Mechatronics 2
Final Project Report

Thomas Kallmann, Yassmeen Youssef, Joseph Neal, Jack Franz,
Thanushraam Suresh Kumar, Daniel Vesselovskii

May 5, 2025

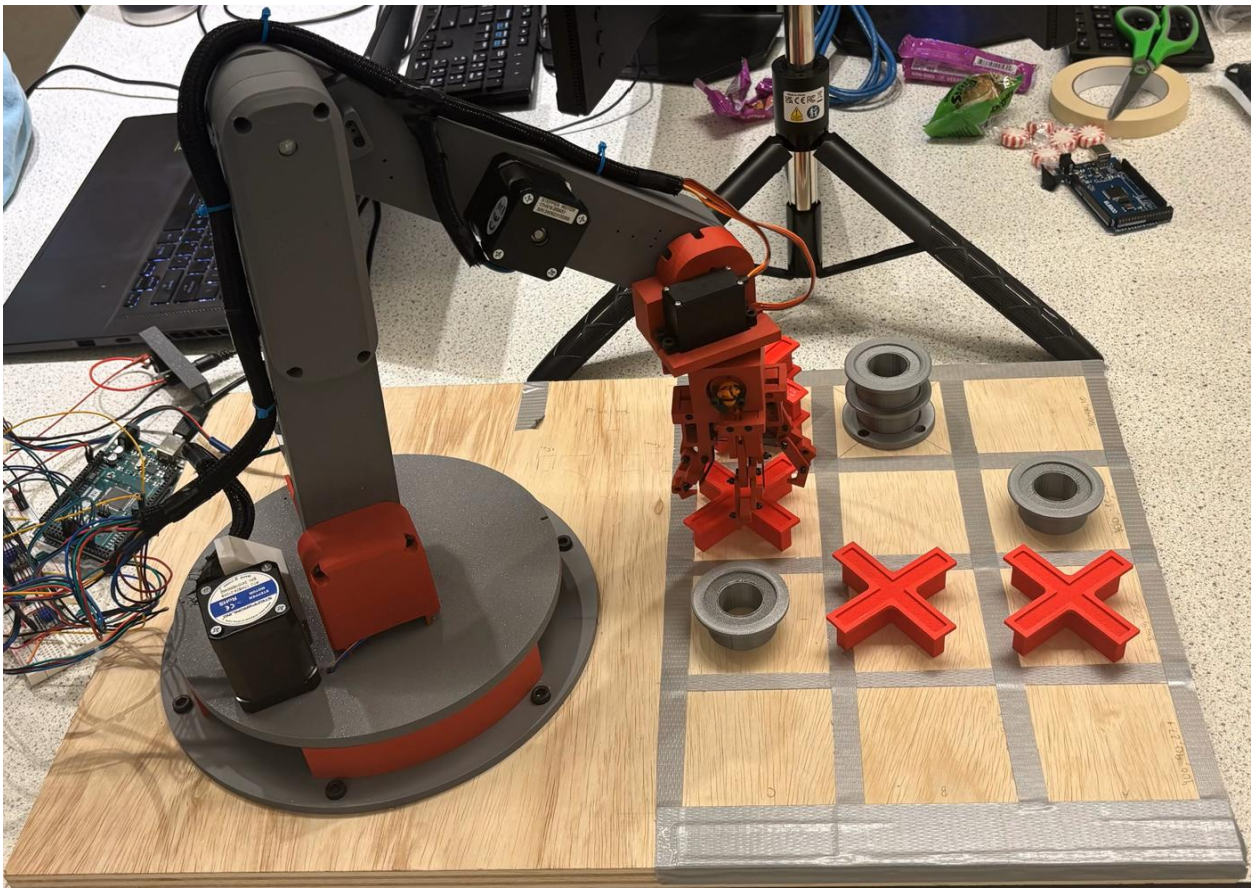

TABLE OF CONTENTS

ABSTRACT

Human-robot interaction (HRI) is an increasingly growing field aimed at exploring safe and efficient exchanges between humans and robots. As robots become increasingly integrated into our daily lives, this field will continue to grow in importance as one of the key limitations of robotic integration is ensuring the safety and well-being of surrounding humans. Thus, team Raspberry Fly aimed to explore the various aspects of HRI by creating a safe and interactive Tic-Tac-Toe game between a human and a robot.

The robot is designed to autonomously play a game of Tic-Tac-Toe. It achieves this through three primary subsystems: a vision and control, a 4-degree of freedom arm, and a 2-degree of freedom end effector. The robot and human interact via the gameboard, wherein the robot first places an X or an O in their desired location. After the human places their game piece, the vision subsystem, positioned above the gameboard, detects this state change, prompting the controls logic to calculate the most strategic location to place the opposing piece. The inverse kinematics to move the arm to this location is then calculated and the arm and end effector proceed to pick up a game piece and place it in the correct location. After completing a turn, the robot homes to a set location away from the gameboard to ensure the human's safety and awaits the human to proceed with its turn.

This system promotes safe human-robot interaction and may help improve public sentiment on integrating robots in our daily lives.

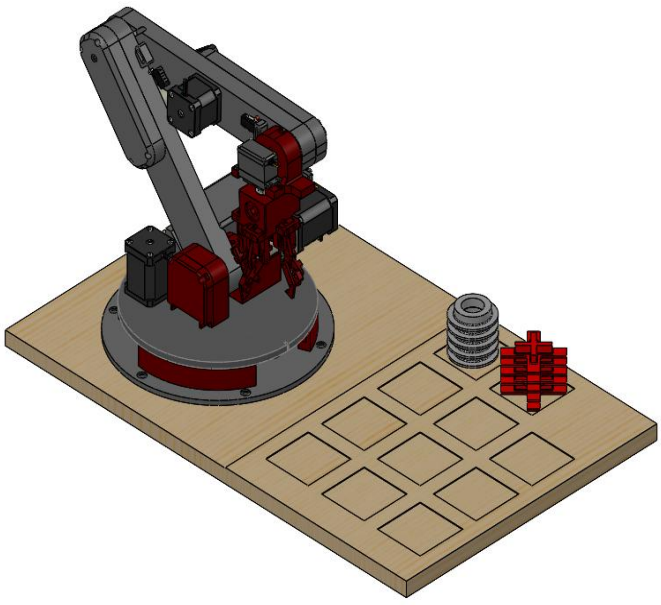

Figure 1: Final CAD assembly

SYSTEM COMPONENTS

In this section, the robot's electronic and physical components form, and function are defined.

Table 1: Electronics Components Form and Function

Part Name	Function	Number of Parts:
TMC2209 Stepper Motor Driver	Each part controls one of the four stepper motors used.	4
Limit Switch	Detects when each robot member is in homed position.	4
Voltage Step Down	Used to step down 12V line to 5V for servo motors	1
Arduino Mega	Controls stepper motor drivers and end effector servo motors.	1
RealSense Camera	Detects changes in the location of the game pieces on the game board.	1
Printed Circuit Board	Connects stepper motors and servo motors with Arduino Mega.	1

Table 2: Physical Component Form and Function

Part Name	Function	Number of Parts
Motors		
Nema 17 Bipolar 45Ncm	Used for the first arm translational DOF.	1
Nema 17 Bipolar 65Ncm	Used for the base vertical and horizontal rotational DOFs.	2
Nema 17 Bipolar 59Ncm	Used for the second arm translational DOF.	1
MG995 Servo Motor	One is used for the rotational DOF of the end effector and the other to actuate the end effector between the open and closed positions.	2
Gear Boxes		
5:1 Gearbox	Increases the motor output of the base rotational DOF.	1
10:1 Gearbox	Increases the motor output of the first arm's translational DOF.	1
20:1 Gearbox	Increases the motor output of the base translational DOF.	1
Miscellaneous		
608 2RS Bearings	Used for rotational joints at link 1 and link 2	6

6202-2RS Bearings	Used for joint 1 rotation	1
693ZZ Bearings	Used to create tension in the pulleys	20
6810ZZ Bearings	Used for base rotation	1
8mm Flange Coupling	Secures output shaft of the gearbox to robot arm.	3
8mm Bore Pulleys	Connects output of gearbox to the rotation points of the arm.	3
5mm Bore Pulleys	Connects out of stepper to rotation joint	2
Threaded Inserts Set	Secures robot to wooden gameboard.	1
Bolts/Nuts Set	Secures robot parts to each other.	1
8mm x 50mm Aluminum Rod	Rotation Joint for link 2 and end effector	2
Nylon Lock Nut Set	Ensures bolts stay in desired location.	1
PLA	Used to 3D print most of the physical components of the arm.	2
Small Rubber Bands	Creates tension in the end effector to keep fingers closed.	8
Nylon String	Connects fingers of end effector to spool to open and close it.	1

MOBILITY SUBSYSTEM

Without considering the end-effector, the robot's 4 degrees of freedom are defined by the following joints: rotation about the vertical axis at the robot's base, and rotation about parallel horizontal axes at the three joints connecting the base, two links, and the end effector. During the initial design stages, we chose to actuate these joints using stepper motors. We estimated preliminary torque requirements for our application and discussed construction techniques that would allow us to use parts that suited our application while reducing cost where possible.

We used various NEMA 17s for all primary joints. A motor rated for 65Ncm of holding torque in conjunction with a custom 6:1 timing belt gear reduction was used for the base's horizontal rotation. For the shoulder joint connected the base, we used an identical motor with the addition of a 10:1 planetary gearbox to ensure a holding torque sufficient for supporting the weight of the fully extended arm. For subsequent joints, we used a 59Ncm motor with a 10:1 for the second link, and a 45Ncm motor for the end effector joint. To reduce load on the most weight bearing joints, we used timing belts to connect the second and third motors mentioned to their respective joints. This allowed us to place the motors in such a way that the center of mass could be shifted closer to the base, reducing compounding torques.

LOW-LEVEL CONTROL

All motors were driven by TMC2209 stepper motor drivers controlled by an Arduino Mega. Rather than configuring the steppers via the TMC2209's simplex UART pin, all drivers were

operated in legacy mode with manual current configuration. To this end, we developed a custom Arduino stepper motor library which not only allowed us to easily configure microstepping for each driver in software but also streamlined the integration of other control subsystems. Additionally, this custom library allowed us to directly integrate the various gear ratios we used into the low-level controller. Aided by having utilized timing belts to reduce load on each motor and having tuned each driver's reference voltage to meet our design parameters, we were able to power all four stepper motors using a single 12V 4A DC power converter.

```
// Constructor
TMC2209LEGACY(int step_pin, int dir_pin, int en_pin, int ms1_pin, int ms2_pin);

// Methods
void initializeDriver();              // Initializes driver pins, zeros current position, sets microstepping to default value of 8
void step();                          // Complete 1 step.  Direction must be changed using setDirection();
void step(int delay);                 // Used to manually set delay between steps in microseconds (recommended above 20 at default microstepping).
void changeDirection();               // Flips direction of step().
void stepToAngle();                   // Step once towards targetAngle (default is 0).  Automatically takes the shortest of the two paths.
void stepToAngle(int delay);          // Same as step(int delay)

// Getters
int getMicrosteps();                  // Returns current microstepping setting.
int getGearRatio();                   // Returns current gear ratio setting.
bool getDirection();                  // Returns current direction of step().
float getAngle();                     // Returns current angle.
float getDegreesPerStep();            // Return the physical angle that corresponds to a single step.
float getTargetAngle();               // Returns targetAngle.
float getAngleDif();                  // Returns distance from current angle to target angle.
long getStepDif();                    // Returns the distance in steps from the current angle to the target angle.
long getNetSteps();                   // Returns the clockwise distance in steps from zero position.

// Setters
void setDirection(bool dir);          // Set direction of step().
void setMicrosteps(int ms);           // Set desired microstepping: 8 (Default), 16, 32, 64
void setGearRatio(int gr);            // Set desired gear ratio.
void setAngle(float newAngle);        // Set new zero position to (current stepper position - newAngle).
void setTargetAngle(float toAngle);   // Sets the angle which stepToAngle() steps towards.
```

Figure 2: Basic documentation for TMC2209LEGACY custom library.

(Full source code for library in Appendix)

HIGH-LEVEL CONTROL AND INVERSE KINEMATICS

Our high-level controller needed to be able to send joint angles to the Arduino controller based on desired coordinates (x, y, z) of the end-effector, and thus needed to calculate the inverse kinematics for our system.

Due to the nature of the specific application of our robot, we were able to significantly simplify the calculation of the inverse kinematics, allowing us to take a strictly trigonometric approach. Because the end-effector would always be facing straight down, and because the orientation of the gripper remains constant relative to the orientation of the playing space, we were able to strictly calculate the base's angle of orientation (and the gripper's orientation to match) by taking the tangent of the X and Y coordinates of our desired end effector position. The end effector's joint angle could simply be calculated by taking the difference between the angle of the first

linkage relative to the base and the angle of the second linkage relative to the first linkage. From here, our system is effectively reduced from a 5 degree-of-freedom system to a 2 degree-of-freedom planar system, where we solve for the angle of joint 1 (connecting the base with linkage 1) and joint 2 (connecting the base with linkage 2) using the law of cosines as follows:

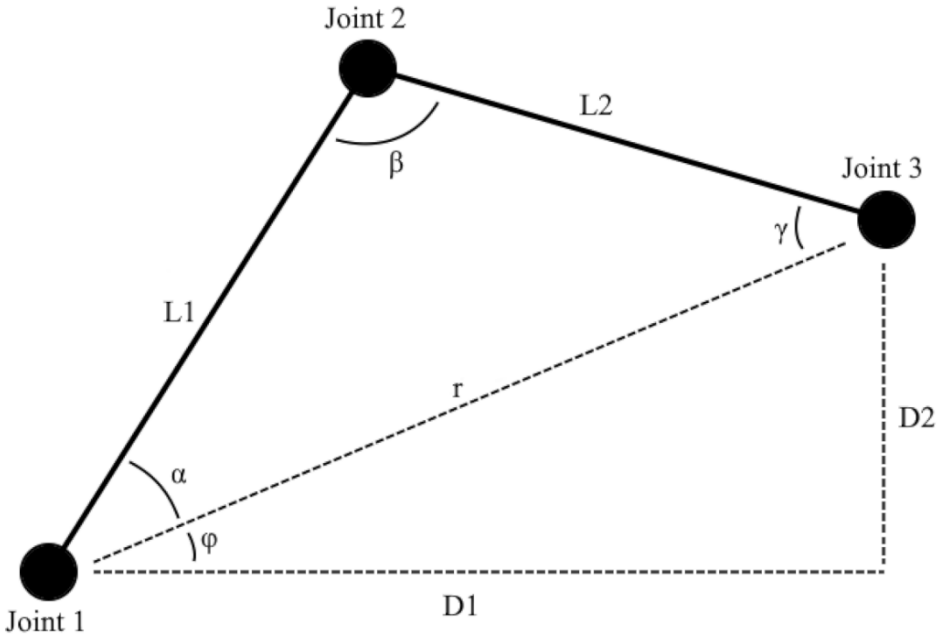

Figure 3: Trigonometric approach to IK

We start by defining D1 and D2 in terms of our desired coordinates so that

$$D1 = \sqrt{(x^2 + y^2)} \text{ and } D2 = z + k \quad (1)$$

Here, k is a constant equal to the distance from the tip of the end effector to joint 3, based on the assumption that this distance is strictly in the Z-direction. From this, we calculate φ using trigonometric identities and r using the Pythagorean theorem.

$$\varphi = \tan^{-1}\left(\frac{D2}{D1}\right) \text{ and } r = \sqrt{(D1^2 + D2^2)} \quad (2)$$

Now, α and β can be calculated using the law of cosines.

$$\alpha = \cos^{-1}\left(\frac{L2^2 - r^2 - L1^2}{-2 \cdot r \cdot L1}\right) \text{ and } \beta = \cos^{-1}\left(\frac{r^2 - L1^2 - L2^2}{-2 \cdot L1 \cdot L2}\right) \quad (3)$$

We know that the angle at Joint 1, $J_1 = \alpha + \varphi$ and the angle at Joint 2, $J_2 = \beta$. So, combining (1), (2), and (3), we arrive at the following:

$$J1 = \cos^{-1}\left(\frac{L2^2 - L1^2 - x^2 - y^2 - (z+k)^2}{-2 \cdot L1 \cdot \sqrt{x^2 + y^2 + (z+k)^2}}\right) + \tan^{-1}\left(\frac{z+k}{\sqrt{x^2 + y^2}}\right) \quad (4)$$

$$J2 = \cos^{-1}\left(\frac{x^2 + y^2 + (z+k)^2 - L1^2 - L2^2}{2 \cdot L1 \cdot L2}\right) \quad (5)$$

We can now simply calculate our remaining joints as follows, according to the assumptions we have made about our system.

$$J0 = \tan^{-1}\left(\frac{y}{x}\right) \quad (6)$$

$$J3 = 90 + J1 - J2 \quad (7)$$

$$J4 = -J0 \quad (8)$$

PRINTED CIRCUIT BOARD

The custom PCB used in this project was designed using Altium Designer and was implemented as a removable shield compatible with the Arduino Mega. As shown in Figure 4, the schematic organizes the key power and signal pathways from the Arduino to four TMC2209 stepper motor drivers, each of which connects to a dedicated four-pin header terminal for motor output. Additionally, the board includes two three-pin headers for controlling servo motors.

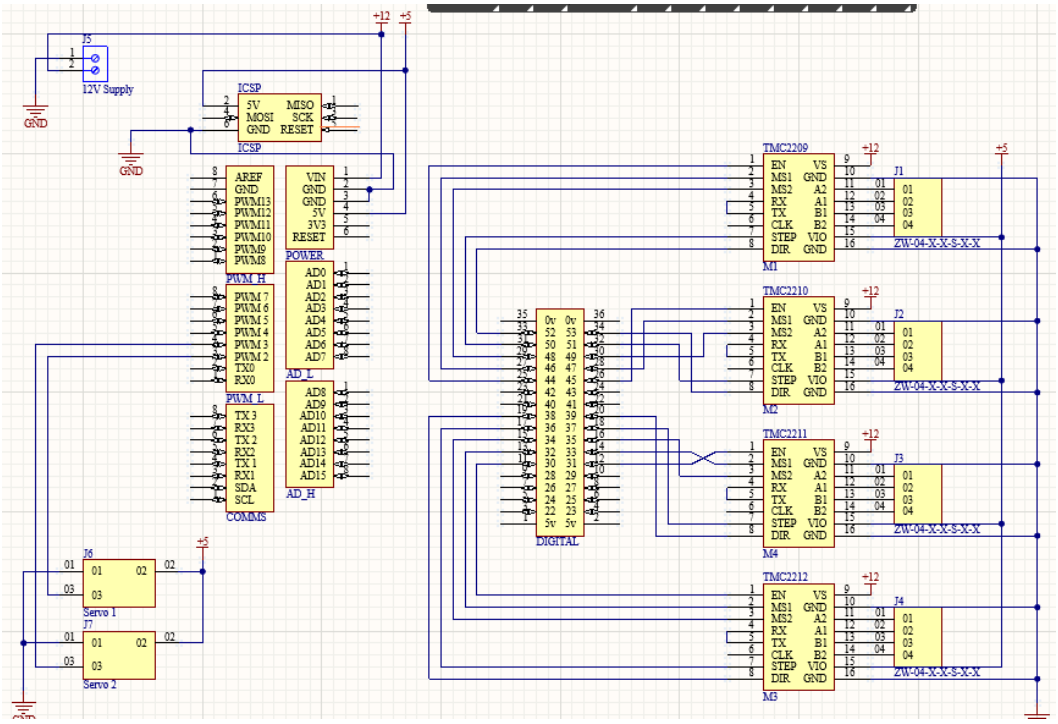

Figure 4: Circuit Schematic Designed for our PCB

Figure 5 illustrates the physical layout of the PCB, with each motor driver labeled (M1–M4) and positioned to minimize routing complexity while optimizing space. The design was intended to enable easy replacement or debugging of the microcontroller by allowing the shield to be detached from the Arduino if needed.

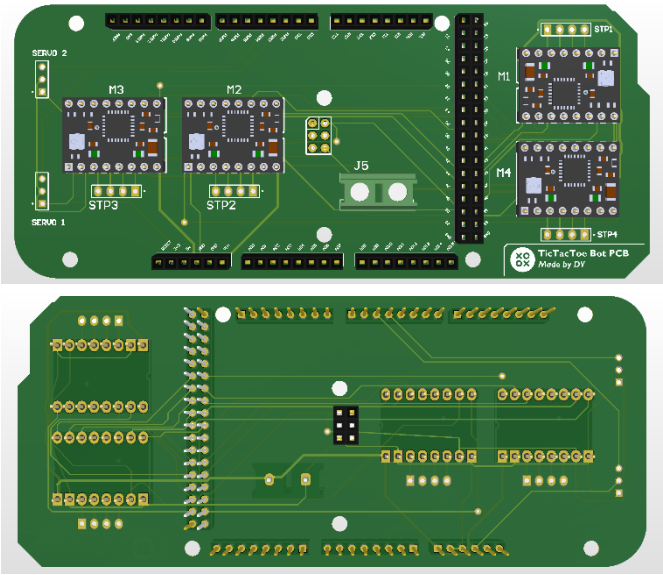

Figure 5: Layout of the Final PCB w/ Soldered Components & Stackable Header Pins

However, during system testing, we observed a critical design oversight. Power to the stepper drivers and servos was routed through shared Arduino connections, which resulted in excessive current draw through the Arduino's onboard voltage regulator. Once all four stepper drivers and both servos were engaged, the cumulative current exceeded the safe limits for the Arduino, ultimately damaging its regulator. This incident underscored the importance of isolating high-current components from low-power control electronics. Future designs should include independent power rails or use opto-isolation to prevent high-load components from directly impacting the microcontroller.

VISION SUBSYSTEM

In this module, we focus on detecting the state of the game board and identifying the position of each tile. To accomplish this, we chose the YOLOv8 deep learning model due to its lightweight architecture and ease of training on custom datasets. After printing a few X and O tiles, we captured several images under varying lighting conditions, angles, and zoom levels. I further applied image augmentation techniques, resulting in approximately 500 images. We used Roboflow for annotating the dataset, which included two classes: X and O. The YOLOv8 model was then trained using an RTX 4070 GPU. Given the small dataset size, training 50,000 epochs took less than 30 minutes and still yielded high accuracy. To isolate relevant areas on the board, we used OpenCV to overlay a virtual grid, defining a region of interest (ROI) so only tiles within the grid would be detected. The detection process was efficient, continuously analyzing frames to detect changes. When a change was identified, the system triggered a function to move the robotic arm to the corresponding cell. While our original plan involved using depth information and camera calibration to compute real-world (x, y, z) coordinates, we ended up manually

hardcoding each cell's coordinates. These were then converted into joint angles using inverse kinematics, which is discussed in detail in the corresponding section.

MECHANICAL DESIGN

The mechanical design of the arm focused on two main goals: easy manufacturing and consistent motion. This was ultimately achieved through careful design considerations including motor placement, pulley optimization, hardware selection, and material selection. Figure 6 shows the final CAD assembly with labeled components.

Designing a fully 3D-printed robot arm was key to achieving both cost-effective and accessible manufacturing. With the exception of off-the-shelf hardware, every component of the arm was printed using Bambu Labs PLA, resulting in a total raw material cost of approximately $20. The design was intentionally optimized for additive manufacturing, featuring geometry with minimal overhangs and limited need for support material. Additionally, the 3D printed parts were further enhanced with other hardware. To enhance structural integrity and ease of assembly, threaded heat-set inserts were embedded at most bolt locations. These inserts, installed using a soldering iron, provided durable threaded connections and significantly improved the assembly process. Additionally, all rotational joints were fitted with bearings to minimize friction and ensure smooth, precise motion. This approach allowed us to develop a high-quality robotic arm that is not only affordable and easy to reproduce but also delivers reliable performance with precise control.

To achieve precise arm actuation, careful optimization of motor placement along each link was essential. The combined weight of the motors and gearboxes introduced significant torque at each joint, which was further amplified by the arm's extended reach—necessary to access all positions on the gameboard. By positioning the motors farther from the joints and using a belt and pulley system (Figure 7), we successfully reduced joint torque and moment loads, allowing each motor to operate well within its performance limits and avoid step loss. However, the increased belt length introduced a new challenge: under high tension, the belts stretched slightly, introducing backlash and reducing joint accuracy. To resolve this, the belt lengths were tuned to strike a balance between minimizing joint loads and link accuracy. Ultimately, we settled on a 200mm belt located on link 1, and 280mm belt located on link 2.

Since the robot relies on open-loop stepper motors without encoders, maintaining accurate positional data requires careful step management. To ensure no steps are lost during operation, each joint was equipped with both a physical hard stop and a limit switch. These components enable the system to perform a homing routine, precisely calibrating each joint's angle at startup or after any disturbance. If a motor skips a step, the robot can re-home itself, reestablishing accurate joint positions and ensuring that each link remains precisely where it needs to be.

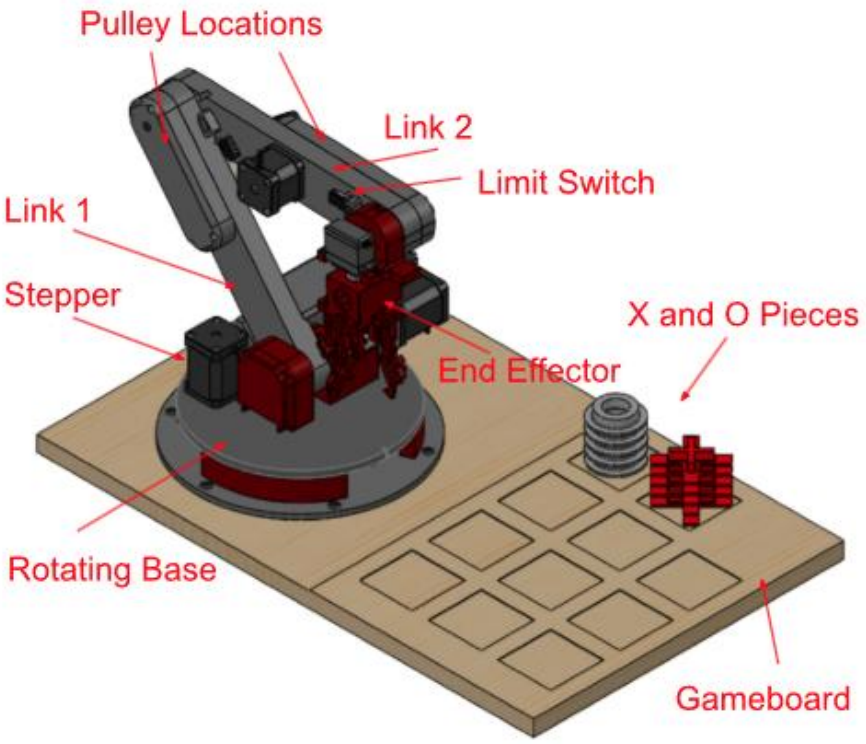

Figure 6: Labeled CAD

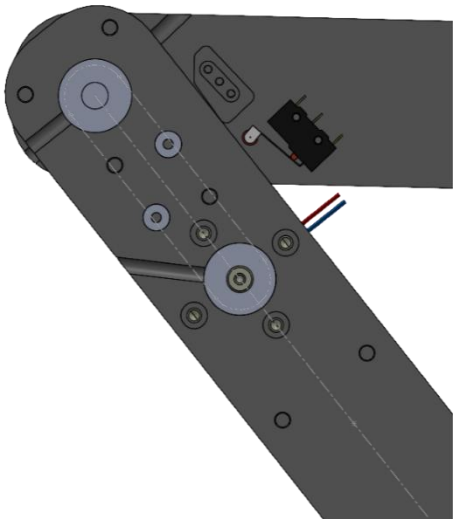

Figure 7: 200 mm Pulley Belt Location

END EFFECTOR SUBSYSTEM

Bio-inspired robotics is a growing field which uses designs and conceptions from nature to create innovative robotic systems. As human hands are some of the most versatile and tactical grippers found in nature, the robot's end effector used it as inspiration to create a versatile gripper that could pick up anything within its maximum opening radius.

The end effector members were modeled after fingers, which consist of bones connected at joints and actuated via muscles. Similarly, the end effector "finger" consists of several members connected via bolts and actuated using a single nylon string, see Figure 8. In this manner, we could control how far the whole finger opened or closed by changing the amount tension in the string. Furthermore, as we needed the fingers to close around the game pieces, rubber bands were added on the interior of the fingers to add counter tension to the string. Thus, the string was used to open the fingers, while the rubber bands were used to close them. This made the finger tension easy to alter as we could add multiple rubber bands when necessary to increase the closing tension if the game pieces became too heavy.

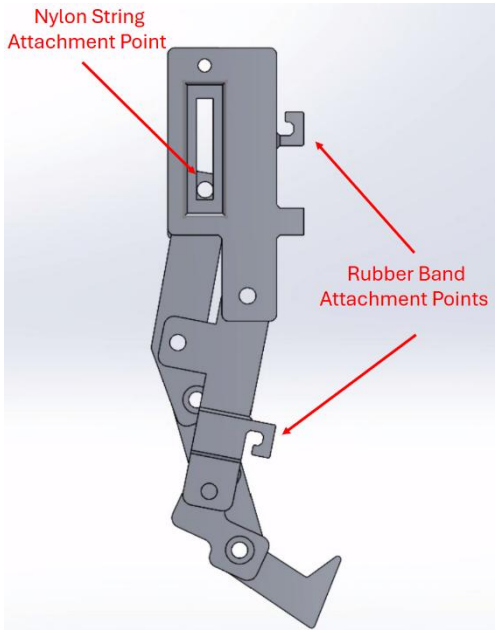

Figure 8: End effector fingers.

The nylon strings from each finger were wrapped around a central spool in the "palm" of the end effector. This spool was connected to a servo motor which when actuated, opened and closed the fingers by increasing or decreasing the tension in the strings. Thus, the fingers were controlled via a single input. The palm of the end effector, see Figure 9, can

easily be redesigned to increase or decrease the number of fingers on it and their location, thus making the end effector very versatile and adaptable to multiple use-cases.

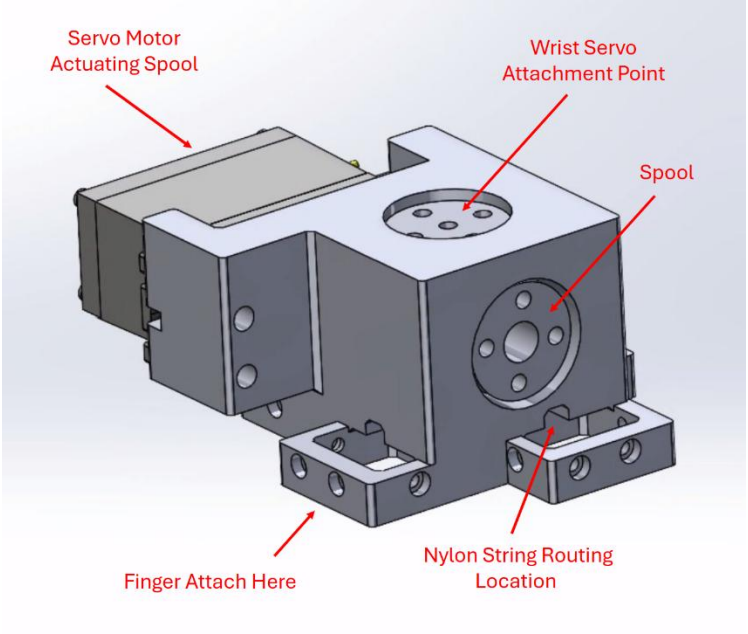

Figure 9: End effector palm.

Finally, the palm was connected to another servo motor mounted on the connection point of the arm and the end effector. This part is analogous to the wrist as it allowed the end effector to rotate and reorientate itself to pick up various shapes such as the “X” piece used in Tic-Tac-Toe. Figure 10 depicts the rotational aspect of the end effector.

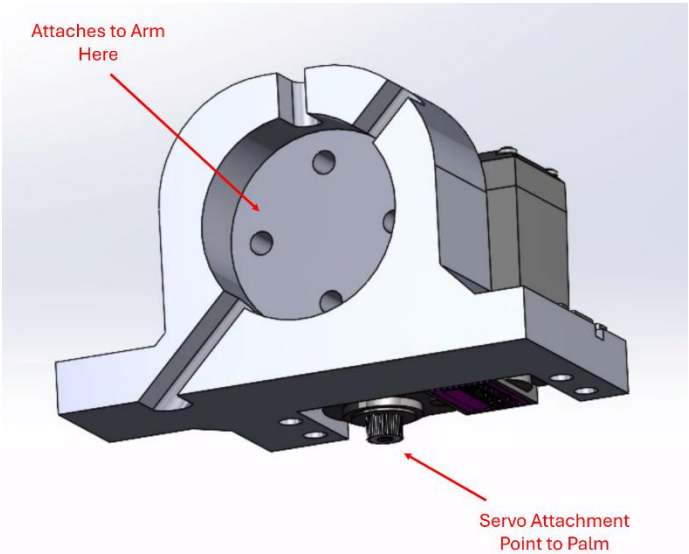

Figure 10: End effector wrist.

This design allowed the robot to effectively and efficiently pick up any of the game pieces with minimal power required, only 5V for both servo motors. Furthermore, the design made it easy to integrate into the final system both in the program and as an add on to the arm.

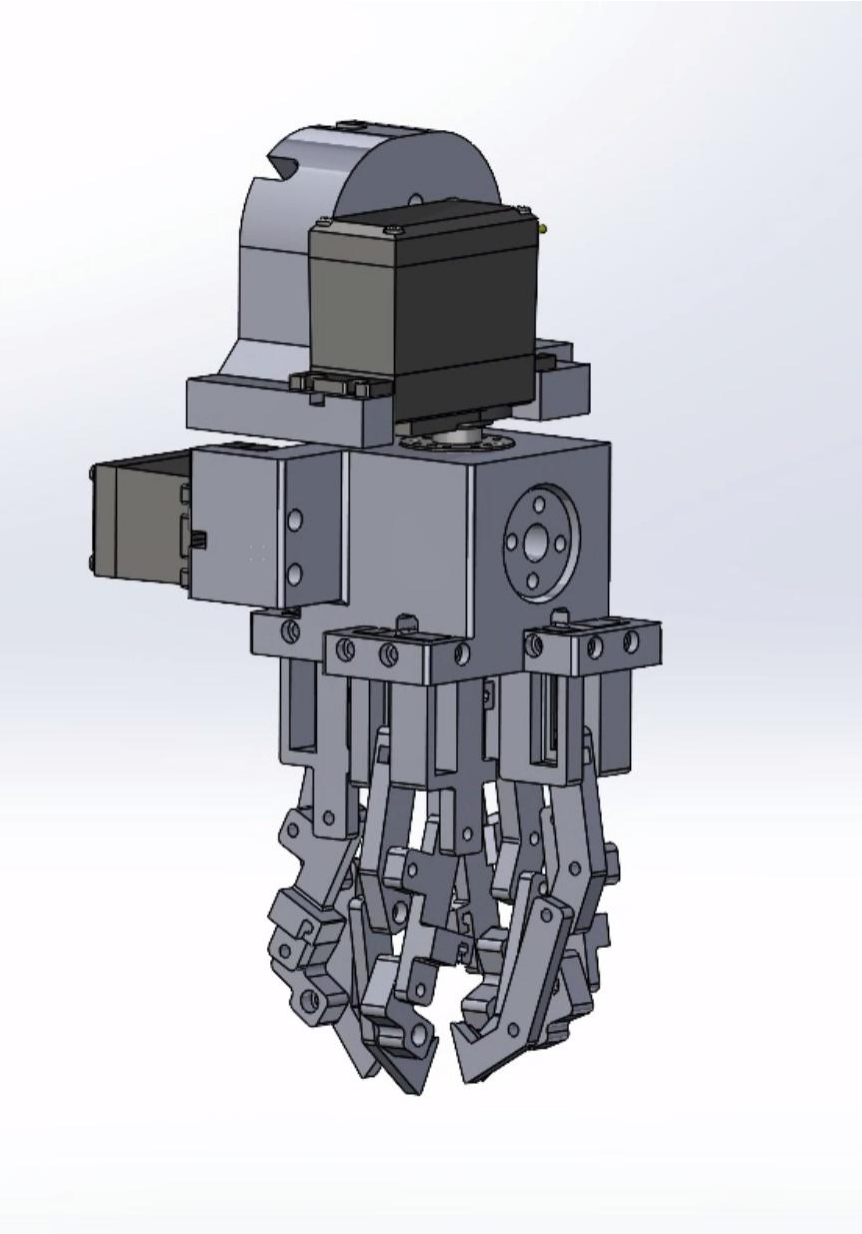

Figure 11: Final end-effector assembly.

DECISION MAKING AND CONTROLS

The decision-making system was straightforward: once a change in the board state was detected, the robot needed to respond by making a move. Using the current camera feed and the previously known board state, we identified all empty cells using OpenCV. After generating an updated list

of available cells, one cell was randomly selected as the target for the robot's move. The 3D coordinates corresponding to the selected cell were then sent to the Inverse Kinematics module to compute the required joint angles. With these angles, the robotic arm followed a predefined motion sequence: it first moved above the X tile stack, opened the gripper, descended to pick up a tile, closed the gripper, and lifted the tile. Next, it navigated to the selected board cell, descended again, and released the tile by opening the gripper. Finally, the arm returned to its default position and awaited the opponent's move. A feedback mechanism was implemented using an Arduino, which sent a confirmation byte after completing each movement, including gripper actions, ensuring synchronized execution and improved reliability.

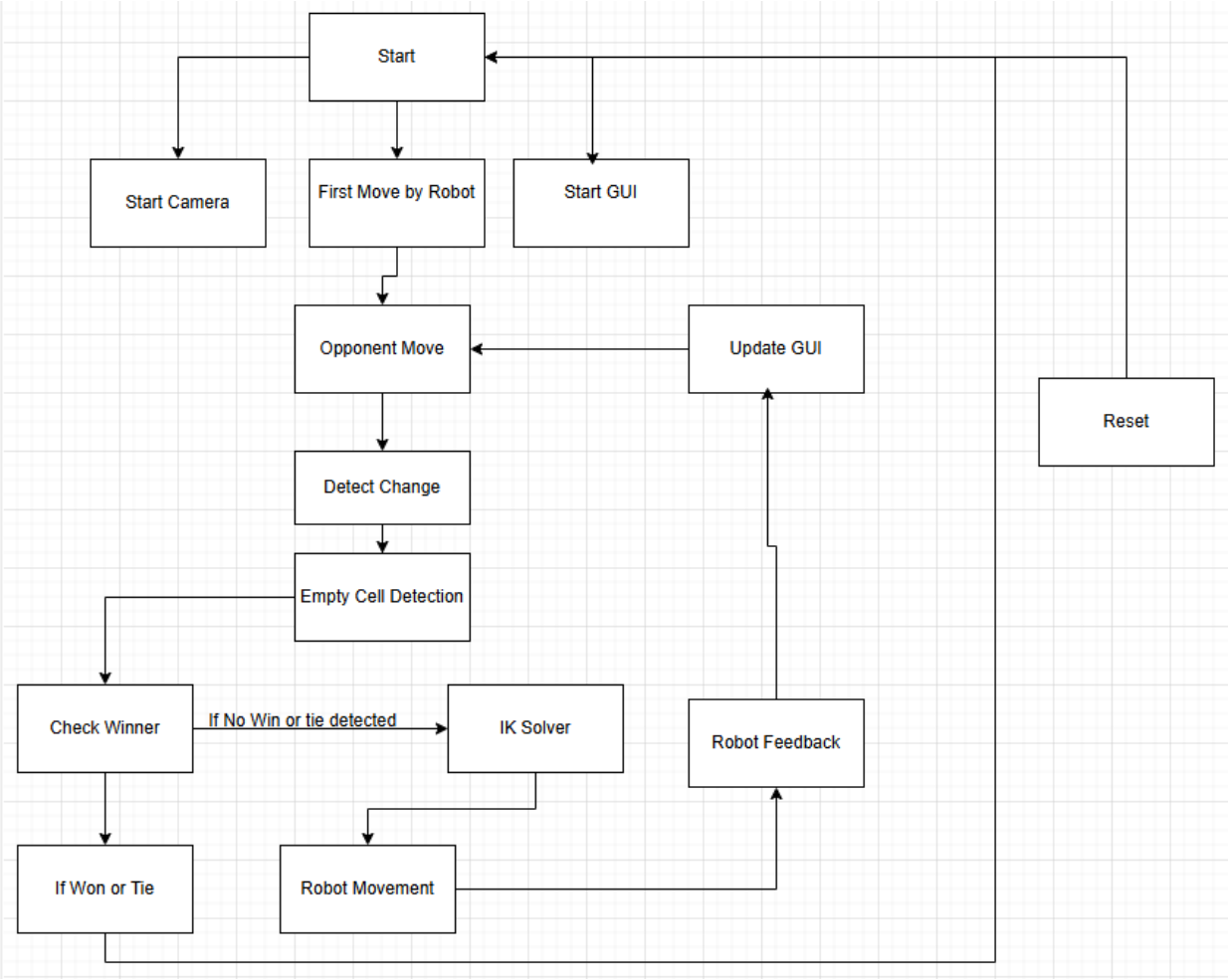

Figure 12: Python code flowchart for game state detection and robot movement initiation.

INTEGRATION AND COMMUNICATION

As discussed in previous sections, our low-level controller runs on an Arduino Mega and is responsible for actuating each joint of the robot according to a desired angle. The controller is designed to take in as an input an array of floating-point values, where each value corresponds to the angle of some joint. Our high-level controller, vision, and decision-making subsystem were developed in a python package running on a PC.

To enable communication between the high-level Python controller and the Arduino, we implemented a custom serial protocol. This protocol allows the Python program to transmit an array of joint angles to the Arduino over a virtual serial (UART) connection established via USB for desired motion. For controlling the end effector, the python program sends the Arduino a single byte, which the Arduino interprets as a command. Additionally, we needed our python program to wait for the completion of any motion before sending angle instructions for subsequent motions. To enable this, the low-level Arduino controller was designed to send a single byte of data as soon as its target angle is reached for every joint, notifying the python program that it is ready to receive a new array of angles. To ensure that serial data would be correctly interpreted by the receiving device, a predefined start byte was used for each of the three aforementioned functionalities.

```python
import serial
import struct
import time

class Serial_Device:
    def __init__(self, port: str, baud: int):
        self.port = port
        self.baud = baud
        # Open serial connection (adjust COM port and baud rate as needed)
        self.ser = serial.Serial(port, baud, timeout=1)
        time.sleep(2)  # Wait for Arduino to reset


    def send_data_f(self, data_array: list[float]):
        start_byte = b'\xFF'
        # Pack the floats into bytes
        data = struct.pack('<' + 'f' * len(data_array), *data_array)  # Little-endian format

        # Send start byte + data
        self.ser.write(start_byte + data)

    def send_data_byte(self, byte):
        start_byte = b'\x7F'

        data = struct.pack('B', byte)
        # Send start byte + data
        self.ser.write(start_byte + data)

    def read_data_byte(self):
        # print("checking serial")
        if self.ser.in_waiting >= 2:
            print("serial available")
            start_byte = self.ser.read(1)
            if start_byte == b'\xAA':  # Start byte detected
                data = self.ser.read(1)
                print("Data: ")
                print(data)
                return data
            else:
                print("Start byte not detected, flushing garbage...")
                self.ser.reset_input_buffer()
        return None  # Return None instead of False
```

Figure 13: Proprietary Python serial class.

CONCLUSION

This Tic-Tac-Toe system developed by Team Raspberry Fly successfully demonstrates a safe and engaging example of human-robot interaction (HRI) in a low-stakes, secure environment. By integrating computer vision, robotic arm actuation, and control logic, the system not only showcases the technical feasibility of interactive robotic systems but also emphasizes the importance of safety measures when dealing with robotic systems. The robot's ability to detect human input, respond strategically, and physically interact with the environment—all while maintaining a safe operational space—exemplifies the core principles of HRI. This project serves as a step towards more complex, collaborative applications where robots and humans can work together intuitively and securely. As robotic systems continue to enter household, educational, and commercial spaces, projects similar to this will play a vital role in shaping public perception and guiding the design of future human-centric robotic technologies.

LESSONS LEARNED

Throughout the development of our Tic-Tac-Toe robotic system, our team encountered several challenges that highlighted critical lessons in both hardware integration and system-level design. One of the most significant takeaways was the importance of separating high-current components from low-power control systems. In our case, plugging our custom PCB shield directly onto the Arduino resulted in excessive current draw from the stepper motor drivers, which ultimately damaged the Arduino's onboard voltage regulator. This failure emphasized the need for dedicated power supplies or isolation techniques when handling components with substantially different current requirements.

Additionally, the project reinforced the importance of robust planning and modular testing. Early iterations of our computer vision system required substantial dataset augmentation and model training, while the inverse kinematics module benefited from precise calibration and spatial reasoning. The integration of subsystems underlined the value of synchronized communication protocols and real-time feedback loops. Overall, the experience deepened our understanding of safe human-robot interaction, emphasized careful power management in embedded systems, and provided practical insight into the complexities of mechatronic system design.

BILL OF MATERIALS

Part	Description	Qty	Cost Per Unit	Total Cost
Stepper Motor Drivers	6 pack	1	$30	$30.00
RealSense Camera		1	Provided	
Pulleys/Belts	For arm	1	$11.59	$11.59
608 2RS Bearings	For arm	1	$6.99	$6.99
6202-2RS Bearings	For arm	1	$5.59	$5.59
693ZZ Deep Groove Ball Bearing	For arm	1	$9.99	$9.99
Nema 17 Bipolar 45Ncm	For arm	2	$9.13	$18.26
Nema 17 Bipolar 65Ncm	For arm	2	$13.24	$26.48
Nema 17 Bipolar 59Ncm	For arm	1	$13.99	$13.99
5:1 Gearbox	For arm	1	$24.08	$24.08
10:1 Gearbox	For arm	1	$24.08	$24.08
20:1 Gearbox	For arm	1	$45.00	$45.00
TMC2209 Stepper Motor Driver	For arm	1	$29.99	$29.99
Threaded Inserts	For arm	1	$9.99	$9.99
Limit Switch	For arm	1	$5.99	$5.99
Bolts/Nuts	For arm	1	$15.49	$15.49
Nylon Lock Nuts	For arm	1	$7.98	$7.98
8mm bore pulleys	For arm	1	$11.98	$11.98
5mm Flange Coupling	For arm	1	$7.99	$7.99
8mm Flange Coupling	For arm	1	$9.99	$9.99
Matte Dark Red PLA	For arm	1	$19.99	$19.99
Matte Nardo Gray PLA	For arm	1	$19.99	$19.99
uxcell 50mm 6810ZZ Bearings	For arm (base)	1	$9.89	$9.89
M2 nylock		1	$10	$10.00
Small Rubber Bands		1	$5.29	$5.29
String		1	$7	$7.00
Arduino Mega			Provided	
2x Servo Motors			Provided	
Bambu Lab PLA Matte	For all 3D printing	2	$20	$40.00

TOTAL				$387.61

APPENDIX:

CODE

Repository for high-level Python serial controller, computer vision, and game logic:
https://github.com/Tr0612/Game_runner/tree/new_version

Repository for low-level Arduino controller and serial communication:
https://github.com/jnealo32/TicTacToeBOT-AruinoController

Repository for custom stepper library:
https://github.com/jnealo32/TMC2209LEGACY

ENGINEERING DRAWINGS

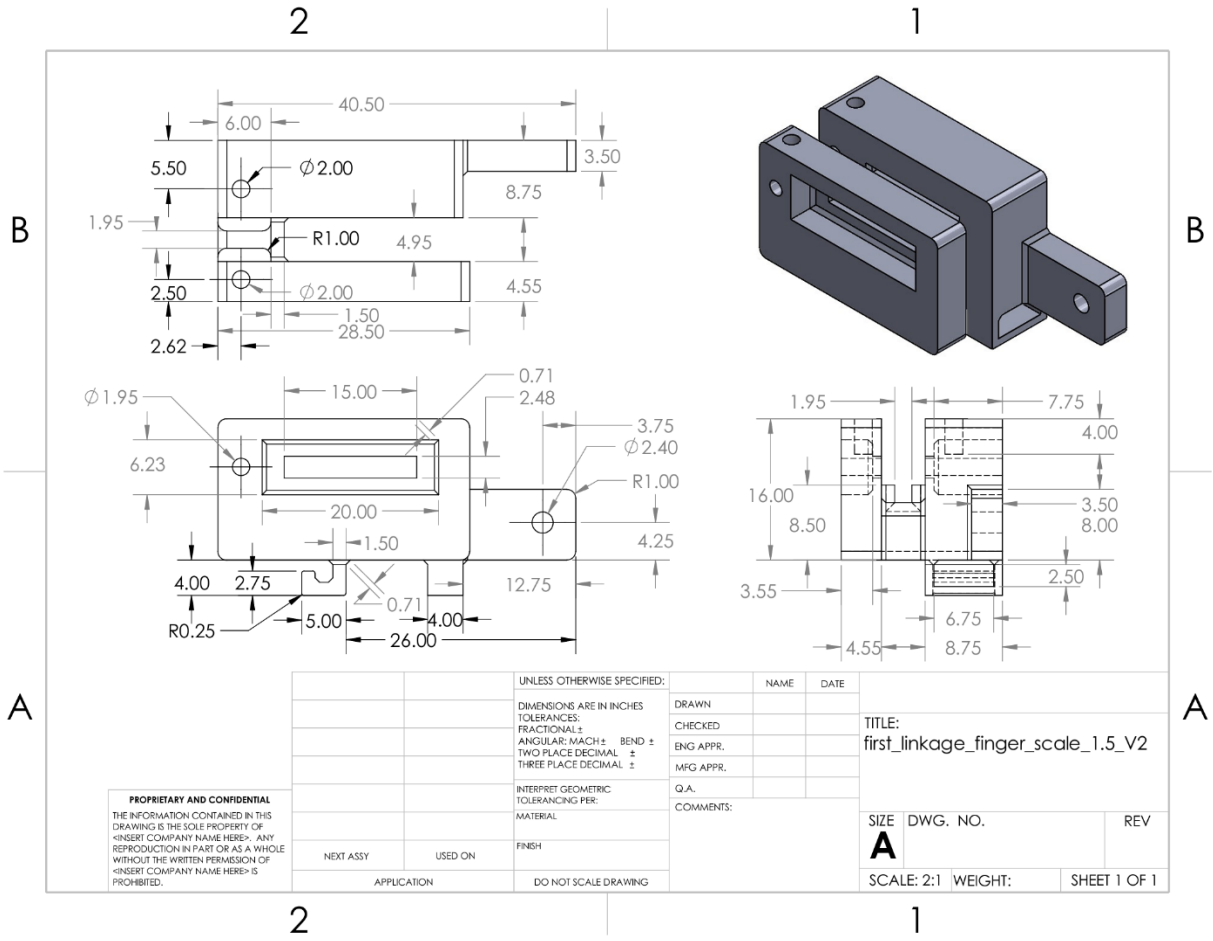

Figure 14: End effector fingers: first linkage engineering drawing.

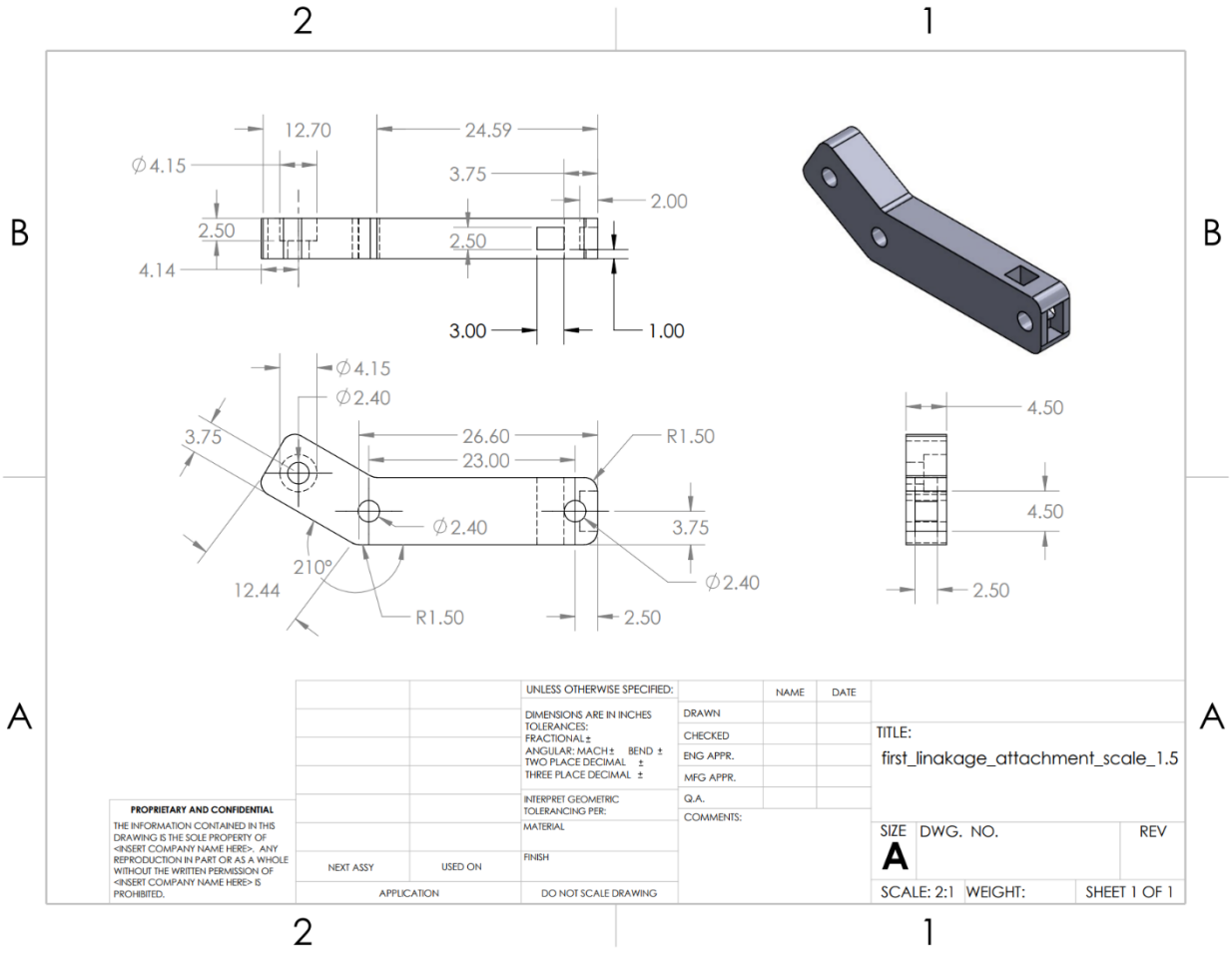

Figure 15: End effector fingers: second linkage engineering drawing.

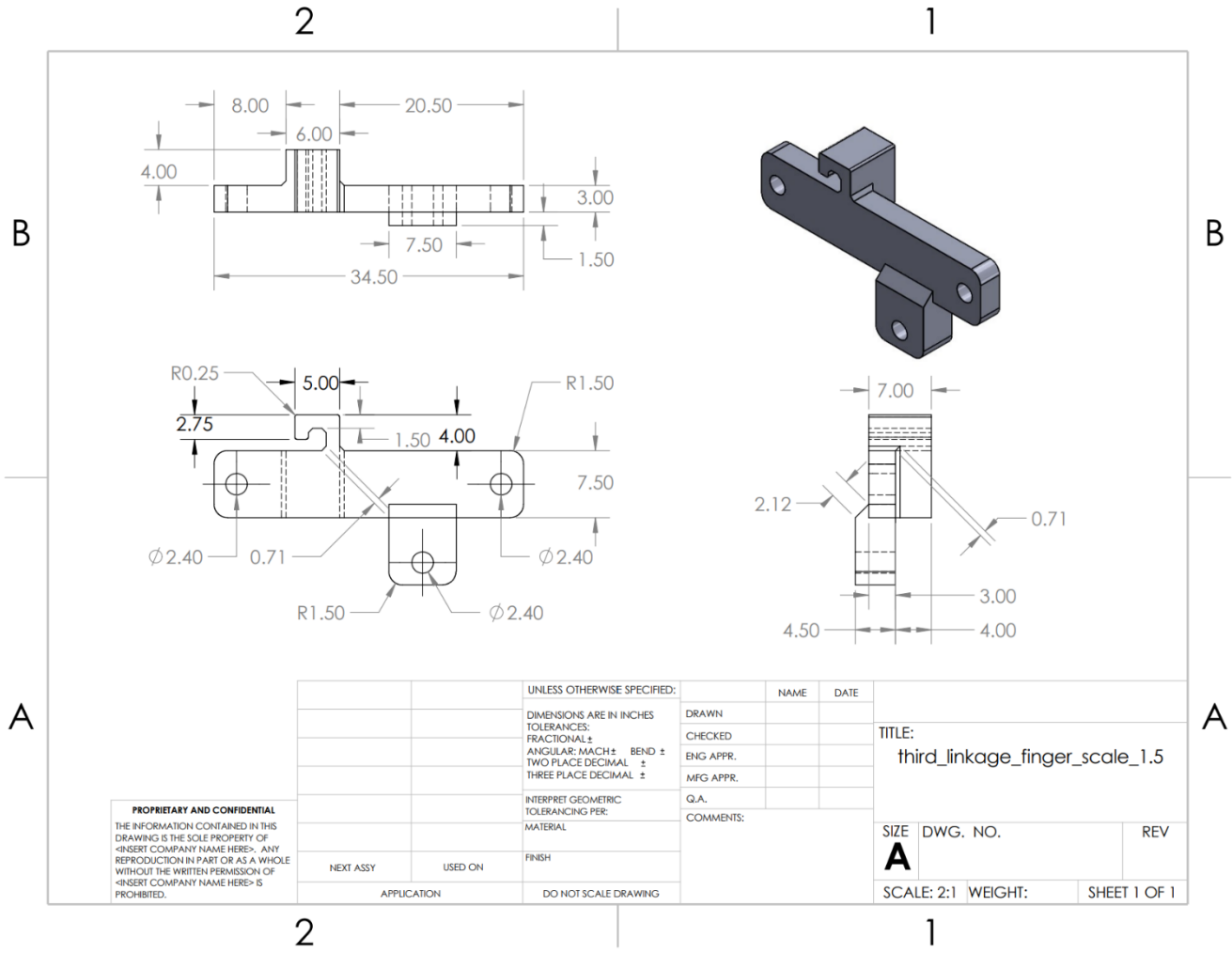

Figure 16: End effector fingers: third linkage engineering drawing.

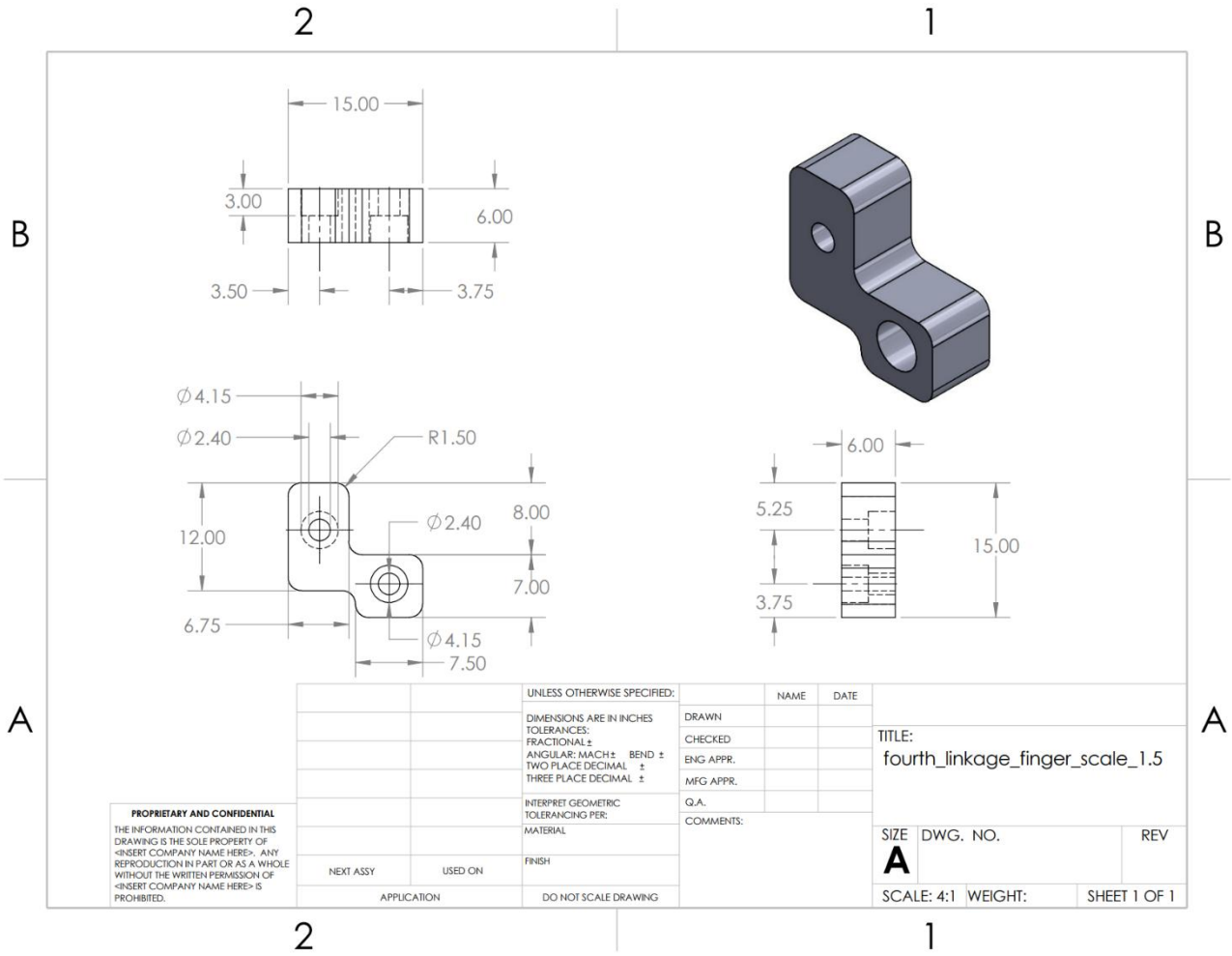

Figure 17: End effector fingers: fourth linkage engineering drawing.

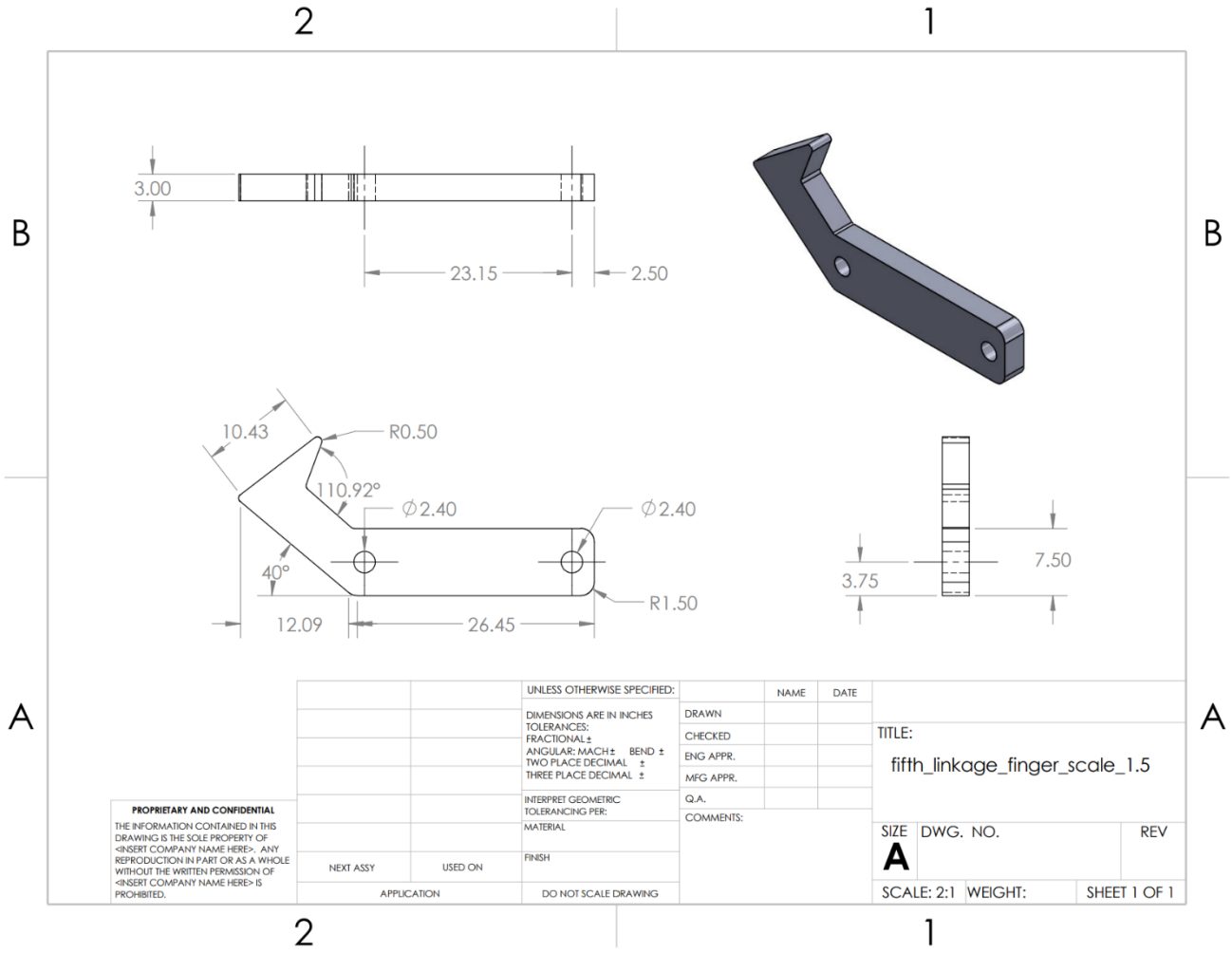

Figure 18: End effector fingers: fifth linkage engineering drawing.

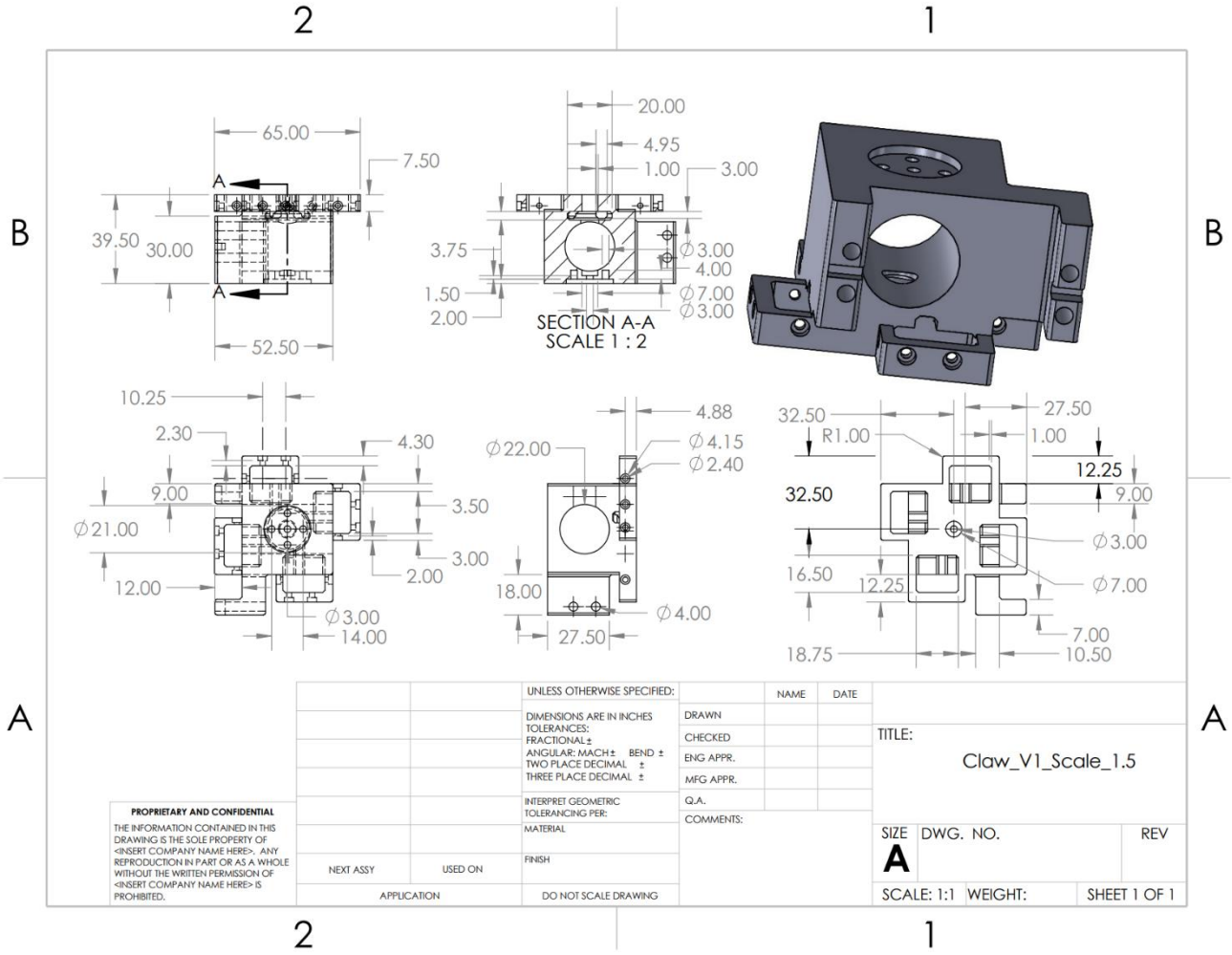

Figure 19: End effector palm engineering drawing.

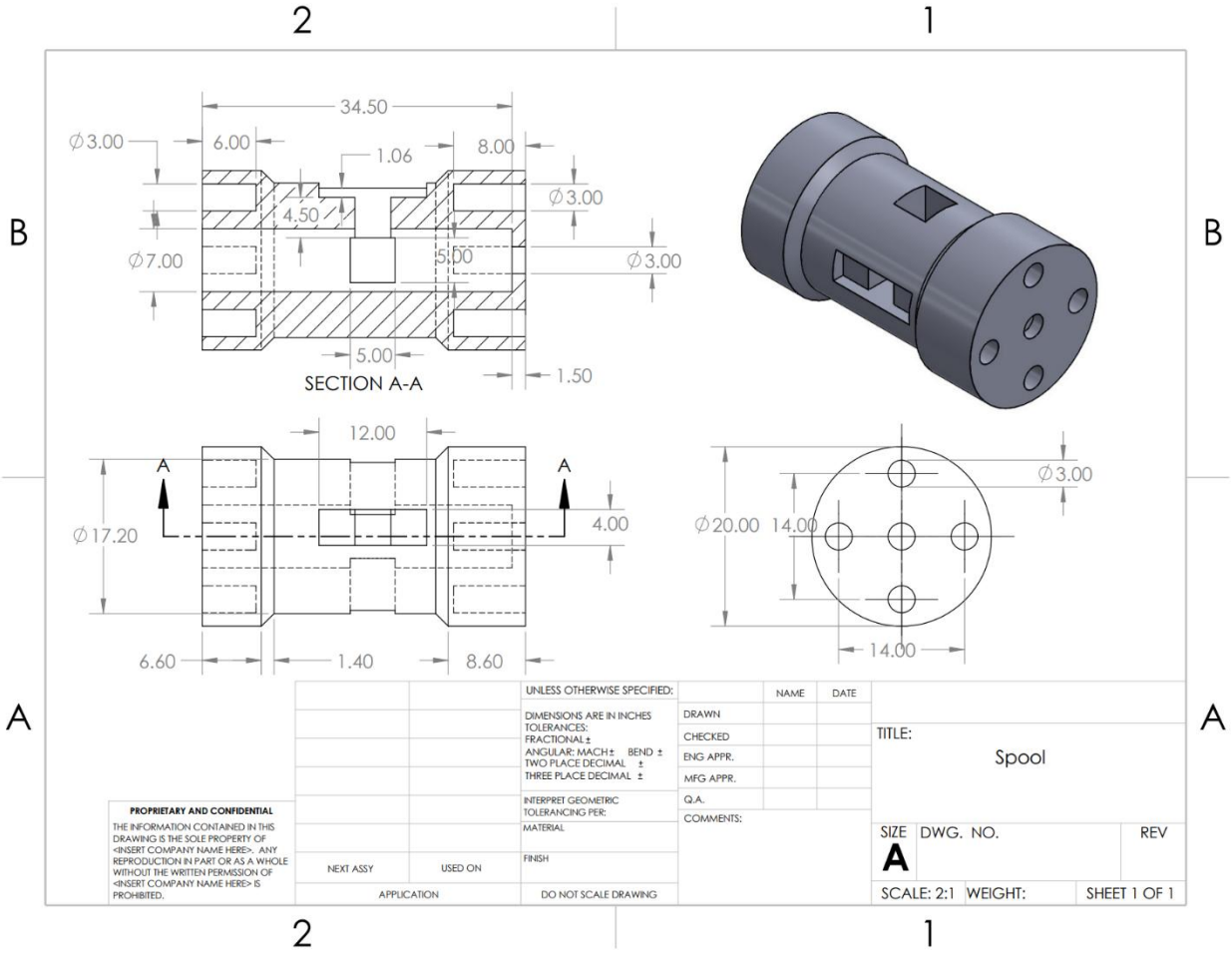

Figure 20: End effector spool engineering drawing.

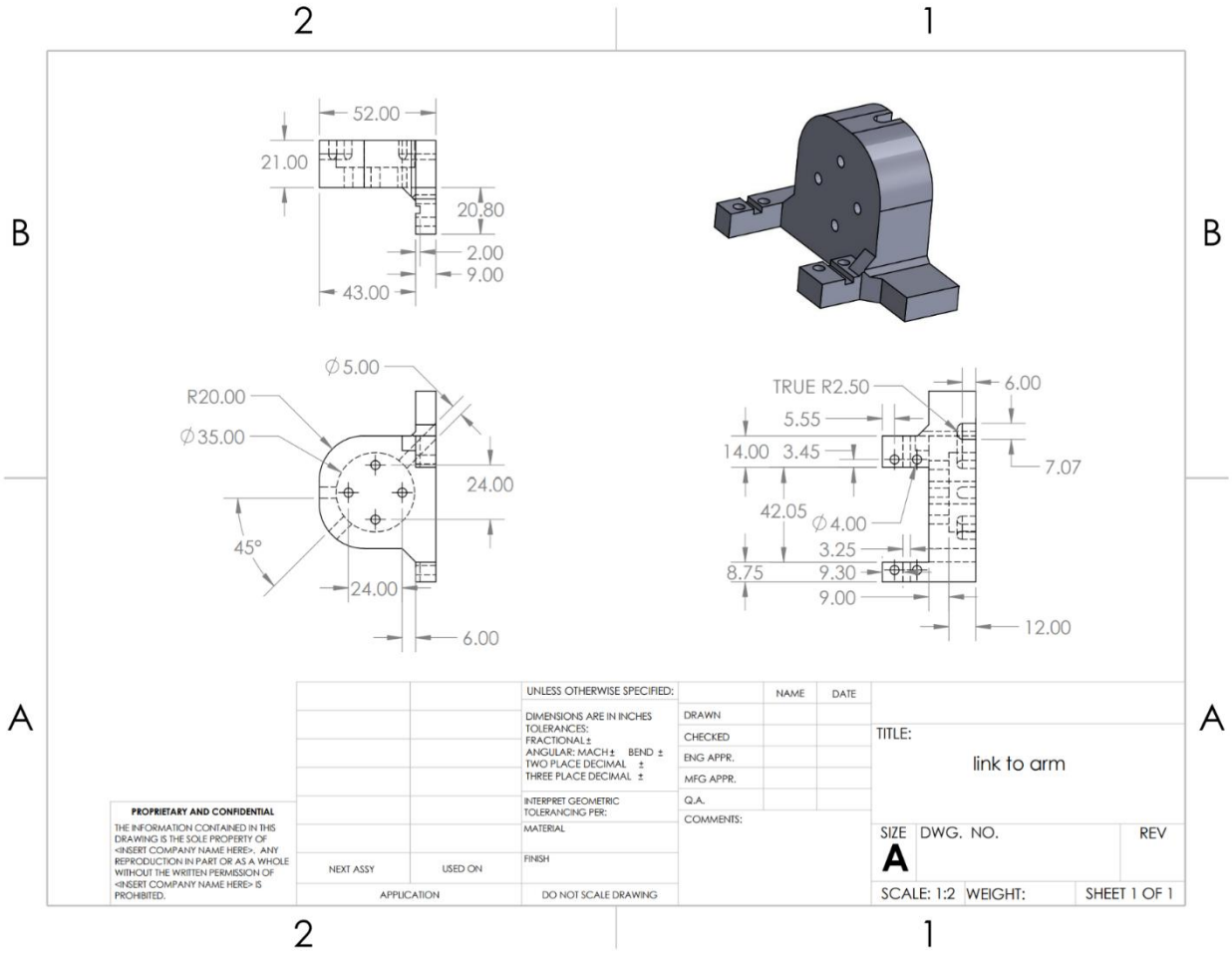

Figure 21: End effector wrist engineering drawing.

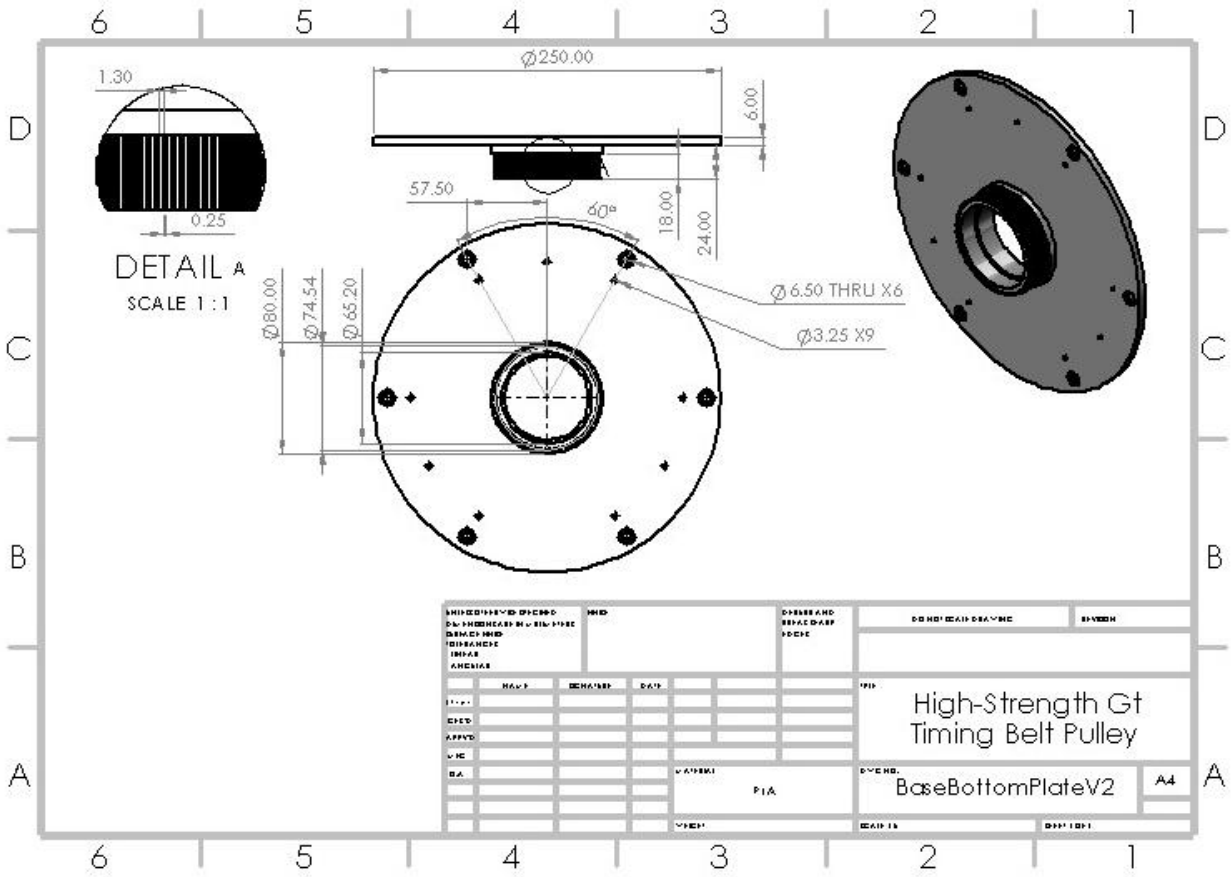

Figure 22: Base Bottom Plate with Belt Pulley engineering drawing

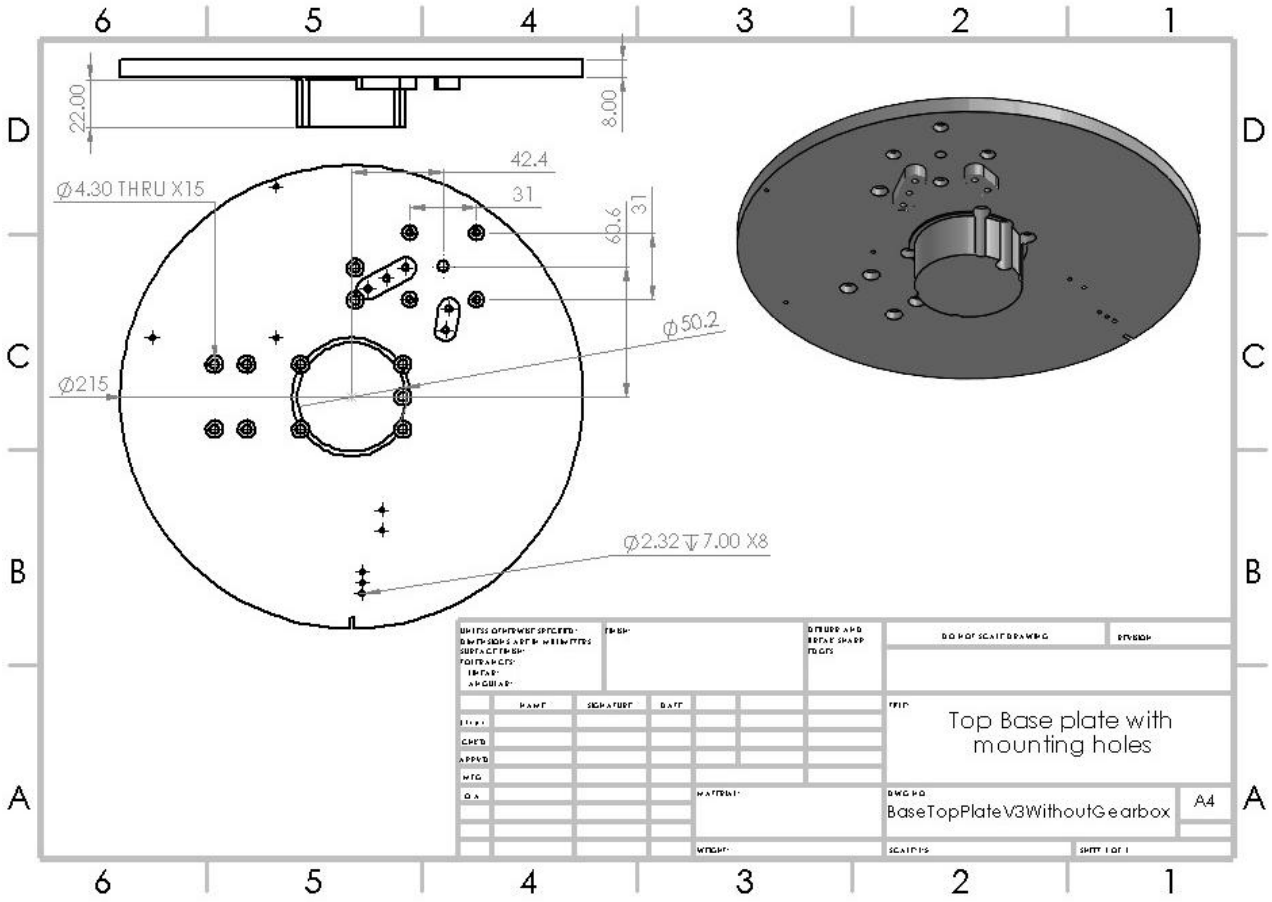

Figure 23: Top Base Plate with press-fit bearing attachment engineering drawing

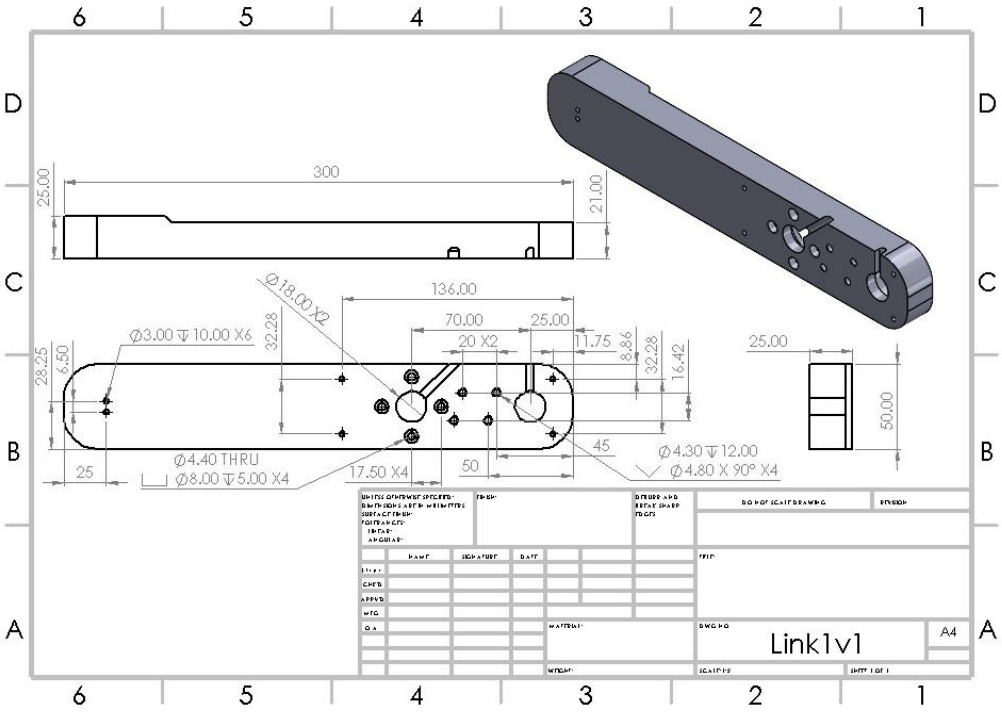

Figure 24: Link 1 Engineering Drawing

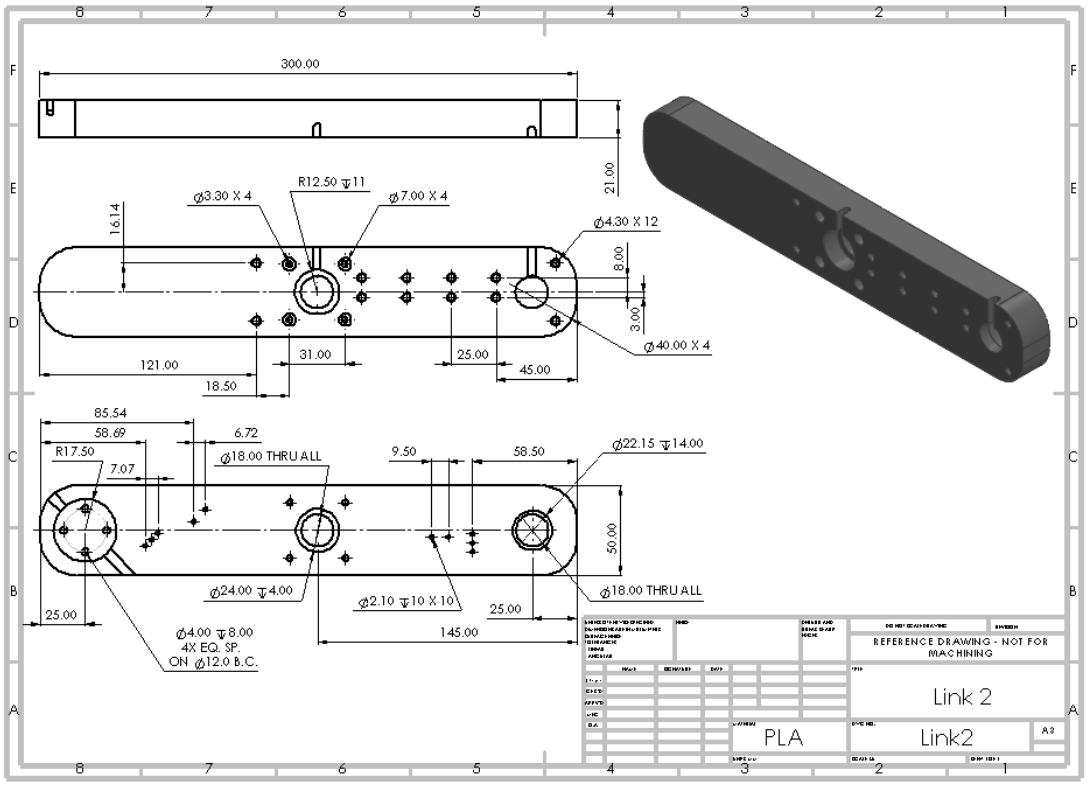

Figure 25: Link 2 Engineering Drawing

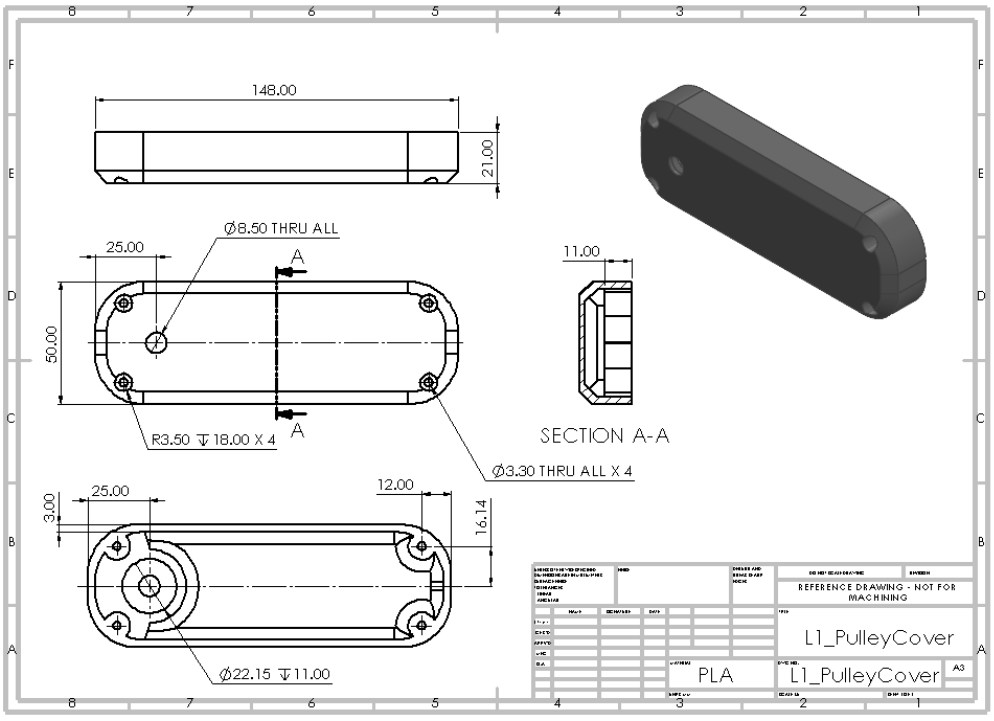

Figure 26: Link 1 Pulley Cover Engineering Drawing

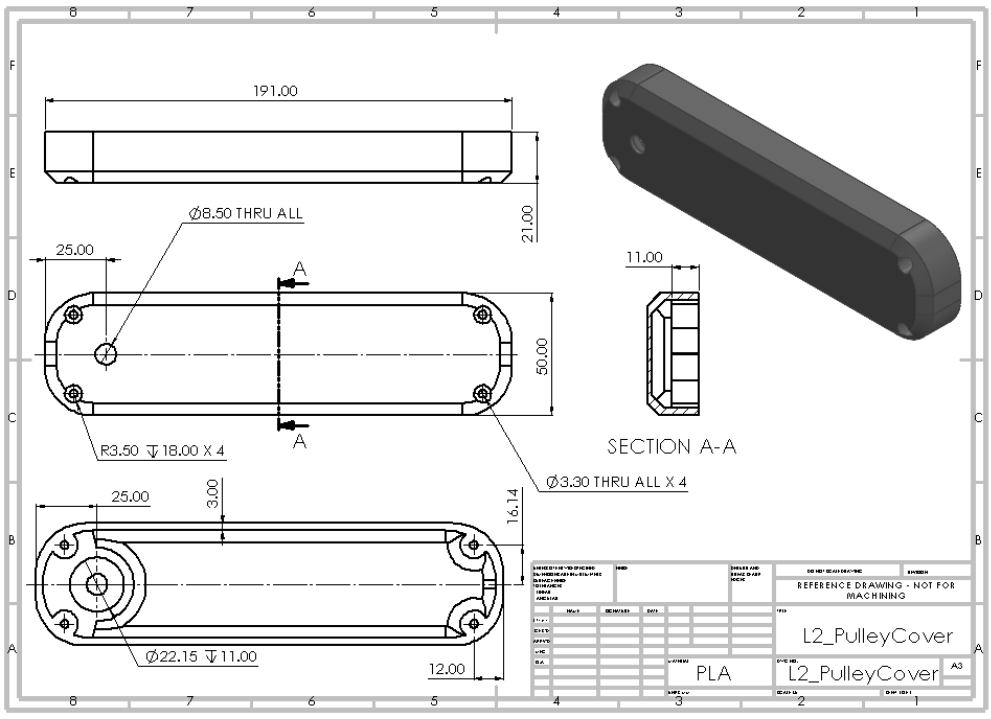

Figure 27: Link 2 Pulley Cover Engineering Cover

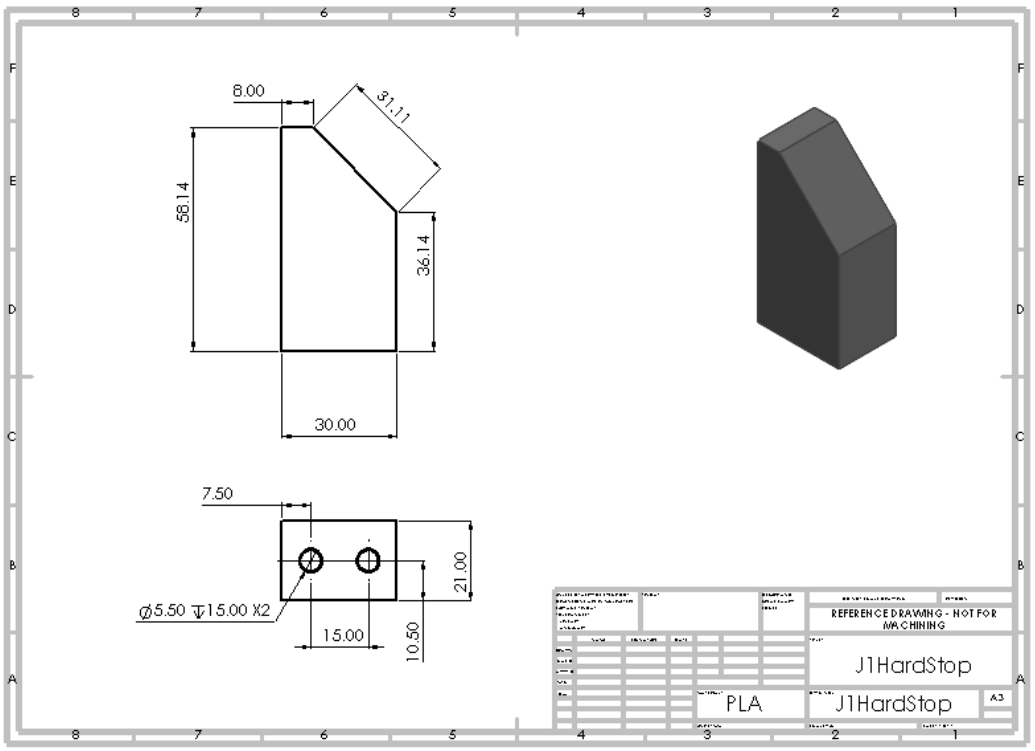

Figure 28: Joint 1 Hard Stop Engineering Drawing

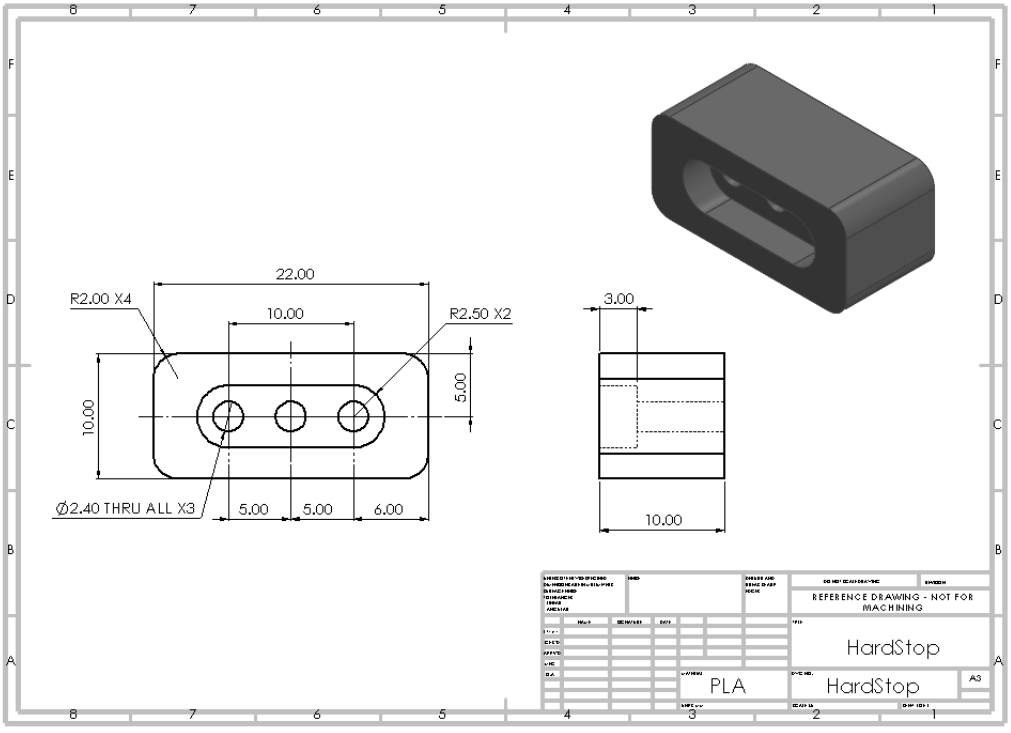

Figure 29: Base, Joint 2, End Effector Hard Stop

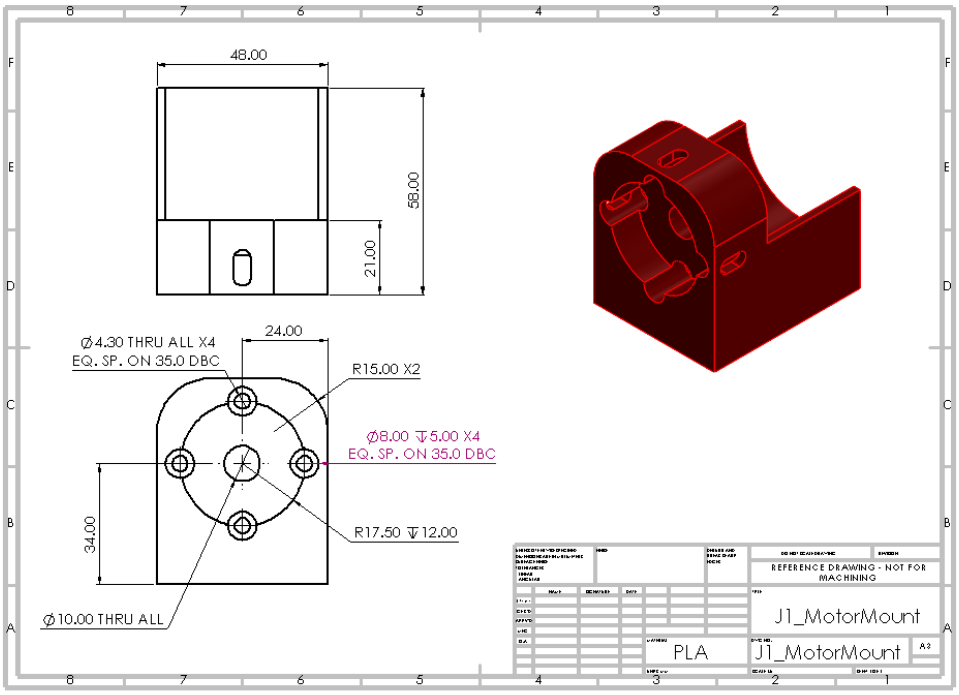

Figure 30: Joint 1 Motor Mount Engineering Drawing

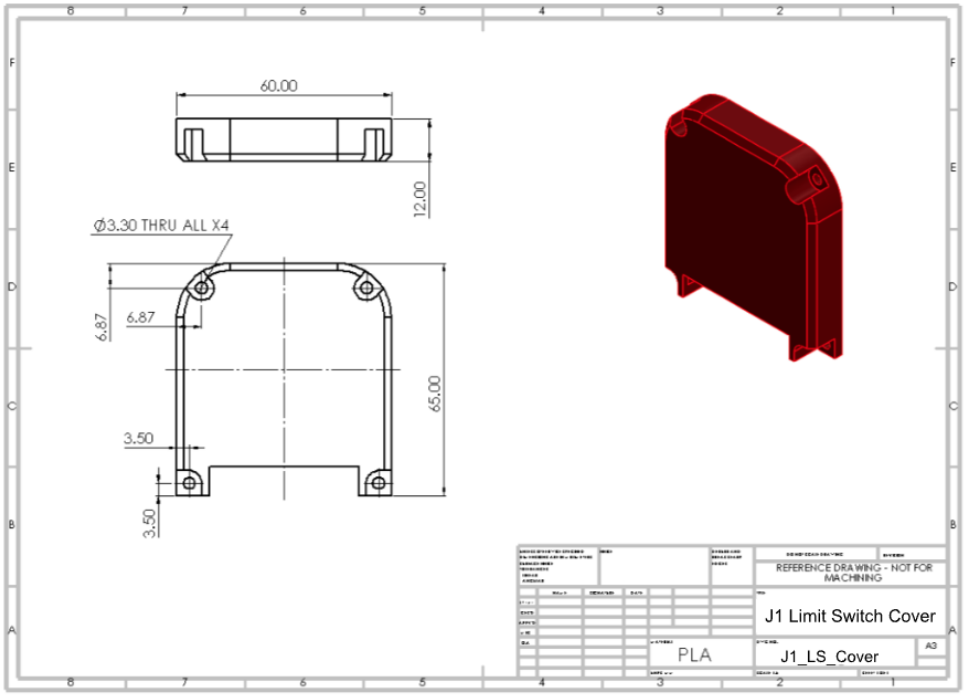

Figure 31: Joint 1 Limit Switch Cover

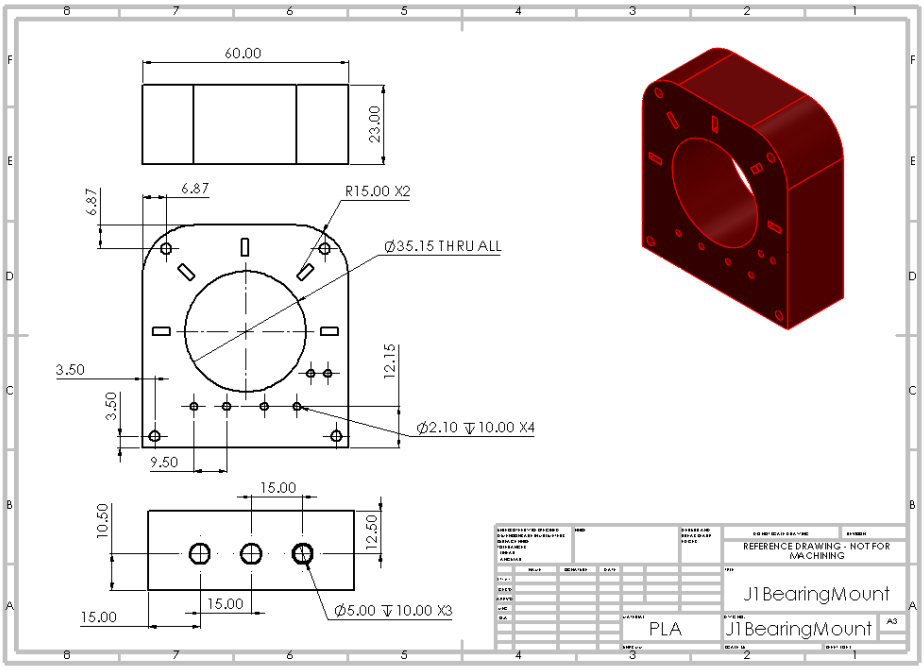

Figure 32: Joint 1 Bearing Mount

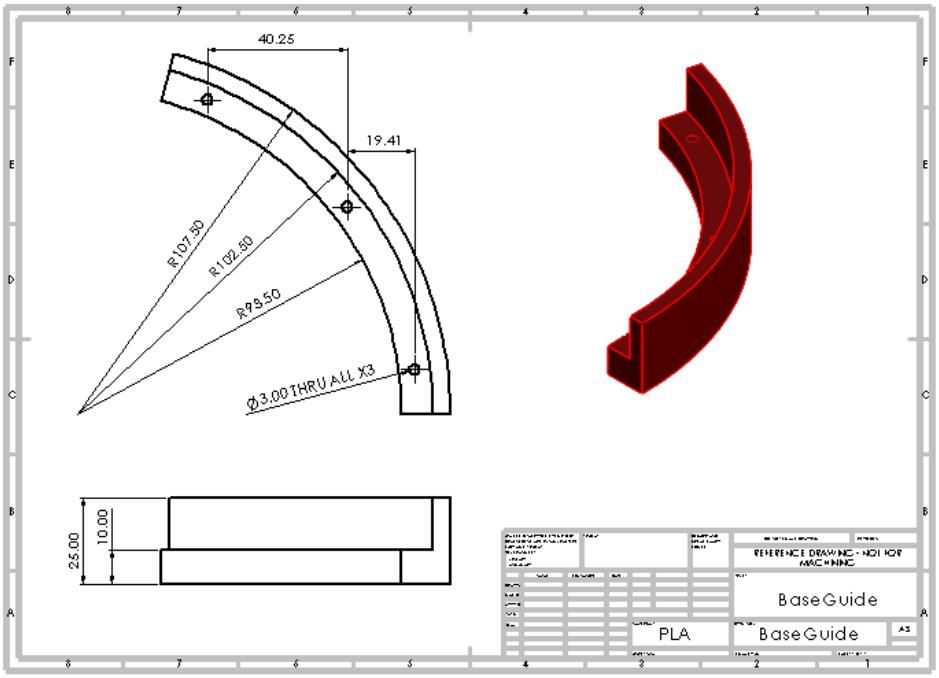

Figure 33: Base Guide Engineering Drawing

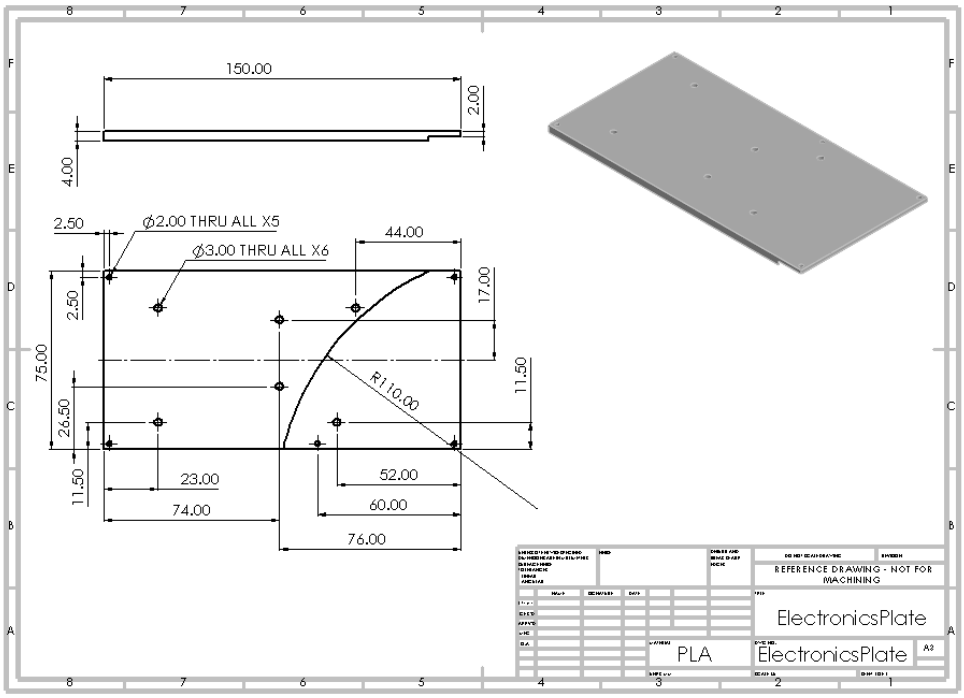

Figure 34: Electronics Plate Engineering Drawing

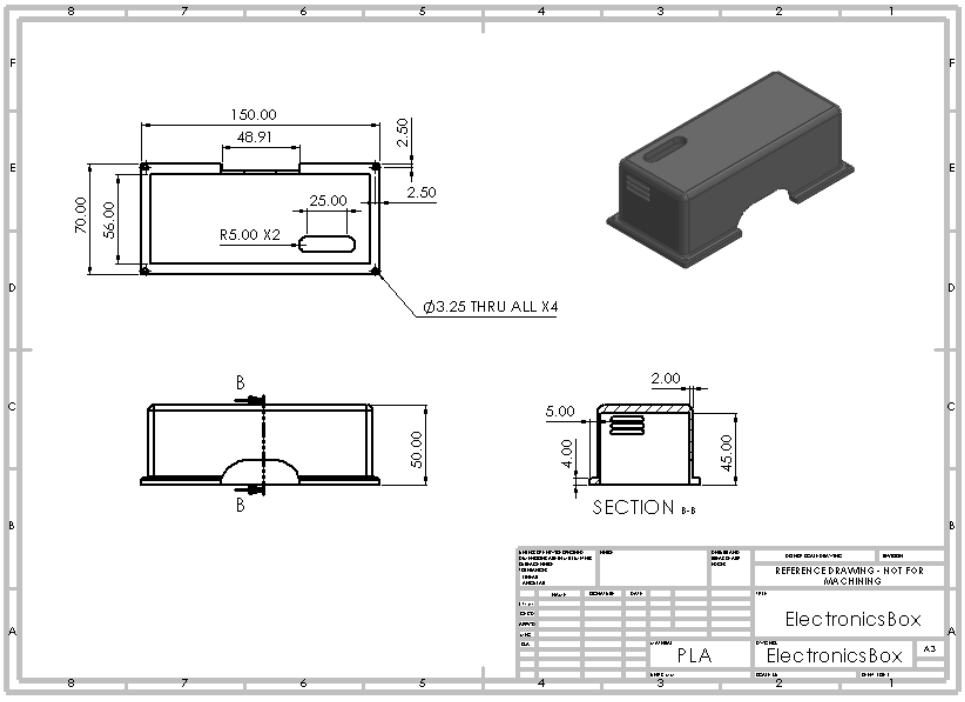

Figure 35: Electronics Box Engineering Drawing

REFERENCES:

https://github.com/PCrnjak/PAROL6-Desktop-robot-arm