UiA University of Agder

Bachelor

# Designing, developing and testing an autonomous waffle making system

Making an easy to carry, semi-automatic waffle making system based on a robot manipulator guided by machine vision.

ALEKSANDER WATSON
ANDREAS CALEB THORSEN
HÅVARD KARLSEN SOLHEIM

SUPERVISORS
Morten Hallquist Rudolfsen
Ilya Tyapin

**University of Agder, 2025**
Faculty of Engineering and Science
Department of Engineering Sciences

# Acknowledgments

We would like to thank the following people for their contributions to the project:

First, we would like to thank the employees at UiA that supported us:

- **Morten Rudolfsen & Ilya Tyapin** for their guidance, encouragement and concept inspiration. Special thanks must also be given to Morten for suggesting the project, and in doing so allowing us to have the dream project of spending half a year dedicated to making waffles.
- **Christian Auby** and the team running the UiA arcade. By hosting the StempaniaX machine, they enabled us to clear our mind between work sessions while getting some exercise in the process.
- The technicians at the mechatronics lab, **Jan Cristian "JC" Strandene & Harald Sauvik**. Their friendly presence and help with brainstorming has helped us greatly. And last but not least for accepting the ridiculous task of machining a waffle iron.

Next, we would like to thank our friends and companions who helped us get through these trying times:

- **Erlend Tregde** for giving his encouragement, support and ideas while working on the project.
- **Hilde C. Løvland** for keeping us in line and ensuring we actually get work done.
- **Alexandra Mathisen** for her guidance and company during report writing and proofreading.
- **Sander Westøll** for contributing to our GitHub readme.

We would also like to thank **Gary**, our robot arm, for always sticking around and giving us a helping hand whenever we asked for it.

Last, but not least, we in the group would like to thank each other for being a good team in working together throughout the project.

# Abstract

This report presents a semi-automated waffle-making system, integrating control of a robot arm and machine vision to enable semi-automated food preparation. The system was developed at the University of Agder to demonstrate a simple mechatronics engineering system to advertise to future students. The system sought to automate stages of the waffle making process, including application of grease spray, ladle-based batter transfer from a bowl, opening of the waffle iron, and delivery of the waffle. A camera combined with ArUco markers provided environmental awareness and facilitated accurate object localization. Robotic motion planning and control were achieved using the ROS2 framework in conjunction with the MoveIt motion planning library. The mechanical design prioritized portability and right to repair principles, incorporating 3D printed components and a touchscreen HMI. Public demonstrations were conducted to assess system performance in realistic settings and to test user engagement. Other tests also identified challenges related to system robustness, positional accuracy, and collision avoidance. Possible future work should focus on enhancing the accuracy of the system, implementing interactive entertainment-oriented modes, and extending the system's adaptability to a broader range of tasks. The societal implications of domestic service robotics are discussed, with an emphasis on open-source development, engagement, and long-term maintainability.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The University of Agder is Norway's leading university for the study of mechatronics engineering. This study tackles problems in machine design, electronics, robotics, and more. To market this study to future students, it was decided to create a demonstration project showing the possibilities of what mechatronics can do. The demonstration project that was chosen to complete this objective was to create a waffle making robot. This was chosen due to the two-fold appeal of the project. Firstly, waffle making is a visually distinct task that people can relate to. Secondly, waffles play into a great strategy of attracting people to a stand, free food.

## 1.1 Background

Robot arms have been used for making waffles in the past. Some notable examples include the ROBI3 project in Macau[13] and the waffle making system made by students at the University of Agder in a previous bachelor project[2]. The core innovation made by this project is the use of machine vision to make the system adaptable to changes in the environment.

Waffles are made using this process:

1. Mix waffle batter in a bowl
2. Connect power to a waffle iron
3. Apply grease to the waffle iron
4. Pour the batter
5. Close the iron
6. Wait until the waffle is cooked
7. When finished, take out the waffle and serve it
8. Eat and enjoy!

The creation of waffle batter and automatic consumption of waffles is considered out of scope for this project. The project therefore focused on automating points 3-7 of the above process, leaving the rest of the job to humans. Due to the partial completion of the waffle process, the system is considered semi-automatic instead of fully automatic. Appendix C contains information on the use of AI in this project.

## 1.2 Report Outline

The rest of this report begins by outlining the specifications of the intended behavior of the waffle making system. In Chapter 2 the technologies the project is built upon are presented. Chapter 3 documents the design of the system as it was built. The results of system testing are presented

and discussed in Chapter 4. Here, the societal impact of the waffle making robot is also discussed. Suggestions for future research and development is given in Chapter 5 before concluding the report in Chapter 6.

## 1.3 Design Specifications

The research question to be answered is:

**How can machine vision be used to assist robotic arms in domestic tasks?**

The robot should be able to make waffles with minimal human intervention. This includes opening and closing the iron, lubricating the iron, applying batter, and serving the finished waffle. While waffles are cooking, the robot has no jobs to do. During this idle time, the manipulator should perform unrelated idle activities to draw attention to itself. Entertaining games should therefore be developed to achieve this.

The robot should be able to move in either of two ways: manual mode and automatic mode. In manual mode, the robot should be able to move to a set of predetermined positions to execute the process of creating waffles without human intervention. By keeping movements predetermined, a reliable mechanism for making waffles in a controlled environment is provided. In automatic mode, the robot should be able to adjust its movements to account for objects being moved around at runtime.

To enable the robot to interact intelligently with its surroundings, a computer vision system should be implemented that provides relevant information about the environment. The purpose of the computer vision system is to allow the robot to perceive the presence and position of objects within its workspace, such that it can adapt its movements. This vision system forms the foundation for recognizing and interpreting spatial information that is essential for decision-making and movement.

A touchscreen display should be used to interact with the system. The touchscreen should provide an interface for ordering new waffles and provide feedback to the user.

The system must be portable and food safe. To market efficiently, the system should be able to be transported between locations easily. The components should be easy to mount and dismount, and compact such that the whole system can fit in the small space offered by a stand environment. To ensure food safety, it should be easy to clean.

# Chapter 2

# State of the art

The system presented in this report was not built from scratch. Many core advances in technology have been made that were prerequisite to the idea of a semi-automatic waffle making robotic system. In this chapter, the primary technological foundations that made this project possible are outlined.

## 2.1 Hardware

The traditional approach to creating waffles relies upon one piece of technology, namely the waffle iron. To perform actions otherwise done by a human, a robotic arm was used as the primary actuator. To react to a changing environment, a sensor was needed. For this purpose, a camera was used. The control of these components was performed using a microcomputer. These three components together form a system following the sense-think-act (camera-microcomputer-robot) model of robotics, as shown in Figure 2.1.

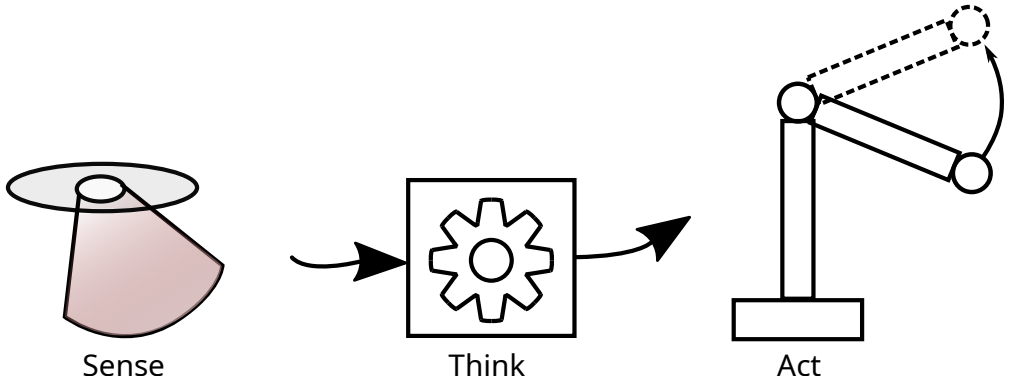



Figure 2.1: The sense-think-act model of robotics

Robotic arms are commonplace in industrial settings. Manufacturing processes often require repeated precise movements, which robot arms excel at. Robot arms currently exist in many forms and sizes, ranging from small arms for simple pick-and-place applications to large-scale industrial robots. It is common for robotic arms to have six movable joints, each moving about a different axis. Each axis of rotation corresponds to movement in a different spatial dimension, commonly referred to as a degree of freedom. By having an arm with six degrees of freedom, movement in all six spatial dimensions (three axes of translation and three axes of rotation) is possible.

Microcomputers are pivotal in modern robotics because they are compact, cost-effective, and efficient computing units. By integrating a processor core, memory, and programmable I/O on a single chip, microcomputers can manage real-time tasks such as sensor data acquisition, actuator control, and algorithm execution within embedded systems. Their deterministic behavior and interrupt handling capabilities make them ideal for applications demanding responsiveness and autonomy.

Intel RealSense depth cameras are part of Intel's machine vision-oriented product line[9]. These cameras are factory calibrated for high-accuracy depth sensing and image processing, making them suitable for machine vision. RealSense cameras offer features such as stereo depth perception, RGB imaging, and robust calibration, which enable precise environmental perception. A RealSense camera is shown in Figure 2.2. An included API, Pyrealsense2, can be used to fetch images and depth data to use in software.

Figure 2.2: Intel Realsense D455 Depth Camera

## 2.2 3D-printing

3D-printing is a form of additive manufacturing, consisting of a 3D-printer and some type of filament. The most common type of 3D-printing is fused deposition manufacturing (FDM). A 3D-printer makes a part based on a three-dimensional model, which is first put in a software named a slicer. The slicer generates a tool path for the tool head, as shown in Figure 2.3. The tool path forms the model, which then makes the part. The part is constructed by adding thin layers of filament on top of each other. These layers build up the part.

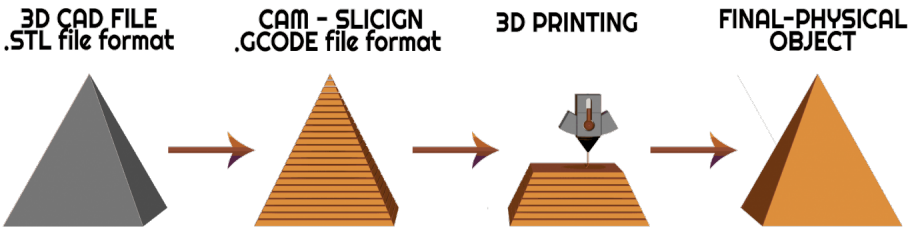



Figure 2.3: How 3D-printing works (picture from my3dconcepts.com[16])

FDM 3D-printers use plastic filament. FDM filaments consist of different types of plastic, such as PLA, ABS, PETG, TPU, and many more. Each material has its own properties, such as hardness, heat resistance, and impact resistance.

PLA+ is one of the most used materials within 3D-printing. PLA+ is a material used for its ease of printing and low cost. The material excels in rapid prototyping, but has low resistance to heat[4].

Onyx is a material with high strength and high thermal resistance[15]. The material is unique to Markforged and is therefore quite expensive. The material is also quite difficult to print with, so a special printer is needed. The heat deflection temperature of the material is 145°C, which is significantly higher than more commonly used materials. Onyx is less attractive due to its high price compared to more commonly used filaments.

ABS is a material that is harder to print than PLA+ because the material shrinks during printing. In return, it has a higher temperature resistance and is stronger. This makes it better for prototyping, where heat might be an issue. The heat deflection temperature of the material is 86°C[32]. ABS is a relatively cheap material compared to materials like ONYX, but slightly more expensive than PLA+.

TPU 95A is a flexible material. It excels in impact resistance and layer adhesion, but is difficult to print with compared to PLA+. TPU is well suited for applications that are subjected to frequent impacts.

Food safety is based on multiple factors, such as which materials are in contact with the food. Food safe materials should not be toxic or leave harmful substances. The waffle iron is inherently food safe, as it was designed specifically for making waffles. 3D-printed parts are generally not food safe, even if the material is food safe. This is due to the way 3D-printing works. The layers create crevices that food can enter and makes it hard to clean. Due to this, no parts that are 3D-printed should be in direct contact with food.

## 2.3 Software

Multiple innovations have been made that simplify the interactions between hardware and software. The following sections outline some of the foundational components of software that make arbitrary movements of a robotic arm and machine vision possible:

The robot's backend is powered by ROS2. According to the Open Robotics Foundation, maintainers of ROS2: *"The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications."*[21] Examples of the aforementioned ROS-based tools include the motion planning framework MoveIt, and digital twin display tool Rviz.

A core function provided by ROS2 is to accommodate communication between devices. At the heart of this communication is the standard ROS2 unit: The node. When working with ROS2, devices are standardized to be "Nodes". Any node can publish a custom "message" to a "topic". These topics can then be "subscribed" to by other nodes who desire their information. A typical ROS-based system has multiple types of node behavior. Publisher nodes (typically representing sensors) publish information, subscriber nodes (typically representing actuators) act on the given information. A logic solver in the middle acts both as a subscriber and a publisher. An illustration of such a system is shown in Figure 2.4.

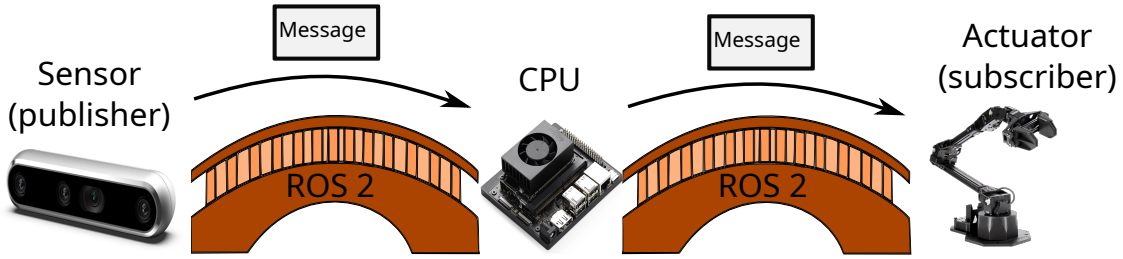



Figure 2.4: Illustration of ROS communication

The Interbotix Python-ROS Interface[23] made by Trossen Robotics was used as the primary software driver for the robot. Trossen robotics provides a software package that controls the underlying ROS system driving the manipulator. This includes pre-made configuration files, motor drivers, and a software library that allows a user to get started with the robot. The aforementioned library was used alongside MoveIt as the primary interface to control the robot using Python.

MoveIt is an open source motion planning framework built on top of ROS2[31]. A core functionality of MoveIt is its ability to plan a collision-free path between given start and end states, provided it has been configured with information about the robot. The Interbotix software package includes a pre-configured launch configuration that tells MoveIt everything it needs to control the robot. By using the Interbotix configuration, a MoveIt-interface ROS2 node named `move_group` is created. This interface can then be accessed by sending messages to the `move_group` node.

The Interbotix-ROS interface and MoveIt use separate motion planning algorithms. The Interbotix-ROS interface uses a collision-unaware motion planning framework based on the Newton-Rhapson

root finding algorithm[10]. It works by selecting a random initial guess of joint states and iteratively moving each joint such that the result is closer to the target pose. MoveIt, on the other hand, is not inherently bound to a single planner. Instead, it uses the Open Motion Planning Library (OMPL), which contains a multitude of algorithms to choose between. The OMPL planning algorithm chosen by Interbotix is the RRT-connect algorithm[11, line 56]. Rapidly-exploring Random Trees (RRTs) are a family of pathfinding algorithms that use search trees to explore space in search of the shortest path. The RRT-connect algorithm uses two of these trees, extending out from both the start pose and the target pose. These two RRTs connect to each other in the middle and combine to form a complete path[12]. The field of motion planning has many options to explore, but these two algorithms were the only core planners used by the Interbotix configurations for collision-agnostic and collision-avoidant paths respectively.

For the machine vision software, Open Source Computer Vision (OpenCV) was used. It is a free library used in computer vision and machine learning projects. It includes many useful algorithms for things like detecting faces, tracking objects, and processing images. OpenCV is available under the Apache 2.0 license, which means it can be used in both research and commercial settings. It works with several programming languages, for example C++, Python, Java, and MATLAB[20].

# Chapter 3

# Design

This chapter details the design of the waffle making system. Firstly, a description of the core hardware components used to create the product is given in Section 3.1. Section 3.2 explains how the rest of the system is built mechanically. Here, the layout is presented as well as the numerous adaptations needed to make the task of making waffles easier to automate. Once the physical design has been established, the software logic that makes the machinery move is detailed. The machine vision system used to detect changes in the environment is described in Section 3.3.2. Building upon that technology, this chapter continues with Section 3.3.1. This section describes the logic, software components, and control flow used to decide how the robot executes movements. The code used in the project was made available on GitHub[6].

## 3.1 Hardware

The design was based around three pieces of hardware, each representing one of the three core concepts of the sense-think-act-model of robotics. The sense aspect was represented by a camera that oversees the workspace. A touchscreen was also used to provide an HMI to the users. In addition, an emergency stop button was placed on the HMI in case anything were to go wrong. The ”act” aspect was performed by the robot arm. The thinking was performed by a Jetson Orin Nano Super developer kit microcomputer[18]. The microcomputer ran the HMI, and the robotics movement logic as well as camera software.

For this project, a touchscreen interface was incorporated to enhance system interactivity. The touchscreen served a dual purpose: It displayed real-time process information, and it enabled users to interact with robot functions directly. This combination offered intuitive control and immediate feedback. The display that was chosen was the Ingcool 7IP-CAPLCD[7], a 7-inch capacitive LCD touchscreen. Figure 3.1 shows the touchscreen with the HMI screen.


Figure 3.1: Ingcool 7IP-CAPLCD 7-inch touchscreen display

The software for this project was hosted on a Jetson Orin Nano Super Developer Kit[18], shown in Figure 3.2. This compact microcomputer offers a robust platform suitable for embedded applications in machine vision and robotics. It served as the computing platform for operating the robot arm and the camera.

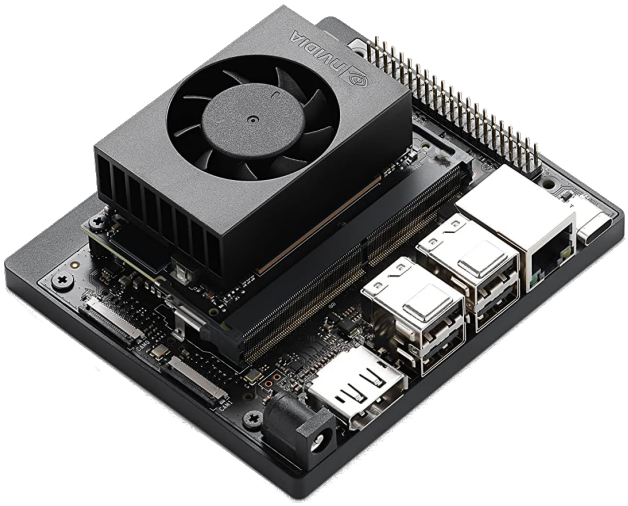


Figure 3.2: Jetson Orin Nano Super Developer Kit

The project initially employed the Intel RealSense L515 depth camera, which was subsequently replaced by the Intel RealSense D435i, because of software incompatibility. In the final stage, the system was upgraded to the Intel RealSense D455. This model was selected due to its extended horizontal field of view, offering increased spatial awareness compared to earlier models in the RealSense series. These features made the D455 particularly well suited for the workspace it was used in. An illustration of the D455 depth camera is shown in Figure 2.2. The previous camera, the D435i, would also be used as a secondary camera.

The ViperX 300S arm[24] sold by Trossen robotics was used as a primary actuator. The ViperX 300S arm has six degrees of freedom, giving it high maneuverability in both translation and rotation. With a lifting capacity of 750g [24], the arm was capable of fully opening the waffle iron without

external assistance. The arm also has a range of 75 cm[24], giving it access to the entire workspace that was used. This arm was a highly capable tool for the job.

A 12V 25A power supply was incorporated to power the manipulator and Jetson. All further components needing power would draw power from this power supply. This power supply was chosen due to the specification of the manipulator needing 10A during normal operation, but can increase to 35A peak if all servos were moved at max speed according to their datasheet[29][28]. This is unlikely to occur, due to this being an unneeded speed, but it is better to have overhead.

## 3.2 Mechanical Design

This section overviews the mechanical concepts and design for this project. This includes the mounting methods, design reasons and assembly guides. All technical drawings for the 3D-printed parts can be found in Appendix B.

### 3.2.1 Layout

The layout of the workspace was designed based on usage aspects, safety considerations, and reduction in spillage. The layout is shown in Figure 3.3. The base plate had to be light enough to transport while still being strong enough to hold everything in place. The plate also needed to be safe for food. The baseplate was made out of a plywood sheet with a wax cloth on top to make cleaning easier. All electrical components that were prone to fail if spilled on were placed as far away from the batter as possible. The batter and iron were placed close to each other to reduce the chance of spillage. By having the manipulator and waffle iron on the side that the operator would stand, the observer's safety was increased. All the components on the table were made such that they could be taken off during transport. By doing this, they became easier to handle and faced a reduced risk of damage during transport. All electronic parts were put in a housing and the manipulator was mounted on the lid. The waffle iron was mounted in front of the manipulator, and the batter was placed beside the iron. The waffle would be delivered in the free space next to the manipulator; on the side that the observers would be standing.

Figure 3.3: Layout of the table

### 3.2.2 Manipulator Modifications

The VX300S manipulator was well suited for general tasks that a manipulator would do, such as picking up light cubes. To increase its reliability, some adaptations were made to the manipulator.

When the manipulator goes to sleep from its idle position, it rests on the bottom servomotor that controls the shoulder joint. In the `Sleep` position, the arm is suspended a couple of centimeters

above the servo situated at the shoulder joint. Therefore, the gripper briefly experiences free-fall when it powers down before landing on the motor. This fall causes wear on the parts of the robot and may cause the paint to chip. This is why the pillow has been incorporated. The pillow absorbs the impact of the fall. The pillow was printed with TPU 95A, to make the pillow soft and resist impact without breaking[33]. It had a raised portion shown in Figure 3.4, this part absorbed the impact. The side pieces had holes to mount on the robot with M2.5 screws.

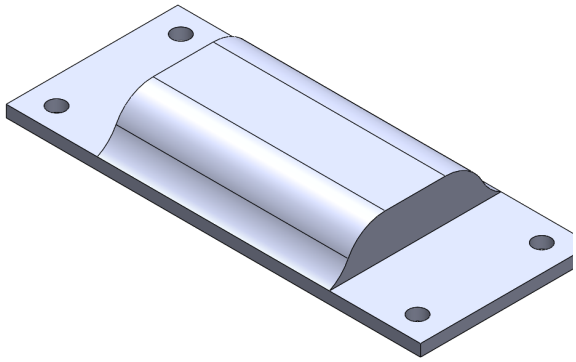


Figure 3.4: Pillow for manipulator

Each item that the manipulator would need to lift needed a reliable way to be picked up. The original grippers offered great versatility in what they could pick up, but a more reliable option was needed. This need inspired the invention of the interface cube. The interface cube's job was to give a more consistent interface for the robot to pick up. Each cube was designed for the specific requirements of their parent part, but all the cubes share 2 properties:

The total height of the cube needed to be at least 30mm and the triangular sides were 90 degrees at the outermost point. The width of the alignment triangles was 15mm. The reason for these sides was to keep the cubes in one set direction at all times relative to the grippers. The interface cube is illustrated in Figure 3.5. Due to the grippers having a minimum separation distance in their closed state, there needed to be a 35mm buffer so that the alignment cubes could properly align within the grippers. All further developments on the interface cubes that are documented aim to further enhance the interface cube's ability to adapt to the gripper, based on each cube's specific task.

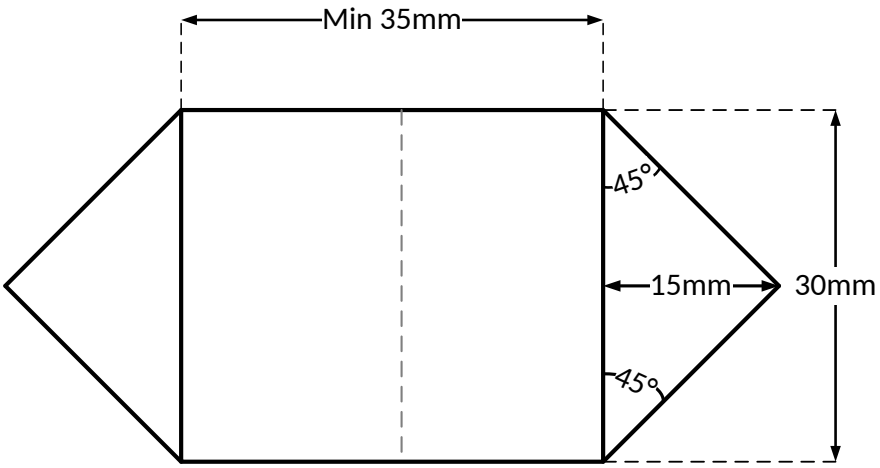
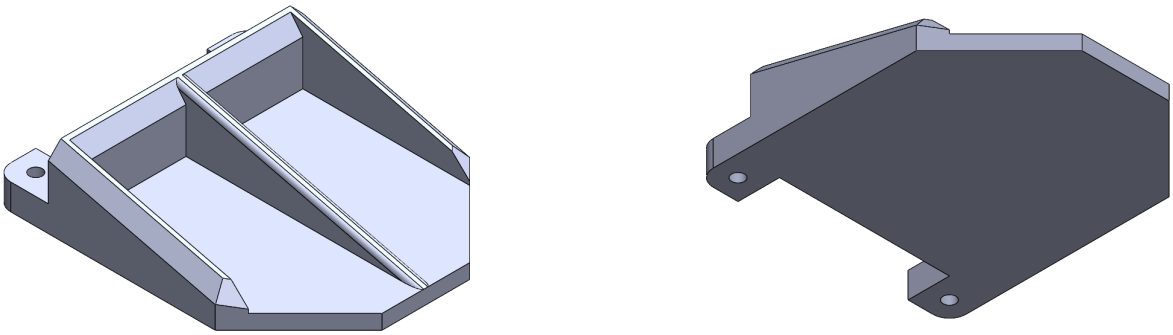



Figure 3.5: Baseline for the interface cubes

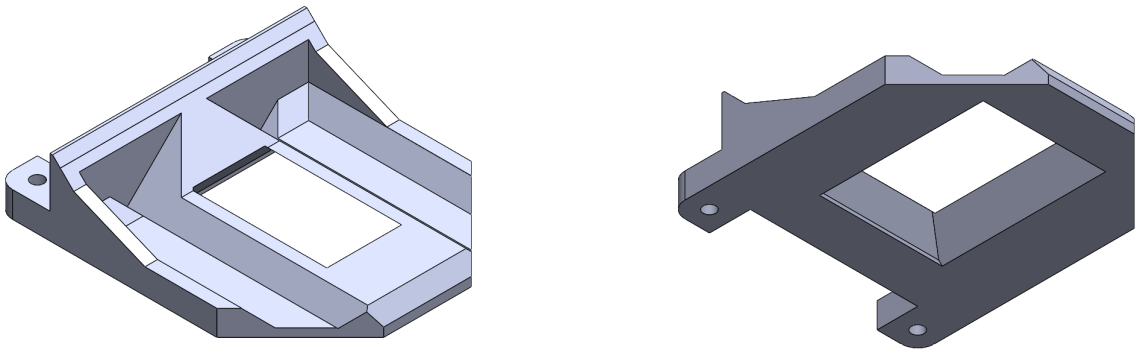
The grippers were redesigned to fit with the interface cubes. The original design, shown in Figure 3.6, was used as inspiration. The old grippers were designed to handle the power of the motors,

and were therefore known to be strong enough for use.



(a) Top view (b) Bottom view

Figure 3.6: The old grippers that came with the robot



(a) Top view (b) Bottom view

Figure 3.7: A new set of grippers, designed during the project

The new grippers, as shown in Figure 3.7, were printed in ABS, giving them resistance to the heat they were being subjected to when opening the iron. The design incorporated a slot that was designed to hold an interface cube, that would be mounted on the parts. The interface tools were adapted from the shape or surface of the target object and made the object easier to lift or move. The slot within the grippers, shown in red in Figure 3.8, was a negative of the interface cubes, so that they would fit together. When paired with the interface tool, the slot hindered movement in all directions. Due to the slanted design of the slot, interface cubes would slide into place when the manipulator gripped. This means that the manipulator did not need to be very accurate to pick up most of the items. The changed design was better for lifting heavy items relative to the loading capacity of the manipulator when using the interface cube.

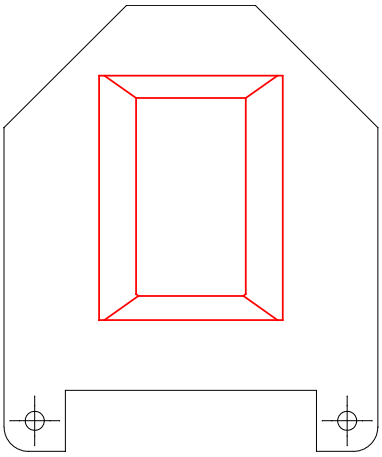


Figure 3.8: The grippers slot highlighted

A cutout was made from the top, which gives the new design a lower profile. This is illustrated in Figure 3.9, where the green illustrates the profile of the original grippers that came with the robot. A lower profile enabled more versatility for tight spaces. The cutout also made the tools visible for the operator to check if the interface cube was properly grabbed. By removing bracing material from the top, the redesigned grippers showed a loss of stiffness, but the part has not shown signs of failure during reasonable use.

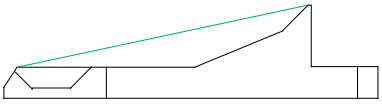


Figure 3.9: Grippers profile change

### 3.2.3 Hardware Housing

The electronics do not react well with fluids. To mitigate this, and to improve aesthetics, they were mounted in a housing to protect the parts.

#### First Iteration

Since the Jetson and the power supply generated excess heat, the thermal aspect of the design had to be considered. The first design had an open solution as seen in Figure 3.10 that allowed for free air flow. By mounting the robot on top, it became apparent that it would not handle the weight.

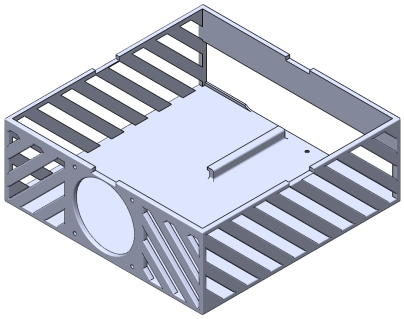


Figure 3.10: First housing iteration

Following the first housing iteration, the main goal for the lid was to have a well-ventilated design, as shown in Figure 3.11. The design had vents on all sides and allowed air to escape. The lid rested on top of the housing.

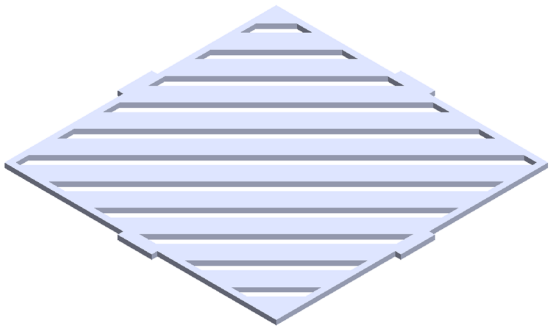


Figure 3.11: First lid iteration

### Second Iteration

A more sturdy box was designed as shown in Figure 3.12. In this design, the fan was solely responsible for the cooling. It pushed air over both the Jetson and the power supply. The exhaust air from the Jetson was pushed back out from the side vents. The power supply allowed air to pass through easily due to its well-ventilated design. The housing had two ventilation holes on the sides. The sides were almost entirely solid, with only cable holes and a hole for an 80mm by 80mm fan and exhaust vents. This housing design was stronger than the previous design, and the cooling proved to be adequate.

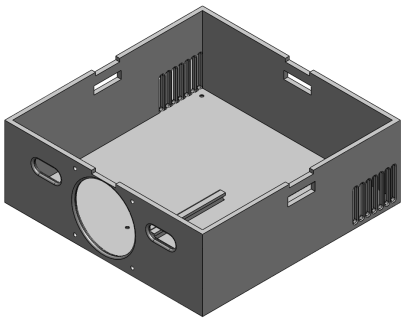


Figure 3.12: Second housing iteration

Once it was decided that the robot would be placed on top of the housing, more strength was needed. Therefore, a support structure around the whole lid was incorporated as shown in Figure 3.13. Slots for clips were also designed, and mounting holes for the robot arm was added. The second iteration was also made without ventilation to increase strength and repel batter. The lid held the robot, but broke during stress testing.

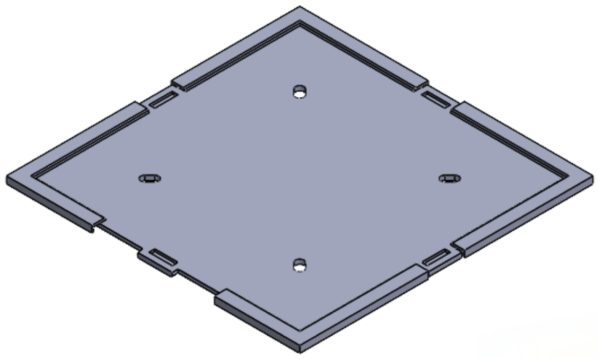
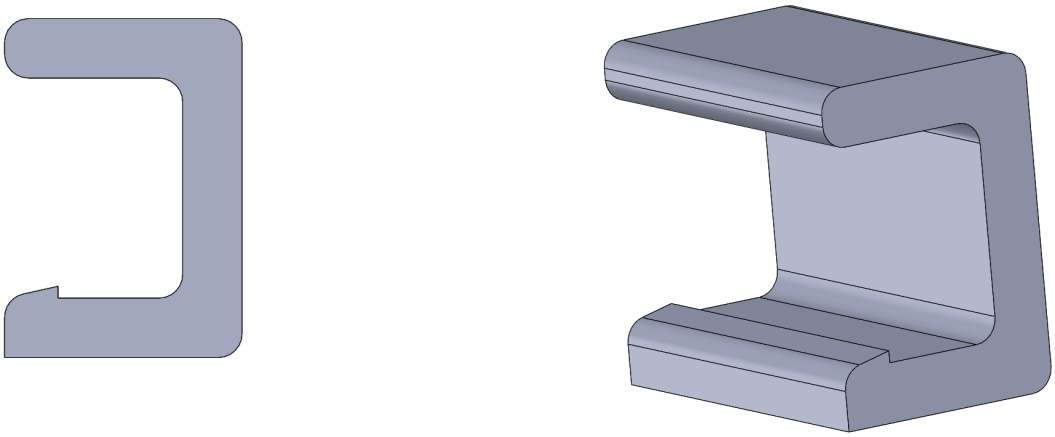


Figure 3.13: Top of second iteration lid

A way to lock the lid down to the housing was needed so that the arm would not cause the housing lid to fall off the housing. The first locking mechanism was a clip, as shown in Figure 3.14, made to hold the cover and the housing together. The clip had a lip on the bottom to lock the lid in place.



(a) Side view

(b) 3/4 view

Figure 3.14: Clip version 1

**Final Iteration**

A proper way to mount the housing was needed. The first plan was to incorporate holes to mount the housing to the plate with bolts. However, it was desirable to make it simple for the user to remove the housing for transport. This requirement also ruled out glue, which would be hard to remove and reapply. Using a lip on the edge of the housing that could slide in a holder was incorporated. This lip, combined with a slot that locked the housing in place provided a way for the operator to remove the housing when traveling. The lip allowed for mounting the housing to the table in a secure manner. The vents were redesigned to have slanted sides as shown in Figure 3.15, to reduce the risk for fluids to enter the housing in case of spillage. The locking mechanism for the lid was redesigned to make it easier to attach and remove the lid by using locking pins instead of clips. The notch on the top of the housing sides was repurposed from lock holder to support brace as the lid that was made for this version incorporated a brace on the underside of the lid.

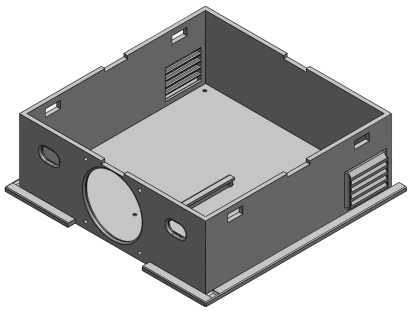


Figure 3.15: Housing final iteration

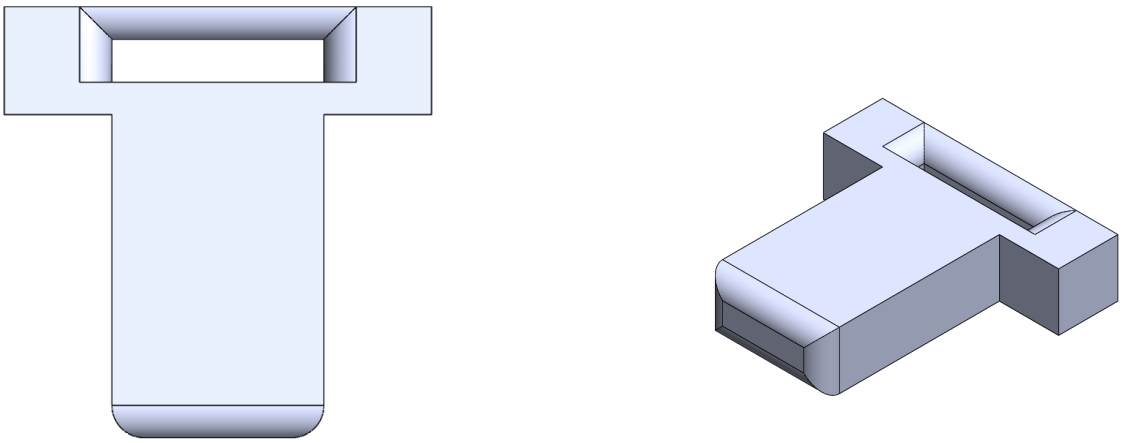
When redesigning the housing to final iteration, the choice of using pins instead of clips was made, and with that, the lid needed to be redesigned. Converting to locking pins helped with rigidity. This was due to the locking pins having lower tolerances. 4 locking pins were distributed evenly between the left and right sides of the lid as shown in Figure 3.16. A cutout was also made to improve airflow for the fan. The lips on the side of the lid were also extended to accommodate the cutouts for the locking pins.

(a) Bottom view

(b) Top view

Figure 3.16: Lid final iteration

When re-designing to final iteration, the clips got changed with locking pins. The locking pins are shown in Figure 3.17. The locking pins were made to be easier to insert and remove than the clips, and would also fill the holes completely to reduce air leakage. By reducing air leakage, the air from the fan would pass through all the components to cool them as intended. Only the head of the locking pin protruded from the holes. The head made it easy to pull the locking pins out if access to the internal components was needed.



(a) Top view (b) 3/4 view

Figure 3.17: Locking pin

[todo] insert pic of entire housing

### 3.2.4 Housing Features

The design of the final housing iteration was detailed. This section goes through keypoints of the design.

#### Structure

On the front of the housing, two oval holes were made for cables to pass through as shown in Figure 3.18. They were aligned with the output terminals of the power supply and Jetson. It had enough room for USB cables to pass through in case a cable change would be needed.

When the housing lid was closed and all wires were connected, there was a chance of the cables hitting the fan on the Jetson. A cable guide was made to counteract this. The cable guide made the cables exit the box straighter and eased strain on the connectors. The cable guide was glued to the wall and suspended the cables over the Jetson as shown in Figure 3.19.

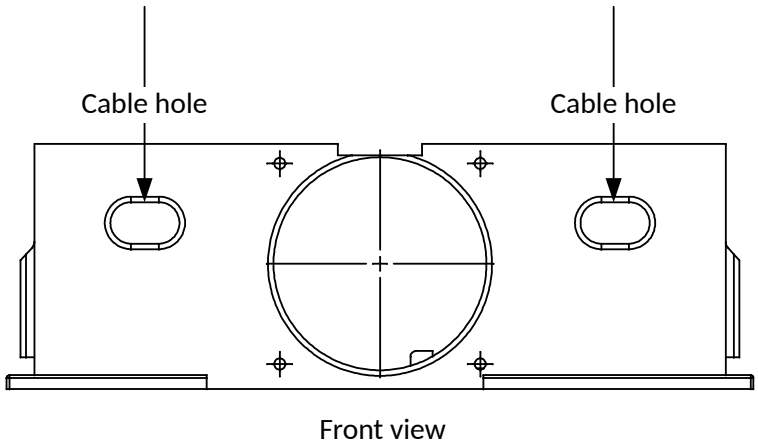



Figure 3.18: Cable holes

The manipulator may have caused the lid to flex while it was lifting heavy loads, so the lid was reinforced on the underside as shown in Figure 3.20. The reinforcement also made the manipulator

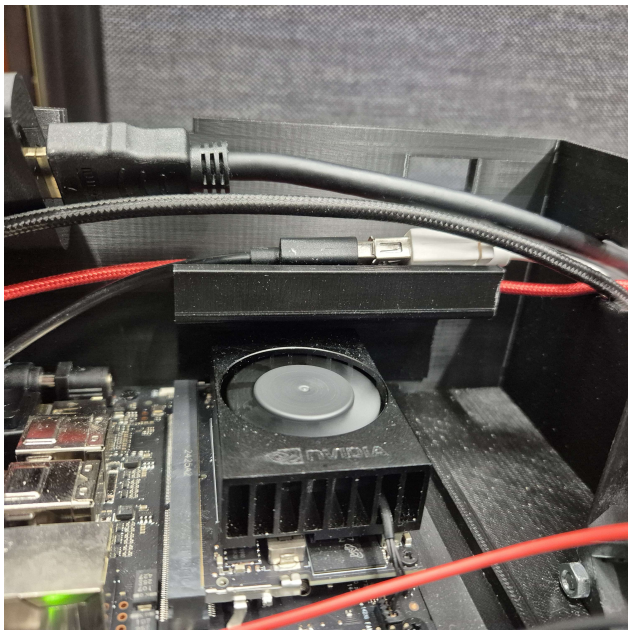


Figure 3.19: The cable guide in the interior of the housing

more accurate, because the base of the manipulator (lid of the housing) would not deflect under load.[todo specify] The slots in the lid reduced backlash. This was done by using slots in the lid which then made a mechanical hindrance between the lid and the housing.

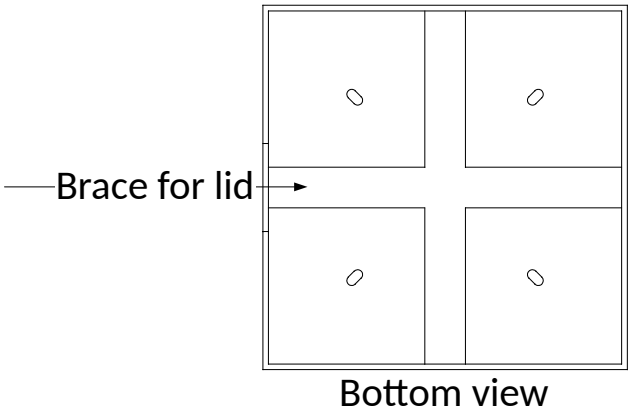



Figure 3.20: Support brace on the underside of lid

### Mounting

To secure the housing to the plate, a mount attached to the baseplate was required. The mount is illustrated in Figure 3.21. This mount allowed the housing to be removed when needed, but prevented it from moving unintentionally. It utilized a slot that fit into the negative slot shown in Figure 3.24, effectively locking the housing in place. There was one mount on either side of the housing. The part was modeled for one of the sides, and the other side was mirrored when printing it.

One of the essential parts of the housing was a Jetson holder. This was made to let the Jetson slide in and out easily, while not letting it slide out during transport. This was done by having a wall on two of the sides of the Jetson as shown in Figure 3.22, which provided support in two directions. The Jetson was also supported upward by one of the sides. The holder also had a nub to hold it from sliding out on the wall.

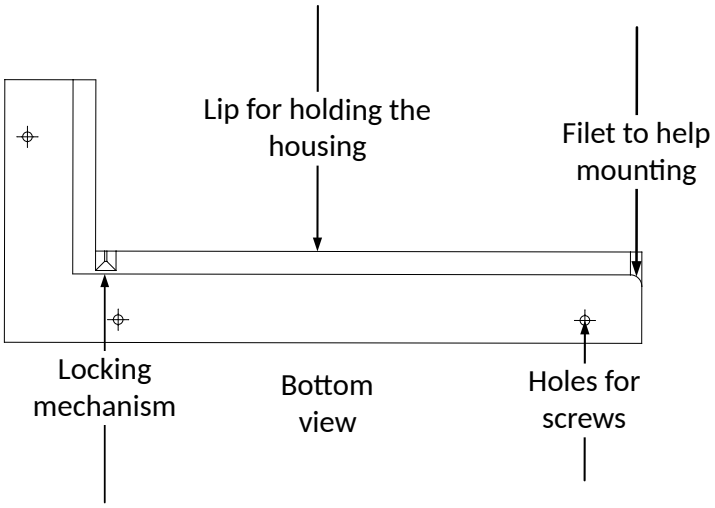



Figure 3.21: Mount for housing

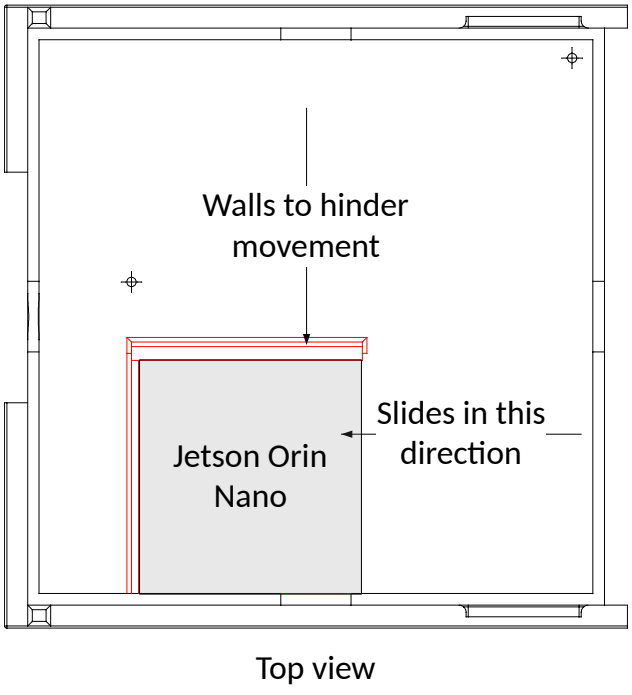



Figure 3.22: Placement of the Jetson

The power supply was mounted using two M3 bolts, which had a sunken in underside so that the nut was flush with the underside of the housing. The power supply was placed on the left side of the housing, and was mounted as far back as possible as shown in Figure 3.23.

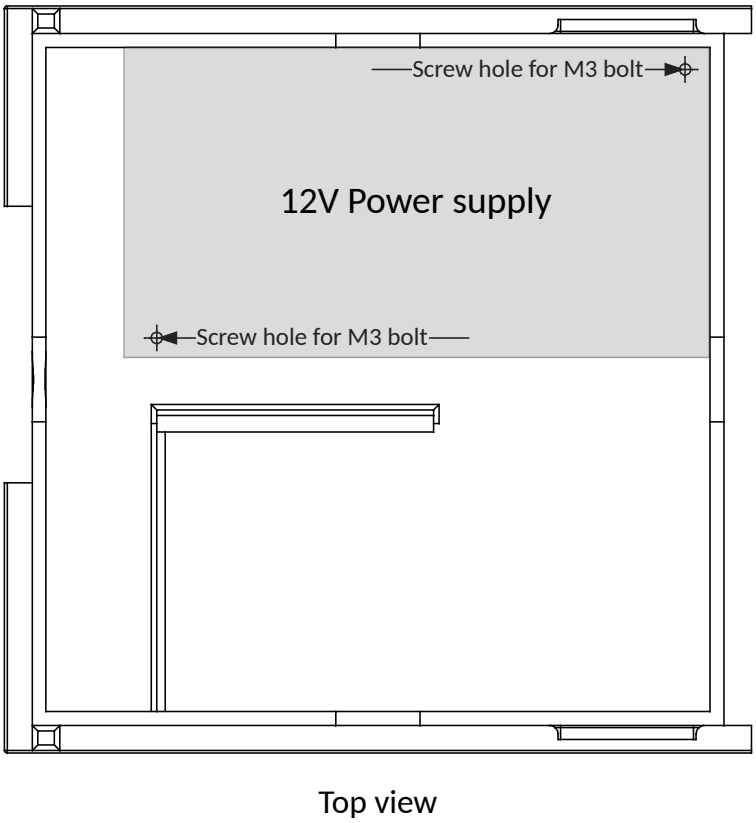



Figure 3.23: Mounting holes for power supply

To mount the housing to the plate, lips were made along the bottom of the housing as shown in Figure 3.24. They worked by sliding the housing into the mount for the lips, mounted on the base plate. The lips had a cutout to lock the housing in place to get consistent placement. The mount for the lips would go into place so it could only be removed from one direction.

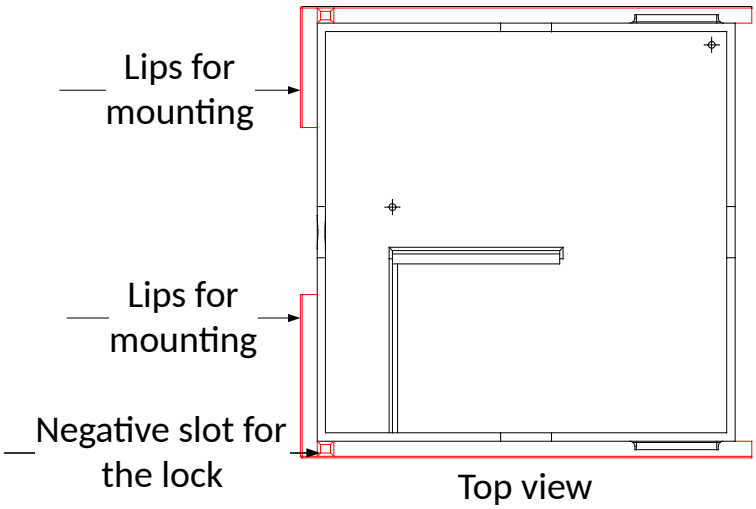



Figure 3.24: Lips for mounting

In case of spillage, the lid was designed to prevent batter from entering the housing. The overlapping lips as shown in Figure 3.25, made room for an easy mounting point for the locking pins while defending against fluids.

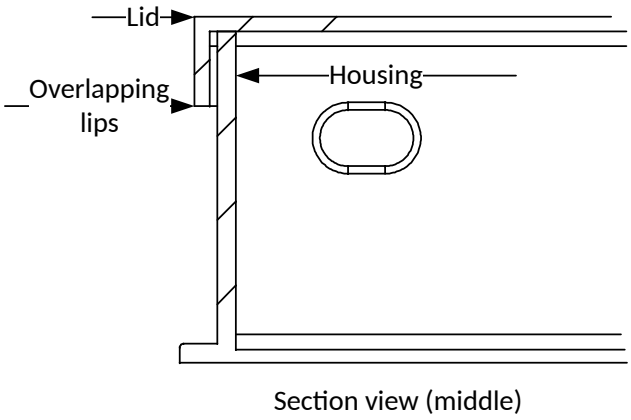



Figure 3.25: Overlapping lips of the lid

Since the lips overlapped, the fan got covered on the top. So a cutout on the lid was made as shown in Figure 3.26, to accommodate the fan.

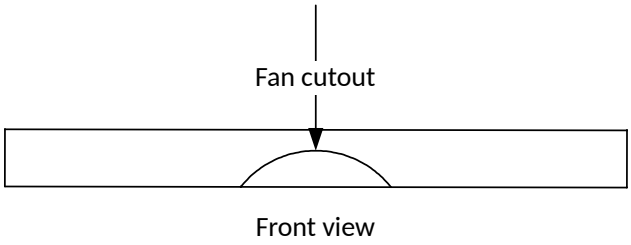



Figure 3.26: Cutout for fan in lid

The arm was to be mounted on top of the lid of the box, so mounting holes for the manipulator were made as shown in Figure 3.27. The mounting holes were made to match the base plate of the manipulator. The holes were made to fit M6 bolts.

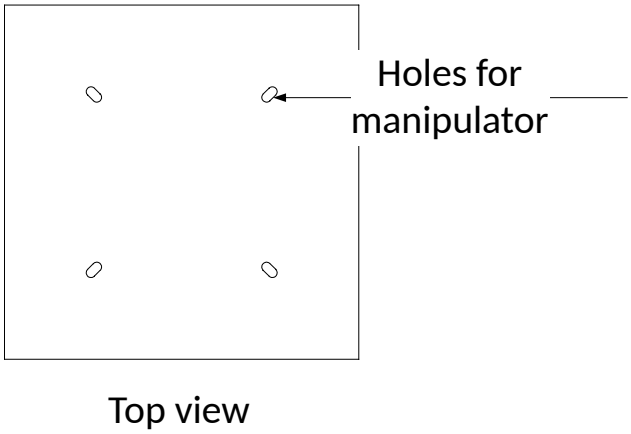



Figure 3.27: Mounting holes for manipulator on lid

The locking mechanism went through the lid and into the housing. The locking pins were inserted from two of the sides and made it easy to remove while still hindering the lid from falling off accidentally. The holes for the locking pins were located close to the edge as shown in Figure 3.32.

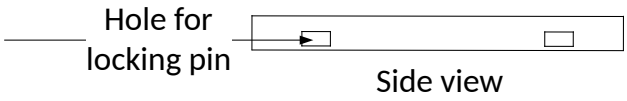



Figure 3.28: Holes for locking pins on lid

To have a stable reference point to the robot, a holder to host a special ArUco marker was made. This would be the placement for the origin marker, which would be helpful for reasons that will be explained in depth later. It attached to the lid. The holder was made to fit a 86mm by 86mm tag. The holder had alignment lips as shown in Figure 3.29, that aligned with the corner of the lid as shown in Figure 3.61.

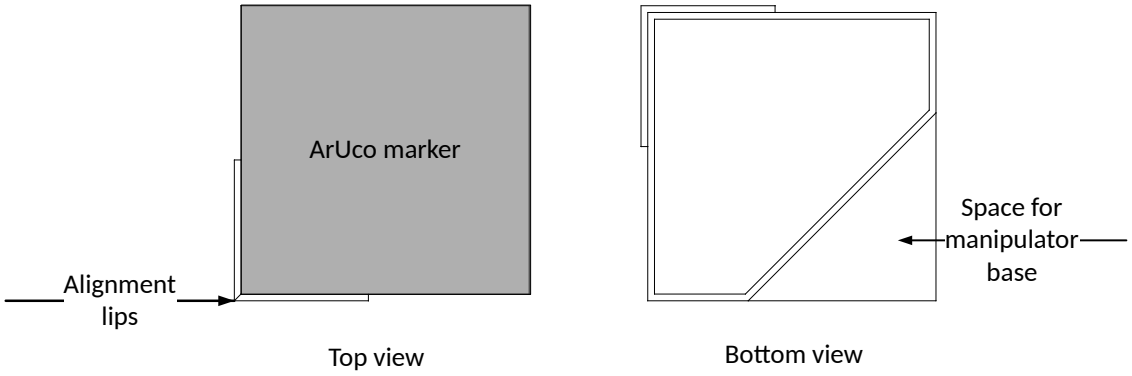



Figure 3.29: Origin marker mount

### Ventilation

To reduce temperatures in the housing, a fan was incorporated. The hole for the fan was made to fit a 80mm by 80mm 12V fan. during stress testing of the Jetson, the highest temperature measured was 38°C by the vents. The fan was centered on the housing as shown in Figure 3.30.

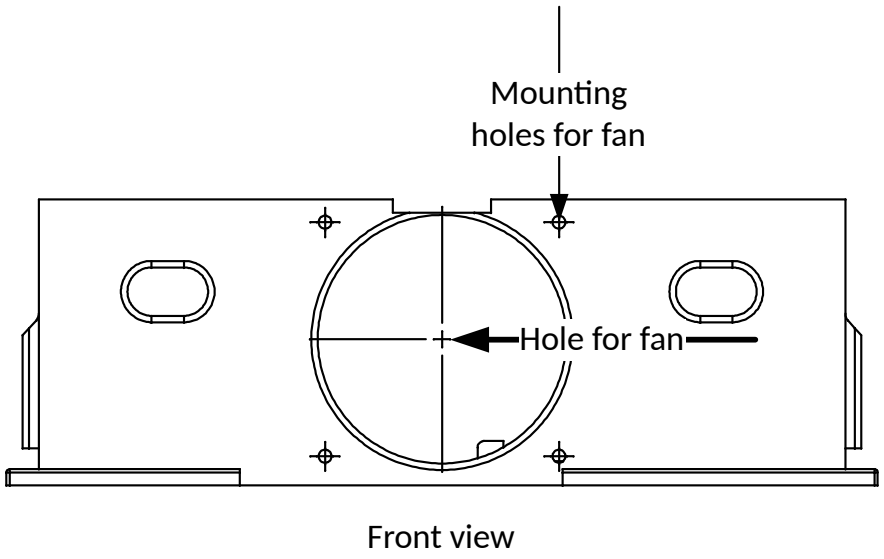



Figure 3.30: Fan hole and mounting points

To have good airflow over the components, vents were incorporated to two of the sides of the housing. The vents were positioned in the back of the housing as shown in Figure 3.31. The

vents were designed to prevent fluids from entering the housing in case of a malfunction, while still ensuring an adequate amount of airflow over the components.

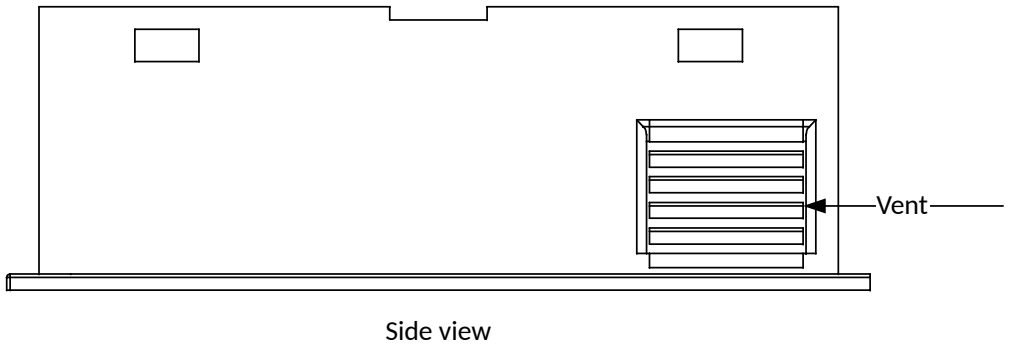



Figure 3.31: Vents on the housing

On two sides of the housing, slots were made for the locking pins to enter as shown in Figure 3.32. The locking pins penetrated the lid and the housing on four points to lock the lid in place in all directions. In Figure 3.32 it was illustrated that space was also made for the support braces that were incorporated into the lid.

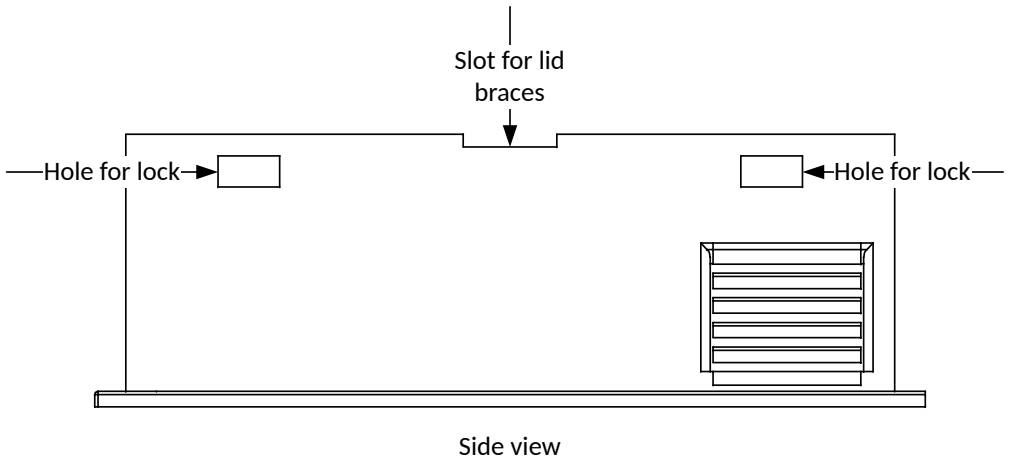



Figure 3.32: Holes for locking mechanism

### 3.2.5 Waffle Iron

The waffle iron was a Wilfa tradition WDWA-516W dual-sided waffle iron. This means that the robot had the potential to increase throughput compared to a single-sided iron. This iron had multiple surfaces that could be used to mount items on it. The waffle iron is shown in Figure 3.33.

Two ways of opening the waffle iron were considered. The first approach was using a motor. The other approach was using the manipulator to open the iron.

The motor based system is not optimal due to the cables needed to drive the motor. An illustration of the motor system is shown in Figure 3.34. The system is large and not space efficient. The leverage tower and motor would introduce extra mechanical failure points.

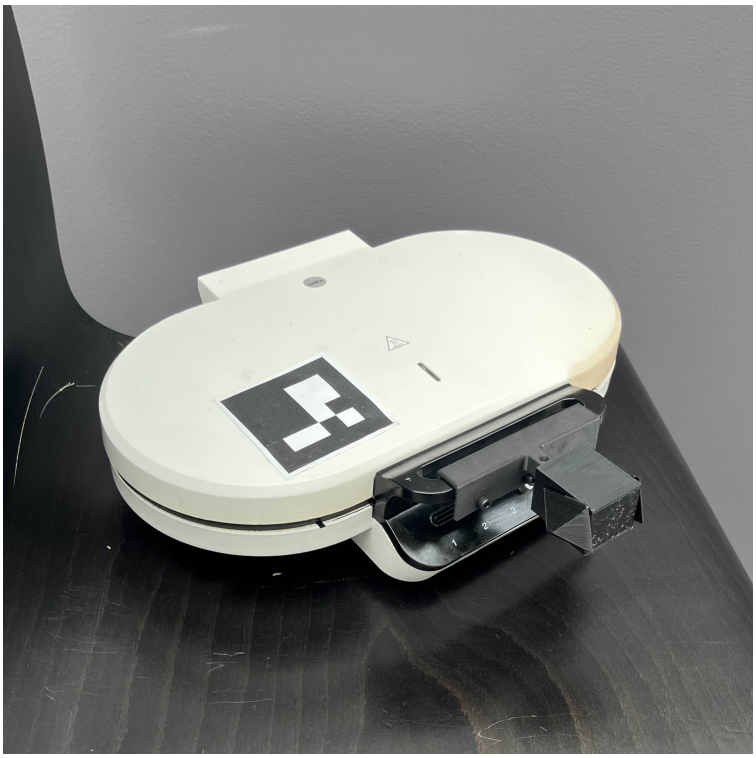


Figure 3.33: The waffle iron used in the project

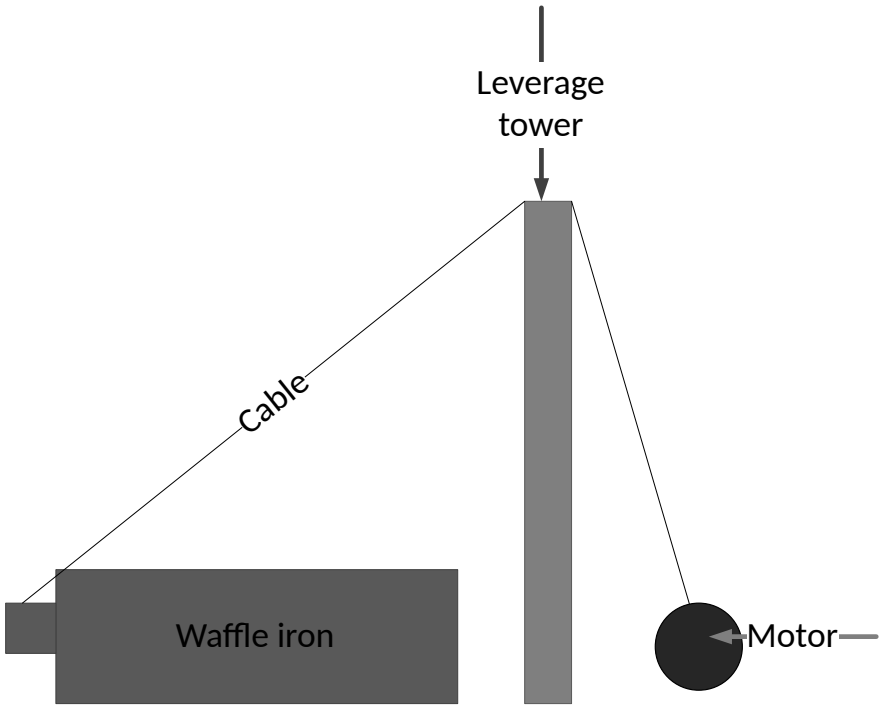



Figure 3.34: Illustration of leverage tower with motor

Since this project uses a manipulator that has a lifting capacity of 750g, the manipulator could open the iron on its own. This system had the benefit of not needing a leverage tower, and therefore lended a more refined look to the layout. By using the manipulator to open the iron, the manipulator ran the risk of being damaged by the heat if something were to happen while it opens the iron. It was decided to open the iron with the manipulator due to the presentation of the system, as well as reducing the number of components it would need.

An adapter from the waffle iron to its interface cube was needed. The adapter clamped onto the

handle of the waffle iron. The adapter allowed for both opening concepts as shown in Figure 3.35. The adapter was attached by clamping it with three M3 bolt and nuts to the handle of the iron.

The adapter would be subjected to heat and humidity from the waffle cooking process, so a material that can handle heat was needed. The material chosen was Onyx due to its high tolerance to heat. It starts to deform at 145°C[15], which would withstand the temperatures of the waffle iron. ABS was also considered as a cheaper heat resistant material. Unfortunately, it can only handle 86°C before deforming, which is not resistant enough for the temperatures it would experience.

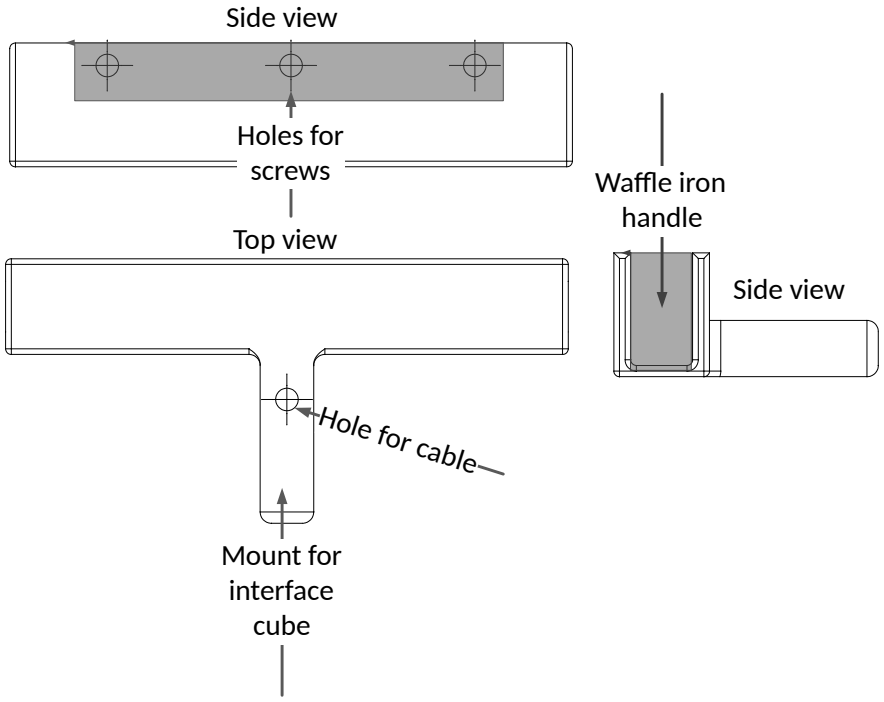



Figure 3.35: Handle adapter

An interface cube needed to be designed for the handle of the waffle iron. The aforementioned adapter was used as a base for the design. The first design was a rigid piece, shown in Figure 3.36, which was used during testing to test the arms ability to lift the iron. The rigid part made it hard for the arm to lift due to the arm needing to follow an arc while rotating the grippers to compensate for the rigid interface cube as shown in Figure 3.37. The arm didn't have enough reach to open the iron fully with the rigid cube so a new cube was made.

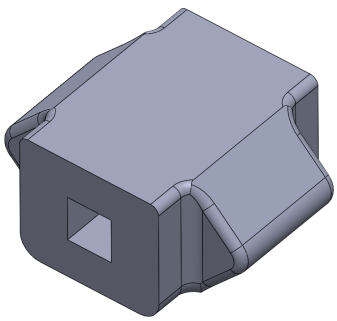


Figure 3.36: Tool version 1 for waffle iron

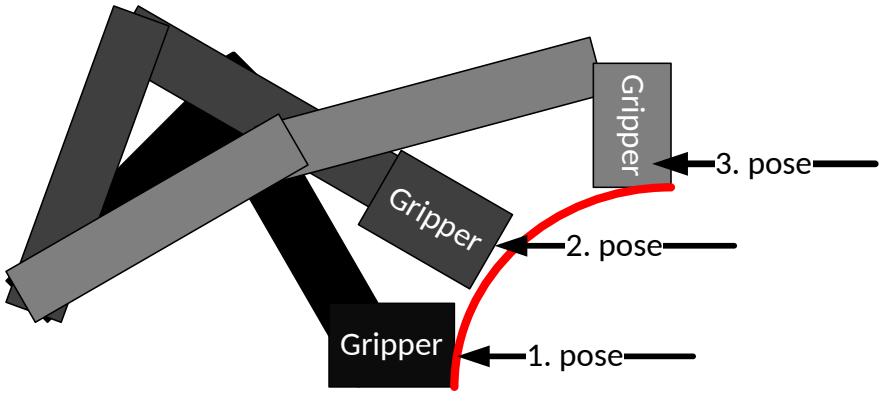
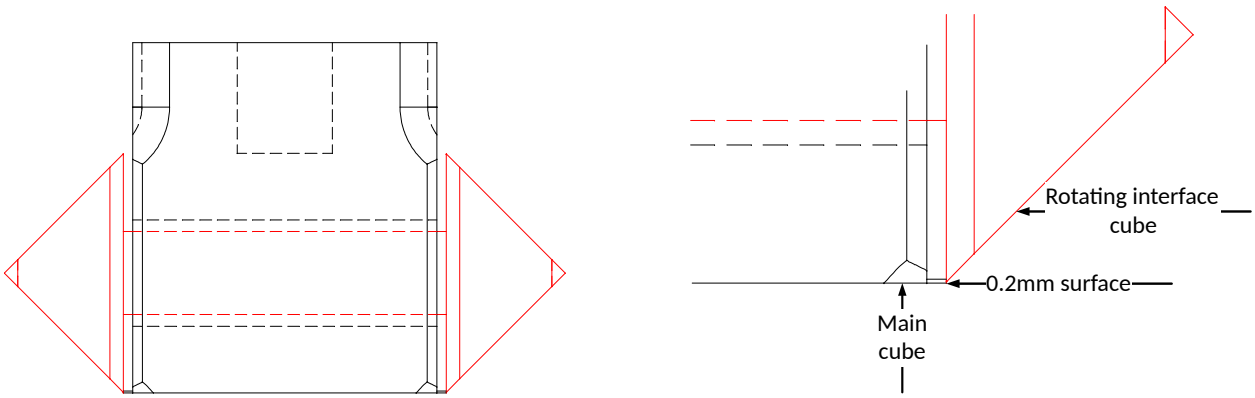



Figure 3.37: Opening the iron with a rigid cube

The second iteration of the design was made with a rotating core inside the interface cube. The rotating part simplified the motion the arm needed to open the waffle iron. This is due to the manipulator not needing to rotate about the iron (see Figure 3.37). The design was made to be a "print in place" design, meaning that it could be printed in one piece without assembly, but still allow for movement. This is a feature not every part has or can have due to their mechanical design. The pin in the middle of Figure 3.38a, marked in red, rotated freely from the body. The front of the rotating cube had a 0.2mm support structure to help with printing, as shown in Figure 3.38b. This surface worked as a supporting structure during printing. It was removed from the finished design once the print was finished.



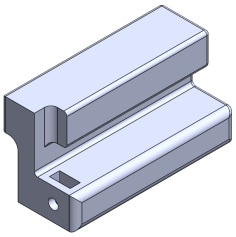

(a) Top view

(b) close up view of the 0.2mm surface

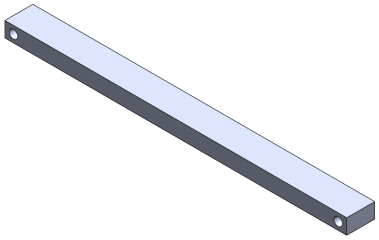
Figure 3.38: Tool version 2 for waffle iron

When the iron was opened, it was at risk of being pushed off the base plate by the robot. Because of this, a mount was created to stop the iron from being pushed off the table or lifted. The mount was made such that the waffle iron could be easily attached or removed, while still preventing movement.

The iron was mounted by attaching an assembly to the iron that slots into a baseplate mount. The assembly is called the waffle iron mount. The waffle iron mount attached to the iron and is a 3 part assembly. The parts within the waffle iron mount were connected together using M3 bolts and nuts. The part that was on the back of the underside of the waffle iron was a long beam as shown in Figure 3.39b, connecting the two side pieces. One of these side pieces are shown in Figure 3.39a. The 3 parts of the waffle iron mount slid under a cable holder on the center of the underside of the waffle iron shown in Figure 3.40a that was used to coil the cable around when storing the iron. Then the 3 parts were attached together to prevent the assembly from falling out. Only one of the two side pieces were designed, then the other side was mirrored in the slicer before printing. The side with the nut and bolt slid under the cable holder. Figure 3.40b shows where the side pieces are attached when fully assembled.



(a) Waffle iron mount side piece



(b) Waffle iron mount backside

Figure 3.39: Parts for assembly for waffle iron mount

(a) The underside of the iron

(b) Closeup view

Figure 3.40: Mount for the Iron

The assembly was assembled by inserting an M3 bolt from the long beam and into the side parts, through the hole marked blue in Figure 3.41. An M3 nut was inserted into the green hole in Figure 3.41.

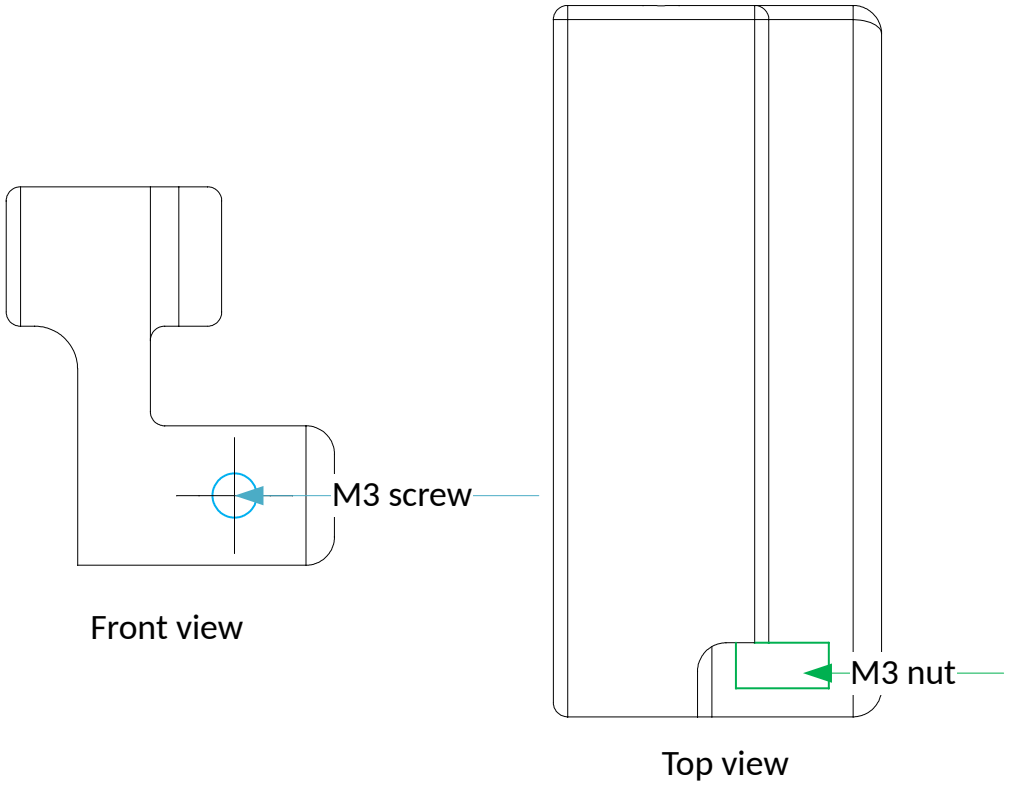
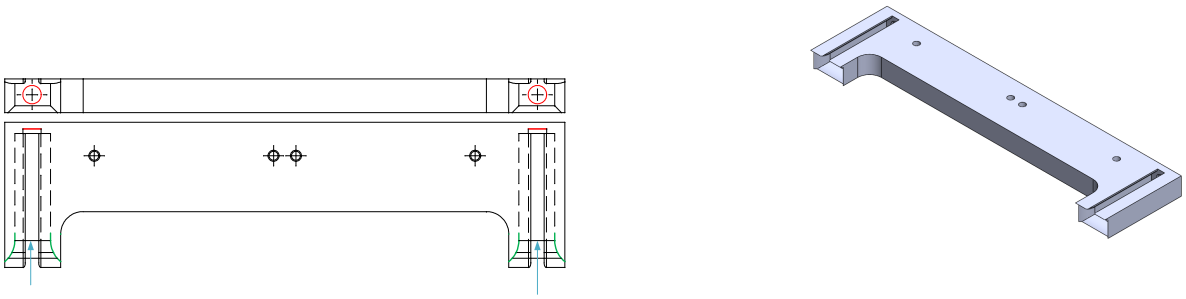



Figure 3.41: Assembly holes for bolts and nuts

The mount in the base plate of the waffle iron was designed such that the iron would not slide out during use, while still being able to be removed during packing for transport. The mount was bolted down into the base plate and allowed for little to no movement, except for the direction pointing towards the robot. In this direction, there was no significant force being applied. In Figure 3.42 the mounting direction was shown with blue arrows. The green color marks the funnels that were used to make the mounting process easier. The area marked in red shows slots for 8mm by 2mm magnets which were supposed to keep the iron firmly in place, but proved under testing to not make a difference. All the parts shown in Figures 3.39a, 3.39b, 3.41 and 3.42 were printed in ABS to have a higher heat resistance. Onyx was not needed here as ABS was resistant enough to heat, at a lower price. The parts that were attached to the iron were glued to further hinder movement of the iron.



(a) Mount on plate for waffle iron

(b) Waffle iron mount for the baseplate

Figure 3.42: Plate mount for the iron

Usually when taking a waffle out, one uses a fork. This works due to the humans ability to act based on what it sees, and dynamically adapt its movements based on visual feedback. For a robot to manually take each waffle out without ripping the waffles or damaging the iron would be an issue.

Inserting metal rods during the baking of the waffles was a concept which seemed to be better than the fork. The rods were inserted into the iron, where the batter would be poured on top. Then the waffles were cooked with the rods inside. The manipulator would then lift the rods with the waffles attached to the rods, extract the waffles and deliver them. The rods were then inserted in the iron again and the cycle continued. The rods were made out of 316 carbon steel which is a well suited material for food[30]. Two rods were inserted on each side of the iron to increase surface area and reduce the chance of ripping apart the waffles.

To extract all the rods at once, a tool to hold all the rods was needed. The rods were mounted 50mm away from each other as shown in Figure 3.43 and mounted 50mm away from the center on each side of the iron. The tool for the rods needed to be long enough to hold all four rods while not restricting the ability of other components. This includes the interface cube for opening the iron or the heat regulator to determine the heat of the iron. The interface cube was mounted to the left side of the holder as shown in Figure 3.43. The design also incorporated a slot for the heat regulator on the waffle iron. The braces on the interface cube further increased stability by bracing against the flat part of the grippers so the mount would have smaller deflection on the opposite side of the interface cube due to rotation within the grippers. The support slots had legs which were mounted in the tool to hinder the rods from falling out when opening the iron. The legs were simple beams which fit into the slot and allowed for height adjustments.

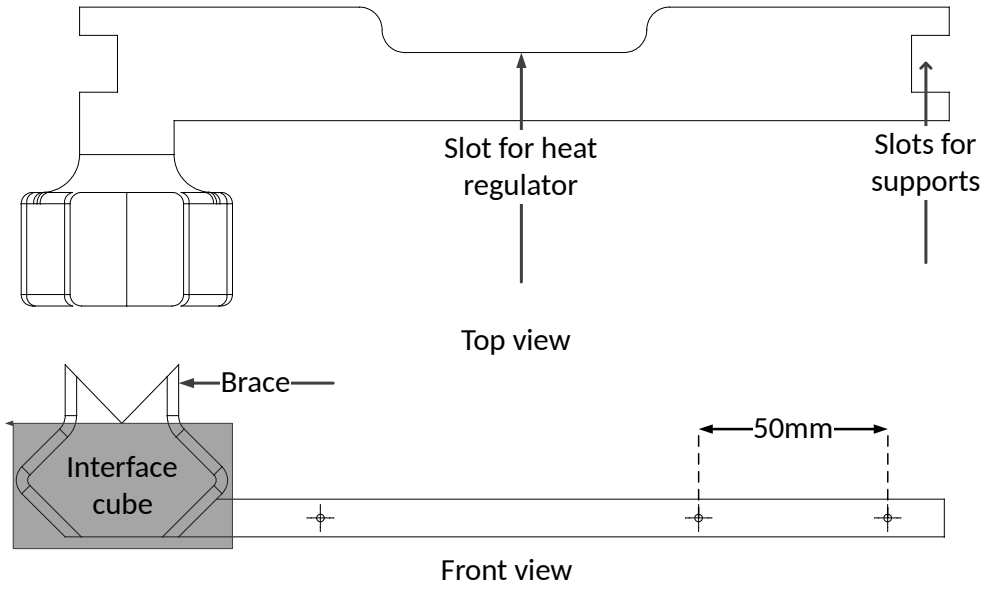



Figure 3.43: Interface cube and mount for rods

Cutouts were made in the waffle iron to accommodate the waffle extraction tool. These cutouts were made 50mm from the center on each side with another slot a further 50mm out from the first slot as illustrated in Figure 3.44. The cutout was made to fit 2mm metal rods.

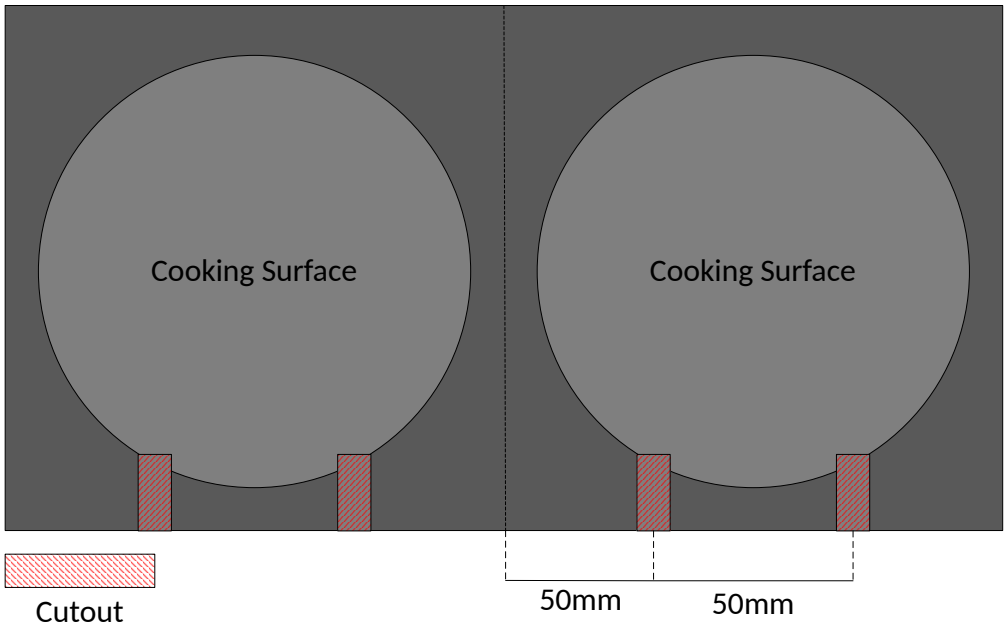



Figure 3.44: Cutout within the bottom of the cooking surface.

When the waffles were delivered, they were pushed off the rods by another set of rods that pointed straight up called delivery rods. At the delivery station, there were two sets of rods which were used to remove the rods baked into the waffles. These allowed the waffle extractor to pull the waffle extractor rods out of the waffle before making new waffles. The rods were made out of 316 carbon steel[30].

### 3.2.6 Batter Storage

To store batter, a bowl was used. A bowl was a solution with low complexity and ease of cleaning. The bowl based concept would require few parts, as this design would only necessitate a way to pick up batter and carry it to the iron. To solve this, a ladle was employed. The design is very human like and it would emphasize that the manipulator can complete typical human tasks, to observers. Using a bowl would be inexpensive, but would cause a risk of spillage of batter.

A way was needed to mount the bowl to the baseplate for the manual control mode. The bowl had slots in its base which could be used for mounting. This served as a way for the bowl to be mounted while still allowing it to be easily removed for washing.

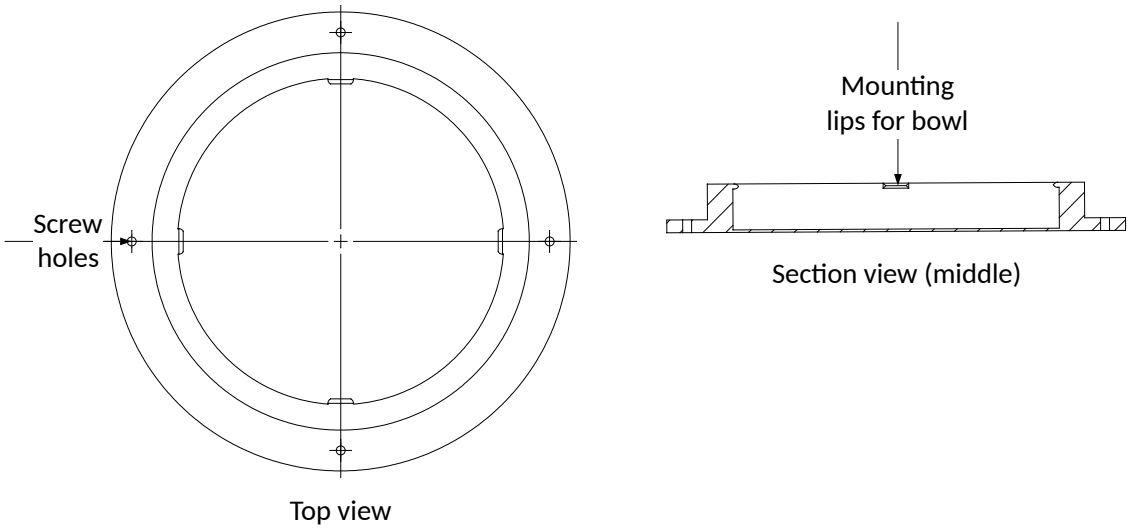



Figure 3.45: Holder for bowl

The holder for the bowl was bolted on the plate to securely fasten it. The holder had four pins on each quadrant of the holder, as shown in Figure 3.45, which sticked into the base of the bowl when it was mounted.

### Ladle Adapter

To reliably pick up the ladle, an interface cube was made for it. The cube consists of two symmetric parts. The ladle was inserted through the holes as shown in Figure 3.46. The parts where then glued together and slid to the top of the ladle.

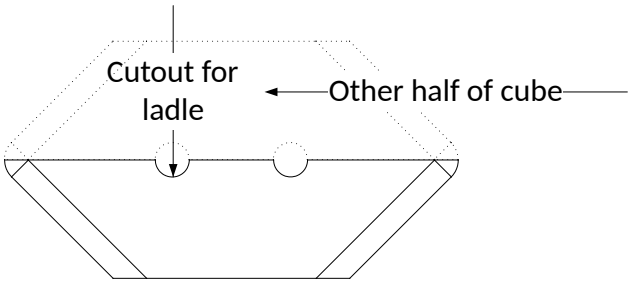



Figure 3.46: Interface cube for ladle

To increase reliability, the ladle should be placed in the same spot every time. The ladle guide was made such that it would guide the ladle into the spot the manipulator would use to pick it up. The ladle guide would also incorporate a mount for the ArUco marker that would be placed for automatic mode as shown in Figure 3.47. The guide was mounted on top of the bowl and held in place by friction as shown in Figure 3.48

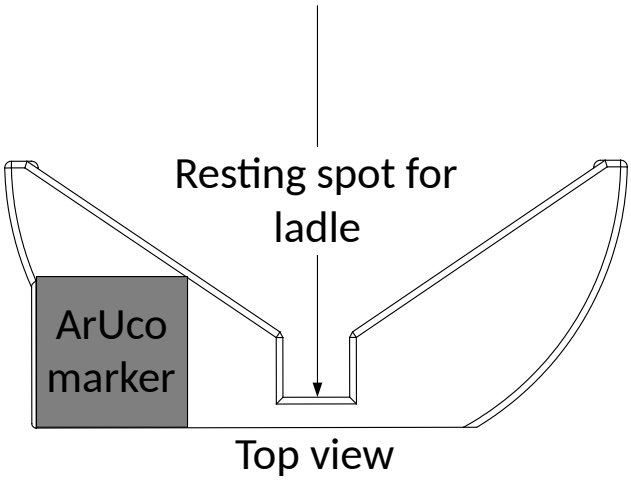



Figure 3.47: Ladle guide and ArUco holder

Figure 3.48: Demonstration of the final iteration of the bowl

### 3.2.7 Emergency Stop Button

A physical emergency button was going to be implemented. The first design had a connection to the GPIO pins on the Jetson, but having a cable routed through the vents that could easily be unplugged was not beneficial. This was why the emergency stop button was moved to the touch screen. It would have a dedicated button on the screen that, when pressed once would freeze the robot. When pressed again it would return to "home" position.

The emergency stop button had an unique need to be used at any time during program execution. Due to this need, a multi-threaded software approach was used. The structure of the program flow can be seen in Figure 3.49.

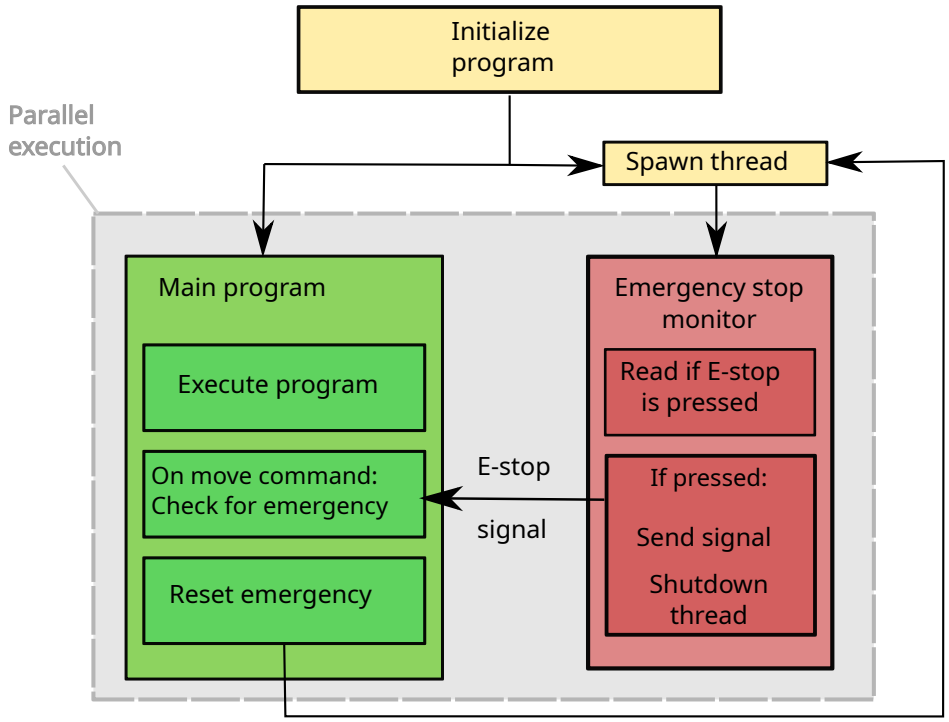



Figure 3.49: Model of the emergency stop software

During program initialization, the main program spawned a second thread that ran concurrently to the main program. If the emergency button were to be pressed, a signal would be sent to the main thread that handled the error handling logic. In the main thread, the program checked if the emergency stop signal was set on a per-movement basis. If a stop signal was received, the program would break its ordinary control flow and enter an error state which could not be exited until the emergency was cleared by the operator.

### 3.2.8 Camera

One of the initial challenges of the project was determining an effective camera layout. The project began with a single Intel RealSense L515 mounted on a pole to overlook the workspace. However, this camera had a limited field of view (FOV) of 70° by 55°, that could only cover about half of the workspace at a reasonable distance. It had minimal software support, prompting its replacement with the D435i, which also suffered from a narrow 69° by 42° FOV. Eventually, the RealSense D455 was used, offering a much wider FOV of 90° by 65°. The increased FOV allowed complete coverage of the workspace without requiring a significantly distant placement[9]. Three layout configurations were considered:

1. **Single stationary camera:** Positioned overhead to capture the entire workspace.
2. **Two stationary cameras:** One placed high for global detection, and another positioned closer to areas requiring precise detection. This would add complexity due to the need for accurate synchronization and calibration between both cameras.
3. **Hybrid layout:** Combining a stationary overhead camera with a dynamic camera mounted on the robot arm. The stationary unit located regions of the workspace, while the mobile camera refined the position by approaching the marker directly.

A **single stationary** D455 camera was found sufficient for the core machine vision tasks. The two stationary cameras setup remains valuable for specialized use cases. Such as being repurposed to detect user actions to power entertainment modes.

The camera was mounted on a 20mm by 20mm aluminum extrusion. The extrusion made it easy to move the camera to an optimal angle and provide the best visibility. The extrusion was mounted to

a camera mount. The camera mount which attaches to the extrusion is shown in Figure 3.50. The camera mount was fastened to the workspace with a slide-in mount as shown in Figure 3.51. This made it so the camera had a set position, but it was able to be removed for travel. The camera was mounted to the extrusion using a 3D-printed GoPro mount. The GoPro mount was chosen because it could be tilted, making it possible to adjust the view angle of the camera.

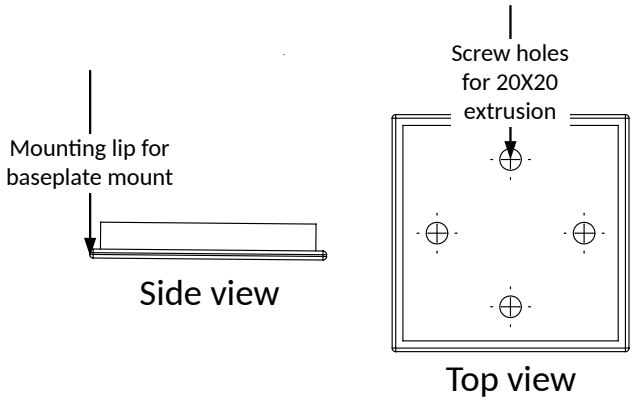



Figure 3.50: Camera mount from extrusion to baseplate

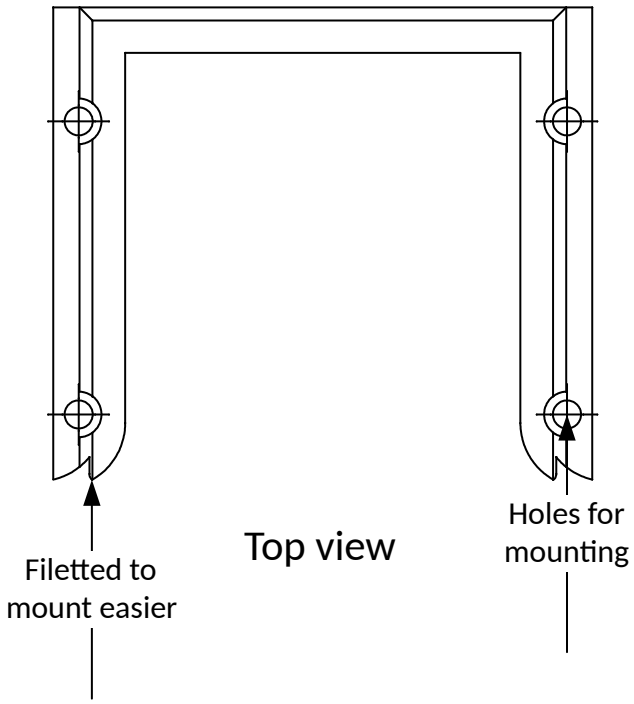



Figure 3.51: Camera mount from baseplate

The GoPro mount was modified from an online model[14], where the bolt holes were added later for the part that attaches to the 20x20 extrusion. The part that attached to the camera was also a GoPro part that was designed to fit the RealSense camera. The assembly shown in Figure 3.52 shows how the two parts were connected with a M4 bolt through both parts to lock them in place. There were T-nuts inserted in the extrusions to hold the assembly fastened to the extrusion.

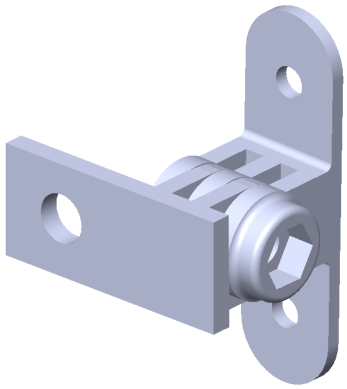


Figure 3.52: Camera holder

## 3.3 Software Design

This section presents the software design of the robotic system, detailing the architecture and key functional components. Subsection 3.3.1 describes the human-machine interface (HMI) and the state machine that manages task flow and system behavior. Subsection 3.3.2 outlines the vision processing pipeline, including fiducial marker pose estimation and hand detection. Subsection 3.3.3 covers how robot motions are handled, including pose recording, manual and automatic operation modes, as well as safety mechanisms such as collision checking and adherence to joint limits.

### 3.3.1 Control Software

The control software was responsible for coordinating all autonomous actions of the robot during the waffle-making process. It integrated three main components: A state machine for high-level decision making, a touchscreen-based Human-Machine Interface (HMI) for user interaction, and a motion control system that defined and executed robot arm movements. Together, these components ensured reliable and repeatable operation with minimal user intervention.

The robot was controlled by a state machine, which decided the robots course of action as it automatically made the waffles. In the state machine, each state represented a distinct step in the waffle-making process. Examples of these behaviors included tasks such as picking up tools, moving to predefined positions, or performing visual detection tasks using the vision system described in Section 3.3.2. Figure 3.53 illustrates the flowchart of the state machine. The process began by opening the waffle iron and then entered a loop that cycled through the subsequent states. If an error occurred in the logic, the system transitioned to an error state. In this state, the robot arm halted, allowing the operator to safely resolve the issue before resuming operation. Each state included logic to handle error detection. For instance, in the "Open Iron" state, the system had to verify that the waffle iron was open before opening the iron. This verification was performed using an ArUco marker mounted on top of the iron. If the iron were not open, the system transitioned to the error state.

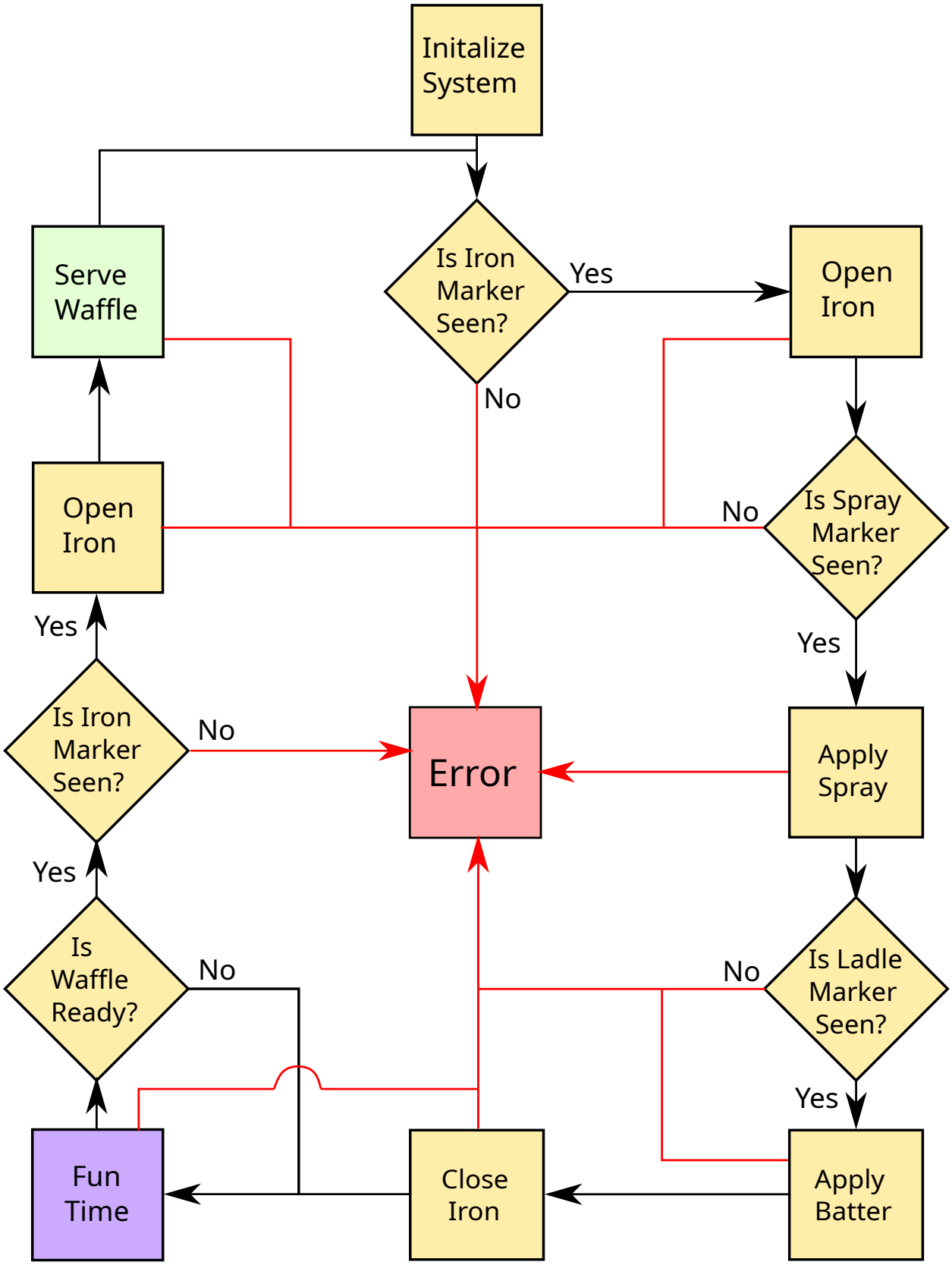



Figure 3.53: Flow chart of the state machine

Transitions were hardcoded and sequential, meaning that after completing one state, the state machine changed automatically to the next state. This state machine was developed to ensure that the robot arm could execute a complete cooking sequence independently, without the need for continuous human supervision. By clearly defining each step and transition, the system guaranteed consistent behavior, making it well-suited for repetitive tasks.

The Human-Machine Interface (HMI) was made as a touchscreen interface. It was built using *Custom Tkinter*, a Python library for GUI development[1]. The home screen let the operator start the waffle-making process by pressing the "Make Waffle" button, seen in Figure 3.54.

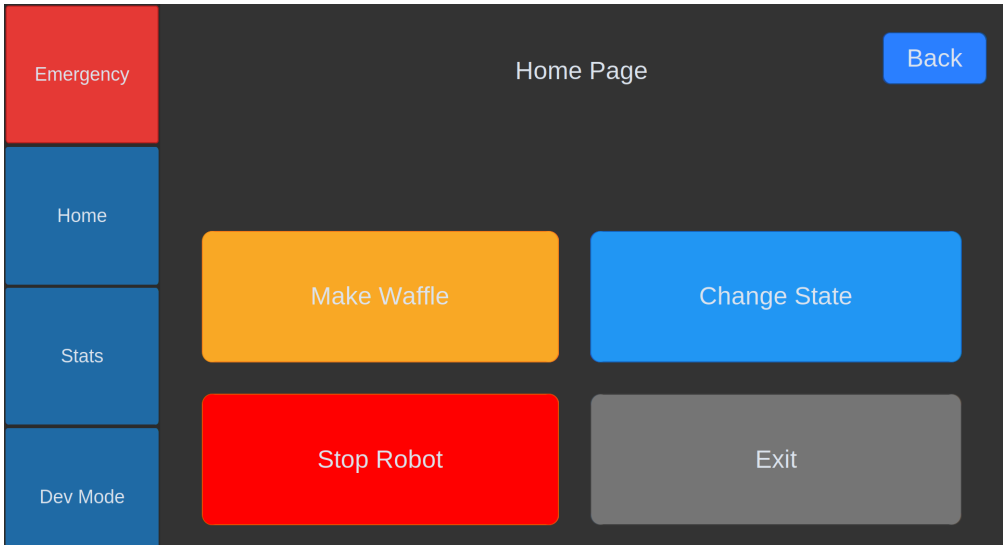



Figure 3.54: Main page of the HMI

There were also options to pause the process and view the camera feed for maintenance. To enter developer mode, the user had to enter a four-digit password on a keypad, shown in Figure 3.55.

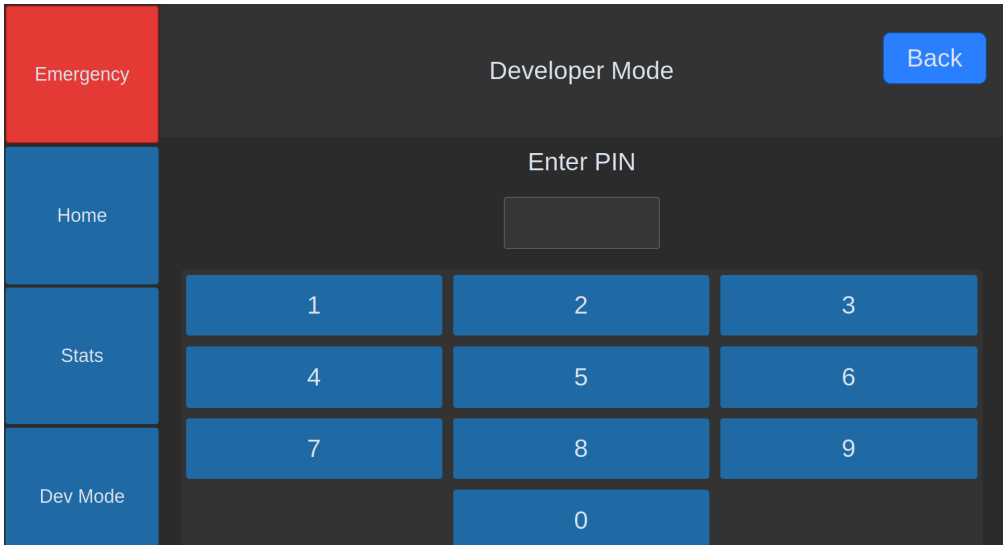



Figure 3.55: Password prompt for accessing developer mode

In developer mode, the user could reboot the system and manually change the current state of the state machine. Developer mode is shown in Figure 3.56.

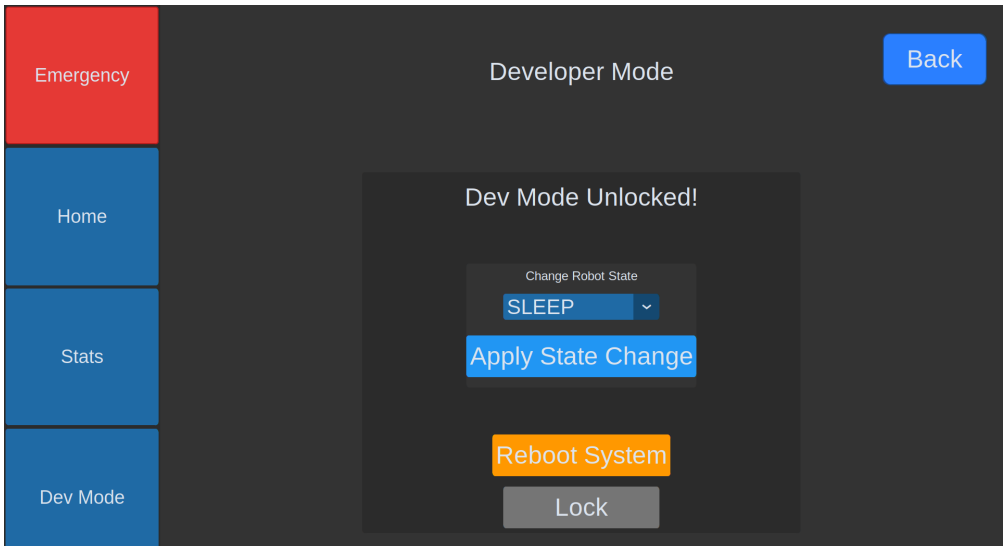



Figure 3.56: Developer mode interface

The stats page shows the current state and whether the robot is idle or moving. It also includes a counter for how many waffles have been made. This page is shown in Figure 3.57.

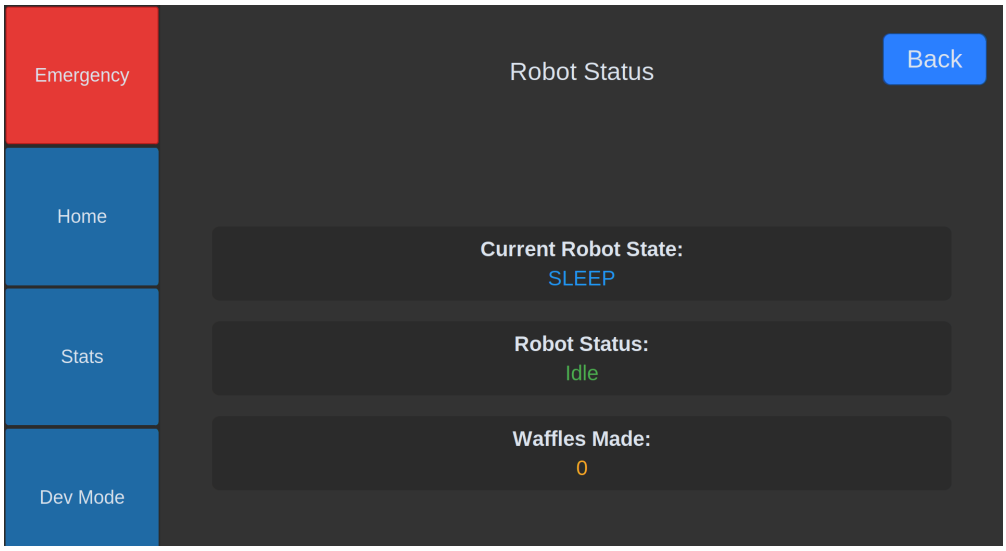



Figure 3.57: Statistics interface

To stop the robot in case of an emergency, an emergency button was placed on the side of the display. This button was always visible for quick access. The emergency screen is shown in Figure 3.58.

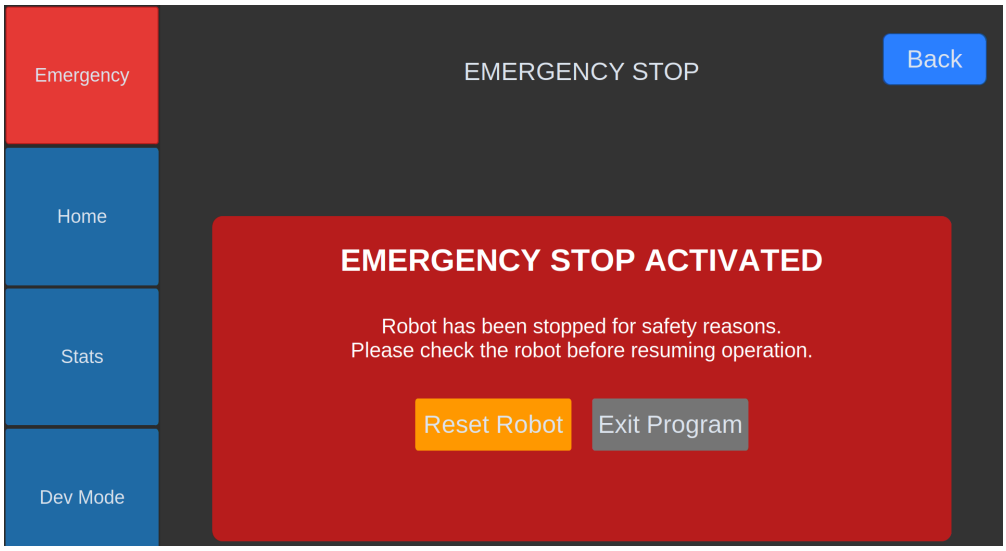



Figure 3.58: Emergency interface

### 3.3.2 Camera Software

To enable the robot to analyze and interact with its environment, several machine vision-based methods were explored. Early attempts were based on using AI for object detection, followed by a more robust final solution based on fiducial markers. Additional features like hand tracking, gesture recognition, and interactive modes were also developed to enhance user engagement.

To fetch images from the camera for the software, the SDK for the Realsense camera had to be installed. The Intel RealSense SDK for the camera was compiled from source with the RSUSB backend on the Jetson. This approach avoids kernel patching, which could conflict with the Jetson's operating system linux kernel. The RSUSB backend facilitated seamless integration with the Jetson's ARM-based system.

An early idea in the project was to use Artificial Intelligence to detect and identify objects in 3D space, using a depth camera to provide spatial context. For this project, the plan was to use an object detection model like YOLOv8, known for fast image and video analysis [34]. An example of the model in use is shown in Figure 3.59.

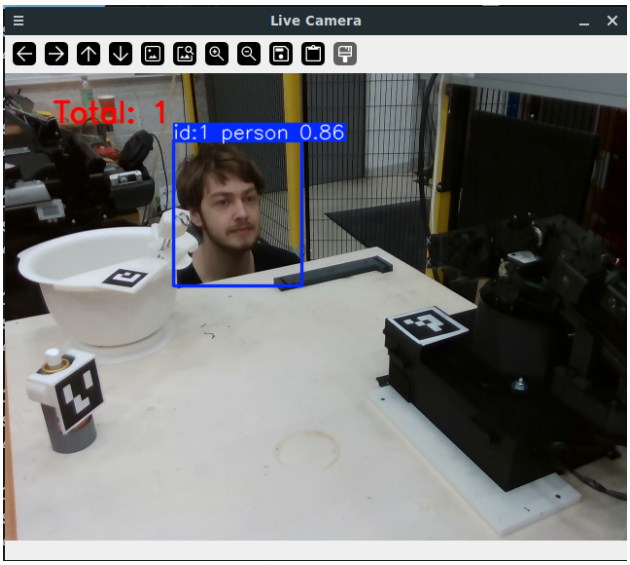



Figure 3.59: Example image demonstrating YOLOv8 in action.

However, this approach was dropped after the workspace layout was designed. While AI offered

spatial awareness, it struggled to reliably track moving or repositioned objects using only the depth camera. Reaching the required level of accuracy would have needed a more complex and stable setup than was realistic for this project. Instead, fiducial markers were chosen. Fiducial markers are visual patterns that make pose estimation simpler and more reliable for dynamic object tracking.

Fiducial markers are commonly used in robotics for visual localization and pose estimation, allowing systems to detect and track known patterns. Initially, AprilTags were considered due to their compatibility with ROS2 and RViz visualization[35]. However, to avoid the added complexity of involving ROS2 in a Python-based environment, the project adopted ArUco markers (see Figure 3.60) using OpenCV. This made integration with the existing robot software simpler and more efficient by removing the overhead in using ROS2 communications.

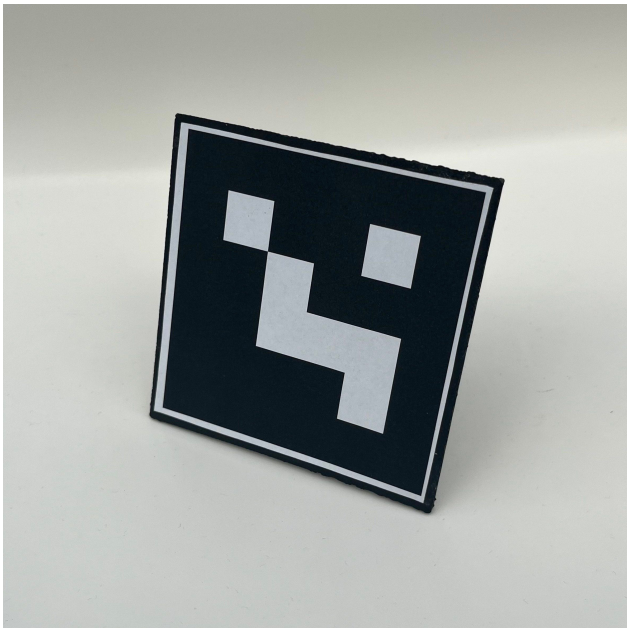


Figure 3.60: An example of an ArUco Marker

OpenCV had an ArUco sub-module that included everything needed for marker detection, pose estimation, and constructing reference frame transforms relative to the robot's coordinate system[19].

To detect the transformation matrix of a marker, the camera software first had to detect the positions and ID's of each marker, and then transform it from pixel coordinates to a real 3D pose. To determine the location of the marker in pixel coordinates, the `cv2.aruco.detectMarkers()` function from the OpenCV library was utilized. Prior to detection, the input image was converted to grayscale, as the ArUco marker detection algorithm relies on pixel intensity (brightness) to identify square patterns. The algorithm then analyzed each detected square to determine the marker's unique ID by interpreting the arrangement of black and white cells within its internal grid. Once a valid ID was recognized, the marker was classified as detected. Subsequently, the algorithm computed the two-dimensional coordinates of each of the marker's four corners.

The function `cv2.solvePnP()` was used to find the position and orientation (pose) of a 3D object relative to the camera. It solves the *Perspective of n Points*, which involves estimating the 3D pose of an object from its known 3D geometry and 2D coordinates from the image. The 3D geometry is known from the size and arrangement of an object given to the function.

The term "perspective" refers to the projection of 3D points onto a 2D image plane through the camera lens, where objects farther from the camera appear smaller and closer points appear larger. This perspective projection causes the object's appearance in an image to change depending on its distance and orientation relative to the camera. By analyzing the relationship between the known 3D geometry and its 2D image projections, `solvePnP` computes the rotation and translation vectors that describe the object's pose with respect to the camera coordinate system.

In this project, ArUco markers were used to provide both sets of points. Each ArUco marker had a known size. Since any given marker was flat and square, its 3D corner coordinates could be defined in a local coordinate system. For a marker centered at the origin, the corners were defined as Equation (3.1) illustrates. Each row represented a different corner of the marker.

$$\begin{bmatrix} -\frac{s}{2}, & \frac{s}{2}, & 0 \\ \frac{s}{2}, & \frac{s}{2}, & 0 \\ \frac{s}{2}, & -\frac{s}{2}, & 0 \\ -\frac{s}{2}, & -\frac{s}{2}, & 0 \end{bmatrix} \tag{3.1}$$

$s$ is the side length of the marker. These represent the 3D geometry of the marker. When the marker was detected in the camera image using `cv2.aruco.detectMarkers()`, the pixel coordinates of its corners were returned. Several optimizations were performed on the 2D pixel coordinates of the markers. These included sub-pixel refinement, which enhances accuracy by estimating marker corners at sub-pixel resolution, utilizing the built-in parameters of the OpenCV library. These were the 2D image points. The `cv2.solvePnP()` function took the 3D object points, 2D image points, the camera matrix (intrinsic parameters), and distortion coefficients of the camera as input. It then calculated two vectors. The rotation vector (`rvec`), which described how the marker was rotated in space, and the translation vector (`tvec`). The translation vector described how the marker was shifted, relative to the camera.

To use the marker pose in transformations, the rotation vector `rvec` was first converted into a 3×3 rotation matrix using the built in function in OpenCV: `cv2.Rodrigues()`. Then, this matrix was combined with the translation vector `tvec` into a 4×4 homogeneous transformation matrix. This matrix defined the full 3D pose of the marker in the camera's coordinate system and could be used for spatial alignment with the robot or other markers.

To ensure a consistent spatial reference for the machine vision software, an ArUco marker, named the origin marker, was designated as a fixed reference point relative to the robot. Using the origin marker and a static offset transform, all other marker poses could be expressed in terms of the robot's coordinate system. Figure 3.61 shows the origin marker mounted on the robot.

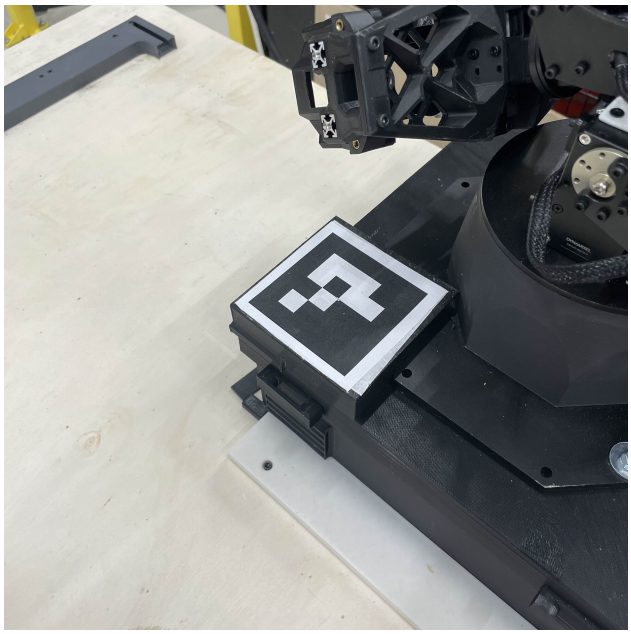


Figure 3.61: Origin marker mounted adjacent to the robot's base frame.

The spatial relationship between the origin marker and the workspace markers was computed using homogeneous transformation matrices. Specifically, the transformation $\mathbf{O_M}$ from the origin marker $\mathbf{O}$ to a target marker $\mathbf{M}$ is given by:

$$\mathbf{O_M} = \mathbf{C_O}^{-1}\, \mathbf{C_M} \tag{3.2}$$

In Equation (3.2), $\mathbf{C_M}$ represents the transformation from the camera to the marker, while $\mathbf{C_O}^{-1}$ is the inverse transformation from the camera to the origin marker. Together, these were used to compute the position of the marker relative to the origin. The resulting matrix was then reoriented to comply with the coordinate orientation used by the robot, where the X-axis points forward, the Y-axis to the left, and the Z-axis upward. The camera used for marker detection is shown in Figure 3.62. It was mounted in a top-down configuration to provide a birds-eye view of the workspace.

Figure 3.62: Camera mount used for workspace marker detection.

Although the origin marker was not physically placed at the robot's true origin, it was placed to be rotationally aligned with the robot's coordinate system. As depicted in Figure 3.63, a fixed translation offset was applied to shift the virtual frame to coincide with the robot's actual base frame.

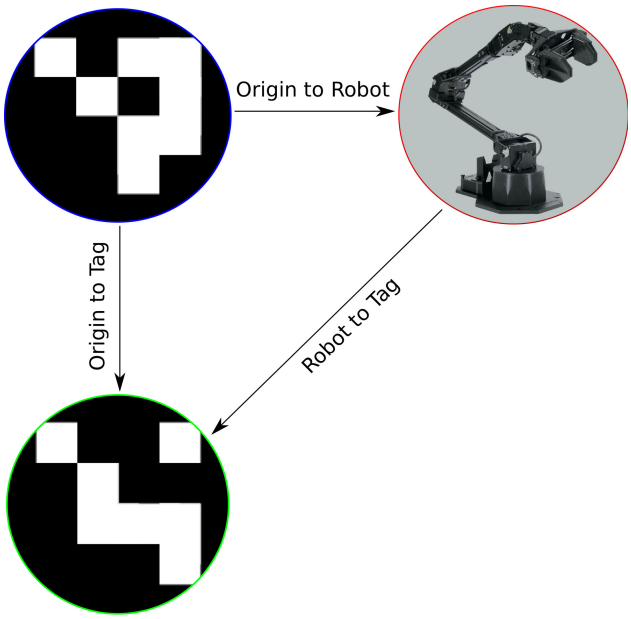



Figure 3.63: Vectors between the origin marker, the robot's true origin, and a workspace marker.

The final transformation from the robots base frame to any marker in the workspace was expressed as a homogeneous transformation matrix that incorporated this static offset:

$$\mathbf{R_O} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x - \text{offset}_x \\ R_{21} & R_{22} & R_{23} & t_y - \text{offset}_y \\ R_{31} & R_{32} & R_{33} & t_z - \text{offset}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Equation (3.3) shows a way to convert the origins position into the robot's coordinate system, where $\mathbf{R_O}$ is the transformation from the robots origin to the origin marker. This made it possible to track positions accurately in the workspace.

For entertainment mode, a marker could be used for interactive play by making the robot arm follow it.

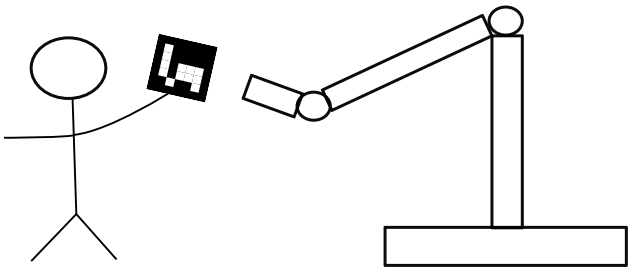


Figure 3.64: Concept: Robot arm following a marker

As illustrated in Figure 3.64, the robot could set a positional offset to target a point near the detected marker. By placing the marker in the hands of a user, they could control the movement of the robot arm simply by moving the marker. This enabled a playful and interactive experience that aimed to evoke a sense of joy and amusement in the user.

Another feature that was implemented was hand tracking, which was used while the waffle was cooking; in entertainment mode. The tracking system was based on Google's pretrained AI model MediaPipe [5], which specializes in 2D hand pose estimation. MediaPipe outputs a set of keypoints in image space, representing specific features on the hand, such as fingertips and joints. To determine the spatial location of the hand in 3D space, each keypoint's depth was measured using depth data

from the camera's built-in depth sensor. This sensor provided real-world distance information for each keypoint.

For each keypoint, the system sampled the depth at its corresponding image coordinate, retrieving real-world distance data from the camera. To convert image coordinates $(u, v)$ to normalized camera coordinates $(x, y)$, the system used the camera's intrinsic parameters: the optical center $(c_x, c_y)$ and the focal lengths $(f_x, f_y)$, as shown in Equation (3.4). The optical center defines the point on the image sensor where the camera's optical axis intersects the image plane, typically near the center of the image. The focal lengths describe the distance between the lens and the image sensor along the horizontal and vertical axes, determining how image coordinates are scaled into physical space.

$$x = \frac{u - c_x}{f_x}, \quad y = \frac{v - c_y}{f_y} \tag{3.4}$$

These normalized coordinates were then scaled by the depth value $d$ (in meters), yielding the 3D position of the hand $(X_h, Y_h, Z_h)$, as shown in Equation (3.5).

$$\begin{aligned} X_h &= x \cdot d \\ Y_h &= y \cdot d \\ Z_h &= d \end{aligned} \tag{3.5}$$

This 3D point defined a translation vector that could be used to construct a homogeneous transformation matrix $\mathbf{C_H}$. Since hand orientation was not estimated, the rotation component was ignored and assumed to be the identity matrix, as shown in Equation (3.6).

$$\mathbf{C_H} = \begin{bmatrix} 1 & 0 & 0 & X_h \\ 0 & 1 & 0 & Y_h \\ 0 & 0 & 1 & Z_h \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

To express the hand's position relative to the robot's coordinate system (rather than the camera), the transformation was adjusted using the transformation matrix of the origin marker that corresponds to the robots origin. This is shown in Equation (3.7), where $\mathbf{O_H}$ is the hand coordinate relative to the origin marker, $\mathbf{C_O}^{-1}$ is the inverse of the camera-to-origin transformation, and $\mathbf{C_H}$ is the camera-to-hand transformation.

$$\mathbf{O_H} = \mathbf{C_O}^{-1} \cdot \mathbf{C_H} \tag{3.7}$$

An example output of hand tracking with respect to the origin marker is illustrated in Figure 3.65.

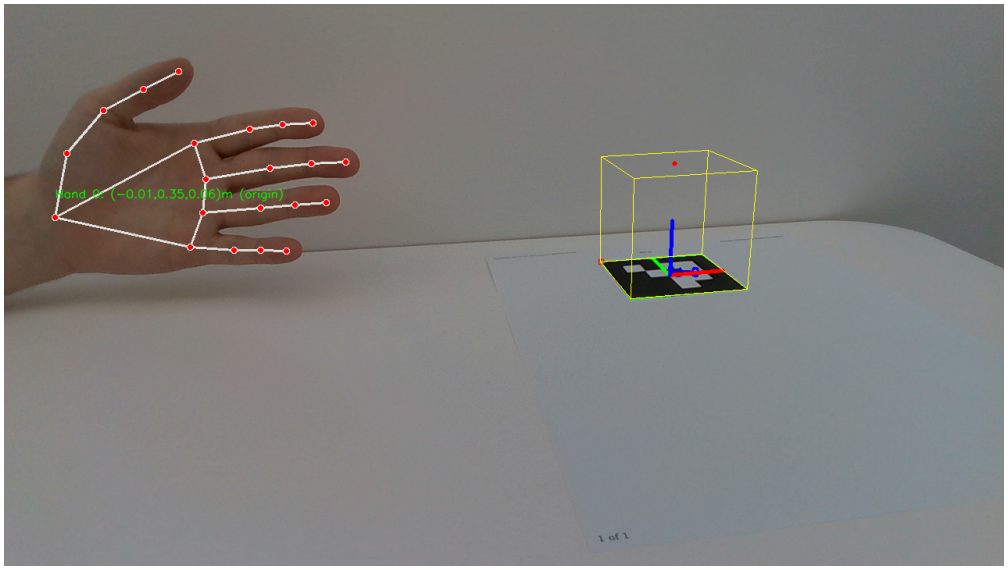


Figure 3.65: Hand detection with origin marker

Since the hand position did not provide orientation data, the robot's orientation was inferred from the hand's translation alone. The system constrained the end effector of the robot to move within a fixed constant-radius (30 cm) sphere centered where the robot's elbow joint's position would be with the manipulator in an upright orientation. This constraint ensured the robot arm always pointed toward the user rather than extending outward, as illustrated in Figure 3.66.

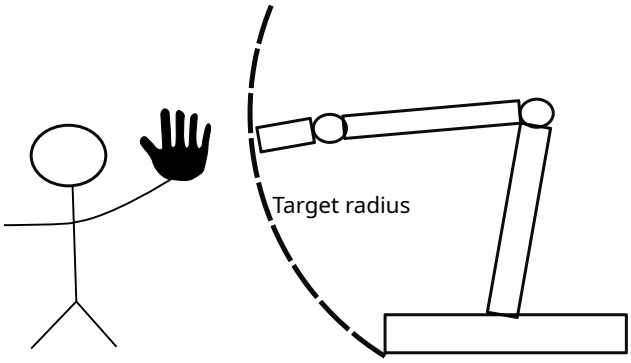



Figure 3.66: Follow hand concept

During the development of the hand detection an idea of a entertainment mode using gestures was born. The idea was to make the user play rock, paper and scissors with the robot. Hand gesture detection was implemented using the camera to pin-point the distance of each hand joint using the pixels of the image. The hand detection utilized hand features, defined by different joints of the hand, detected by the MediaPipe framework[5]. A binary state (extended or not extended) was assigned to each finger based on the computed distances between the human wrist and specific human finger features. This resulted in a list of boolean values, one per finger, from index to pinky. This represented the gesture detection. The algorithm computed three pixel distances for each finger: $D_{\text{tip}}$ (wrist to fingertip), $D_{\text{mid}}$ (wrist to middle joint), and $D_{\text{ref}}$ (wrist to base of the middle finger, used as a reference distance). A finger was considered extended if the condition for Equation 3.8 was met. Otherwise, it was considered not extended.

$$D_{\text{tip}} > 1.3 \times D_{\text{mid}} \quad \text{and} \quad D_{\text{tip}} > 0.7 \times D_{\text{ref}} \tag{3.8}$$

The resulting finger states were evaluated using the logic illustrated in Figure 3.67, which mapped different finger configurations to predefined gestures.

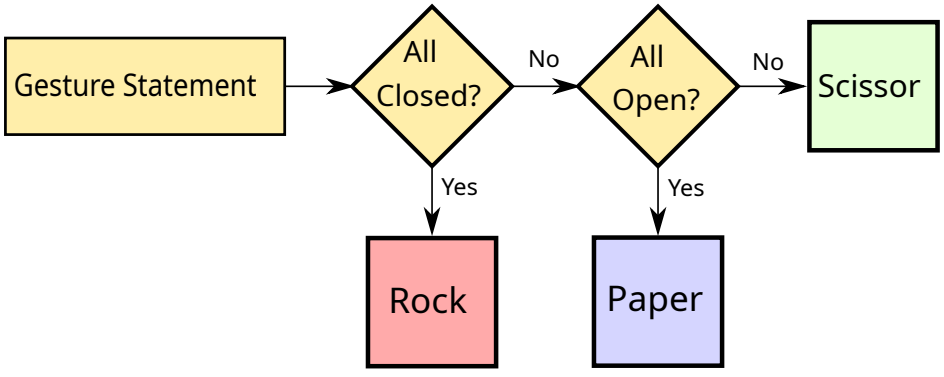



Figure 3.67: Flowchart illustrating the gesture logic

Any configuration that did not match a known gesture (e.g. only some fingers extended) defaulted to the "Scissors" gesture. This method of defaulting prevented the system from getting stuck on unknown or ambiguous states. Once a gesture was detected, the robotic arm responded accordingly by moving toward the object associated with the recognized symbol. for instance, the "Paper" gesture would trigger a movement toward a crumpled paper ball placed in the scene. An example of the gesture detection system in use is shown in Figure 3.68.

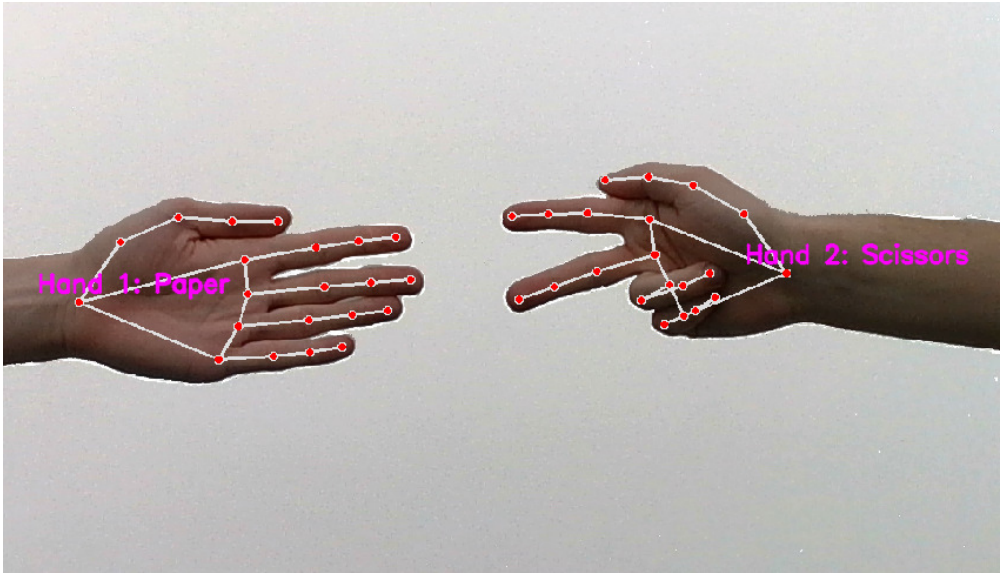



Figure 3.68: Gesture recognition system used to play Rock–Paper–Scissors

The camera software was centered around the **Vision** class, which integrated the marker detector, the hand detector and the spatial reference system. A machine vision `config` file was made to give the subclasses parameters.

The Vision class received images from the Camera class, which were then processed by the detectors to extract marker and/or hand coordinates. These coordinates were then sent to the spatial reference system for reorientation and transformations. The updated data was returned to the Vision class. The processed transformations were then saved to a JSON file for the robot to use later. An image with the detected markers and/or hands drawn on top of the original image, was returned for visual feedback. The topology of the camera software is shown in Figure 3.69.

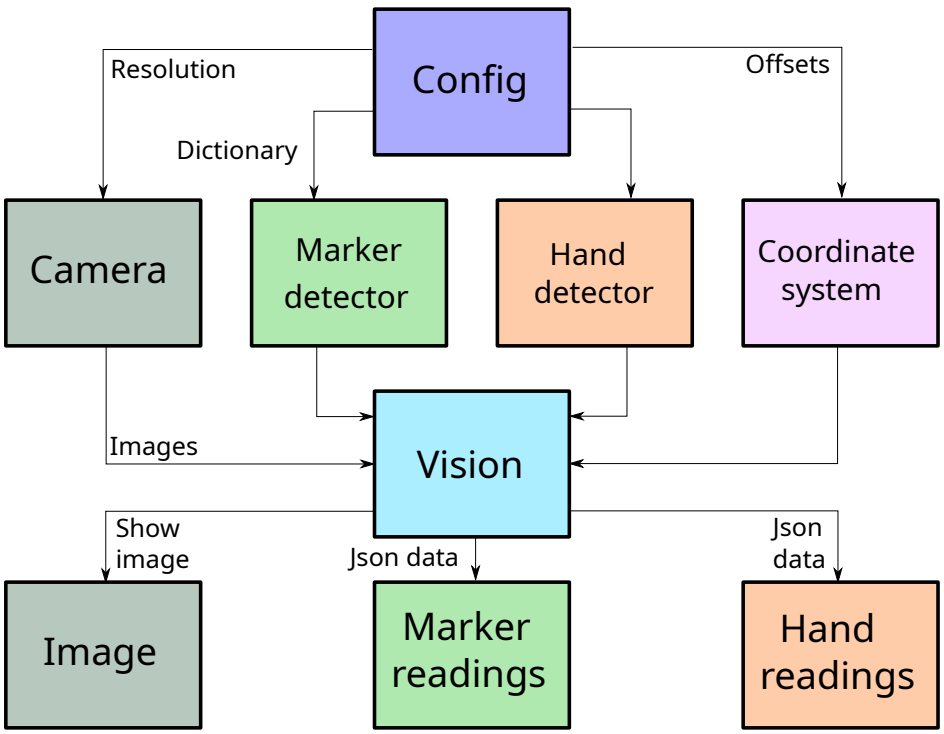



Figure 3.69: Topology of the machine vision software

### 3.3.3 Robot Movements

The robot movements were dictated by both pre-recorded data and runtime adjustments. In the preparation stage, a program was written that allowed the developer to record pose data based on positions the robot arm was moved to manually while the motors were disabled. At runtime, the robot would replay recorded positions in one of two ways based on its selected mode.

The robot control had two modes: Manual mode completed a set of predetermined positions to ensure repeatability. Automatic mode used the machine vision system to adjust its movements based on its environment. Both modes also featured dynamic collision detection/avoidance strategies of tracked objects which were able to be toggled on or off according to the operator needs. Manual mode and automatic mode each had their own distinct implementations, and are therefore covered separately when discussing each mode.

Once the movement concepts have been established, the underlying technical implementation choices that were made to realize the design and handle edge cases will be addressed.

#### Pose Recording

To move between positions, the robot needed a way to know which positions to move to. To achieve this, it was decided to use a system of recording poses by manually moving the robot. Each pose was described by the joint states the robot had at the time of recording, and the resulting end effector pose matrix was also stored. These poses could then be played back when the waffle making program was executed.

To account for movable objects, the pose recording software needed to allow relative movement. When a pose was recorded, the camera simultaneously recorded the position of a specified marker using the method described in Equation (3.2). The position of the marker could be combined with the pose matrix of the arm to generate an offset from the marker to the arm as shown in Figure 3.70. The offset of the arm was found using Equation (3.9).

$$\mathbf{O} = \mathbf{M}^{-1}\mathbf{A} \tag{3.9}$$

Where $\mathbf{O}$ is the offset matrix, $\mathbf{M}$ is the pose matrix of the marker and $\mathbf{A}$ is the position of the arm at time of recording. Equation (3.10) could be used to return the arm to the same relative position,

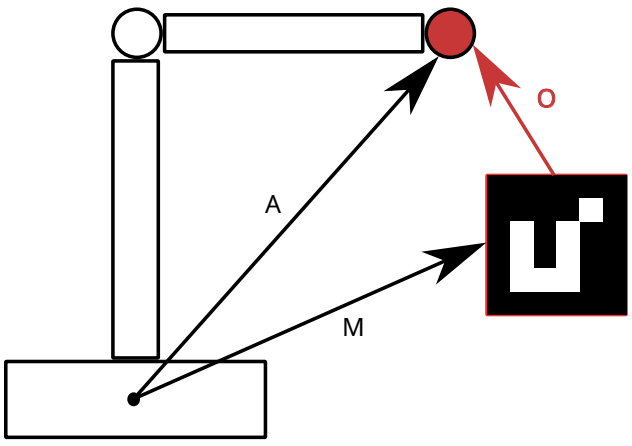



Figure 3.70: Concept behind offset recordings

even if the reference marker was moved.

$$\mathbf{A} = \mathbf{MO} \tag{3.10}$$

To implement pose recording, a program was made that allowed the user to move the arm around with the motors powered off. On a button press, the position of the joints was recorded along with

the data shown in Table 3.1. This data was then named and stored in a JSON format to be retrieved later.

| Name | description |
|---|---|
| Name | Name of the position |
| Basepose | The end effector pose matrix of the arm, determined by forward kinematics |
| Joints | The joint states at time of recording |
| Offset | Transformation from a reference marker to the arms pose |
| Marker | The marker id used to determine offset |

Table 3.1: Parameters stored during pose recording

Using the pre-recorded data, the robot movements could be defined in two different ways. When using manual mode, the joint data was simply read back to the program, causing the robot to move to the same position the recording was taken in. Since it replayed joint positions, it was impervious to inverse kinematics problems like not being able to find a valid solution for a pose and finding multiple configurations with the same result. In dynamic mode, the ”Marker” and ”Offset” values were used to reconstruct the pose of the robot arm, as recorded at that specific point, relative to the reference marker. The pose is given by Equation (3.10). To obtain $\mathbf{M}$, the recorded marker ID was used as a lookup ID for all the marker matrices that were retrieved by the vision software.

Some operations needed to follow a specific path from A to B. An example of this need occurred when the waffle iron was opened. The opening movement required that the arm follow a curved path. A linear movement from the bottom to the top would not allow the hinge of the waffle iron to rotate. To perform path recording, the pose recording software was expanded with the option to record trajectories. Recording trajectories was done by recording multiple consecutive positions, with a time delay ($\leq 0.1$ s)[1] between each recording. The positions were stored separately. Their names were followed by a numbered index (e.g. `recording_309`) such that positions could be read back to the program in order by looking up consecutive indices.

### Manual Mode

When using manual mode, the language of robot movement was joint state transitions. Each movement was performed by setting the joint state of the robot to a set of predetermined positions. The simplistic movements used in manual mode did not allow for collision avoidance. Instead, a collision detection algorithm, was used to alert the operator before a collision-triggering movement was executed. The operator would then have time to clear away colliding objects before continuing, or they may choose to abort the movement altogether.

Collision detection was applied by simulating the robot's movement before execution and comparing it to the position of tracked objects in the scene. First, the position of trackable collision objects could be updated by using the camera to detect their associated ArUco markers. Then, the robot simulated its movement to generate its own collision hitbox. The movement was represented by linearly interpolating from the start joint states to the target joint states. The position of the each part of the robot was then obtained by using forward kinematics. For each of these parts, bounding boxes were generated around the arm based on hand measurements for the dimensions of every link. The generated boxes were divided into sections as shown in Figure 3.71. These boxes were then tested against the bounding boxes of the tracked collision objects in the scene. If a collision was found along the path, the course of action was left for the operator to decide. An illustration of this process is presented in Figure 3.73.

Note that some longer joints were split into smaller subdivisions. This was done to account for the boxes being defined as axis-aligned to simplify collision calculations. By making the bounding boxes cover less distance per box, the error from the true position of the arm was not allowed to compound, as is shown in Figure 3.72.

[1]The time step was varied during recordings based on perceived playback results

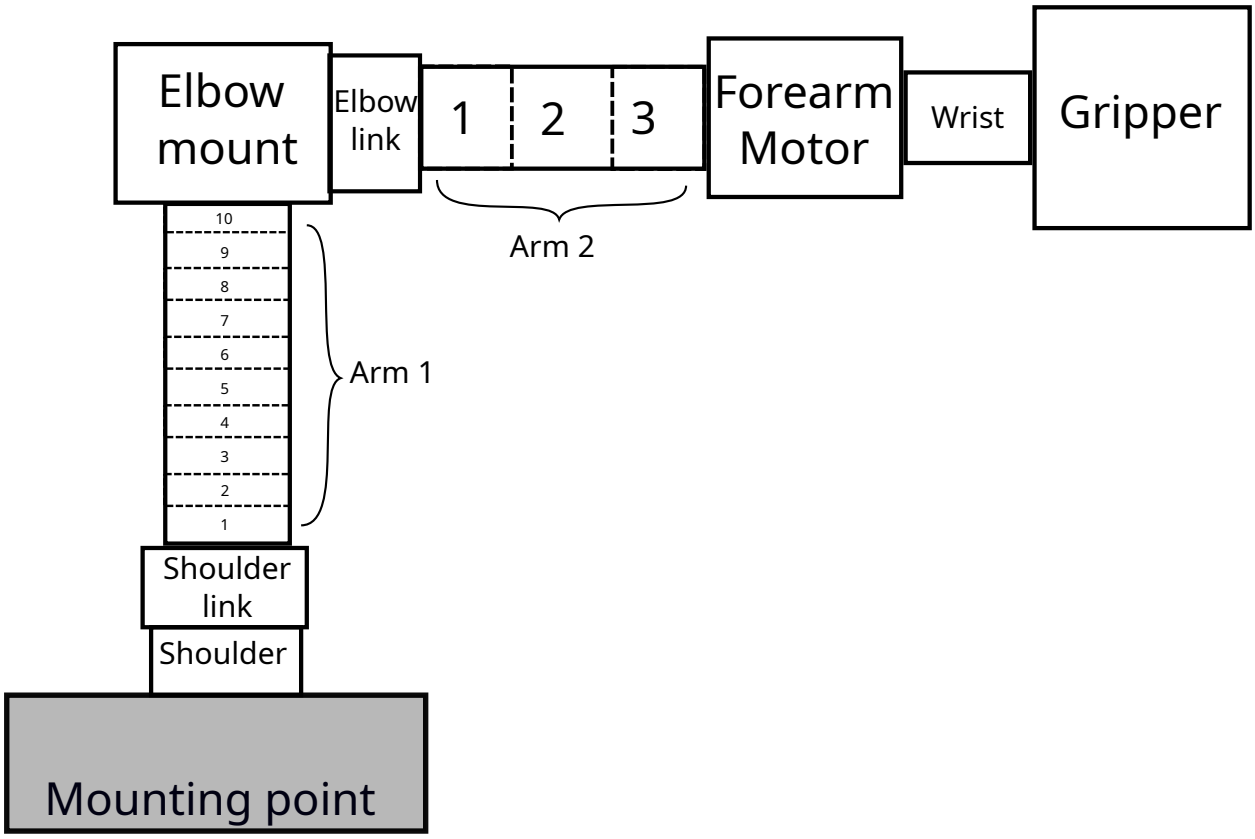
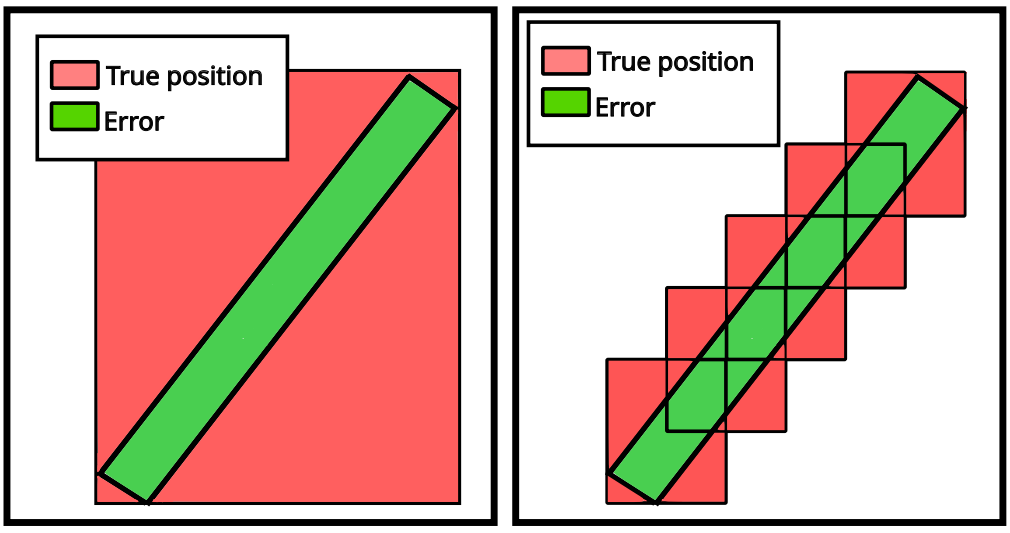


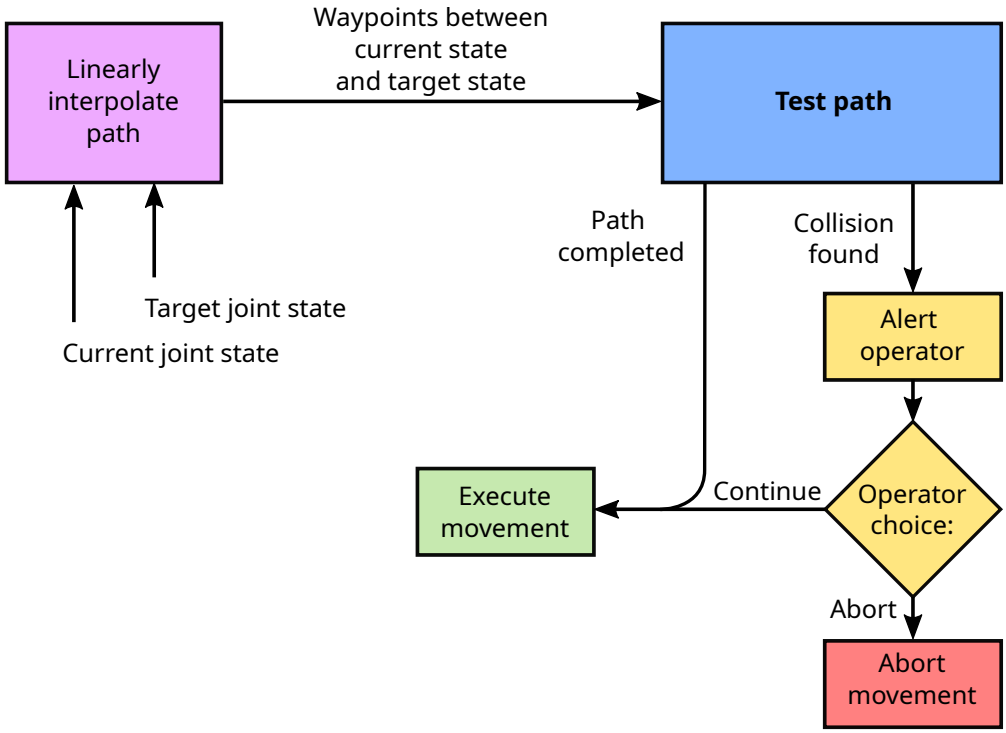

Figure 3.71: Subdivisions of the arm used for manual collision detection



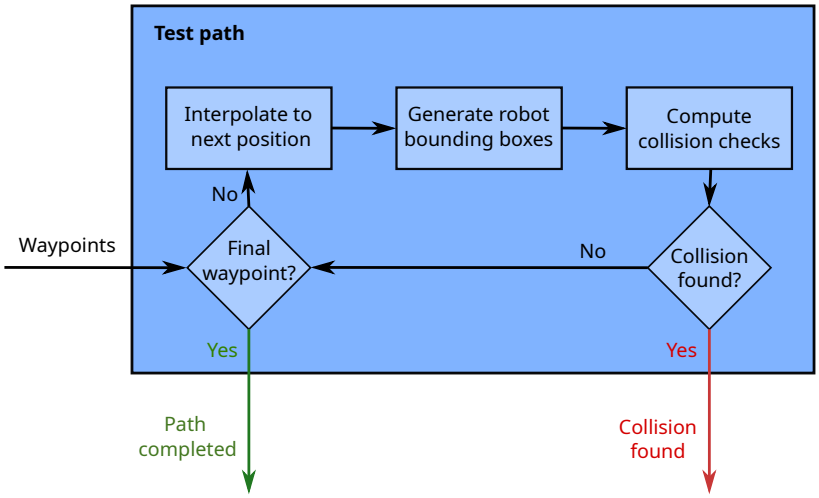

(a) Single box (b) Multiple boxes

Figure 3.72: Comparison of errors in bounding box representations using one box, versus using five boxes.

As Figure 3.73 shows, the primary job of this part of the program was to simulate the path and test for collisions. The execution of the movement was made trivial by the Interbotix-ROS API, enabling the movement with a single `set_joint_positions()` command.




(a) Main control flow




(b) Path test subroutine

Figure 3.73: Motion planning for manual mode

### Automatic Mode

The underlying software that was used to avoid collisions was **MoveIt**. MoveIt path planners typically support collision-free trajectory planning, using an internal map of collision objects. To use collision-free motion planning, MoveIt must be given information on the whereabouts of collision objects in the workspace.

By using the positions of all detected ArUco markers, MoveIt's model of the workspace could be dynamically updated. Positions were updated from the marker pose matrix of each tracked marker based on the object they represented. The bounding boxes that were generated based on the markers were then sent to MoveIt before beginning path planning of any movement.

When handling collisions, a unique compatibility challenge arose. The typical Python interface for interfacing with MoveIt, the library called "moveit_commander", had not yet been ported from ROS 1 at the time of writing[22]. The project was built primarily in Python for compatibility with the

Interbotix interface, so workarounds were used to interact with MoveIt. For most applications, a ROS node built in Python could be used to interact with the services directly, as was the case in the example shown in Figure 3.75. Unfortunately, adding collision objects with this approach proved unsuccessful. To access the relevant interface a separate application was made in `C++`, which had a more mature ROS 2 library environment. The structure of the programs is shown in Figure 3.74. The `C++`

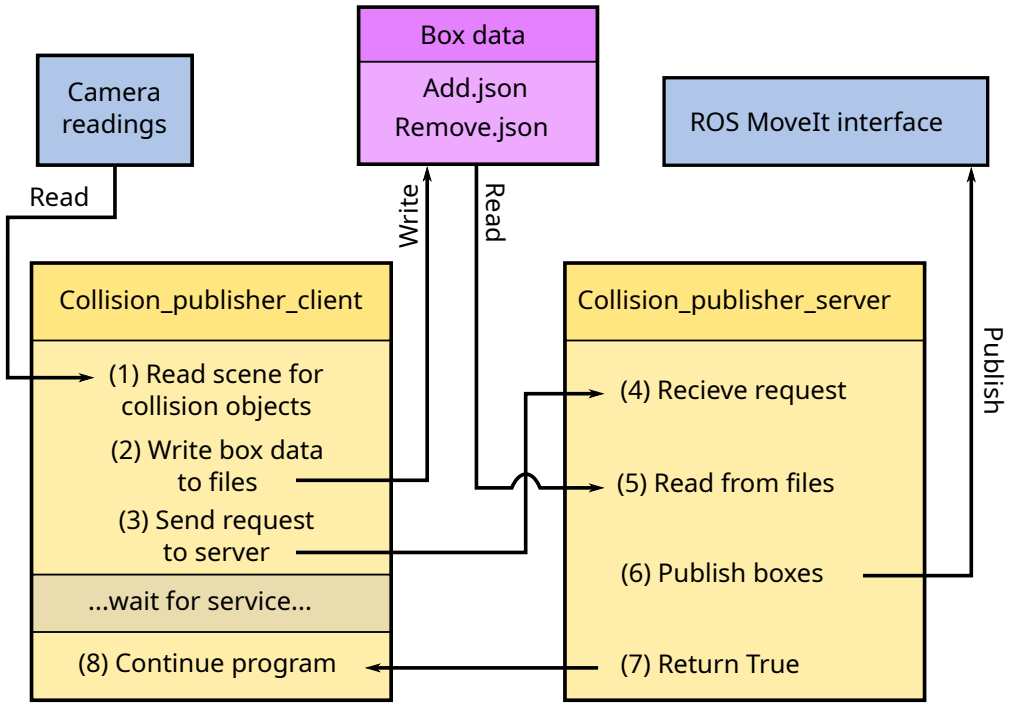



Figure 3.74: Communication structure between the main MoveIt interface (client) and the C++ extension (server)

program was made to perform the single task of publishing collision objects. When the main program was initialized, it launched the `C++` program which created a `Collision_publisher_server` node. On movement, the main program loaded collision data to a pair of files. One file represented collision objects to be added/moved and the other contained objects to be removed/ignored. Then, the main program sent a trigger message from the `Collision_publisher_client` node to the `C++` `Collision_publisher_server` node. When triggered, the collision program read the files with box data, transformed it to the proper format and published them to the relevant ROS endpoint using the `C++` MoveIt interface library. Once the collision data was loaded into MoveIt, the program was ready to execute the movement. Executing the movement was done by sending requests to the `Move_group` node. The interface used to execute a movement with MoveIt is shown in Figure 3.75.

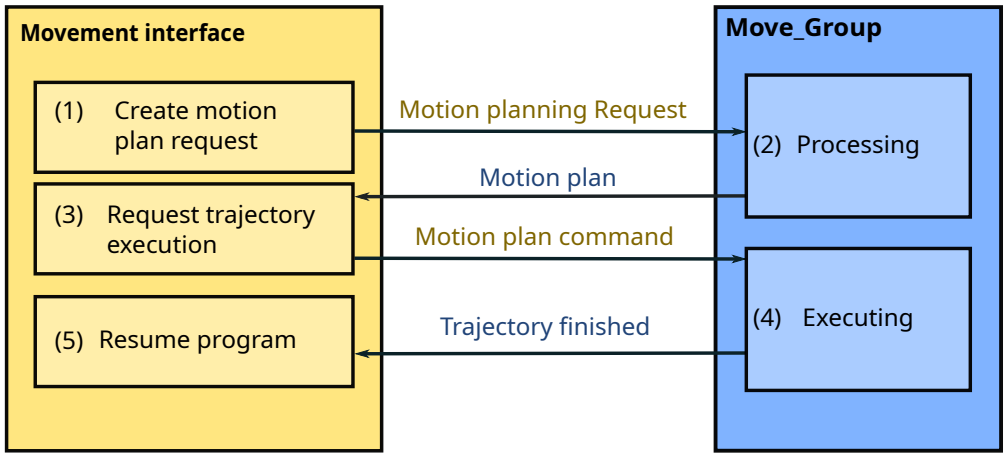



Figure 3.75: Illustration of the motion planning process

**Collision Checking Specifications**

The bounding boxes that were used to perform collision tests needed to be stored. Each bounding box could be determined uniquely with two points $P_{min}$ and $P_{max}$ located at each outer corner of the box, as defined in equation (3.11).

$$P_{min} = \begin{bmatrix} x_{min} \\ y_{min} \\ z_{min} \end{bmatrix} \qquad P_{max} = \begin{bmatrix} x_{max} \\ y_{max} \\ z_{max} \end{bmatrix} \tag{3.11}$$

By representing the boxes as a set of minimum and maximum points, the data was easy to extract to compute collision checks. The bounding boxes could be reconstructed with the min/max points, defined as the volume bounded by the corners listed in (3.12):

$$\begin{bmatrix} x_{min} \\ y_{min} \\ z_{min} \end{bmatrix} \quad \begin{bmatrix} x_{min} \\ y_{min} \\ z_{max} \end{bmatrix} \quad \begin{bmatrix} x_{min} \\ y_{max} \\ z_{min} \end{bmatrix} \quad \begin{bmatrix} x_{min} \\ y_{max} \\ z_{max} \end{bmatrix} \\ \begin{bmatrix} x_{max} \\ y_{min} \\ z_{min} \end{bmatrix} \quad \begin{bmatrix} x_{max} \\ y_{min} \\ z_{max} \end{bmatrix} \quad \begin{bmatrix} x_{max} \\ y_{max} \\ z_{min} \end{bmatrix} \quad \begin{bmatrix} x_{max} \\ y_{max} \\ z_{max} \end{bmatrix} \tag{3.12}$$

When using MoveIt, another standard was used to represent a box. This standard represented boxes with the cartesian coordinates $(X, Y, Z)$ of its center-point and its length $(L)$, width $(W)$ and height $(H)$. To compute between the standards, the conversions in Equations (3.13) and (3.14) were used.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = P_{min} + \frac{1}{2} P_{max} \tag{3.13}$$

$$\begin{bmatrix} L \\ W \\ H \end{bmatrix} = P_{max} - P_{min} \tag{3.14}$$

For manual mode collision checking, two bounding boxes knew when they have collided because they knew when they have not collided. While direct collision testing is possible, an easier alternative was to check based on when a collision does not occur. If no non-collisions have been computed, the result must be a collision. Since the points were, by definition, sorted from smallest to largest, testing for a non-collision was trivial. For instance, if the maximum point of a box $B_1$ has a lesser value than the minimum value of box $B_2$ along a given axis, then there is no point along $B_1$ that can intersect with $B_2$. Similarly, if the the minimum point of $B_1$ is larger than the maximum point of $B_2$ then that is not an intersection. These conditions are shown in (3.15) and (3.16). If any of the statements are true for a given axis, then no collision can occur along that axis.

$$B_1 max < B_2 min \tag{3.15}$$

$$B_1 min > B_2 max \tag{3.16}$$

If a non-collision has been found in any axis, then no collision could occur for the object. For a 3D object that means that a collision only occurred if a collision was found in the $X$ axis, $Y$ axis and the $Z$ axis. In Figure 3.76, box $A$ does not intersect with box $D$ in any axis. Box $B$ intersects with box $D$ in the $X$ axis and not in the $Y$ axis. And box $C$ intersects box $D$ in the $X$ and $Y$ axis. Only box $C$ intersects box $D$ overall, since each individual axis intersects.

The collision logic used to test for collision between two boxes is shown in Figure 3.77. Each collision was tested using the relevant axis of equations (3.15) and (3.16).

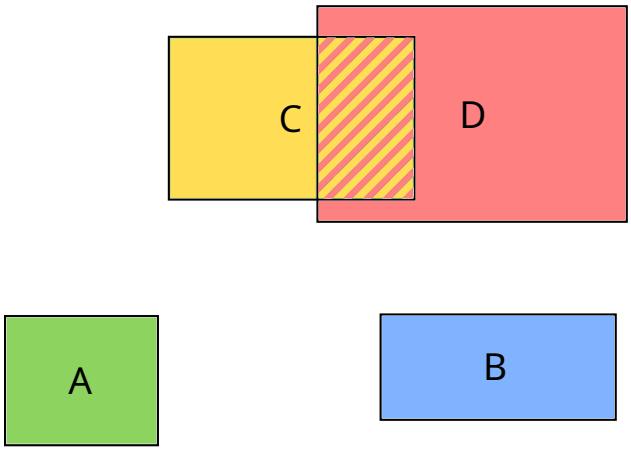



Figure 3.76: Boxes showing intersection test examples

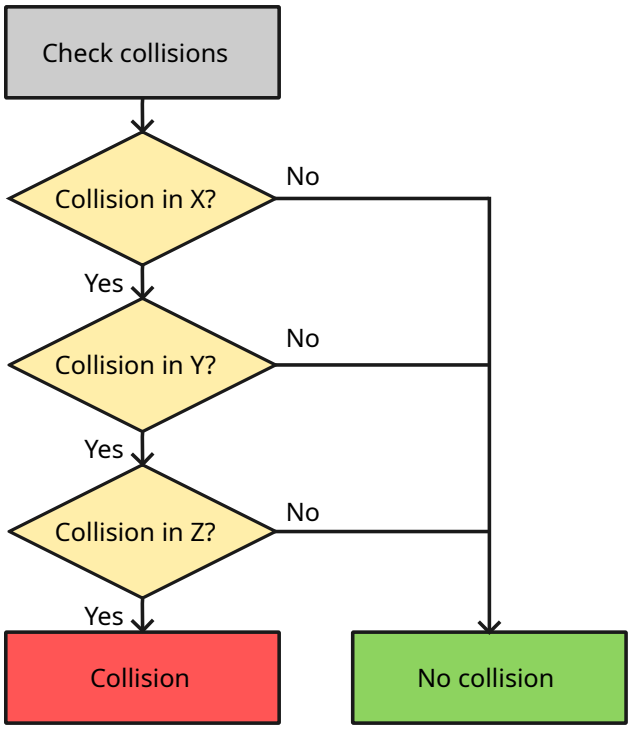



Figure 3.77: Bounding box intersection tests logic

### Working with Joint Limits

When using a robotic arm with multiple degrees of freedom, there is usually a set of end effector poses that can be reached with multiple distinct joint states. For example, the pose shown in Figure 3.78 can be reached with either of the two configurations that are drawn.

Both these positions are valid solutions, but that does not mean that they are necessarily interchangeable. For instance, if the arm was commanded to move to the same position repeatedly then the arm should not alternate between those two solutions. That would cause unnecessary back and forth movements of the joints without changing the resulting end effector pose. When applied to the ViperX 300S arm[24], these problems were solved by artificially setting stricter joint limits. Specifically, this was implemented by modifying the `min_position` and `max_position` parameters in the MoveIt `joint_limits.yaml` file.

One case of the robot reaching the same pose with two different configurations as shown in Figure 3.78 occurred with the gripper. Both the forearm and wrist had the freedom to move $\pm\pi$ rad each[25], which in sum gave them the ability to move a full $2\pi$ rad back and forth between poses.

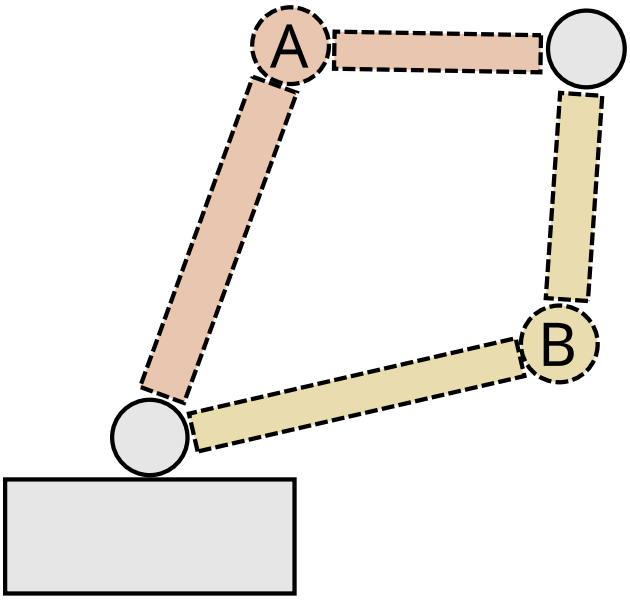



Figure 3.78: Two joint configurations that reach the same end effector pose

To remove this possibility, each joint was set to a maximum of $\pm 1.55$ rad. In sum, the two joints have a reach of 3.1[rad] or slightly less than $\pi$ rad. A value of exactly $\pi$ would still barely allow a full rotation, and therefore a small margin was added.

Another problem that arose with the generous joint limits of the arm was found when the shoulder joint was used. Since it had the ability to rotate $\pm\pi/2$ rad or more , there were poses that could be reached in reverse, as shown in Figure 3.79.

Figure 3.79: Alternate home pose configuration

This was an undesirable state as it increases the total moment felt by the arm. See Appendix A for calculations demonstrating the increased moment. More importantly, this set of poses also tended to cause large movements when poses were being transitioned between. Most poses could not be reached with this alternate movement approach, and as such the transition between this pose and its temporal neighbors were likely to cause large sudden movements. To disallow this type of movement, large backwards rotations of the shoulder was banned. The limit imposed was a minimum shoulder joint position of $-\pi/4$ rad (a negative position in this case corresponds to a "backwards" movement from the upright position). The robot typically did not need to move the

shoulder backwards significantly to reach any pose, with one notable exception: The `sleep` pose.

Since MoveIt did not allow movement to the sleep pose with the given joint limits, another workaround was used. The Interbotix API did not consider MoveIt's joint limits, and commands sent through the API were therefore free to move past the limits that were imposed. The normal approach of using the built-in `go_to_home_pose()` and `go_to_sleep_pose()` methods did not pair well with MoveIt. The MoveIt configuration disabled the speed control system used internally by the Interbotix API, causing the arm to move dangerously fast. A simple trick was used to slow the arm down. By linearly interpolating between the current pose and the home/sleep pose in small steps and adding a brief delay between each pose, a slower movement was emulated. The movement commands were continuously overwritten before the interpolated movement steps finished, ensuring the arm did not stop moving before reaching its destination.

# Chapter 4

# Results and Discussion

This chapter discusses the results of the finished system. It begins by outlining performance characteristic. A demonstration of the robot in its intended environment is documented, followed by general observations about how the system succeeded and failed. After establishing core performance characteristics, the complications of navigating camera software were discussed. Similar discussions follow for mechanical design and robot control. Finally, the chapter ends with a review of the societal impact of the technology.

## 4.1 Field Testing and Observations

To get realistic data on how the system would perform in action, a stand to showcase the product was held in the cafeteria of the University of Agder. A picture of the stand is shown in Figure 4.1. The stand was a success, both in terms of technical performance and marketing potential. On


Figure 4.1: Waffle stand in action

the technical side, the robot achieved a 100% success rate while making a total 16 waffles[1]. On the marketing side, the stand drew a crowd estimated at 15 people looking onto the robot during its startup phase. The robot continued to draw a crowd during its first two hours of operation, averaging at approximately 5 concurrent onlookers. Towards the end of the waffle making process, interest dwindled as people grew accustomed to the robot in their presence. One 16 pack of waffles lasted long enough to keep the system running for as long as there was significant interest in the system.

[1]The total batter used was one 16-pack of waffles, and slightly more that had been stored after the previous day of testing. The robot cooked every waffle until the batter no longer filled up the ladle unassisted, putting the pouring system in a unpredictable state. At that point, a human cooked the final few waffles by hand.

Onlookers were primarily interested in the parts of the process where the robot prepared the waffle for cooking, and the final serving of the waffles. Most observers left the stand once the waffles began cooking, in the transition time between waffle making and entertainment modes. The entertainment mode that was used in the field test was the hand tracking concept. Due to how the software was configured in the days leading up to the field test, the arm was set up to track a marker instead of using hand tracking. Once the entertainment mode was running, it did not raise interest in the stand significantly. The HMI was shown to a few interested onlookers after the main test. They gave positive feedback on the HMI concept.

The finished system could semi-automatically create waffles in manual mode, with some limitations. The arm could open and close the waffle iron consistently. Picking up the lubrication spray was also consistent, but the servo motor did not run due to a noisy signal. The movement of the ladle poured two waffles worth of batter, split between the two plates of the iron. The batter was not evenly distributed. The left side of the waffle iron overflowed, resulting in spillage of the batter. Meanwhile, the right side of the iron barely had enough for a single waffle. Overall spillage outside of the main process was present, but not excessive. The rod inserts ensured no waffles were lifted when the waffle iron was opened. The success of the inserted rods varied based on waffle iron lubrication and cooking time. In general, they were reliable if the waffles were cooked "well done" in a lubricated iron. The pole system that was used to remove waffles from the rod inserts were too flimsy to remove the waffles. Due to their thin size, they occasionally managed to make cuts into the waffles before being pushed aside by the waffles.

Automatic mode was tested with partial success. Compared to equivalent movements in manual mode, automatic mode exhibited a higher failure rate. Tests conducted on a per-movement basis demonstrated a greater degree of repeatability than full-sequence system tests, likely due to increased care taken during pose recordings, particularly for tasks such as picking up the lubrication spray. In some cases, the gripper's position relative to the spray was insufficiently accurate to allow a successful grasp.

Small-scale tests involving camera-based detection of ArUco markers showed promising results, enabling the robot to react to and grasp objects placed in arbitrary positions. However, during the task of opening and closing the iron, the gripper failed to hold the interface cube of the handle long enough to complete the motion successfully.

A notable limitation of the automatic mode is its reliance on visual detection of ArUco markers. If a marker is rotated out of the camera's view, the system becomes incapable of recognizing and retrieving the corresponding object. Large-scale testing yielded less favorable results overall, with the most successful test at the time of writing achieving only partial success. This test failed at the final stage, during the removal of the waffle from the waffle iron.

Fun modes and screen interaction was coded, but not properly integrated into the final product.

## 4.2 Mechanical Design

All mechanical components functioned as intended, and the system successfully completed the task of making waffles. However, due to the number of interacting variables, certain limitations were identified during testing.

The ladle required precise placement for successful grasping by the robotic arm. When correctly positioned, the ladle was picked up reliably, aided by the interface cube.

The waffle iron mount performed as expected, securely holding the appliance in place. A notable issue arose when the iron was overfilled, resulting in batter seaping under the mount in most cases. This was mitigated by placing paper beneath the iron to contain the spillage. If the spillage was more than the paper could absorb, the iron mount had to be unscrewed and washed or maybe replaced if the mount could not be properly cleaned. The interface cube allowed for effective rotational movement of opening and closing the iron.

The adapter for the handle presented a mechanical interference with the grippers. During the iron-closing motion, the bolts on the adapter from the handle blocked the grippers from closing entirely, causing the operation to fail. This issue was resolved in software by aiming the grippers higher during the grasping of the interface cube, enabling the grippers to bypass the obstruction. This was also partially due to the reach of the manipulator. The arm was fully stretched out when grabbing the iron when the iron was open, reducing options for movements.

The housing structure remained stable throughout testing. During preliminary system tests using water, which was used as a substitute for batter, some spillage occurred. However, the integrated vents effectively prevented water from entering into the housing. However, the addition of a wax sheet made attaching the housing more challenging due to the sheet wrinkling.

The rods, which were inserted into the waffle iron before cooking, enabled reliable waffle pickup after baking. During cooking, the rods became embedded in the waffles. This allowed the system to extract the finished waffle by lifting the rods with the manipulator. The solution worked effectively when the iron was properly lubricated and the waffles were thoroughly cooked. In some cases, waffles adhered too strongly to the iron, leading to partial detachment during extraction. Furthermore, the delivery of the waffle would sometimes not succeed because the waffle adhered too well to the rods. Despite this, the method proved to be a viable solution overall.

## 4.3 Camera Software

This section presents the results and insights gained from implementing marker-based perception in the robotic system. It includes an evaluation of ArUco marker pose estimation accuracy, challenges encountered during camera integration, a comparison of marker detection libraries, and an assessment of methods considered for detecting waffle readiness.

### 4.3.1 Marker Pose Estimation Accuracy

During testing of automatic mode, the accuracy of the fiducial markers was evaluated over 10 to 20 trials per object. The evaluation criteria was whether the robot successfully grasped the object and continued the task sequence without triggering a fault.

Although the measurements were only approximate, the observed positional error ranged from 0.5cm to 1cm. Which was within acceptable limits for the manipulation tasks. The measured positional error may be due to the tolerance settings of the pose goal constraints in the robots control software, rather than inaccuracies in the ArUco marker pose estimation. This reasoning is based on the fact that ArUco pose estimation should, in theory, have near millimeter accuracy. Also a RealSense camera was used, which was factory calibrated with near zero distortion. Additionally, the workspace lighting was excellent during the testing, and several optimizations were applied to the 2D pixel coordinates of the markers. These included sub-pixel refinement, a technique that increases accuracy by estimating marker corners between pixel values, using the built-in parameters of the ArUco library in Python.

The marker used to detect the state of the waffle iron consistently functioned as intended. When the iron was open at inappropriate times, the robot transitioned to a corrective state in the state machine. The marker placement is shown in Figure 3.33.

Tool pickup markers also performed reliably, with the robot achieving an approximate 75% success rate in grasping both the spray bottle and the ladle. Failures generally occurred due to slight misalignments during the grasping process. While the robot lacked direct feedback to confirm successful grasping, marker visibility served as an indirect indicator.

In the case of the spray bottle, the marker was expected to be occluded by the gripper upon successful pickup. If the marker remained visible, this triggered a state inconsistency. For the ladle, no occlusion-based success condition was available, as the marker was placed on the ladle guide

rather than on the ladle itself. This occasionally led the robot to proceed with the batter-pouring task despite an unsuccessful grasp.

In a few isolated instances, the system unexpectedly transitioned between states. Since ArUco marker visibility drives the state machine logic, these transitions were likely caused by temporary detection failures lasting a few milliseconds.

### 4.3.2 Challenges in Camera Software Integration

The project initially began with the Intel RealSense L515 depth camera. Although it offered acceptable performance, the L515 had reached end-of-life (EOL) status and required legacy versions of both firmware and SDK. This posed significant challenges on the ARM-based Jetson platform, as the official APT repository supported only the latest version of the SDK and x86 architecture. Consequently, the SDK had to be built from source, leading to a time-consuming setup process and underlining the importance of hardware/software compatibility.

To resolve these issues, the L515 was replaced by the Intel RealSense D435i. This model supported the latest SDK and allowed seamless integration with the official RealSense ROS package under ROS2 Humble. The D435i thus provided a more stable and future-proof solution during the development phase.

However, the project's software evolved over time. A fully OpenCV-based machine vision pipeline was adopted, eliminating the need for the ROS2 RealSense integration. As a result, SDK version requirements became irrelevant in the final implementation.

### 4.3.3 Evaluating Marker Detection Libraries

The initial objective for the machine vision system was to enable real-time performance. To achieve this, the use of hardware acceleration via the GPU on the Jetson platform was explored. Fortunately, a ROS2 package from Isaac ROS that supports marker detection with GPU acceleration was identified[17]. This package employed AprilTag fiducial markers and published their pose to a ROS node.

However, the system experienced stuttering, resulting in inconsistent pose estimation. This issue likely stemmed from the overhead associated with ROS2's message handling between nodes, which introduced latency and jitter. If frames were not perfectly aligned, additional delays could occur. The detection process itself was computationally demanding, and when combined with the runtime overhead of ROS2, it could lead to periodic spikes in CPU or GPU usage.

As an alternative, OpenCV offered a lightweight, single-threaded ArUco marker detection system that operated with negligible lag and in real time. ArUco markers were designed for simplicity and ease of integration, typically requiring only a single Python script to function. Unlike ROS2 based systems, this method did not require multiple nodes or inter-process communication; pose data could be transmitted directly to the system using JSON or other structured data formats. Furthermore, ArUco detection could have direct access to camera frames and employed an efficient pose estimation algorithm. In practice, ArUco markers provided pose accuracy comparable to that of ROS2 AprilTag detection, despite their simpler architecture.

### 4.3.4 Detecting Waffle Readiness

Multiple approaches were considered to dynamically detect when a waffle was ready. The first approach that was considered, would use AI trained on pictures of ready waffles to detect if the waffle was finished, when the iron was opened. Another approach was to use machine vision to detect the color of the indicator light on top of the waffle iron. The final way that was considered for dynamic detection was to read the signal directly from the temperature sensor in the waffle iron. All of these ideas were proven to be unhelpful during system testing. Since the robot pours batter in a consistent way, using a time delay would consistently create a fully cooked waffle, without the need for dynamic sensors.

## 4.4 Robot Control

The software used to control the robot had a wide variety of features, and there were hundreds of decisions that went into creating it.[2] Some of these decisions ended up having negative side effects. This section highlights some of the major design decisions that were made, and goes into detail about the trade-offs of each choice.

The Interbotix Python-ROS API was used as the core of the project, with MoveIt being used for its collision-free path planning features. Using MoveIt came at a cost, as enabling MoveIt disabled some core Interbotix features. Specifically, the velocity controller and any functions relying on the joint position subscriber were found to be unusable when MoveIt was enabled. Due to these shortcomings, MoveIt was only used when advanced path planning was needed. For tasks needing simpler movement, e.g. the pose recording software, MoveIt was disabled.

### 4.4.1 Hardware Limitations

The computing power of the microcomputer was taken into consideration during the project. As the microcomputer was used for a wide variety of tasks from AI recognition and marker detection to path planning, there was a worry that the computing power would be a significant bottleneck. A slight increase in program startup time on the Jetson computer compared to the computers used for development was noticed based on subjective impressions. During testing, informal performance benchmarks were performed by recording the time the program spent computing the movement. These benchmarks did show a slight decline in runtime performance compared to development machines. The difference was not significant, and the Jetson did not show any signs of failing to meet computation demands. The Jetson Orin Nano Super Developer Kit can therefore confidently be used to comply with the demands of this application.
The pose recording software consistently recorded joint states that were slightly inaccurate. The inaccuracy was estimated to be approximately 1.5 cm movement between perceived position at the time of recording and resulting pose when the joint state was replayed, as illustrated in Figure 4.2. This recording error appeared to be based on inaccuracy in the shoulder joint, but no rigorous testing was performed to verify this observation. It should also be mentioned that the error appeared to be a systematic error, in the sense that the same position would give the same (incorrect) recording result if repeated.

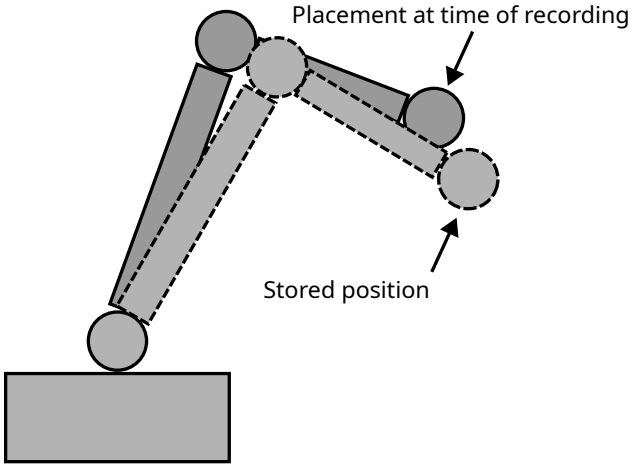



Figure 4.2: Recording offset from an arbitrary pose

The error that was introduced into the recordings is believed to be caused by a discrepancy between how poses are recorded and how movements are executed. The pose recording software relied on data polled from the encoders on the motors. When the motor was de-torqued the encoders are

[2]Did you know: We spent 2 hours arguing over JSON vs CSV storage. We ended up using a combination of JSON... and YAML.

believed to have a larger margin of error. When the movement was played back, the servo used its full accuracy to play back the movement. Regardless of the exact cause, the error margin seems to be introduced during recording. Replayed movements maintained a high degree of repeatability.

To overcome the inaccuracies that were introduced by the recording principle, the pose recording software was modified to play back the movement before confirming a recording. While this measure does not modify the margin of the error, it gives the recording operator feedback on whether or not the recording played out as expected. This extra step made it possible to redo recordings until a satisfactory resulting pose was achieved.

### 4.4.2 Automatic Mode

Automatic mode went through less testing than manual mode. Manual mode was therefore less likely to encounter unexpected errors compared to automatic mode. Recording in automatic mode introduced the marker recording as a variable, and therefore the recording margin of error was inherently larger. Despite these shortcomings, automatic mode was surprisingly consistent at following objects. Tests of individual movements showed the arm accurately following moving objects. Failure of automatic movements tended to occur only when the movement required high precision, or when the camera could not see the marker.

There were also an inherent risk of failing movements when using automatic mode. For instance, the object may be moved such that the arm attempts to move to an impossible pose. There were also an issue of the robot being limited in movement by its joints. This physical constraint means that there are cases where small changes in position may cause a large change in joint states and therefore a large unpredictable movement. While the latter were not a direct failure to move, these kinds of movements represent a failure to operate safely.

Automatic tracking of objects was an overall success, but due to time constraints it was not rigorously tested. Manual mode was deemed the go-to option for consistent waffle making, but automatic mode showed great promise. Given some further development and testing, automatic mode had great potential to surpass the usefulness of manual mode.

### 4.4.3 Collision Detection

Collision detection seemed like a natural feature to implement during development due to the dynamic nature of the system. By not colliding with any object that was placed in the planned path, the robot would be at less risk of damaging itself or other objects. In doing so it would lessen the risk of causing a critical error. Collision-free movement would also have a marketing effect, as the robot movements would be perceived as more sophisticated. All of these effects are true, but during development and testing there were several problems that arose.

First of all, the chances of encountering an unpredictable collision were smaller than what was expected before testing. During testing it was hard to create a situation where the arm would consistently experience a collision while successfully moving from A to B. The setups that were most successful at forcing a collision were achieved by placing the test collision zone close to the middle of the workspace, such that the arm extending from the elbow and out would have to pass through it when the waist moves. The movement is illustrated in Figure 4.3. The collision forcing movement in Figure 4.3 relies on the collision object being placed near the robot, such that the elbow joint with its radius moves past it during movement of the waist joint. Contrary to this example, the layout used in the project placed collideable objects far away from the robot. This layout means that any likely collision situation can be avoided by preemptively creating trajectories that move the elbow further in before moving on, like what is shown in Figure 4.4. This strategy of movement makes collision avoidance largely redundant.

Secondly, the risk of collision from untracked objects were more significant than the risk of colliding with tracked objects. ”Untracked objects” in this case primarily means human beings. Other untracked objects include the touchscreen display, objects placed in the scene by onlookers or

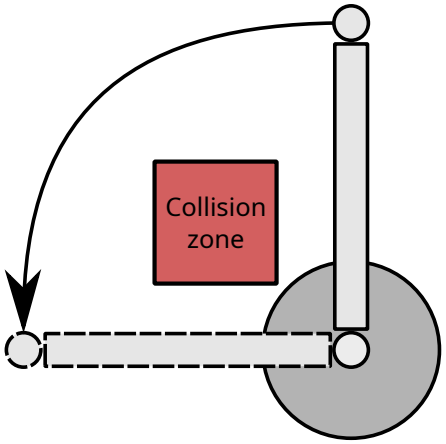



Figure 4.3: Movement forcing a collision, shown in a 2D plane. Top view.

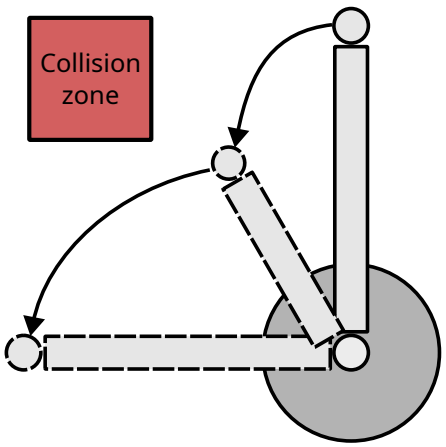



Figure 4.4: Movement avoiding a collision, shown in a 2D plane. Top view.

operators, and objects that were used to make waffles but not marked with an ArUco marker (e.g. the ladle).

Thirdly, collision detection greatly increased the complexity of the movement algorithm. A significant portion of the limited product development time was spent on implementing (and not least, debugging) collision detection. Another complexity rooted in collision detection was its reliance on MoveIt interfaces. Since MoveIt has limited compatibility with the Interbotix ROS-Python API, the collision-aware automatic mode and collision-unaware manual mode had to be launched with separate robot configurations. If collision detection had not been employed, it may have been possible to increase interoperability to the point where manual mode and automatic mode could be swapped between at runtime. Notably, this would allow the interactively following entertainment modes to be employed during waffle cooking downtime, even when the rest of program runs on manual mode.

Finally, the development effort was higher than expected. The disadvantages mentioned above are not disadvantages so much as they are inefficiencies. The drawback with collision detection lies not in its inherent cost, but in its opportunity cost. Collision detection took a large amount of development resources and yielded little to no benefit. This development time could have been used developing new entertainment modes, creating new functionality or refining existing features. With the benefit of hindsight, collision detection was a slightly useful feature that required a large time investment, giving it a poor cost-to-benefit ratio.

### 4.4.4 Entertainment Modes

During the field test it was observed that the entertainment modes that were added did not draw much attention. The hand tracking mode that was used may have been an uninteresting activity, but there is also a possibility that games are inherently less interesting to onlookers than cooking waffles. Further testing with more entertainment options are needed to conclude if entertainment modes are valuable. The field test does however suggest that waffle making was the most interesting part of the process, vastly surpassing entertainment modes.

Some features were originally considered for entertainment mode usage, but were deprioritized due to time constraints. One concept that was considered early in the development process was making the robot able to play chess. By connecting to a chess engine and using vision for the chess pieces, the technology needed to generate moves based on a physical chess board, would not cause much difficulty. The hardware part causes a challenge, as a chessboard that is large enough to interact with the robot arm gripper would be too large for the available workspace. Similarly tic-tac-toe and other board games were seen as less desirable options due to space constraints.

Another fun mode that was considered was a gravity compensating system that would let the stand user move the robot arm around with their human arms. This would theoretically use the motor current to register when a external force is applied, adjust to account for gravity and respond with a corresponding movement. Unfortunately, a system of this complexity could not be implemented in a reasonable time.

## 4.5 Societal Impact

The experiment with waffle making proved the robots ability to assist in domestic tasks. By extrapolating the framework to more general applications, the robot may be used to assist in multiple domestic tasks. One example of a situation where domestic assistance robots may come to use is in assisted living situations. After all, people who are unable to independently operate a waffle iron should not be denied their right to delicious waffles.

When used for marketing, the robot was a powerful tool that served as an engaging introduction to robotics and mechatronics. Engineers help shape the future of the world, and hopefully this engaging experience helps steer more people towards pursuing mechatronics or engineering in general.

### 4.5.1 Life Cycle

Most 3D-printed parts in the system can be replaced if damaged. However, it is important to avoid prolonged exposure to direct sunlight, as materials like ABS and PLA+ tend to become brittle when subjected to excessive UV radiation. A consideration is to use ASA as it has higher UV protection compared to ABS, but has similar properties. Under normal indoor conditions, these components should maintain their integrity for an extended period.

Supporting structures such as mounts are expected to last at least a year, provided the system is handled with care and used as intended.

The manipulator was inspected for loose screws prior to delivery and is expected to function reliably throughout the project duration, barring any unforeseen issues with the control system.

The waffle iron and camera was sourced from a reputable manufacturer and, with proper use, should remain operational for several years.

Internal components within the housing should remain functional as long as dust is periodically removed and the internal cooling fan continues to operate properly.

### 4.5.2 Economy

This project demonstrated a very expensive way to make waffles. The labor reduction from using a robotic manipulator to automate the process of waffle-making would not be deemed significant enough on its own to offset the cost of the robotic arm. The economical benefit of this project lies therefore not in what it could do in its current state, but the possibilities it unlocked as a proof of concept. There are two ways to improve the cost/return ratio of this project going forward. Unsurprisingly, they are either by reducing costs or by increasing the returns gained for the same cost.

The robot may be repurposed for tasks beyond its original objective of waffle-making. The robot setup used in this project was made to be highly customizable and adaptable. To perform a new task was as simple as creating new markers, attaching new interface cubes to the moveable objects and creating new paths. Research into other applications, e.g. vegetable chopping, greatly increases the utility a single robot arm can provide. The usage of the robot in its intended use case of marketing could also be considered as increasing the return on investment, by virtue of drawing in potential students to the mechatronics study program. It is, however, hard to put a price on the marketing value.

The main driver of costs in this project was the ViperX 300S 6DOF robotic manipulator, boasting a price of $7,149.95[24]. A design using a cheaper arm (e.g. the Trossen Robotics WidowX AI arm[27]) would significantly reduce costs. Exploring the possibility of adapting the tasks for a smaller arm (e.g. the WidowX 250S arm[26]) could also significantly drive down costs.

Another avenue of cutting costs was found when considering the RealSense camera used. The camera that was chosen has many features that were not used for machine vision purposes. The main benefit of the Realsense D455 camera was the accurate calibration that came out-of-the-box. This allowed the camera to be used without any manual calibration. A project where calibration is accounted for can therefore save costs by using a standard webcam instead of a multi-purpose camera like the D455. Options to use smartphone cameras may also be a way to reduce de facto cost, due to the availability of smartphones in the median household of the target markets. By cutting costs with the above measures, the financial feasibility of waffle automation increases to allow more potential customers.

### 4.5.3 Right to Repair

The right to repair is an ideology that is concerned with the consumers ability to maintain their product without external assistance. Products with a low right to repair include products that must be discarded entirely if a small fault occurs, or products that require trained experts to make

standard repairs. These factors can be counteracted by making the design modular and avoiding unneeded complexity. This design was highly modular and each component was replaceable. The hardest parts to repair were the heat resistant pars mounted on the waffle iron, due to the need for special 3D-printing filaments. Otherwise, most of the mechanical innovations can be replaced with a 3D-printer, using the models provided in the projects GitHub page[6]. The hardware was also modular and replaceable. The Jetson Orin Nano, power supply, RealSense cameras and robot manipulator were all replaceable "off the shelf" components, although the robot in particular was an expensive part to replace. Fortunately the robot had a repairable design as well, and therefore most components could be replaced individually with store-bought replacement parts. All replacement parts were drop-in replacements for their original part, aside from the Jetson which required software setup. To ease software setup, a setup script was provided as part of the projects codebase[6].

# Chapter 5

# Further Research

As with all projects, this project was done with a limited scope and with limited time. Due to these fundamental restrictions there were many avenues for improving or expanding upon the design that were discovered, but never followed. This chapter details the dreams that were never fulfilled and the ideas that were never brought to fruition. The following are suggestions for further research into improving and/or expanding upon the design.

## 5.1 Mechanical Design

This section covers concepts which could further enhance the systems ability to make waffles or make the process better, but were not incorporated due to time restrictions, cost or food safety aspects.

### 5.1.1 Filling Technology

To allow the manipulator to make waffles autonomously, it needs the ability to pour the batter into the iron. The method used would need a way to have reduced spillage while still allowing for easy cleaning and reliable use.

#### Gravity feed via Bottle

Mounting the bottle upside down as a container was a concept which would make the batter naturally run down and into a valve, which could be used for precise portioning. The bottles could be changed by rotating the mounting point so the bottle would be upright when mounting and could be exchanged without spillage. The batter would be transferred directly from the bottle to the iron.

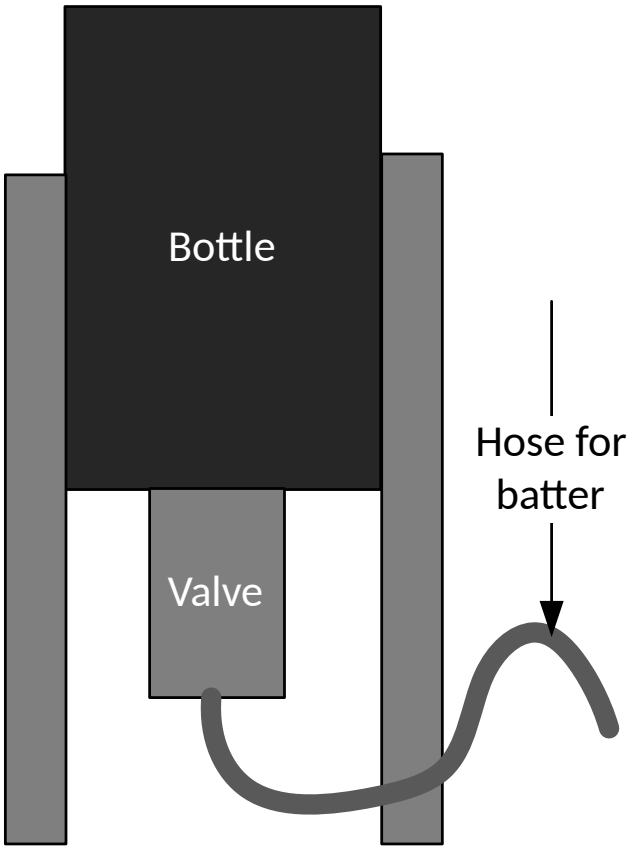



Figure 5.1: Concept for bottle storage

The bottle would be lifted as shown in Figure 5.1. As for drawbacks, the hose could cause some lag in the system, as well as making it hard to clean properly. The end piece would also need a way to be moved around so an interface cube would need to be designed.

To accurately dose the amount of batter needed to make one or two waffles a portioning system would need to be implemented to the gravity system. This system would need a way to accurately dose batter, while still being easy to clean and reduce spillage. Multiple possibilities were considered.

Another idea was to connect a hose to the valve as illustrated in Figure 5.1. This hose would then be ran to the waffle iron. In this concept the hose would work as a buffer, slowing down the reaction time of the system. The valve should therefore instead be placed on the end of the hose to help reduce lag. The second problem was that the pressure through the hose would vary as the level of batter in the bottle was reduced. The glugging effect[36] would hinder the movement of the batter, so a way to normalize the pressure within the bottle would be needed.

#### Distributing Fixed Ammounts of Batter

Using a valve for allowing or hindering the batter from running is not a new concept. The glugging effect[36] can be used to hinder the batter from overflowing a cup. The system would consist of the bottle, a valve and a cup as shown in Figure 5.2. The cup would be placed under the valve in such a way that the endpoint of the valve would be at the same height as the wanted batter. When the batter reached this level, it would stop the flow from the valve as the pressure from the top of the bottle would be lower than the atmospheric pressure outside of the bottle. The pressure difference would prevent the batter from flowing. This method could be made without electronics, but would make the system hard to clean by using a push-to-dispense valve. The amount of batter in the cup could be regulated by adjusting the height of the valve in the cup.

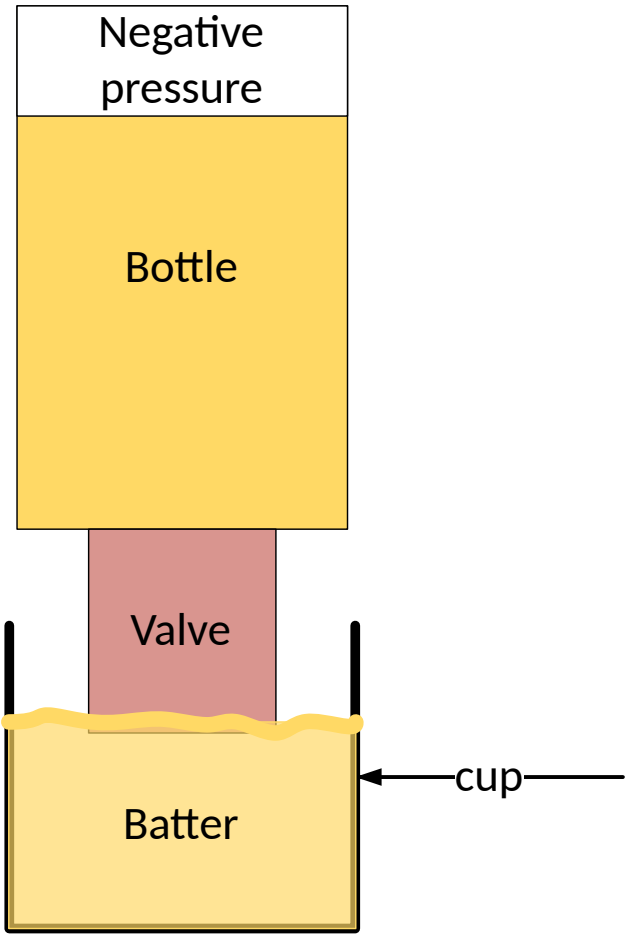



Figure 5.2: Concept for batter dosage

Using a weight sensor to measure the amount of batter is a concept which would make simple the regulation of the batter. A load cell would be placed under the cup and the valve would be opened electronically. The load cell would be connected to the microcomputer and the position of the valve would be controlled based on the weight of the cup, as detected by the sensor.

### 5.1.2 Lubrication of Iron

To lubricate the iron, a mechanism which the manipulator could interact with, was needed. The following section mentions different ways to lubricate the iron.

#### Spraying Mechanism

A bottle of spray grease was used to lubricate the iron. The first concept would use the grippers to squeeze a lever to activate the spraying mechanism as shown in Figure 5.3. This mechanism would be triggered by the grippers, when they compressed and recharged when the grippers were released. A screw attached the lever and main body together and worked as the pivot of the lever mechanism. The mechanism worked when the lever was lifted, but the grippers were not supplied with enough power to move the lever, either due to the length of the lever, lack of gripping force or due to friction.

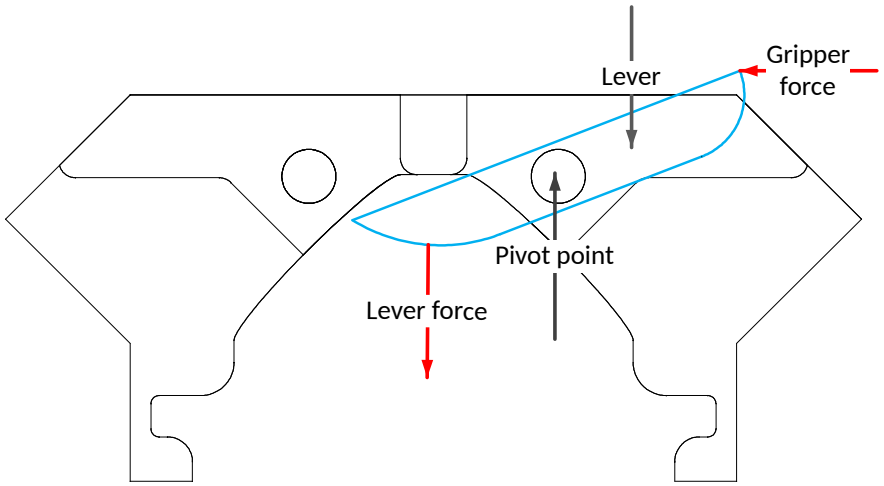



Figure 5.3: First spraying concept

The holder for the bottle was designed to be interchangeable. This part used the shape of the bottle to lock the bottle into place, where the interface cube would lock the bottle into place vertically. This was done both at the top and the bottom of the bottle, as shown in Figure 5.4.

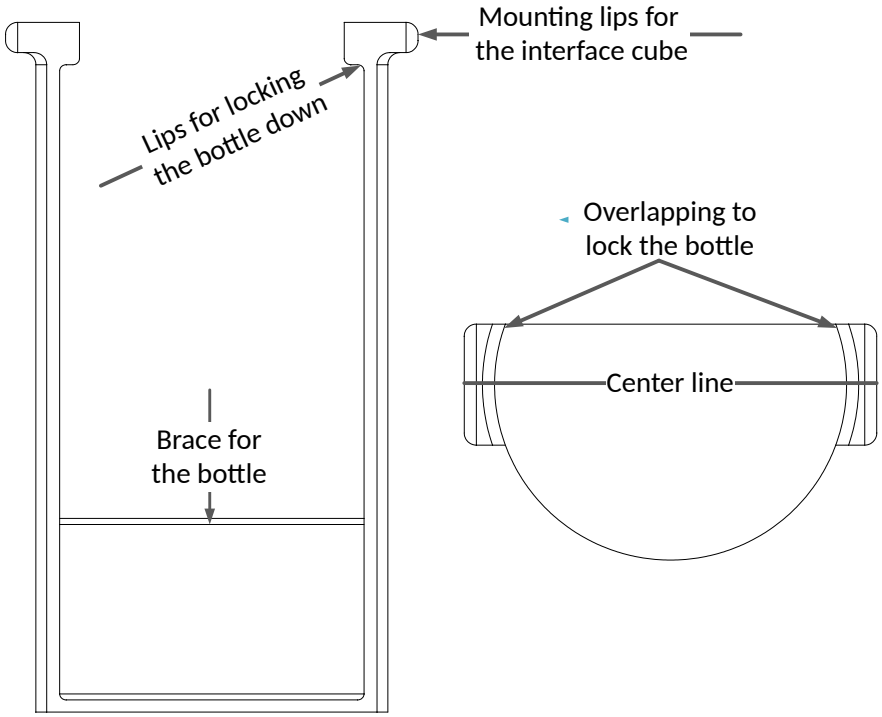



Figure 5.4: Grease bottle holder

Since the arm had a maximum grip size of 88 mm, the interface cube could not be larger than the gripper's max opening distance. It would also benefit from being a bit smaller than 88 mm to introduce a margin of error. This is why the interface cubes on the top of the grease holder were rounded as shown in Figure 5.5. This was opposed to being pointy, such as the interface cube framework suggested. This was due to the cube being too big to fit in between the grippers if they were the full size. This modification has proven not to affect the ability for the manipulator to grip the interface cubes and appears to be as reliable as following the original design framework would be.

Since the top of the bottle was shaped like a cone, some extra space was needed at the top, to lock the bottle down while still allowing the spraying mechanism to protrude. This was made by making

a cutout in the top part as shown in Figure 5.5. The back side of the bottle holder has a surface to place an ArUco marker for machine vision.

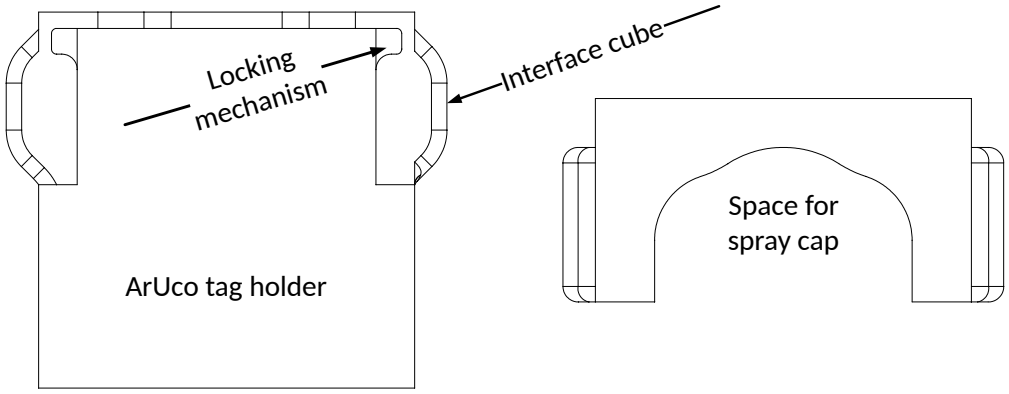



Figure 5.5: Top part of grease bottle holder

The two parts slid onto each other to lock the spray bottle in place. The ArUco marker was glued on with stick glue. The two part assembly was mounted as shown in Figure 5.6 and the real counterpart is shown in Figure 5.7.

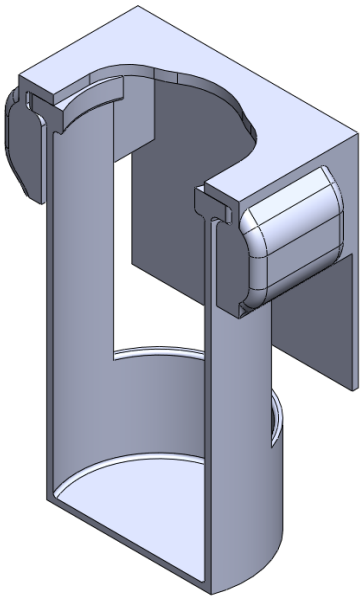


Figure 5.6: Spray adapter assembly

Figure 5.7: Spray adapter

The manipulator could not use the spraying mechanism by itself with the first lever concept. Therefore a servo was attached to the arm. This servo would be used to interact with the spraying mechanism. The servo that was used was a Dualsky servo model number DS6811[3]. This servo offered sufficient power, with 20 kg of power. It had good mounting options. The servo would be powered through a LM2596 buck converter[8] which reduced the voltage to 7.4V. The PWM signal to drive the servo would come from the Jetson. The mounting position was chosen such that the arm of the servo would press down onto the spray cap when moved. The servo was placed such that it extended out and above the grippers. A mount was made that incorporated a slot for the servo, as shown in Figure 5.8. The design also featured braces to ensure that the part would not deflect significantly.

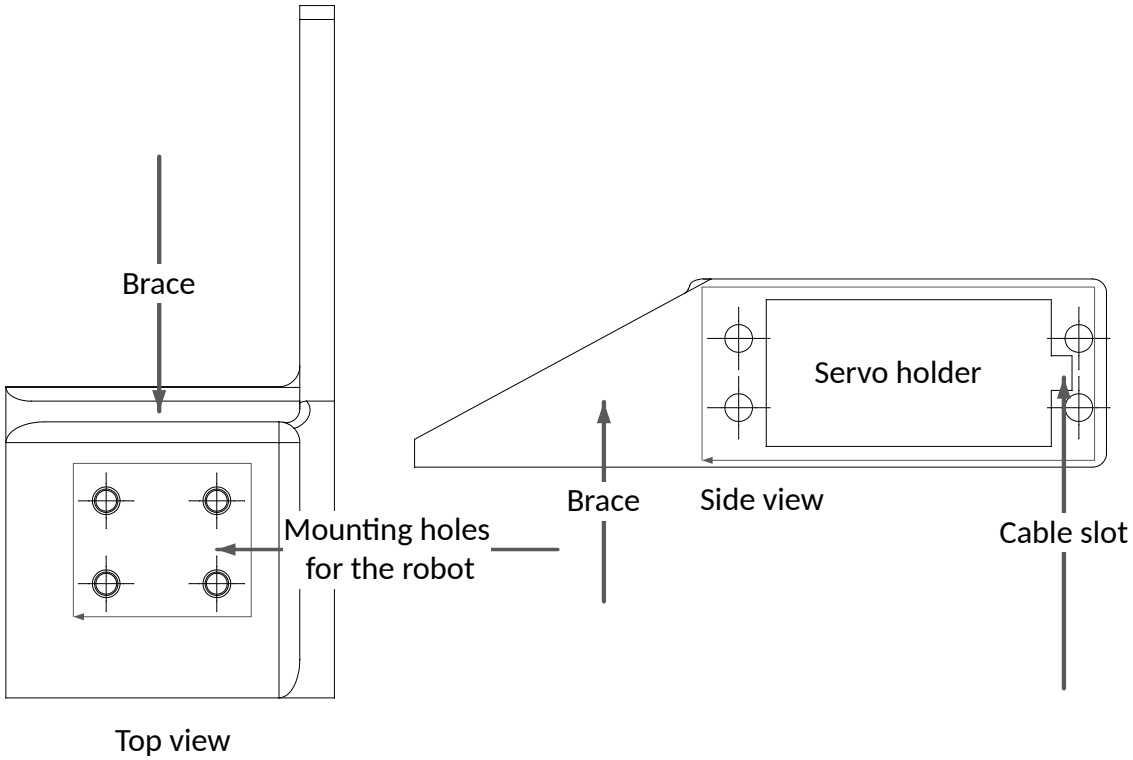



Figure 5.8: Servo holder

### Motorized Grease Spray Mechanism

The spraying mechanism was scrapped due to issues with controlling the servo. The signal had extremely bad noise that was thought to be from the induction from the power cables. This could also be from the buck converter used, as it does not deliver pure DC power. The signal became so unstable that the servo didn't react at all. In Figure 5.9, it is shown how bad the PWM signal was. On the left is the signal after a pulldown resistor was added, and the right side shows before a pulldown resistor was added. This was measured with all three cables twisted together.

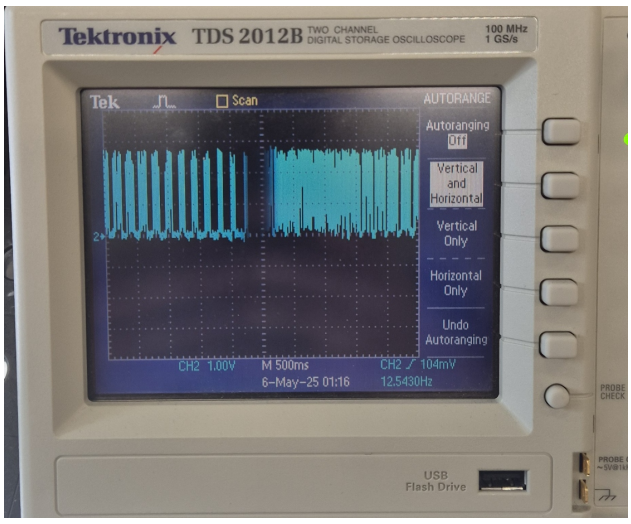



Figure 5.9: Signal from the servo signal cable

So to try to reduce noise, the signal cable (PWM) was twisted with its own ground and the power cable had its own separate ground cable it was twisted with. The cables were then separated from each other, and a capacitor of $50\mu F$ between power and ground was added on the end of the cable

to reduce noise. The ground cables were connected close to the servo to have a common ground close to the servo. Which helped, but not enough to get a reliable signal. The idea was scrapped.

#### Using a Brush

For lubricating the iron, a brush could also be used. This would be a crude but effective solution. The brush would need an interface cube and something to store the lubricator. As long as the passes with the brush would cover a sufficient area of the iron, this method could be implemented. A literature review into existing painting robot technologies may be helpful toward this goal.

## 5.2 Software

This following section explores the possibility on expanding upon or improving the software of this project. Some improvements of existing implementations and possible solutions to issues that occurred are mentioned. The sections begins by talking about improvements to the machine vision software and then the robot control software. Finally avenues of new software approaches will be discussed at the end of the section.

### 5.2.1 Camera Software

Following the completion of the primary development cycle, several areas were identified that could enhance system functionality in future iterations. These include exploring alternate camera configurations to support more dynamic user interactions, and refining the marker detection process to address observed inconsistencies during operation. The following text outlines these potential improvements.

#### Alternate Camera Setups

Several camera configurations were explored. It was initially concluded that a single camera would be sufficient for performing object pose estimation. However, these camera setups could provide added functionality and flexibility in the system operation.

One improvement involves integrating an auxiliary camera mounted on the robot's wrist. This configuration directly supports the systems entertainment mode. The primary camera can remain focused on core machine vision tasks, while the secondary, robot mounted, camera enhances user interaction during dynamic activities. For example, during a rock paper scissors interaction, the robot facing the user, can more reliably detect hand gestures using the robot mounted camera, eliminating the need for users to reach into the workspace monitored by the main camera.

Another proposed setup involves using a static, tripod mounted camera positioned on the table and oriented toward the user. While this configuration offers ease of implementation and simplicity in programming due to its fixed placement, it lacks the versatility of a mobile camera, which can provide near-complete coverage of the interaction space.

Moreover, deploying a hybrid camera system for pose estimation offers both functional and entertainment benefits. From a technical standpoint, the main camera could estimate the general direction of a marker, while the robot-mounted camera pinpoints its precise location. From a user perspective, this creates a more dynamic and interactive experience, which may enhance the systems appeal in demonstration or marketing contexts by showcasing camera cooperation in real time.

#### Further Improvements to the Marker Detection

During the testing phase of the pose estimation system, several issues were encountered that occasionally caused the system to transition into an incorrect state. If a marker was momentarily not recognized for only a few milliseconds, the state machine could erroneously interpret this as a valid transition condition, leading to a wrong state. The precise cause of this behavior remains uncertain. However, preliminary analysis suggested that the problem originated from the marker

detection module, not the pose estimation itself. Specifically, the detection parameters may have been overly restrictive, causing the system to invalidate markers that should have been accepted.

To address this issue, fine-tuning these parameters is recommended. Lowering the confidence threshold slightly, relaxing the minimum contour size, or increasing the allowable tolerance for marker rotation and deformation could improve detection robustness. It is important to emphasize that the pose estimation process once a marker is successfully detected, operates correctly. The malfunction lies solely in the upstream detection phase.

### 5.2.2 Robot Software

This section outlines the development potential of the robot. It focuses on three key areas: reliability in automatic mode, accuracy in pose recording, and precision in collision bounding boxes.

#### Automatic Mode Improvements

Automatic mode was a conceptual success, but it lacked reliability. Improvements to the motion concept should therefore first and foremost be focused on making movements repeatable and stable. The most significant problems in reliability were imprecise movement and dynamic movement leading to impossible poses. The primary way to increase software-side reliability would be by increasing the precision of movements.

The primary fundamental bottleneck of automatic mode movements comes from the cameras ability to sense the ArUco markers. Aside for the trivial solutions of increasing marker size and camera resolution, there are also ways to re-purpose the fun time camera to increase marker detection accuracy.

The easiest way to do this would be to use a dual camera setup. A filtering algorithm, illustrated in Figure 5.10, could then be used to prioritize the camera that was closest to the marker. The closest camera has the clearest view of the marker and therefore the most accurate reading.

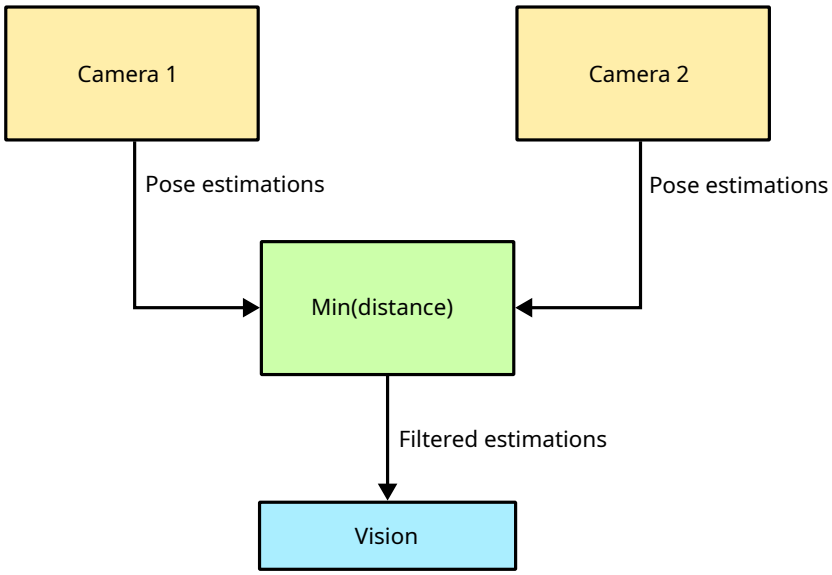



Figure 5.10: Filtering concept for dual camera setup

A more complicated approach would be to use the camera-on-robot setup. Here the main camera could be used to guide the robot in the general direction of the marker, and the robot camera would then be used for precision movement. This would primarily be an exercise in movement logic, and selection of cameras to best solve the task at hand. This approach would give the most precise pose estimations, and the camera on the robot could also be used for entertainment modes.

Fixing the issue of the robot attempting impossible movements or movements that cause large joint movements requires a full rework of the movement sequencing. The most straightforward approach

would be to implement a fallback to manual mode recordings if the movement cannot be reasonably executed. This method could not be done in this project due to the compatibility issues with MoveIt and the Interbotix interface. If the compatibility issues were resolved, this approach would be a useful way to increase system robustness. Another way to increase reliability would be the implementation of backup movement plans that could be used if the intended path was impossible to execute.

#### Pose Recording Improvements

The pose recording method that was used in the project had a noticeable inaccuracy. This inaccuracy made it difficult to prepare positions. The solution that was used was to use a method of rapid recording and testing, but this method is cumbersome and time consuming. Using another more reliable approach to record poses would lower preparation time and therefore increase system adaptability. An alternate approach using the established vision system along with a marker mounted on the robot end effector may improve accuracy as opposed to relying on the feedback from the robot itself. Solving this problem in software might also be possible. One software based solution would be a statistical approach using measured errors in different poses to add an offset to the final recordings based on expected measurement error.

#### Bounding Box Rework

The bounding boxes used for MoveIt's collision avoidance could be improved to be more precise. The system in its current state used the axis-aligned rectangular box paradigm due to their simplicity and compatibility with manual mode collision checks. MoveIt does however include more sophisticated parameters for collision object generation. By using the orientations of the markers, the bounding boxes could be updated to dynamically rotate if the collision objects were moved instead of using axis-aligned boxes. Another feature that may be helpful is MoveIt's collection of collision object shapes beyond a rectangle. For example, the spray lubrication bottle may be better represented by a cylinder shape than a rectangle. By improving the bounding boxes, the robot may more accurately avoid collisions.

## 5.3 Framework Expansion and Future Directions

While the current system demonstrates effective functionality, further development is essential to fully realize the potential of the existing framework. This section explores prospective features that build upon the current architecture, aiming to expand both the entertainment capabilities and the task adaptability of the system. By leveraging the established motion and vision frameworks, new functionalities can be implemented with minimal structural changes, offering promising directions for future research and application.

### 5.3.1 New Entertainment Modes

The most straightforward way to increase viewer retention is to do the interesting part of the process, which is making waffles, and scaling it to multiple waffle irons in parallel. By doing so, the amount of customers served is increased, and the robot has less downtime where onlookers may lose interest. The motion framework that has been established makes expanding the design with more movements fairly trivial. The challenge with this feature lies rather in the software side, as keeping track of multiple waiting sessions simultaneously might be difficult to implement.

The development of new interesting entertainment features remains a significant area for improvement. Creative games or demonstrations of the robot's abilities would hopefully bring more attention to the robot and raise interest in the field of robotics that mechatronics is tightly interconnected to. Field testing showed that onlookers were more interested in watching the robot work independently than interactive games that required them to participate. Further research in entertainment modes should therefore first and foremost be in tasks that the robot can perform independently, without requiring interaction.

Blurring the lines between entertainment and cooking, the system could also be expanded to allow the user to request the waffles served with jam or other toppings. The technology to apply this would be similar as that of pouring batter, by using spoons to pick up the topping of choice from a container, and recording movements to mimic the pouring or other application of the topping. This feature would fit well into the time slot reserved for entertainment mode as it would minimize the downtime where the iron is not cooking any waffles.

### 5.3.2 Adapting to Future Tasks

The pose recording framework, along with the interface cube concept, makes re-purposing the system for other tasks fairly simple. Examples of other cooking applications include the acts of making pancakes and serving coffee. Non-cooking applications are also worth consideration. To make the system easier to adapt, research into an easy-to-use programming framework for sequential robot movements would also be valuable. By generalizing the framework, the same robot arm could be used to perform multiple tasks.

To make the system even more adaptable, research could also be done in the field of a movable robot arm. One could imagine a single arm dedicated to assisting with kitchen tasks moving across the counter to do multiple tasks simultaneously. Starting with chopping vegetables, the robot could move to another place to help with mixing batter before finally moving to the frying pan to flip pancakes. While many of the specific tasks mentioned might not be feasible with the current hardware, the concept of a moving robot may very well be worth researching. One way of creating such a movable robot would be to mount it on a cart attached to a rail spanning the length of the workspace. Using the machine vision system, the robot could then fine-tune its movements to account for inaccuracies in the cart position. A sketch of the proposed concept is shown in Figure 5.11.

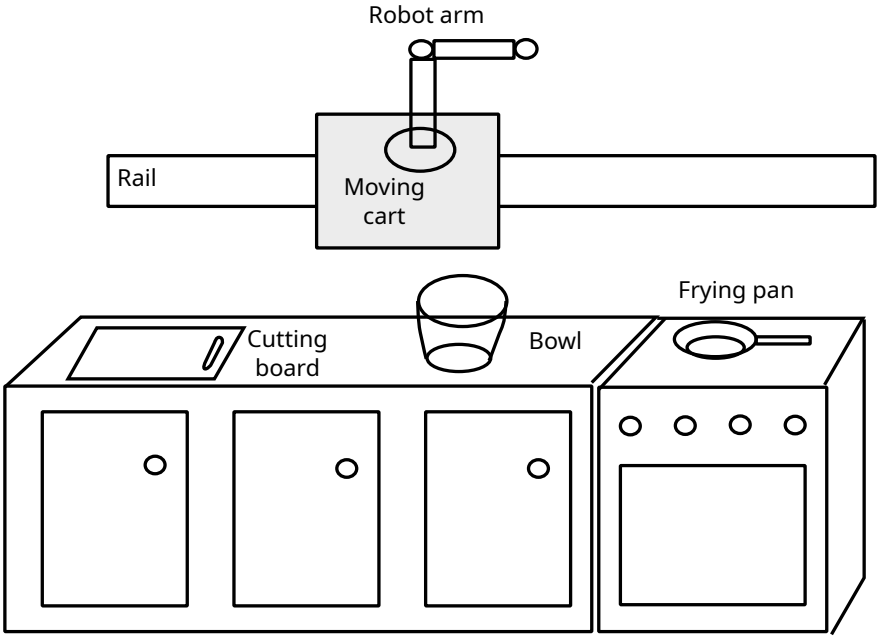



Figure 5.11: Kitchen robot on movable rail concept

# Chapter 6

# Conclusion

The project presented in this report originated from a desire to promote the mechatronics engineering program at the University of Agder. This was to be done by creating a demonstration of a mechatronic system that is accessible and compelling. The concept of a waffle making robot was chosen for its immediate appeal and its potential to showcase a wide range of mechatronic principles: robotics, control systems and machine vision. The project served as an advertisement and a platform to explore how robotic manipulation and computer vision can be applied together. This project was also intended to set the stage for further research into collaborative and assistive robotics for domestic applications.

Many parts of the system were built using existing technologies and software. The robotic arm, the ViperX-300S from Trossen Robotics[24], came with software support and integration for the Robot Operating System (ROS2). ROS2 made it possible to communicate with and control the manipulator. For advanced tasks like path planning and inverse kinematics, the system used the MoveIt framework [31]. The cameras used in the system were Intel RealSense models [9]. These were factory-calibrated to produce images with very low distortion, which made them easy to use in a machine vision setup. Image processing was handled by OpenCV, a widely used computer vision library[20]. It enabled the use of ArUco markers for 3D object detection and pose estimation. 3D-printing was also employed to produce novel mechanical parts and adapters to make the system more accessible by the manipulator. 3D-printing also allowed for rapid prototyping of the parts. Multiple 3D-printing materials were used. The materials used were suitable for functional components, enabling the use of parts with enhanced properties. Together, these technologies formed the foundation of the system. These also allowed the focus of the project to shift toward integrating components, rather than developing everything from scratch.

The system was based on a robot arm controlled with a microcomputer using a camera. The microcomputer was placed in a housing along with the power supply to shield the electronic components from harm. The roof of the housing was used to mount the arm, increasing the compactness of the system. The arm was the primary actuator, handling every step of the process from opening the waffle iron to placing waffles in a serving area. A servo was intended to be used as a secondary actuator, triggering the spray mechanism of the spray grease. It was however not incorporated, due to technical difficulties. The camera detected objects using ArUco markers. A touch screen HMI was used to interact with the robot controller. This also could control the access to custom games used for entertainment when the robot had downtime during cooking.

The camera system enabled the robot to perceive and interact with its environment. An Intel RealSense depth camera was used to provide both RGB and depth data, which allowed the system to localize objects in 3D space. Object detection and pose estimation were implemented using ArUco markers through OpenCV's Python interface. This approach enabled accurate tracking of tools within the robot's workspace. The RealSense camera's factory calibration ensured low image distortion and accurate depth alignment, reducing the complexity of setup and calibration. Since all camera processing was handled directly in Python using OpenCV, the vision system remained

lightweight. This setup allowed the robot to determine the spatial relationship between itself and objects, enabling relative movement planning.

If an object or reference marker was moved beforehand, the robot could adapt its movements accordingly. To plan movements, the robot was moved by hand to a target location and its position was recorded with a relative offset to a reference marker. If the marker was later moved, the robot could move to the same relative pose, so the robot had the same interaction with the target object. Two modes of control were implemented. A simple approach called ”manual mode” where the movements were predetermined, and an ”automatic mode” that adjusted the target poses based on camera feedback. Both modes had the capacity for collision-aware movement, although they were handled in different ways based on the selected mode.

A public test was performed to estimate the system's marketing capacity. The system was able to attract a constant stream of approximately 5 onlookers during its operation. The system boasted a 100% success rate in waffle making with spillage during the public test in manual mode. Some features of the system did not work as expected. Such as the spray not being fully functional and the delivery station poles bent during the attempted delivery of the waffle. Entertainment mode games did not notably draw attention from the customers. In other testing, when automatic was engaged, the system performance dwindled, and the manipulator did not reliably pick up items. Testing showed the machine vision gave high precision input from pose estimations, but the robot arm failed to move precisely based on its given input data. Therefore, the manual mode was ultimately preferred. Considering the research question, **machine vision could successfully be used to assist robot arms in domestic tasks such as waffle making, although this project failed to integrate these concepts into a finished product.**

# Bibliography


[1] HUD Framework for python | CTkinter. *CTkinter*. URL: https://customtkinter.tomschimansky.com/. Accessed: 2025-05-09.

[2] Grimstad Adressetidende. *Se video av roboten som steker vafler*. URL: https://www.gat.no/nyheter/i/0GwK40/se-video-av-roboten-som-steker-vafler. Accessed: 2025-04-20.

[3] Dualsky. *SERVO Specification*. Accessed: 2025-05-13. 2022. URL: http://www.dualsky.com/AjaxFile/DownLoadFile.aspx?FilePath=/UpLoadFile/20220926/SERVO%20specification.pdf%20&fileExt=file.

[4] eSUN 3D. *eSUN PLA+ Filament Technical Data Sheet*. Accessed: 2025-03-05. 2021. URL: https://www.esun3d.com/uploads/eSUN_PLA+-Filament_TDS_V4.0.pdf.

[5] Google. *Hand-detection*. URL: https://ai.google.dev/edge/mediapipe/solutions/guide. Accessed: 2025-05-06.

[6] havarduia. *vaffelgutta*. URL: https://github.com/havarduia/vaffelgutta. Accessed: 2025-05-15.

[7] Ingcool. *7IP-CAPLCD: 7-Inch HDMI IPS Display with Capacitive Touch*. Accessed: 2025-05-13. 2022. URL: http://www.ingcool.com/wiki/7IP-CAPLCD.

[8] Texas Instruments. *LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator*. Tech. rep. SNVS124G. Accessed: 2025-05-13. Texas Instruments, 2023. URL: https://www.ti.com/lit/ds/symlink/lm2596.pdf.

[9] Intel. *Intel camera specs*. URL: https://www.intelrealsense.com/compare-depth-cameras/. Accessed: 2025-01-17.

[10] Interbotix. *IKinSpace.m*. URL: https://github.com/Interbotix/ModernRobotics/blob/53b49d91d43b72dc4442e9565b31bd0b281511f7/mr/IKinSpace.m. Accessed: 2025-02-26.

[11] Interbotix. *ompl_planning.yaml*. part of the *interbotix_ros_manipulators* GitHub repository. Branch name: "humble". URL: https://github.com/Interbotix/interbotix_ros_manipulators/blob/b66d5b905725351dd71d3251a06cd3f4c777940f/interbotix_ros_xsarms/interbotix_xsarm_moveit/config/ompl_planning.yaml. Accessed: 2025-04-20.

[12] J.J. Kuffner and S.M. LaValle. "RRT-connect: An efficient approach to single-query path planning." In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. 2000, 995–1001 vol.2. DOI: 10.1109/ROBOT.2000.844730.

[13] Robi3 Robotics Limited. *Egg-waffle Making Robot*. URL: https://research.robi3.com/. Accessed: 2025-04-20.

[14] Mar_lud. *GoPro Mount*. Accessed: 2025-04-13. 2022. URL: https://www.printables.com/model/125526-gopro-mount/files.

[15] Markforged. *Composites Material Datasheet*. URL: https://s3.amazonaws.com/mf.product.doc.images/Datasheets/Material+Datasheets/CompositesMaterialDatasheet.pdf. Accessed: 2025-02-26.

[16] My 3D Concepts. *How 3D Printing Works*. Accessed: 2025-04-07. unknown. URL: https://my3dconcepts.com/explore/how-3d-printing-works/.

[17] NVIDIA. *IsaacROS*. URL: https://developer.nvidia.com/isaac/ros. Accessed: 2025-05-15.

[18] NVIDIA. *Jetson orin nano super*. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/. Accessed: 2025-05-06.

[19] OpenCV. *ArUco Docs*. URL: https://docs.opencv.org/4.x/d5/dae/tutorial_ArUco_detection.html. Accessed: 2025-01-17.

[20] OpenCV. *OpenCV Website*. URL: https://opencv.org/about/. Accessed: 2025-04-15.

[21] Open Robotics. *ROS - Robot Operating System*. URL: https://www.ros.org/. Accessed: 2025-03-31.

[22] Trossen Robotics. *MoveIt2 Interface and API*. URL: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros2_packages/moveit_interface_and_api.html. Accessed: 2025-04-26.

[23] Trossen Robotics. *Python-ROS Interface*. URL: https://docs.trossenrobotics.com/interbotix_xsarms_docs/python_ros_interface.html. Accessed: 2025-03-24.

[24] Trossen Robotics. *ViperX 300 S*. URL: https://www.trossenrobotics.com/viperx-300. Accessed: 2025-04-01.

[25] Trossen Robotics. *ViperX-300 6DOF - Default Joint Limits*. URL: https://docs.trossenrobotics.com/interbotix_xsarms_docs/specifications/vx300s.html#default-joint-limits. Accessed: 2025-02-26.

[26] Trossen Robotics. *WidowX 300 S*. URL: https://www.trossenrobotics.com/viperx-250. Accessed: 2025-05-15.

[27] Trossen Robotics. *WidowX AI*. URL: https://www.trossenrobotics.com/widowx-ai. Accessed: 2025-05-15.

[28] ROBOTIS. *DYNAMIXEL XM430-W350-T*. Accessed: 2025-05-18. n.d.

[29] ROBOTIS. *XM540-W270-T/R DYNAMIXEL Actuator e-Manual*. Accessed: 2025-05-19. 2025. URL: https://emanual.robotis.com/docs/en/dxl/x/xm540-w270/.

[30] Marlin Steel. *What is the Best Food Grade Stainless Steel?* Accessed: 2025-05-10. 2025. URL: https://www.marlinwire.com/blog/what-is-the-best-food-grade-stainless-steel.

[31] Ioan A. Sucan and Sachin Chitta. *MoveIt*. URL: moveit.ros.org. Accessed: 2025-04-01.

[32] Ultimaker. *Ultimaker ABS Technical Data Sheet (TDS)*. Accessed: 2025-04-01. 2022. URL: https://um-support-files.ultimaker.com/materials/2.85mm/tds/ABS/Ultimaker-ABS-TDS-v5.00.pdf.

[33] Ultimaker. *Ultimaker TPU 95A Technical Data Sheet v5.00*. Accessed: 2025-05-16. 2021.

[34] Ultralytics. *YoloV8*. URL: https://yolov8.com/. Accessed: 2025-04-29.

[35] APRIL Robotics Laboratory at the University of Michigan. *Apriltag*. URL: https://april.eecs.umich.edu/software/apriltag. Accessed: 2025-05-06.

[36] Wikipedia contributors. *Glugging*. Accessed: 2025-05-13. 2025. URL: https://en.wikipedia.org/wiki/Glugging.

# Appendix A

# Arm Moment Calculations

The example shown in Figure A.1 demonstrates the reason why the upright position experiences less moment than the reverse position. To illustrate this, a simplified model of the robot with two joints carrying a load represented by a force F is used.

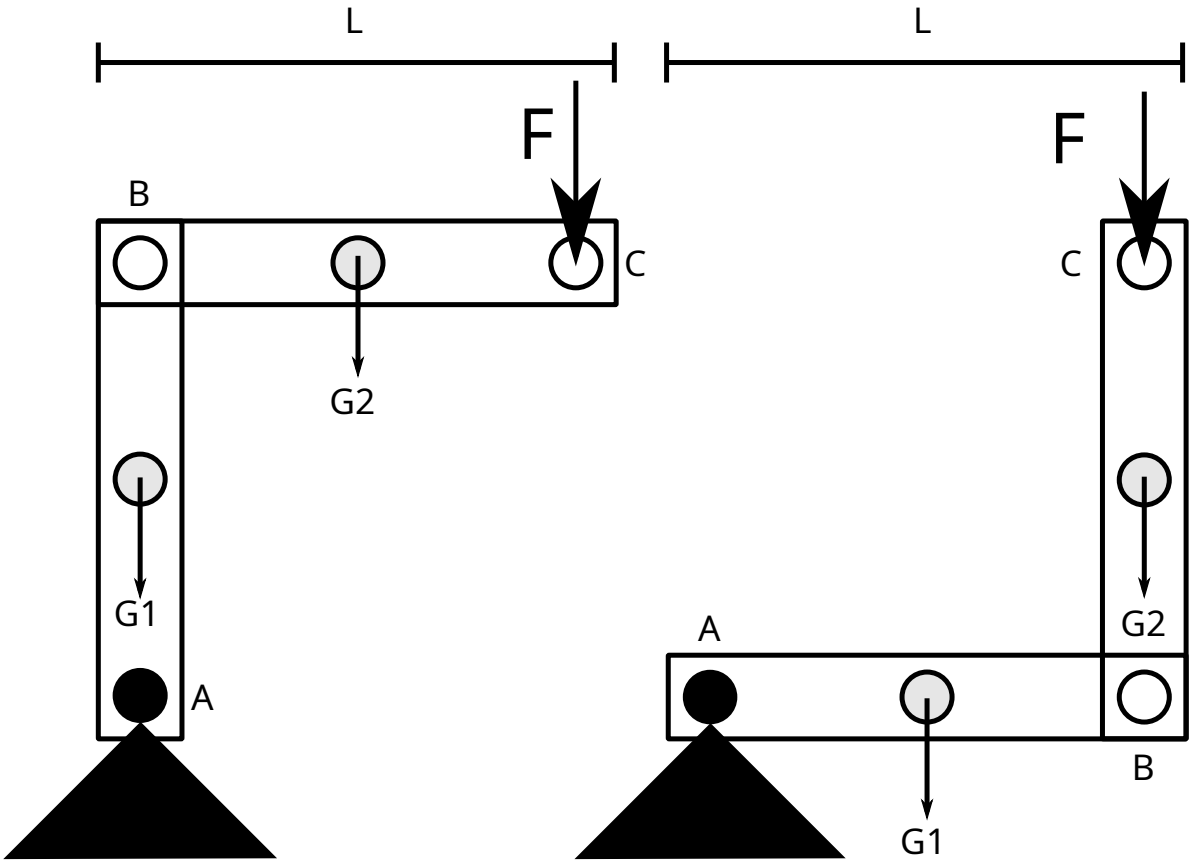



Figure A.1: Arm in an upright position (left) and reverse position (right)

In the example on the left, the moment felt by point A is given by:

$$M_A = \sum_{n=0}^{N} F_n a_n = G_1 \cdot 0 + G2 \cdot \frac{L}{2} + F \cdot L$$

And on the right it is given by:

$$M_A = \sum_{n=0}^{N} F_n a_n = G_1 \cdot \frac{L}{2} + G2 \cdot L + F \cdot L$$

comparing the two sides:

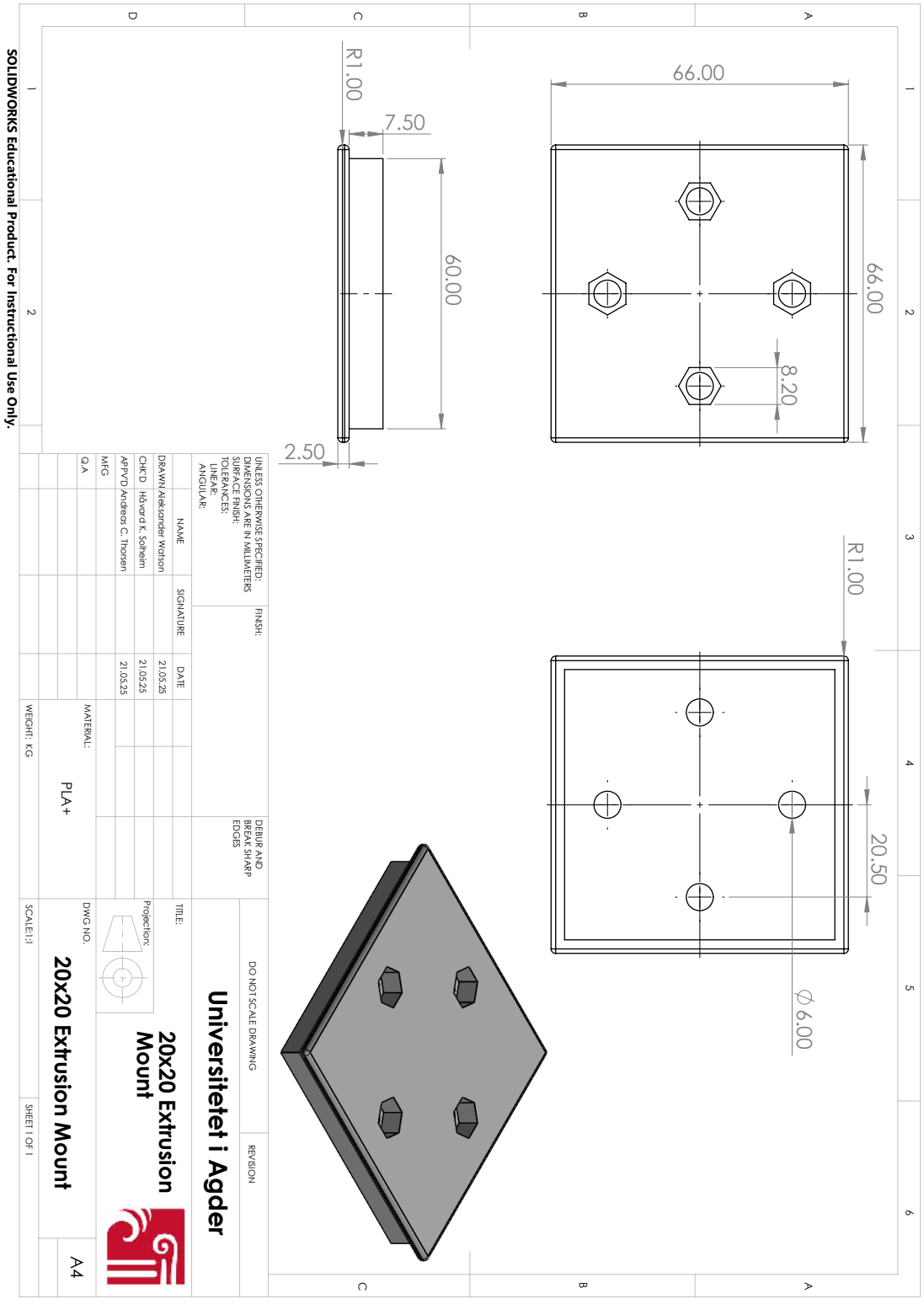
$$\begin{array}{rcll} G2 \cdot \frac{L}{2} + \cancel{F \cdot L} & VS & G_1 \cdot \frac{L}{2} + G2 \cdot L + \cancel{F \cdot L} & | \; - G2 \cdot \frac{L}{2} \\ 0 & < & \frac{L}{2}(G_1 + G_2) & \end{array}$$

This example demonstrates that the upright position is subjected to less moment than the reverse position, by a factor of half the length of the arm multiplied by the forces inflicted by gravity on the arm itself. The same principle applies for the real arm in its complexity. The calculation also shows that, if the arm were to be operated without gravity, e.g. in space, this moment consideration would no longer be relevant.

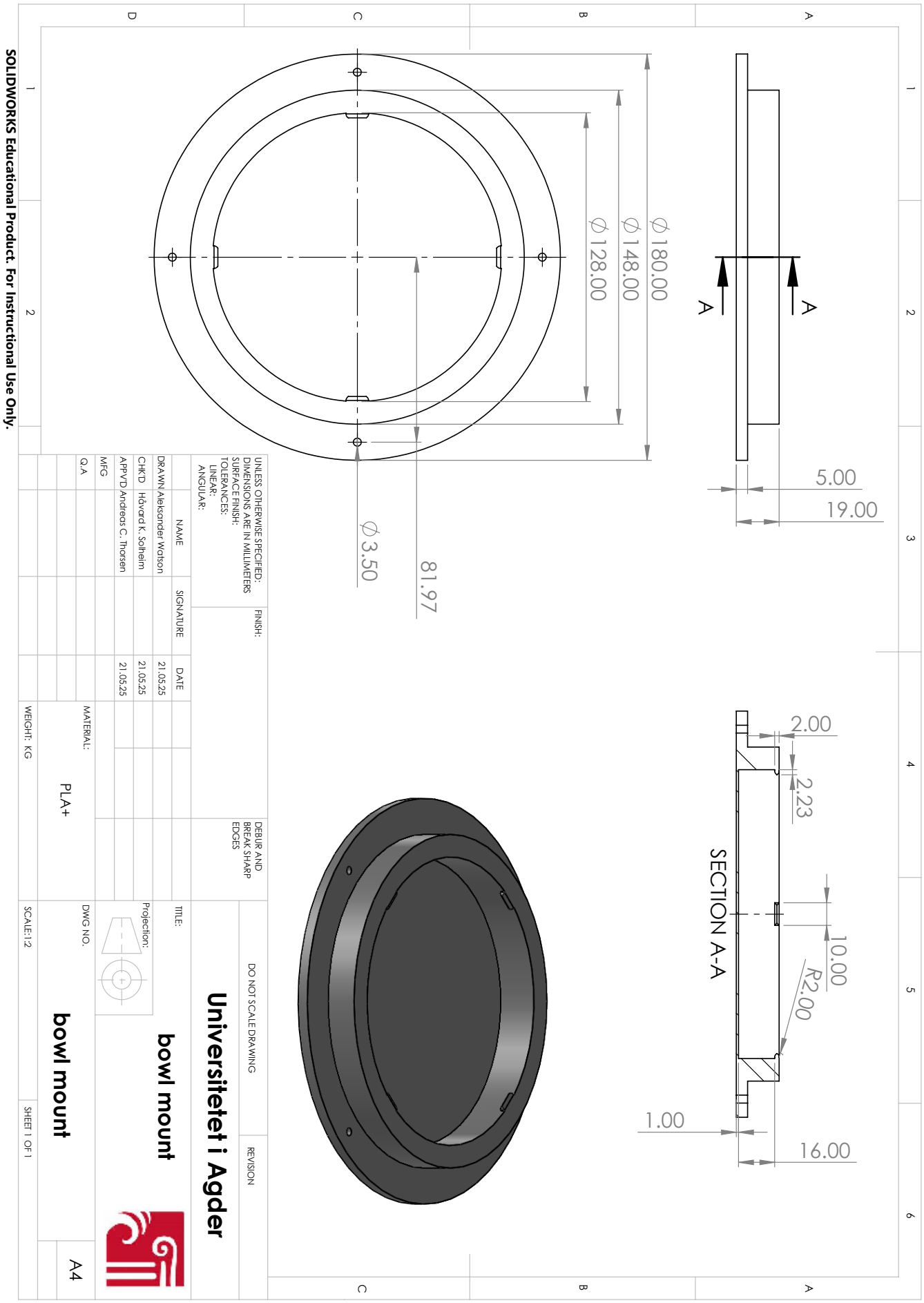
# Appendix B

# Technical Drawings

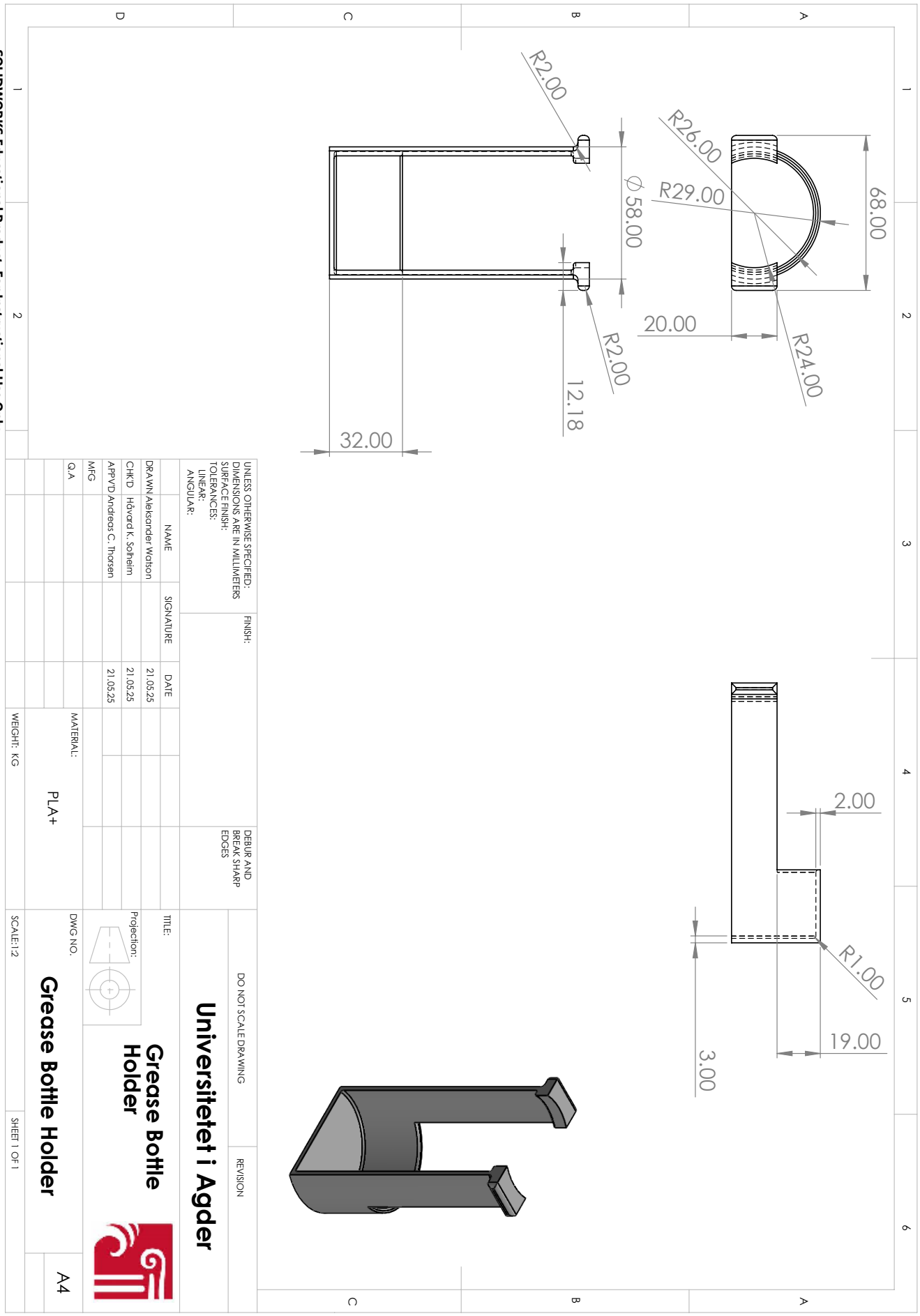
## 20x20 Extrusion Mount



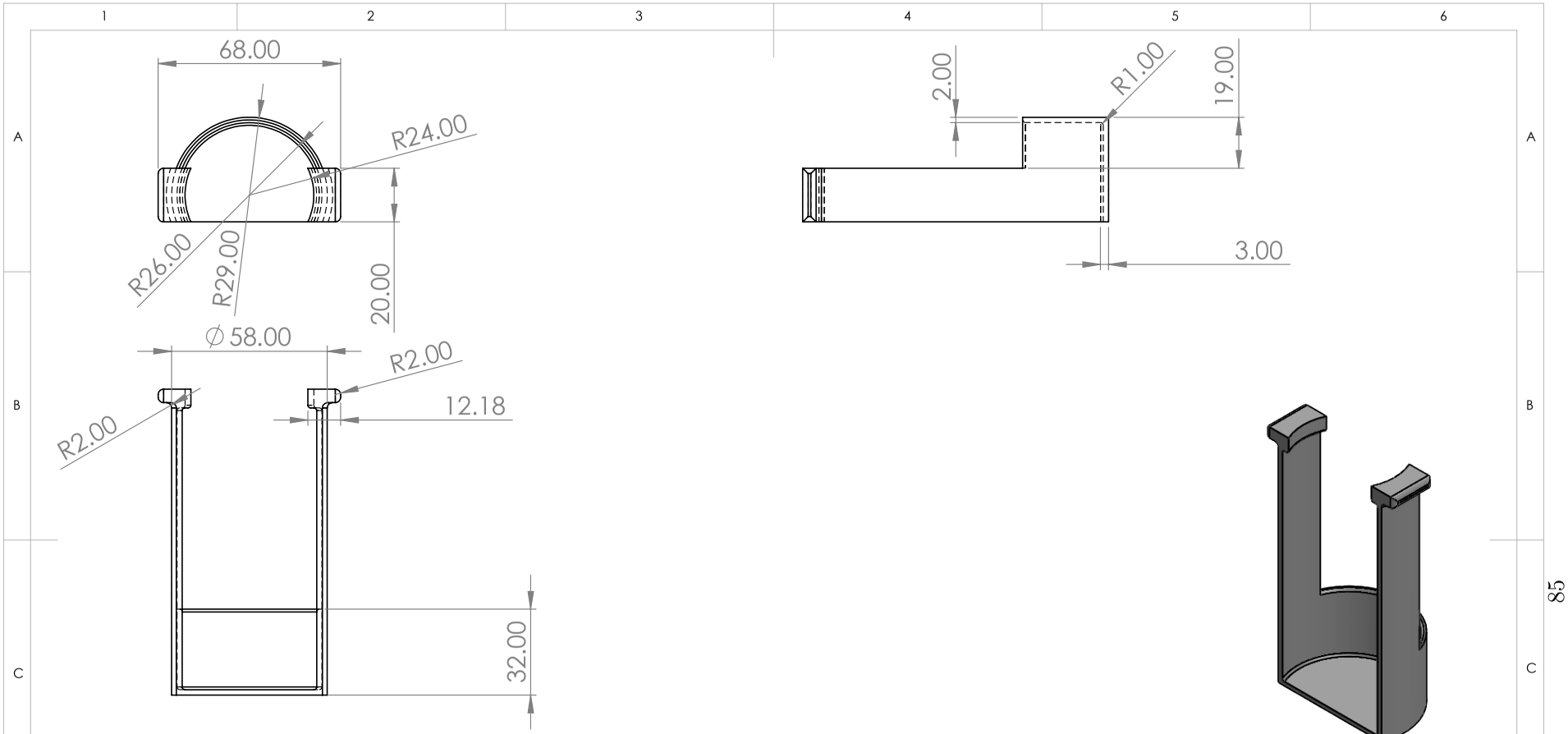
# Bowl Mount


A
A
5.00
19.00
Ø180.00
Ø148.00
Ø128.00
81.97
Ø3.50
SECTION A-A
2.00
2.23
10.00
R2.00
1.00
16.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBUR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN Aleksander Watson
21.05.25
CHK'D Håvard K. Solheim
21.05.25
APPV'D Andreas C. Thorsen
21.05.25
MFG
Q.A
MATERIAL:
PLA+
WEIGHT: KG
DO NOT SCALE DRAWING
REVISION
Universitetet i Agder
TITLE:
bowl mount
Projection:
DWG NO.
bowl mount
A4
SCALE:1:2
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

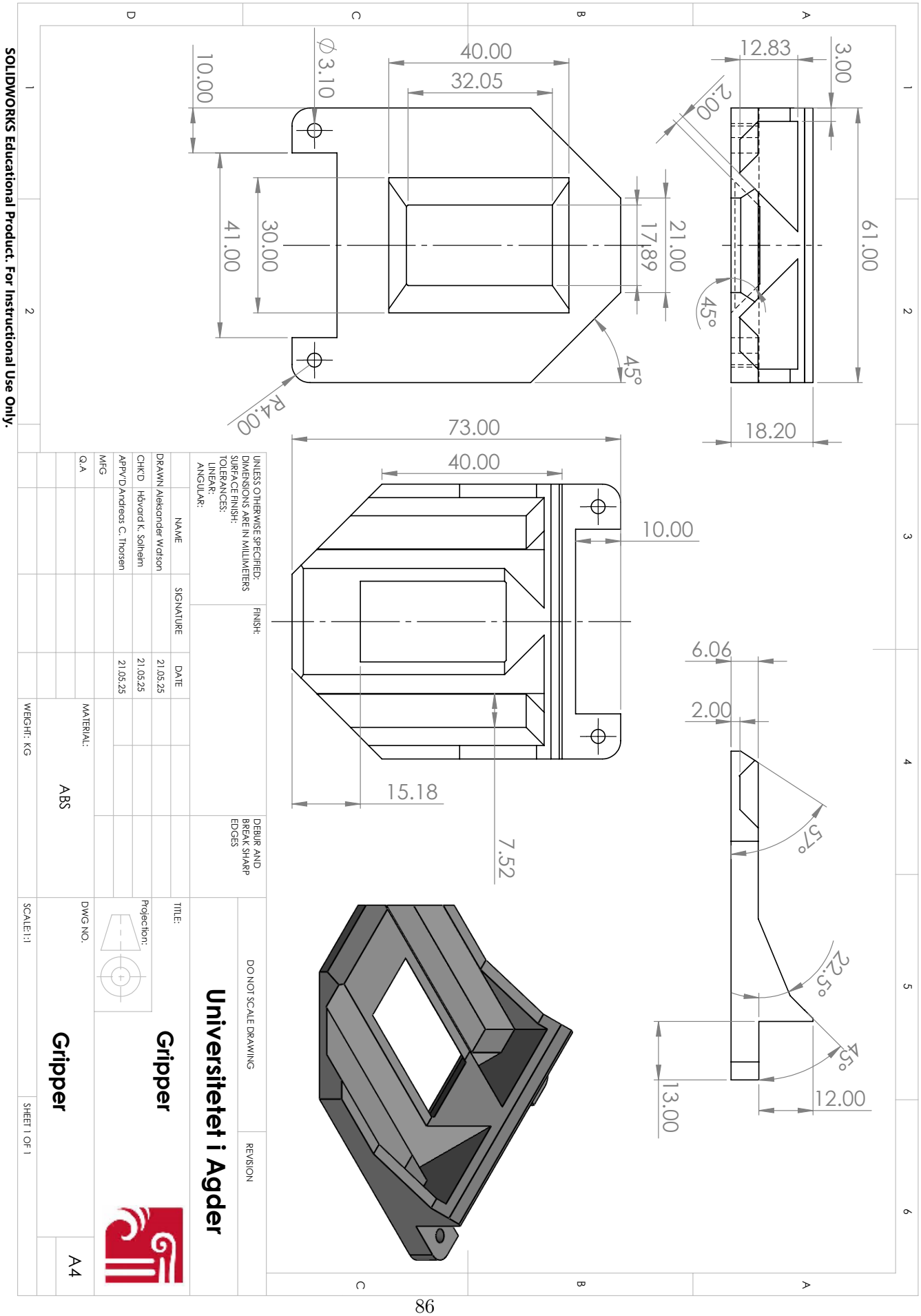
# Grease Bottle Holder

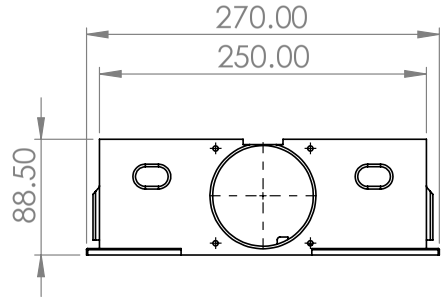


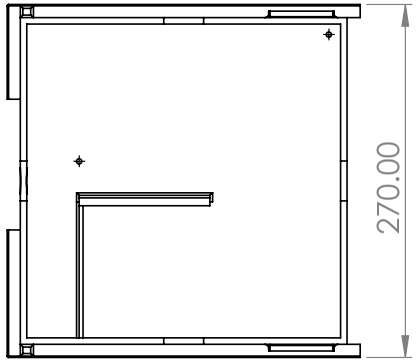
Grease Spray Holder

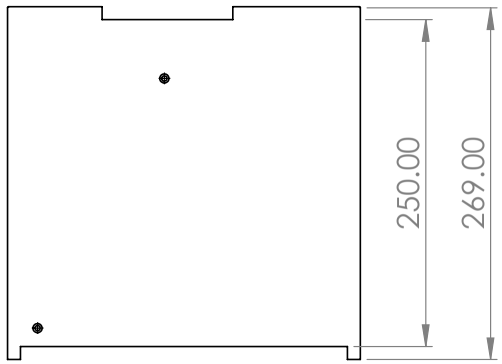


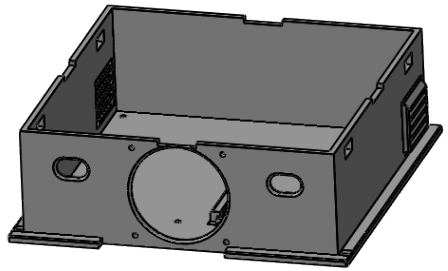

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | | FINISH: | | | DEBUR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING | REVISION |
| | | | | | | **Universitetet i Agder** | |
| | NAME | SIGNATURE | DATE | | | TITLE: **Grease Bottle Holder** | |
| DRAWN | Aleksander Watson | | 21.05.25 | | | Projection: | |
| CHK'D | Håvard K. Solheim | | 21.05.25 | | | | |
| APPV'D | Andreas C. Thorsen | | 21.05.25 | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: PLA+ | | DWG NO. **Grease Bottle Holder** | A4 |
| | | | | | | | |
| | | | | WEIGHT: KG | | SCALE:1:2 | SHEET 1 OF 1 |


12.83
3.00
2.00
61.00
45°
18.20
40.00
32.05
10.00
Ø3.10
21.00
17.89
30.00
41.00
45°
R4.00
73.00
40.00
10.00
15.18
7.52
6.06
2.00
57°
23.5°
55°
13.00
12.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBUR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN Aleksander Watson 21.05.25
CHK'D Håvard K. Solheim 21.05.25
APPV'D Andreas C. Thorsen 21.05.25
MFG
Q.A
MATERIAL:
ABS
WEIGHT: KG
Universitetet i Agder
DO NOT SCALE DRAWING
REVISION
TITLE:
Gripper
Projection:
DWG NO.
Gripper
A4
SCALE:1:1
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.



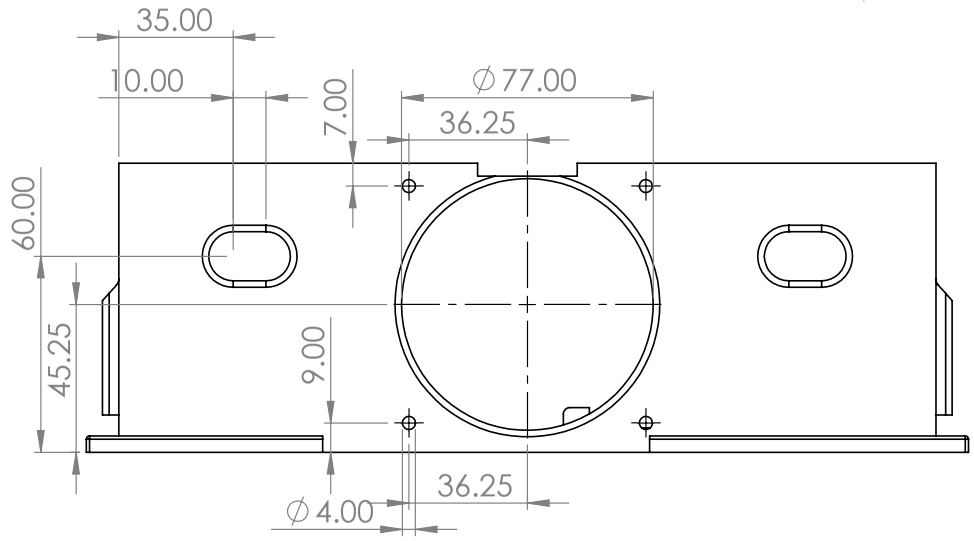



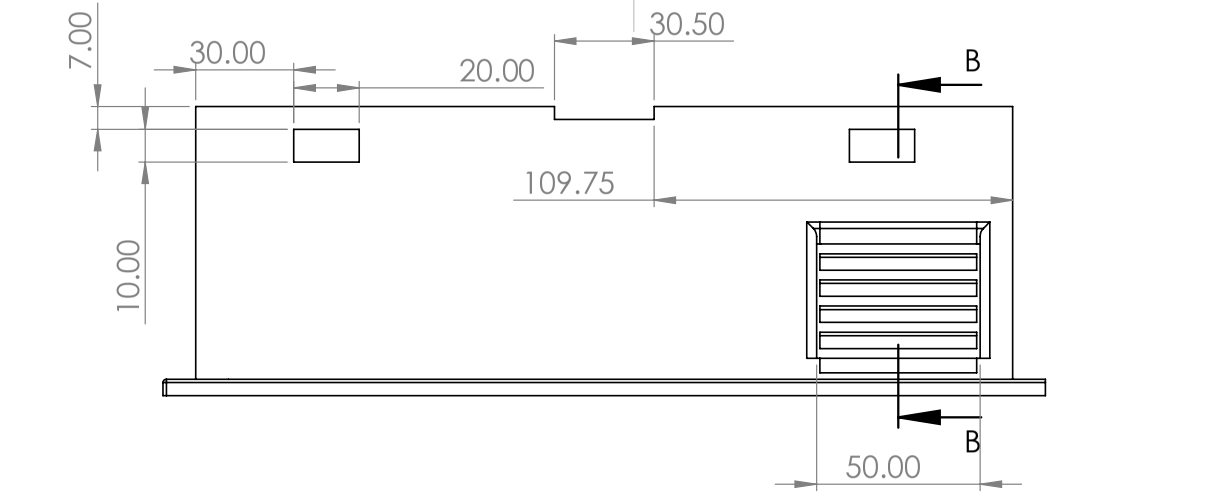



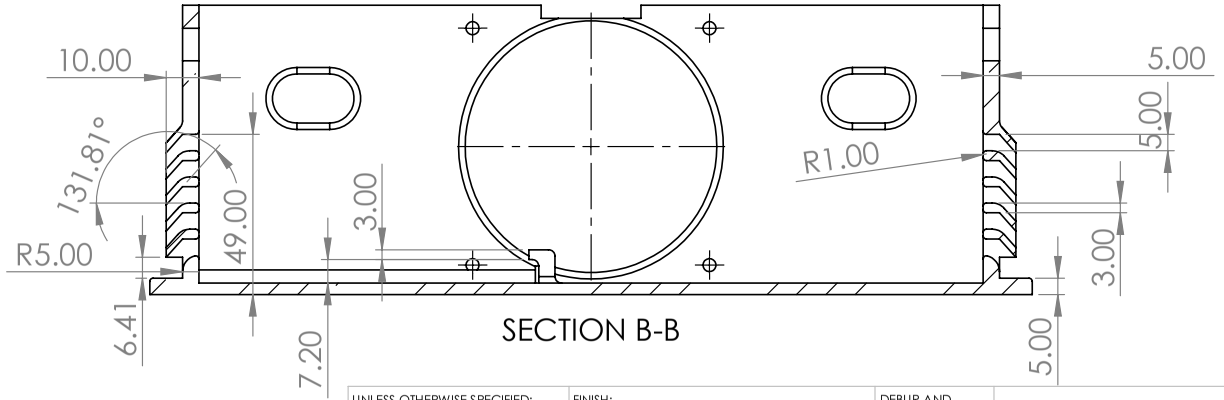
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR: | | | FINISH: | | | DEBUR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING |
| | NAME | SIGNATURE | DATE | | | | TITLE: Housing for Electroics |
| DRAWN | Aleksander Watson | | 21.05.25 | | | | Projection: |
| CHK'D | Håvard K. Solheim | | 21.05.25 | | | | |
| APPV'D | Andreas C. Thorsen | | 21.05.25 | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: PLA+ | | | DWG NO. Housing for Electroics |
| | | | | | | | A4 |
| | | | | WEIGHT: KG | | | SCALE:1:5 SHEET 1 OF 4 |

Universitetet i Agder



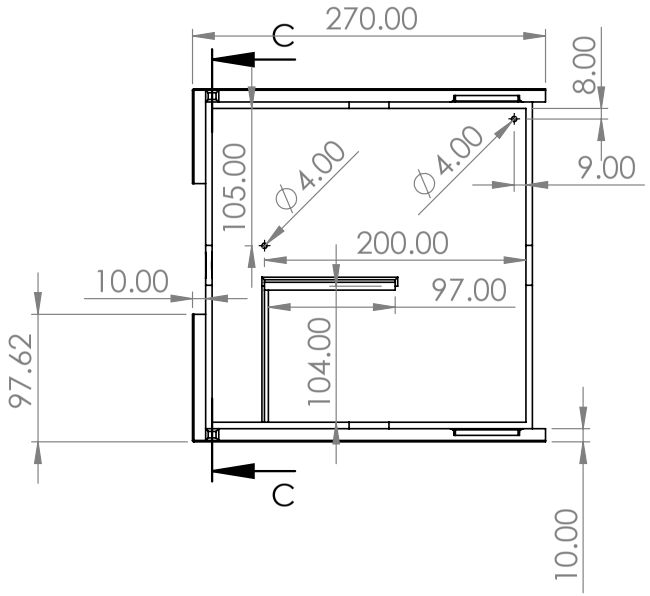

| | | | | | | |
|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | | | DEBUR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING / REVISION |
| | NAME | SIGNATURE | DATE | | | **Universitetet i Agder** |
| DRAWN | Aleksander Watson | | 21.05.25 | | | TITLE: **Housing for Electroics** |
| CHK'D | Håvard K. Solheim | | 21.05.25 | | | Projection: |
| APPV'D | Andreas C. Thorsen | | 21.05.25 | | | |
| MFG | | | | | | |
| Q.A | | | | MATERIAL: PLA+ | | DWG NO. **Housing for Electroics** / A4 |
| | | | | WEIGHT: KG | | SCALE:1:2 / SHEET 2 OF 4 |

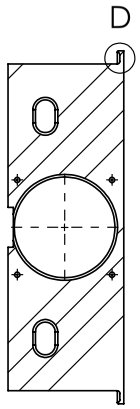


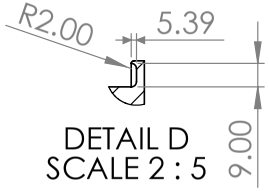

SECTION B-B

| | | | | | | |
|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | DEBUR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING | REVISION | |
| | NAME | SIGNATURE | DATE | **Universitetet i Agder** | | |
| DRAWN | Aleksander Watson | | 21.05.25 | TITLE: **Housing for Electroics** | | |
| CHK'D | Håvard K. Solheim | | 21.05.25 | Projection: | | |
| APPV'D | Andreas C. Thorsen | | 21.05.25 | | | |
| MFG | | | | | | |
| Q.A | | | | MATERIAL: PLA+ | DWG NO. **Housing for Electroics** | A4 |
| | | | | WEIGHT: KG | SCALE:1:2 | SHEET 3 OF 4 |



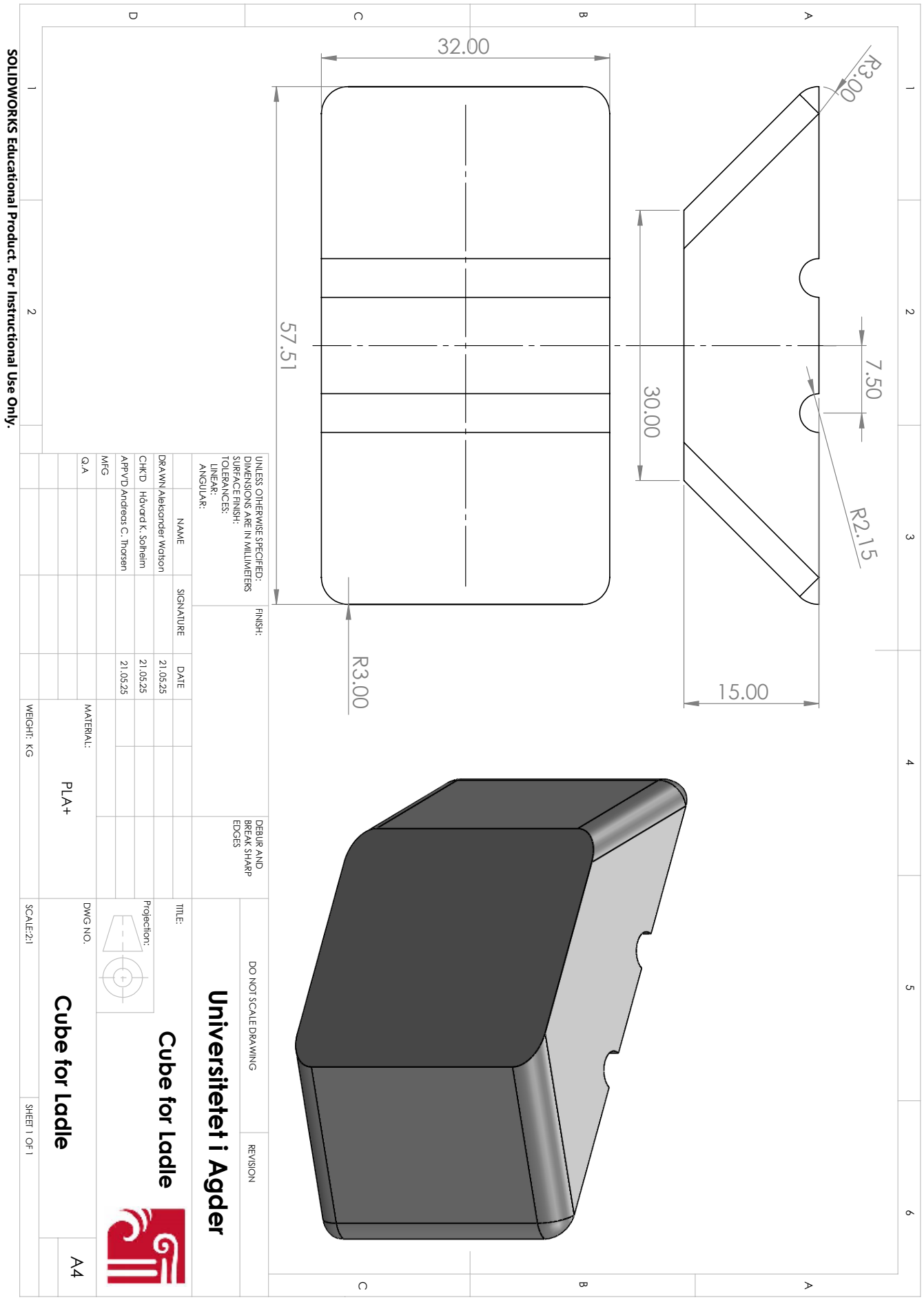

SECTION C-C



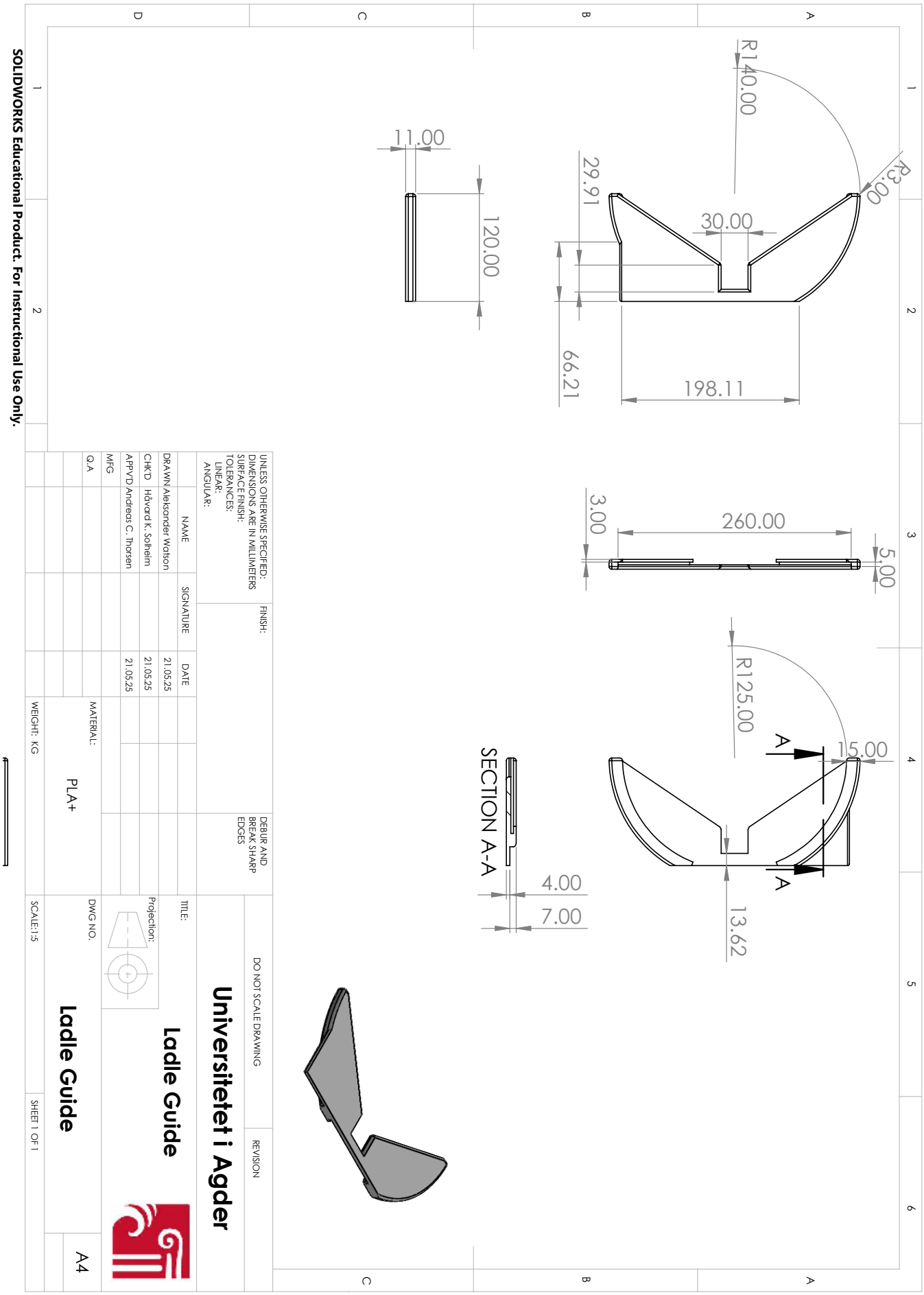

DETAIL D
SCALE 2 : 5

| | | | | | | |
|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | | FINISH: | | | DEBUR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING / REVISION |
| | | | | | | **Universitetet i Agder** |
| | NAME | SIGNATURE | DATE | | | TITLE: **Housing for Electroics** |
| DRAWN | Aleksander Watson | | 21.05.25 | | | Projection: |
| CHK'D | Håvard K. Solheim | | 21.05.25 | | | |
| APPV'D | Andreas C. Thorsen | | 21.05.25 | | | |
| MFG | | | | | | |
| Q.A | | | | MATERIAL: PLA+ | | DWG NO. **Housing for Electroics** A4 |
| | | | | WEIGHT: KG | | SCALE:1:5 SHEET 4 OF 4 |

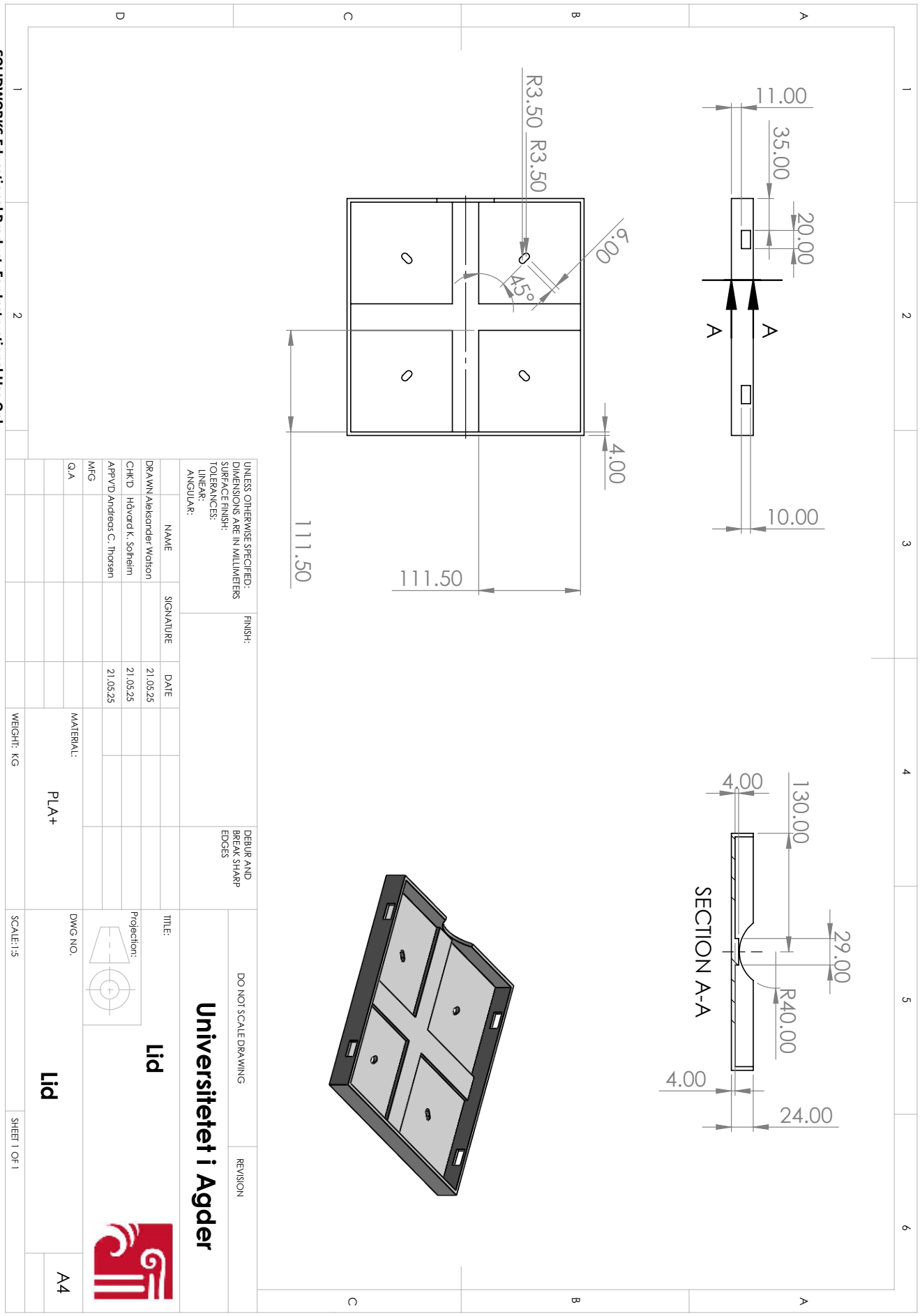
# Interface Cube for Ladle



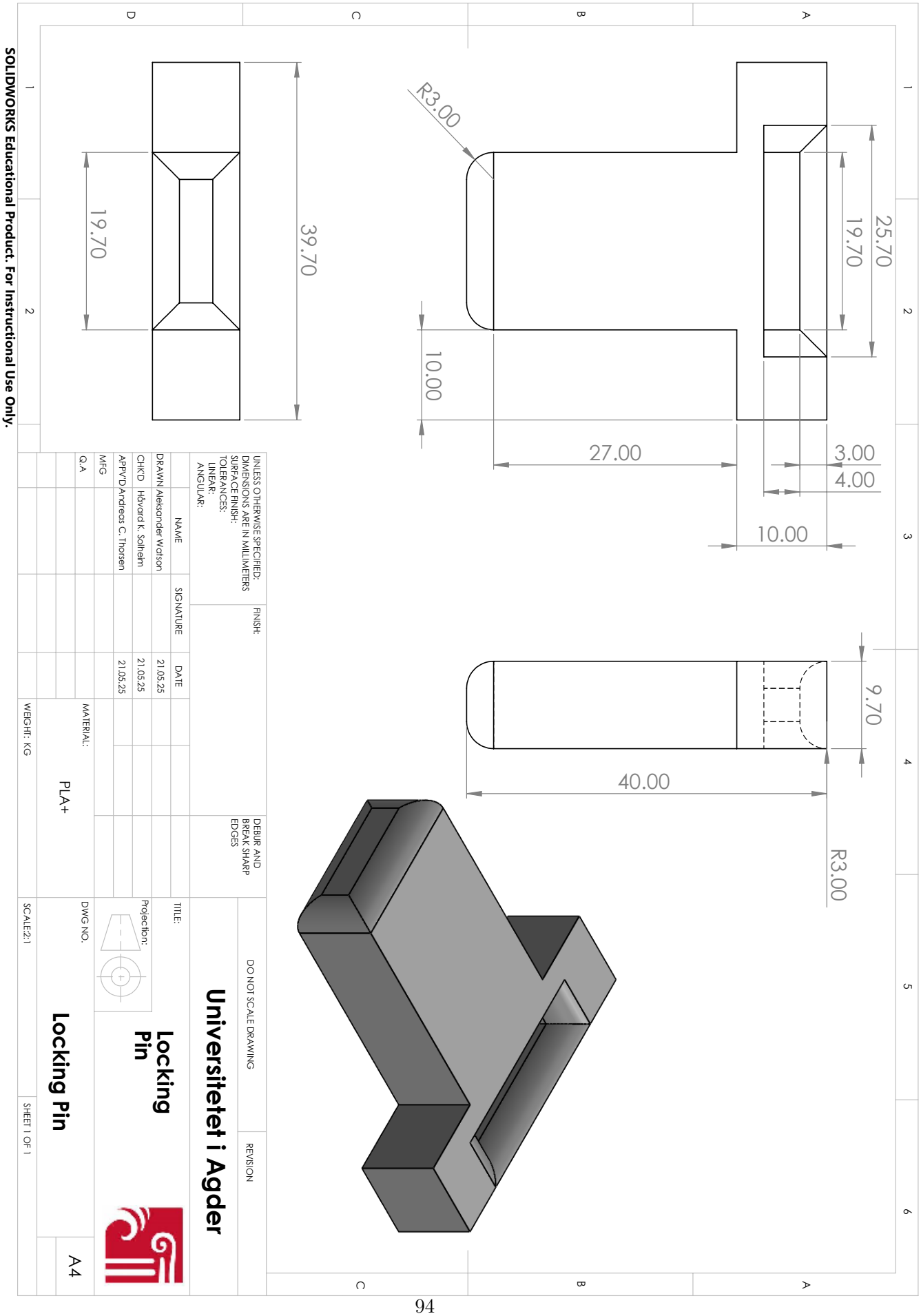
# Ladle Guide


R140.00
R3.00
30.00
29.91
66.21
198.11
11.00
120.00
3.00
260.00
5.00
R125.00
15.00
A
A
SECTION A-A
4.00
7.00
13.62
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBUR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN Aleksander Watson
21.05.25
CHK'D Håvard K. Solheim
21.05.25
APPV'D Andreas C. Thorsen
21.05.25
MFG
Q.A
MATERIAL:
PLA+
WEIGHT: KG
DO NOT SCALE DRAWING
REVISION
Universitetet i Agder
TITLE:
Ladle Guide
Projection:
DWG NO.
Ladle Guide
A4
SCALE:1:5
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

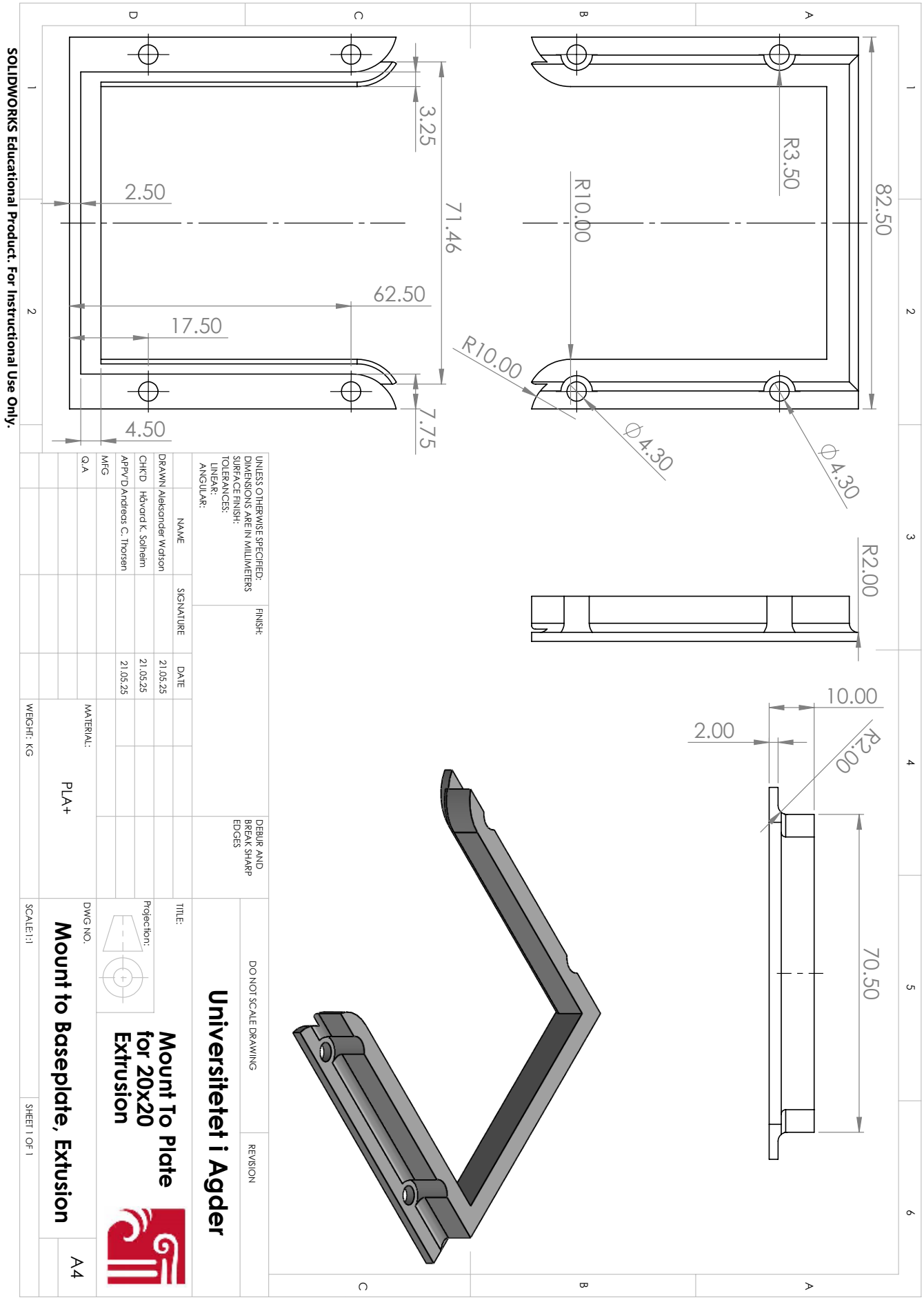
# Lid Final


11.00
35.00
20.00
10.00
A
A
R3.50
R3.50
9.00
45°
4.00
111.50
111.50
4.00
130.00
29.00
R40.00
24.00
SECTION A-A
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBUR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN Aleksander Watson
21.05.25
CHK'D Håvard K. Solheim
21.05.25
APPV'D Andreas C. Thorsen
21.05.25
MFG
Q.A
MATERIAL:
PLA+
WEIGHT: KG
DO NOT SCALE DRAWING
REVISION
Universitetet i Agder
TITLE:
Lid
Projection:
DWG NO.
Lid
A4
SCALE:1:5
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

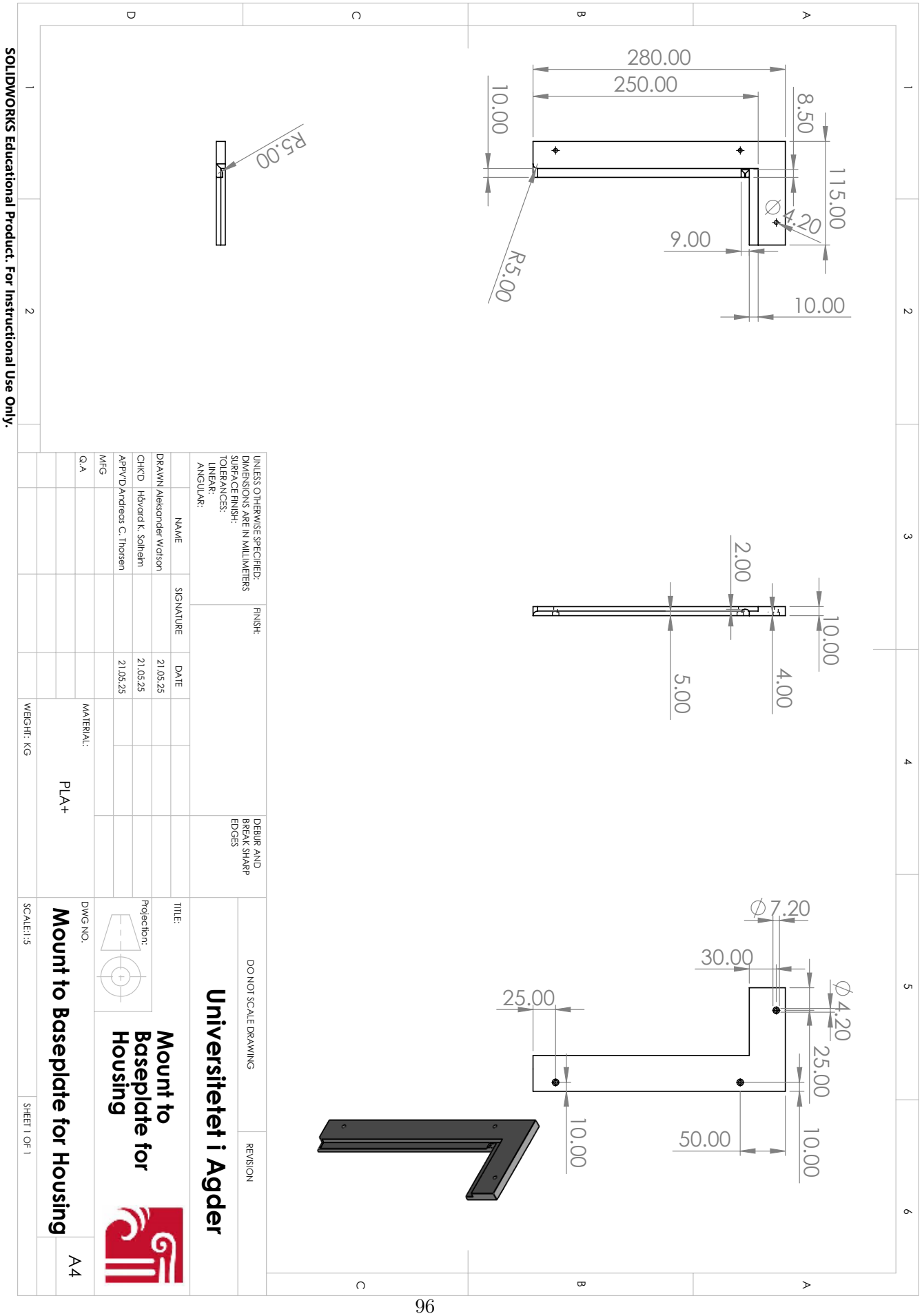
# Locking Pin


R3.00
25.70
19.70
10.00
27.00
3.00
4.00
10.00
39.70
19.70
9.70
40.00
R3.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBUR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN Aleksander Watson 21.05.25
CHK'D Håvard K. Solheim 21.05.25
APPV'D Andreas C. Thorsen 21.05.25
MFG
Q.A
MATERIAL:
PLA+
WEIGHT: KG
DO NOT SCALE DRAWING
REVISION
Universitetet i Agder
TITLE:
Locking Pin
Projection:
DWG NO.
Locking Pin
A4
SCALE:2:1
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

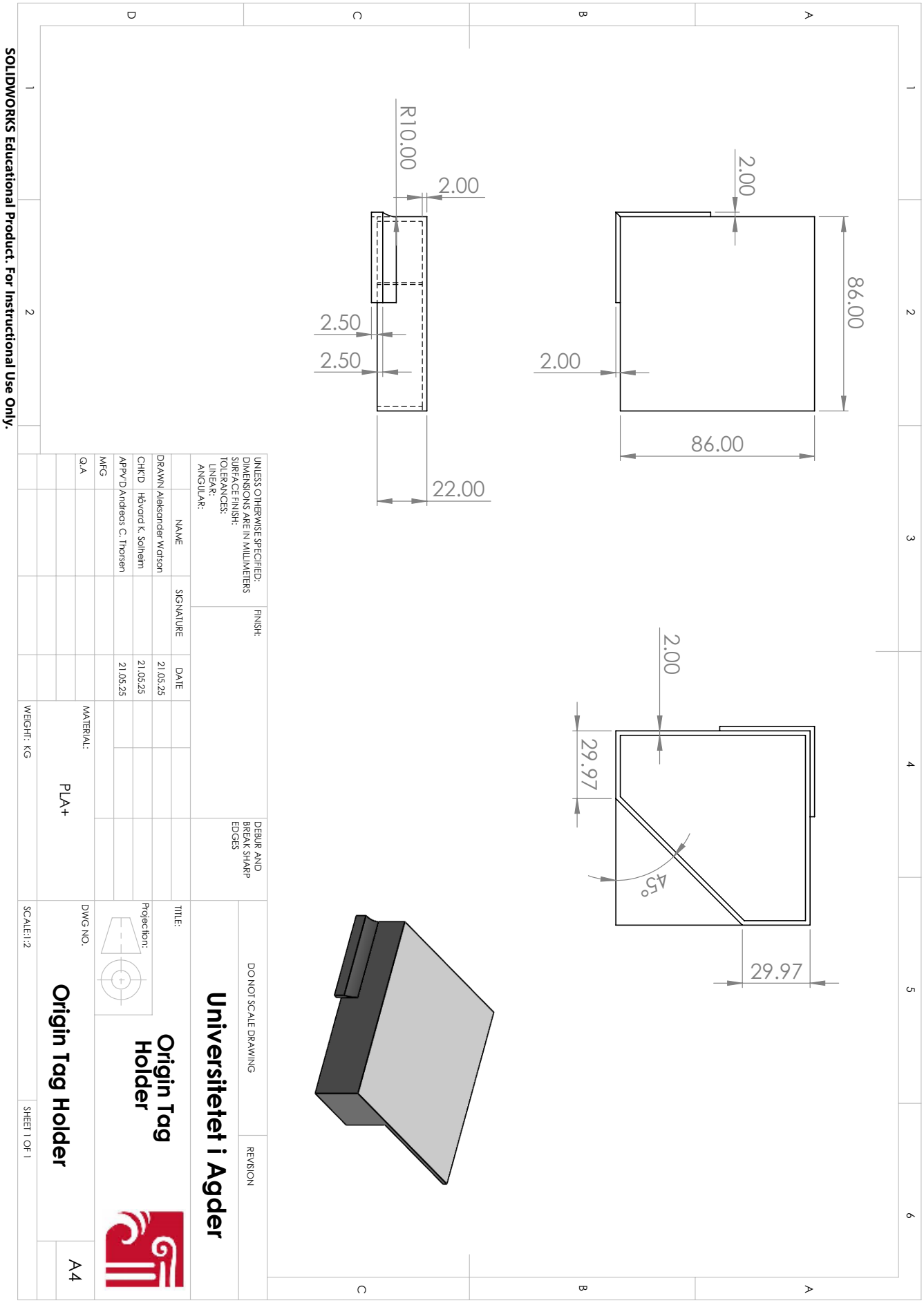
# Mount to Baseplate for 20x20 Extrusion

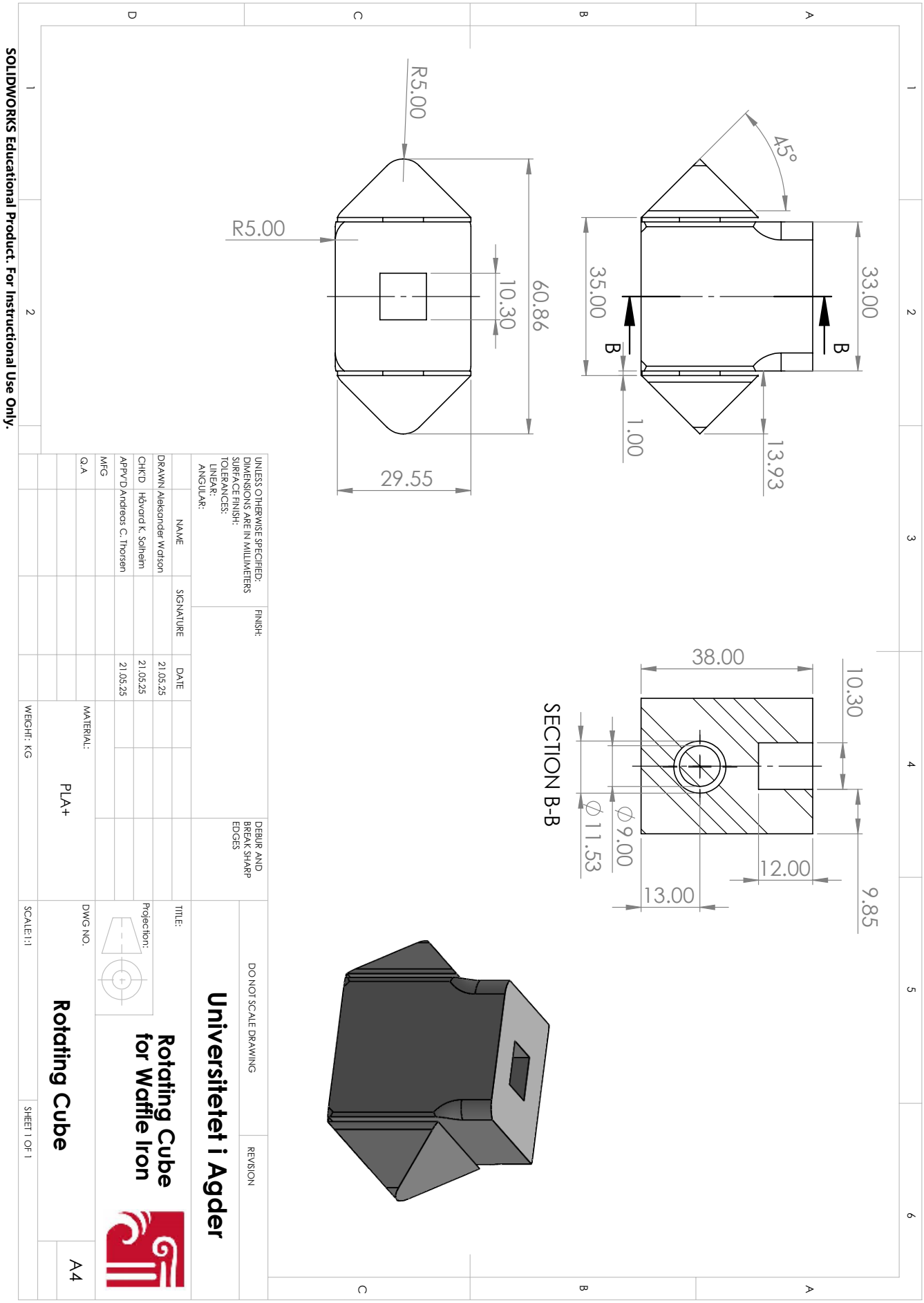


# Mount to Baseplate for Housing

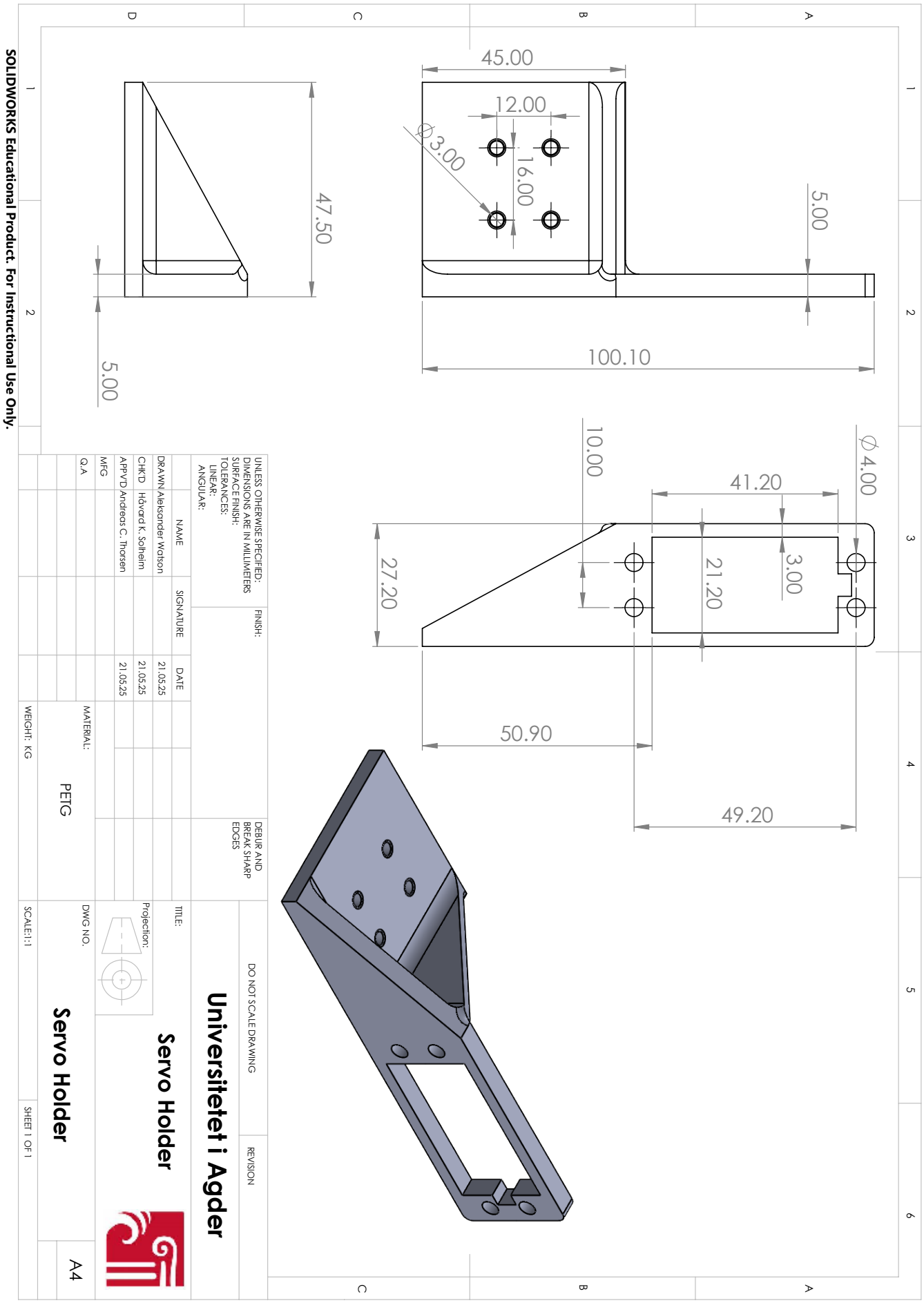


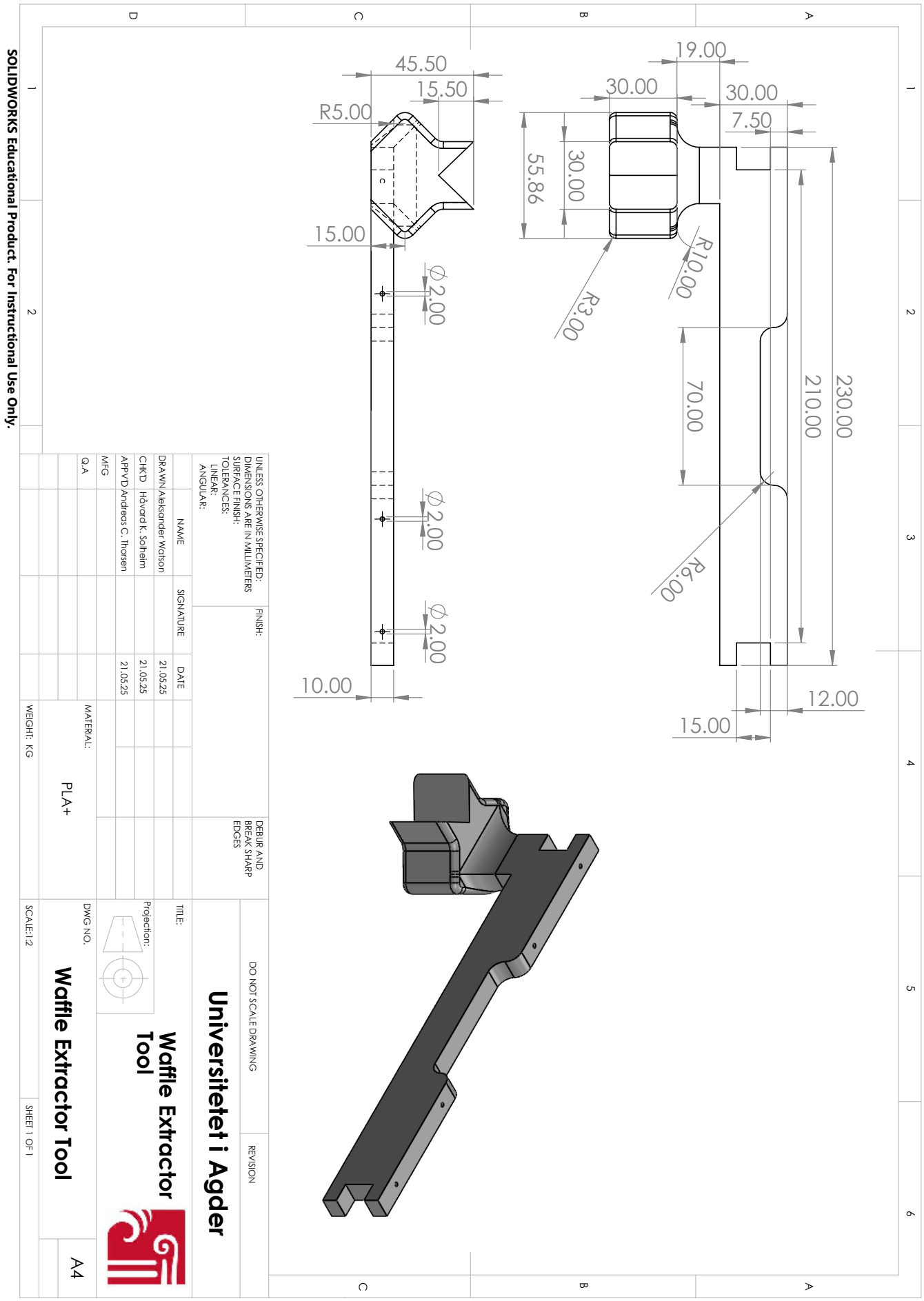
# Origin Tag Holder



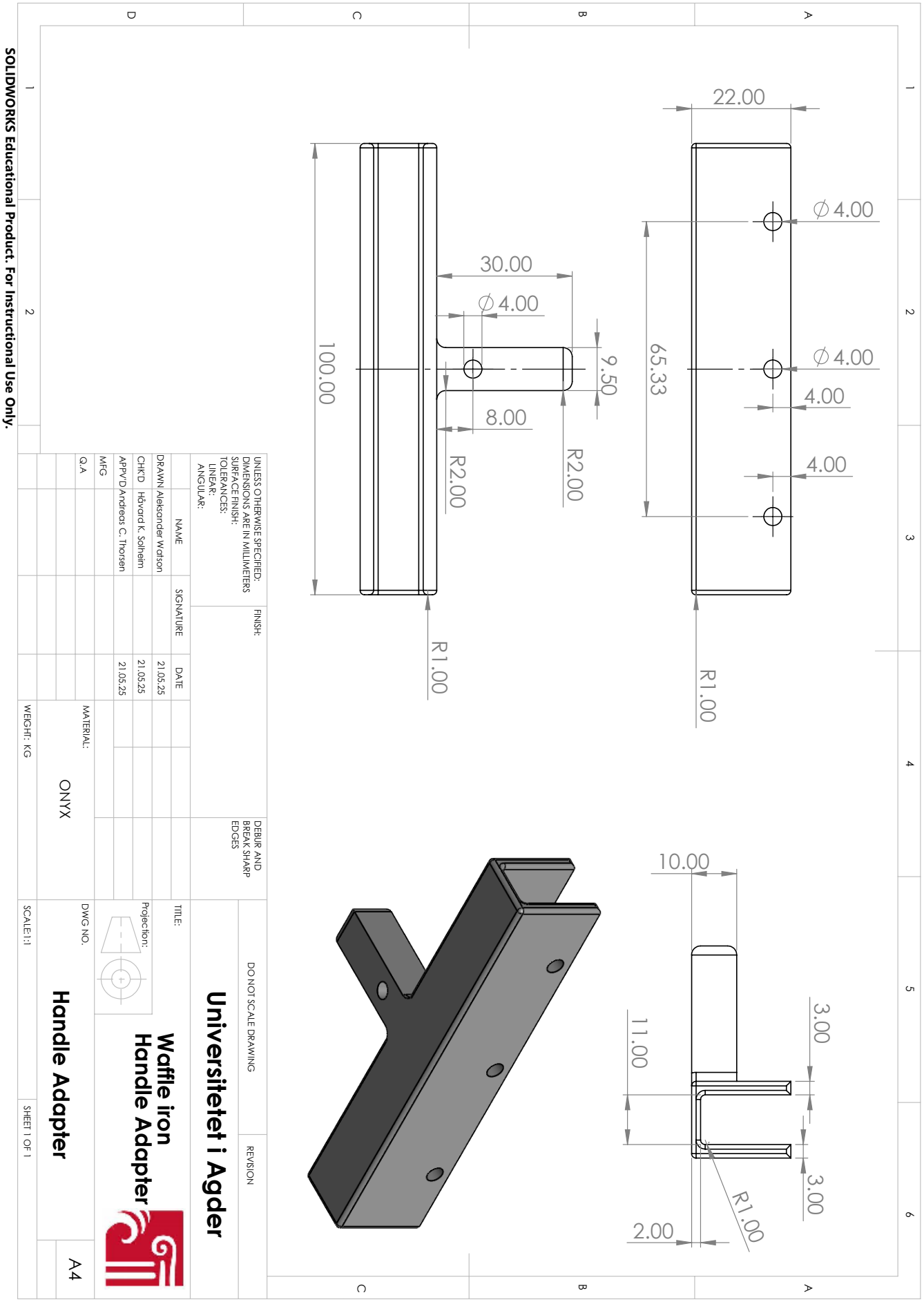

R5.00
R5.00
60.86
10.30
29.55
45°
33.00
35.00
B
B
1.00
13.93
38.00
10.30
Ø9.00
Ø11.53
12.00
13.00
9.85
SECTION B-B
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBUR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN Aleksander Watson 21.05.25
CHK'D Håvard K. Solheim 21.05.25
APPV'D Andreas C. Thorsen 21.05.25
MFG
Q.A
MATERIAL:
PLA+
WEIGHT: KG
Universitetet i Agder
DO NOT SCALE DRAWING
REVISION
TITLE:
Rotating Cube for Waffle Iron
Projection:
DWG NO.
Rotating Cube
A4
SCALE:1:1
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

# Servoholder

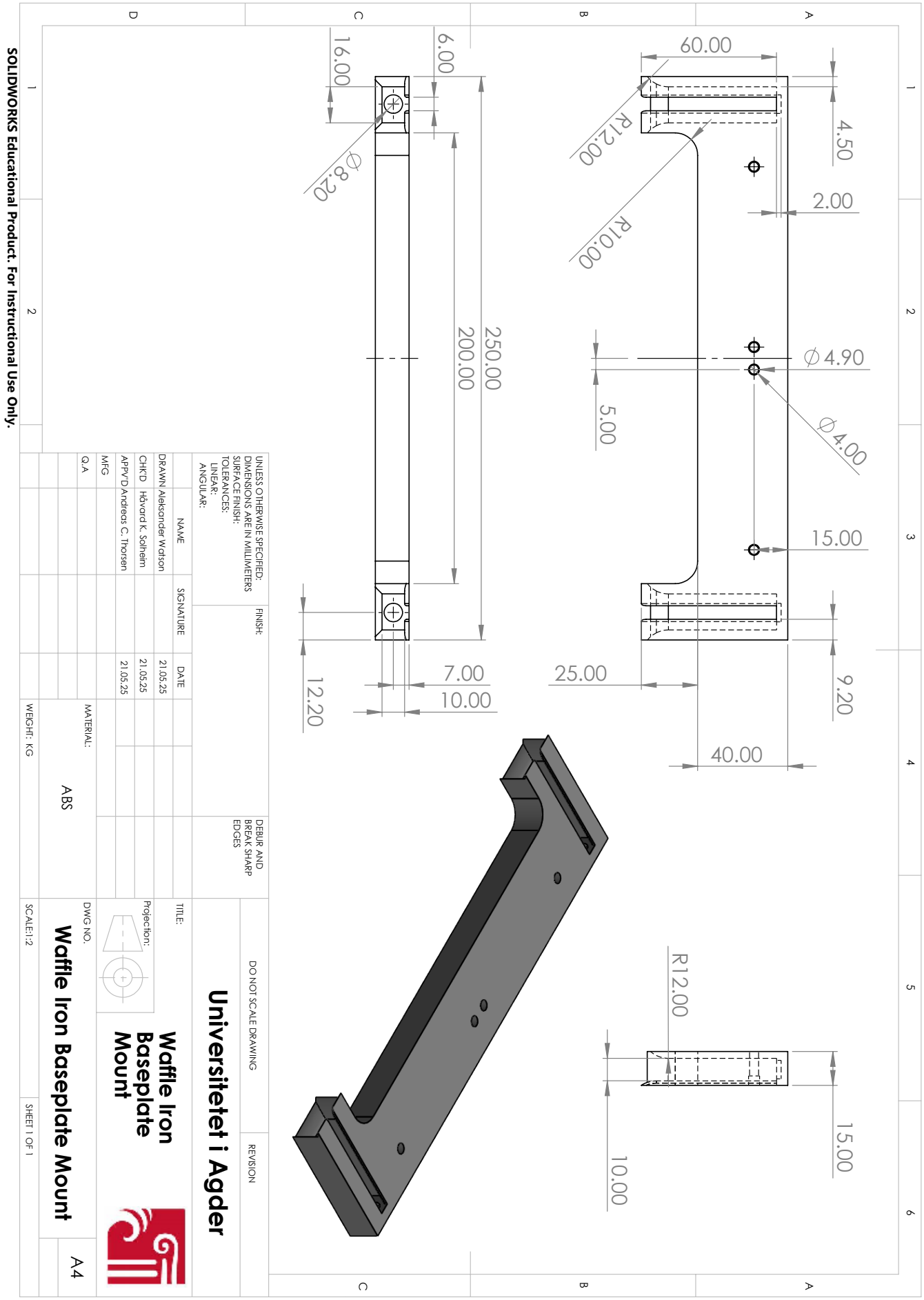



19.00
30.00
30.00
7.50
55.86
30.00
R10.00
R3.00
230.00
210.00
70.00
R9.00
12.00
15.00
45.50
15.50
R5.00
15.00
Ø2.00
Ø2.00
Ø2.00
10.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBUR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN Aleksander Watson 21.05.25
CHK'D Håvard K. Solheim 21.05.25
APPV'D Andreas C. Thorsen 21.05.25
MFG
Q.A
MATERIAL:
PLA+
WEIGHT: KG
DO NOT SCALE DRAWING
REVISION
Universitetet i Agder
TITLE:
Waffle Extractor Tool
Projection:
DWG NO.
Waffle Extractor Tool
A4
SCALE:1:2
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

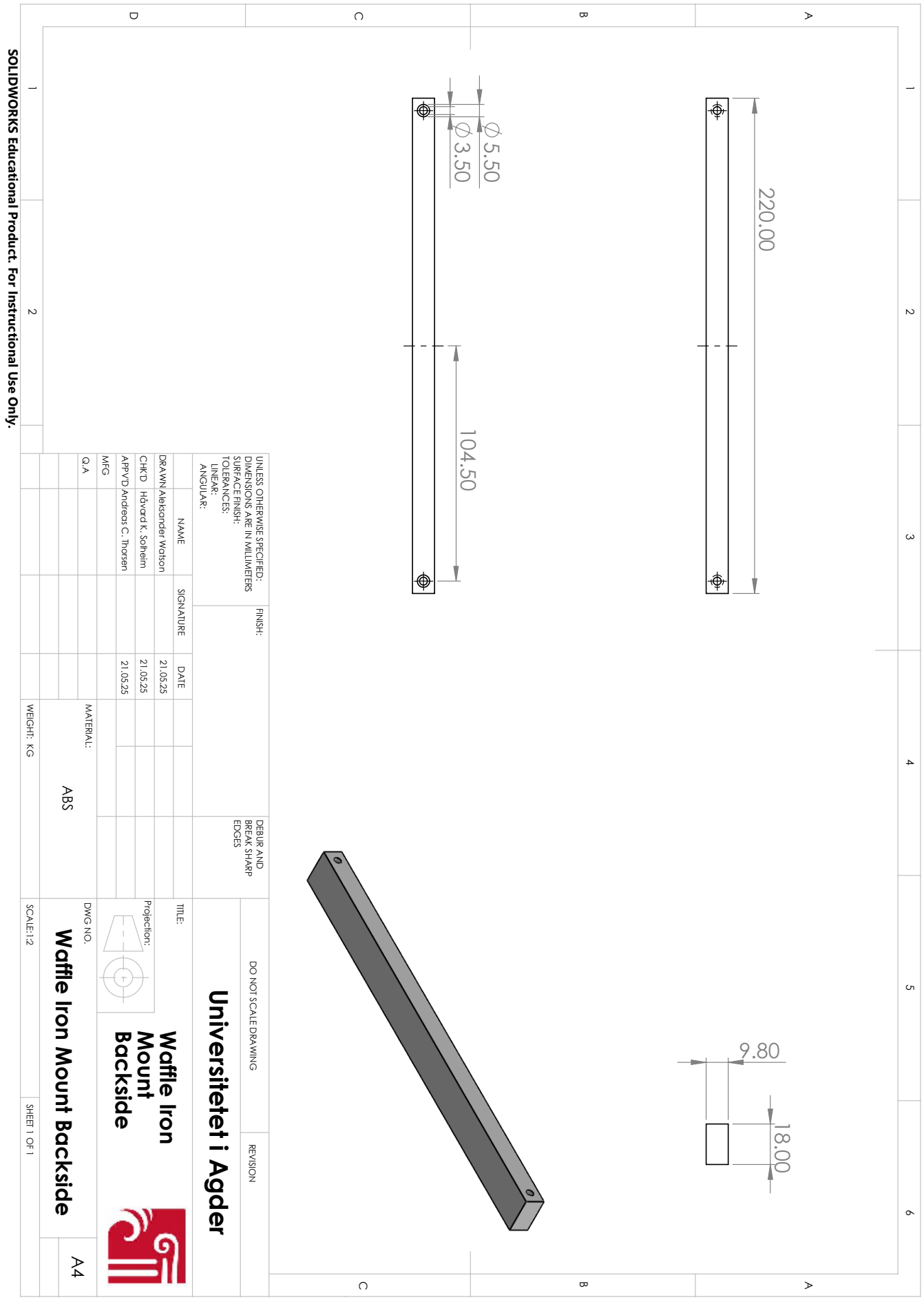
# Waffle Handle Adapter



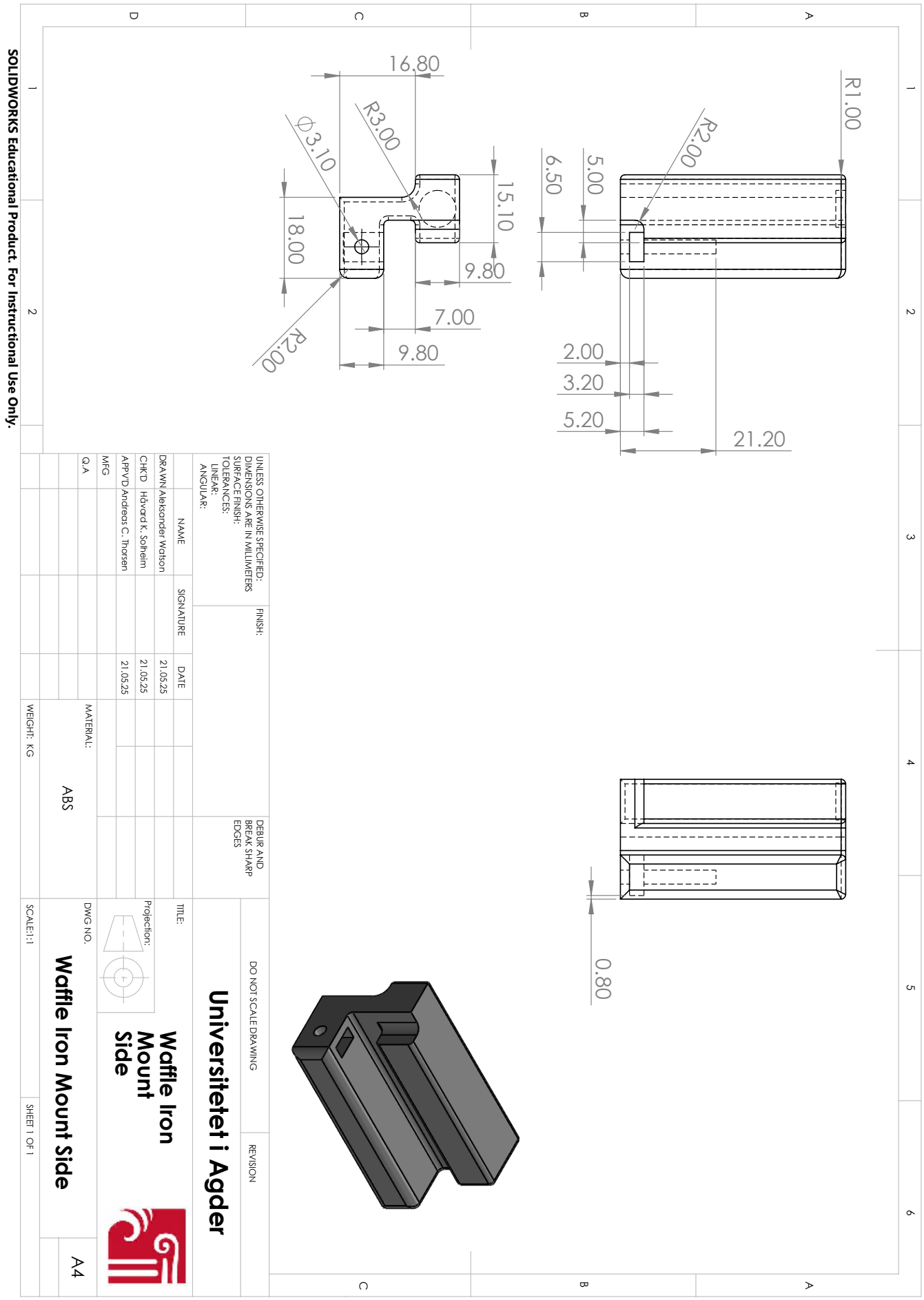
# Waffle Iron Baseplase Mount



# Waffle Iron Mount Backside



# Waffle Iron Mount Side



# Appendix C

# Use of Artificial Intelligence

This report makes use of AI for language editing and improving the structure of the text. The use of artificial intelligence follows the University of Agder's guidelines for AI in academic writing. The guidelines permits such tools to be used for language and grammar enhancement and structural assistance, provided the use is clearly stated. All academic content and analysis have been independently developed.