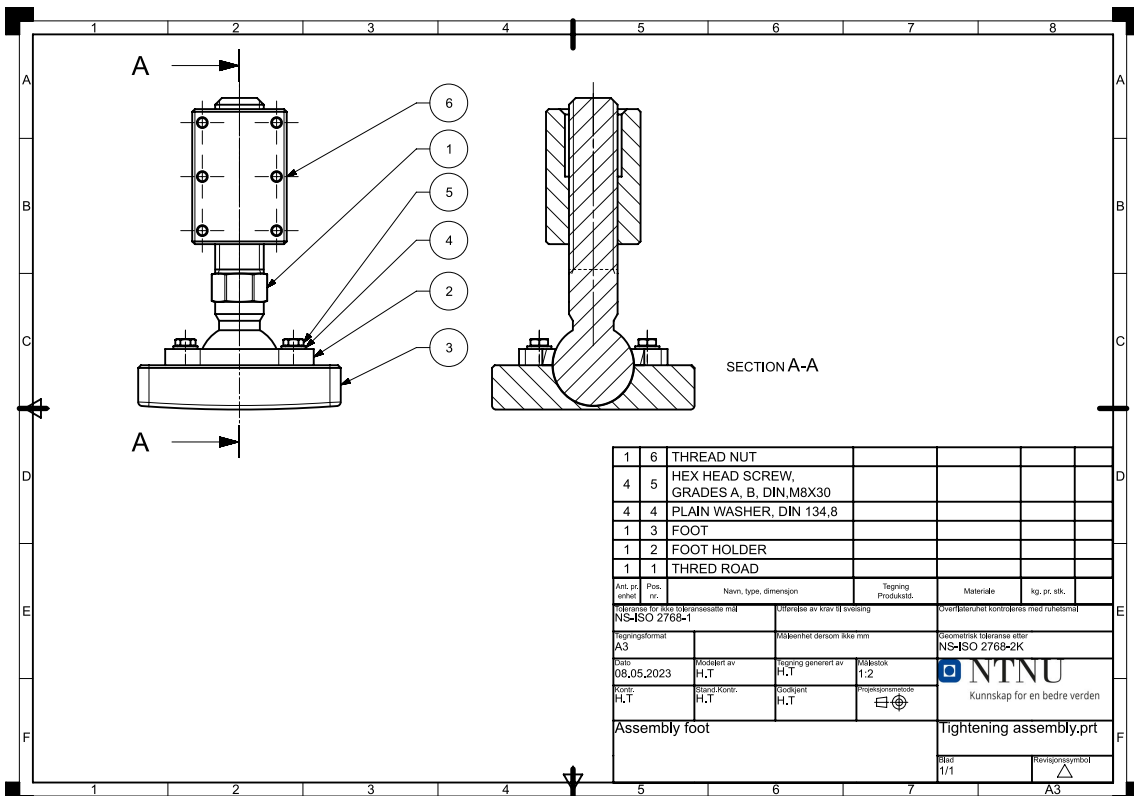
E Machine drawings













 **NTNU**

Norwegian University of
Science and Technology