



ROBOT DESIGN AND MECHATRONICS REPORT

Harish Raj Venkatnarayanan

Abstract

Group Members: Harish Raj Venkatnarayanan 231176846 James Brownson 230643699
Victor Osuofia 230870228

Group 15 Robot P

Table of Contents

| | |
|---|-----------|
| 1.0 Abstract | 2 |
| 2.0 Introduction | 2 |
| 3.0 Materials and Methods | 3 |
| 3.1 Forward and Inverse Kinematics..... | 3 |
| 3.2 DH Kinematics | 6 |
| 3.3 Design Sketches and Description..... | 8 |
| 3.4 CAD Design and description | 10 |
| 3.5 Manufacturing Processes | 12 |
| 3.6 Electromechanical System..... | 13 |
| 3.7 Control System Integration | 14 |
| 4.0 Results..... | 17 |
| 5.0 Discussion and Conclusion | 18 |
| 6.0 References | 19 |
| 7.0 Appendix | 19 |

1.0 Abstract

This report details the design, analysis, and implementation of Robotic Arm P, developed as part of a group engineering project. The team consisted of three members: two robotics engineers (myself and Victor) and one design engineer (James). Team communication was maintained through WhatsApp, Microsoft Outlook, and regular independent and supervised lab sessions.

The workload was split relatively evenly across the group, with James and I contributing slightly more than Victor overall. Due to the mechanical complexity of the robot, James and I collaborated on the CAD design, with James taking the lead in modelling. I focused primarily on the kinematics analysis, developed the electronic system, and programmed the robot's control logic. Victor supported the mechanical build and worked as an assisting hand in both hardware and build-related tasks.

I also led the preparation and delivery of the final presentation. The outcome was a functional robotic arm capable of responding to keyboard commands and executing accurate planar motion using inverse kinematics.

2.0 Introduction

The aim of this project was to design and build a 2 DoF (Degrees of Freedom) robotic arm actuated by a revolute and a prismatic joint. The system was driven by 2 DC motors with encoders enabling the arm to operate in the vertical X-Z plane. A PID (Proportional-Integral-Derivative) control system was implemented to minimise the positional error of the robot's end effector. The robot was designed to follow user-defined target coordinates, using inverse kinematics to convert these inputs into motor commands. The control system continuously adjusts the motor outputs to bring the arm closer to its desired configuration. This process enabled the autonomous navigation of the robotic arm to target positions with high precision.

However, due to challenges with achieving consistent and reliable behaviour from the inverse kinematics implementation, an alternative manual control system was also incorporated. This used keyboard-based WASD inputs to adjust the position incrementally, with PID control still active to ensure smooth motion and stabilisation.

3.0 Materials and Methods

3.1 Forward and Inverse Kinematics

Forward Kinematics are a set of equations that calculate the position (and optionally velocity or acceleration) of the robot's end effector in the X-Y-Z coordinate space, given the known values for angle θ (for revolute joint) and extension Δd (for prismatic joint). Inverse kinematics, on the other hand, are a set of equations to calculate the required joint parameters- angle θ (of any revolute joint) and extension Δd (for any prismatic joints) given the position of the robot's end effector. This process is essential for ensuring that the robot reaches specific target coordinates.

Robot P was based on a 4-bar linkage system, actuated by one revolute and one prismatic joint. The forward kinematics were first derived by analysing the geometry of the linkage system as can be seen below:

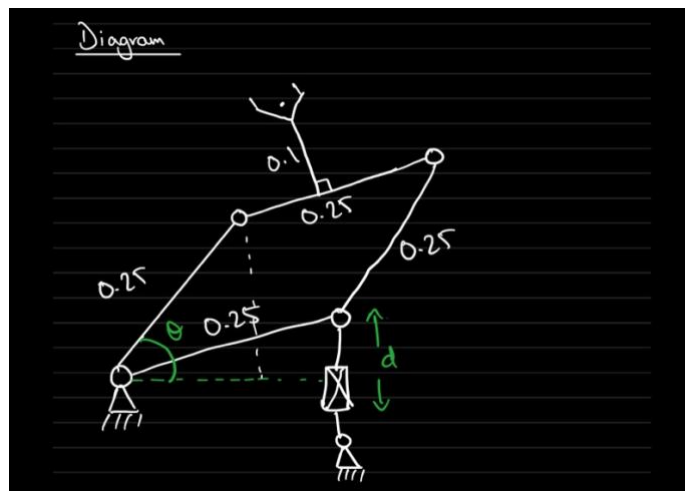


Figure 1: Diagram of robot P

Forward Kinematics

$$\alpha = \sin^{-1}\left(\frac{d}{0.25}\right)$$

< inside trapezium $\beta = \theta - \alpha$

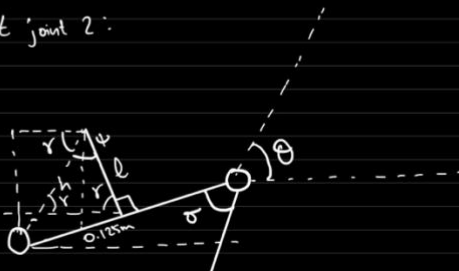


At joint 1:

$$x = 0.25 \cos \theta$$

$$z = 0.25 \sin \theta$$

At joint 2:



$$\psi = \tan^{-1}\left(\frac{0.125}{l}\right)$$

$$\sigma = \frac{360 - 2(\theta - \alpha)}{2}$$

$$\sigma = \frac{360 - 2\theta + 2\alpha}{2}$$

$$= 180 - \theta + \alpha$$

$$\sigma = 180 - \theta + \sin^{-1}\left(\frac{d}{0.25}\right)$$

$$h = \sqrt{l^2 + 0.125^2}$$

$$x = h \cos \gamma$$

$$z = h \sin \gamma$$

$$\therefore x = 0.25 \cos \theta + h \cos \gamma$$

$$z = 0.25 \sin \theta + h \sin \gamma + \Delta d$$

$$2\gamma + \psi = 180^\circ$$

$$\gamma = \frac{180 - \tan^{-1}\left(\frac{0.125}{l}\right)}{2}$$

Figures 2-3: Kinematics derivation

These equations were then algebraically rearranged and manipulated to obtain the inverse kinematics equations, using inverse trigonometry and simultaneous equations, as shown below:

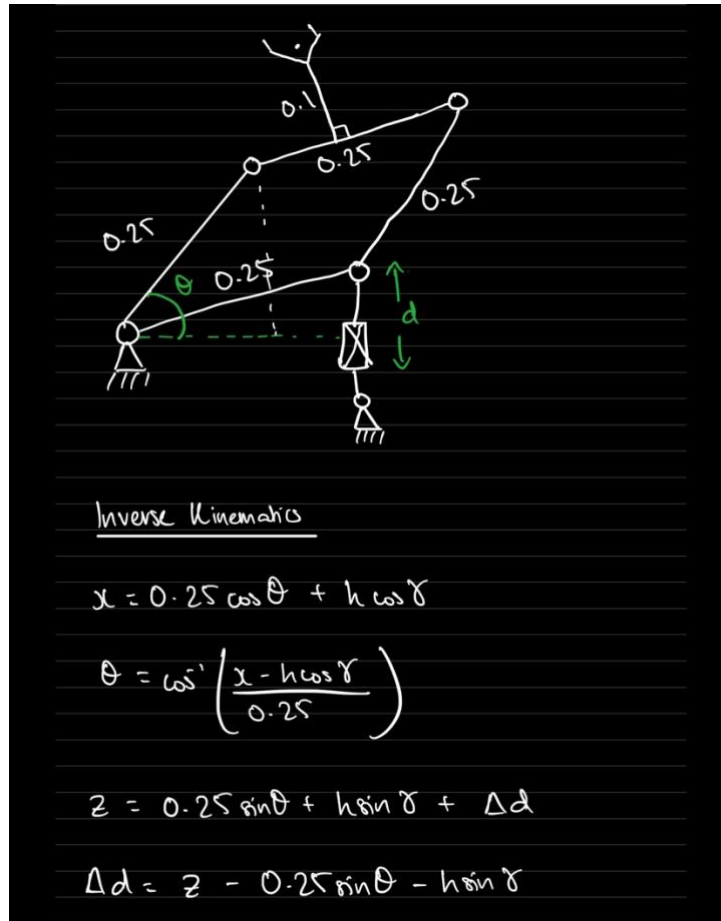


Figure 4: Inverse Kinematics Equations

This analysis resulted in 2 forward kinematics equations and 2 corresponding inverse kinematics equations.

Forward kinematics equations:

$$x = 0.25 \cos(\theta) + h \cos(\gamma)$$

$$z = 0.25 \sin(\theta) + h \sin(\gamma) + \Delta d$$

Where:

$$h = \sqrt{0.1^2 + 0.25^2}$$

$$\gamma = \frac{180 - \tan^{-1}(\frac{0.125}{0.1})}{2}$$

Inverse kinematics equations:

$$\theta = \cos^{-1} \left(\frac{x - h \cos(\gamma)}{0.25} \right)$$

$$\Delta d = z - 0.25 \sin(\theta) - h \sin(\gamma)$$

Where once again:

$$h = \sqrt{0.1^2 + 0.25^2}$$
$$\gamma = \frac{180 - \tan^{-1}\left(\frac{0.125}{0.1}\right)}{2}$$

These equations were implemented in Arduino IDE (through use of C++) to be used in conjunction with encoder values from both motors to form the feedback-based control system.

```
// Kinematic Parameters
const double L1 = 0.25; // Link length (m)
const double L2 = 0.125; // Offset (m)
const double L3 = 0.1; // Base link (m)
const double h = sqrt(L3 * L3 + L2 * L2);
const double gamma = (180.0 - atan(L2 / L3)) / 2.0;
```

Figure 5: Implementation of kinematic parameters in C++

3.2 DH Kinematics

To systematically model the movement of the robot's end effector, Denavit-Hartenberg (DH) kinematics were used. The DH convention is a standardised method of assigning coordinate frames to each link in a robotic system. This reduces the complexity of deriving transformation matrices by defining each joint through 4 parameters:

- θ : joint angle (rotation about the y-axis- angle between x_{i+1} and x_i)
- α : link twist (angle between y_{i+1} and y_i)
- a : link length (distance between y_{i+1} and y_i along x_i)
- d : link offset (distance between x_{i+1} and x_i along y_i)

Robot P has 2 DoF: a revolute joint followed by a prismatic joint. However, unlike conventional serial robotic chains, robot P has 2 connections to the ground (forming a 4-bar linkage), instead of 1, forming a closed kinematic loop.

Standard DH convention is most easily applied to open-chain manipulators, so applying it to Robot P required additional consideration. To account for the closed-loop configuration, the loop was virtually "opened" by introducing a virtual joint. This allowed the use of DH parameters on 1 branch, while the constraints for the loop closure were mathematically enforced via loop-closure equations [1]. These equations ensure the geometry remains consistent and both kinematic branches converge at the same end effector position.

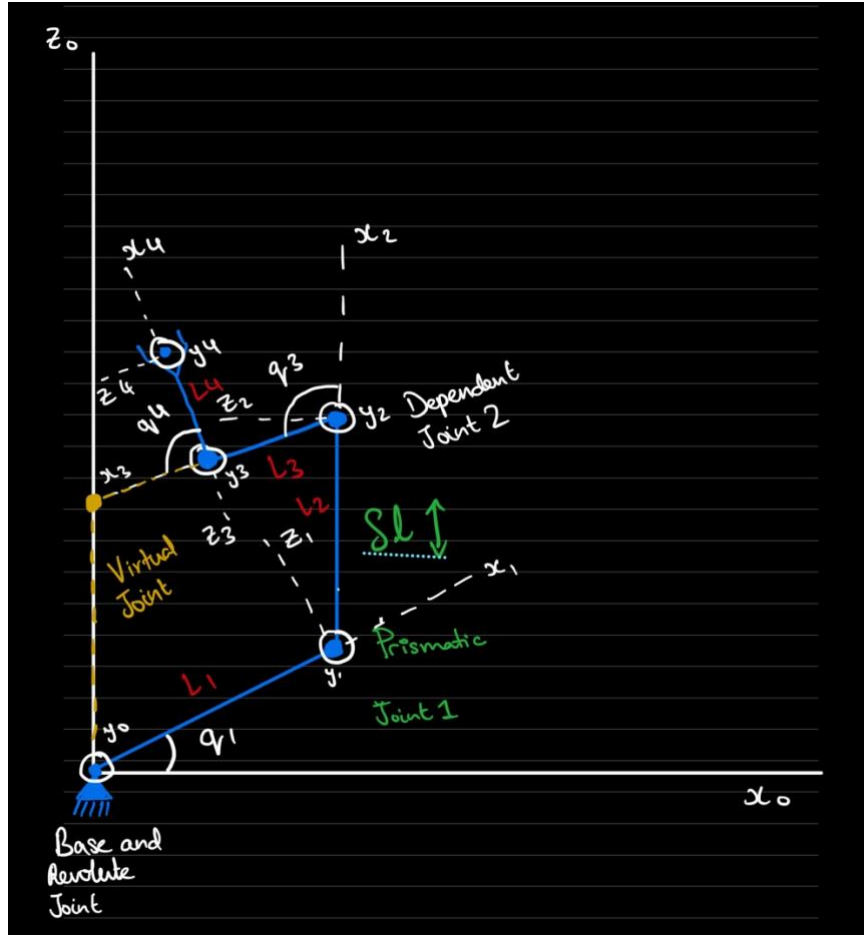


Figure 6: DH convention applied to Robot P

| i | θ_i (joint angle) | d_i (link offset) | a_i (link length) | α_i (link twist) |
|---|--------------------------|---------------------|---------------------|-------------------------|
| 1 | q_1 | 0 | L_1 | 0 |
| 2 | 0 | δl | L_2 | 0 |
| 3 | q_3 | 0 | L_3 | 0 |
| 4 | q_4 | 0 | L_4 | 0 |

Table 1: DH parameter values for each transformation

$$\text{Transformation Matrix } A_i = \begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & a\cos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using the values in the table above and the transformation matrix A the transformation matrices A_1, A_2, A_3 and A_4 can be derived:

$$A_1 = \begin{bmatrix} \cos(q_1) & -\sin(q_1) & 0 & L_1\cos(q_1) \\ \sin(q_1) & \cos(q_1) & 0 & L_1\sin(q_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \delta l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} \cos(q_3) & -\sin(q_3) & 0 & L_3 \cos(q_3) \\ \sin(q_3) & \cos(q_3) & 0 & L_3 \sin(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} \cos(q_4) & -\sin(q_4) & 0 & L_4 \cos(q_4) \\ \sin(q_4) & \cos(q_4) & 0 & L_4 \sin(q_4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using the equation:

$$T_0^4 = A_1 \cdot A_2 \cdot A_3 \cdot A_4$$

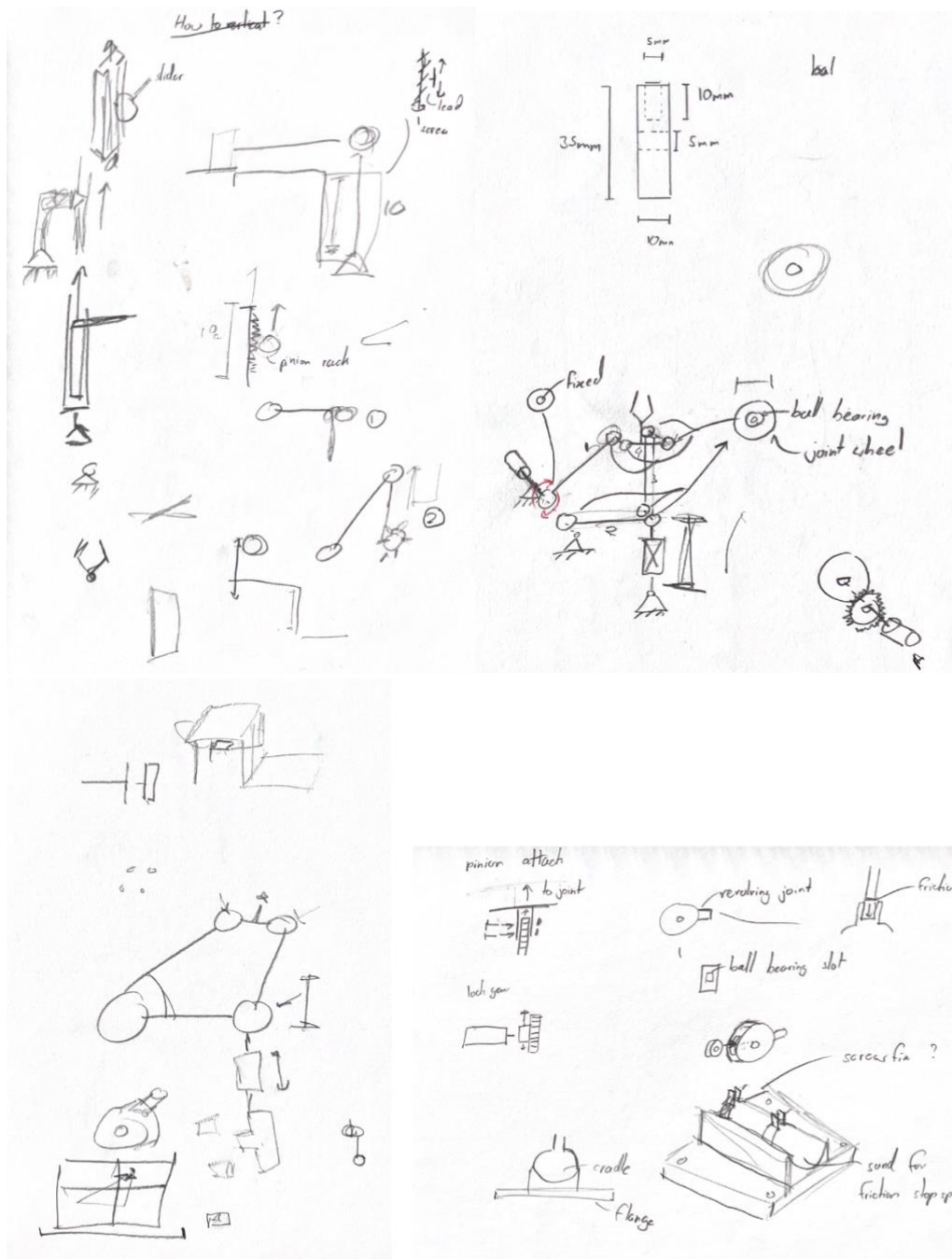
Gives an overall transformation matrix:

$$T_0^4 = \begin{bmatrix} \cos(q_1 + q_3 + q_4) & -\sin(q_1 + q_3 + q_4) & 0 & L_1 \cos(q_1) + L_3 \cos(q_1 + q_3) + L_4 \cos(q_1 + q_3 + q_4) \\ \sin(q_1 + q_3 + q_4) & \cos(q_1 + q_3 + q_4) & 0 & L_1 \sin(q_1) + L_3 \sin(q_1 + q_3) + L_4 \sin(q_1 + q_3 + q_4) \\ 0 & 0 & 1 & \delta l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The overall transformation matrix models the complete movement of all joints and links in the robotic arm, providing a complete mathematical model to describe the end-effector's position relative to the base which was crucial for both the control and the verification of the system's motion.

3.3 Design Sketches and Description

The robot's main structure is based on a 4-bar linkage configuration. The revolute joint enables angular displacement in the X-Z plane, while the prismatic joint allows for vertical linear movement of the end effector. Both joints are actuated using high torque DC gear motors equipped with encoders.



Figures 7-10: Depict Initial Ideas and Design Sketches

The vertical motion of the robot was actuated by a DC motor driving a pinion gear, which interfaces with a rack. As the pinion rotated, the rack translated vertically, extending or retracting the end effector. The angular motion was achieved using a gear, driven by the DC motor, to drive another gear of the same size. However, several mechanical inefficiencies were revealed during the testing of the robot such as slippage and torque inefficiency. There was inconsistent meshing between the 2 gears (of the revolute joint) and between the pinion rack (for the prismatic joint). This led to jerky and occasional stalling of the motion. The torque transferred to the rack was insufficient for smooth movement under load.

3.4 CAD Design and description

The mechanical design of Robot P was developed using CAD software (Fusion 360) to ensure accurate part manufacturing and mechanical integration. The overall system consisted of a 2-DoF robotic arm actuated by a revolute and prismatic joint. The design emphasised modularity and ease of assembly, using custom 3D printed components to house motor assemblies and provide structural support to moving parts. The 2 key components below were designed by the author of this report.

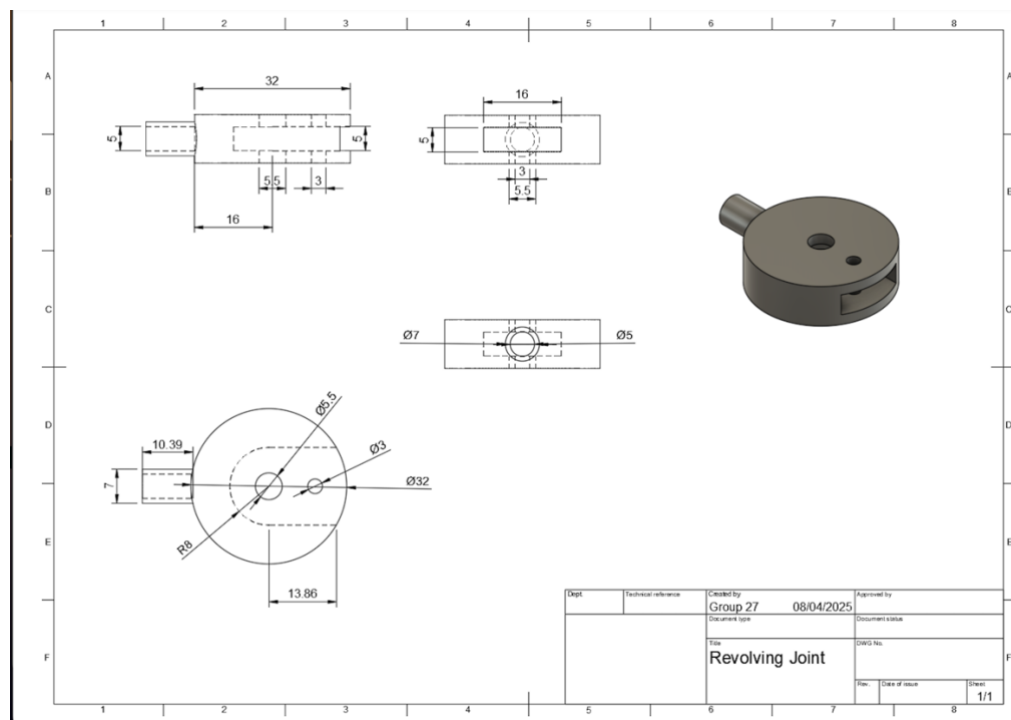


Figure 11: CAD of Revolving Joint

Each of the 4 main linkage points in the robotic arm utilised 3 identical revolving joints (see Figure 11) parts, providing the rotational freedom at each pivot point. These joints were designed to be compact, yet robust, with ball bearings that ensure smooth rotation (by reducing friction between moving parts) and even load distribution. The used of multiple revolving joints also improved the stability of the 4-bar linkage system. Using 12 of these revolving joints simplified the manufacturing and reduced the assembly time. It also made it much easier to replace parts.

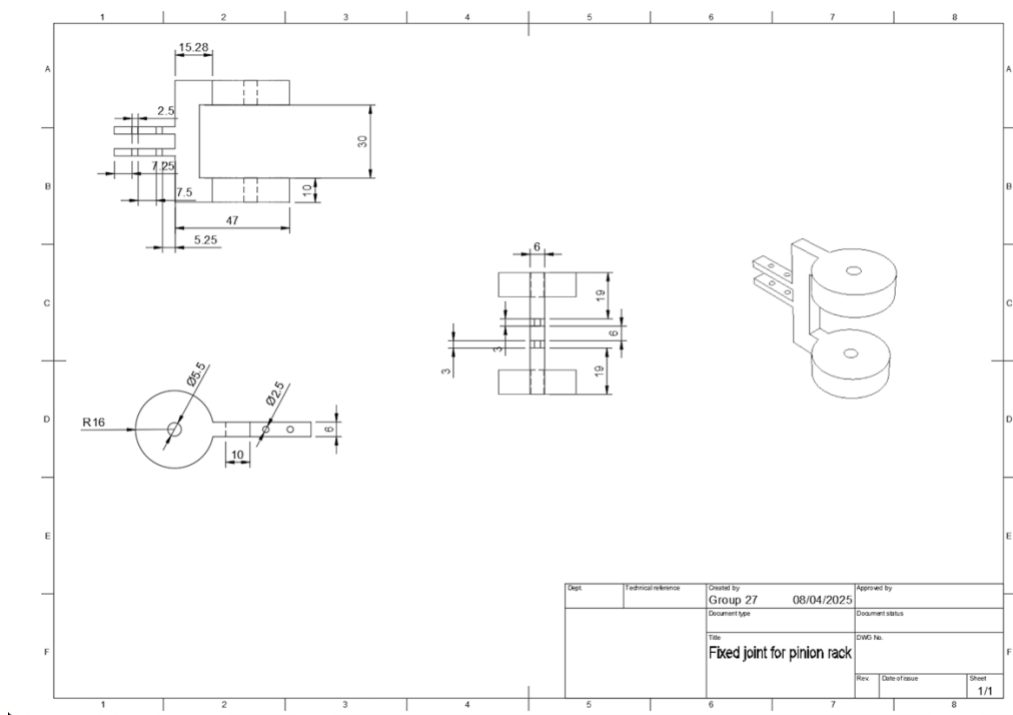


Figure 12: CAD of fixed joint for pinion rack

The fixed joint for pinion rack (Figure 12) is responsible for securing the vertical prismatic mechanism. It acts as a rigid interface between the pinion gear and rack assembly, allowing for 3 revolving joints to be placed in this joint. The outside 2 revolving joints connect to the revolute joint (actuated by a motor) whilst the middle revolving joint connects directly upwards to a dependent revolute joint. The design incorporates radial holes for attachment and precision cut-outs for motor and shaft mounting. The spacing and alignment of these holes were optimised for maximum stability during motor actuation.

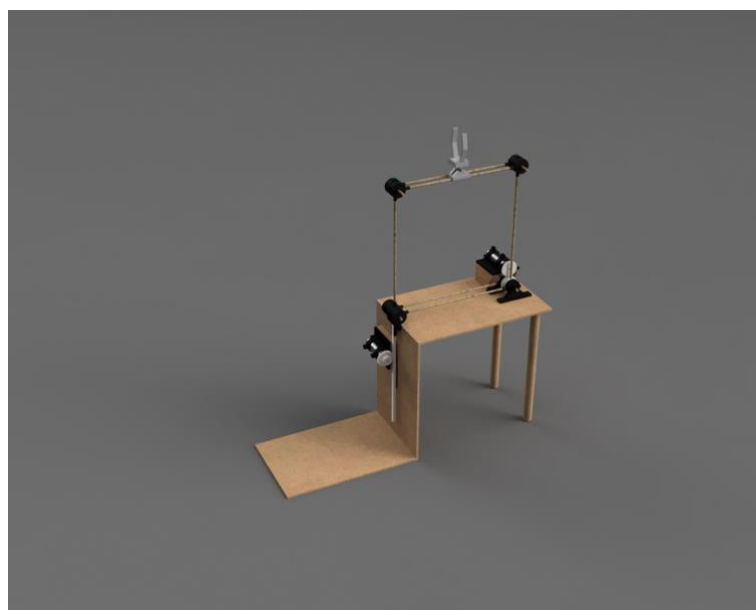
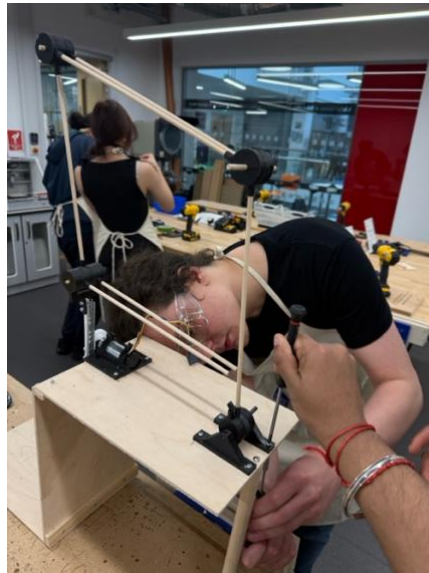


Figure 13: Full assembly of the robot

3.5 Manufacturing Processes

The construction of Robot P involved a combination of manufacturing methods, tools and manual assembly techniques to build a functional prototype of the CAD design. The main joints and 3D printed parts such as the motor mounting bracket, revolving joint, end effector gripper, etc were made with HDPE (high-density polyethylene) for the 3D printed parts plywood for the base platform. HDPE was used for the 3D printed parts due to its durability, lightweight properties and moderate flexibility- making it suitable for robotic applications where repeated motion and impact absorption are to be withstood. After printing, these parts were lightly sanded and checked for dimensional accuracy before integration. For the base, plywood sheets were cut to size using hand saws, creating flat and rigid foundation onto which the arm was mounted. Plywood was chosen for its ease of machining and structural rigidity, providing a stable base for the robotic arm.



Figures 14: Manufacturing of Robot P

The assembly process primarily relied on screwing and friction fits, allowing for quick disassembly and reassembly. The mounting bracket was secured to the base using drilled pilot holes and self-tapping screws to ensure a tight and secure fit. Additional holes were also drilled using power drills. (See Figure 19 in Appendix for Revolving Joint)

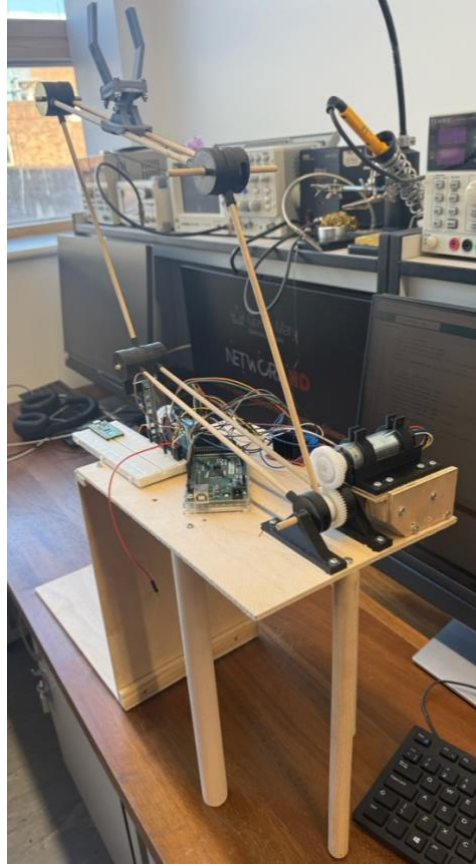


Figure 15: Final functional prototype

3.6 Electromechanical System

The robotic system for the Robot P uses electromechanical actuation for both the revolute and prismatic joints using brushed DC gearmotors with integrated encoders. These motors convert electrical energy into mechanical rotations and execute the joint movements required by the control algorithm.

Each joint is actuated by a 12V high-power brushed DC motor with a 46.85:1 metal spur gearbox. [2] The gearbox provides the necessary torque amplification to handle mechanical loads while reducing the speed to a manageable level for precise control. There is a 48 counts per revolution (CPR) quadrature encoder (2248.8 counts per revolution at the output shaft- see Figure 26 in Appendix) mounted on the motor shaft. This high resolution enables the accurate measurement of angular and linear displacements, essential for precise joint control. To test each DC motor independently before full system integration, basic Arduino code (see Figure 20 in Appendix) was used to ensure proper functionality.

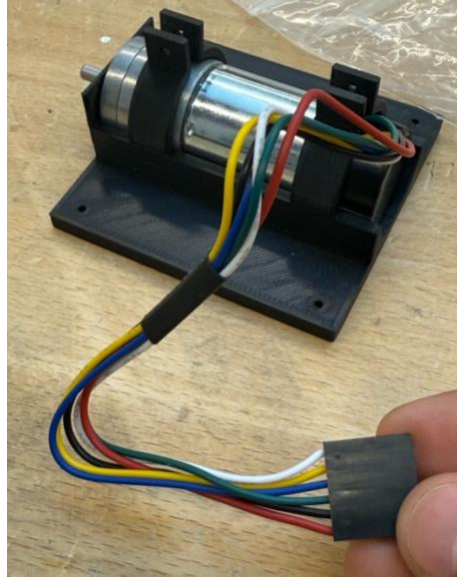


Figure 16: DC motor with encoder

To interface the motors with the Arduino Mega microcontroller, a L298N dual H-bridge driver was used (see Figure 23 in Appendix). This allows the bidirectional control of both motors and enables PWM (Pulse Width Modulation) control for the DC motor's speed. The ENA and ENB pins on the H-Bridge are PWM inputs (connected to digital PWM pins 10 and 13 on the Arduino Mega), while IN1, IN2, IN3 and IN4 were used for motor direction control.

The Arduino Mega is a relatively low cost, easily programmable microcontroller that works with a variety of components. Use of the Arduino guaranteed the simplicity and reproducibility of this project. Furthermore, C++ programming on the Arduino IDE allows libraries such as "PID_v1.h" to simplify the design process of the PID controller. The Arduino Mega's high number of I/O pins and multiple hardware interrupts make it well suited for this project.

3.7 Control System Integration

The control system was developed in Arduino IDE and modelled in MATLAB Simulink, to simulate the entire system. The process begins with the user defining target coordinate in the X-Z workspace of the robot. These inputs are passed to a MATLAB Function block (see Figure 24 in Appendix) that computes the required joint parameters θ_{target} (for the revolute joint) and Δd_{target} using inverse kinematics equations derived earlier in the project.

A Proportional-Integral-Derivative (PID) controller is one of the most used feedback control algorithms in engineering applications due to its simplicity and effectiveness in a variety of systems. The PID controller consists of 3 distinct parts: Proportional (P), Integral (I) and Derivative (D). In a closed loop, the PID controller continuously calculates an error value as the difference between a desired setpoint and the actual measured value. It then attempts to

minimise this (immediate) error by adjusting the system input accordingly. The proportional term produces an output that is proportional to the current error. The larger the error, the greater the output signal (error-correction signal generated). However, the steady-state error (difference between desired output and the actual output of a system after it has settled, after transient time) can only be addressed through the integral term. This eliminates offset and ensures that the system reaches the desired target. Finally, the derivative term predicts the future trend of the error based on the rate of change of the error, allowing the controller to apply dampening and reduce overshoot and oscillations.

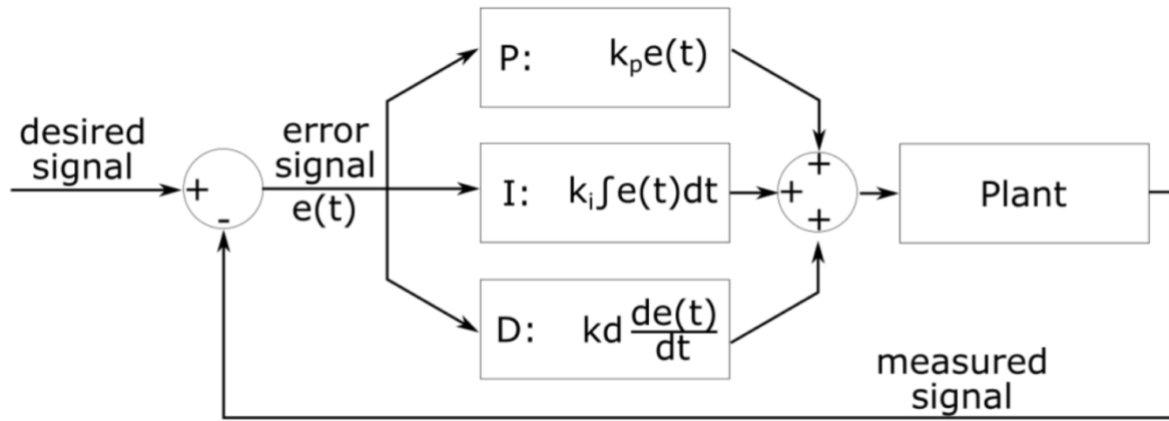


Figure 17: A typical PID system [3]

By combining these 3 error processing functions, the PID controller can achieve fast response times, minimal overshoot and zero steady-state error when tuned (for optimal coefficients). The gains for each term (K_P, K_I, K_D) must be chosen by thorough experimentation of the system's response in real time. For Robot P, the PID controller is ideal for precise position control, ensuring that the joint movements respond accurately to user commands, even in the presence of background noise. The final fine-tuned values for PID gain coefficients for Robot P were:

$$K_P = 2$$

$$K_I = 0.5$$

$$K_D = 0.1$$

Each joint is controlled independently using its own PID feedback loop. In the revolute joint loop, the calculated θ_{target} is compared to the actual θ_{actual} using a summing block. The error is passed to a PID controller. The PID output is scaled through a gain block (to represent the dynamics of the motor) and integrated to simulate the angular position. The prismatic joint also follows the same structure: Δd_{target} is compared to Δd_{actual} , and the error is passed through a second PID loop followed by gain and integration to model the extension of the prismatic motor.

To verify system performance, another MATLAB Function block (see Figure 25 in Appendix) was used to calculate the forward kinematics using θ_{actual} and Δd_{actual} . This allows a comparison of the actual position of the end effector with the original X-Z target input. The target and actual values are combined using a Mux block and visualised using a Scope block, allowing for clear observation of the system's behaviour.

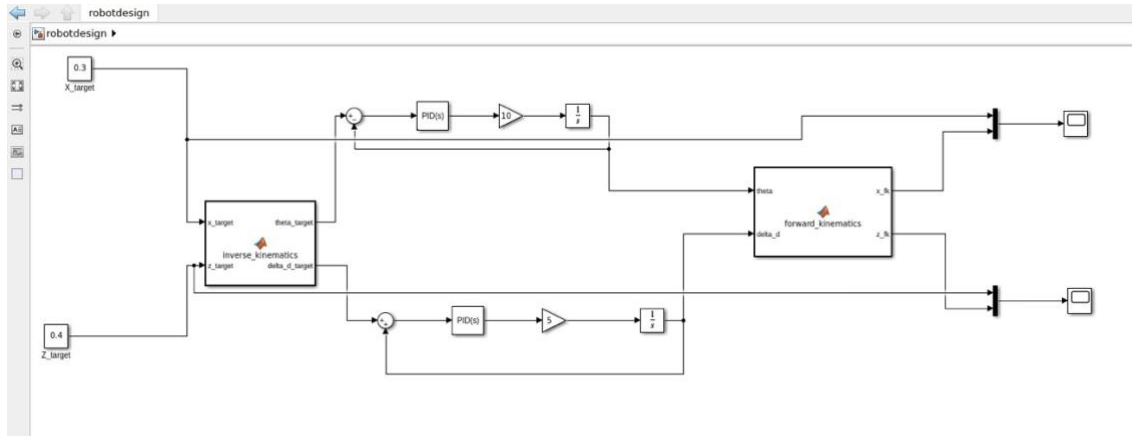


Figure 18: Full Control System Integration

The PID control system was implemented both in simulation (using Simulink) and on hardware (using Arduino Mega using PID_v1 library). Each joint of the robot was assigned an independent PID controller. The revolute joint's angle (θ_{actual}) and the prismatic joint's extension (Δd_{actual}) were measured using encoders, and PID output signals were sent to the respective motors via PWM through the L298N H-Bridge driver (see Figure 27 in Appendix). The `pid_theta.Compute()` function calls the PID control algorithm for the revolute joint. It then calculates the required `rev_motor_speed` (PWM output) based on the error between `theta_target` and `theta_actual`. The `pid_linear.Compute()` function does the same for the prismatic joint. Both `Compute()` functions are called in the `loop()` to constantly update the control signal. The `controlMotor()` function sends the `rev_motor_speed` output (from the PID controller) to the DC motor that contains the revolute joint. This same function takes in the `prism_motor_speed` argument to control the speed of the DC motor controlling the prismatic joint. The Simulink model was used to validate the general control structure and tuning approach, while the Arduino implementation was used for real-time control of the robot.

Despite the successful development of the full kinematics-based control system on Simulink, implementing the same complexity directly on Arduino IDE proved challenging due to the sheer complexity of the kinematics model. Therefore, a simpler control strategy was used for real-time manual control using a WASD keyboard interface. This allowed the user to manually control the revolute and prismatic joints using key presses, while still leveraging the PID

feedback control system for smooth motion. The code used can be seen in Figure 31 of the Appendix.

This manual control system, although less automated than the full kinematics-based system, proved highly effective for validating PID control loops of Robot P due to its simplicity.

4.0 Results

To evaluate the performance of the robotic arm's control system, a Simulink model that replicates the inverse kinematics, PID control and forward kinematics of the physical robot was developed. This simulation-based approach provided a way to test the system's response to different target inputs without using physical hardware. The results focus on the system's ability to control both the revolute and prismatic joints independently using PID controllers and to accurately compute the end effector's position using forward kinematics. Time-domain plots were generated to observe how effectively each joint responded to its target position.

| Trial | Kp | Ki | Kd | Notes |
|-------|-----|-----|-----|---|
| 1 | 0.5 | 0.5 | 0.5 | Unstable, simulation error |
| 2 | 1.0 | 0.3 | 0.3 | Improved, lot of overshoot (see Figure 28 in Appendix) |
| 3 | 2.0 | 0.2 | 0.2 | Faster rise time and better with slight overshoot (see Figure 29 in Appendix) |
| 4 | 2.0 | 0.5 | 0.1 | Best result, fast rise time, minimal overshoot (see Figure 30 in Appendix) |

Table 2: PID finetuning process

To ensure stable and accurate motion of the robot arm, the PID were fine-tuned using the Simulink model. Initially, equal gain values of 0.5 were applied to all three PID components (Kp, Ki, Kd) however, this led to a numerical instability and a simulation error due to the integral component. In the second iteration, Kp was increased to 1 whilst Ki and Kd were lowered to 0.3 to reduce the instability and overshoot. This configuration showed improved performance but still exhibited a noticeable overshoot (see Figure 28 in Appendix).

In Trial 3, K_p was increased to 2, with K_i and K_d reduced to 0.2. This resulted in a much faster rise time and better accuracy with only a slight overshoot (see Figure 29 in Appendix). Finally, in Trial 4, the gains were adjusted to 2.0, 0.5 and 0.1 for K_p , K_i and K_d respectively. Although this setup showed slightly more overshoot than the previous trial, a much steeper initial gradient, meant that the system settled faster to the θ_{target} (see Figure 30). Therefore, the final PID gains of 2.0, 0.5 and 0.1 were selected. This fine-tuning process highlighted the importance of balancing proportional, integral and derivative gains to achieve stable control of both joints in the robot arm.

5.0 Discussion and Conclusion

This project successfully achieved the core objectives set at the start: the robotic arm with 2 degrees of freedom (DoF) was designed, built and controlled effectively. Complex DH kinematics were derived to model the robotic system, including forward and inverse kinematic equations specifically tailored to the mechanical design of Robot P. A PID control system was implemented and successfully integrated both in Arduino IDE (for real-world testing) and in Simulink for simulation and verification. Controlled motion of both the revolute and prismatic joints was achieved, demonstrating effective independent feedback control for each motor.

Throughout this module, a wide range of skills and technical knowledge were developed. CAD skills were significantly improved through the detailed modelling of the robot in Fusion 360. Practical electronics skills were strengthened through the wiring of H-bridge motor controllers and encoders, as well as the implementation of encoder feedback systems. Software development skills were also improved through programming in Arduino IDE, using OOP concepts and learning how to tune and integrate PID controllers in both hardware and simulation environments. Presentation skills were also developed by pitching the robot to the module organiser and answering their questions.

While testing, it was observed that the robot's mechanical structure presented additional challenges. The arm relied on friction to hold joints together, which required higher torque from the motors to maintain stability during movement. Additionally, due to robot having long arms, there was noticeable flexing that occurred when the motors operated at higher speeds. This flex increased the load on the motors and reduced the accuracy of the control system, suggesting that future designs should have shorter arm lengths. The system may have also been more stable if more bracing was added between the links.

6.0 References

- ## 7.0 Appendix

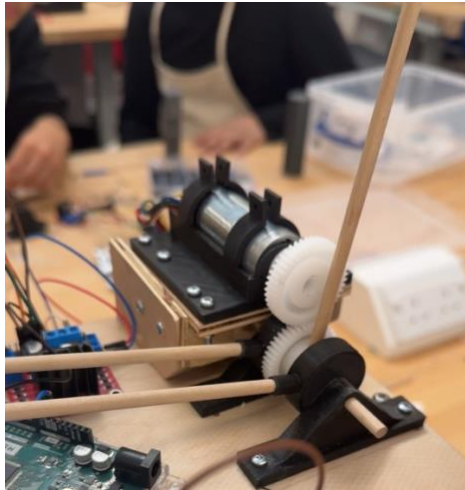


Figure 19: Close up of revolute joint

```
controlMotor(100, REV_MOTOR_IN1, REV_MOTOR_IN2, REV_MOTOR_PWM); // Forward
delay(2000);
controlMotor(-100, REV_MOTOR_IN1, REV_MOTOR_IN2, REV_MOTOR_PWM); // Reverse
delay(2000);
controlMotor(0, REV_MOTOR_IN1, REV_MOTOR_IN2, REV_MOTOR_PWM); // Stop
```

Figure 20: DC motor control check

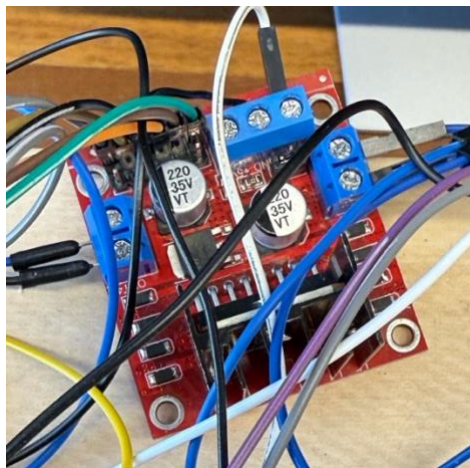


Figure 21: L298N H-Bridge

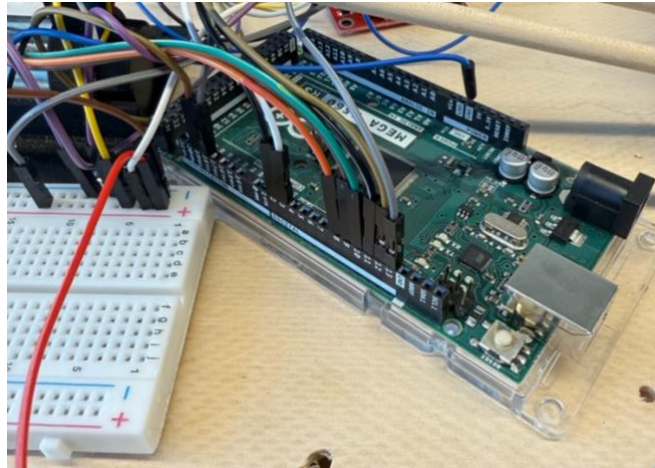


Figure 22: Close up of Arduino Mega

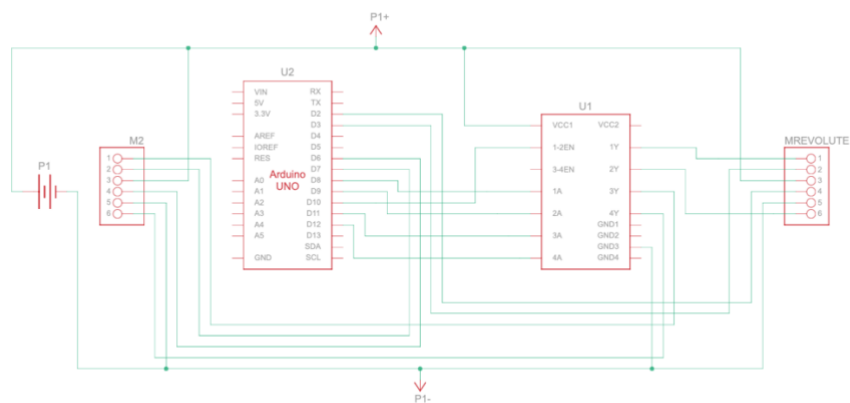


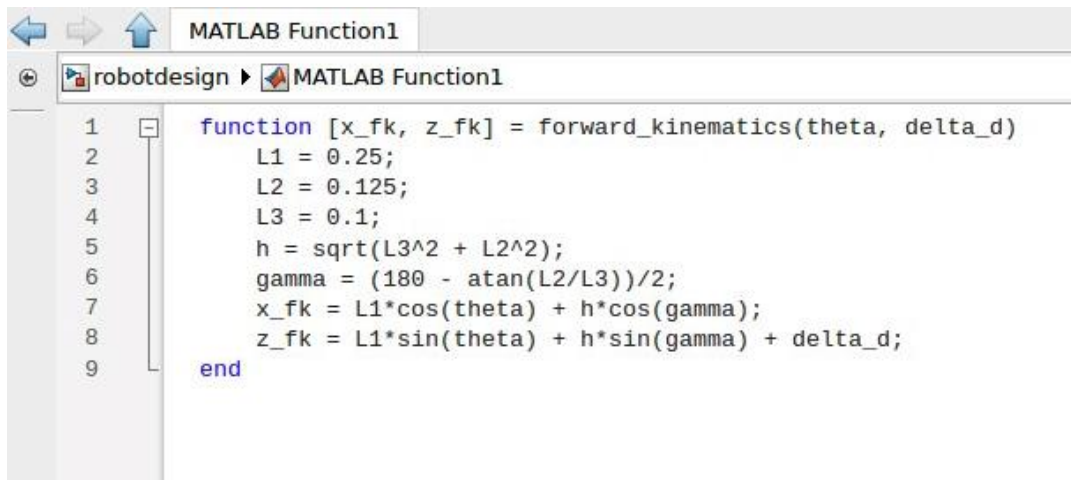
Figure 23: Schematic of Electromechanical System

```

MATLAB Function
robotdesign MATLAB Function
1 function [theta_target, delta_d_target] = inverse_kinematics(x_target, z_target)
2     L1 = 0.25;
3     L2 = 0.125;
4     L3 = 0.1;
5
6     h = sqrt(L3^2 + L2^2);
7     gamma = (pi - atan(L2/L3))/2;
8
9     theta_target = acos(x_target - h * cos(gamma) / 1);
10    delta_d_target = z_target - (L1 * sin(theta_target) + h * sin(gamma));
11    end

```

Figure 24: MATLAB code for Inverse Kinematics



```

1  function [x_fk, z_fk] = forward_kinematics(theta, delta_d)
2      L1 = 0.25;
3      L2 = 0.125;
4      L3 = 0.1;
5      h = sqrt(L3^2 + L2^2);
6      gamma = (180 - atan(L2/L3))/2;
7      x_fk = L1*cos(theta) + h*cos(gamma);
8      z_fk = L1*sin(theta) + h*sin(gamma) + delta_d;
9  end

```

Figure 25: MATLAB code for Forward Kinematics

```

void updateRevEncoder() {
    int stateB = digitalRead(REV_ENCODER_B);
    if (stateB == HIGH) {
        rev_encoderCount++;
    } else {
        rev_encoderCount--;
    }
    theta_actual = (rev_encoderCount * 2.0 * 3.1416) / encoderCountsPerJointRev;
}

void updatePrismEncoder() {
    int stateB = digitalRead(PRISM_ENCODER_B);
    if (stateB == HIGH) {
        prism_encoderCount++;
    } else {
        prism_encoderCount--;
    }
    delta_d_actual = prism_encoderCount * mmPerCount;
}

```

Figure 26: Method to turn feedback from encoders into theta actual and delta d actual

```

// Compute PID outputs
pid_theta.Compute();
pid_linear.Compute();

// Control Motors
controlMotor(rev_motor_speed, REV_MOTOR_IN1, REV_MOTOR_IN2, REV_MOTOR_PWM);
controlMotor(prism_motor_speed, PRISM_MOTOR_IN3, PRISM_MOTOR_IN4, PRISM_MOTOR_PWM);

```

Figure 27: PID controller computing required motor speed

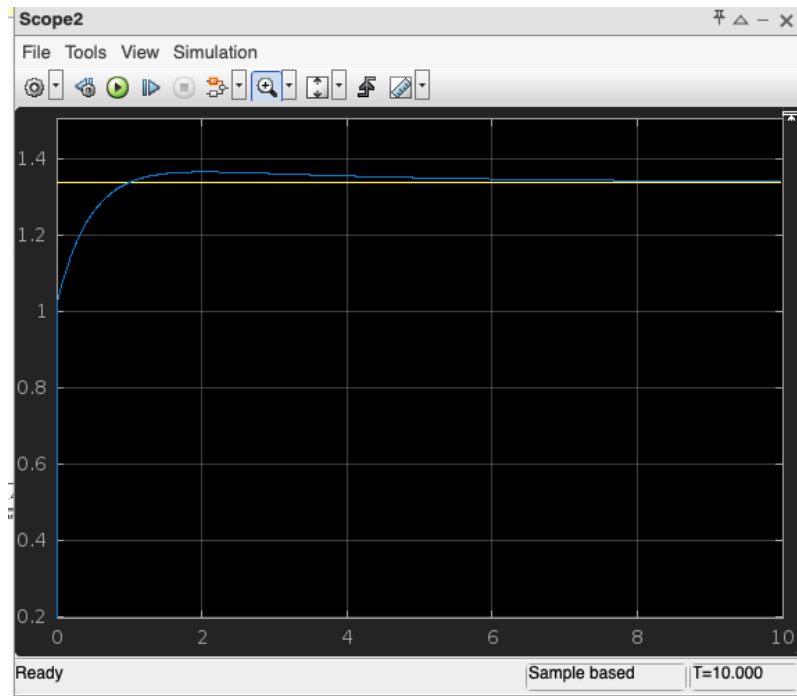


Figure 28: θ_{actual} (blue line) vs θ_{target} (yellow line) PID: 1, 0.3, 0.3

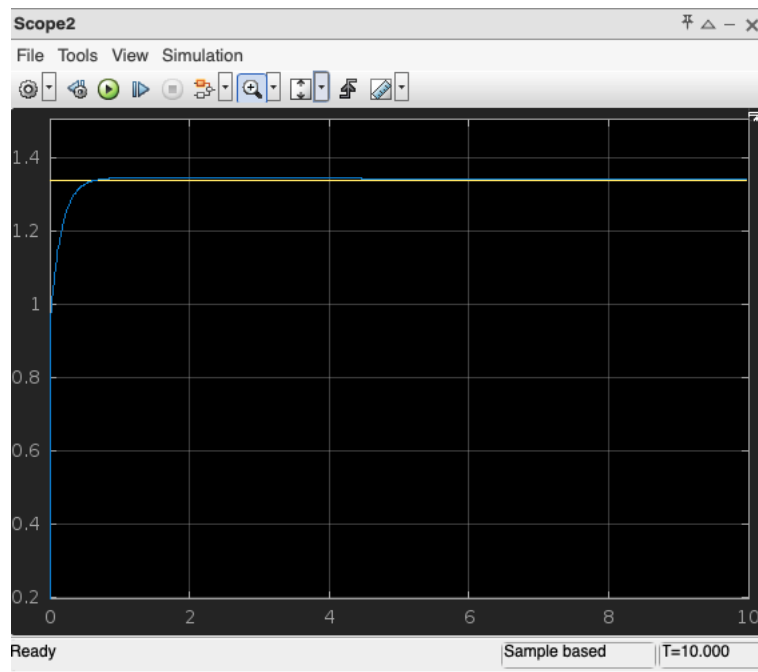


Figure 29: θ_{actual} (blue line) vs θ_{target} (yellow line) PID: 2, 0.2, 0.2

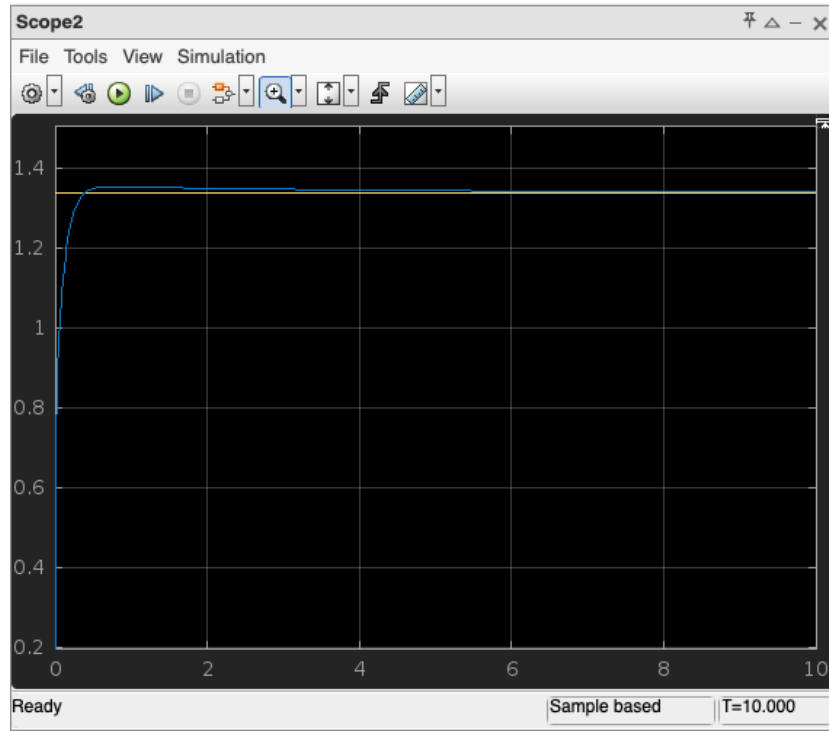


Figure 30: Θ_{actual} (blue line) vs Θ_{target} (yellow line) PID: 2, 0.5, 0.1

| Component | Quantity |
|--------------------------------------|----------|
| Arduino Mega | 1 |
| L298N H-Bridge | 1 |
| DC Motors with encoders | 2 |
| Revolving joints | 12 |
| Power Supply | 1 |
| Breadboard | 1 |
| Jumper Wires | 24 |
| Usb C to Usb B cable | 1 |
| Power leads | 2 |
| Crocodile clips | 2 |
| 0.25m Dowels | 6 |
| Rack (for prismatic joint) | 1 |
| Spur gear | 3 |
| Motor mounting bracket | 2 |
| Plywood boards | 3 |
| Slider (for prismatic joint) | 1 |
| Fixed joint for pinion rack | 1 |
| Supporting legs (1.25cm radi dowels) | 3 |
| End effector gripper | 1 |
| End effector arm link | 1 |
| End effector middle | 1 |

Table 3: BOM (Bill of Materials)

```

void loop() {
  // Read keyboard input (simulated here for demonstration)
  if (Serial.available()) {
    char key = Serial.read();

    // Control theta with left/right arrow keys
    if (key == 'a') { // Left arrow key
      Setpoint -= 1; // Decrease setpoint
    } else if (key == 'd') { // Right arrow key
      Setpoint += 1; // Increase setpoint
    }

    // Control extension with up/down arrow keys
    if (key == 'w') { // Up arrow key
      analogWrite(PRISM_MOTOR_PWM, 255); // Extend prismatic joint
    } else if (key == 's') { // Down arrow key
      analogWrite(PRISM_MOTOR_PWM, 0); // Retract prismatic joint
    }
  }
}

```

Figure 31: Implementation of keyboard based control