Summer intern project

AUTOMAZE

August 7, 2023

Lyngstad, Vebjørn Chabab, Simon Tran, Emil D.
Kokai, Daniel

Contents

1 Introduction

The maze project concerns upgrading and automating the old labyrinth toy concept. By controlling the pitch and roll axis of the board, Maneuvering the ball around the maze could be achieved. The project involves three main disciplines such as;

- **Mechanical Design:** Design the system, compile the assembly and customize parts to fulfil the requirements. In Figure 1.1, the final design of the system is seen, laying the groundwork for its future development. In appendix B a list of parts is provided for all components designed related to the project. Most of the parts are manufactured by 3D-printing, and some are bought by external suppliers.
- **Programming:** Programming how the hardware should work to ensure precise movement of the maze, and make a GUI to easily shift between the different control styles.
- **Vision Technology:** Utilize vision to recognize mazes and solve different maze designs.

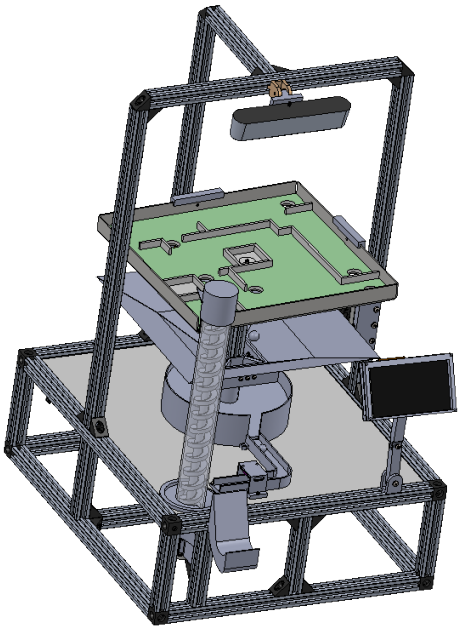


Figure 1.1: Assembly of the system.

The goal with the project is to have three different game modes. The first one is a manual control of the system by using a Xbox controller. This step will serve as the foundation for the subsequent advancements. The second mode is a automatic mazesolver based on timestamped actuator movement. The last mode will implement artificial intelligence (AI) to the system, using vision based machine learning algorithm to solve the maze layouts automatically.

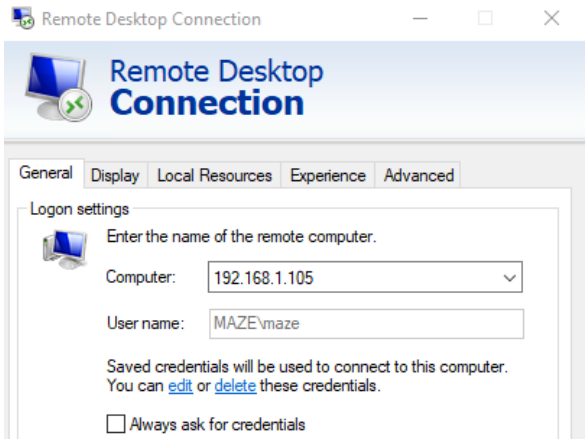
2 User Manual

In this section we will go through how to log into the Jetson Nano, set up remote access, navigating the GUI and know issues.

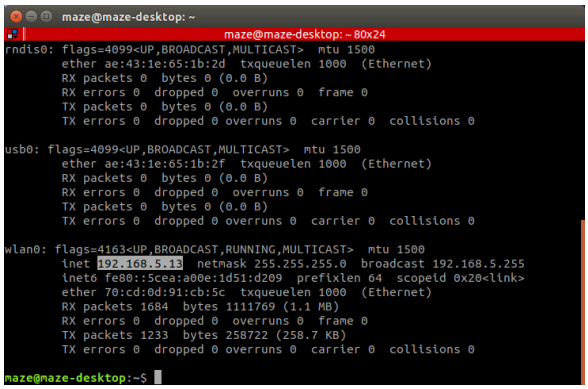
2.1 Jetson Nano

Username: maze
Password: 123

The jetson nano can also be accessed using a remote desktop connection. If using windows Remote Desktop Connection you will need the WiFi IP-address, this can be found calling the command "ifconfig" in a terminal on the jetson nano. Figure 2.1b shows the output in the terminal after calling "ifconfig".



(a) Remote accsess using windows Remote Desktop Connection.

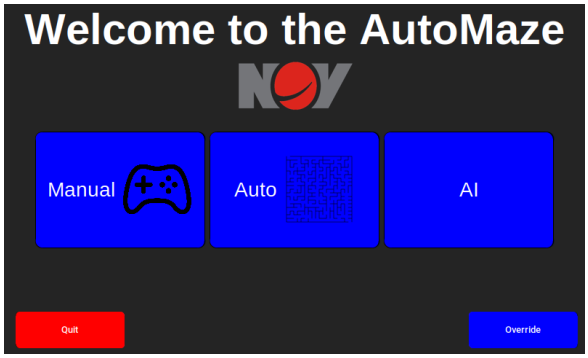


(b) The terminal after calling "ifconfig"

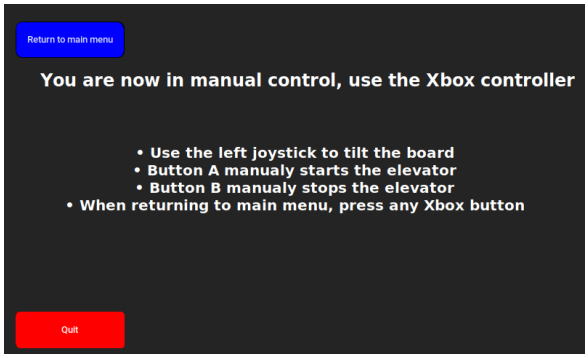
Figure 2.1: Remote Accsess

This project also utilizes the "Startup applications" on the Jetson nano. This is used to launch the GUI-application on boot of the Jetson. For the GUI to run it needs to be connected to a serial port (Arduino) and the Xbox controller thru Bluetooth. It seems to be a small delay (About 10 seconds) in the startup of the serial connection, therefore the GUI needs a small delay before the connection is initialized. This is only applicable when launching the GUI from boot.

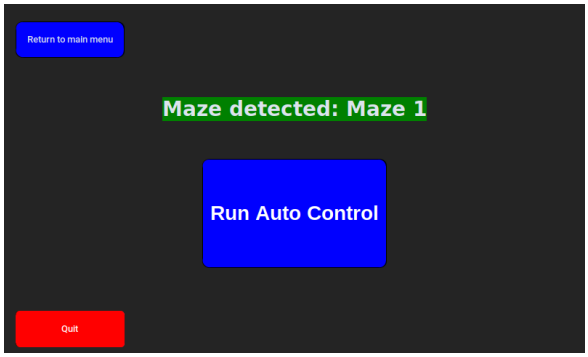
2.2 Navigating the GUI



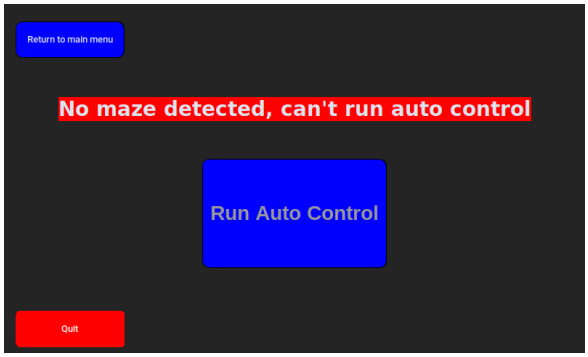
(a) Main menu



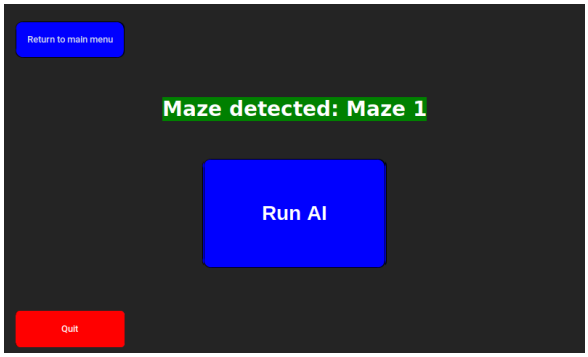
(b) Manual control



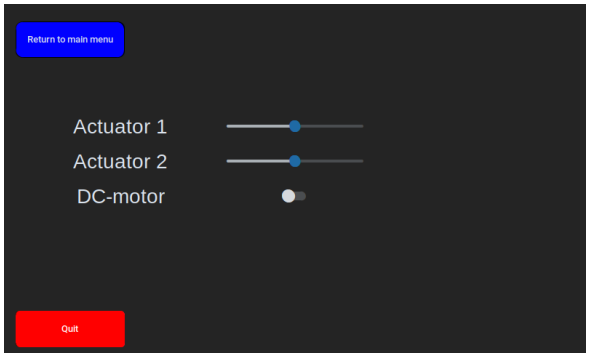
(c) Auto control, maze found



(d) Auto control, maze not found



(e) AI - To be implemented in the future.



(f) Override function.

Figure 2.2: Showcase of the GUI.

After starting the Jetson Nano, the Main menu will show up, shown in figure 2.2a. From there, you can click/push on five buttons to access the manual, auto, AI, override, or quit modes.

Manual Control (figure 2.2b): Ensure the Xbox controller is on and paired with the Nano (the Xbox controller LED is steady and not blinking). This window provides instructions on how to tilt the board and control the elevator using the Xbox controller. To return to the main menu, click/push the "return to menu" button. Afterward, You need to give an input from the Xbox controller to go back to the main menu.

Auto Control (figure 2.2c and figure 2.2d:) If a maze is correctly placed (start point in the bottom-left corner), it will be displayed in green to indicate the selected maze. Click/push the "Run Auto Control" button to let the program solve the maze automatically. If no maze is detected, the "Run Auto Control" button will be disabled. You can return to the main menu by clicking/pushing the "return to menu" button.

AI Control (figure 2.2e): By clicking/pushing the "AI" button, you can see which maze is selected. If a maze is detected, the "Run AI" button will enable, allowing you to run the AI to solve it. If there is no maze detected, the "Run AI" button will be disabled. Click/push the "Run AI" button to return to the main menu.

Override Control (figure 2.2f): In this mode, you can control two actuators (Actuator 1 for the y-axis and Actuator 2 for the x-axis) using sliders. Additionally, you can turn the DC-motor/elevator on and off.

Quit: If you click/push the "Quit" button, the GUI program will stop running.

2.3 Known issues

- When clicking "the return to menu" button in manual control, then it won't go back before after getting a new input from the Xbox controller
- The GUI can open with wrong graphics, seems only to be happening when the program is running on the jetson nano.
- The actuators can go in positions that they isn't suppose to be in (when in main menu/starting up manual control)
- The different mazes don't fit exactly into the holder(there is some wiggle room). This can lead to the auto control not being able to solve it.
- The sensor is not perfectly calibrated, this can lead to the elevator going of or not go of when it is suppose to it(happen less then 1 prosent of the time).
- The ball can go outside the top drain

3 Setup

In this section we will go through how the Arduino is connected to the actuators, DC-motor and sensor, Configuring Linear Actuator Control Boards and the technical design.

3.1 Arduino Circuit

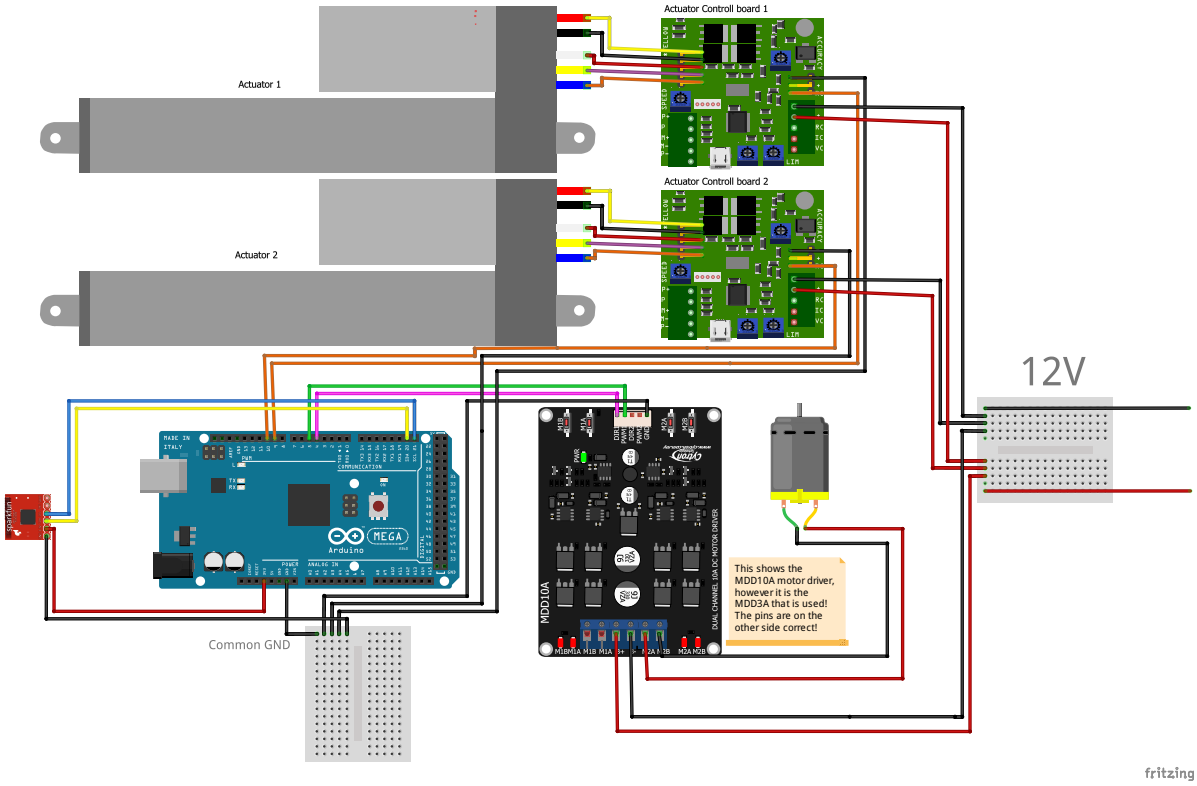


Figure 3.1: Wire Diagram for Arduino MEGA 2560 rev3.

Figure 3.1 shows the complete Arduino circuit in the project. It shows how the actuators, DC motor and proximity sensor is connected. The proximity sensor is powerd directly from the Ardino, using 3.3V. Both actuators and DC motor requires 12V and therefore need a separate power supply. The Arduino is powered thru the USB port witch also works as serial communication with the Jetson Nano. Thru this port the Python script running on the Jetson Nano can send data to the Arduino witch then controls the actuators or the DC motor.

The sensor is utilizing the I^2C bus of the microcontroller, this is corresponding to the SDA and SCL port on the Arduino. The actuators are getting RC signals from the Arduino and behave just like a servo. This makes it possible to write values between 1000 and 2000 microseconds to the actuator, where 1000 microseconds corresponds to a fully retracted actuator and 2000 microseconds corresponds to a fully extended actuator. On the Arduino the two linear actuators are connected to PWM pins 9 and 10. The DC

motor is using two PWM signals (Port 4 and 5 on the Arduino). All the components connected to the Arduino will need a common ground to function properly.

3.2 Configuring Linear Actuator Control Boards

The Linear Actuator Control (LAC) Boards from Actuonix came in the box with a preset that was unusable due to the actuators constantly overshooting the set value. This was then tried to fix by turning the four blue configure screws on the board, but this did not give any consistent results.

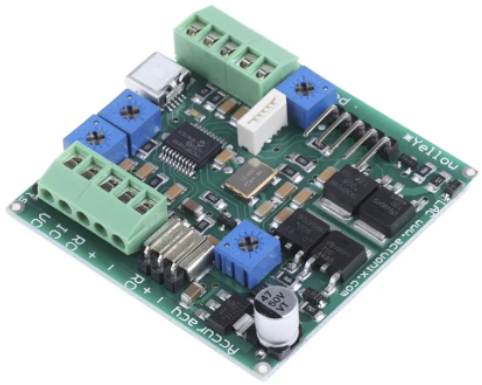


Figure 3.2: Actuonix, Linear Actuator Control Board.

It was a time consuming job to get the actuators to be accurate enough and at the same time not overshooting. After a lot of trial and error in the configure tool, the settings shown in figure 3.3 were used in the project.

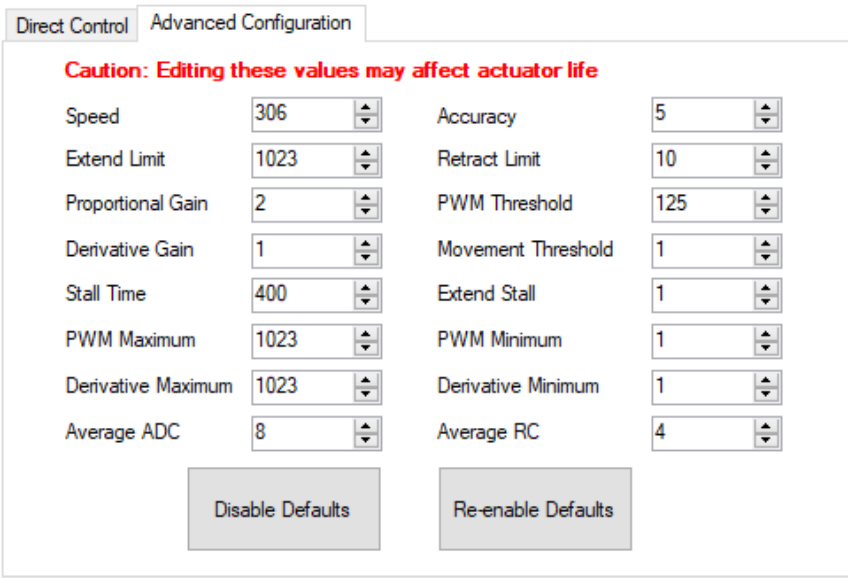


Figure 3.3: Settings used for the LAC Boards.

For more detailed information about the configure software check out USB Control and Configuration of the LAC.

3.3 Implementation of Machine Vision

To enable the system to identify the currently loaded maze, machine vision was implemented, which relies on the utilization of the advanced Zed 2 camera from stereo labs. This camera offers exceptional capabilities, boasting a wide range of applications due to its impressive field of view. The camera is mounted about 20cm above the gametray, so the wide field of view came in handy.

Figure 3.4: Zed 2 by Stereo Labs

3.4 Computer Vision

3.4.1 Template Matching

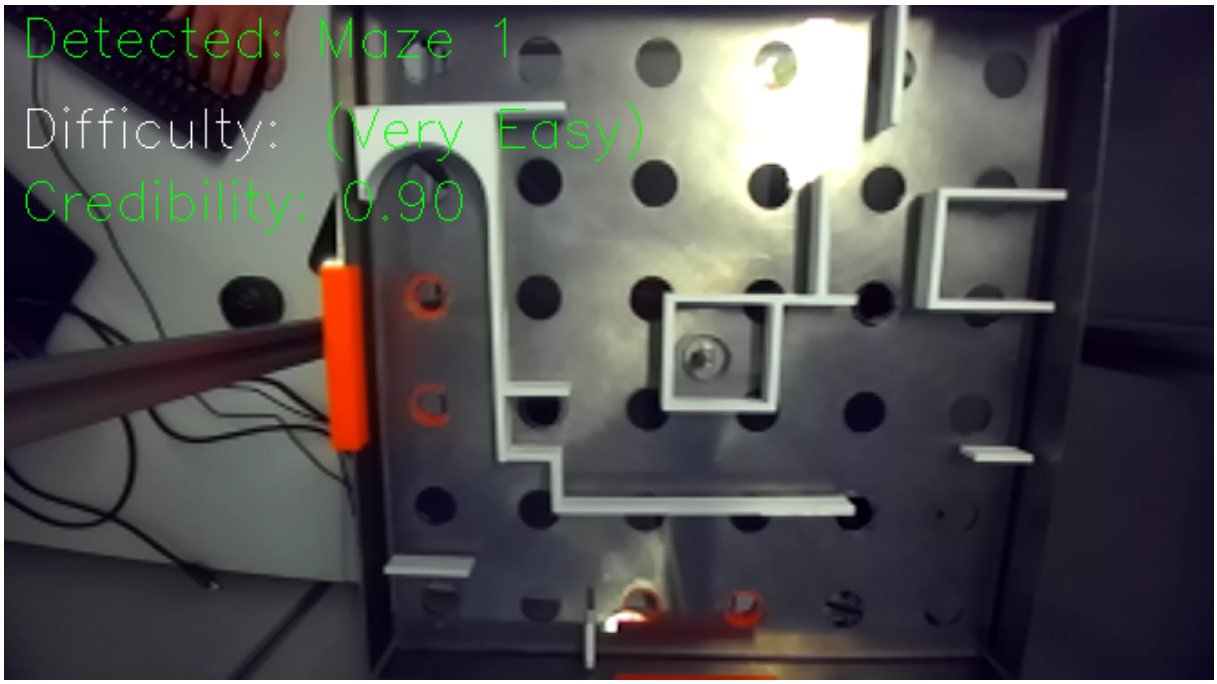


Figure 3.5: Camera Detection

The three different mazes are stored in the script as images and the camera captures live footage. OpenCV is used to initiate template matching by comparing the live footage from the camera, with the three different images of the mazes. The observation the camera makes is then evaluated and a score of the credibility will be printed. If the credibility score is above 70%, the text will be green to indicate that the camera is quite confident. By experimenting with different lightning and camera setting, we discovered that the credibility will be drastically affected when the lights settings are changed. Updating the template images of the mazes continuously will ensure that the current light settings are up to date and the template matching as credible as possible.

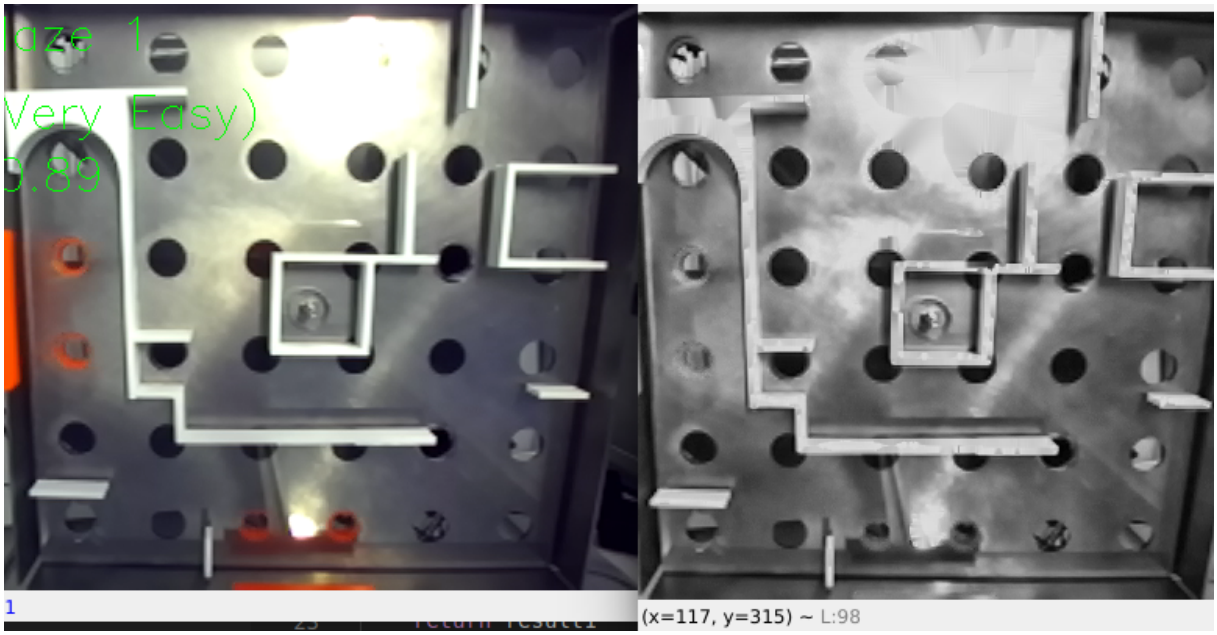


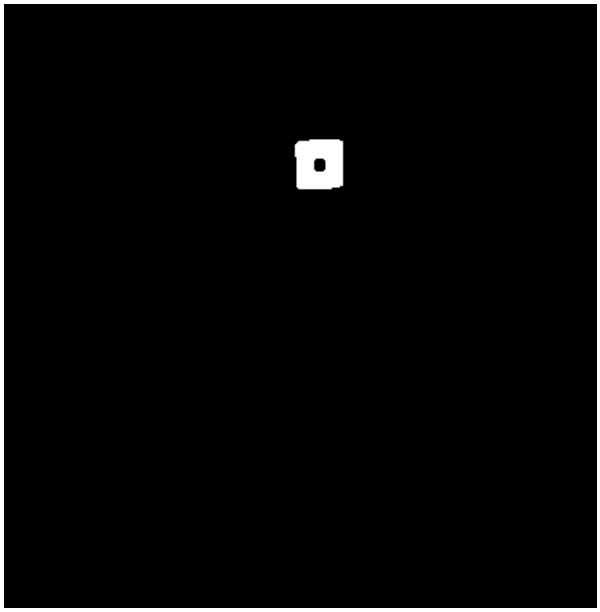
Figure 3.6: Grayscale Videocapture

In the top right corner of figure 3.5 glare appears due to lights from the ceiling, this will disturb the template matching, so an improvement is to apply a clahe filter. Figure 3.6 shows the grayscale format of the videocapture which the script uses as source for the template matching. By comparing the left and right side, you can see that the glare in the top right is slightly improved visually, and this will increase the credibility of the template matching.

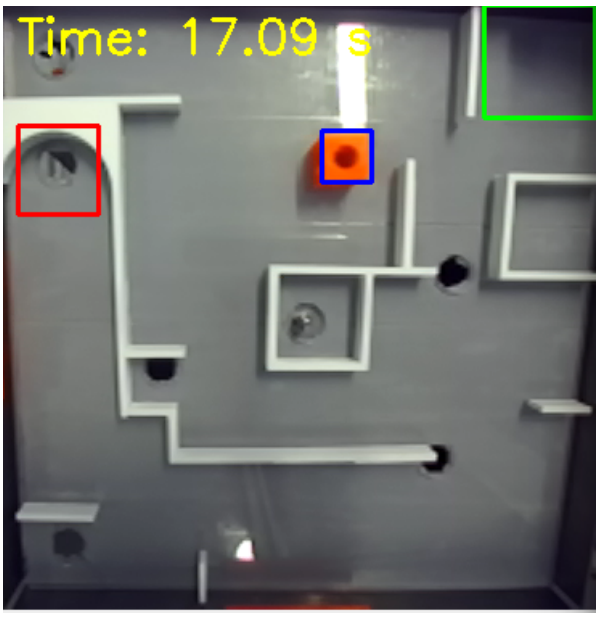
3.4.2 Time Tracking

By filtering the video capture, its possible to separate distinct colors from white and black. The result of this filtering is shown in 3.7a. By filtering the footage, the camera can easily detect the object of interest if the color is distinct. When the object is detected, there will be drawn a blue rectangle around the object. The start position is located at the bottom left corner for the operator of the system when facing the HMI Display, due to how the camera is mounted the capture is inverted, so the start point for the camera is in the top right corner. The start position is however indicated by a green rectangle, and the goal is indicated by a red rectangle see figure 3.7b.

The timer is programmed to be initiated when the blue rectangle is leaving or no longer intersecting with the start point (Green rectangle), and the timer will keep counting until the blue rectangle intersects with the goal (Red rectangle). If the ball falls into a hole, or you physically remove the ball with your hand, the timer will keep counting, the only factor that interferes with the timer is the entrance of the blue rectangle to the goal area. The goal area had to be more narrow than the walls of the maze close to the goal area, so that any interactions with the walls not would manipulate the timer and print the elapsed time.



(a) Computer Vision of the Color Filtering

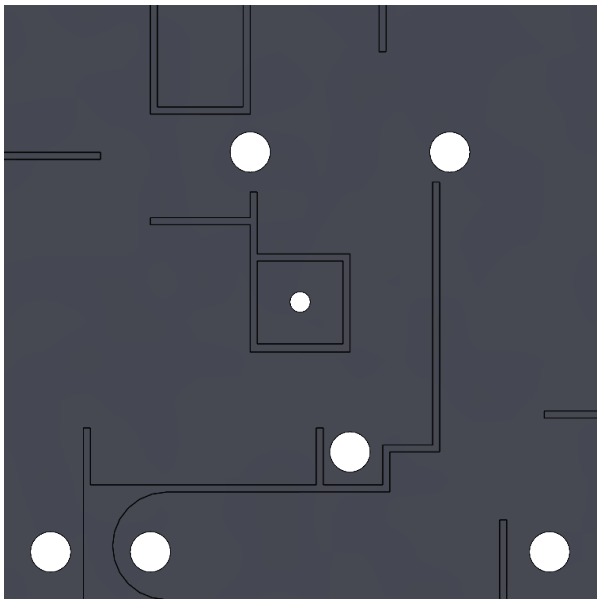


(b) Computer Vision of the Maze

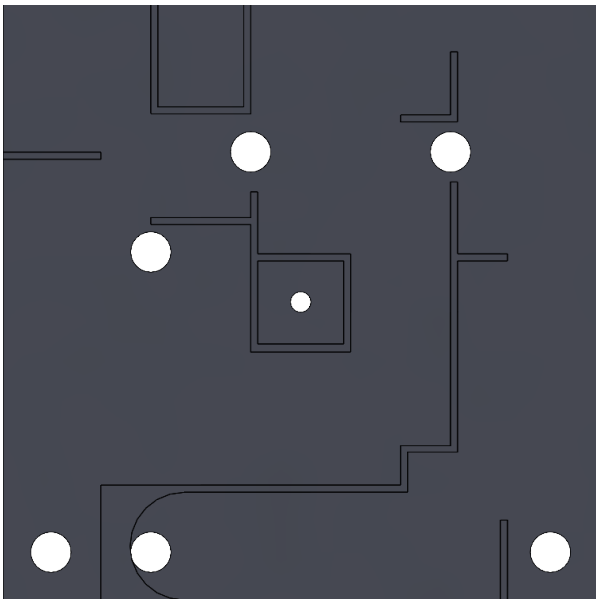
3.5 Technical Design

In technical design we will go through the different gametrays and parts that is needed to get the ball up to the maze again automatically after completing/failing the maze.

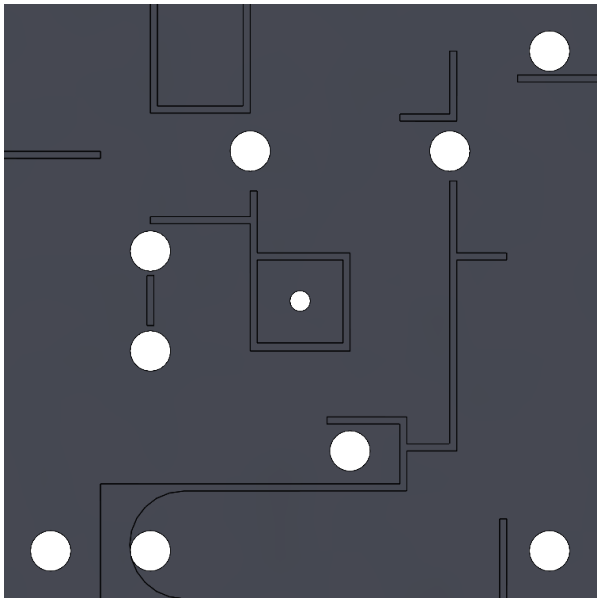
3.5.1 Gametrays



(a) Gametray 1, Easy



(b) Gametray 2, Medium



(c) Gametray 3, Hard

Figure 3.8: Showcase of the different gametrays.

During this project there has been made three different gametrays. Gametray 1 shown in figure 3.8a is the easiest one. This gametray is designed to be beginner-friendly and suitable as a demonstration. It has a simple layout with clear and wide paths, making it

easy to solve. This also make the layout ideal as a start for the AI learning. The medium layout shown in figure 3.8b offers a a harder maze to solve, however it is still possible to use the walls to your advantage. The hard layout show in figure 3.8c makes it almost impossible to use the walls as end stops. It presents a maze that require more accuracy, both from the user and from the actuators.

3.5.2 Mounting of the gametrays

To mount the gametray to the system, a mounting part was needed. The gametray needs to be able to rotate freely in both pitch and roll axis, therefore the current design were implemented; see Appendix H. By using a half sphere, the gametray will be able to maneuver freely on the surface of the sphere while being attached to the actuators to maintain movement only in roll and pitch. The actuators are connected to the gametrays via the connector part see Appendix C. This part allows the gametray to stay rigid in yaw axis rotation, and is designed to be easily detached by snapping it on/off.

3.5.3 Drain

Many designs were discussed when it came to the funnel of the system. Due to limitations of 3D-printing large parts, the final design became multiple small parts mounted together; see Appendix I , Appendix J and Appendix G.
Below the top drain, the bottom drain is located to pass the ball on to the chute and so on.

3.5.4 Chute

The chute is designed to transfer the ball to the elevator. On this path there is an infrared sensor above to capture if the ball is passing by which then starts the elevator. In order for the sensor to detect the passing of the ball, the ball has to pass the sensor slow. Therefore the chute is designed with a 90 degree angle to slow down the ball as much as possible.

3.5.5 Cylinder Ball Cup

The first initiation of the elevator, is after the transportation of the ball via the chute. From the chute, the ball will land in the cup which is lowered below the main plate and is then being pushed into the helix elevator due to gravitation. The dc motor should be activated and the next step is initiated.

3.5.6 Helix Elevator

One of the most complex and challenging parts of the design, were the helix elevator due to the precision needed in order for it to actually work. The ball requires enough friction against the cylinder to elevate, so the dimensions had to be perfectly correct in order to

obtain enough friction and flow. Due to inaccuracy in the printers, the elevator had to be manufactured and re-dimensioned several times.

3.5.7 Ball Pusher

When the ball has arrived at the top of the cylinder, it will fall out due to the "ball pusher". This part is designed to be mounted on top of the cylinder so that when the ball is elevated above the cylinder it will no longer be experiencing friction against the walls and will then be pushed out of the elevator. The ball is now back to the gametray in the bottom left corner which is the starting position for every maze. *(See the assembly in Appendix B for a complete overview of the system.)*

4 Result

In result we will go through the design of construction, how manual and automatic control works and how far we come with the machine learning and AI system. All the python code mentioned in this section can be found on GitHub.

4.1 Design

In this project, the mechanical design showcases a seamless integration of aluminum profiles, see-through acrylic glass, and 3D-printed components, resulting in a robust and effective system. A notable highlight of this design lies in its user-friendliness, allowing for easy maintenance and swift part replacement if required. The frame's construction boasts a hassle-free assembly process, enabling convenient modifications and adjustments as needed. This adaptability ensures that the system remains highly versatile and accommodating to changes without compromising its structural integrity.

Particularly noteworthy is the ingenious design of the game trays' mounting mechanism, which functions precisely as intended. The combination of the ball joint and actuator mounts guarantees that the game board only moves along the pitch and roll axes, preventing any undesirable yaw movements. This deliberate choice in design ensures stability and accuracy during gameplay, contributing to an engaging and enjoyable experience for users.

In conclusion, the utilization of aluminum profiles, see-through acrylic glass, and 3D-printed components, along with the emphasis on ease of maintenance and adaptable part replacements, showcases a well-executed mechanical design. The careful consideration of the game trays' mounting mechanism further reinforces the system's reliability and performance, creating an efficient and immersive gaming setup.

In our quest to find a creative and efficient solution to transport the ball to the starting point of the maze automatically, we devised a unique mechanism - a spiral stair/slide system enclosed within an acrylic cylinder. The primary driving force behind this inventive approach is a DC-motor that powers the spiral stair/slide, creating a captivating and functional setup. The concept revolves around the principle of friction and dynamic interaction. As the DC-motor spins the spiral stair/slide, it generates a compelling force between the stair/slide's surface and the ball. This frictional interaction propels the ball upwards along the spiral stair/slide, elevating it to the desired starting point of the maze. The spiral stair/slide's curved design and the precisely calculated angle ensure the ball's smooth ascent within the acrylic cylinder. This ingenious solution eliminates the need for manual intervention, providing an automated and efficient way to position the ball at the beginning of the maze. The integration of a DC-motor as the driving force, combined with the unique properties of friction, culminates in a self-sustaining system that not only solves the problem of ball placement but also adds an engaging and entertaining element to the overall maze experience. This innovative approach exemplifies our commitment to creative problem-solving and engineering excellence.

The Jetson Nano GUI is designed to provide an intuitive and user-friendly interface for controlling and interacting with the system. Upon starting the application, the Main menu will be displayed, featuring five buttons that grant access to different control modes: Manual, Auto, AI, Override, and Quit.

Manual Control: Upon selecting the Manual Control mode, the user is prompted to ensure that the Xbox controller is powered on and paired with the Jetson Nano. A steady LED on the Xbox controller confirms successful pairing. This mode facilitates board tilting and elevator control using the Xbox controller. The instructions for these actions are provided within the window. To return to the main menu, the user can click the "return to menu" button. Afterward, providing any input from the Xbox controller will automatically take the user back to the main menu.

Auto Control: The Auto Control mode is enabled when a correctly positioned maze is detected, with its start point located in the bottom-left corner. The selected maze is indicated with a green outline. To allow the program to automatically solve the maze, the user can click the "Run Auto Control" button. If no maze is detected, the "Run Auto Control" button will be disabled. To return to the main menu from this mode, the user can click the "return to menu" button.

AI Control: In the AI Control mode, the user can check which maze has been selected for solving. If a maze is detected, the "Run AI" button will be enabled, allowing the user to activate the AI to solve it. Conversely, if no maze is detected, the "Run AI" button will remain disabled. By clicking the "Run AI" button, the user can initiate the solving

process and return to the main menu afterward.

Override Control: The Override Control mode empowers the user to have direct control over two actuators: Actuator 1 for the y-axis and Actuator 2 for the x-axis. Sliders enable the user to adjust the positions of these actuators as desired. Furthermore, the user can control the DC-motor/elevator by toggling it on or off. This mode allows for manual intervention and fine-tuning of the system.

Quit: To terminate the GUI program, the user can simply click the "Quit" button. This action halts the program's execution and closes the interface.

The GUI design aims to provide a seamless and efficient user experience. Clear and concise instructions accompany each mode, guiding the user through the various functionalities available. The interface features visually distinguishable buttons, sliders, and indicators, making it easy for users to understand their current state and available actions. Additionally, the "return to menu" button in each mode ensures quick navigation back to the Main menu, promoting a smooth user workflow.

Overall, the GUI design on the Jetson Nano streamlines interaction and control, enhancing the overall user experience while managing the complexities of the underlying functionalities.

4.2 Manual control

The manual control of the board thru the xbox controler works well, and the response feels good. In some cases it can feel like the actuators are moving a bit slowly, however by trial and error it was discovered that the board was more uncontrollable when the speed was higher.

4.2.1 Xbox Controller

The XboxController class in this Python script provides a convenient way to interact with an Xbox controller and use its inputs to control external devices. The class is designed to work with a specific Xbox controller connected to the system. By default, it uses the serial port /dev/ttyACM0 and the joystick device /dev/input/js0, but you can specify different paths if needed. Upon initialization, the class establishes a serial connection to communicate with external devices. It also defines mappings for the various axes and buttons on the Xbox controller, assigning names to each ID. This mapping enables easy access to specific input events later on. The class provides a method called findXboxController(), which is intended to list all available joystick devices in the /dev/input directory. However, this method is currently not used in the class and might have been left for potential future improvements or debugging purposes.

The openJoystickDevice() method is used to open the joystick device for reading input events. It does this by creating a file descriptor to interact with the device directly. To determine the number of supported axes and buttons on the connected Xbox controller, the class offers two methods: getNumAxes() and getNumButtons(). These methods query the joystick device using specific IOCTL calls and provide the information needed for setting up the controller. The setupJoystick() method is responsible for mapping the axes and buttons of the controller and setting up their initial states. It uses the previously queried information to build the mappings and state dictionaries for both axes and buttons. The method also prints the name of the joystick device, indicating successful initialization.

Once the controller is set up, you can start reading input events from the Xbox controller using the start() method. This method creates a separate thread to handle the continuous reading of input events, so it doesn't block the main program execution. The main loop for reading and processing input events is defined in the run() method. Inside this loop, the class continually checks for input events from the joystick device. If it detects a button press event, it sends a corresponding command through the serial port. Similarly, if there is an axis event (e.g., joystick movement), it calculates the corresponding value and sends it to the serial port. To stop reading input events, you can call the stop() method, which sets the self.running flag to False, effectively ending the loop in the separate thread.

Overall, the XboxController class encapsulates the functionality needed to interface with an Xbox controller and control external devices. By providing a simple and clear interface, this class makes it easier for developers to incorporate Xbox controller support into their projects, enabling diverse applications ranging from robotics and automation to gaming and remote control systems.

4.3 Automatic control

To accomplish automatic solving of any given maze we chose to do that by making a script that reads the values from the Xbox controller with timestamp between each input from the controller. This allowed us to solve the maze first manually and then use another code called AutomaticMaze to read the text file to as if it was inputs from a Xbox controller. To find out witch maze it is going to solve, we made a program called MazeDetect that can recognize the different mazes. How the different scripts works can you read more about in the next three subsections.

4.3.1 Write To File With Timestamp

You can write to any text file with this script, but the name "Maze 3.txt" is used as an example. The script starts by opening a file named "Maze 3.txt" in write mode to log the

actuator's position and the time difference between input events. Next, the script queries the system to find available joystick devices in the /dev/input directory. It checks for devices starting with the name 'js' and prints the available device paths to the console. The script establishes a serial connection with the actuator, using the specified serial port "/dev/ttyACM0" and a baud rate of 115200. The serial connection is flushed to prepare for communication.

The script then proceeds to set up the Xbox controller for reading input events. It opens the joystick device file "/dev/input/js0" for reading events. The number of axes and buttons supported by the controller is queried, and the corresponding information is used to initialize axis states and mappings. The main loop starts, which continuously reads input events from the Xbox controller using jsdev.read(8). It interprets the event buffer and processes the input accordingly. Specifically, it checks for axis events (joystick movements) by inspecting the event type (evType) and handles the left joystick's X and Y axes.

For the left joystick's Y-axis (number 1), the script calculates the position value based on the normalized value from the input event. It then writes this value to the actuator through the serial connection, reads any response from the actuator, and updates the axis state. The actuator position is also printed to the console. Similarly, for the left joystick's X-axis (number 0), the script calculates the position value and writes it to the actuator, updates the axis state, and prints the position to the console.

In addition to updating the actuator position, the script calculates the time difference between successive input events using the current time and the previous time recorded in the loop. This time difference is also printed to the console. Finally, the script writes the actuator position and the time difference to the "Maze 3.txt" file in each iteration, formatted as "actuatorPosition timeDiff" on separate lines. The loop continues indefinitely, continuously reading input events, updating the actuator position, and logging the data to the file.

Overall, this script enables real-time control of an actuator based on input events from an Xbox controller. It also logs the actuator's position and time differences between input events for analysis or debugging purposes. The provided "Maze 3.txt" file serves as a log for recording these data points.

4.3.2 Maze Detection

The MazeDetector class provides functionality for detecting mazes in a video stream using OpenCV. When instantiated, the class sets up a video capture using the specified video capture index, frame width, and frame height. The video capture is initialized to

read frames from the specified video source, which could be a webcam (default index 0) or another video input device. The class also loads multiple maze templates, each represented as a grayscale image. The templates are stored in a list, and each template is associated with a unique name, such as "Maze 1," "Maze 2," and so on. These templates serve as reference images to detect the presence of specific mazes in the video stream.

The core method of the MazeDetector class is detectMaze(). This method captures a frame from the video stream and performs template matching on a region of interest (ROI) extracted from the frame. The ROI is selected to focus on the area where the maze is expected to be present. The best match is determined by comparing the template match scores with predefined thresholds for each maze. The template matching process involves resizing each template image to match the size of the ROI. Then, OpenCV's template matching function (cv2.matchTemplate()) is applied, and the resulting match score is compared with the threshold specific to the template. If a match score surpasses the threshold and is better than any previous match, the corresponding maze name is considered the "detected" maze. The class includes four predefined maze templates, each with its specific matching threshold. You can extend the class by adding more templates and setting appropriate thresholds for each new maze.

In the example usage provided at the end of the script, the MazeDetector object is created, and the detectMaze() method is called in a loop to continuously search for mazes in the video stream. Once a maze is detected, the loop breaks, and the detected maze name is printed. You can replace the print statement with custom code to process the detected maze information further. Finally, after the loop exits, the release() method is called to release the video capture and close any OpenCV windows used during the process.

Overall, the MazeDetector class is a useful tool for real-time maze detection in video streams. It offers flexibility to detect different mazes, as long as the templates and matching thresholds are appropriately set. This class can be employed in various applications, such as robotics, games, or any scenario where real-time maze recognition is necessary.

4.3.3 Automatic Maze

The AutomaticMaze class allows for automated control of an Arduino device connected via serial communication. It can read data from a text file containing commands and send them to the Arduino to navigate through a maze based on a detected maze pattern.

The class is initialized with the serialPort parameter, which specifies the serial port used to communicate with the Arduino. During initialization, the serial connection is not established yet; it is done explicitly through the connect() method. The connect() method

is used to establish the serial connection with the Arduino. It takes optional parameters for the baud rate and timeout. After connecting, the class waits for a brief period to ensure a stable connection with the Arduino. The disconnect() method allows closing the serial connection with the Arduino when needed. This method is called at the end of the operation or when the communication with the Arduino is no longer required.

Data can be sent to the Arduino from a file using the sendDataFromFile() method. It takes a file path as an input and reads commands from the file. Each line of the file contains two space-separated values: an integer value (interpreted as a command) and a float value (interpreted as a time delay). The class sequentially sends these commands to the Arduino with the specified time delays between each command. The main functionality of the class lies in the fromFile() class method. This method creates an instance of AutomaticMaze, establishes the serial connection, sends the data from a file (determined by the detected maze name), and then disconnects the Arduino.

The data file is generated based on the detected maze name (provided as a parameter to the fromFile() method). The file name is formed by appending the maze name with the .txt extension. The data in the file corresponds to the specific commands needed to navigate through the maze and control the Arduino's movements. To use the AutomaticMaze class, you would typically call the fromFile() method after detecting a maze using the MazeDetector class. This will automatically send the necessary control commands to the Arduino for navigating the detected maze.

The AutomaticMaze class provides a powerful tool for automating the navigation of robots or devices in mazes. By combining maze detection with automatic control, it enables various applications, such as autonomous robot navigation and maze-solving scenarios.

4.4 Machine learning and AI

There have been experiments and testing of a potential vision based machine learning algorithm to solve the maze layouts automatically. These experiments were conducted in a simulated environment, in the unity game engine. Using the solidworks files a small game was created where the machine learning agent was allowed to control the rotation about Z- and X-axis. The agent was programmed using the MLAgents package which is a package specifically made for machine learning within a unity simulation. This package allows for a tensorflow machine learning algorithm to interact with unity, by reading and writing data.

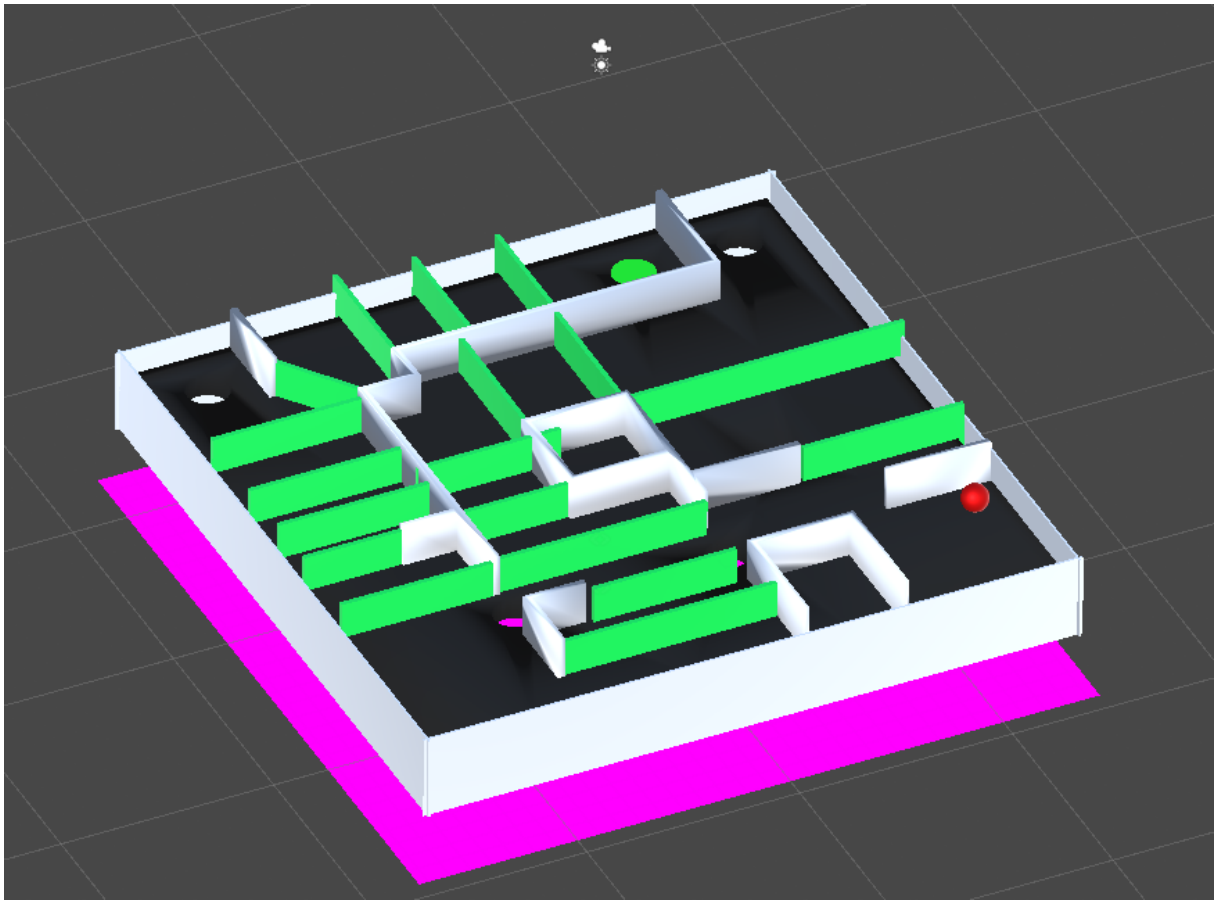


Figure 4.1: The unity enviroment for testing

4.4.1 Machine learning algorithm

When choosing which algorithm to use it was a choice in between two different algorithms, Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO). SAC is a smart way for machines to learn by trying different actions in a game or task. It encourages the machine to explore new strategies while also trying to get the best possible rewards. while PPO is another method for machines to learn, but it focuses on being safer and more efficient. It helps the machine gradually improve its performance without making big, risky changes all at once.

4.4.2 Scripting and observations

The unity script called mazeSolvingAgent is the main script which allows the MLAgent to interact with the unity enviroment. It controls what behaviour the agent is allowed to react to and what actions it is allowed to do. Currently these actions are only give three numbers. The first and second action correspond to the movement of the board, where a value -1 to 1 are given. These are mapped respectively where action is X and action 2 is Z. The third action controls whether or not to move the board, this is a value of either 0 or 1, if 0 then don't move, if 1 then move. The actions are then sent to the objectRotation script which controls the movement of the board, it sends a rotate transform command to the pivot point which then rotate with the respective axis that corresponds to the action.

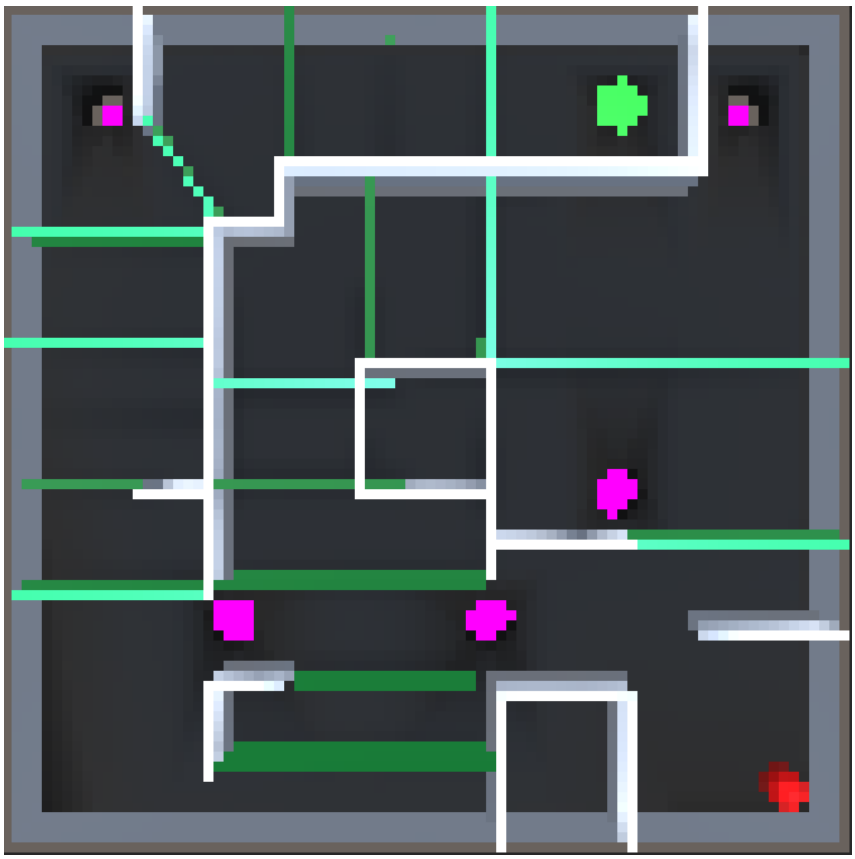


Figure 4.2: The AI camera view

The agent is camera based within unity. The camera is positioned the same height and length away from the center of the board. However, only has 84x84 pixels with rgb values. This it to limit the amount of observations the machine learning algorithm has to process at any given time, as each individual pixel count as a singular observation by the Machine learning agent(MLA). This means for a 84x84x3 the amount of observations made are 21168 every frame taken by the camera. In addition to this 5 frames are sent at a time to the MLA, while this does result in less frequent sent data, as it only sends every 5 frames. This data is valuable to know as it allows the MLA to read the velocity of the metal ball, as it saves different position within a small time frame.

4.4.3 Rewards and punishments

When training a machine learning agent rewards and punishments are the most important parts, an appropriate reward structure can be the difference in between 10 and 10000 hours of training. When solving the maze we opted for a more simple reward structure of checkpoints to the goal and goal as rewards versus time and falling through holes as punishments. While one could train the entire MLA visually with only on goal, the time it would take would outweigh the benefits, and result in some unexpected behaviour such as sitting in corners, to minimize the potential punishments. The checkpoint system gave active rewards for exploring the maze, and no reward for back tracking, meaning that the MLA was able to maneuver the maze in only one direction. While this works, it is also a painfully slow way of training the MLA, this however might be due to the large amount of observations the MLA takes in per action. More observations do mean more training time, which further exaggerates the need for a good reward structure.

4.4.4 Physics

The observation and actions allowed the agent to move the ball around the maze by tipping the ball around. The physics are trying to mimic the real life environment as much as possible, using Nvidias physx physics engine for 3d physics. The fixed timestep is set to around 0.02, this dictates how many physics equations are solved by the engine. These can happen more times per frame, if set low enough however it can lead to performance issues. The fixed time step in the simulation used to be 0.001, this however lead to a significant slowdown where the frame rate kept up, but the physics engine calculated too slowly. Making the physics act as if in slow motion, making for inaccurate training.

Friction

The friction parameter in physx is a way to calculate the friction in-between two gameObjects. The friction parameter within physx is not based on any scientific number, rather is a simple value that dictates the amount of friction. For the case of the maze solver, a simple value of 0.1 was chosen.

4.4.5 User instructions

To install unity and mlagents, either the youtube tutorial by CodeMonkey or the official github page has great resources.

Installation Tutorials:

- CodeMonkey & GitHub

Assuming that you have installed unity and mlagent properly. There are couple ways to train the MLA, we could run the base MLA without any changes. Even tho it is recommended to changet he config parameters, to suit the desired outcome and speed up training time.

MLagent launch command

```
cd <Path to your project>
\venv\Scripts\activate
mlagents-learn --run-id="name of MLA"
```

After you launch this, simply click the play button in the unity environment and the MLA will start training. The previus MLA brains can be found in the "Results" folder in your unity project.

A couple of useful commands when training:

MLagent useful commands

```
mlagents-learn --force
```

This will force the MLA to train even if there is already a brain. It will however override the old one and start from step = 0

MLagent useful commands

```
mlagents-learn --resume
```

This will resume training from the last step.
If you wish to train using a specific config:

MLagent launching a different configuration

```
cd <Path to your project>
\venv\Scripts\activate
mlagents-learn --run-id="name of MLA" Config/"config".yaml
```

This will launch the MLA with a specific configuration of your choosing. This is where we specify what type of algorithm and its parameters.

4.4.5.1 The config files

Due to the sheer amount of different configuration possibilities there are only a couple that will be outlined and explained here. Those being the most important ones for either SAC or PPO.

SAC config file

```
My Behavior:
  trainer_type: sac
  hyperparameters:
    batch_size: 128
    buffer_size: 50000
    learning_rate: 3.0e-4
    learning_rate_schedule: linear
    buffer_init_steps: 0
    tau: 0.005
    steps_per_update: 10.0
    save_replay_buffer: false
    init_entcoef: 0.5
    reward_signal_steps_per_update: 10.0
```

- **batch_size:** 128
  This parameter determines how many experiences (data samples) are used at once to update the learning algorithm. A larger batch size can lead to more stable learning but may require more memory. Typical range: (Continuous - PPO): 512 - 5120; (Continuous - SAC): 128 - 1024; (Discrete, PPO & SAC): 32 - 512.
- **buffer_size:** 50000
  It sets the size of the replay buffer, which is a memory storage that holds past experiences. The replay buffer helps the algorithm to learn from previous experiences, making learning more efficient and robust. Typical range: PPO: 2048 - 409600; SAC: 50000 - 1000000
- **learning_rate:** 3.0e-4
  This specifies the step size used to update the neural network's parameters during training. A smaller learning rate can lead to slower but more precise learning.
- **learning_rate_schedule:** linear
  The learning rate schedule defines how the learning rate changes over time. In this case, it's set to linear, which means the learning rate decreases linearly over the training process. Constat keeps it at a set value. For SAC a constant learning schedule is better, and for PPO a linear would be better.
- **buffer_init_steps:** 0
  It represents the number of initial steps taken by the agent to collect experiences

before the learning starts. This helps in filling up the replay buffer before the actual training begins.

- **tau:** 0.005
  Tau is a parameter used in some algorithms for soft updating target networks. It determines how fast the target networks are updated with the main networks.
- **steps_per_update:** 10.0
  This indicates how many steps the agent takes in the environment before each update of the neural network. Smaller values can lead to more frequent updates but may be computationally expensive.
- **save_replay_buffer:** false
  If set to true, this will save the replay buffer to disk, which can be useful for analysis and debugging.
- **init_entcoef:** 0.5
  The initialization value for the entropy coefficient, which is used in some reinforcement learning algorithms to control the level of exploration the agent performs.
- **reward_signal_steps_per_update:** 10.0
  This parameter specifies how many steps the agent takes before each update of reward-related calculations. It can help to decouple reward updates from policy updates and stabilize the learning process.

PPO differs from only in the hyperparameters:

PPO config file

```
My Behavior:
  trainer_type: ppo
  hyperparameters:
    batch_size: 1024
    buffer_size: 4096
    learning_rate: 0.0003
    beta: 0.01
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    shared_critic: false
    learning_rate_schedule: constant
    beta_schedule: linear
    epsilon_schedule: linear
```

- **batch_size:** 1024
  Determines how many experiences (data samples) are used together to update the agent's learning algorithm. Larger batch sizes can provide more stable learning.
- **buffer_size:** 4096
  Sets the size of the memory storage called the replay buffer. The replay buffer helps the agent learn from its past experiences, making its learning more efficient.
- **learning_rate:** 0.0003
  Represents the step size the agent takes to learn from its experiences. A smaller learning rate can lead to more precise learning, but it may take longer.
- **beta:** 0.01
  Beta is a value that affects the agent's learning process. It helps control the trade-off between exploration and exploitation during training.
- **epsilon:** 0.2
  Epsilon is another value that influences the agent's learning process. It helps control how much the agent explores new actions versus exploiting its current knowledge.
- **lambd:** 0.95
  Lambda is a parameter that influences the agent's learning algorithm. It helps balance the importance of current and future rewards during training.
- **num_epoch:** 3
  Indicates how many times the agent's neural network is updated using the collected experiences from the replay buffer. More epochs can help improve learning, but it also requires more computation.
- **shared_critic:** false
  If set to false, it means the agent doesn't share its knowledge between different tasks or behaviors, making it learn more specifically for the current task.
- **learning_rate_schedule:** constant
  The learning rate schedule defines how the learning rate changes over time. In this case, it's set to constant, meaning the learning rate remains the same during training.
- **beta_schedule:** linear
  Beta schedule defines how the beta value changes over time. In this case, it's set to linear, indicating the beta value decreases linearly over the training process.
- **epsilon_schedule:** linear
  Epsilon schedule defines how the epsilon value changes over time. In this case, it's set to linear, indicating the epsilon value decreases linearly over the training process.

5 Further Development

5.1 Design & software

The system needs the external influences to be at a minimum to be able to preform consistently. The imperfections also need to to be low. It therefore might be desirable to redesign the gametrays so that they no longer can move around inside the frame, to insure consistency. It might also help to reconfigure the actuator control boards to work even better. The mounting brackets for the actuators to connect to the gametray also flex a bit, witch is undesirable. The elevator is only just working, if the acrylic tube or the "screw" itself tears down it might not work any longer. Another weakness with the elevator design is that it can only use one sized balls.

The design is not very compact, but it is serviceable. Now that we know that the design works well it can be made smaller and more compact. This will be desirable so that the system easily can be moved and used in different places.

We were planning to introduce an exciting new feature in the form of a class that harnesses computer vision capabilities to enhance the maze game experience. This class will enable the system to detect and track the ball's movement, specifically noting when it departs from the starting area and when it successfully reaches the goal. Additionally, it will record the time taken by the player, along with their initials, to create a personalized time log. To facilitate this function effectively, the ball used in the game must possess a distinct and strong color, preferably red or orange, making it easily recognizable by the computer vision system.

The proposed class will have the ability to control the camera, ensuring it can be turned on and off as needed. As the same camera is utilized for various functionalities, such as detecting the maze configuration or potentially implementing an AI component, this camera control feature becomes essential for efficient usage. Upon capturing the ball's movement and timing the gameplay, the class will save the recorded data, including the player's initials and time taken, for future reference. The system will maintain a scoreboard, which showcases the five best times achieved by different players, providing a competitive and engaging element to the game.

Overall, this new class with computer vision capabilities will add a whole new dimension to the maze game, enhancing user engagement and creating an immersive and interactive gaming experience. It will not only improve the game's functionality but also pave the way for future enhancements and potential AI integration, contributing to a dynamic and evolving gaming platform.

5.2 AI control

While the machine learning agent(MLA) is able to solve the maze, from time to time. It seems to be mostly from random inputs. Even with the checkpoint system the MLA, prefers corners over actual exploration. This is one of the reasons why SAC was chosen instead of PPO. PPO would get very comfortable with corners after winning a little bit of positive reward. To combat this SAC was used in addition to adding seperate colliders for corners, these act as trigger and give the MLA a negative reward from sitting in them. However, SAC has a perceived . As this was a learning experience during the project, the reward structure could have been better. Instead of checkpoints a form of "area" checkpoint might have been more beneficial. Rewarding for exploring rather, than reaching a certain part of the maze.

Due to dynamic environment of the maze the MLA has trouble compensating and balancing the board properly. This leads to some unfortunate behaviour when the MLA tries to control against balls high inertia. Due to it being modeled after the real life system, there can unfortunately not be made changes to how fast the MLA reacts as the limit is within the physical system.

When training the MLA, due to using a camera it was not possible to run multiple training environments at the same time. Something that would be possible if no camera was used, but then the MLA would solve the maze only on memory rather than solving it. This should be possible, but due to issues getting it to work, and lack of computing power

Curriculum training could also have been implemented, and the MLA would likely benefit greatly from such a system. By for example making an open plane, and only the goal. Such that the MLA could more easily adjust to the more difficult maze environment.

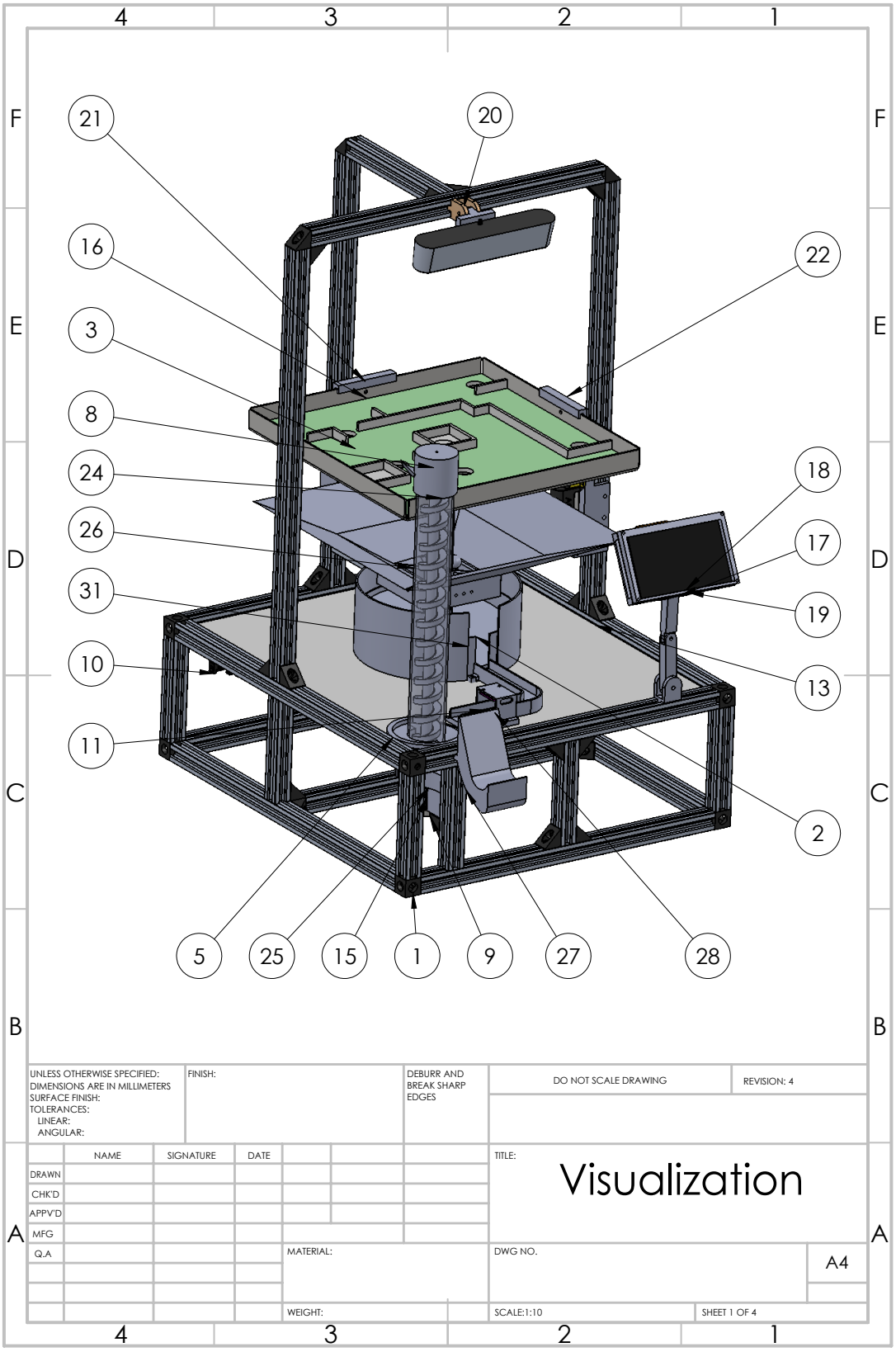
A list of an example curriculum training environment:

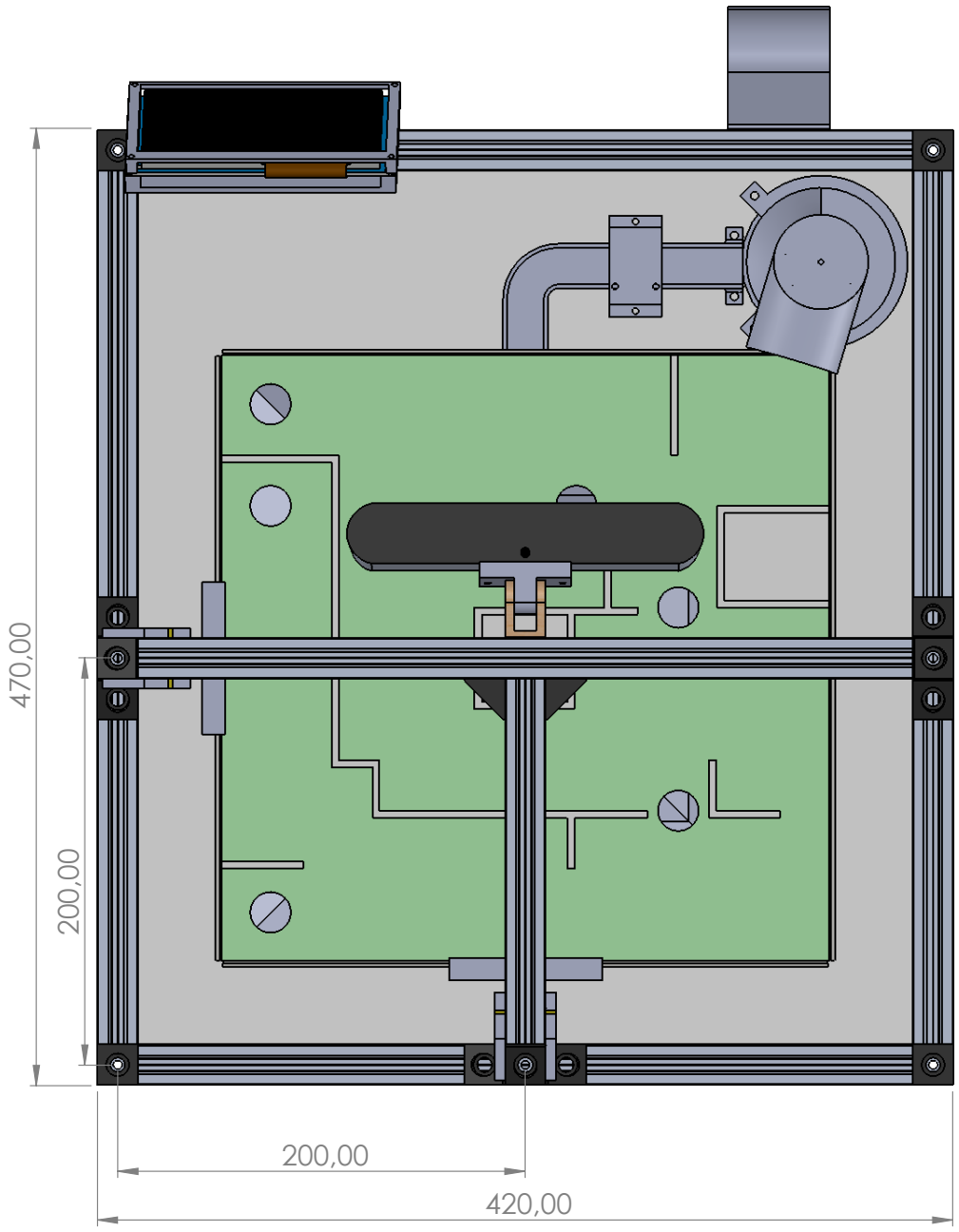
- A singular tiling plane, with goal
- Introduce holes.
- Introduce walls.
- Introduce the entire maze
- Randomize mazes and lighting conditions

Appendix A Component list

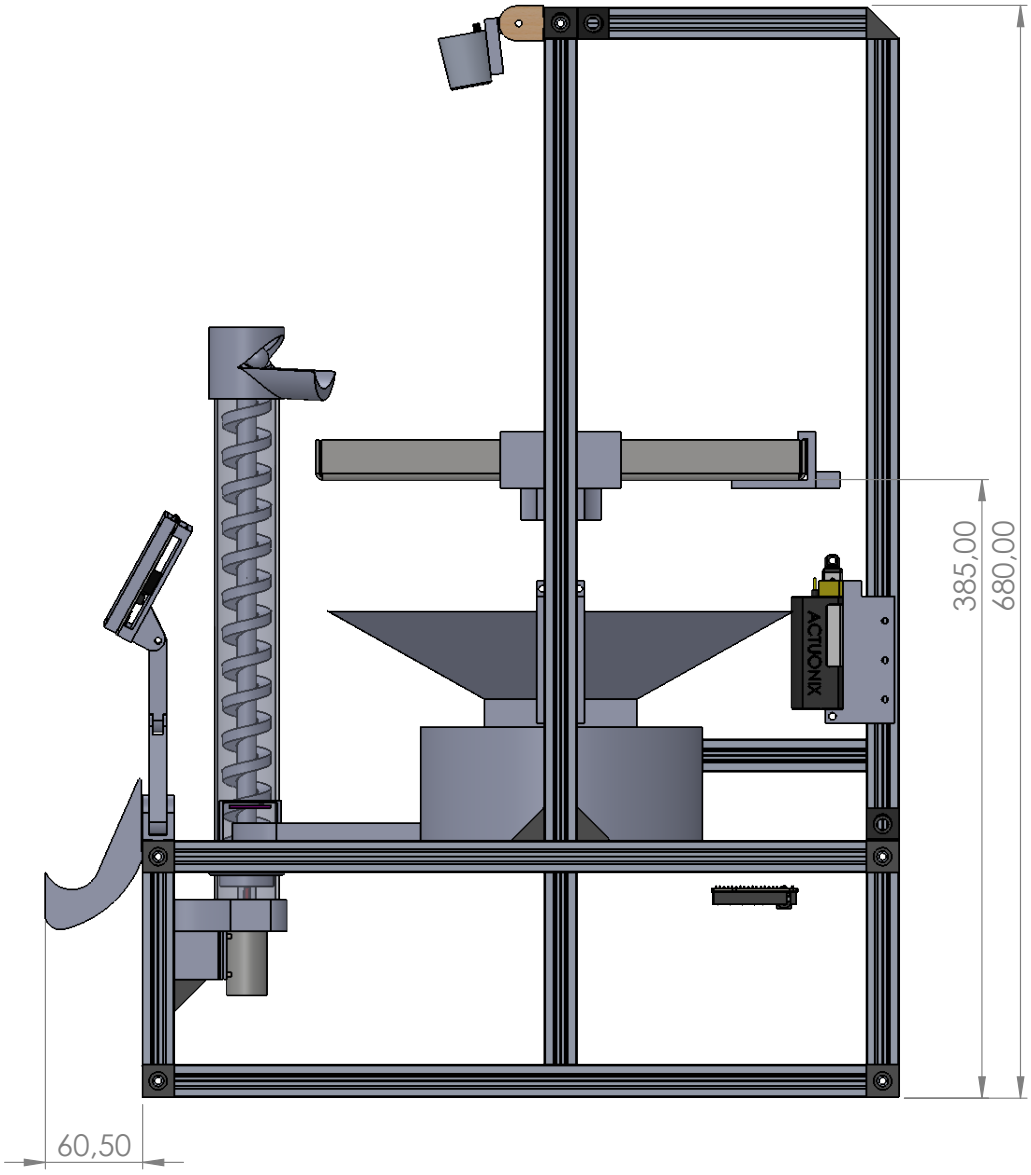
Description	Type	Price	Amount	Total	Site	Datasheet
Actuonix Miniature Electric Linear Actuator, 50mm, 12V dc, 46mm/s Config: Mfr. Part No.: P16-50-22-12-P	Electronics	kr 2 277,41	2	kr 4 554,82	Rocketronics	Datasheet
Actuonix, Linear Actuator Control Board, Analogue, Digital Control, 5 → 24 V dc, 4 A, Panel Mount	Electronics	kr 990,96	2	kr 1 981,92	Rocketronics	Datasheet
MAKER MDD3A - DUAL CHANNEL 3A DC	Electronics	kr 74,59	1	kr 74,59	DigiKey	Datasheet
RS PRO Brushed Geared, 1.31 W, 12 V dc, 10 Ncm, 66 rpm, 4mm Shaft Diameter	Electronics	kr 202,21	1	kr 202,21	Rocketronics	Datasheet
Okdo Nano C100 Developer Kit powered by NVIDIA® Jetson Nano Module	Electronics	kr 2 069,81	1	kr 2 069,81	Rocketronics	Datasheet
Arduino, Mega 2560 Rev 3	Electronics	kr 576,55	1	kr 576,55	Rocketronics	Datasheet
RS PRO Silver Ball Clamping Knob, M5, Threaded Hole	Structural	kr 142,69	1	kr 142,69	Rocketronics	Datasheet
52Pi EP-0113 for use with NVIDIA Jetson Nano (vifte)	Electronics	kr 397,89	1	kr 397,89	Rocketronics	Datasheet
RS PRO 20W Plug-In AC/DC Adapter 5V dc Output, 4A Output	Electronics	kr 226,08	1	kr 226,08	Rocketronics	Datasheet
Laird Connectivity 453-00048 3.3V WiFi and Bluetooth Module, 802.11a, IEEE 802.11ac, IEEE 802.11b/g, IEEE 802.11n UART	Electronics	kr 289,18	1	kr 289,18	Rocketronics	Datasheet
SparkFun Proximity Sensor Breakout - 20cm, VCNL4040 (Qwiic)	Electronics	kr 74,65	1	kr 74,65	Sparkfun	Datasheet
Kapasitiv Skjemmoduler – LCD grafisk TFT - farge HDMI, USB 5” (127,00mm) 800 x 480	Electronics	kr 810,75	1	kr 810,75	DigiKey	Datasheet
XONES CONTROLLER BLACK WITH USB ADAPTER	Electronics	kr 699,00	1	kr 699,00	Power	
ZED 2 Stereo Camera	Electronics	kr 5 004,58	1	kr 5 004,58	StereoLab	Datasheet
Ratrig Startpakke med 12 aluminiumsprofiler	Structural	kr 999,00	1	kr 999,00	Kjell & Company	
PLEXIGLAS 1200*800*4 OPAL	Structural	kr 439,00	1	kr 439,00	Biltema	
Akrylrør Lengde 1 meter, 40mm	Structural	kr 275,00	1	kr 275,00	Sløyd-Detaljer	
Transcend 32 GB MicroSDHC Micro SD Card, Class 10, UHS-1 U1	Electronics	kr 230,20	1	kr 230,20	Rocketronics	Datasheet
Ekstrautstyr stålkule til BRIO Labyrint spill		kr 39,00	5	kr 195,00	Kidsa.no	

Appendix B System Drawing





UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN						Visualization	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
				WEIGHT:		SCALE:1:10	SHEET 2 OF 4



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE: Visualization

MATERIAL:

DWG NO.

A4

WEIGHT:

SCALE:1:10

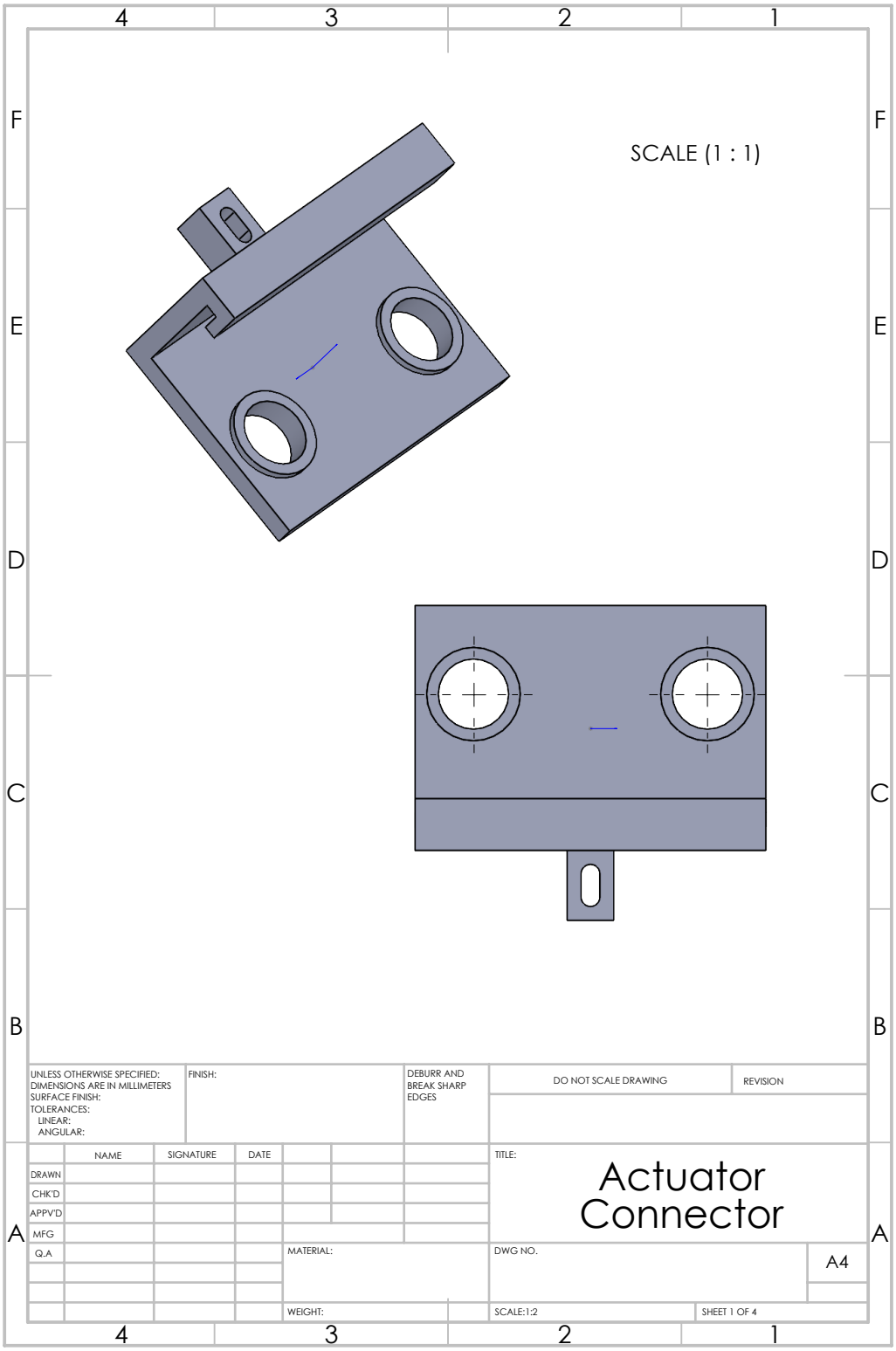
SHEET 3 OF 4

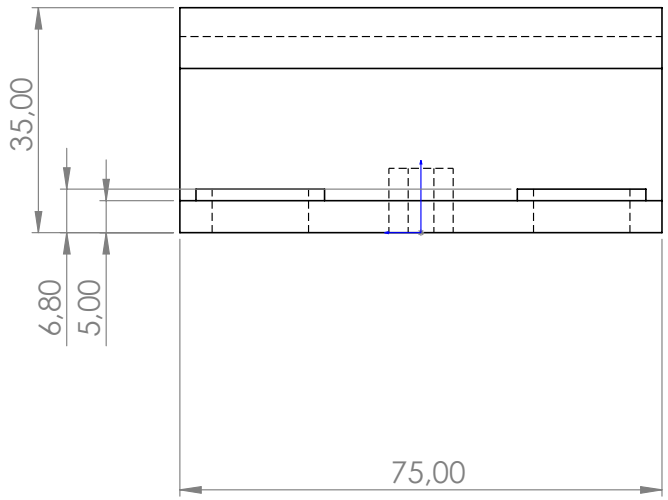
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Aluminium Frame		1
2	Drain		1
3	Actuator Holder		2
4	Mounting Plate		1
5	Cylinder Ball Cup		1
6	Helix Elevator		1
7	ball		2
8	Plexi Cylinder		1
9	DC Motor		1
10	ARDUINO 2560		1
11	Chute		1
12	V-Slot 20x20x120		2
13	Holding Plate		1
14	Black Angle Corner Connector		5
15	DC Motor Holder		1
16	GameTray1		1
17	Monitor Mount Assembly		1
18	NEXTION NX8048T070		1
19	Monitor Cover		1
20	Camera and Mount		1
21	2023_Summer_Project_GameTray_001		1
22	Actuator Connector		2
23	V-Slot 20x20x180		1
24	ball pusher		1
25	cylinder holder		1
26	Top Drain		1
27	xbox one controller holder		1
28	Proximity Sensor and Holder		1
29	Top Drain Side Part		4
30	Anti windup		1
31	Drain Shield		1
32	ACTUONIX-P16-50-64-12-P		2

Q.A | MATERIAL: | DWG NO. Visualization | A4

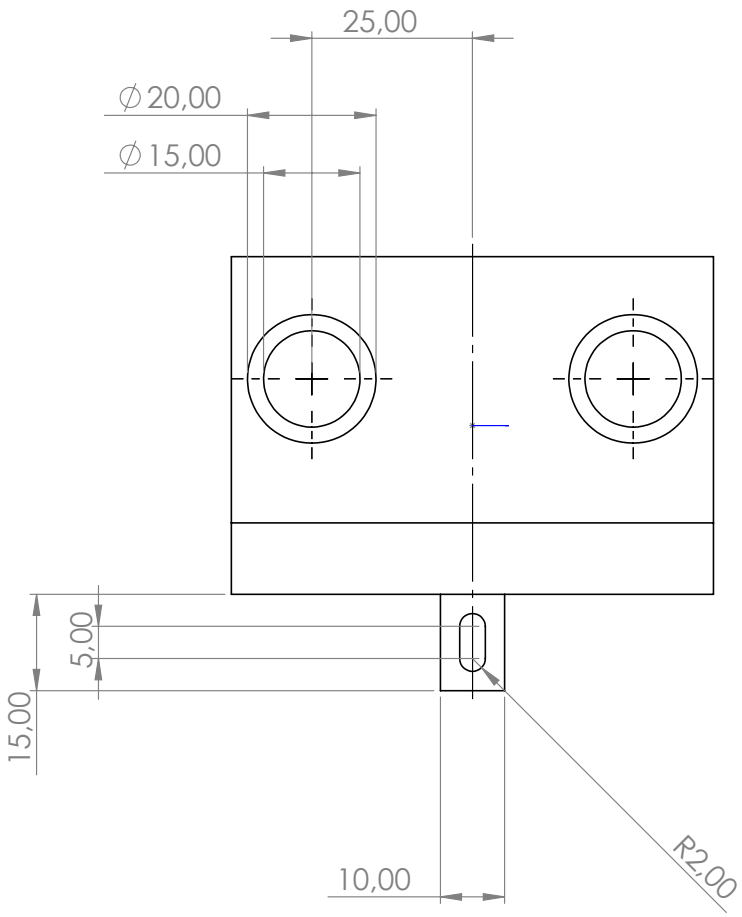
WEIGHT: | SCALE:1:10 | SHEET 4 OF 4

Appendix C Actuator connector



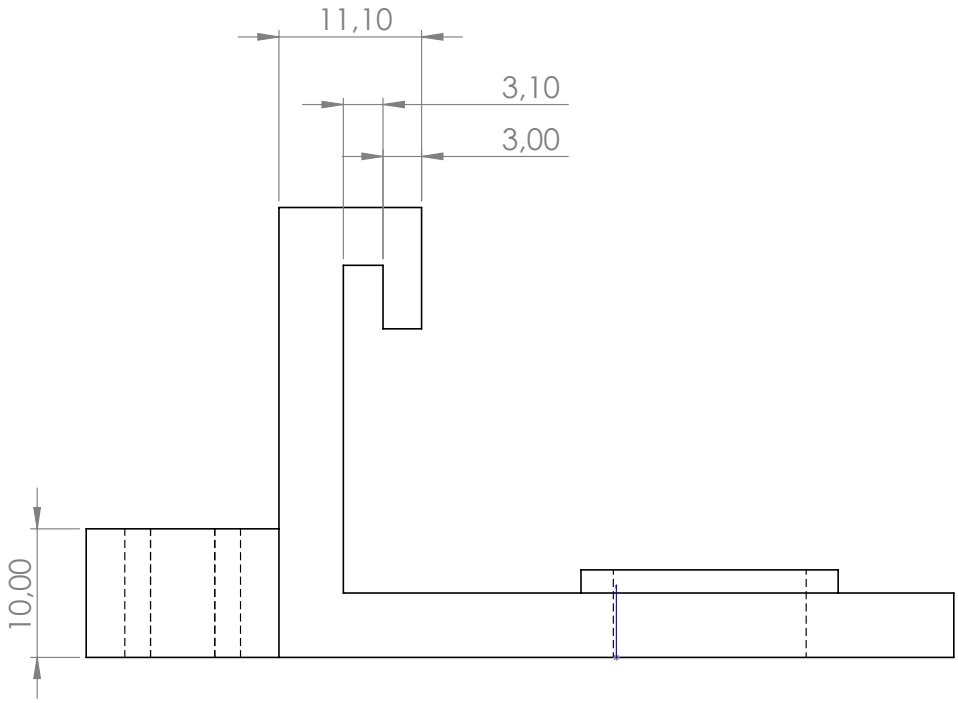


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN						Actuator Connector	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
				WEIGHT:		SCALE:1:2	SHEET 2 OF 4



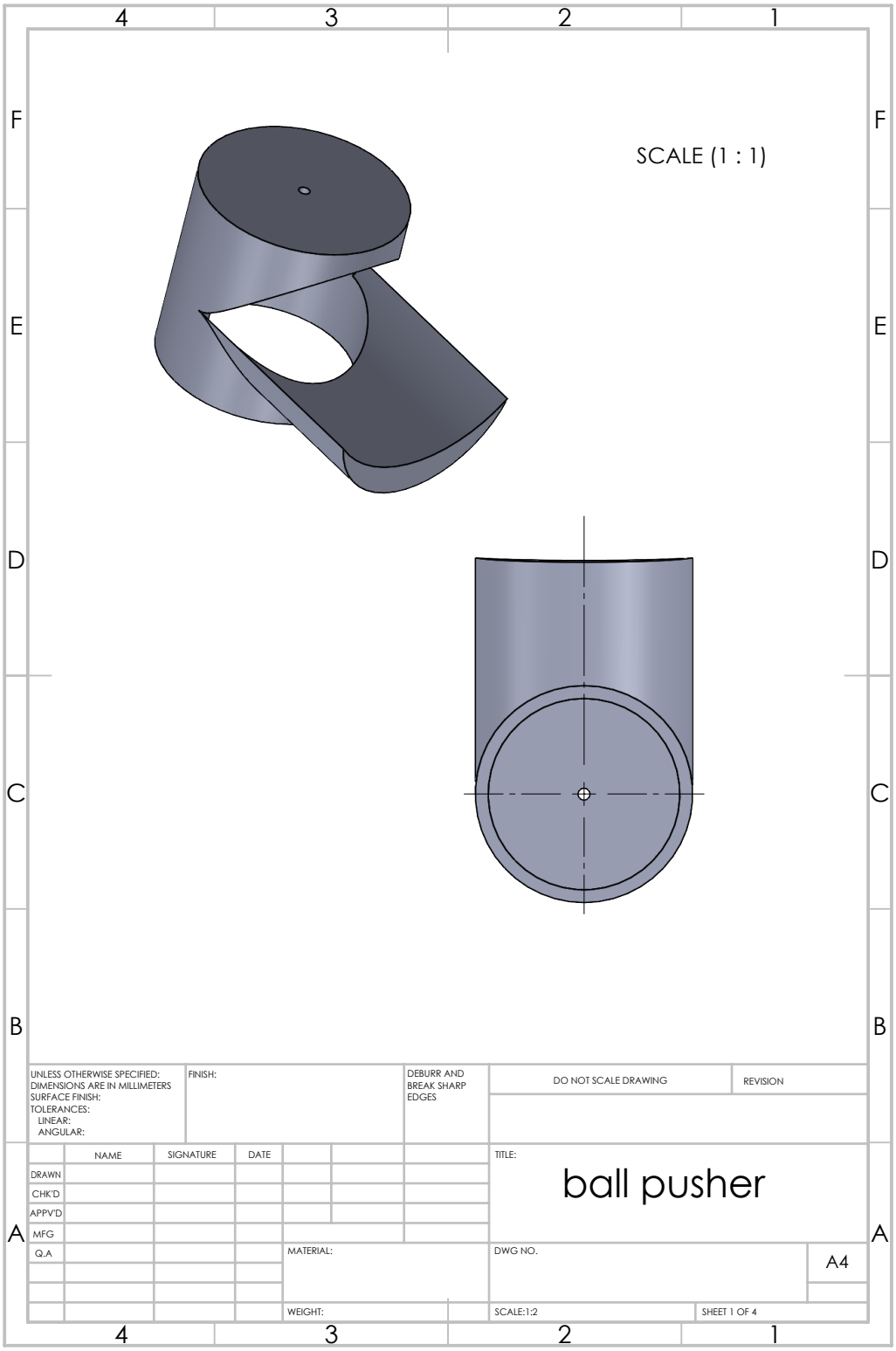
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN						Actuator Connector	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
				WEIGHT:		SCALE:1:2	SHEET 3 OF 4

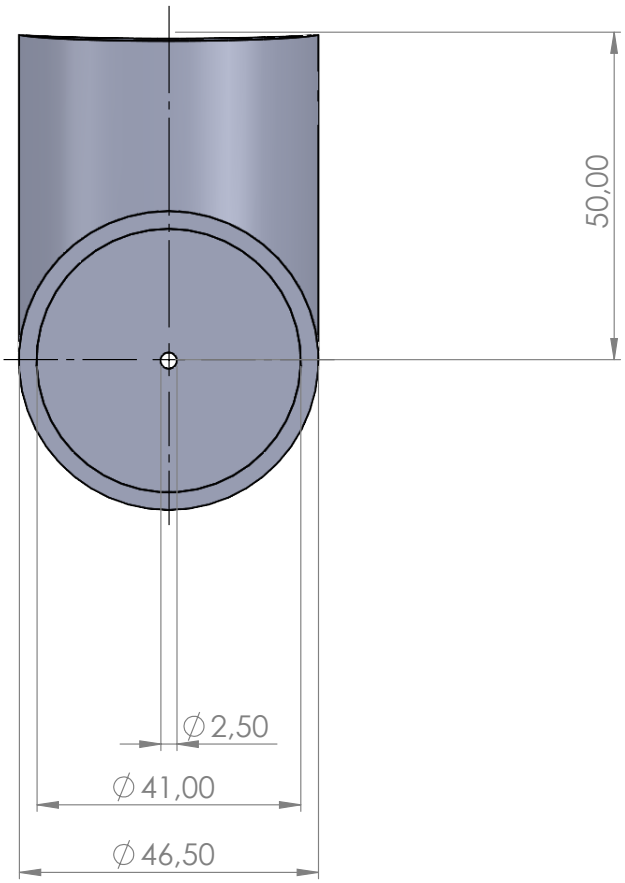
SCALE (2 : 1)



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:	
DRAWN				Actuator Connector	
CHK'D					
APPV'D					
MFG					
Q.A				DWG NO.	A4
			WEIGHT:	SCALE:1:2	SHEET 4 OF 4

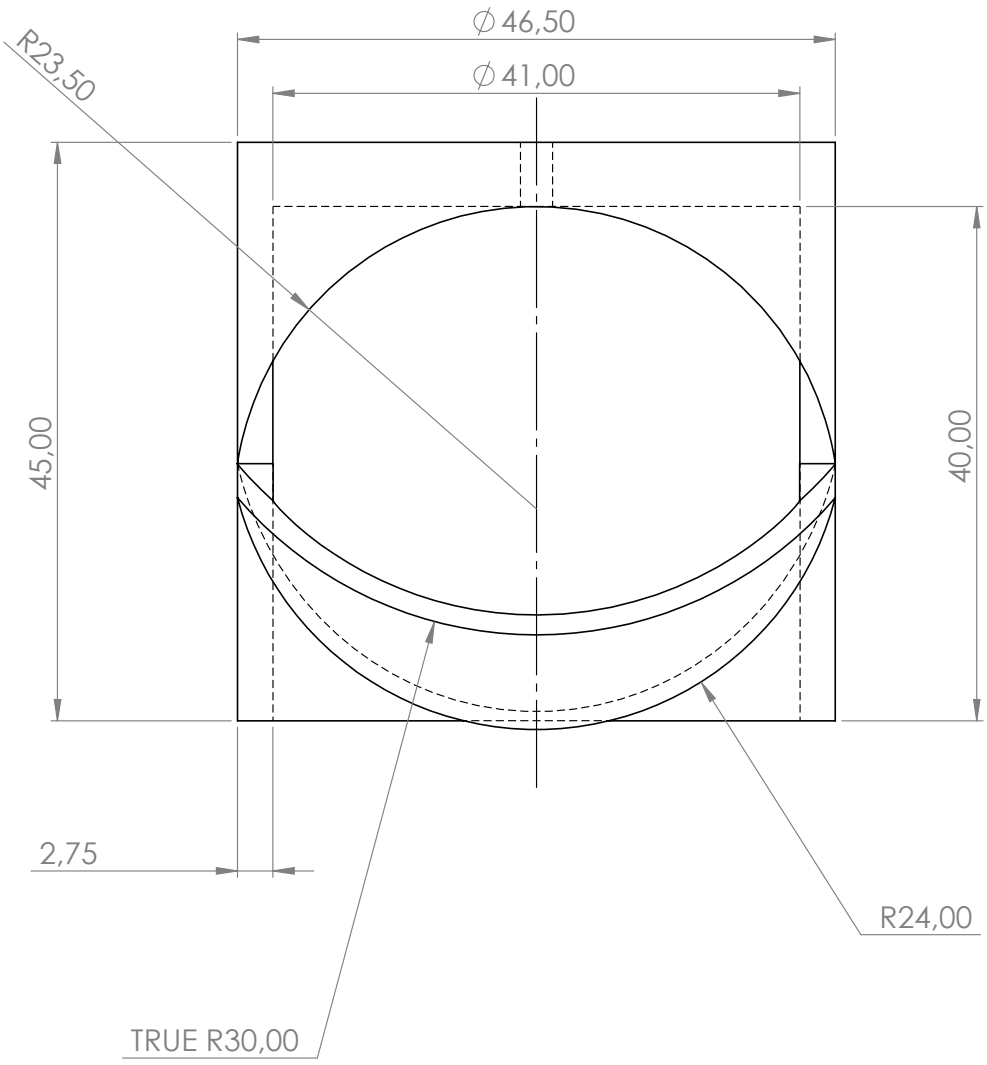
Appendix D Ball pusher





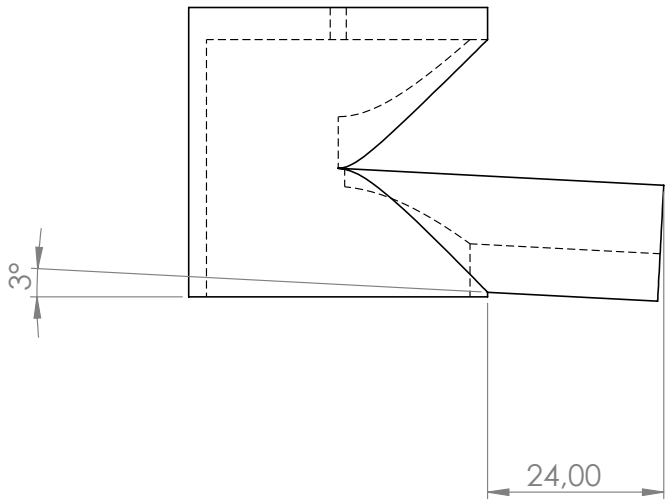
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN						ball pusher	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
				WEIGHT:		SCALE:1:2	SHEET 2 OF 4

SCALE (2 : 1)



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE	TITLE:			
DRAWN				ball pusher			
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO.	A4	
WEIGHT:	SCALE:1:2	SHEET 3 OF 4					

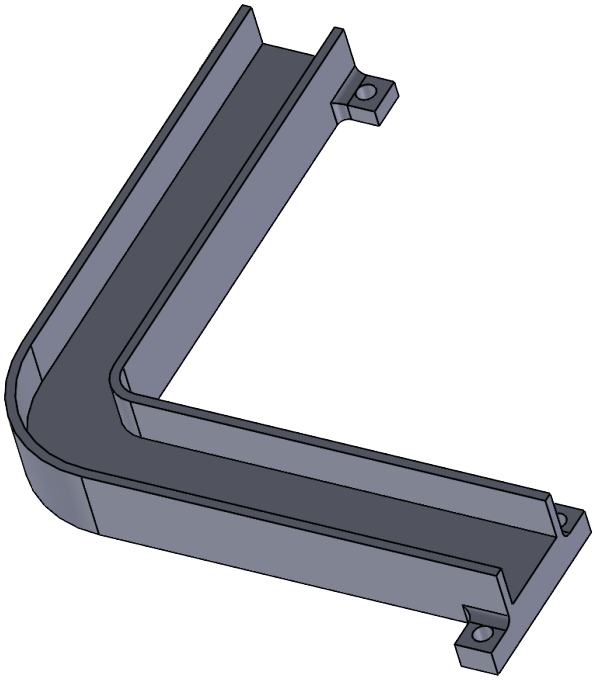
SCALE (1 : 1)



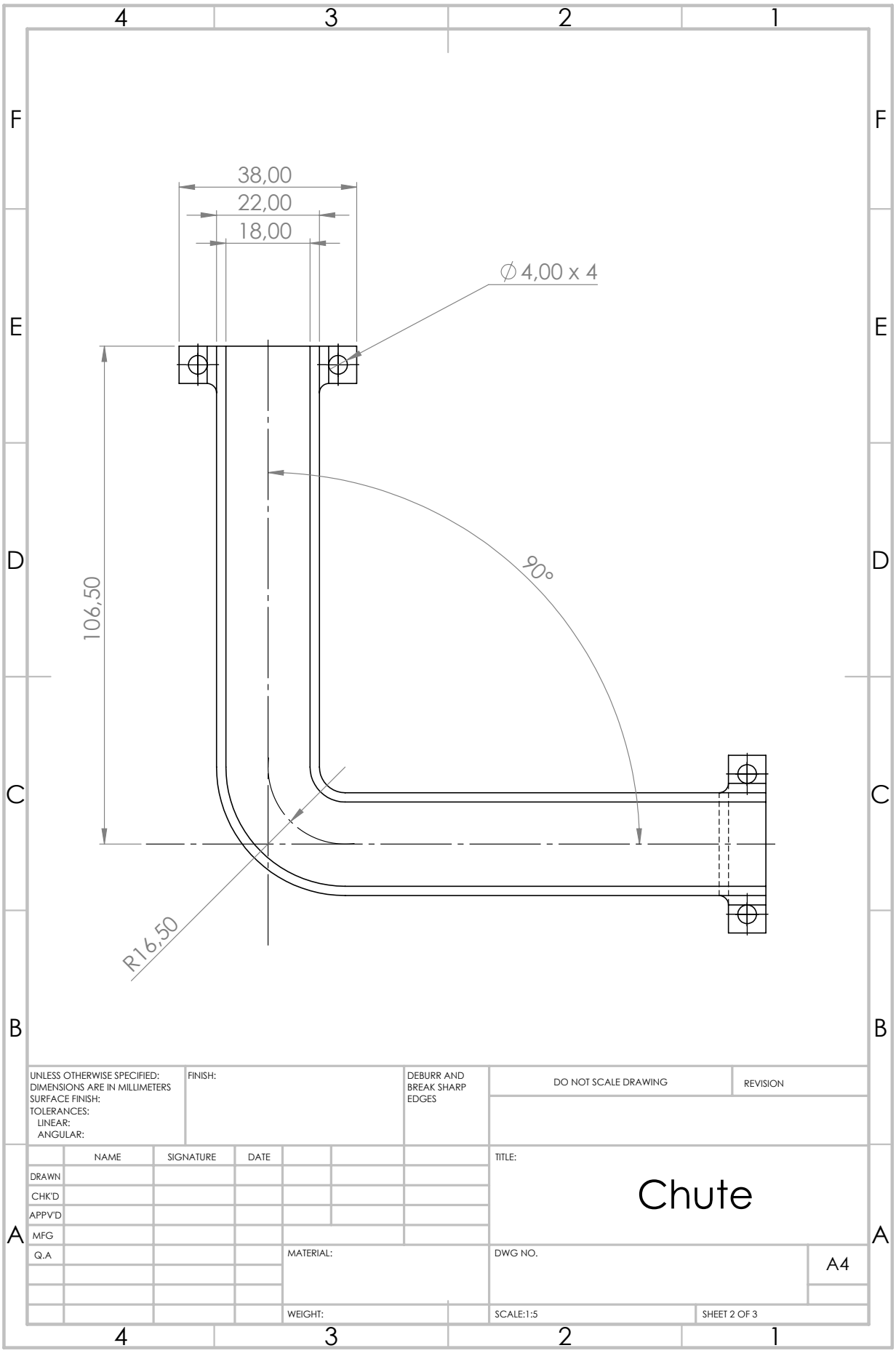
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE	TITLE:			
DRAWN				ball pusher			
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO.	A4	
WEIGHT:		SCALE:1:2		SHEET 4 OF 4			

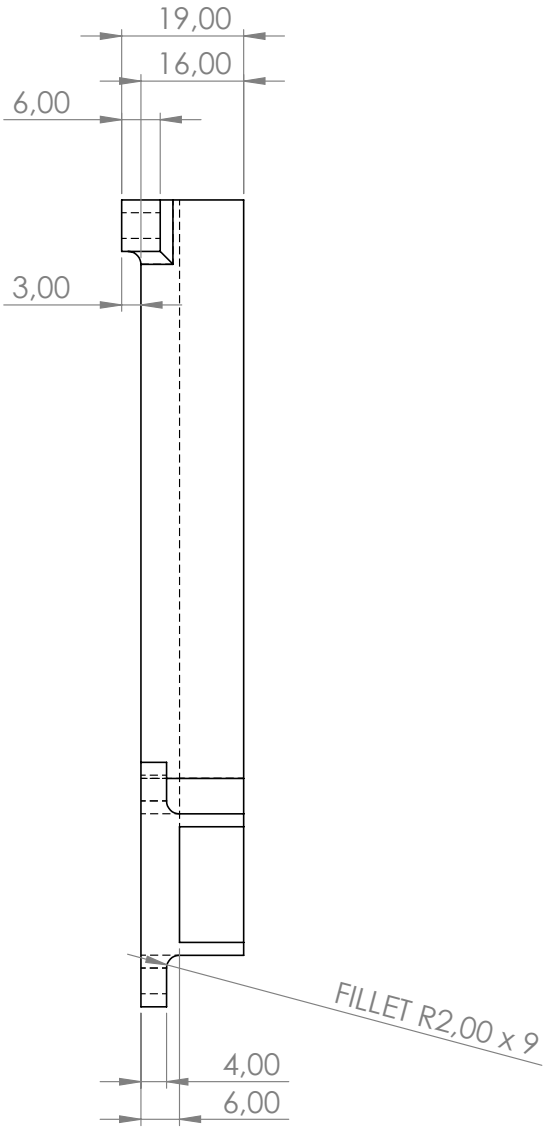
Appendix E Chute

SCALE (1 : 1)



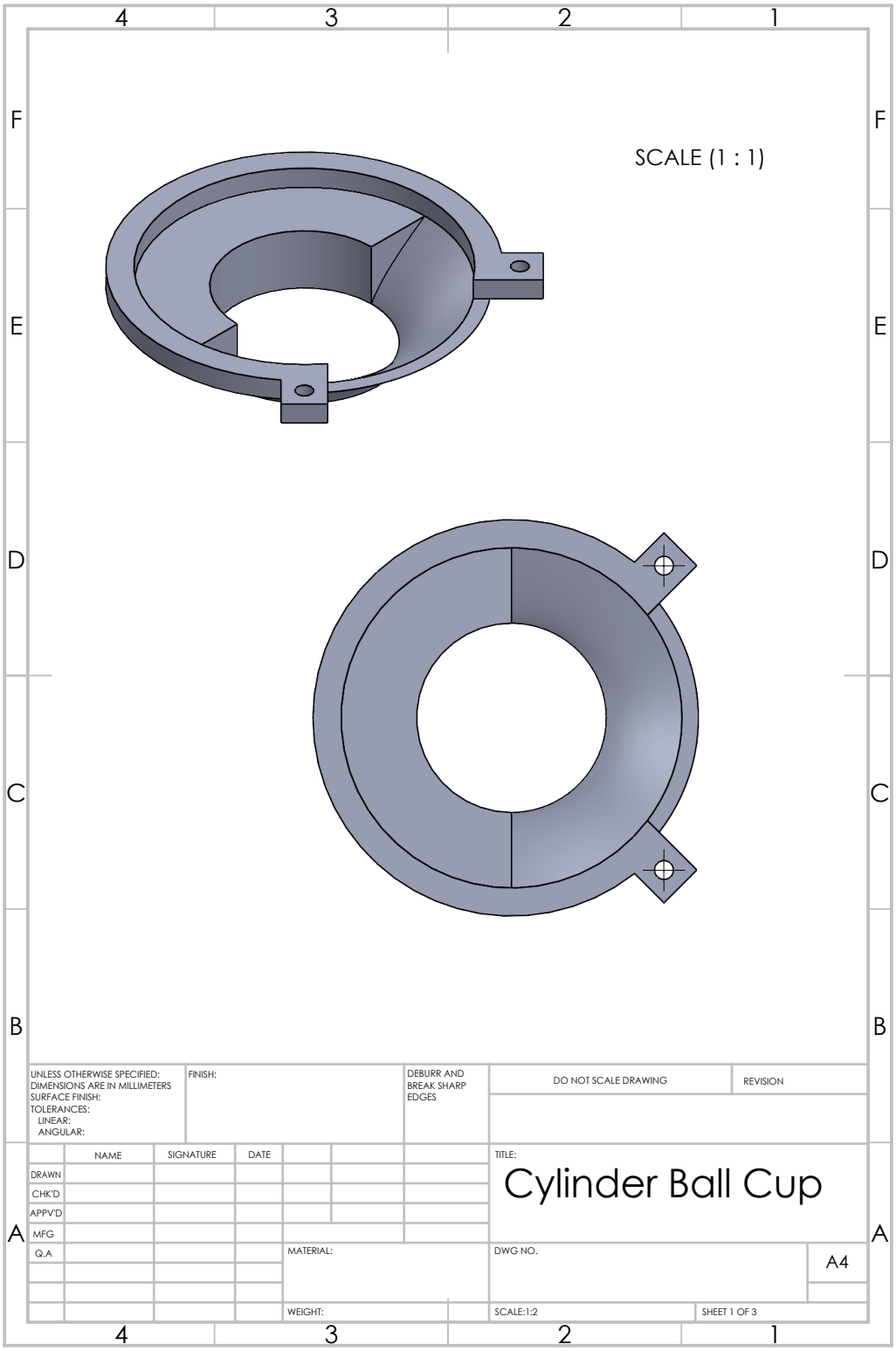
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE		TITLE:	
DRAWN					Chute	
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO.	A4
				WEIGHT:	SCALE:1:5	SHEET 1 OF 3

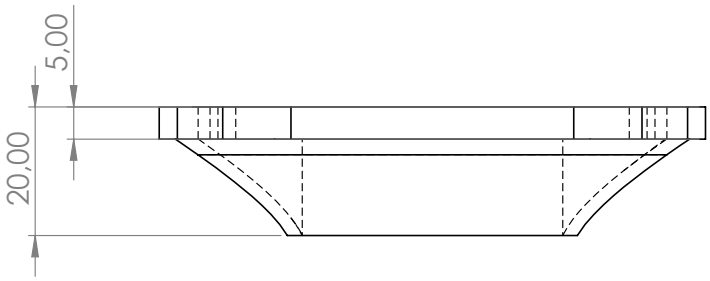

4
3
2
1
F
E
D
C
B
A
38,00
22,00
18,00
∅ 4,00 x 4
106,50
90°
R16,50
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
Chute
MATERIAL:
DWG NO.
A4
WEIGHT:
SCALE:1:5
SHEET 2 OF 3



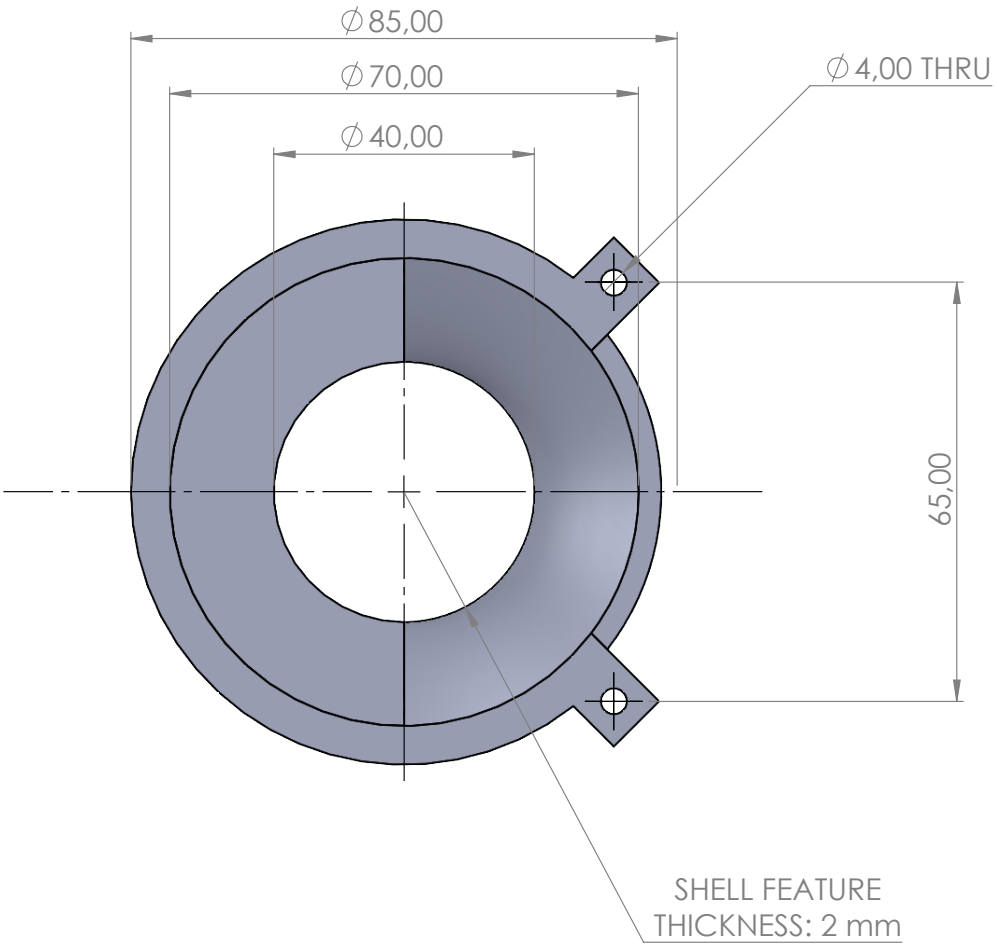
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN						Chute	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
				WEIGHT:		SCALE:1:5	SHEET 3 OF 3

Appendix F Cylinder Ball Cup





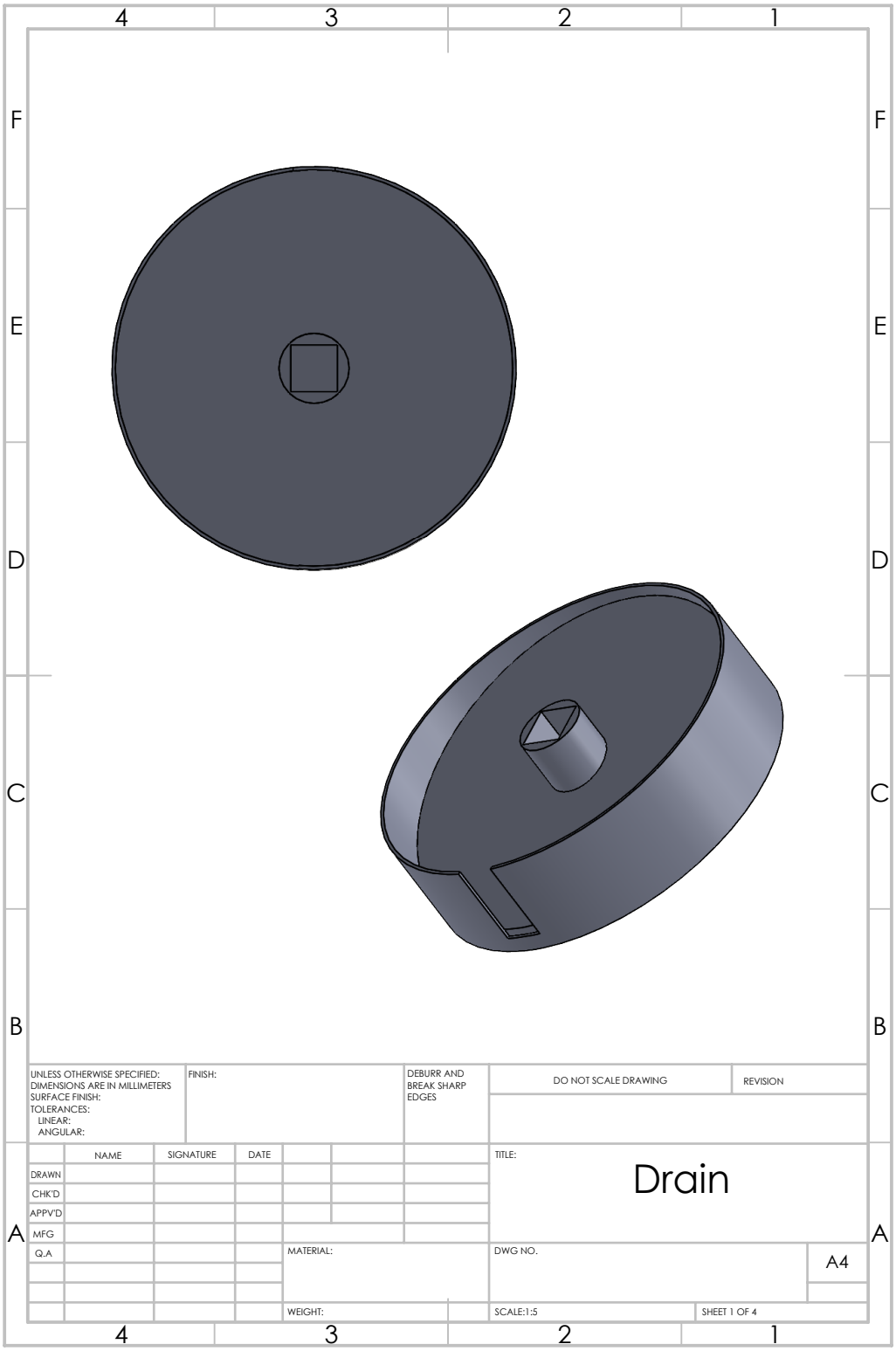
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE	TITLE:			
DRAWN				Cylinder Ball Cup			
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO.	A4	
				WEIGHT:	SCALE:1:2	SHEET 2 OF 3	

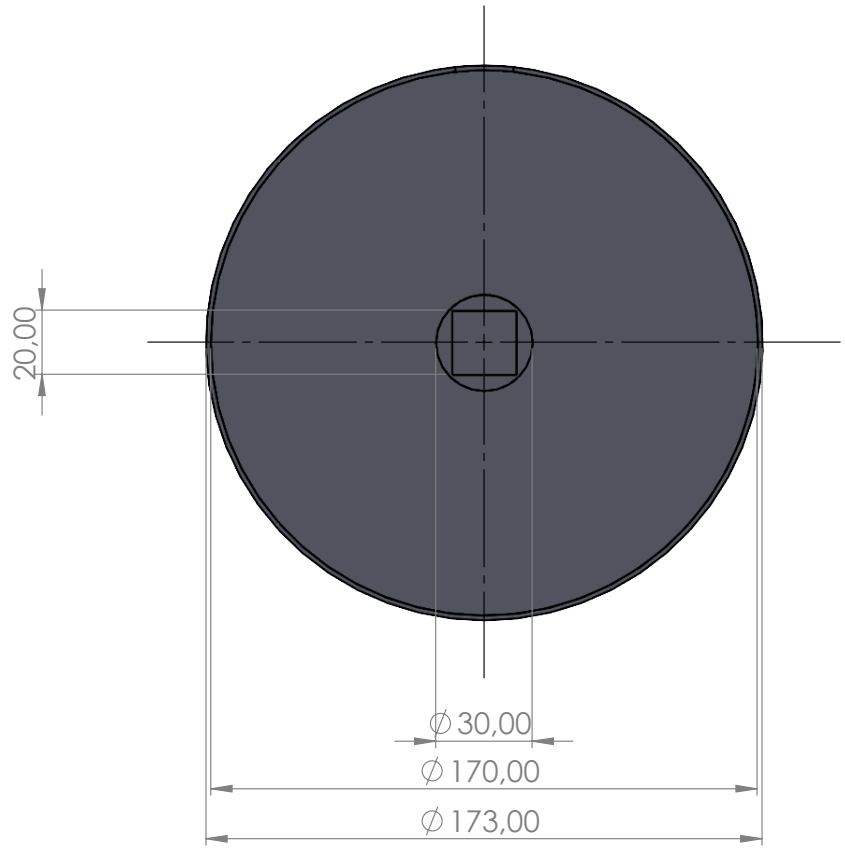


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	

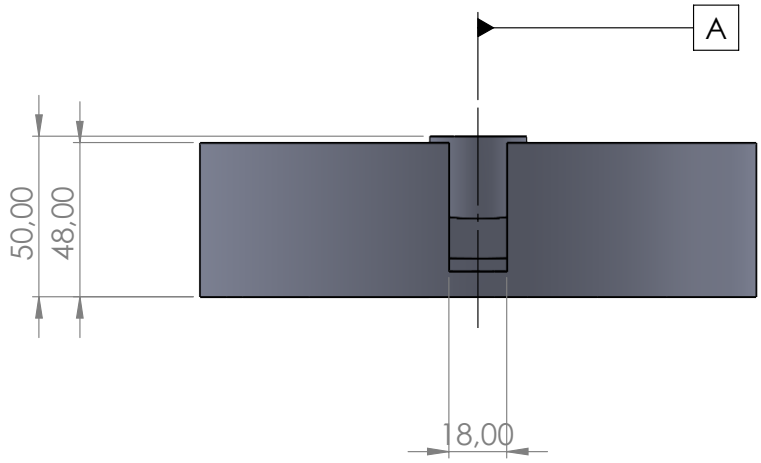
	NAME	SIGNATURE	DATE	TITLE:
DRAWN				Cylinder Ball Cup
CHK'D				
APPV'D				
MFG				
Q.A				
MATERIAL:				DWG NO. A4
WEIGHT:				SCALE:1:2 SHEET 3 OF 3

Appendix G Drain

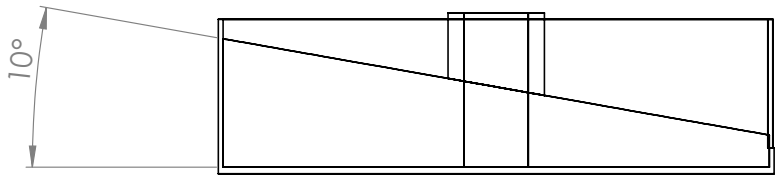




UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN						Drain	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
				WEIGHT:		SCALE:1:5	SHEET 2 OF 4

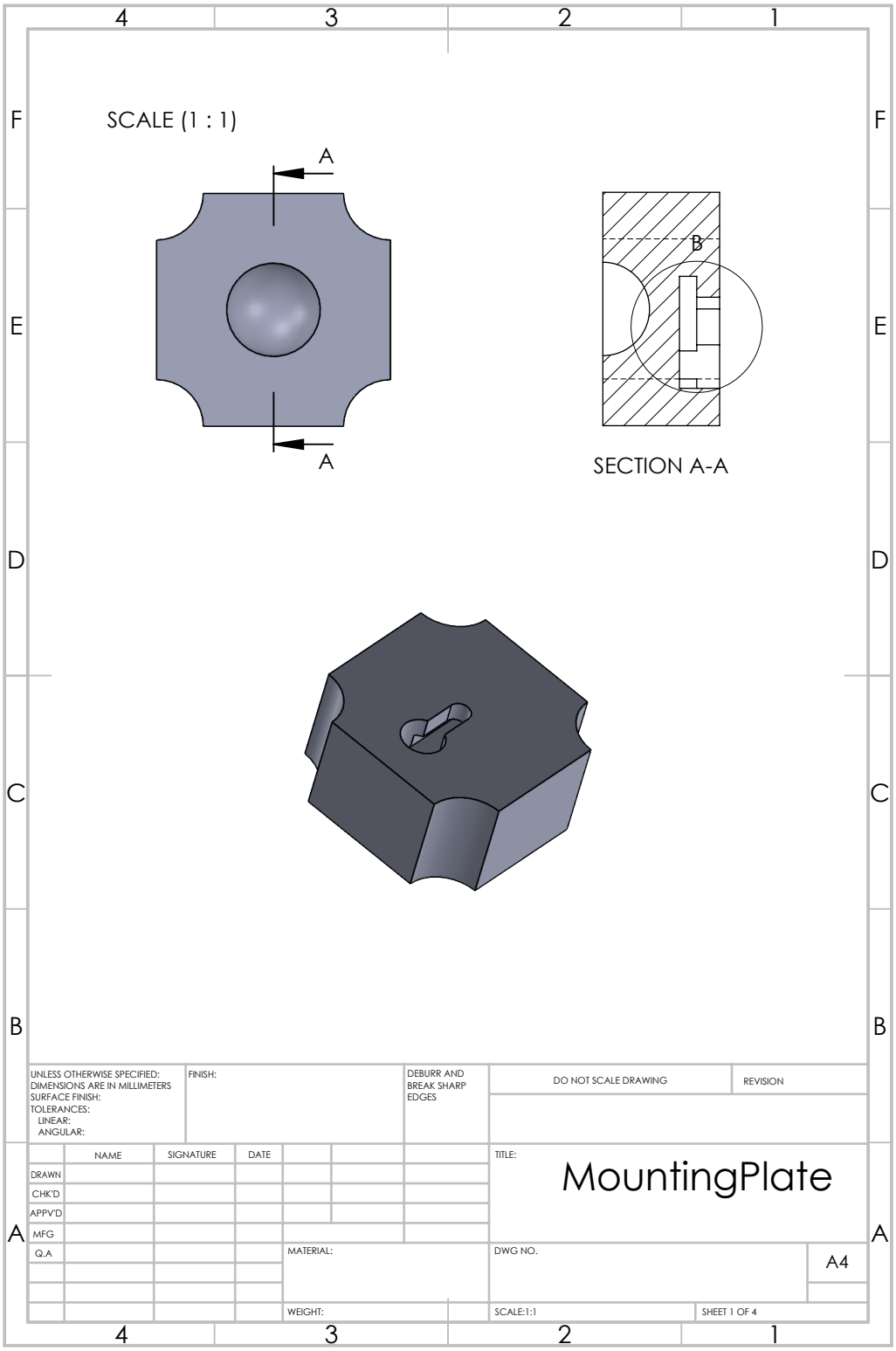


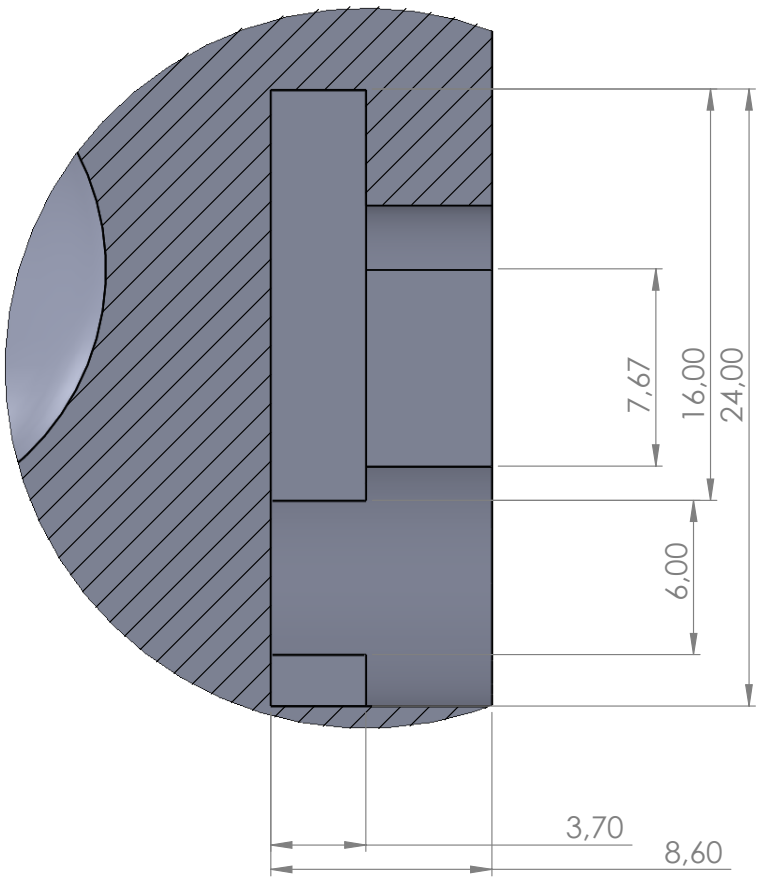
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN						Drain	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
				WEIGHT:		SCALE:1:5	SHEET 3 OF 4



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE	TITLE:			
DRAWN				Drain			
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO.	A4	
				WEIGHT:	SCALE:1:5	SHEET 4 OF 4	

Appendix H MountingPlate

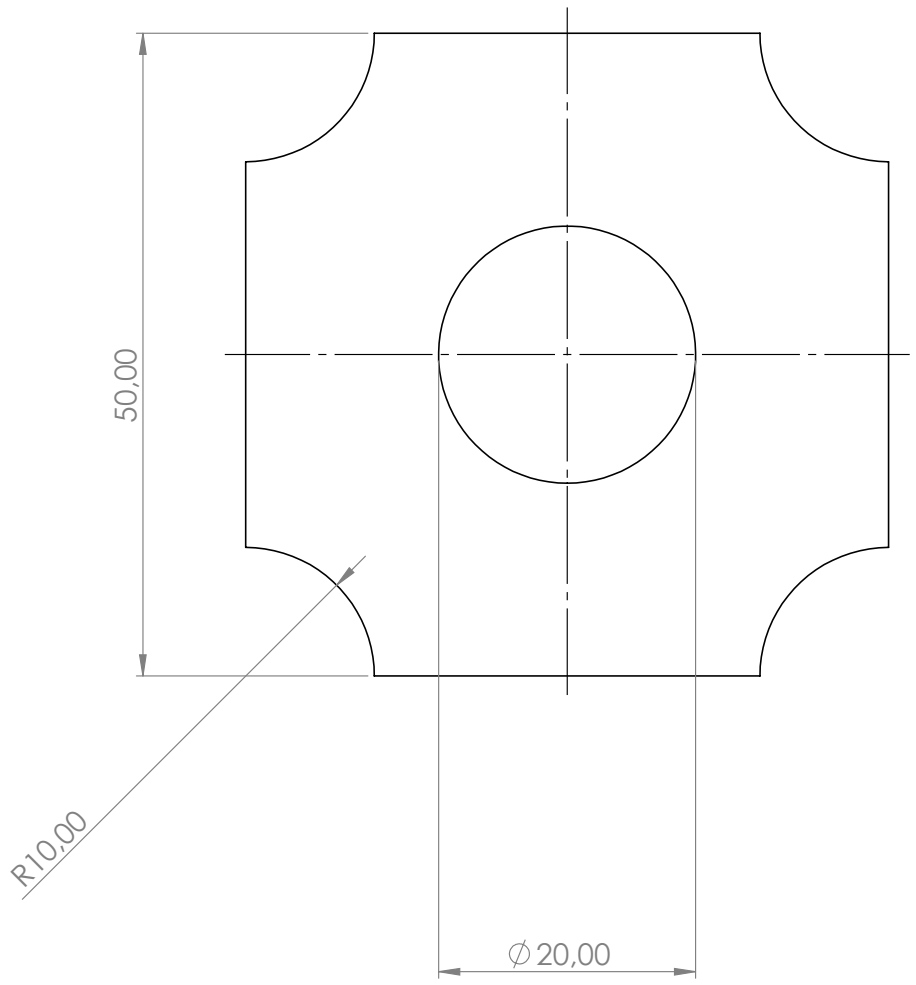




DETAIL B
SCALE 4 : 1

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE: MountingPlate	
DRAWN								
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL:			DWG NO.	A4
				WEIGHT:			SCALE:1:1	SHEET 2 OF 4

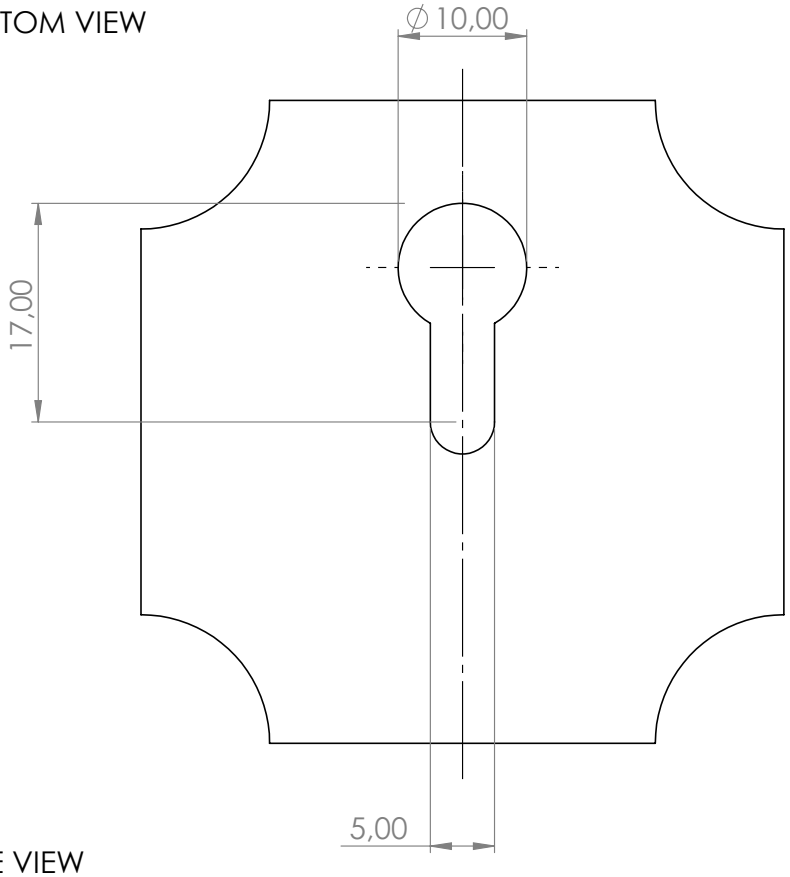
SCALE (2 : 1)



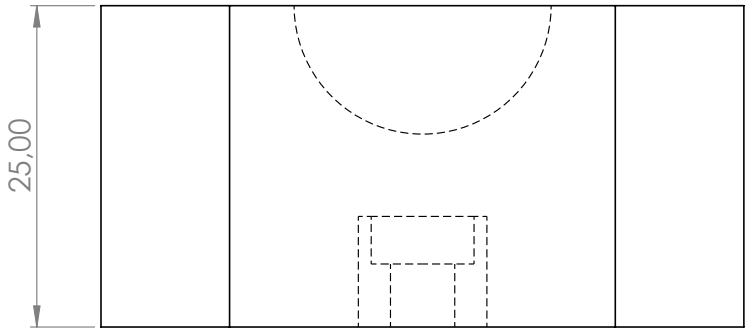
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:	MountingPlate
DRAWN					
CHK'D					
APPV'D					
MFG					
Q.A				MATERIAL:	DWG NO. A4
				WEIGHT:	SCALE:1:1 SHEET 3 OF 4

BOTTOM VIEW

SCALE (2 : 1)

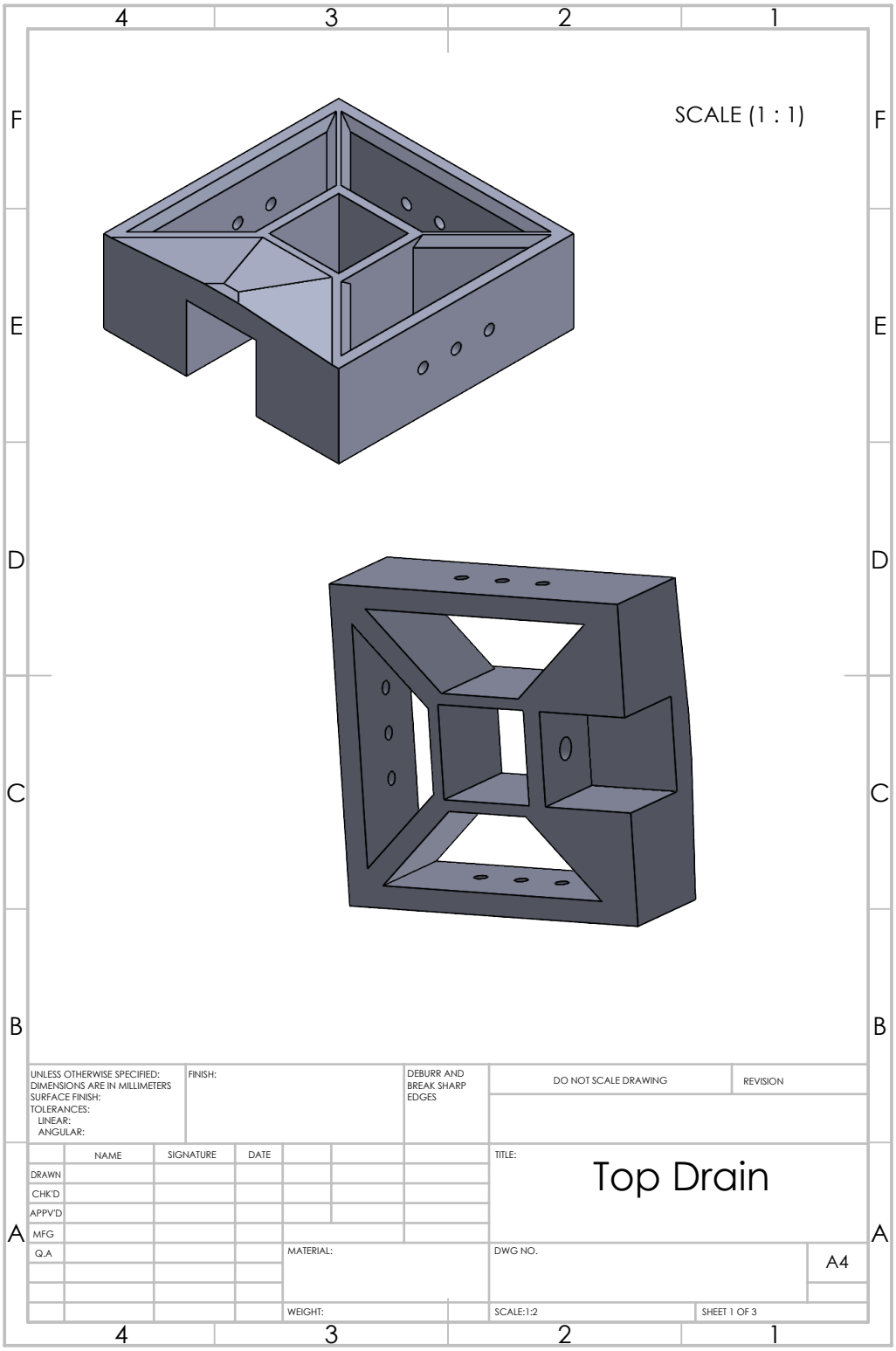


SIDE VIEW

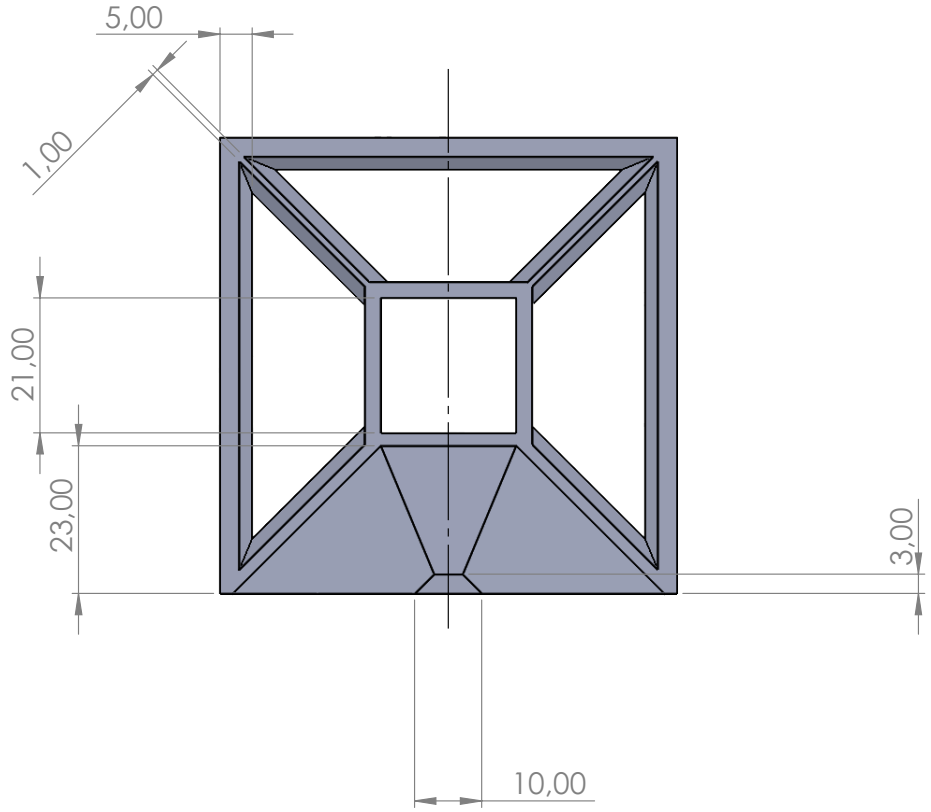


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE	TITLE:			
DRAWN				MountingPlate			
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO.	A4	
				WEIGHT:	SCALE:1:1	SHEET 4 OF 4	

Appendix I Top Drain



SCALE (1 : 1)



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE: Top Drain

MATERIAL:

DWG NO. A4

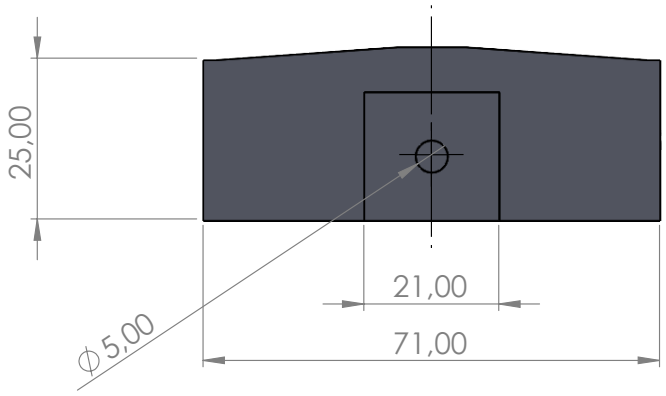
WEIGHT:

SCALE:1:2

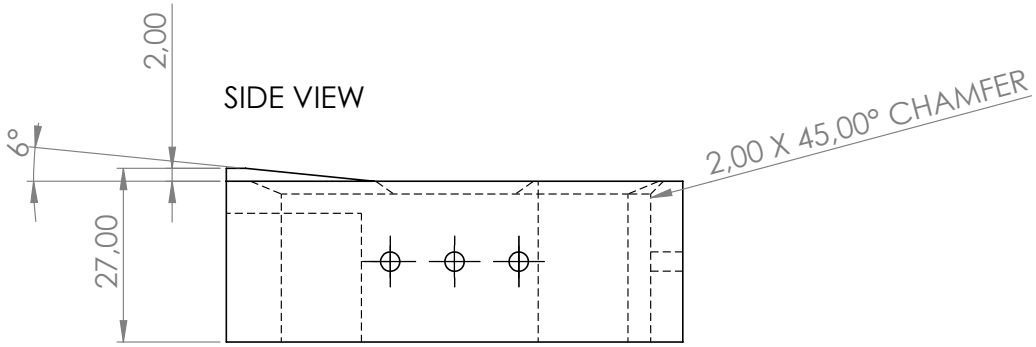
SHEET 2 OF 3

FRONT VIEW

SCALE (1 : 1)



SIDE VIEW



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE: Top Drain

MATERIAL:

DWG NO.

A4

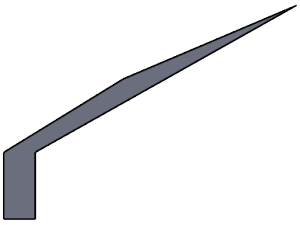
WEIGHT:

SCALE:1:2

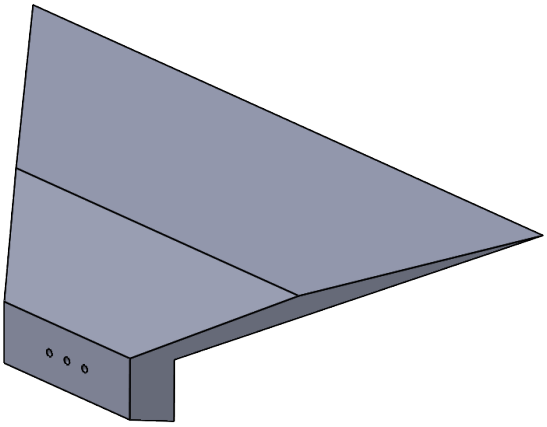
SHEET 3 OF 3

Appendix J Top Drain Side Part

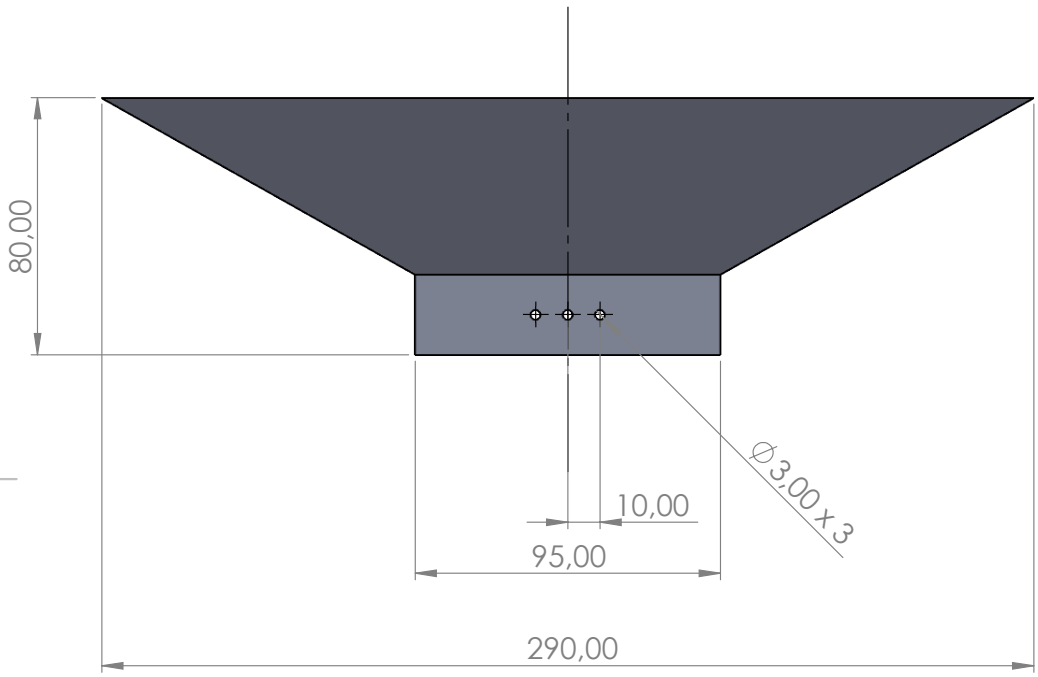
SCALE (1 : 2)



SCALE (1 : 2)



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE: Top Drain Side Part	
DRAWN								
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL:			DWG NO.	A4
				WEIGHT:			SCALE:1:5	SHEET 1 OF 3



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	

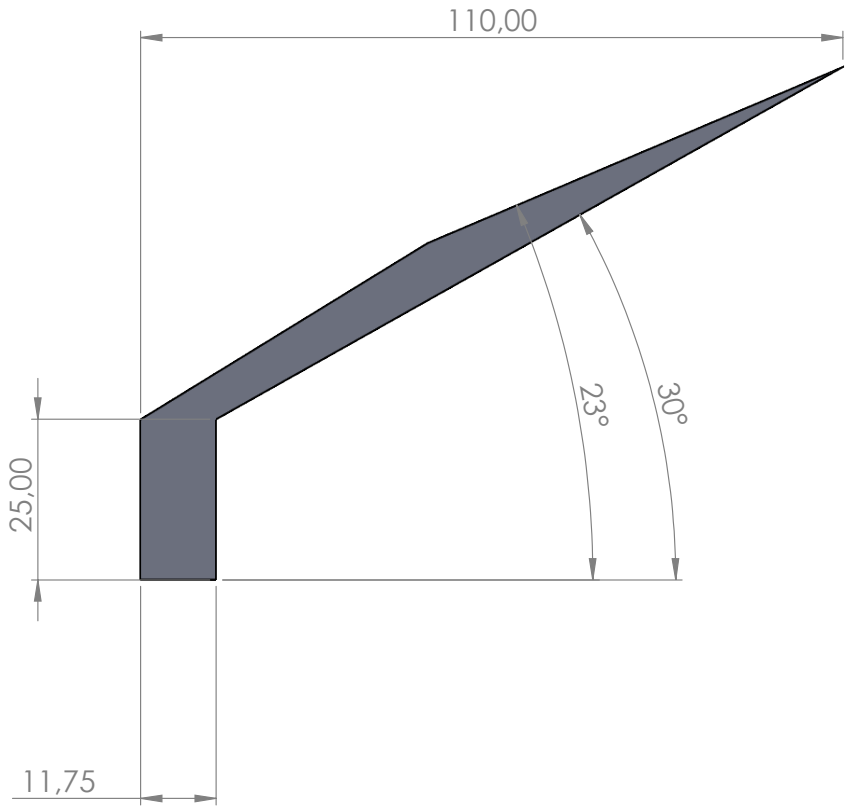
	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE:

Top Drain Side Part

MATERIAL:	DWG NO.	A4
WEIGHT:	SCALE:1:5	SHEET 2 OF 3

SCALE (1 : 1)



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE: Top Drain Side Part

MATERIAL:

DWG NO. A4

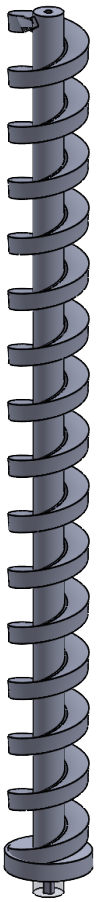
WEIGHT:

SCALE:1:5

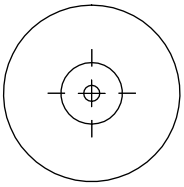
SHEET 3 OF 3

Appendix K Helix Elevator

SCALE (1 : 2)



SCALE (1 : 1)

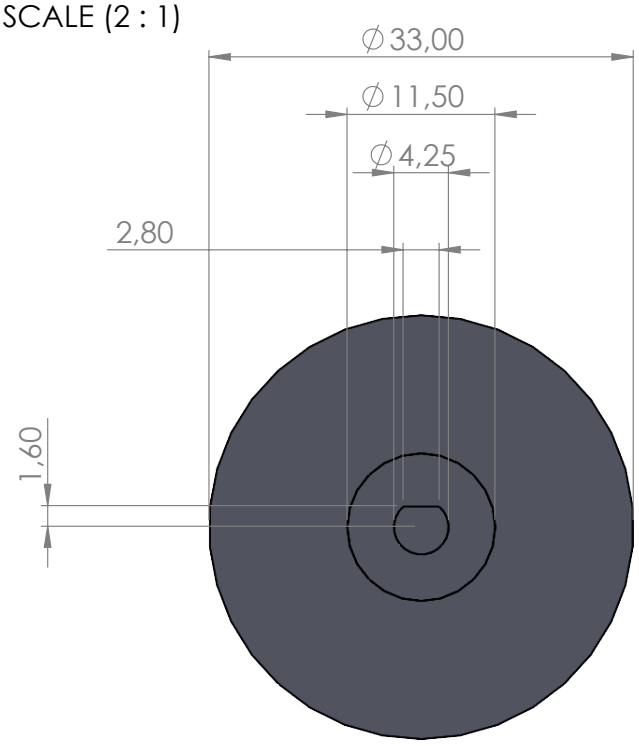


HELIX DIM

REV: 15
PITCH: 22 mm

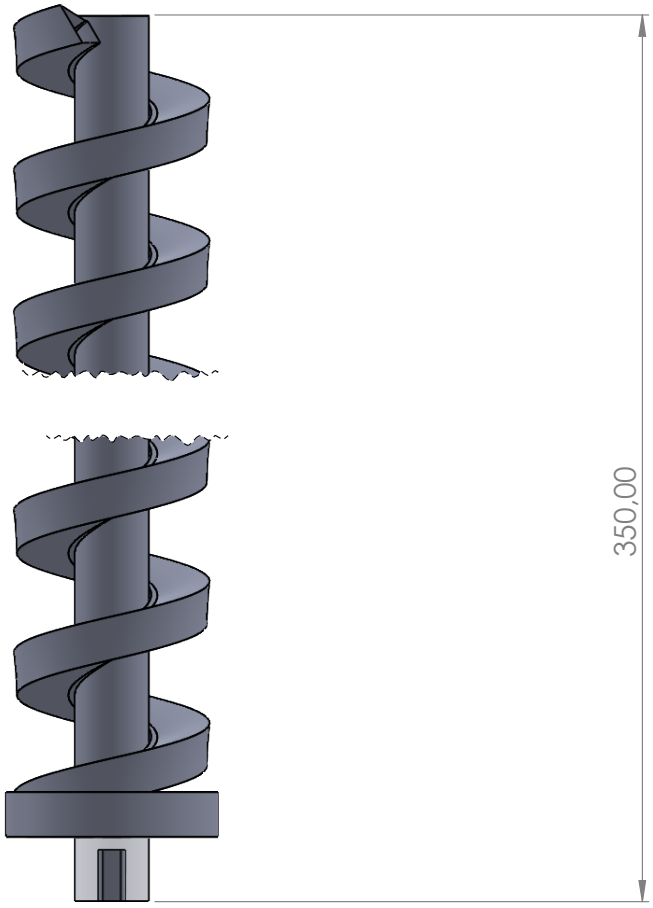
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE: Helix Elevator	
DRAWN								
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL:			DWG NO.	A4
				WEIGHT:			SCALE:1:5	SHEET 1 OF 3

SCALE (2 : 1)



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION		
	NAME	SIGNATURE	DATE			TITLE: Helix Elevator	
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
				WEIGHT:		SCALE:1:5	SHEET 2 OF 3

SCALE (1 : 1)



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE: Helix Elevator

MATERIAL:

DWG NO. A4

WEIGHT:

SCALE:1:5

SHEET 3 OF 3