Abstract

This thesis explores the construction of a ball balancing platform, a control system that stabilizes an inherently unstable metallic ball on a plate using robust control algorithms working in synchronization with real-time sensory feedback. The platform is mounted on top of a 3RRS (Revolute-Revolute-Spherical) parallel manipulator. The platform utilizes a touch screen (resistive touchpad) to detect the ball's position and employs three stepper motors to adjust the plate's tilt, aiming to minimize the error between the ball's actual and desired positions. The effectiveness of a PID (Proportional-Integral-Derivative) controller is evaluated within the linear domain of control. A Simulink model of the same is developed using Lagrangian Mechanics for a better theoretical understanding and in-depth analysis of the proposed control strategy. The thesis first establishes a theoretical model of the ball-on-platform scenario using MATLAB Simulink environment, then compares these theoretical results with those obtained from the physical construction.

Keywords: Mechatronics, Control Theory, PID, Microcontrollers, Balancing, Parallel Manipulator, Simulink.

Abstrakt

Niniejsza praca bada konstrukcję platformy do balansowania piłki, systemu sterowania, który stabilizuje naturalnie niestabilną metalową piłkę na płycie, wykorzystując solidne algorytmy sterowania działające w synchronizacji z informacjami zwrotnymi w czasie rzeczywistym. Platforma jest zamontowana na manipulatorze równoległym 3RRS (Revolute-Revolute-Spherical). Platforma wykorzystuje ekran dotykowy (rezystancyjny panel dotykowy) do wykrywania pozycji piłki i stosuje trzy silniki krokowe do regulacji nachylenia płyty, dążąc do minimalizacji błędu między rzeczywistą a pożądaną pozycją piłki. Skuteczność sterownika PID (Proporcjonalno-Integralno-Derivative) jest oceniana w ramach kontrolowania w liniowym obszarze sterowania. Został opracowany model w Simulinku oparty na mechanice Lagrange'a w celu lepszego zrozumienia teoretycznego i dogłębnej analizy proponowanej strategii sterowania. Praca najpierw opracowuje teoretyczny model scenariusza piłki na platformie przy użyciu środowiska MATLAB Simulink, a następnie porównuje wyniki teoretyczne z wynikami uzyskanymi z fizycznej konstrukcji.

Słowa kluczowe: Mechatronika, Teoria sterowania, PID, Mikrokontrolery, Balansowanie, Manipulator równoległy, Simulink.

Abbreviations

PID	Proportional-Integral-Derivative
BPS	Ball and Plate System
DOF	Degrees of Freedom
RRS	Revolute-Revolute-Spherical
DC	Direct Current
PV	Process variable
SP	Setpoint
PM	Parallel Manipulator
FLC	Fuzzy Logic Controller
LQR	Linear Quadratic Regulator
MPC	Model Predictive Controller
SMC	Sliding Mode Controller

Table of Contents

1 Introduction 1

1.1 Implementation summary: 1

1.2 Research Questions: 1

1.3 Similar Works 3

2 Literature Review 4

2.1 The Evolution of Control theory 4

2.2 PID Controller 5

2.3 Parallel Manipulators 7

2.4 Lagrangian Mechanics 7

3 System Design and Requirements 8

3.1 Objectives 8

3.2 Performance Metrics 8

3.3 Assumptions 8

3.4 Hardware Requirements 9

4 General Concepts of the system 10

4.1 Control Software 10

4.2 Inverse Kinematics of the system 13

4.3 Equations describing the motion of the ball. 16

5 Hardware Implementation 19

5.1 Mechanical Configuration 19

5.2 Electrical Configuration 20

5.3 Construction and Assembly 24

6 Software Implementation 27

6.1 Simulink Modelling 27

6.2 Position feedback from touchpad. 28

6.3 Calculating normal vectors 28

6.4 PID Tuning 29

6.5 Creating Trajectories 29

7 Results and Analysis 30

7.1 Assembled platform 30

7.2 PID Gains 31

7.3 Step Response testing 31

7.4 Frequency Response 33
7.5 Disturbance Rejection Test 34
7.6 Creating Trajectories 35
8 Conclusion 37
9 Further Research 39
10 References ..... **Error! Bookmark not defined.**

1 Introduction

Laboratories play a vital role in the effective education of engineering students. In the context of automatic control, students use the classical laboratory processes such as the water tanks, the inverted pendulum[1], the magnetic levitation system[2] and the ball-on-plate benchmark. In particular, the ball-on-plate process is very interesting since it requires multivariable stabilizing control. Additionally, the system dynamics are fast, necessitating short sampling intervals for the controller. Given that the control of fast, unstable, and multivariable systems is crucial in various practical applications, the ball-on-plate process is an effective example for control courses. [1]

This study investigates the implementation and effectiveness of linear control methodologies, specifically a PID controller in stabilizing the Ball-on-Plate System (BPS). The work encompasses both theoretical modelling of the proposed system in Simulink environment and building a working prototype to conduct experiments. System responses to external disturbance like human interaction and environmental factors will be primarily studied.

1.1 Implementation summary:

The work will be carried out on dual tracks: one theoretical simulation and the other physical experimentation. Arduino UNO based Microcontroller; Maker UNO will be the brains of the system. The required parts are fabricated through 3D printing. Performance evaluation will be based on step response and frequency response analysis. More about performance evaluation is described in Section 3.2. Comparative analysis between simulated and experimental results is carried out to identify and quantify important performance factors and system limitations. After successful stabilization, an attempt will be made to trace different trajectories on the platform by moving the ball accordingly.

1.2 Research Questions:

1. **Is the linear control methodology adequate for satisfactory performance in a ball-balancing platform system?**
   This question investigates the adequacy of linear control techniques in managing an inherently nonlinear mechanical system.

[1] https://en.wikipedia.org/wiki/Inverted_pendulum
[2] https://en.wikipedia.org/wiki/Magnetic_levitation

2. **Is it possible to create a digital twin of the prototype? If not than what are the causes of these differences between theoretical simulations and actual experiments?**

   The project aims at finding out the source of the difference between the theoretically predicted behaviour and practically observed behaviour and to quantify the causes-mechanically, electrically, and environmentally.

3. **Is it possible to trace different shapes by changing the SP at a regular interval and moving the ball accordingly?**

   This question intends to focus on exploring the ability of the system to not only stabilize the ball but also move the ball with precise control.

1.3 Similar Works

Over the years, various approaches have been developed to solve the problem of balancing a ball on a plate, each with distinct strengths and limitations. Vision-based feedback systems like the one in Figure 1.2. use cameras for precise ball tracking but face challenges with environmental sensitivity and computational demands. A resistive touchpad is a better alternative due to its ease of usability. Systems with two actuators like the one in Figure 1.1 provide straightforward control and easier implementation at a lower cost but are limited in trajectory flexibility as described in the conclusions of the work by Alexander Hasp Frank and Morgan Tjernström [1]. Multi-actuator setups offer better control and precision and hence three actuator system with resistive touchpad-based feedback will be used for this work.

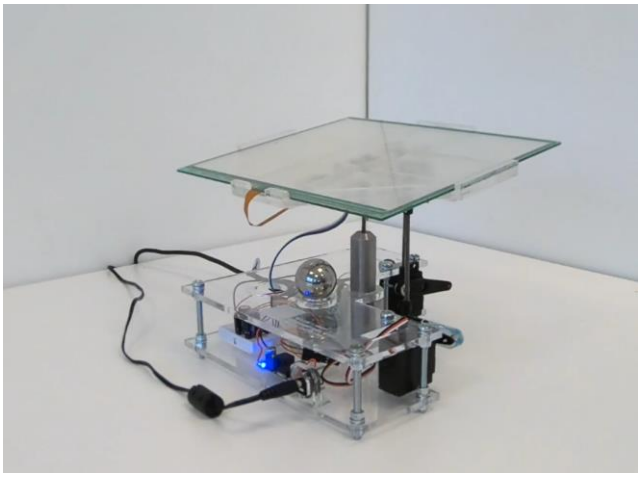

Figure 1.1 Balancer with 2 actuators built at KTH Royal Institute of Technology [1]

Figure 1.2 Ball Balancer with camera-based feedback built at TU Eindhoven [2]

2 Literature Review

Control theory, a subfield of mathematics, plays a pivotal role in understanding and regulating dynamic systems. The goal of a controller is to stabilize an inherently unstable system or to modulate system variables to achieve a desired outcome. Emerging from foundational Mathematical principles, Control theory is deeply intertwined with modern technological advancements. This chapter explores the evolution of Control theory, use of PID controllers and design principles of parallel manipulators.

2.1 The Evolution of Control theory

Control theory is rooted in classical mechanical systems like water clocks[3] and windmills. In the 17th-century, Christiaan Huygens invented the pendulum clock[4]. While his invention of the pendulum clock in 1656 revolutionized timekeeping, his design was not to study feedback mechanisms. In the 18th century, Huygens' centrifugal governor[5] was adopted by James Watt in the steam engine, an early example of feedback control in a mechanical system. In the 19th century, control concepts were beginning to mathematically conceptualize. For instance, a well-known landmark paper is James Clerk Maxwell's 1868 paper "On Governors" [3], which developed the mathematical analysis of feedback control systems.

Only during the first half of the 20th century did modern control theory begin to take shape, impelled by contributions arising especially from electrical engineering, with now standard tools such as Bode plots and Root Locus diagrams. It received a new boost during World War II, spurred on by breakthroughs in missile guidance and radar systems. The rise of more practical control techniques was most associated with the invention of the PID controller which will also be the central topic of this thesis, and the foundational work by Rudolf E. Kálmán, who proposed the Kalman filter [4] and thus revolutionized estimation and control.

Then came state-space methods and optimal control in the second half of the 20th century-the robust control frameworks really signified the turning to complex and nonlinear systems. Digital computation has enabled substantial advances in the areas of adaptive control and nonlinear control, plus the more recent approaches of fuzzy logics and machine-learning-based control systems.

[3] https://en.wikipedia.org/wiki/Water_clock
[4] https://en.wikipedia.org/wiki/Pendulum_clock
[5] https://en.wikipedia.org/wiki/Centrifugal_governor

2.2 PID Controller

A PID controller[6] is used to dynamically regulate the platform's orientation by combining three control terms: proportional, integral, and derivative. The proportional term addresses the current error, the integral term eliminates small steady-state errors, and the derivative term predicts and counters rapid changes in the error for improved stability.

A PID or three term controller is a feedback-based control loop mechanism commonly used to manage machines and processes that require continuous control and automatic adjustment. It is typically used in industrial control systems and various other applications where constant control through modulation is necessary without human intervention. The PID controller automatically compares the desired target value (SP) with the actual value of the system PV). The difference between these two values is called the error value.

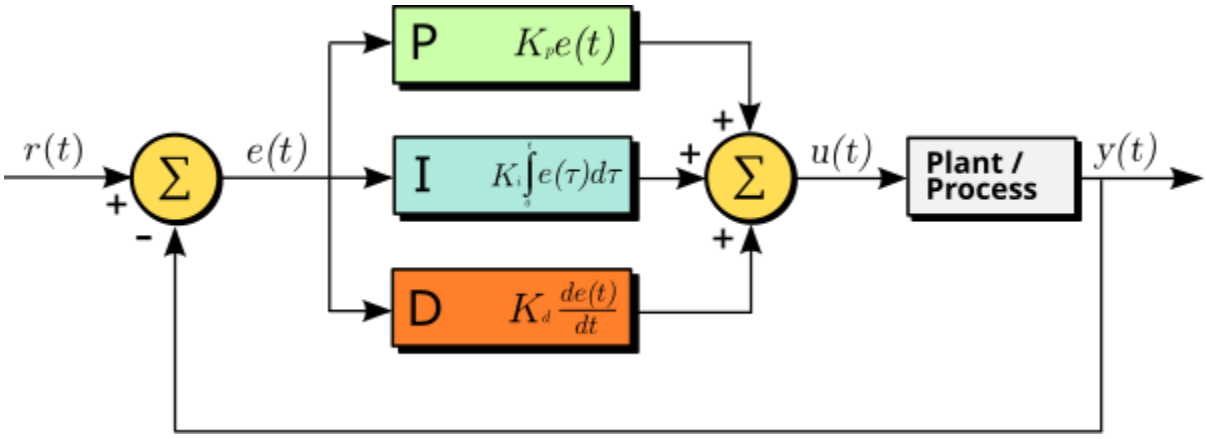

Figure 2.1 A block diagram of a PID controller in a feedback loop. r(t) is the desired process variable (PV) or setpoint (SP), and y(t) is the measured PV.

The overall control function $u(t)$ is described by,

$$u(t) = K_p \times e(t) + K_i \int e(t)dt + K_p \frac{de}{dt} \quad (1)$$

In the case of a ball-balancing platform as described earlier, the proportional gain responds to the distance between the ball's current position and the set point. The derivative controller accounts for the speed at which the ball is moving away from the set point, while the integral term addresses any small steady-state error.

[6] https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller

2.2.1 Proportional Control

The purpose of the proportional controller is to minimize the steady-state error by increasing the system's proportional gain. By introducing a constant K_P, known as the proportional gain, the proportional response is adjusted such that the steady-state error decreases inversely with the proportional gain. The proportional term, or output signal, is defined by equation (5.2).

$$u(p) = K_p e(t) \tag{2}$$

However, if the gain is set too high, it may lead to instability in the system by causing excessive output fluctuations. On the other hand, if the gain is too low, the system may struggle to effectively handle both external and internal disturbances.

2.2.2 Integral Control

The integral controller is proportional to both the duration and magnitude of the error over time. It corrects any accumulated offset in the system by summing the errors over a finite period, effectively eliminating steady-state error. However, the integral gain K_i can sometimes degrade the system's response, potentially causing transient or oscillatory behaviour. The output signal from the integral term is given by equation (5.3).

$$u_i(t) = K_i \int_0^t e(t)\ d(t) \tag{3}$$

2.2.3 Derivative Control

The derivative controller is determined through the calculation of the error's response slope over time. The slope of the error is then multiplied with K_D, the derivative gain. The derivative term is defined in equation (5.4).

$$u_D(t) = K_D \frac{d\ e(t)}{dt} \tag{4}$$

2.3 Parallel Manipulators

A parallel manipulator[7] is a mechanical system that uses several computer-controlled serial chains to support a single platform, or end-effector. A 3-RRS PM is used in this work. Each limb has the same kinematic structure (RRS), and these limbs are attached to the base and moving platform symmetrically. The moving platform has 1-dof translating motion along the vertical axis (z) and 2-dof rotational motion about the horizontal axes (x and y). [5]

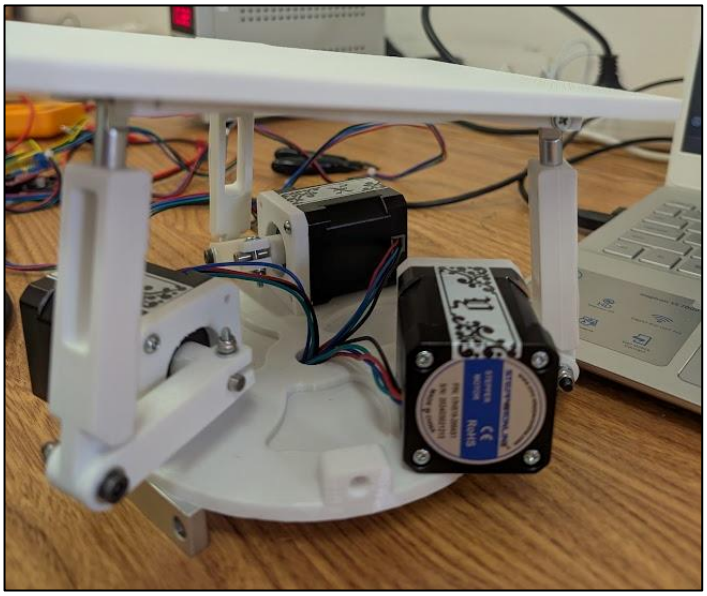

Figure 2.2 3RRS PM platform

2.4 Lagrangian Mechanics

Lagrangian mechanics[8] simplifies complex mechanical systems by using the principle of least action and the Lagrangian (Equation 6) to derive equations of motion (Equation 5). This approach is particularly useful for systems with multiple degrees of freedom.

$$\frac{\partial}{\partial t}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = 0 \tag{5}$$

$$L = E_{kin} - E_{pot} \tag{6}$$

This methodology is applied to derive the equations of motion for a ball-on-platform system, which will be used for Simulink modelling.

[7] https://en.wikipedia.org/wiki/Parallel_manipulator

[8] https://en.wikipedia.org/wiki/Lagrangian_mechanics

3 System Design and Requirements

This chapter outlines the objectives of the project in a systematic order of implementation. It defines the key steps of implementation, the requirements and how the project results will be evaluated.

3.1 Objectives

The objectives of the project are as follows:

1. **Construction:** Fabricate and assemble the platform successfully.
2. **Balancing:** Successfully stabilize a ball on a platform with PID control.
3. **Simulation:** Develop a Simulink model to simulate the system.
4. **Drawing Curves:** Implement control to enable the ball to trace predefined figures on the platform.
5. **Research**: Conduct research of other control strategies that can be employed.

3.2 Performance Metrics

The outcome of the work and the system's success will be measured by the following metrics:

1. **Stability**: The ability of the system to reach steady state.
2. **Steady State Error**: Average offset between the SP and the PV
3. **Reaction time**: The system's response time to input changes or disturbances.
4. **Robustness**: Avoiding overshoot.
5. **Motion Accuracy (only for the path tracing case)**: The deviation between the predefined path and the actual path traced by the ball.

3.3 Assumptions

This section outlines the assumptions made in work. As the Ball Balancing Platform is a demonstration built in a lab environment, the required assumptions are not very rigid. The following assumptions were made:

1. Friction of roll is not considered.
2. The ball that is to be balanced is perfectly spherical and has a smooth surface.
3. The ball does not slide on the platform or moves upwards.
4. The ball in static conditions is unstable even without external disturbances.
5. Vibrations and jerky motion are not considered for simulation.
6. The speed of actuation (the angular velocity) is not taken into consideration for simulation.

3.4 Hardware Requirements

The hardware requirements for this project can be divided into two main categories: **Electronics** and **General Parts**. The electronics section includes all the essential components needed for the system's actuation, control, and power supply.

The general parts section encompasses mechanical components and other peripherals necessary for constructing the physical platform and interfacing with the electronics. Together, these hardware components enable the integration of the mechanical and electronic subsystems, facilitating the realization of the ball-balancing platform.

3.4.1 Electronics

1. Maker Uno Microcontroller
2. Nema 17 59 Ncm Stepper Motors (Bipolar)
3. TMC2209 V2.0 Stepper Motor Drivers
4. CNC shield
5. 30V Bench Power Supply (or any 24V power supply)
6. 5V Regulator

3.4.2 General Parts

1. 8.4" 4 Wire Resistive Touch Panel
2. 1" Steel Bearing Ball
3. 22mm long m3 tie rod
4. M3 x 6mm threaded inserts
5. M3 x 5mm Standoffs
6. M3 x 5mm Screws
7. M3 x 8mm Screws
8. M3 x 10mm Screws
9. M3 x 35mm Screws
10. M3 Nylon Locknuts
11. M4 x 20mm Screws
12. M4 x 25mm Screws
13. M4 Nylon Locknuts

The requirement list is supplemented with items like General mechanical tools, Soldering Iron, Electrical wire, and 3D printer.

General Concepts of the system

This chapter explains the process of stabilizing the ball to achieve the desired outputs. To accomplish this, it is essential to develop three key models:

1. **Control Software**: Defines the logic and algorithms for maintaining balance and achieving desired system behaviours.
2. **Inverse Kinematics**: Describes the mathematical relationship between the platform's movements and the ball's position.
3. **Ball Motion Model**: Simulates the dynamics of the ball's movement, serving as a tool for testing and validating system performance in a virtual environment.

3.5 Control Software

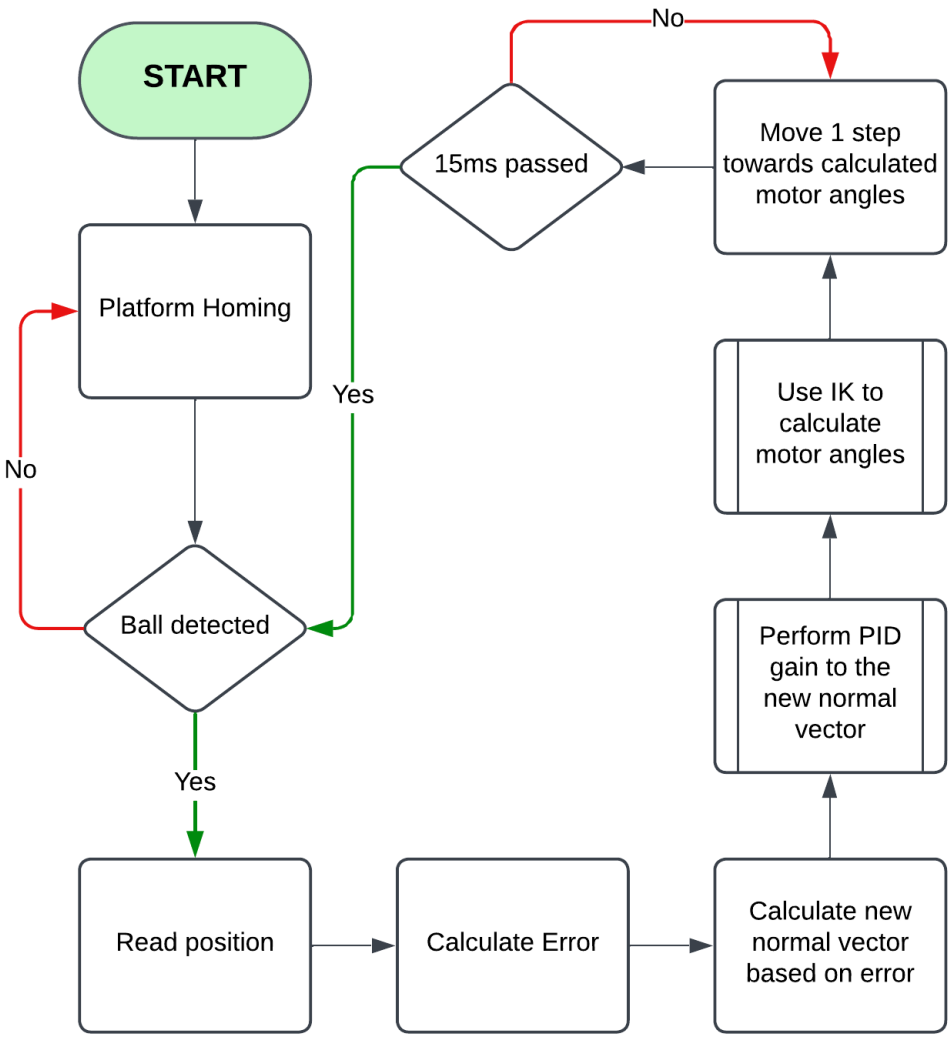

Figure 3.1 Control Algorithm describing the process.

The following is a detailed description of the flowchart shown in Figure 5.1:

1. **Start**: The process begins.
2. **Platform Homing**: The platform undergoes a homing sequence, moving to a predefined position and orientation described by the normal vector [0,0,4.25] to ensure a consistent starting point. When powering on the system for first time it is essential for the motor to be pushed down.
3. **Ball detection:** The system checks whether a ball is present on the platform. Depending on the outcome the following happens:
    - If no ball is detected, the system returns to the 2nd step.
    - If detected, the process continues to the next step.
4. **Read Position**: The current position of the ball is identified by reading the touchpad measurements.
5. **Calculate Error**: The error is calculated, which is the difference between the ball's current position and the Set point of the platform. This error indicates how far the ball is from the set position in [x, y] vector format.
6. **Calculate New Normal Vector**: Based on the calculated error, a new normal vector is computed by subtracting the Error from the normal vector describing the platform's current position. This new vector represents the required orientation to move the ball toward the centre.
7. **Perform PID Gain to the New Normal Vector**: A PID controller is implemented for each axis [X and Y]. The output of PID controller is constrained between the range [-0.25,0.25] to prevent the platform from over-tilting.
8. **Use IK to Calculate Motor Angles**: Pre-defined Inverse Kinematics (IK) equations is utilized to translate the desired platform tilt, represented by the normal vector, into specific angles for each motor that controls the platform.
9. **Move 1 Step Towards Calculated Motor Angles**: The motors are incrementally adjusted with a non-blocking function, moving towards the calculated angles. This gradual movement ensures smooth control and prevents abrupt shifts that might destabilize the ball.
10. **15ms wait:** The system introduces a wait time of 15 milliseconds for the steppers to reach the desired angle. The code moves to next step even if the required motor angles are not reached after 15ms.

11. **Loop**: If the ball is detected by the touchpad the loop continues, if ball is not detected the platform returns to home.

In essence, the system starts by homing the platform, then continuously detects the ball's position and calculates the necessary platform tilt using a PID controller and inverse kinematics. Finally, it moves the platform's motors incrementally to adjust its position, maintaining the ball's balance, looping back until no longer detected.

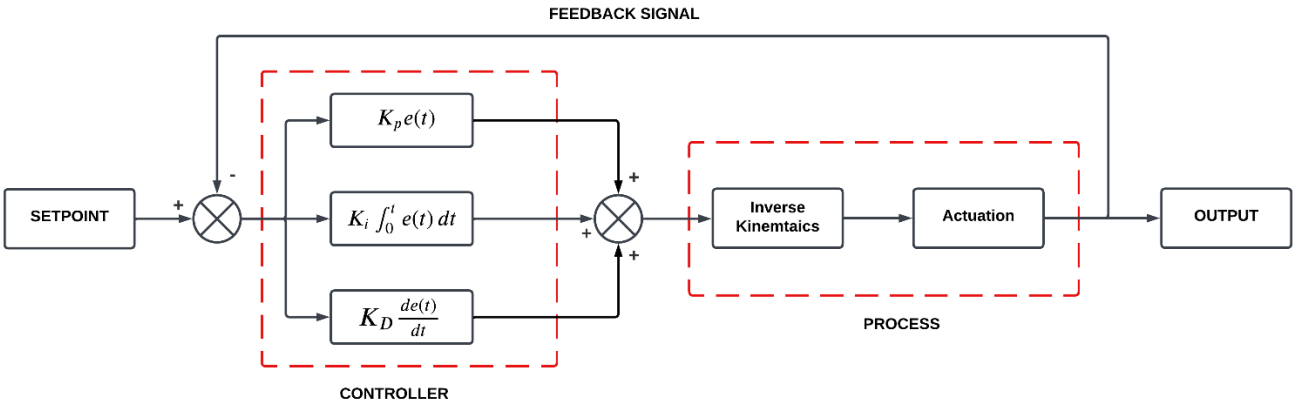

Figure 3.2 Control System block diagram for a single axis

The control system block diagram in Figure 3.2 represents the architecture of a ball-balancing system which was described earlier, where a PID controller is used to manage the movement of the ball along a single axis on a platform. Two such controllers are implemented one for each axis X and Y. The system receives setpoints for the desired positions of the ball in the X and Y directions, which are compared with the actual positions obtained through feedback. The SP can be constant or a changing value. By changing the SP in a logical manner, it is possible to trace trajectories of different shapes which is discussed in detail in section 5.5.

The errors in both directions are calculated using summation blocks, where the error is the difference between the setpoint and the actual position. These errors serve as inputs to separate PID controllers for the X and Y axes. Each PID controller generates control signals based on the proportional, integral, and derivative terms: the proportional term responds to the current error, the integral term addresses accumulated past errors, and the derivative term anticipates future errors based on the rate of change as described in section 2.2. The control signals are then fed into an inverse kinematics block, which translates them into appropriate motor commands for the stepper motors. The stepper motors actuate the platform, adjusting its tilt to correct the ball's position, thereby minimizing the error. This feedback loop ensures continuous adjustment of the platform's position to maintain the desired trajectory and balance of the ball.

3.6 Inverse Kinematics of the system

In this sub-chapter, the inverse kinematics equations required are described to orient the platform in such a way that it sends the ball towards setpoint. The platform is supported by three legs, each modelled as a two-link manipulator connecting the base to the platform. The configuration of each leg consists of two revolute joints and two rigid links, forming a planar structure. Chapter 3 of the work by Tetik and Halil [5] provides an excellent explanation of modelling the manipulator and formulating the equations.

Given a unit normal vector Z' of the platform plane, the IK equations then compute the required angles of the revolute joint B_1 , B_2 *and* B_3 [6].

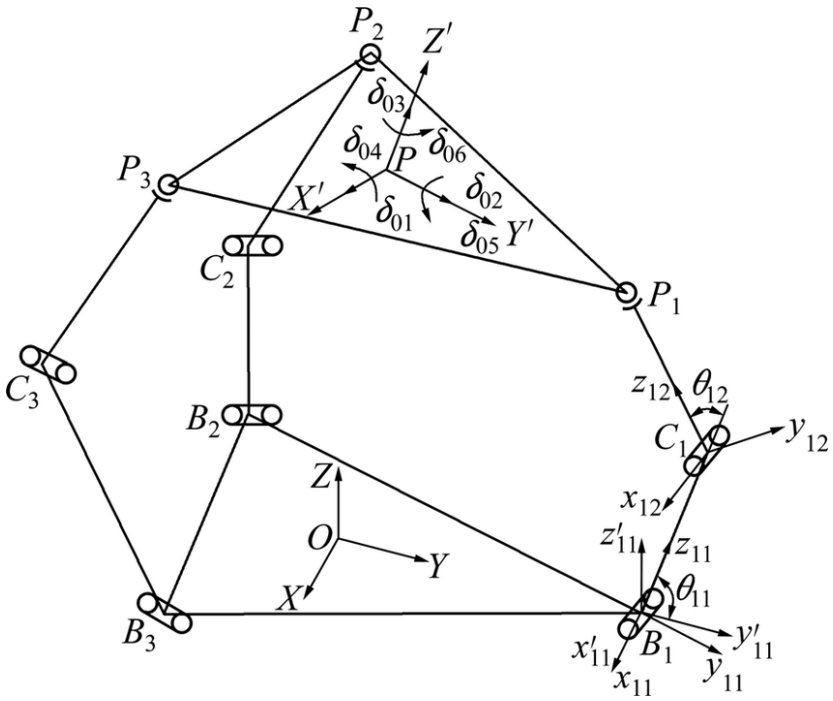

Figure 3.3 Kinematic sketch of 3RRP parallel manipulator [6]

Table 3.1 Description of symbols in Figure 4.2

SYMBOL	DESCRIPTION
O	Origin of the global co-ordinate system
$B_1, B_2, B_3, C_1, C_2, C_3$	Revolute Joints
P_1, P_2, P_3	Spherical Joints
P	Platform Plane origin
Z'	Unit normal vector of the platform plane

The system is divided into three separate kinematic chains. The origin for the global frame (O) is marked at the mid-point of the base plane. The mid-point of the platform plane is represented by O_1.The required rotor angles is calculated as a function of tilt of the platform plane represented by $\vec{N}$ and the height between the platforms represented by h.

$$[\,\theta_1, \theta_2, \theta_3] = f(\vec{N}\,, h\,) \tag{7}$$

The following equations describe the Inverse Kinematics for all three chains:

Unit normal Vector:

$$n_{mag} = \sqrt{n_x^2 + n_y^2 + 1} \tag{8}$$

$$n_x = \frac{n_x}{n_{mag}}, \quad n_y = \frac{n_y}{n_{mag}}, \quad n_z = \frac{1}{n_{mag}}$$

Equations for $\boldsymbol{\theta_1}$:

$$\boldsymbol{\theta_1} = \arccos\left(\frac{y}{mag}\right) + \arccos\left(\frac{mag^2 + f^2 - g^2}{2 \cdot mag \cdot f}\right) \tag{9}$$

Where:

$$y = d + \frac{e}{2}\left(1 - \frac{n_x^2 + 3n_z^2 + 3n_z}{n_z + 1 - n_x^2 + \frac{n_x^4 - 3n_x^2 n_y^2}{(n_z + 1)^2}}\right)$$

$$z = h_z + e.n_y$$

$$mag = \sqrt{y^2 + z^2}$$

Equations for $\boldsymbol{\theta_2}$:

$$\boldsymbol{\theta_2} = \arccos\left(\frac{\sqrt{3}x + y}{-2 \cdot mag}\right) + \arccos\left(\frac{mag^2 + f^2 - g^2}{2 \cdot mag \cdot f}\right) \tag{10}$$

Where:

$$x = \frac{\sqrt{3}}{2}\left(e\left(1 - \frac{n_x^2 + \sqrt{3}n_x n_y}{n_z + 1}\right) - d\right)$$

$$y = \frac{x}{\sqrt{3}}$$

$$z = h_z - \frac{e}{2}(\sqrt{3}n_x + n_y)$$

$$mag = \sqrt{x^2 + y^2 + z^2}$$

Equations for $\boldsymbol{\theta_3}$:

$$\boldsymbol{\theta_3} = \arccos\left(\frac{\sqrt{3}x - y}{2 \cdot mag}\right) + \arccos\left(\frac{mag^2 + f^2 - g^2}{2 \cdot mag \cdot f}\right) \tag{11}$$

Where:

$$x = \frac{\sqrt{3}}{2}\left(d - e\left(1 - \frac{n_x^2 - \sqrt{3}n_x n_y}{n_z + 1}\right)\right)$$

$$y = \frac{-x}{\sqrt{3}}$$

$$z = h_z + \frac{e}{2}(\sqrt{3}n_x - n_y)$$

$$mag = \sqrt{x^2 + y^2 + z^2}$$

Table 3.2 Inverse Kinematics parameters

Symbol	Description	Value	Unit
$\boldsymbol{d}$	Distance from centre of base to corner	5.08	[cm]
$\boldsymbol{e}$	Distance from centre of platform to corner	7.93	$[cm]$
$\boldsymbol{f}$	Length of Link 1	4.445	[cm]
$\boldsymbol{g}$	Length of Link 2	9.32	[cm]
$\boldsymbol{h_z}$	Height of the platform plane	variable	$[cm]$
$\boldsymbol{n_{mag}}$	Magnitude of Normal vector	variable	[-]
$\mathbf{n}$	Normal Vector	variable	[-]

3.7 Equations describing the motion of the ball.

To simulate the movement of the ball, a model describing the equations of motion of the ball is necessary. The three-dimensional ball-on-platform system will be split into two separate two-dimensional ball-on-beam systems, as shown in Figure 3.4 These systems will share the same equations but with different variables. Index 1 will refer to the system observed in the x-z plane of the room, while index 2 will correspond to the system viewed in the y-z plane. The equations will be derived for the first system and later translated into their equivalent form for the second system.

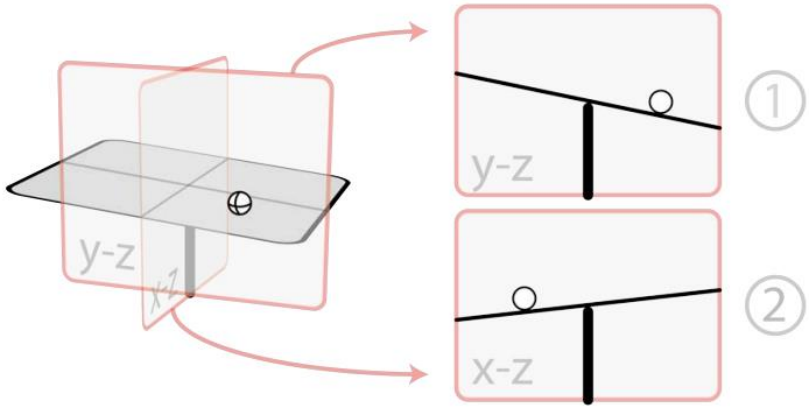

Figure 3.4 Three-dimensional system represented as a pair of two-dimensional system [1]

The goal is to model the motion of the ball in XY plane given the tilt angles of the YZ and XZ planes. The motion is influenced by the Forces acting on the ball illustrated in Figure 3.5.

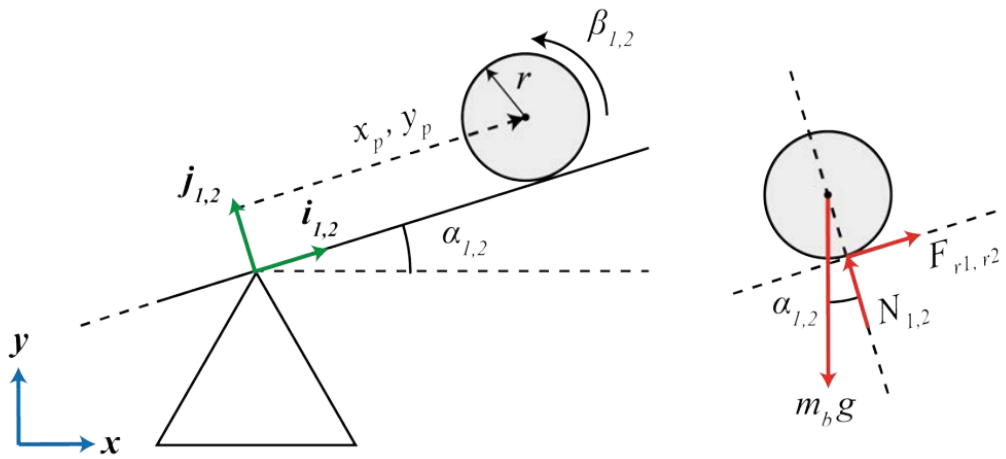

Figure 3.5 Free body diagram of the ball in two dimension [1]

Lagrangian Method: Equations of Motion

The Lagrangian Method derives the equation of motion through the relation of kinetic and potential energy of the system. This method is particularly useful for complex systems with multiple degrees of freedom. Below are the detailed derivations and descriptions. A similar approach as M. T. Alexander Hasp Frank is used to derive the equations [1].

Lagrangian Equation:

$$\frac{\partial}{\partial t}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = 0 \tag{12}$$

L (Lagrangian) is the difference between the kinetic and potential energy of the system:

$$L = E_{kin} - E_{pot} \tag{13}$$

Kinetic Energy:

$$E_{kin} = E_{kin,T} + E_{kin,R} = \frac{1}{2} m_b v_b^2 + \frac{1}{2} J_b \omega_b^2 \tag{14}$$

Where:

$$E_{kin,T} = \frac{1}{2} m_b v_b^2 = \frac{1}{2} m_b \left(\dot{x_b^2} + \dot{y_b^2}\right)$$

$$E_{kin,R} = \frac{1}{2} J_b \omega_b^2 = \frac{1}{2} J_b \frac{v_b^2}{r_b^2} = \frac{1}{2} J_b \frac{\left(\dot{x_b^2} + \dot{y_b^2}\right)}{r_b^2}$$

Thus, the total kinetic energy becomes:

$$E_{kin} = \frac{1}{2}\left(m_b + \frac{J_b}{r_b^2}\right)\left(\dot{x_b^2} + \dot{y_b^2}\right)$$

Potential Energy:

$$E_{pot} = m_b g h_b = -m_b g x_b \sin(\alpha) - m_b g y_b \sin(\beta)$$

Lagrangian:

$$L = E_{kin} - E_{pot} = \frac{1}{2}\left(m_b + \frac{J_b}{r_b^2}\right)\left(\dot{x_b^2} + \dot{y_b^2}\right) + m_b g x_b \sin(\alpha) + m_b g y_b \sin(\beta)$$

Equation of Motion (x-direction):

$$\frac{\partial}{\partial t}\left(\frac{\partial L}{\partial \dot{x_b}}\right) = \frac{\partial}{\partial t}\left(\left(m_b + \frac{J_b}{r_b^2}\right)\dot{x_b}\right) = \left(m_b + \frac{J_b}{r_b^2}\right)\ddot{x_b}$$

$$\frac{\partial L}{\partial x_b} = m_b g \sin(\alpha)$$

Substituting these into Equation 4.1, the differential equation becomes:

$$\left(m_b + \frac{J_b}{r_b^2}\right)\ddot{x_b} - m_b g \sin(\alpha) = 0$$

Simplifying for $\ddot{x_b}$, the equation of motion is derived as:

$$\ddot{\boldsymbol{x_b}} = \frac{\boldsymbol{m_b g r_b^2}}{\boldsymbol{m_b r_b^2 + J_b}}\boldsymbol{\sin(\alpha)} \tag{15}$$

Equation of Motion (y-direction):

Applying the same method to the y-direction, the equation of motion is:

$$\ddot{\boldsymbol{y_b}} = \frac{\boldsymbol{m_b g r_b^2}}{\boldsymbol{m_b r_b^2 + J_b}}\boldsymbol{\sin(\beta)} \tag{16}$$

α *and* β angles will be calculated from the normal vectors using the formula:

$$\alpha = \operatorname{atan}\left(\frac{n_x}{4.25}\right) \tag{17}$$

$$\beta = \operatorname{atan}\left(\frac{n_y}{4.25}\right) \tag{18}$$

Table 3.3 Ball motion model parameters

Symbol	Description	Value	Unit
$\boldsymbol{m_b}$	Mass of the ball	0.067	[kg]
$\boldsymbol{J_b}$	Rotational inertia of the ball	0.0000435	$[kg/m^2]$
$\boldsymbol{r_b}$	Radius of the ball	0.0125	[m]
g	Acceleration due to gravity	9.81	$[m/s^2]$
$\boldsymbol{\alpha, \beta}$	Tilt angles of XZ and YZ planes	variable	[radian]
$\mathbf{x_b}, \boldsymbol{y_b}$	Position coordinates of the ball in XY plane	variable	[m]

4 Hardware Implementation

4.1 Mechanical Configuration

This sub-chapter details the mechanical design of the 3RRS parallel manipulator platform. The 3RRS platform has **3 DOF and 3 points of actuation**. Three stepper motors with TMC2209 drives are used for precise actuation.

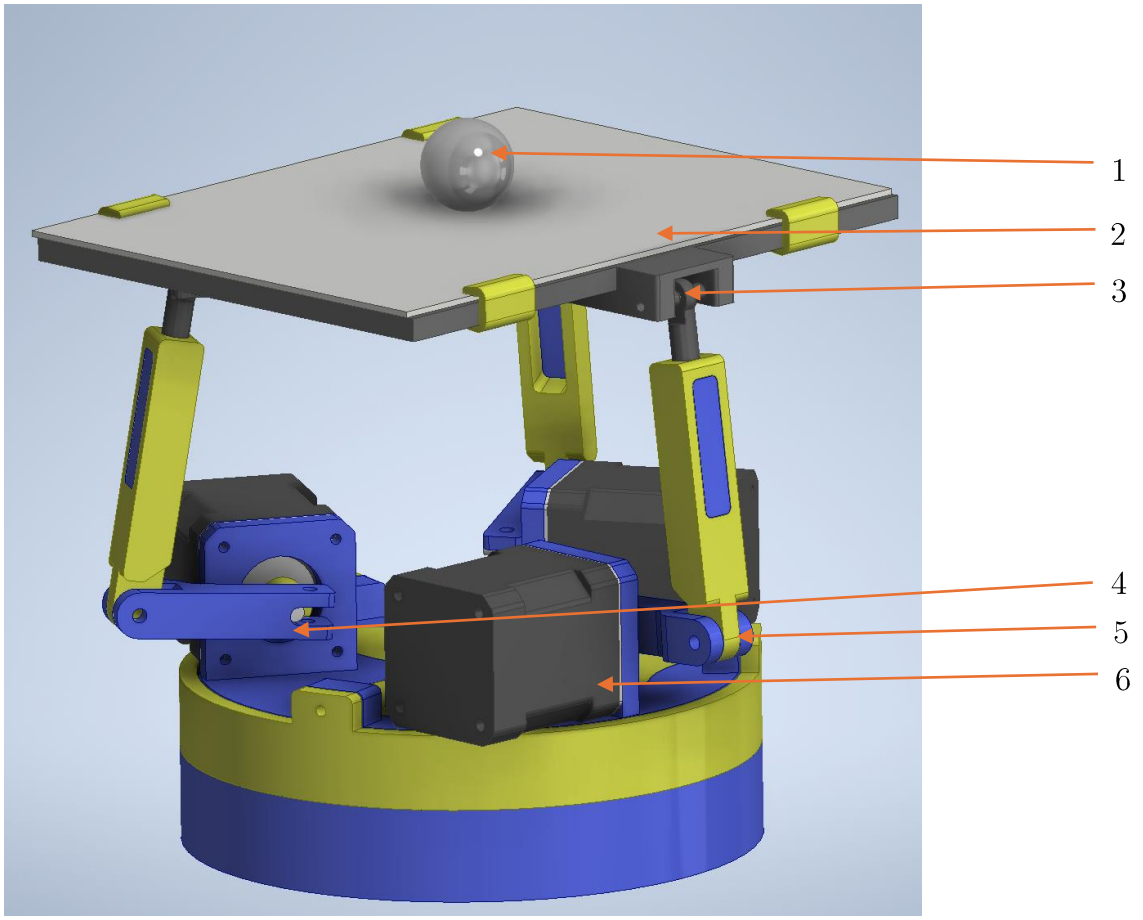

Figure 4.1 3RRS Ball Balancer mechanical design by Aaed Musa

A metallic ball **1** is placed on a resistive touchpad **2** which outputs the position of the ball by detecting the contact point and returning it as a cartesian point. The platform is connected to the actuator legs with a spherical joint **3.** The actuator leg has two links connected by a revolute joint **5.** The lower leg is connected to the rotor of the stepper motor **6** with a revolute joint **4.** The design of the platform is credited to **Aaed Musa**[9].

[9] https://www.aaedmusa.com/

4.2 Electrical Configuration

This chapter outlines the electrical configuration of the project, including the Maker UNO, TMC2209 stepper driver, CNC shield, stepper motor, and resistive touchpad. A NEMA 17 stepper motor provides accurate movement, with manual homing at power-up. A resistive touchpad is employed to track the ball's position, offering a simple and reliable solution.

4.2.1 Maker UNO

Maker UNO is a similar microcontroller to Arduino UNO. It has an identical pinout configuration as compared to Arduino UNO. The ease of use, versatile connectivity with ample IO ports and analogue ports, native programming IDE and real time control with low latency make it an ideal choice.

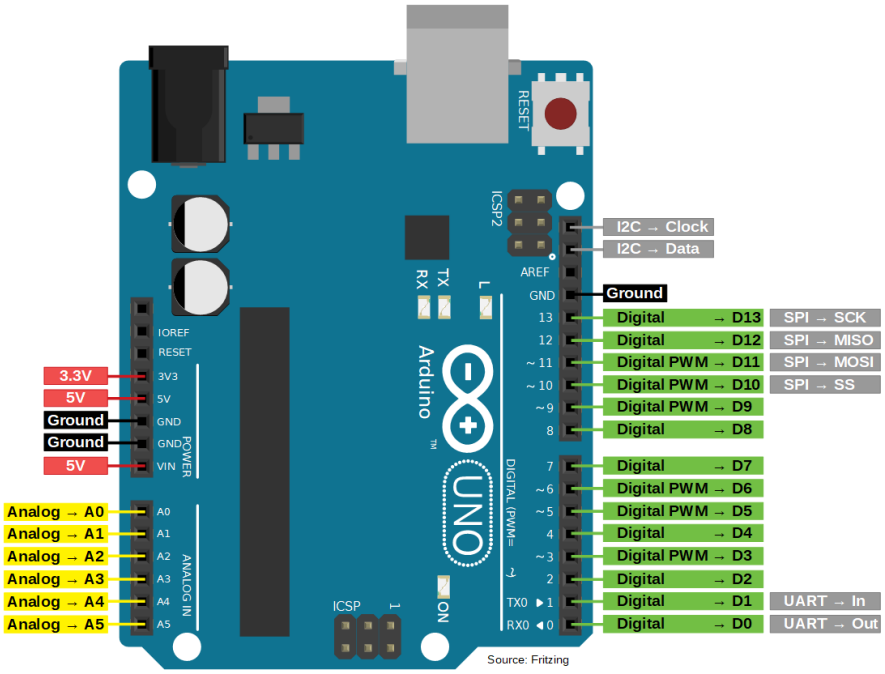

Figure 4.2 Arduino UNO pinout diagram. Maker UNO has an identical configuration.

Table 4.1 Maker UNO important technical specification.

Category	Specification
Microcontroller	ATmega328P
Operating Voltage	5V
Digital I/O Pins	14
Analog Input Pins	6
Flash Memory	32 KB (ATmega328P)
Clock Speed	16 MHz

4.2.2 TMC2209 V2.0

The TMC2209 is known for its silent operation. Low noise levels, different micro stepping options, current control and low heat generation makes it an ideal choice to control the stepper motors. The V_{ref} is set using a potentiometer screw.

$$V_{ref} = I_{motor} \times 8 \times R_{sense} \tag{19}$$

From the stepper documentation,

$$I_{motor} = 1.5\ A$$
$$R_{sense} = 0.11\ \Omega$$
$$V_{ref} = 1.5A \times 0.11\ \Omega \times 8$$

$$\boldsymbol{V_{ref} = 1.32\ V}$$

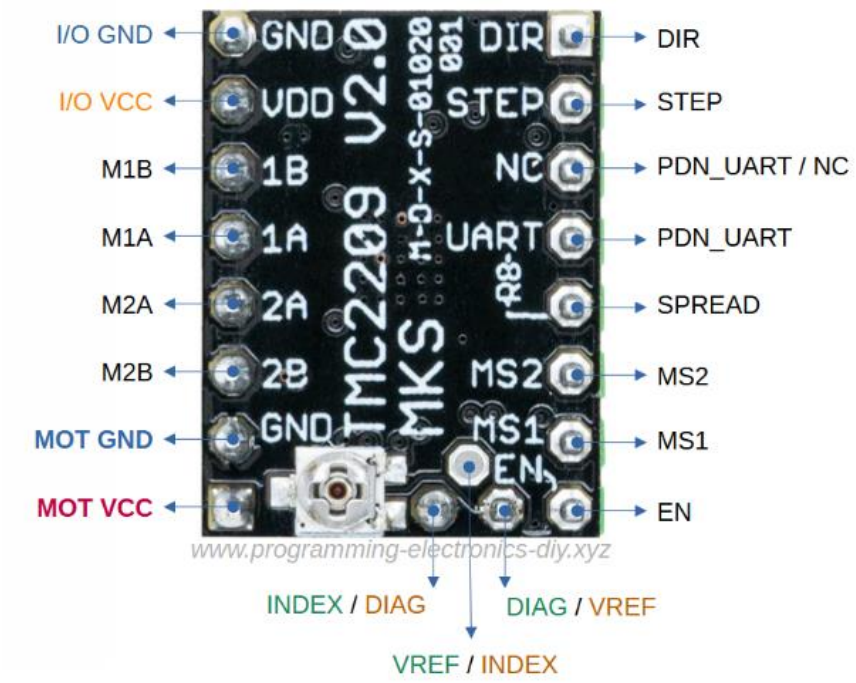

Figure 4.3 TMC2209 V2.0 stepper driver pinout diagram.

Table 4.2 TMC2209 Micro stepping Configuration.

MS1	MS2	Micro stepping Mode
Low	Low	Full Step (1x)
High	Low	Half Step (2x)
Low	High	Quarter Step (4x)
High	High	Sixteenth Step (16x)

Table 4.3 TMC2209 technical specifications.

Specification	Value
Operating Voltage (VM)	4.75V to 29V DC
Output Current	2.0A RMS, 2.4A Peak (per phase)
Operating Temperature	-40°C to +150°C

4.2.3 CNC Shield

A CNC shield will be used instead of manually connecting the jumpers through a breadboard. The CNC shield has built in decoupling capacitors and integrates with Maker UNO seamlessly. Addition of CNC shield makes electrical management easier.

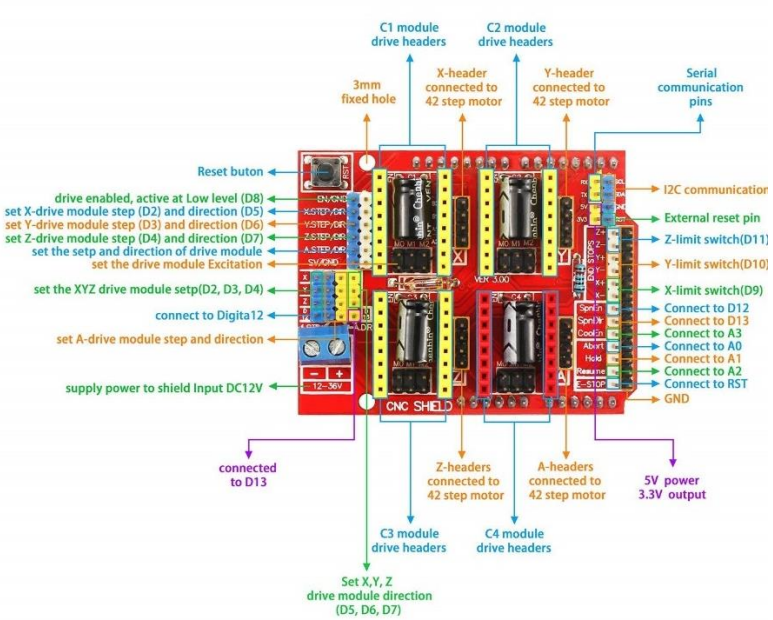

Figure 4.4 CNC shield pinout diagram

Table 4.4 Cross-referencing of Arduino pins

UNO Pin	CNC Shield Pin	Function
Pin 2	X Step	Step signal for X-axis motor
Pin 3	Y Step	Step signal for Y-axis motor
Pin 4	Z Step	Step signal for Z-axis motor
Pin 5	X Dir	Direction signal for X-axis motor
Pin 6	Y Dir	Direction signal for Y-axis motor
Pin 7	Z Dir	Direction signal for Z-axis motor
Pin 8	Enable	Enable signal for X-axis motor

In addition to the digital pins, analogue pins are also used. The resistive touchpad is connected to A0, A1, A2 and A3 pins shown in the schematics.

4.2.4 Stepper Motor

NEMA 17 bi-polar stepper motors from STEPPERONLINE were used for the project. Cost-efficient, precise control and low noise make it an ideal choice. Positional encoders are not used. Instead, homing is done by manually pushing the platform down. When the steppers are powered off the platform automatically falls to the home position.

Figure 4.5 Nema 17 Bi-polar stepper motor.

Table 4.5 Nema 17 technical specifications

Property	Value
Rated Current (A)	1.5
Step Angle (deg.)	1.8
Bipolar/Unipolar	Bipolar
Holding Torque (Ncm)	42

4.2.5 Resistive touchpad[10]

A four-wire resistive touchpad works by detecting touch through two flexible, transparent resistive layers that come into contact when pressure is applied. Each layer has electrodes along its edges: the top layer for the X-axis and the bottom layer for the Y-axis. To determine the touch point, a voltage is first applied across one layer (e.g., X-axis), and the resulting voltage at the contact point is measured to find the horizontal position. Then, a voltage is applied across the other layer (Y-axis) to measure the vertical position. By alternating these measurements, the touchpad calculates the X and Y coordinates of the touch point using the voltage divider principle.

[10] https://www.sparkfun.com/datasheets/LCD/HOW%20DOES%20IT%20WORK.pdf

4.3 Construction and Assembly

The construction of the ball balancer was the most exciting part of the work. The process began with purchasing of the listed components in the section 3.4. Followed by 3D printing of the CAD models. PETG filament was used for printing. It took about 6 days of continuous printing to make all the parts.

4.3.1 Fabrication of parts

3D printing is one of the most versatile ways to prototype and build nonstandard parts. All the parts with some exceptions like the touchpad and the joints were fabricated using FDM printing. Standard slicer settings with raft and higher infill density were used. It is important to take into consideration the shrinkage of parts and adjust the tolerances accordingly beforehand.

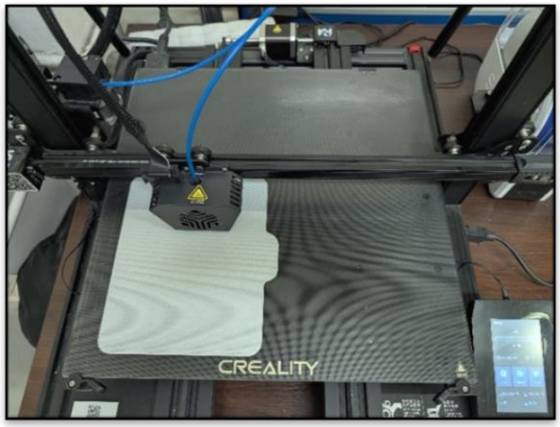

Figure 4.6 Picture taken during printing the top platform

Figure 4.7 Newly printed parts with raft

4.3.2 Electronics Assembly

MakerUNO is the main control unit. The CNC shield is compatible with an Arduino UNO and is stacked on top of it. This eliminates the need to use protoboard and jumper wires which makes the electronics highly organized and easy to handle.

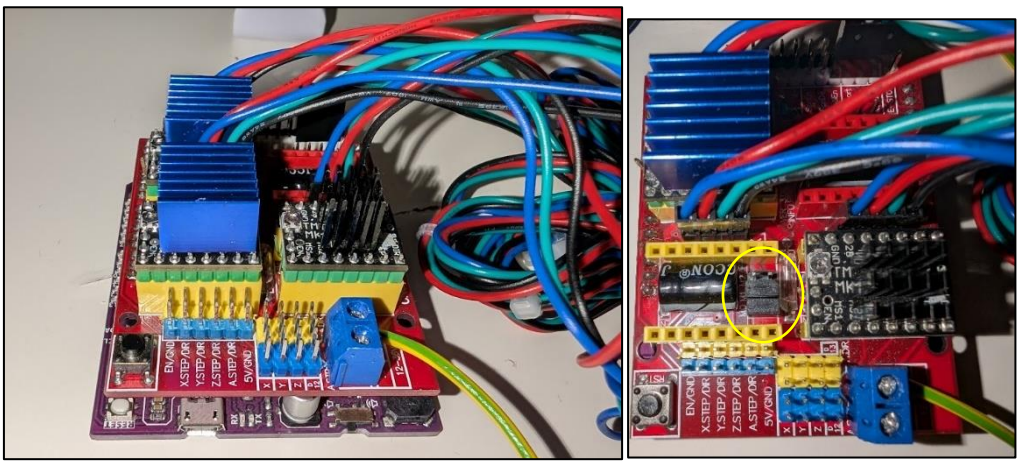

Figure 4.8 CNC shield with drivers connected to UNO (L) Micro stepping jumper pins ($1/16^{th}$ configuration) (R)

The CNC shield is powered with a 24V bench power supply. It is important to set the right micro stepping configuration ($1/16^{th}$) for a smooth motion. Heat sinks are also necessary as the drivers tend to overheat quickly.

4.3.3 Mechanical Assembly

The assembly of the platform was done in the following sequence.

1. Spacers were added to each stepper motor rotor shaft to ensure proper alignment of legs during assembly
2. Each stepper motor was secured onto the base plate using M3 x 10mm screws (x12).
3. The Maker UNO with CNC shield was screwed to stand using M3 x 5mm screws (x4).
4. Each link1 was attached to its corresponding stepper motor using an M4 x 20mm screw and an M4 locknut.
5. A tie rod was attached to one end of each link2 using an M3 x 8mm screw.
6. Each link2 was then connected to the platform frame at the tie rod end using an M3 x 35mm screw. M3 x 5mm standoffs (x2) were used on either side of each tie rod to fill the gap.

7. The other end of each link2 was fastened to the corresponding link1 using an M4 x 25mm screw and an M4 locknut. During this step, it was ensured that Stepper A was connected to the designated side of the platform frame.
8. The base plate was mounted onto the base stand using M3 x 8mm screws (x3).
9. Finally, the resistive touchpad was clipped onto the platform frame using four retainer clips.

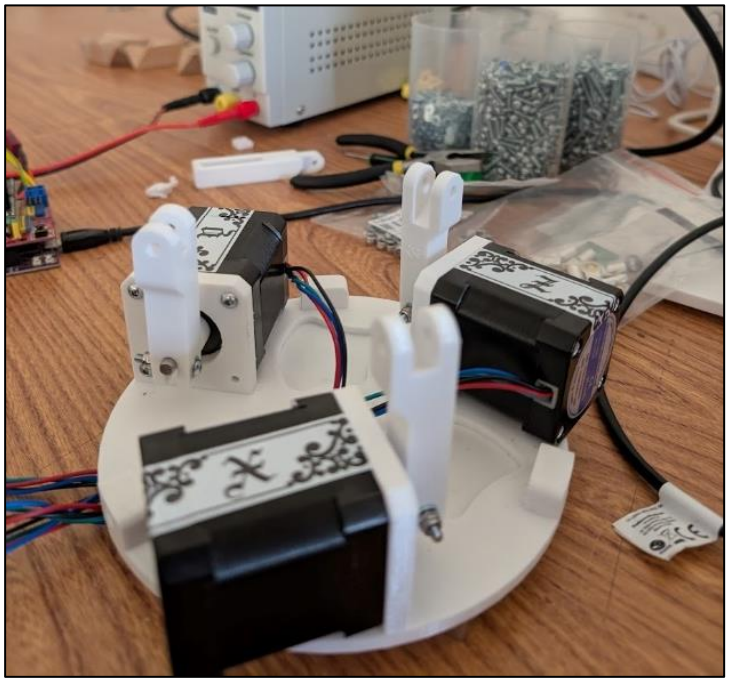

Figure 4.9 Partially assembled platform

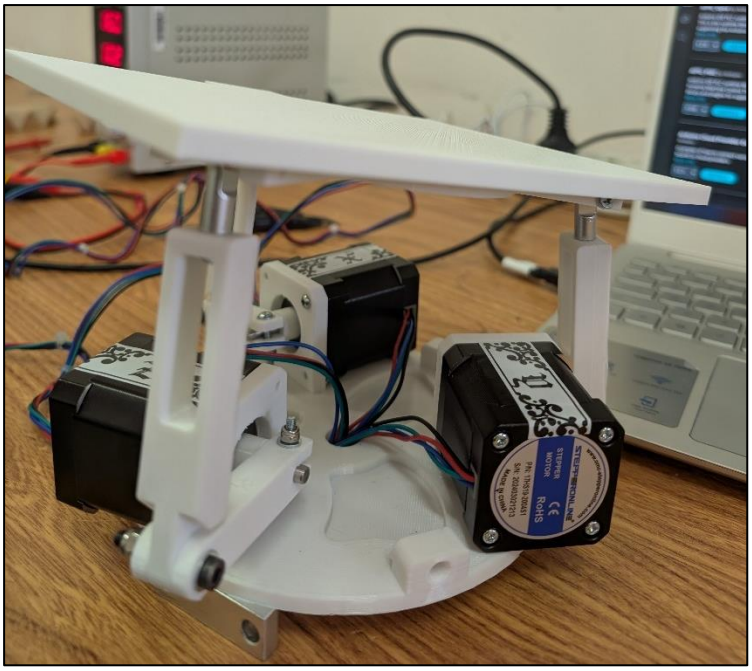

Figure 4.10 Assembled 3RRS PM

5 Software Implementation

This chapter describes the Simulink Model of the system in the first section followed by important function blocks of the controller in the following sections.

5.1 Simulink Modelling

The following Simulink models describe the system that is intended to be simulated. The Ball Motion model is based on section 3.7. The simulations were run using the ODE45 solver. System Simulation Techniques with MATLAB and Simulink [7] provided valuable guidance in modelling the system.

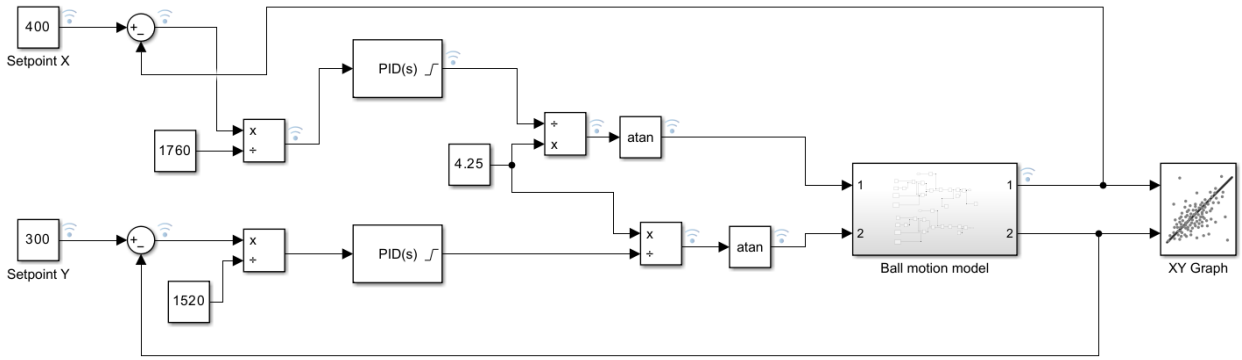

Figure 5.1 Simulink Model of the system

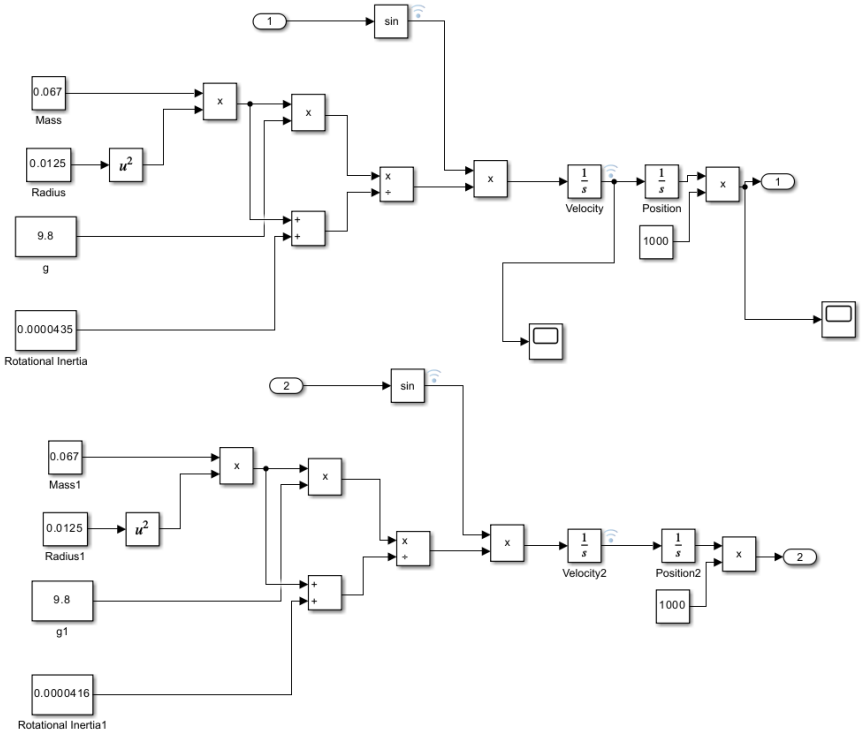

Figure 5.2 Simulink Model of Ball Motion Block

5.2 Position feedback from touchpad.

The position data is acquired from an analogue signal from the resistive touchpad. The signal conversion is performed using the Adafruit Touchscreen library[11]. The signal from the touchpad contains noise but of a minimal amount. Hence filtering is not performed. However, an initial manual calibration is required.

5.3 Calculating normal vectors

Once the error is calculated it is necessary to determine the new normal vector required to move the ball to SP. But the platform can only tilt 0.25 units, anything more than that would destabilize the system as it becomes impossible to slow down the ball and hence it is necessary to scale the error to a value in the range [-0.25,0.25].

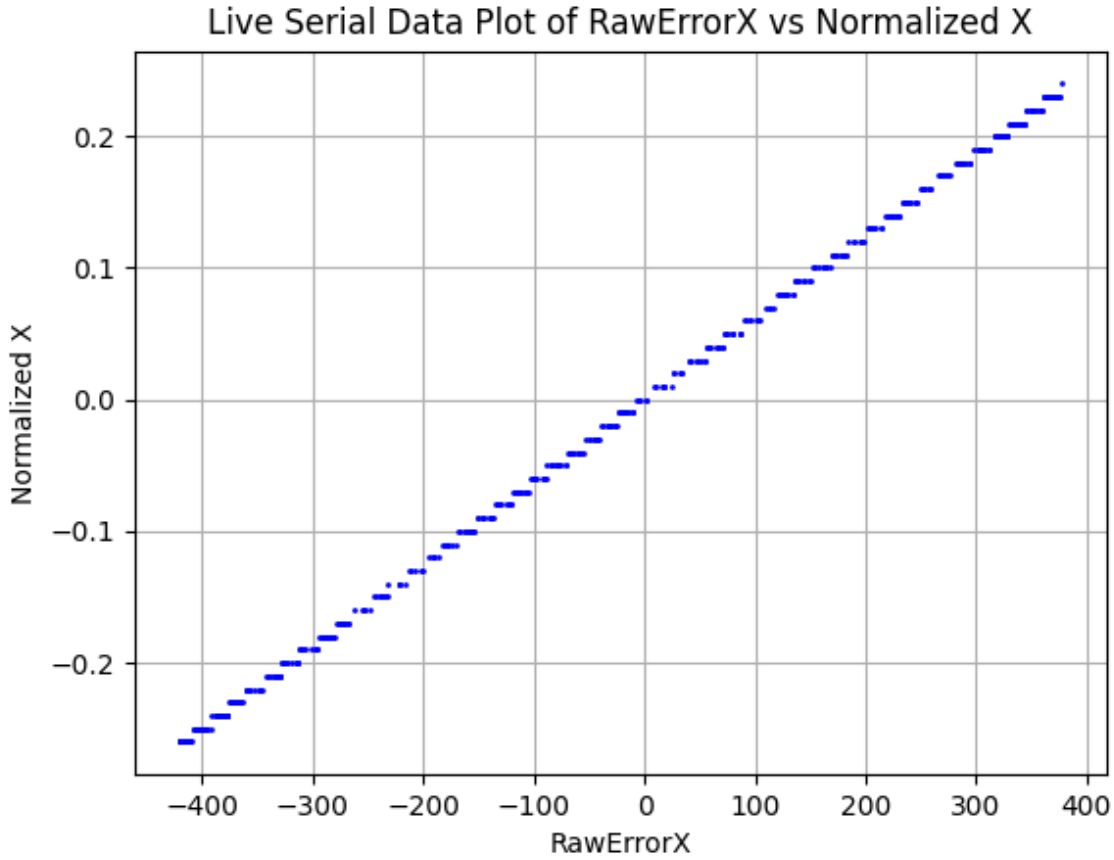

Figure 5.3 Scaling X error values to be in the range [-0.25,0.25]

[11] https://github.com/adafruit/Adafruit_TouchScreen

5.4 PID Tuning

As mentioned in section 3.5 two PID controllers will be used, one for each axis [X and Y]. As the constraints are the same for both the dimensions the same PID gain values will be used for both the controllers. PID tuning for the real system is done manually using the following approach:

1. Values from Simulation offer a good starting point.
2. Obtain oscillatory behavior by reducing the K_D or by increasing the K_P.
3. Tune the values until stability is achieved.
4. Introduce K_I to fix the steady state error.

For both real and simulated systema PID Output Saturation of [-0.25, 0.25] is applied. Integral windup is not applied.

5.5 Creating Trajectories

Once stability is achieved for a single SP, different trajectories can be traced by continuously changing the SP. Initially, discrete signals with defined vertices are used followed by continuous signals to trace complex curves. The equations used for generating a continuous signal is described below.

5.5.1 Circle

By incrementing the angle within range $[0,2\pi]$ at a constant rate and applying the following equations (Eq. 21 and 22) a circular reference is created.

$$X = radius \times \cos(angle) \tag{20}$$

$$Y = radius \times \sin(angle) \tag{21}$$

5.5.2 Infinity symbol

By incrementing the angle within range $[0,2\pi]$ at a constant rate and applying the following equations (Eq. 22 and 23) a reference infinity curve is created.

$$X = amplitude \times \sin(angle) \tag{22}$$

$$Y = amplitude \times \sin(angle) \times \cos(angle) \tag{23}$$

6 Results and Analysis

This chapter discusses the results from simulated and real models. The real model analysis is conducted by sending data via serial port and using python to receive the data and plot it using the Pyplot library.

It is also worth mentioning that there was a slight reduction in the performance of the system when broadcasting the data via serial port. This might be due to the increased latency because of processor time spent on serial communication. Hence, the actual performance of the system is slightly better than the data used for analysis in this section.

6.1 Assembled platform

6.2 PID Gains

Initially the PID values are tuned in Simulink model. The PID tuner functionality of Simulink did not yield reasonable values and hence the parameters were obtained by trial-and-error method. After obtaining the theoretical values, using the approach described in section 5.4 PID gains for the real model were identified.

Table 6.1 PID Gain Values

System	K_P	K_I	K_D
Simulink	0.1970	0.00075	0.1950
Real	0.1670	0.00085	0.1700

6.3 Step Response testing

Step response of the system is tested by placing the ball at the coordinates [400,400] and setting the reference point to [0,0]. By analyzing the transient response of the system, we can quantify the system's behavior to measure the performance metrics outlined in section 3.2 .

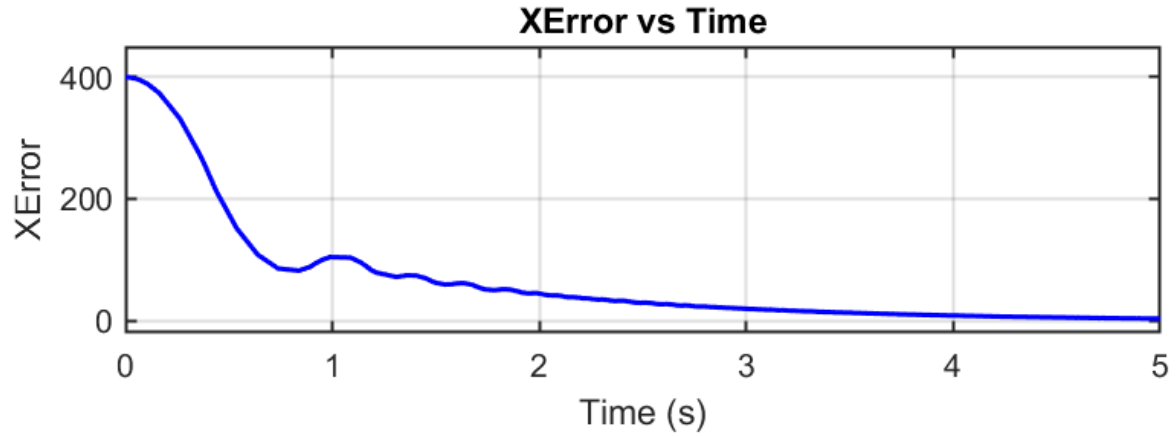

Figure 6.1 X Axis step response of the simulated system

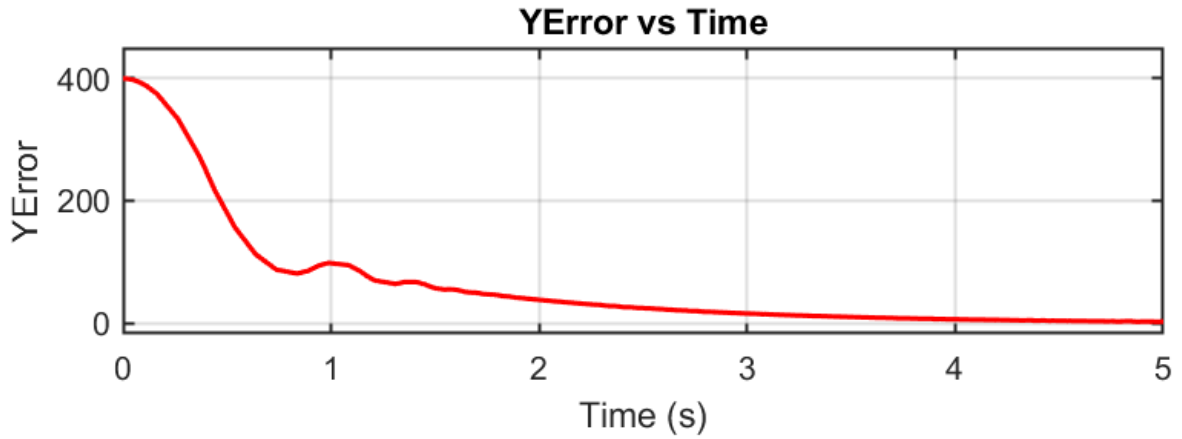

Figure 6.2 Y Axis step response of the simulated system

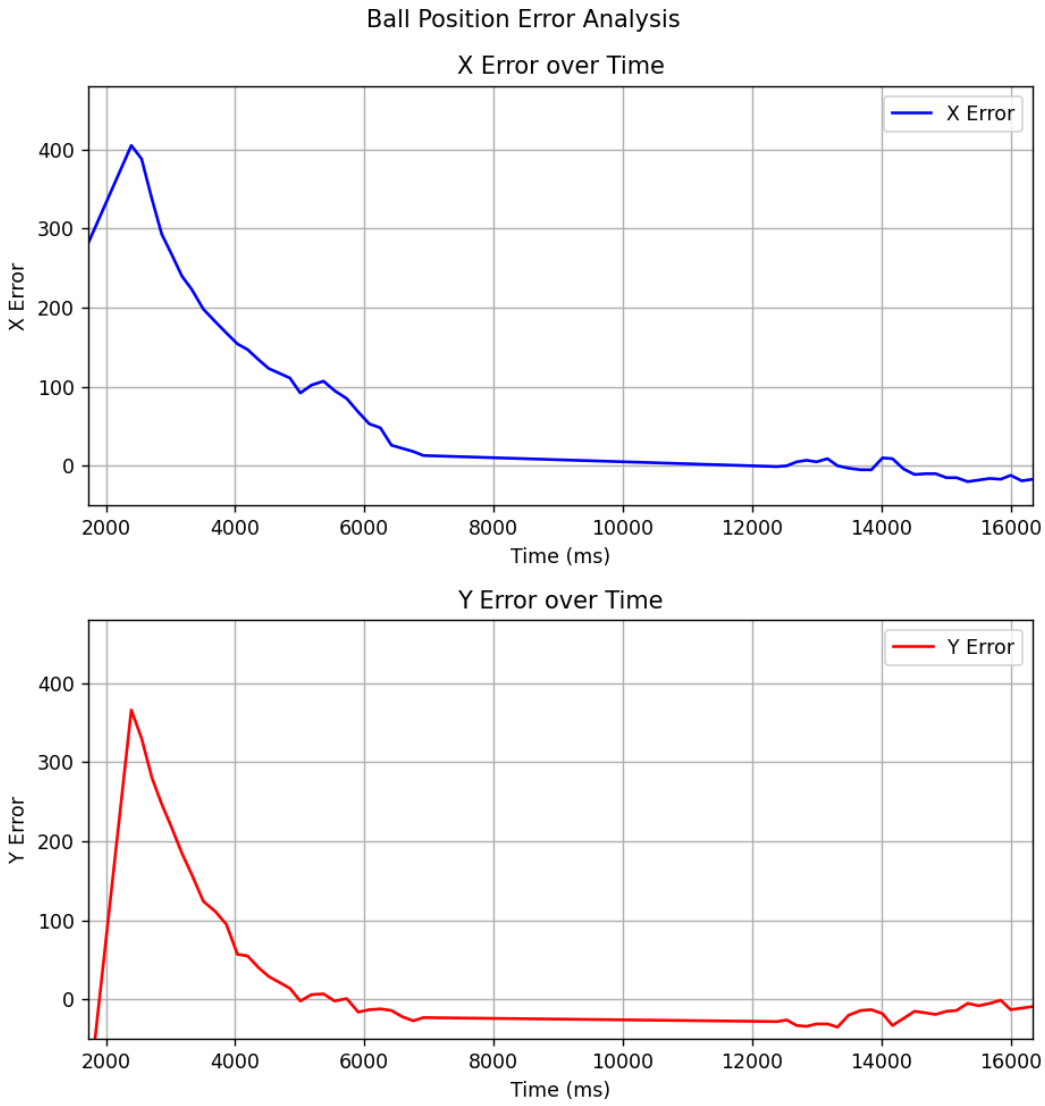

Figure 6.3 Step response of the real system

As described earlier, in both cases the ball was placed at the edge represented by coordinates [400,400]. The error is plotted against time. From the plots we can determine the following:

Table 6.2 Step Response Analysis

Parameter	**Simulink**		**Real**	
	X Axis	**Y Axis**	**X Axis**	**Y Axis**
Settling time	4.5 sec	4.6 sec	4.2 sec	4 sec
Overshoot	0	0	~ 0 %	~10 %
Steady state error [mm]	0	0	~ 0	~ 25
Peak [mm]	400	400	~ 400	~ 380
Stability	Stable	Stable	Jitter	Jitter

Let's start by analyzing the simulated and real results individually, followed by a comparative analysis of the step response.

Simulated result analysis:

1. In the simulated system, both axes (X and Y) stabilize without any overshoot which indicates precise and robust control.
2. Exponential decay is observed which confirms that the system is stable.
3. No steady state error or jittering is observed, which indicates the output is that of an ideal system.

Real system result analysis:

1. In the real system, both axes (X and Y) stabilize without any overshoot which indicates precise and robust control.
2. Exponential decay is observed which confirms that the system is stable.
3. While the X axis is satisfactory, steady state error is observed in the Y axis. This is due to a slight tilt of the platform in the Y direction, which was measured using a mercury level.
4. Jittering after stabilization was observed in both X and Y axes. This is most likely due to the noise in the input signal from the touchpad.

The real system performs similarly to the simulated system in terms of stabilization, overshoot, and exponential decay, indicating that the control strategy is effective in both environments.

6.4 Frequency Response

Instead of a static reference point a sine wave with frequency of 0.25 Hz and an amplitude of 300 units was set as the input signal.

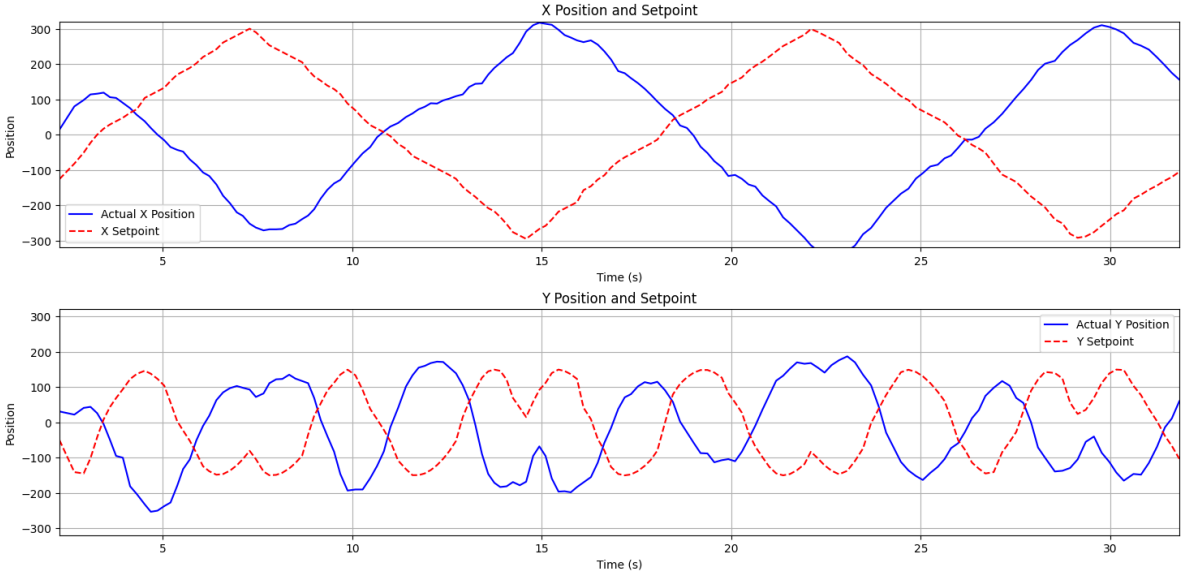

Figure 6.4 Frequency response of the real system

From the response plotted in Figure 6.4 it can be observed that the real system had a lag of $\sim 2\,sec$. For the same signal the simulated system had no lag. This difference might be due to one or more of the following reasons.

1. In the real system the touchpad sampling rate and processing time might introduce a delay which is absent in the simulation.
2. The actuator response time is instantaneous in the simulation while its significantly higher in the real system.

The simulated system doesn't consider the system dynamics like friction, friction in the links and other unmodeled higher order dynamics.

6.5 Disturbance Rejection Test

The reference point is set to origin [0,0], and the ball is subjected to random disturbances (random movements) to analyse how the system responds.

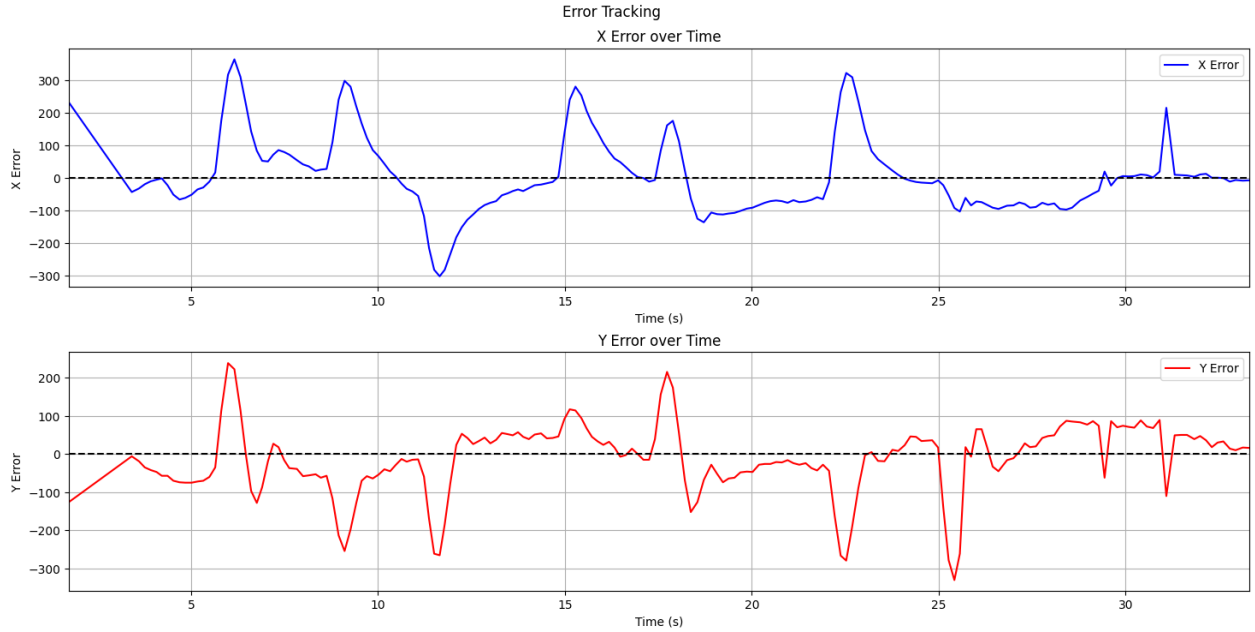

Figure 6.5 Disturbance rejection of the real system

In Figure 6.5, the peaks correspond to the moments when the ball was subjected to external force or disturbance. The average settling along both axes is about $\sim 4\,sec$, which is consistent with the step response analysis in Section 6.3. This consistency indicates that system's dynamic behaviour as characterized by the step response analysis translates well in scenarios involving external disturbance.

Unlike the step response, a slight overshoot can be observed at some instances. Additionally, some jittery motion is noticeable. An average steady-state error of around $\sim 25mm$ is also evident, which is consistent with the results of the step response analysis.

6.6 Creating Trajectories

In this section the different shapes will be traced by controlling the movement of the ball as explained in Section 5.5.

6.6.1 Circle

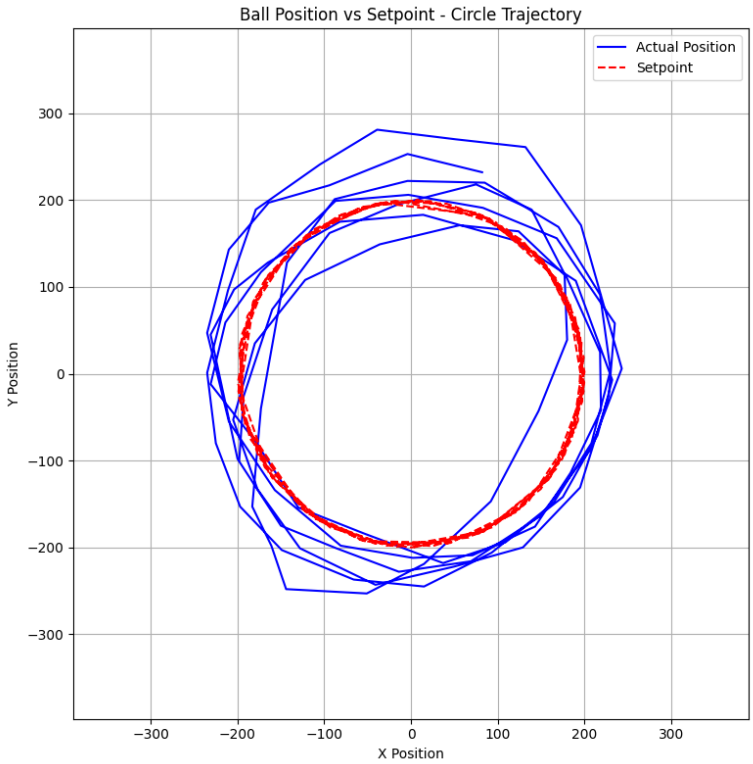

Figure 6.6 Circle trajectory traced by the ball motion

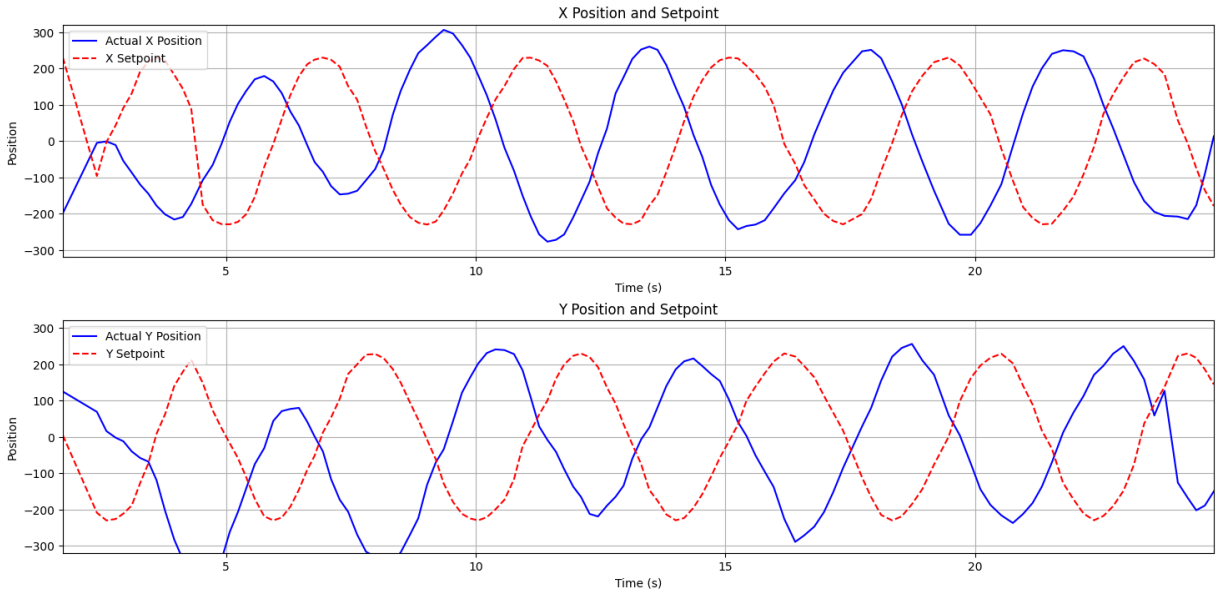

Figure 6.7 Plot of delay and error – Circle trajectory

6.6.2 Infinity

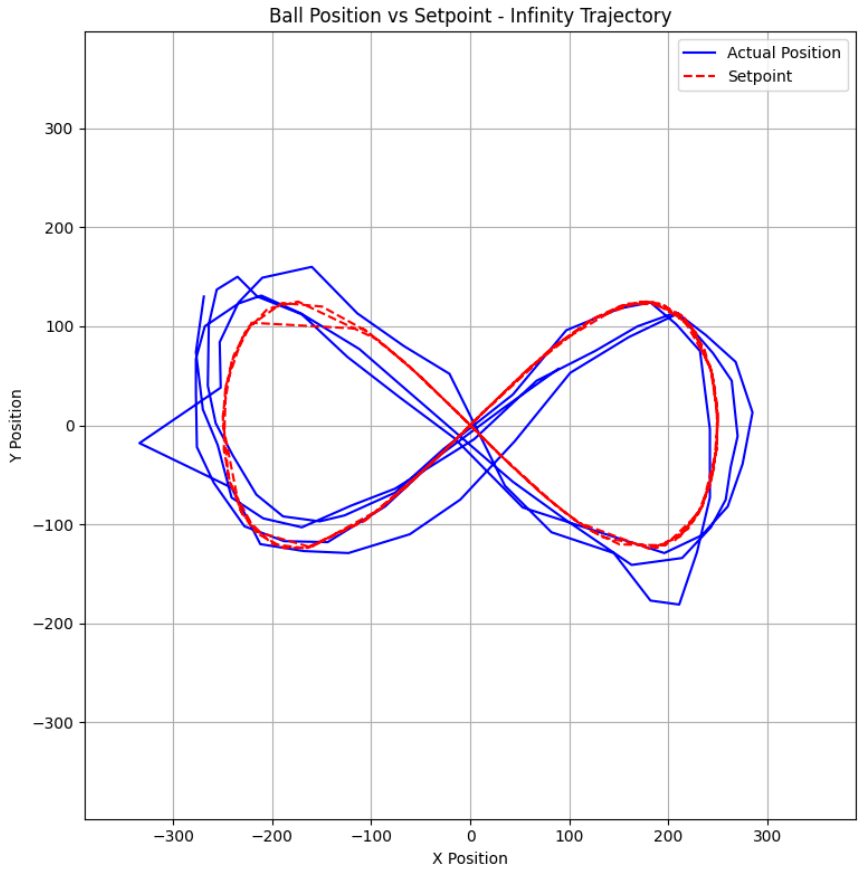

Figure 6.8 Infinity trajectory traced by the ball motion

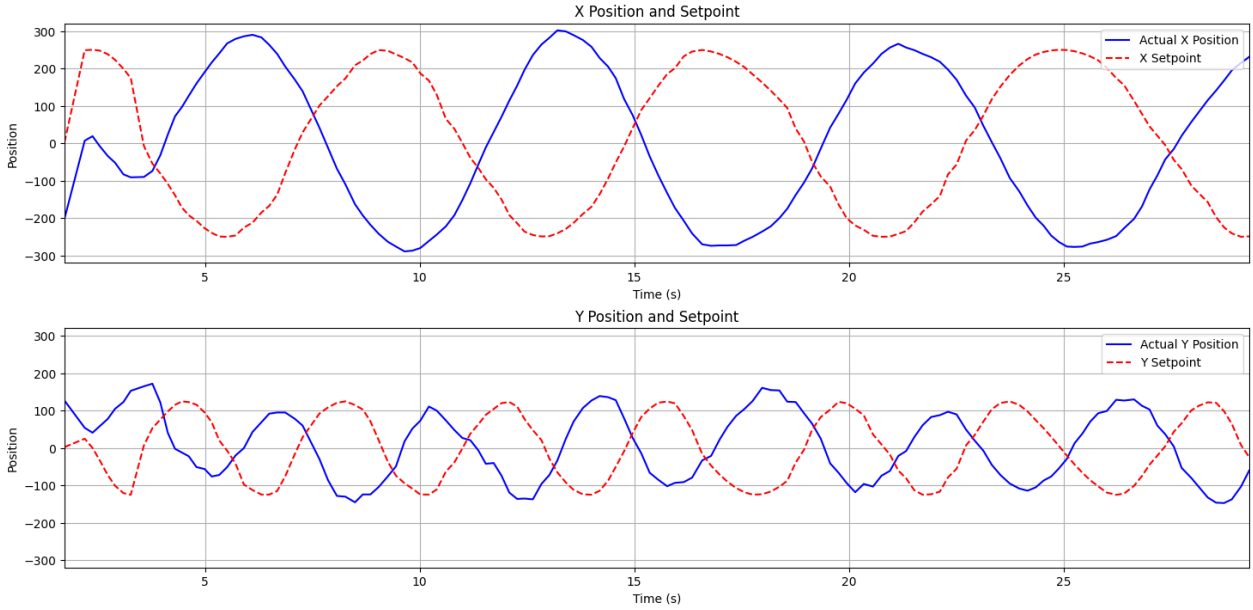

Figure 6.9 Plot showing delay and error – Infinity trajectory

6.6.3 Square

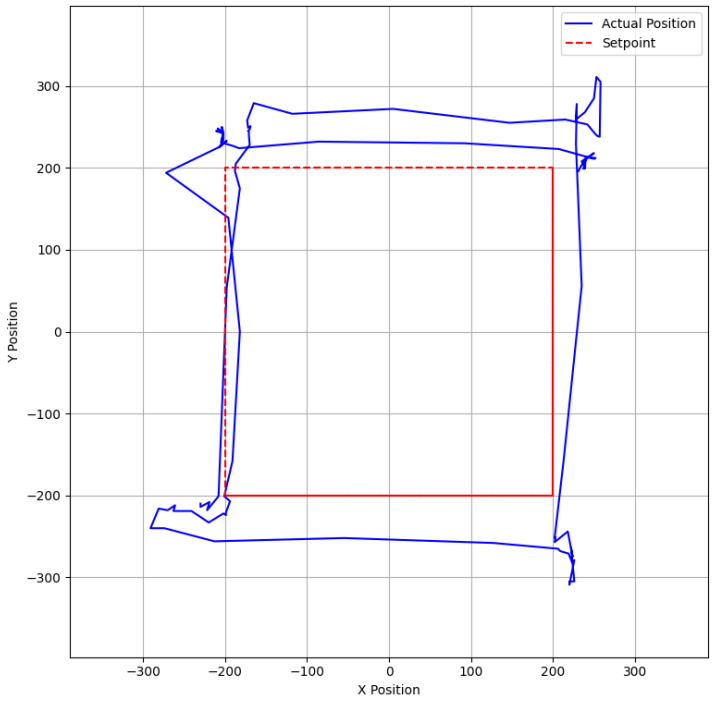

Figure 6.10 Square trajectory traced by the ball motion

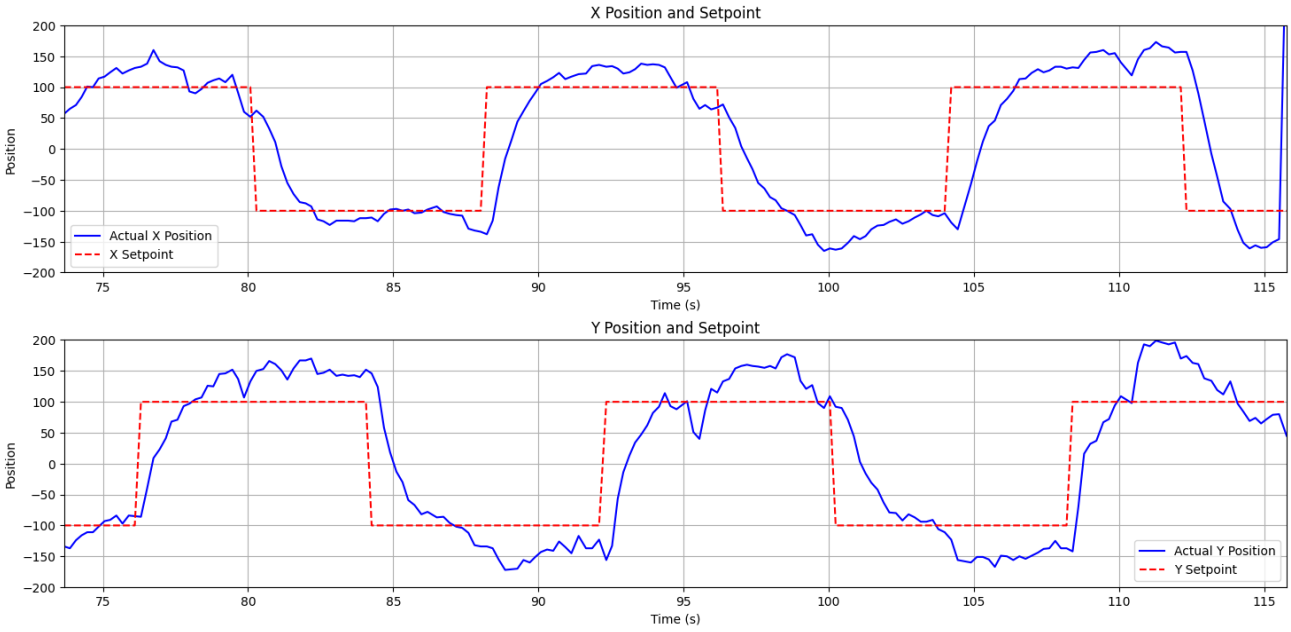

Figure 6.11 Plot showing delay and error - square trajectory

Based on the above plots, it can be concluded that tracing various trajectories was achievable. Interestingly, continuous curves (circle and infinity) achieved more precision compared to sending discrete vertices at regular intervals, as in the case of the square. The observed lag of approximately 2 seconds, as discussed in Section 6.4, is also evident here.

7 Conclusion

8 Further Research

In recent years, different control strategies for the ball on a plate problem have been proposed and tested. PID control was chosen in this work due to its ease of implementation and well-established reliability. However, there are several alternative approaches worth considering, which will be discussed in this chapter. These alternative controllers can be implemented in the existing system by making modifications solely to the software.

8.1 Sliding Mode Controller

8.2 Model Predictive Controller

8.3 Linear Quadratic Controller

8.4 Fuzzy Logic Controller

In Boolean logic, a statement is either true or false. However, there are many situations where we cannot strictly say whether something is true or false in the real world. Fuzzy logic was developed to tackle this problem. It makes it possible to give a statement a value on a scale between entirely true and entirely false in a structured way. A Fuzzy Logic Controller can be combined with a PID controller to create a Fuzzy PID controller. These can be implemented in two main ways, direct action and gain scheduling. Gain scheduling uses a FLC to dynamically change the PID gains depending on some input from the system, for example, position and or speed. [8]

In context to the ball on a plate system, the work by Strand Filip and Wahlund Martin [8] compares the effectiveness of FLC in conjunction with PID with that of a standalone PID. The authors conclude that FLC controller offers significantly more possibilities for adjustment according to the requirement while being more time-consuming and difficult to implement.

9 References

[1] M. T. Alexander Hasp Frank, "Construction and theoretical study of ball balancing platform," KTH Royal Institute of Technology, Stockholm, Sweden, 2019.

[2] E. Lakzaei, "Balancing ball on a plate-Final Report," Eindhoven University of Technology, Eindhoven, 2020.

[3] J. Maxwell, "ON GOVERNORS," 1868.

[4] R. Kalman, "A New Approach to Linear Filtering," *Journal of Basic Engineering,* 1960.

[5] H. Tetik, "Modelling and Control of a 3-Rrs Parallel Manipulator," Izmir Institute of Technology, Izmir, 2016.

[6] B. Zi, C. Liu and Y. Yueqing, "Dynamics of 3-DOF spatial parallel manipulator with flexible links," 2010.

[7] Y. Chen, System simulation techniques with MATLAB and Simulink, John Wiley & Sons, 2013.

[8] F. D. Cero, "Ball and Plate MPC Control of a 6," Universita degli Studi di Padova , Padua, Italy, 2022.

[9] W. M. Strand Filip, "Control of the ball and plate - A comparative study of controllers," KTH Royal Institute of Technology, Stockholm, 2022.

[10] K. Ł. M. Zarzycki, "Fast Real-Time Model Predictive Control for a Bal-on-a-plate process," *Sensors,* 2021.

[11] H. Bang, "Implementation of a Ball and Plate Control System Using Sliding Mode Control," *IEEE Access PP,* 2018.

APPENDIX

Table of Figures

Figure 1.1 Balancer with 2 actuators built at KTH Royal Institute of Technology [1] ... 3
Figure 1.2 Ball Balancer with vision-based feedback built at TU Eindhoven [2] 3
Figure 2.1 A block diagram of a PID controller in a feedback loop. r(t) is the desired process variable (PV) or setpoint (SP), and y(t) is the measured PV. [5] 5
Figure 2.2 3RRS PM platform 7
Figure 4.1 Control Algorithm describing the process. 10
Figure 4.2 Control System block diagram for a single axis 12
Figure 4.3 Kinematic sketch of 3RRP parallel manipulator [7] 13
Figure 4.4 Three-dimensional system represented as a pair of two-dimensional system [1] 16
Figure 4.5 Free body diagram of the ball in two dimension [1] 16
Figure 5.1 3RRS Ball Balancer mechanical design by Aaed Musa (description below). 19
Figure 5.2 Arduino UNO pinout diagram. Maker UNO has an identical configuration. 20
Figure 5.3 TMC2209 V2.0 stepper driver pinout diagram. 21
Figure 5.4 CNC shield pinout diagram 22
Figure 5.5 Nema 17 Bi-polar stepper motor. 23
Figure 5.6 Picture taken during printing the top platform 24
Figure 5.7 Newly printed parts with raft 24
Figure 5.8 CNC shield with drivers connected to UNO (L) Micro stepping jumper pins ($1/16^{th}$ configuration) (R) 25
Figure 5.9 Partially assembled platform 26
Figure 5.10 Assembled 3RRS PM 26
Figure 6.1 Simulink Model of the system. 27
Figure 6.2 Simulink Model of Ball Motion Block 27
Figure 6.3 Scaling X error values to be in the range [-0.25,0.25] 28
Figure 7.1 X Axis step response of the simulated system 31
Figure 7.2 Y Axis step response of the simulated system 31
Figure 7.3 Step response of the real system 32
Figure 7.4 Frequency response of the real system 33
Figure 7.5 Disturbance rejection of the real system 34
Figure 7.6 Circle trajectory traced by the ball motion 35
Figure 7.7 Plot of delay and error – Circle trajectory 35

List of Tables

Table 4.1 Description of symbols in Figure 4.2 13
Table 4.2 Inverse Kinematics parameters 15
Table 4.3 Ball motion model parameters 18
Table 5.1 Maker UNO important technical specification. 20
Table 5.2 TMC2209 Micro stepping Configuration. 21
Table 5.3 TMC2209 technical specifications. 22
Table 5.4 Cross-referencing of Arduino pins 22
Table 5.5 Nema 17 technical specifications 23
Table 7.1 PID Gain Values 31
Table 7.2 Step Response Analysis 32

Code – Arduino Controller

```
// Forward declarations
void setup();
void loop();
void calculateIKPositions(double hz, double nx, double ny, long* positions);
void moveTo(long* positions);
void PID(double setpointX, double setpointY);

// Stewart Platform Control Code

// Libraries
#include <AccelStepper.h>
#include <InverseKinematics.h>
#include <MultiStepper.h>
#include <stdint.h>
#include <TouchScreen.h>
#include <math.h>

// Machine Parameters
Machine machine(2, 3.125, 1.75, 3.669291339);     // (d, e, f, g) lengths of the machine
TouchScreen ts = TouchScreen(A1, A0, A3, A2, 0);  // Touchscreen pins (XGND, YGND, X5V, Y5V)

// Stepper Motor Definitions
AccelStepper stepperA(1, 2, 5);  // (driver type, STEP, DIR) Driver A
AccelStepper stepperB(1, 3, 6);  // (driver type, STEP, DIR) Driver B
AccelStepper stepperC(1, 4, 7);  // (driver type, STEP, DIR) Driver C
MultiStepper steppers;           // Multi-stepper control instance

// Stepper Motor Variables
long pos[3];                     // Target positions for each stepper motor
int ENA = 8;                     // Enable pin for the drivers
double angOrig = 206.662752199;  // Original angle for each leg

// Speed Control Variables
double speed[3] = {0, 0, 0};     // Current speed of stepper motors
double speedPrev[3];             // Previous speed of stepper motors

// Touchscreen Variables
double Xoffset = 500;  // X offset for touchscreen center
double Yoffset = 500;  // Y offset for touchscreen center

// PID Control Variables
double kp = 0.1670, ki = 0.00085, kd = 0.1700;  // PID constants
double error[2] = {0, 0};                    // Current error for X and Y directions
double errorPrev[2];                         // Previous error for X and Y directions
double integr[2] = {0, 0};                   // Integral terms for X and Y
double deriv[2] = {0, 0};                    // Derivative terms for X and Y
double out[2];                               // PID output for X and Y
double rawErrorX, rawErrorY;                 // Variables to store raw error values before normalization

// Miscellaneous Variables
double angToStep = 3200.0 / 360;  // Angle-to-step conversion factor (steps per degree)
bool detected = false;            // Ball detection flag
long timeI;                       // Timing variable for delay
unsigned long lastTime = 0;       // Timing for PID updates

void setup() {
  Serial.begin(115200);

  // Configure MultiStepper
  steppers.addStepper(stepperA);
  steppers.addStepper(stepperB);
  steppers.addStepper(stepperC);

```

```
  // Enable Pin Configuration
  pinMode(ENA, OUTPUT);
  digitalWrite(ENA, HIGH);  // Disable drivers initially
  delay(1000);              // Allow user to reset the platform
  digitalWrite(ENA, LOW);   // Enable drivers

  // Set initial speeds for homing
  stepperA.setMaxSpeed(600);
  stepperB.setMaxSpeed(600);
  stepperC.setMaxSpeed(600);
  stepperA.setAcceleration(300);
  stepperB.setAcceleration(300);
  stepperC.setAcceleration(300);

  // Calculate home position
  long homePos[3];
  calculateIKPositions(4.25, 0, 0, homePos);

  // Move to home position
  steppers.moveTo(homePos);
  steppers.runSpeedToPosition();  // This blocks until all steppers reach position
}

void loop() {
  PID(0, 0);  // Run PID control with setpoints (X: 0, Y: 0)
}

// Function to calculate inverse kinematics positions
void calculateIKPositions(double hz, double nx, double ny, long* positions) {
  for (int i = 0; i < 3; i++) {
    positions[i] = round((angOrig - machine.theta(i, hz, nx, ny)) * angToStep);
  }
}

// Function to Move the Platform
void moveTo(long* positions) {
  stepperA.moveTo(positions[0]);
  stepperB.moveTo(positions[1]);
  stepperC.moveTo(positions[2]);

  // Run Steppers Incrementally
  stepperA.run();
  stepperB.run();
  stepperC.run();
}

void PID(double setpointX, double setpointY) {
  TSPoint p = ts.getPoint();  // Read touchscreen position
  long positions[3];          // Local array for positions
  static unsigned long lastUpdateTime = millis();  // Last update time
  unsigned long currentTime = millis();            // Current time
  double deltaTime = (currentTime - lastUpdateTime) / 1000.0;  // Time difference in
seconds

  if (p.x != 0) {  // Ball detected
    detected = true;

    // Calculate Errors
    rawErrorX = Xoffset - p.x - setpointX;
    rawErrorY = Yoffset - p.y - setpointY;
    double errorZ = 4.25;
    // Normalize Error Vector
    double magnitudeX = 1760;
    double magnitudeY = 1520;
    double normX = rawErrorX / magnitudeX;
    double normY = rawErrorY / magnitudeY;

    // Compute error magnitude for dynamic speed adjustment
    double errorMagnitude = sqrt(rawErrorX * rawErrorX + rawErrorY * rawErrorY);

    // Compute PID Terms for X and Y
    for (int i = 0; i < 2; i++) {
```

```
136.       errorPrev[i] = error[i];
137.       error[i] = (i == 0) ? normX : normY;
138.       integr[i] += error[i];      // Simple integration without thresholding
139.
140.       // Derivative term calculation with time normalization
141.       deriv[i] = (error[i] - errorPrev[i]) / deltaTime;
142.       deriv[i] = isnan(deriv[i]) || isinf(deriv[i]) ? 0 : deriv[i];
143.
144.       out[i] = kp * error[i] + ki * integr[i] + kd * deriv[i];
145.       out[i] = constrain(out[i], -0.25, 0.25);  // Constrain the output to prevent
saturation
146.     }
147.
148.     // Calculate IK positions before the timing loop
149.     calculateIKPositions(4.25, -out[0], -out[1], positions);
150.
151.     // Dynamic speed and acceleration calculation based on error magnitude
152.     double normalizedError = constrain(errorMagnitude / 400.0, 0, 1);
153.
154.     // Set dynamic speed and acceleration based on error magnitude
155.     double maxSpeed = 1200;          // Maximum speed
156.     double minSpeed = 900;           // Minimum speed
157.     double dynamicSpeed = minSpeed + normalizedError * (maxSpeed - minSpeed);
158.
159.     double maxAccel = 2400;          // Maximum acceleration
160.     double minAccel = 1200;          // Minimum acceleration
161.     double dynamicAccel = minAccel + normalizedError * (maxAccel - minAccel);
162.     stepperA.setMaxSpeed(dynamicSpeed);
163.     stepperB.setMaxSpeed(dynamicSpeed);
164.     stepperC.setMaxSpeed(dynamicSpeed);
165.
166.     stepperA.setAcceleration(dynamicAccel);
167.     stepperB.setAcceleration(dynamicAccel);
168.     stepperC.setAcceleration(dynamicAccel);
169.
170.     // Pass dynamic speed and acceleration to moveTo function
171.     timeI = millis();
172.     while (millis() - timeI < 12) {
173.       moveTo(positions, dynamicSpeed, dynamicAccel);  // Pass positions, dynamic
speed, and acceleration
174.
175.       // Exit loop if all motors have finished their motion
176.       if (stepperA.distanceToGo() == 0 && stepperB.distanceToGo() == 0 &&
stepperC.distanceToGo() == 0) {
177.         break;
178.       }
179.     }
180.     Serial.println(String(rawErrorX) + "," + String(rawErrorY)+","+
String(millis()));
181.     lastUpdateTime = currentTime;  // Update the last update time
182.   } else {
183.     // Handle Ball Not Detected
184.     long homePos[3];
185.     calculateIKPositions(4.25, 0, 0, homePos);
186.     moveTo(homePos, 800, 1000);  // Pass 0 error magnitude for home position and no
speed/acceleration
187.   }
188. }
190. // Updated moveTo function to accept dynamic speed and acceleration
191. void moveTo(long* positions, double dynamicSpeed, double dynamicAccel) {
192.   // Set speed and acceleration dynamically
193.
194.   // Move steppers to target positions
195.   stepperA.moveTo(positions[0]);
196.   stepperB.moveTo(positions[1]);
197.   stepperC.moveTo(positions[2]);
198.
199.   // Run steppers incrementally
200.   stepperA.run();
201.   stepperB.run();
202.   stepperC.run();
203. }
```

Attachments