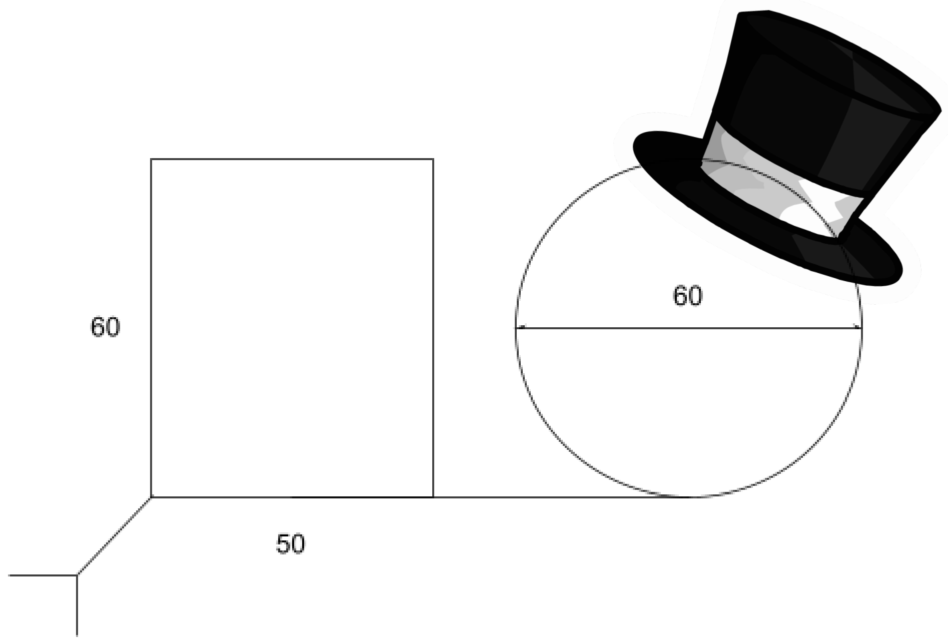


Project 1 – Planar Worktable Control

MECHENG 306: Design of Sensing and Actuating Systems

Group 2
Eloise Beattie, Luke Hynds, Oliver Reedy & Otto Walker

August 25, 2023

Table of Contents

Introduction

This report presents a comprehensive analysis of the design and programming decisions involved in creating a planar worktable capable of accurately sketching both rectangles and circles of specified dimensions, all while minimising errors and optimising speed. This report begins with highlighting a critical aspect of this project, the calculation of the worktable kinematics, which played a pivotal role in precisely determining the required motor rotations to achieve specified coordinates. The implementation of limit switches was essential to safeguard the worktable and establish a common home position that are elaborated on in subsequent sections along with the control strategy. This employed a combination of closed-loop P controllers and time-based position control to effectively manage motor speed and rotation. The following sections of this report elaborate on the details of our planar worktable design, control mechanisms, and the methodologies adopted to achieve precise geometric sketching with maximum efficiency and minimal error.

.

1 Mechanical Platform Kinematics

1.1 Design choices and decisions

Equations governing the worktable were given to the team along with the above diagram tracing the line of the two belts that transmit motor power to the pen holder. As all design choices had already been made by the platform manufacturer, the job of the team was now to interpret and work around the limitations of the hardware. This involved deriving further equations, and determining how exactly the program would interface with the platform.

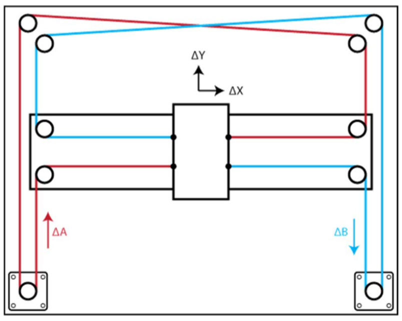


Figure 1 - Gantry Coordinate System

1.2 Implementation

The platform is controlled by two motors, denoted by the red and blue lines in Figure 1. For the rest of this section, the left motor denoted in red will be called A, and the right blue motor will be called B.

The equations given detail motor displacement:

$$\Delta A = \Delta X + \Delta Y$$
$$\Delta B = \Delta X - \Delta Y$$

As well as pen holder displacement:

$$\Delta X = \frac{\Delta A + \Delta B}{2}$$
$$\Delta Y = \frac{\Delta A - \Delta B}{2}$$

Both in the cartesian plane. The motors contain 2 encoders each, so to accurately determine the distance the motors were moving, the gear ratio of 171.19:1 was obtained from the motor datasheet, and the amount of encoder counts in one revolution was decided as 24, as shown in the closed loop control section, and the wheel that ran the belts attached to the motor was measured to be 14.26mm in diameter, leading to the equation:

$$EncRatio = \frac{171.79 \times 24.0}{14.26 \times \pi}$$

Where EncRatio is the ratio of encoder counts to distance in millimetres, determined by dividing the gear ratio and encoder counts in a circle by the circumference. This is because the encoders are measured before the motor is geared down, to allow for a higher degree of accuracy. The encoder ratio is then used to multiply a given distance from the motor displacement equations to turn it into an amount of encoder counts that the controller can send to the motor.

2 Implementation of Limit Switches

Limit switches serve a dual purpose in our gantry system, contributing to both the system's operational safety and its precision. This section explores the two implementations of our limit switches: Hardware Protection and Datum Calibration.

2.1 Operational Bounds

To ensure that we don't damage our gantry, we have created a safety system to stop the motors moving the gantry out of system boundaries. To do this, we use limit switches positioned at the limits of the gantry at X-, X+, Y- and Y+, which we refer to as Left, Right, Bottom and Top. These are shown in figure 2 below.

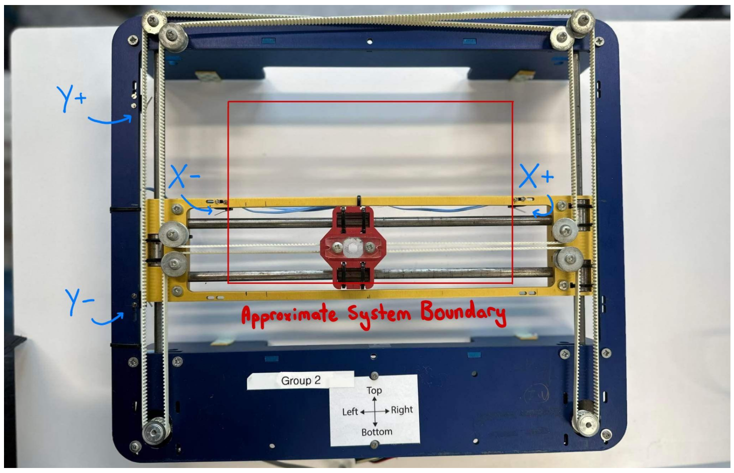


Figure 2 - Location of Limit Switches & System Bounds

The system works by constantly polling in the background checking if any of the limit switches are pressed. Our system polls for state changes of the limit switches at 100Hz. Upon detecting the activation of a limit switch, the system immediately stops both motors to avoid damage, and then moves the gantry in the opposite direction to the switch that was pressed. The program then goes into an idle state, requiring a new datum.

The main purpose of the safety system is to prevent movements that could result in mechanical strains or stresses, such as belt rupture or motor overexertion. While our finished design should never be able to exit system boundaries, this system was essential while designing our program. The knowledge that there was a safeguard in place if something unforeseen were to occur, allowed us to test with confidence.

2.2 Datum Calibration

We need to ensure that our drawing is always in the same position within the system bounds. This needs to be accurate and consistent to ensure the precision of the gantry's movement. To do this we need to define a datum, or reference point, which serves as the origin for all subsequent movements. Therefore implementing a method to reliably and automatically establish this datum is the most important step in the initialisation process.

Upon the start of the system's process is a homing sequence. The sequence orchestrates the movement of the gantry until the datum is firmly established. This process involves several steps:

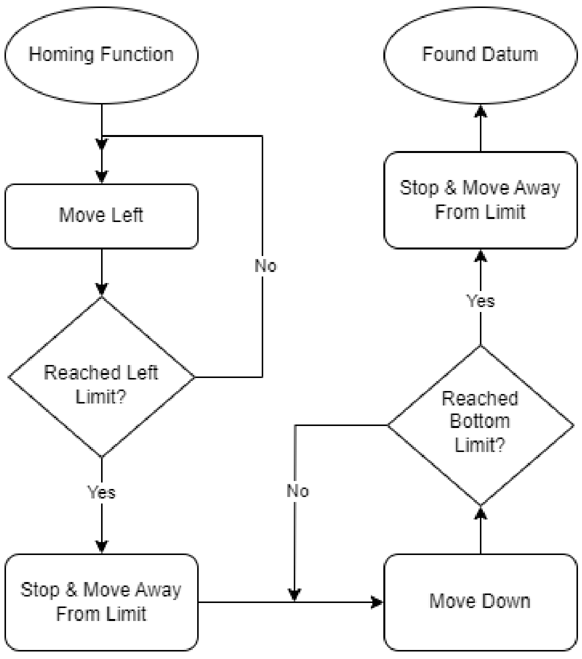


Figure 3 - Homing Function Flowchart

Leftward Movement: The homing sequence commences by moving the gantry to the left, a programmed infinite distance, until the activation of the left limit switch. This action serves as a marker of the extreme left boundary.

Rightward Adjustment: Subsequently, the gantry makes a controlled shift to the right, approximately 5mm, ensuring it is comfortably positioned within the system's boundaries.

Downward Movement: The gantry then moves downward, once again with a programmed infinite distance, until the activation of the bottom limit switch. This point signifies the lowest point within the system's boundary.

Upward Adjustment: A controlled upwards movement follows, shifting the gantry upward by about 5mm. This manoeuvre restores the system to a position well within the established operational bounds.

This predefined set of movements establishes the datum point, which we use as the (0, 0) coordinate of our XY coordinate system, as in figure 4. All movements made after the homing function has run use this point as reference to where it is within the system bounds.

In essence, the integration of limit switches not only ensures operational safety but also creates a consistent reference point. This combined functionality lays the groundwork for precise and reliable gantry operations.

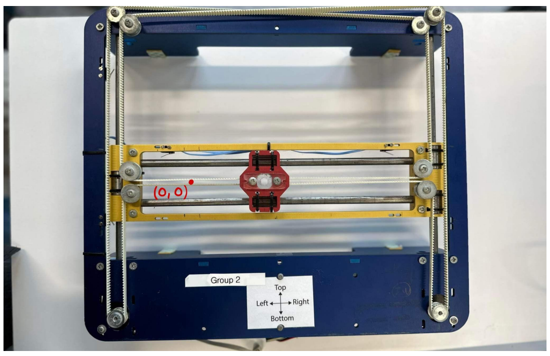


Figure 4 - Location of Datum on Worktable

3 Closed Loop Control

3.1 Controller Design Choice

The design choice of implementing a closed loop control system was decided after preliminary testing at the beginning of the project. After comparing the accuracy and speed of both open loop (OL) and closed loop (CL) controllers it was clear that the OL controller was too unreliable and a higher level of accuracy was required than what the OL controller could provide. Therefore the following CL control system was designed.

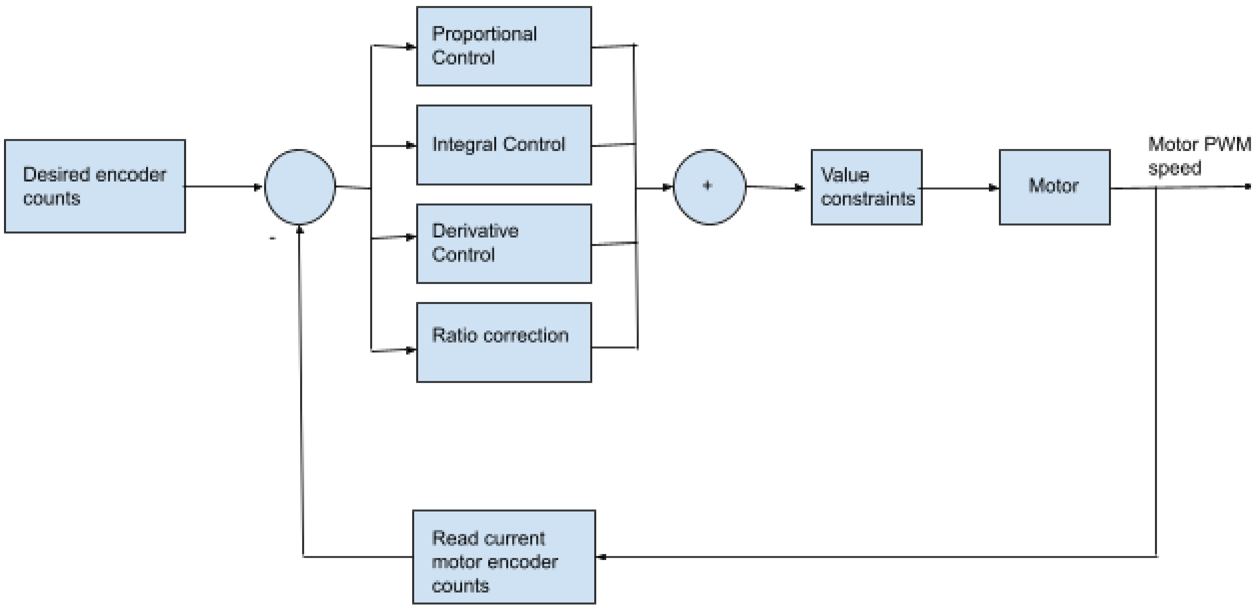


Figure 5 - PID Closed Loop Controller

As shown in the figure above, the decision was made to use encoder counts for the control system. This was to make the system more integrated with the hardware and minimise the number of calculations occurring every iteration. A complete PID controller was designed originally with the output velocity constrained between 0 and 255, the range valid for a PWM motor.

$$u(t) = K_P e(t) + K_I \int_0^t e(t) + K_D \frac{d}{dt} e(t) + offset$$

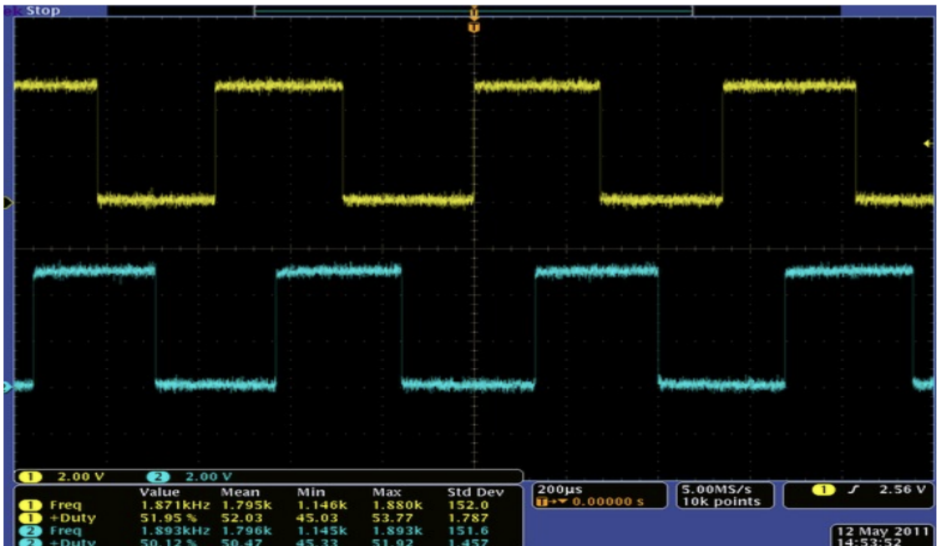
Use of different controllers was trialled by setting some controller coefficients to 0 and monitoring the effect. Due to the minimal effect of the integral and derivative controllers, only the proportional controller was implemented in the final design, allowing for the code itself to be far simpler.

3.2 Implementation of classes

3.2.1 MotorsWithEncoders

The implementation used C++ methods and programming along with two classes, MotorsWithEncoders and CoordinateFinder, which were used to fully implement this controller and CL system in tandem with the worktable kinematics calculations. The left and right motors labelled A and B respectively were created as instances of MotorsWithEncoders called motorA and motorB. This class implemented the following methods; setVelocity, stopMotor, setDirection, goToPosition, and incrementCount along with more getter and setter functions for variables.

The current position of each motor was found using the individual motor's encoders which were wired to interrupts on the Arduino. The motors used in this project implemented a two-channel Hall effect encoder that outputs two channels; A and B. Upon a change in output A, the interrupt called the increment motor count function which would add or subtract from the variable countPulses dependent on the direction of the motor. As the outputs A and B were 90 degrees out of phase, the direction could be identified by comparing outputs A and B and setting a direction boolean accordingly. Using only one encoder output for interrupts was found to be more reliable and resulted in consistent results with minimal error.



	Value	Mean	Min	Max	Std Dev
1 Freq	1.871kHz	1.795k	1.146k	1.880k	152.0
1 +Duty	51.95 %	52.03	45.03	53.77	1.787
2 Freq	1.893kHz	1.796k	1.145k	1.893k	151.6
2 +Duty	50.12 %	50.47	45.33	51.92	1.457

Figure 6 - Encoder A and B Outputs from the Motor Datasheet

The P-Controller was implemented within the goToPosition method which took in the desired counts and Kp. The error was calculated within this function using the following equation:

$$error \;=\; desired\, position \;-\; countpulse$$

The set velocity function was used then to send power to the motor in the correct direction based on the sign and value of the error and the proportional coefficient, Kp. Kp was variable as it was found a different value was more effective in completing the circle to the square.

3.2.2 CoordinateFinder

The coordinate finder class was used to call the motor instances simultaneously and calculate their respective desired encoder counts when given a coordinate with respect to the zero set in the bottom left corner. Before the curve function was implemented, a ratio offset, as shown in Figure 5, was used to synchronise the motors. While this was effective for straight lines, it was not as consistent for angles which led to issues implementing the circle function. This resulted in the time-based curve function explained in the following section. The original method included setting each coordinate within an array and iterating through that array once the error was zero. This was unsuccessful for the circle task and instead, a method where the target position was smoothly changed in order to create the desired paths and shapes using the time-based calculation explained in section 4.

4 Time-Based Position Controller

4.1 Class Structure

A class called 'Curve' was created in order to calculate the motion of the controller between coordinates. The constructor takes the total time for the motion to complete in, the final position as a desired coordinate to reach, and an initial position to calculate the length of the line. Once a new coordinate is required, updateCurve is called which takes the same inputs as the constructor to reinitialise them to an updated position. Two instances of this class are created at the beginning of the main code, one for the X coordinates and one for the Y.

In order for the motion to be performed, the function that calculates the target position must be called within a loop, whilst passing in the current time each loop. Additionally, as the calculation is time sensitive, a separate function called beginCurve, which sets the initial time, must be called right as the motion is about to begin.

4.2 Position Calculation

A simple way to demonstrate the functionality of this class is with a linear line.

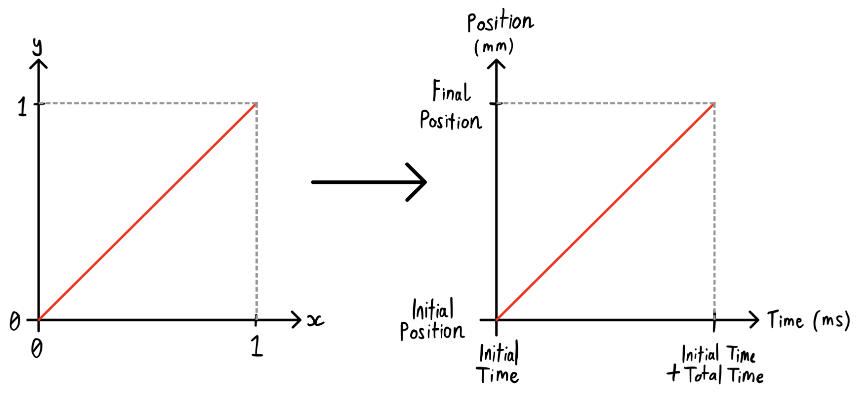


Figure 7 - Position vs Duration

As shown in Figure 7, a simple mathematical function is used to convert the current time into a ratio between the initial time and the final time. This value, now referred to as 'x,' is multiplied by the desired length of the line. Since the closed-loop controller and kinematic maths works by passing in coordinates to travel to, the 'desired length of the line' is calculated by subtracting the initial position of the curve's respective coordinate from the final targeted position. After the multiplication, the initial position is added to correct for the position it travels from.

Of note, the final iteration of the code uses the theoretical motion to determine when it is complete. Essentially, once the final time has been reached, and 'x' is 1, the curve will update and attempt to target the next position. This is a change from the initial technique of only updating to the next desired position once the encoders detect that they have reached the final position. Although the new method requires finer tuning to ensure the motors are precisely at the desired positions, it allows for exact control of the overall timing.

Once the 'Curve' class was confirmed to be performing the theoretical motion successfully, it became clear that targeting lines which varied linearly through time would cause the corners to be curved as the motor would not have time to slow down and speed up as it reached the corners. Instead of further tuning, mathematical functions were used to convert the linear line into different types of curves that would afford us precise control of how the pencil gantry would move between the points.

$$\text{Sine Ease-Out} : y = sin(\frac{x\pi}{2})\ \{0 < x < 1\}$$

$$\text{Quad Ease-In} : y = 2x^2\ \{0 < x < 0.5\}$$

$$\text{Quad Ease-Out} : y = 1 - \frac{(-2x+2)^2}{2}\ \{0.5 < x < 1\}$$

Converting the path into these curves was as simple as applying the correct function to the value of 'x,' whilst keeping the rest of the positioning equation the same. Two major curves were tested; a Quad Ease-In Ease-Out function, and a Sine Ease-Out function. In green is the original Linear Line function. The Quad Ease-In Ease-Out function was tested first with the prospect of ensuring that the corners would be performed smoothly. However, an analysis of the derivatives showed that although the quad would allow for the smoothest starts and stops, it would require the greatest speeds if attempted in the same time frame as other potential curves. It was found that the Sine controller provides a nice balance. Additionally, omitting the ease-in part of the function turned out to save additional time as the physical constraints of the motor combined with the P controller would act as an ease-in as the pencil gantry's physical position caught up the desired line, whilst the built-in Sine Ease-Out smoothed out the end of the line allowing for 90-degree corners.

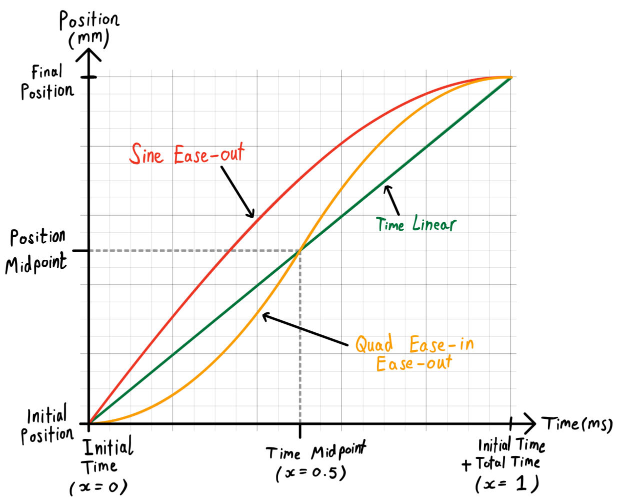


Figure 8 - Curves Plotted Position vs Time

Two final functions were required for creating circles, one for returning an X coordinate, and one for the Y coordinate. For this, the 'x' value was multiplied by 2 Pi, passed into a sin/cos function, and multiplied by the size of the desired radius. For the optimal route across the board, the circle needed to begin from the bottom and go counterclockwise, which meant that the X position would use a sine function, and the Y position would use a negative cosine function, with the radius added on to ensure it would not begin from the middle of the circle. The movement of the XY positions is shown in the dual-axis graph.

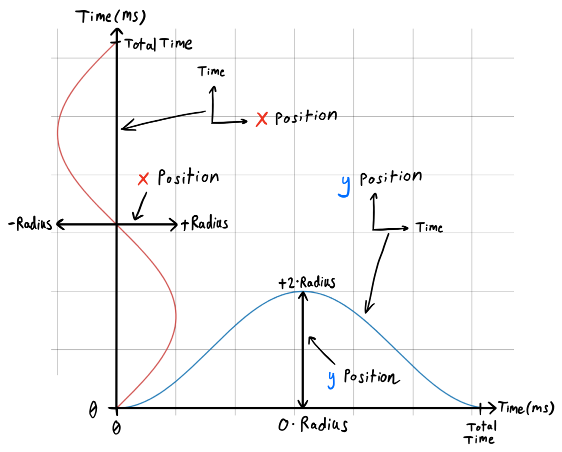


Figure 9 - Waves Constructing a Circle

Conclusion

Critical reflection on the project

Our project's aim was to successfully complete both tasks, the rectangle and the circle, optimising time and accuracy. We received a total error of zero millimetres during our display and completed the demonstration within the fastest time echelon, hence receiving full credit. Our design choices paid off and the result was an extremely robust and reliable system. If we were to repeat the project, some things to improve would be code sharing, version control, and time management by investing less time into ineffective and redundant features of the code, which would have resulted in a less stressful experience.

Contribution statement

I, Eloise Beattie, wrote the Closed Loop Control section of the report, worked on work-table kinematics, building the coordinate finder and motor controller system (MotorsWithEncoders), and assisted with integrating the final code.

I, Luke Hynds, wrote the Implementation of Limit Switches section of the report, worked on the implementation of limit switches and homing as well as final tuning to complete the demonstration in under 25s.

I, Oliver Reedy, wrote the Time-Based Position Controller report section, wrote the Time-Based Position Controller (Curve class), as well as worked on code architecture, MotorsWithEncoders class, polling, tuning, and main sketch implementation.

I, Otto Walker, wrote the Mechanical Platform Kinematics section of the report, assisted with writing and bug fixing of classes and final sketch.

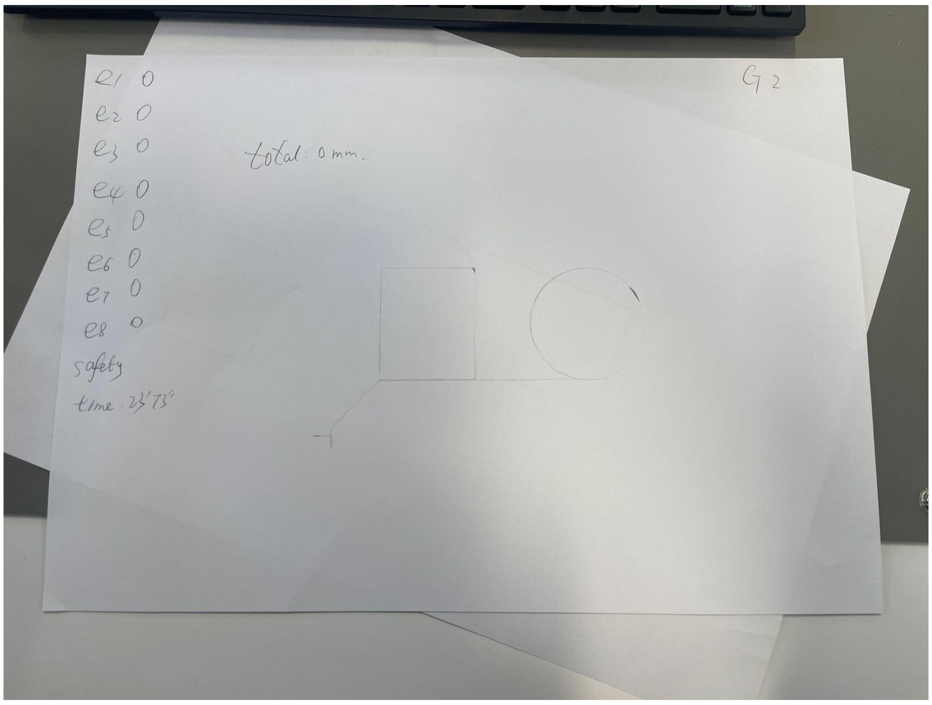


Figure 10 - Our Demonstration Drawing