****Abstract****

This thesis explores the construction of a ball-balancing platform. This control system 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 |
| **BoP** | Ball-on-Plate |
| **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 |

Table Of Contents

[1 Introduction 1](#_Toc188819681)

[1.1 Background and Motivation 1](#_Toc188819682)

[1.2 Project Summary 1](#_Toc188819683)

[1.3 Thesis structure 2](#_Toc188819684)

[1.4 Scope of the Work 2](#_Toc188819685)

[2 Literature Review 3](#_Toc188819686)

[2.1 Bop Systems: Existing Designs 3](#_Toc188819687)

[2.2 Control strategy 6](#_Toc188819688)

[2.3 Research questions 6](#_Toc188819689)

[3 Project Overview And Requirements 7](#_Toc188819690)

[3.1 Objectives 7](#_Toc188819691)

[3.2 Performance metrics 8](#_Toc188819692)

[3.3 Assumptions 8](#_Toc188819693)

[4 System Description 9](#_Toc188819694)

[4.1 Process description 10](#_Toc188819695)

[4.2 Normal vector calculation 10](#_Toc188819696)

[4.3 PID controller 11](#_Toc188819697)

[4.1 Block diagram of the system 13](#_Toc188819698)

[4.2 Inverse Kinematics of the system 14](#_Toc188819699)

[5 Hardware Implementation 17](#_Toc188819700)

[5.1 Specific hardware requirements 17](#_Toc188819701)

[5.2 Electrical configuration 18](#_Toc188819702)

[5.3 Construction and Assembly 22](#_Toc188819703)

[6 Simulink Modelling 25](#_Toc188819704)

[6.1 Equations describing the motion of the ball 25](#_Toc188819705)

[6.2 Simulink block diagrams 29](#_Toc188819706)

[6.3 PID tuning – simulation 30](#_Toc188819707)

[7 Software Implementation 31](#_Toc188819708)

[7.1 Reading touchpad data 33](#_Toc188819709)

[7.2 PID tuning – real system 33](#_Toc188819710)

[7.1 Creating trajectories 33](#_Toc188819711)

[8 Results And Analysis 34](#_Toc188819712)

[8.1 Assembled platform 34](#_Toc188819713)

[8.2 PID gains 35](#_Toc188819714)

[8.3 Step Response testing 35](#_Toc188819715)

[8.4 Frequency response 37](#_Toc188819716)

[8.5 Disturbance rejection test 38](#_Toc188819717)

[8.6 Creating trajectories 39](#_Toc188819718)

[8.7 Results – discussion 42](#_Toc188819719)

[9 Conclusion 43](#_Toc188819720)

[10 Further Work 45](#_Toc188819721)

[11 References 46](#_Toc188819722)

# Introduction

This chapter outlines the motivation behind the project, provides a detailed project summary, and presents the structure of the thesis. Finally, the chapter presents key questions that will be addressed through the outcomes of this work.

## Background and Motivation

Laboratories play a vital role in the effective education of engineering students. In the context of automatic control, students use classical laboratory processes such as water tanks and inverted pendulums.[[1]](#footnote-1), the magnetic levitation system and the ball-on-plate benchmark. In particular, the ball-on-plate process is exciting 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 a compelling example of control courses. [1] Thus, the **educational value** and the **practical applicability** are the primary motivating factors for this work.

## Project summary

This project investigates the implementation and effectiveness of linear control methodologies in stabilizing the Ball-on-Plate (BoP) System. The work encompasses modelling the proposed system and building a working prototype to conduct experiments.

The system utilizes a resistive touchpad to detect the ball's position and uses three stepper motors to adjust the plate's tilt dynamically. The core objective is to stabilize the ball's position using a linear controller. A Simulink model will be developed to understand the system's behaviour better and make PID tuning easier. Lagrangian mechanics will be used to formulate the ball's motion on the platform. The Simulink model will help to predict the system's response and fine-tune the PID controller gains before implementing them on the physical prototype.

Once stabilization is achieved in the physical system, different shape trajectories will be traced by precisely controlling the ball's position. Finally, a comparative analysis will be made between the theoretical results derived from the Simulink model and the experimental results obtained from the physical prototype. This analysis will help identify discrepancies between the model and the real system and understand the PID controller's limitations in stabilizing and controlling the ball's motion.

## Thesis structure

The **Literature Review** examines prior work on BoP systems and design choices and provides some theoretical foundations. Following that**,** the **Project Overview and Requirements** outlines objectives, performance metrics, scope, and assumptions made. The next chapter, General Concepts of the System, explains key elements, such as PID controller, inverse kinematics, and ball motion modelling. The **Hardware Implementation** chapter describes the prototype's fabrication and assembly, including the electrical configuration. The Simulink modelling is then described in a separate chapter. After this**,** the **Software Implementation** chapter details the real system's control algorithm, PID tuning, and trajectory generation methods. Following this**, the Results and Analysis chapter** presents experimental and simulated results and trajectory tracking outcomes. **The Conclusion chapter** summarises findings, evaluates the PID controller's performance, and discusses outcomes and limitations. Finally, further improvements are described.

## Scope of the work

The scope of the thesis includes constructing the BoP system, simulating it in the Simulink environment, implementing a linear controller, successfully balancing the ball and tracing some simple trajectory curves. The mechanical design of the manipulator system and the implementation of non-linear controllers are beyond the scope of this work. The mechanical design itself is taken from open-source work by Aaed Musa[[2]](#footnote-2). The electrical configuration, control software, fabrication and construction, trajectory creation, and Simulink modelling are within the scope of this thesis.

# Literature Review

In this chapter, the prerequisite knowledge for the implementation of the proposed BoP system will be discussed. The project is interdisciplinary and requires some basic electronics, mechanics, and control design knowledge. However, two fundamental choices must be made: BoP **system design** and the **control algorithm**. Researchers have proposed and developed many solutions, some of which will be considered when making the appropriate choices.

## Bop Systems: Existing Designs

BoP system design can be split into the **actuator design** and the **implementation of the position feedback system**. There are multiple approaches to meet the required design criteria, and factors like ease of implementation, cost, and performance will influence the final choice.

### Position feedback system

The position feedback system aims to track the ball's position with high precision, accuracy, and low latency. A vision-based feedback system like the one in Figure 2.1 and Figure 2.4 use cameras for precise ball tracking but face challenges with physical setup, portability, and computational demands. [2] Another approach is to use a resistive touchpad like in the Figure 2.2. This method is more straightforward to set up, compact, cost-efficient, and computationally less demanding.

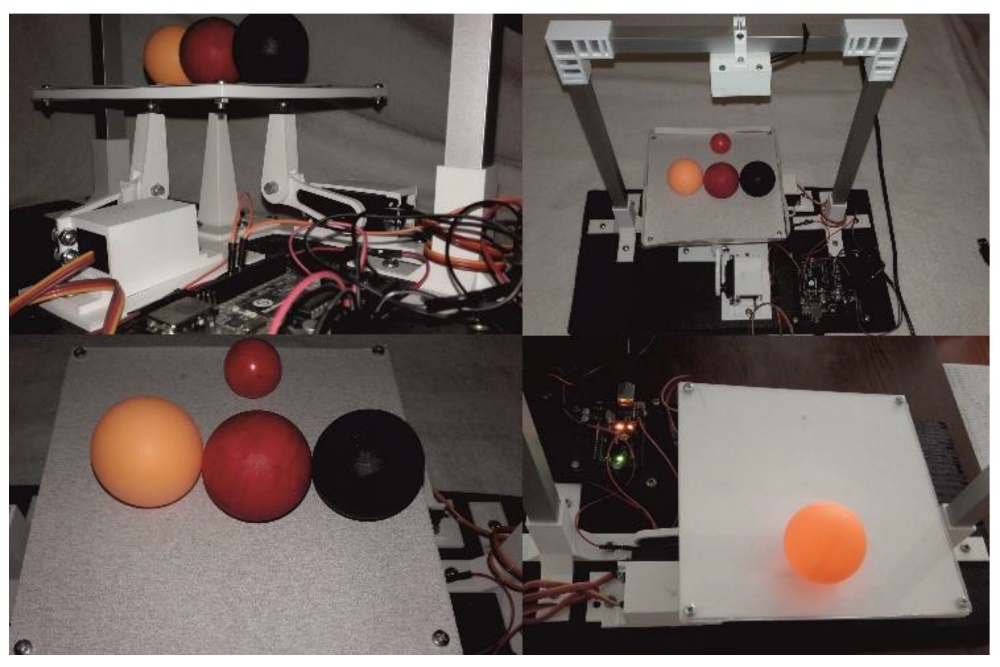


Figure 2.1 BoP system with camera-based feedback control and two actuators [3]

A glass and metal device with wires

Description automatically generated

Figure 2.2 BoP system with two actuators and a resistive touchpad-based feedback built at KTH Royal Institute of Technology [4]

### Manipulator design

A manipulator[[3]](#footnote-3) is a mechanical system that uses several computer-controlled serial chains to support a single platform or end-effector. A Stewart platform is one of the most popular manipulators, which offers 6 DOF. A minimum of 2 DOF is required for the proposed BoP system; thus, a simpler manipulator is sufficient.

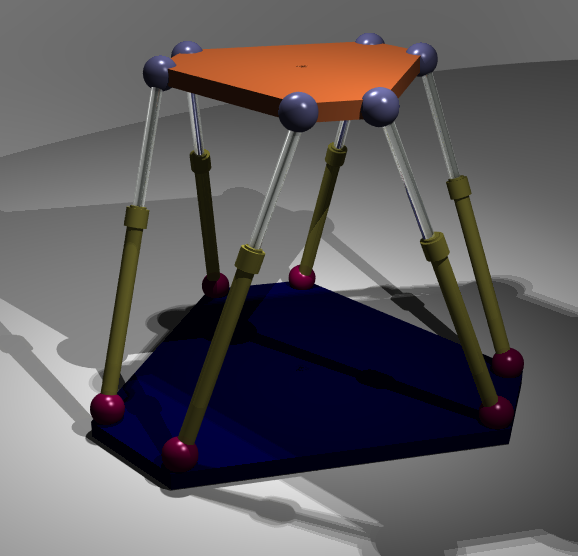


Figure 2.3 Hexapod platform (Stewart Platform). Source: Wikipepedia3

In the context of the BoP system, the manipulator's goal is to enable the platform to tilt along the X and Y direction and hence 2 DOF. Systems with two actuators like the one in Figure 2.2 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 [4].

A white rectangular object on a metal stand

Description automatically generated A metal structure with black wires

Description automatically generated with medium confidence

Figure 2.4 BoP system with camera-based feedback control and three actuators built at TU Eindhoven [5]

Multi-actuator setups like in the Figure 2.4 offer better control and precision and hence, a 3-RRS PM is used in this work. In a 3RRS-PM, each limb has the same kinematic structure (RRS), and these limbs are attached to the base and move symmetrically to the platform. 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). [6] The kinematic sketch of a 3RRS-PM is illustrated in Figure 2.5

A diagram of a triangle and a triangle with circles and lines

Description automatically generated

Figure 2.5 Kinematic sketch of a 3RRS parallel manipulator [7]

To summarise, a 3RRS-PM with resistive touchpad-based position feedback is implemented.

## Control strategy

Choosing the proper control strategy is a crucial aspect of the project. There are multiple approaches to solving this problem. The classical PID controller remains the most straightforward and widely used controller. If state equations are used to describe the ball’s position, an LQ can also be used. [8] Similarly, SMC can also be used to achieve more accuracy, especially when tracing curves. [9]. More advanced solutions include Fuzzy controllers, MPC, and disturbance-observer-based friction compensation schemes [1]. However, non-linear controllers are beyond the scope of this thesis and are not considered for implementation. Thus, a classical PID controller was chosen to be implemented.

A robotic arm with a glass plate

Description automatically generated

Figure 2.6 BoP system modeled in the ABB RobotStudio with LQR controller. [8]

## Research questions

This work aims to address the following questions with proper justification obtained by means of conducting experiments and measurements.

1. **Is the linear control methodology adequate for satisfactory performance in a non-linear system like a BoP system?**

This question investigates the adequacy of linear control techniques in managing an inherently nonlinear system.

1. **Is it possible to create a digital twin of the prototype? How beneficial is such a system in meeting the required objectives?**

The project aims to determine the source of the difference between the theoretically predicted and practically observed behavior and quantify the causes-mechanically, electrically, and environmentally.

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

# Project Overview and Requirements

This chapter outlines the methodology for developing a BoP control system, detailing objectives, performance criteria, design assumptions, and general hardware requirements

## Objectives

The objectives of the project are as follows:

1. **Construction:** Fabricate and assemble the platform successfully.

The first objective is to construct the BoP system physically. This involves purchasing some components and 3D printing the required parts, including the platform, actuator legs, and base structure. The assembly process includes integrating stepper motors, a resistive touchpad, and a microcontroller to create a functional system. Proper alignment and calibration of all parts are crucial to ensure smooth operation. Finally, the mechanical and electrical integrity of the assembled platform must be verified to ensure it is ready for control implementation.

1. **Simulation:** Develop a Simulink model to simulate the proposed system.

The second objective is to simulate the BoP system in Simulink environment. Proper models describing the motion of the ball must be formulated. This simulation is initially used for PID tuning. The tuned parameters are then applied to the actual system. The simulated model also serves as an ideal system for reference. The prototype's accuracy is validated by comparing the experimental and simulated results.

1. **Balancing:** Successfully stabilise a ball on a platform with a linear controller.

The third objective aims to achieve stable ball balancing using a linear controller in the physical system. This involves implementing a control algorithm that dynamically adjusts the platform's tilt based on real-time position feedback from the sensor. The goal is to maintain the ball at a pre-defined SP, even in the presence of external disturbances.

1. **Tracing Curves:** Implement control software to enable the ball to trace predefined trajectories on the platform.

The fourth objective is to continuously change the SP, to trace different curves. This further tests the system’s performance and helps analyze the control lag and the platform’s adaptability to different conditions.

1. **Further study**: Research other control strategies that can be used.

## Performance metrics

The following metrics will measure the outcome of the work:

1. **Overshoot < 5%**: The ability of the system to reach a steady state without divergence. A quick convergence to SP is required.
2. **Steady State Error ()**: Average offset between the SP and the PV at equilibrium. An average value of less than **20 mm** is required.
3. **Settling time (5%) ()**: The time required for the system's response to input changes or disturbances to converge within 5% of the set value should be less than 4 seconds.
4. **Robustness**: The system should be able to smoothly reach the SP even in varying conditions with minimal jitter and oscillations.
5. **Motion Accuracy (only for the path tracing case)**: The deviation between the predefined path and the actual path traced by the ball.

## Assumptions

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

1. Friction between the ball and platform 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 move 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 actuation speed (the angular velocity) or the rotor speed is not considered for simulation.

It is assumed that general mechanical and electrical tools are available for use. Additionally, a 3D printing facility is required to fabricate the necessary parts. Basic programming skills and knowledge of microcontrollers are also needed to complete the work successfully.

# System Description

This chapter details the system design of the BoP system. The mechanical design of the BoP system is credited to **Aaed Musa.**[[4]](#footnote-4) The 3RRS platform has **3 DOF and 3 points of actuation**. Three stepper motors with TMC2209 drives mounted on a CNC shield are used for precise actuation. A Maker UNO board is used as the central controller. Plastic or lightweight balls cannot be used as they don’t generate enough pressure for the touchpad to detect contact.



4

2

3

5

6

1

Figure 4.1 3RRS BoP system CAD design by Aaed Musa

A metallic ball **1** is placed on a resistive touchpad **2,** which outputs the ball's position 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.**

## Process description

The process of the BoP system can be simplified into three major blocks. The process starts by calculating the error, which is then passed on to the system controller. The system controller determines the required orientation of the platform plane to compensate for the error. The Inverse kinematics block is responsible for actuating the plane to the desired orientation.

A diagram of a system controller

Description automatically generated

Figure 4.2 Process overview of the system

The following sections of this chapter explain each of the blocks in Figure 4.2 in detail. The system controller block has been split into two sections: the PID controller and the system block diagram.

## Normal vector calculation

The error is defined as the difference between the coordinates of SP and PV on the two-dimensional platform plane. Once the error is determined, the next step is calculating the new normal vector required to move the ball to the SP. The normal vector of the platform plane is denoted by . The new normal vector ( ) after compensating for the error is given as:

A diagram of a line with a point and a circle

Description automatically generated with medium confidence

Figure 4.3 Figure representing the position of the ball X, error , and the normal vector . It is assumed that at home position,

The platform can only tilt 0.25 units, anything more than that would destabilise 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] before calculating the .

A graph with a line graph

Description automatically generated

Figure 4.4 Scaling X error values to be in the range [-0.25,0.25]

## PID controller

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.

A screenshot of a computer

Description automatically generated

Figure 4.5 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.

In the context of a BoP system, a PID controller 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 by summing the past error, and the derivative term predicts and counters rapid changes in the error to improve stability. The overall control function is described by,

Where is the output control signal at time t and e(t) is the error at time t, is the proportional gain, is the integral gain, and is the derivative gain. The same symbol description applies to the rest of this section.

### 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 , known as the proportional gain, the proportional response is adjusted such that the steady-state error decreases inversely with the proportional gain. However, if the gain is set too high, it may lead to instability in the system by causing excessive oscillations and output fluctuations. On the other hand, if the gain is too low, the system may struggle to compensate for disturbances effectively. The proportional term, or output signal, is defined by the equation:

Where is the output control signal associated with the gain and error e(t).

### 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 errors. However, the integral gain can sometimes degrade the system's response, potentially causing transient or oscillatory behaviour. The output signal from the integral term is given by equation (3).

Where is the output signal, is the gain, and error e(t) is integrated over t.

### 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 , the derivative gain. The derivative term is defined in equation (4). In a BoP system, the derivate control compensates for the ball's velocity.

Where is the output signal associated with the gain and derivative of error e(t).

## Block diagram of the system

Designing and implementing a proper control system is crucial for optimal performance. By convention, the system is split into the process and the controller. The process refers to the physical system or machine that is being controlled, and the controller computes corrective actions to minimise the error and directs the process towards achieving the desired SP.

The control system block diagram in Figure 4.6 represents the control architecture of the proposed BoP system. A PID controller manages the ball's movement along a single platform axis. Two such controllers are implemented, one for each axis, X and Y. The gains for both axes **are the same** as they share the same constraints and geometry.

A computer screen shot of a computer

Description automatically generated

Figure 4.6 BoP control System block diagram for a single axis.

The system receives setpoints **r(t)** for the desired positions of the ball in the X and Y directions, which are compared with the actual positions obtained through the position feedback **p(t)** read from the touchpad. The SP can be constant or a changing value. By changing the SP continuously with a pre-defined function, it is possible to trace trajectories of different shapes, which is discussed in detail in the section 7.1.

The error in both axes is calculated using summation blocks, where the error **e(t)** 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 signal **u(t)** 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 4.1.

The control signals are then fed into an inverse kinematics block, which translates them into appropriate motor signals **a(t)** for the stepper motors. The stepper motors actuate the platform, adjusting its tilt to correct the ball's position represented by the signal **y(t)**, thereby minimising the error. This feedback loop ensures continuous adjustment of the platform's position to maintain the desired trajectory and balance of the ball.

## Inverse Kinematics of the system

This sub-chapter describes the required inverse kinematics equations to orient the platform so that it sends the ball towards SP. Three legs support the platform, 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 Halil [6] provides an excellent explanation of modelling the manipulator and formulating the equations.

Given a unit normal vector of the platform plane, the IK equations then compute the required angles of the revolute joint [10].

A diagram of a hexagon with points and lines with Great Pyramid of Giza in the background

Description automatically generated

Figure 4.7 Kinematic sketch of 3RRS parallel manipulator described in Figure 4.1 [10]

Table 4.1 Description of symbols in Figure 4.7

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| O | Origin of the global coordinate system |
|  | Revolute Joints |
|  | Spherical Joints |
| P | Platform Plane origin |
|  | Unit normal vector of the platform plane |

Table 4.2 Inverse Kinematics parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Description** | **Value** | **Unit** |
|  | Distance from centre of base to corner | 5.08 | [cm] |
|  | Distance from centre of platform to corner | 7.93 | [] |
|  | Length of Link 1 | 4.445 | [cm] |
| ***g*** | Length of Link 2 | 9.32 | [cm] |
|  | Height of the platform plane | variable |  |
|  | The magnitude of Normal vector | variable | [-] |
|  | Normal Vector | variable | [-] |

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 P. The required rotor angles are calculated as a function of the tilt of the platform plane represented by and the height between the platforms represented by h.

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

**Unit normal Vector of the platform plane:**

**Equations for :**

Where:

**Equations for :**

Where:

**Equations for :**

Where:

# Hardware Implementation

This chapter describes the prototype's mechanical and electrical configuration and specific hardware requirements, followed by construction and assembly procedures.

## Specific hardware requirements

For the construction of the platform, the following items must be sourced or purchased.

Table 5.1 Specific Requirements Table

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Item | Qty | Cost ($) |
| Electronics | Maker Uno Microcontroller | 1 | 10 - 15 |
| Nema 17 59 Ncm Stepper Motors (Bipolar) | 3 | 30 - 40 |
| TMC2209 V2.0 Stepper Motor Drivers | 3 | 10 - 20 |
| CNC Shield | 1 | 10 |
| 30V Bench Power Supply (or any 24V power supply) | 1 | 50 |
| 5V Regulator | 3 | 10 |
| General Parts | 8.4" 4 Wire Resistive Touch Panel | 1 | 30 - 35 |
| 1" Steel Bearing Ball | 1 | 2 |
| 22mm long M3 Tie Rod | 5 | 2 |
| M3 x 6mm Threaded Inserts | 10 | 2 |
| M3 x 5mm Standoffs | 10 | 2 |
| M3 x 5mm Screws | 10 | 2 |
| M3 x 8mm Screws | 10 | 2 |
| M3 x 10mm Screws | 10 | 2 |
| M3 x 35mm Screws | 10 | 2 |
| M3 Nylon Locknuts | 10 | 2 |
| M4 x 20mm Screws | 10 | 2 |
| M4 x 25mm Screws | 10 | 2 |
| M4 Nylon Locknuts | 10 | 2 |
|  | TOTAL COST |  | ~USD 200 |

The overall budget for construction is approximately USD 200. There are some additional requirements like general mechanical tools, soldering iron, electrical wire, filament and 3D printer that are not listed in the specific requirements.

## 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 tracks the ball's position, offering a simple and reliable solution.

### Maker UNO

Maker UNO is a similar microcontroller to Arduino UNO. It has a pinout configuration identical to that of 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.

A blue circuit board with many wires

Description automatically generated

Figure 5.1 Arduino UNO pinout diagram. Maker UNO has an identical configuration.

Table 5.2 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 |

### TMC2209 v2.0

The TMC2209 is a popular stepper motor driver with a standard pinout configuration. Low noise levels, different micro-stepping options, current control and lower heat generation make it a good choice to control the stepper motors. The is set using a potentiometer screw.

From the stepper documentation,

A black circuit board with many small screws and many small screws

Description automatically generated

Figure 5.2 TMC2209 V2.0 stepper driver pinout diagram.

Table 5.3 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 5.4 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 |

### CNC shield

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

A circuit board with many components

Description automatically generated

Figure 5.3 CNC shield pinout diagram

Table 5.5 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 Figure 5.3.

### Stepper motor

NEMA 17 bi-polar stepper motors from STEPPERONLINE were used for the actuation of the platform. Cost efficiency, precise control and low noise make it an ideal choice. External positional encoders are not used. Instead, when the platform is powered off the limbs always bend down to a default position. This default position is the home position. Once the motor is powered on the driver keeps track of the rotor angle.

A black and silver square device

Description automatically generated

Figure 5.4 Nema 17 Bi-polar stepper motor.

Table 5.6 Nema 17 technical specifications

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

### Resistive touchpad[[5]](#footnote-5)

A **four-wire resistive touchpad** detects touch through two flexible, transparent resistive layers that come into contact with pressure. Each layer has electrodes along its edges: the X-axis's top layer and the Y-axis's bottom layer. 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.

## Construction and Assembly

This section details the construction steps of the BoP. The process began with purchasing the listed components in the section 5.1, followed by 3D printing the CAD models. PETG filament was used for printing. It took about 6 days of continuous printing to make all the parts.

### 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.

A three dimensional printer with a black and white object

Description automatically generatedA white plastic object on a table

Description automatically generated

Figure 5.5 Picture taken during printing the top platform (L) Printed parts (R)

### Electronics assembly

MakerUNO is the main control unit. The CNC shield is compatible with Maker UNO and is stacked on top of it as shown in the Figure 5.6. This eliminates the need to use protoboard and jumper wires, making the electronics highly organized and easy to handle. Since the electrical configuration follows a simple plug-and-play type system, additional schematics have not been included. Figure 5.6 shows how the components are attached together.

A circuit board with many colorful wires

Description automatically generatedA red circuit board with colorful wires

Description automatically generated

Figure 5.6 CNC shield with drivers connected to UNO (L) Micro stepping jumper pins (1/16th configuration) (R)

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

### 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. This step 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, four retainer clips clipped the resistive touchpad onto the platform frame.

A machine on a table

Description automatically generated

Figure 5.7 Partially assembled platform

A white device with wires and a computer

Description automatically generated

Figure 5.8 Assembled 3RRS PM

# Simulink Modelling

This chapter details the implementation of the Simulink system, describing the BoP system. The chapter begins by formulating a model describing the motion of the ball as a function of the tilt of the platform plane. The derived equations are then used to create Simulink blocks. The Simulink model is used to tune the PID gains, which will then be applied to the real system. The simulation was run using the ODE45 solver.

## 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 6.1. These systems will share the same equations but with different variables. Index 1 will refer to the system observed in the XZ plane of the room, while index 2 will correspond to the system viewed in the YZ plane.

The equations will be derived for the first system and later translated into their equivalent form for the second system. [4]

A diagram of a rectangular object with a circle and a circle with a circle

Description automatically generated with medium confidence

Figure 6.1 A three-dimensional system is represented as a pair of a two-dimensional system [4]

The goal is to model the motion of the ball in the 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 6.2.

A diagram of a triangle and a circle

Description automatically generated

Figure 6.2 Free body diagram of the ball in two dimension [4]

**Lagrangian Method: Equations of Motion**

The Lagrangian Method derives the equation of motion through the relation of the 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 [4].

Table 6.1 Ball motion model parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | Description | Value | Unit |
|  | Mass of the ball | 0.067 | [kg] |
|  | The rotational inertia of the ball | 0.0000435 | [] |
|  | The radius of the ball | 0.0125 | [m] |
| **g** | Acceleration due to gravity | 9.81 |  |
|  | Tilt angles of XZ and YZ planes | variable | [radian] |
|  | Position coordinates of the ball in XY plane | variable | [m] |

**Lagrangian Equation:**

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

**Kinetic Energy:**

Where:

Thus, the total kinetic energy becomes:

Potential Energy:

**Lagrangian:**

**Equation of Motion (x-direction):**

Substituting these into Equation 4.1, the differential equation becomes:

Simplifying for , the equation of motion is derived as:

**Equation of Motion (y-direction):**

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

The equation 13 and equation 14 describe the motion of the ball as a function of the angles and respectively. As illustrated in the Figure 6.1 and Figure 6.2, is the angle between the platform plane and the imaginary plane parallel to the base. The calculation of these angles is done as follows:

A diagram of a straight line

Description automatically generated

Figure 6.3 Figure describing the angle , GE is the platform plane, and HF is the imaginary plane parallel to the base.

If is the projection of the normal vector in the XZ plane, the length L of is given by:

The is component is fixed as a unit vector, is the required tilt input from the PID block.

The angle is given by,

The angle is given by,

Applying the same for the YZ plane,

## Simulink block diagrams

The following are the snapshots of the Simulink block diagrams:

A diagram of a computer flowchart

Description automatically generated

Figure 6.4 Simulink Model of the system

A diagram of a flowchart

Description automatically generated

Figure 6.5 Ball motion model block

A diagram of a computer flowchart

Description automatically generated

Figure 6.6 Angle calculation block

## PID tuning – simulation

There are multiple approaches to tuning PID parameters in MATLAB. Using the PID tuner function, which is a part of Simulink Control design toolbox is the most straight forward approach. However, PID tuning in this approach is not possible without including a first-order derivative filter. The real system’s performance severely degrades with the inclusion of a derivative filter with a lower coefficient. Hence manual tuning is performed by following these steps:

**Step 1:** Set the derivative and integral gains to zero. Gradually increase the proportional gain until the system starts to oscillate.

**Step 2:** Set the derivative gain to a value proportional to the proportional gain and adjust it to dampen oscillations and achieve a suitable system response.

**Step 3:** Adjust the integral gain to an appropriate value to eliminate any steady-state error without destabilizing the system.

To replicate the real system Discrete-time setting was used. The sample time is the same as the real system's average sampling time, which is ~0.020 seconds.

A screenshot of a computer

Description automatically generated

Figure 6.7 Simulink PID block settings

# Software Implementation

This chapter describes the implementation of the control software for the BoP system. The control software is run on Maker UNO. The touchpad data is sent via the serial port to the host system to obtain the test results. The host system receives serial data and plots the graph using Python code. The serial communication is not considered for algorithm design.

A diagram of a computer program

Description automatically generated

Figure 7.1 Control Algorithm describing the process.

The following is a detailed description of the flowchart shown in Figure 7.1:

1. **Start**: The process begins. *(Next Step:* 2*)*
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. *(Next Step:* 3*)*
3. **Ball Detection**: The system checks whether a ball is on the platform. Depending on the outcome, the following happens:
   * If not detected, the system returns to *Step* 2.
   * If detected, the process continues to *Step* 4.
4. **Read Position**: The current position of the ball is identified by reading the touchpad measurements (section **Error! Reference source not found.**). *(Next Step:* 5*)*
5. **Calculate Error**: The error is calculated, which is the difference between the ball's current position and the SP of the platform. This error indicates how far the ball is from the SP in [X, Y] vector format. *(Next Step:* 6*)*
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 as explained in the section 4.2. This new vector represents the required orientation to move the ball toward the center. *(Next Step:* 7*)*
7. **Perform PID Gain to the New Normal Vector**: A PID controller is implemented for each axis [X and Y] (section 4.3). The output of the PID controller is constrained between the range [-0.25, 0.25] to prevent the platform from over-tilting as described in section 4.2. *(Next Step:* 8*)*
8. **Use IK to Calculate Motor Angles**: Pre-defined inverse kinematics equations (section 4.2) are utilized to translate the desired platform tilt, represented by the normal vector, into specific angles for each motor that controls the platform. *(Next Step:* 9*)*
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. *(Next Step:* 10*)*
10. **15ms Wait**: The system introduces a wait time of 15 milliseconds for the steppers to reach the desired angle. The code moves to the next step even if the required motor angles are not reached after 15ms. *(Next Step:* 11*)*
11. **Loop**: If the ball is detected by the touchpad, the loop continues. If the ball is not detected, the platform returns to home. *(Next Step:* 2*)*

## Reading touchpad data

The position data is acquired from an analog signal from the resistive touchpad. The signal conversion is performed using the Adafruit Touchscreen library[[6]](#footnote-6). The signal from the touchpad contains noise but of a minimal amount. However, an initial manual calibration is required.

Implementing a filter could improve reading. A low pass filter was tested, and it was observed that the system becomes less responsive to sudden changes in position; hence, a more sophisticated filter needs to be investigated and implemented. This is beyond the scope of this thesis.

## PID tuning – real system

As mentioned in the section 4.1, 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. Depending on the response increase or decrease value.
3. Tune the values until stability is achieved.
4. Introduce to fix the steady-state error.

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

## 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. A continuous curve in the two-dimensional plane can be described as a function of two parametrized variables, x and y.

Where and represent the coordinates of the curve as a function of time t.

# 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.

## Assembled platform

(Final pictures will be added)

## PID gains

Initially the PID gains are tuned in Simulink model by following the steps outlined in section 6.3. After obtaining the theoretical values, using the approach described in section 7.2 PID gains for the real model were identified. The following sections describe the results obtained with the PID gains in Table 8.1.

Table 8.1 PID Gain Values

|  |  |  |  |
| --- | --- | --- | --- |
| **System** |  |  |  |
| **Simulink** | 0.20 | 0.0008 | 0.35 |
| **Real** | 0.1670 | 0.00085 | 0.1700 |

## Step Response testing

Step response of both the real and simulated system is tested by placing the ball at the coordinates [400,400] and setting the SP 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 .

A graph with a line going up

Description automatically generated

A graph with a line going up

Description automatically generated

Figure 8.1 X axes step responses comparison: Real system (top) and simulated system (bottom)

A graph with a red line

Description automatically generated

A graph of error and error

Description automatically generated

Figure 8.2 Y axes step responses comparison: Real system (top) and simulated system (bottom)

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 in Figure 8.1 and Figure 8.2 the following can be determined:

Table 8.2 Step Response Analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Simulation** | | **Real** | |
|  | **X Axis** | **Y Axis** | **X Axis** | **Y Axis** |
| **Settling time ()** | 3.2 sec | 3.4 sec | 3.9 sec | 4 sec |
| **Overshoot (M)** | 0 | 0 |  |  |
| **Steady-state error ()** | 0 | 0 |  |  |
| **Peak [mm]** | 0 | 0 |  |  |
| **Stability** | Stable | Stable | Stable | Stable |

Initially the simulated and real results will be analyzed 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.

## Frequency response

In the real system, 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.

**A graph of a diagram

Description automatically generated with medium confidence**

Figure 8.3 Frequency response of the real system

From the response plotted in Figure 8.3 it can be observed that the real system had a lag of . 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.

## 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.

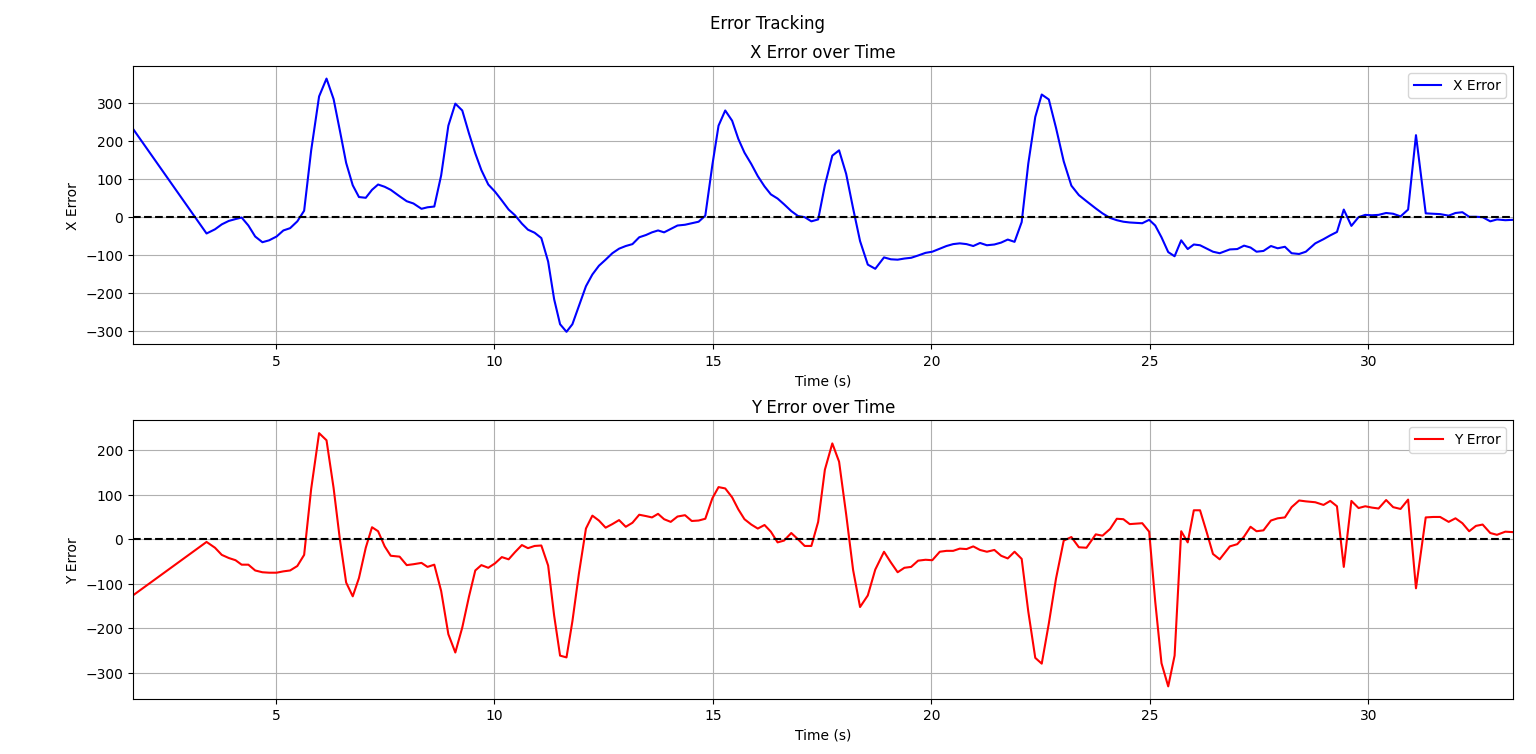


Figure 8.4 Disturbance rejection of the real system

In Figure 8.4, the peaks correspond to the moments when the ball was subjected to external force or disturbance. The average settling along both axes is about , which is consistent with the step response analysis in section 8.3. 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 is also evident, which is consistent with the results of the step response analysis.

## Creating trajectories

In this section the different shapes will be traced by controlling the movement of the ball as explained in section 7.1.

### Circle

By incrementing the angle within range at a constant rate and applying the following equations (Eq. 21 and 22) a circular reference is created.

A graph with a circle in the middle

Description automatically generated

Figure 8.5 Circle trajectory traced by the ball motion

A diagram of a waveform

Description automatically generated

Figure 8.6 Plot of delay and error – Circle trajectory

### Infinity

By incrementing the angle within range at a constant rate and applying the following equations (Eq. 22 and 23) a reference infinity curve is created.

A graph with a line drawn in red and blue

Description automatically generated

Figure 8.7 Infinity trajectory traced by the ball motion

A graph of a waveform

Description automatically generated with medium confidence

Figure 8.8 Plot showing delay and error – Infinity trajectory

### **Square**

A graph with a red line and blue lines

Description automatically generated

Figure 8.9 Square trajectory traced by the ball motion

A graph of a graph

Description automatically generated

Figure 8.10 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 8.4, is also evident here.

## Results – discussion

The physical prototype functioned as intended, with well-controlled vibration and noise levels. This can be credited to the appropriate selection of drivers (section 8.1). Both the simulated and physical systems were able to stabilize the ball, confirming the reliability of the control design (section 8.3). The simulated and physical systems were nearly identical. However, minor deviations were observed in the real system due to non-linearities like noise in touchpad reading and friction affecting the ideal behaviour (section 8.3).The average settling time in the real system was recorded to be around 3.8 seconds. It is adequate but further improvements can be made (section 8.3).A slightly higher-than-anticipated steady-state error of about 25 mm was observed in the Y-axis of the real system.(section 8.3)Some jittering was noted, likely because of the ball's natural instability, where even minor platform tilts cause rolling, and the system must constantly compensate for it. The noise in the input from touchpad might also be a reason for the jittering. (section 8.3). A lag of approximately 2 seconds was recorded, which, while satisfactory, can be improved (section 8.4). The real system responded well when subjected to random external disturbances, maintaining stability within acceptable limits (section 8.5). The system satisfactorily traced predefined trajectories, including complex shapes like infinity (section 8.6).

Based on the results it is safe to conclude the stated objectives in section 0 has been successfully achieved.

# Conclusion

To conclude the thesis the research questions that were stated in section 0 will be answered with justification based on the implemented work.

**Is the linear control methodology adequate for satisfactory performance in a non-linear system like a BoP system?**

A satisfactory behaviour can be defined as a performance slightly less than the optimal behaviour. In this context, **yes** linear control methodology is adequate for satisfactory performance in a non-linear system. The results discussion in section 8.7 proves that a satisfactory performance can be achieved with only linear control strategy. It is also true, that this might not be the case for all the systems. There were some shortcomings in the performance of the physical prototype like the jittering. The jittering is likely a result of nonlinearity, more specifically the noise in the touchpad readings. Accounting for this might improve the performance of the system and take it closer to the ideal system. In conclusion, while a linear control approach can provide satisfactory performance, more advanced non-linear control techniques may be necessary to achieve results that approach the ideal performance. Next chapter introduces some non-linear controllers that can achieve better performance.

**Is it possible to create a digital twin of the prototype? How beneficial is such a system in meeting the required objectives?**

A digital twin is a virtual model of a physical system that can be used to simulate the behaviour of the physical system. In the context of the BoP system, a digital twin could be used to simulate the behaviour of the ball on the plate, and to test different control algorithms. This would allow for the development of more robust and efficient control algorithms including the tuning of a PID system without the need to physically test the prototype.  The digital twin could also be used to predict the behaviour of the physical system under different conditions, such as different ball masses, different plate angles or different input signals. In this work, a nearly identical Simulink model of the physical system was successfully implemented (section 6). The key difference is that the simulated model doesn’t account for non-linearities of the physical system. The simulated model serves as an ideal system. Its performance can be used as a reference but cannot be replicated in the physical system. In conclusion a nearly identical simulation model can indeed shorten the development time and costs related to physical testing.

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

There is two ways to achieve this, one by sending a continuous signal and the other by sending discrete signals of a set of vertices at regular intervals. The former method proved to be more efficient and simpler to implement. Trajectories of different shapes were traced successfully. The results are described in section 8.6. The precision can certainly be improved with a more robust control strategy, some of which are mentioned in the next chapter.

# Further Work

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 that can be researched and implemented, which will be discussed in this chapter. These alternative controllers can be implemented in the existing system by making modifications solely to the software, given that the compute demand is within limits of the processor used in the project.

### Model predictive controller

Model predictive control (MPC) is an advanced method of non-linear [process control](https://en.wikipedia.org/wiki/Process_control). MPC predicts the future system behaviour to optimize control actions. In context of a BoP system the MPC works by predicting the ball’s future movement using a model of the system and calculating a sequence of tilt adjustments to maintain balance while minimizing the cost function, which in this case is the predicted error over the control horizon.

The study by Krzysztof Zarzycki and Maciej Ławrynczuk [1] at Warsaw University of Technology describes the implementation of such a controller. The findings of their work demonstrate that MPC significantly outperforms traditional PID controllers in terms of the performance metrics described in section 3.2. Another study by Federico Dal Cero [11] describes the implementation of such a controller but for a 6 DOF platform.

### Fuzzy logic controller

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. [12]

In context to the ball on a plate system, the work by Strand Filip and Wahlund Martin [12] 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.

# References

|  |  |
| --- | --- |
| [1] | K. Zarzycki and M. Ławryńczuk, "Fast Real-Time Model Predictive Control for a Bal-on-a-plate process," *Sensors,* pp. 1, 11-15, 2021. |
| [2] | D. A. G. &. J. J. Núñez, "Control of a ball-and-plate system using a State-feedback controller.," 2020. |
| [3] | V. Tudić, D. Kralj, J. Hoster and T. Tropčić, "Design and Implementation of a Ball-Plate Control System and Python Script for Educational Purposes in STEM Technologies," *Sensors,* 2022. |
| [4] | A. H. Frank and T. Morgan, "Construction and theoretical study of ball balancing platform," KTH Royal Institute of Technology, Stockholm, Sweden, 2019. |
| [5] | E. Lakzaei, "Balancing ball on a plate - Final Report," Eindhoven University of Technology, Eindhoven, 2020. |
| [6] | H. Tetik, "Modelling and Control of a 3-Rrs Parallel Manipulator," Izmir Institute of Technology, Izmir, 2016. |
| [7] | H. Tetik, R. Kalla, G. Kiper and S. Bandyopadhyay, "Position Kinematics of a 3-RRS Parallel Manipulator," 2016. |
| [8] | L. Spacek, J. Vojtesek, F. Gazdos and T. Kadavy, "Ball & Plate Model For Robotic SystemBall & Plate Model For Robotic System," in *32nd Conference on Modelling and Simulation*, 2018. |
| [9] | H. Bang, "Implementation of a Ball and Plate Control System Using Sliding Mode Control," *IEEE Access,* pp. 1,6,7, 2018. |
| [10] | S. Liu, Z. Zhencai and B. Zi, "Dynamics of 3-DOF spatial parallel manipulator with flexible links," in *Springer*, Beijing, China, 2010. |
| [11] | F. D. Cero, "Ball and Plate MPC Control of a 6 DOF Stewart Platform," Universita degli Studi di Padova, Padua, Italy, 2022. |
| [12] | S. Filip and W. Martin, "Control of the ball and plate - A comparative study of controllers," KTH Royal Institute of Technology, Stockholm, 2022. |
| [13] | Y. Chen, System simulation techniques with MATLAB and Simulink, John Wiley & Sons, 2013. |

****APPENDIX****

****Table of Figures****

[Figure 2.1 BoP system with camera-based feedback control and two actuators [3] 3](#_Toc188819723)

[Figure 2.2 BoP system with two actuators and a resistive touchpad-based feedback built at KTH Royal Institute of Technology [4] 4](#_Toc188819724)

[Figure 2.3 Hexapod platform (Stewart Platform). Source: Wikipepedia3 4](#_Toc188819725)

[Figure 2.4 BoP system with camera-based feedback control and three actuators built at TU Eindhoven [5] 5](#_Toc188819726)

[Figure 2.5 Kinematic sketch of a 3RRS parallel manipulator [7] 5](#_Toc188819727)

[Figure 2.6 BoP system modelled in the ABB RobotStudio with LQR controller [8] 6](#_Toc188819728)

[Figure 4.1 3RRS BoP system CAD design by Aaed Musa 9](#_Toc188819729)

[Figure 4.2 Process overview of the system 10](#_Toc188819730)

[Figure 4.3 Figure representing the position of the ball X, error , and the normal vector . It is assumed that at home position, 10](#_Toc188819731)

[Figure 4.4 Scaling X error values to be in the range [-0.25,0.25] 11](#_Toc188819732)

[Figure 4.5 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. 11](#_Toc188819733)

[Figure 4.6 BoP control System block diagram for a single axis. 13](#_Toc188819734)

[Figure 4.7 Kinematic sketch of 3RRS parallel manipulator described in Figure 4.1 [10] 14](#_Toc188819735)

[Figure 5.1 Arduino UNO pinout diagram. Maker UNO has an identical configuration. 18](#_Toc188819736)

[Figure 5.2 TMC2209 V2.0 stepper driver pinout diagram. 19](#_Toc188819737)

[Figure 5.3 CNC shield pinout diagram 20](#_Toc188819738)

[Figure 5.4 Nema 17 Bi-polar stepper motor. 21](#_Toc188819739)

[Figure 5.5 Picture taken during printing the top platform (L) Printed parts (R) 22](#_Toc188819740)

[Figure 5.6 CNC shield with drivers connected to UNO (L) Micro stepping jumper pins (1/16th configuration) (R) 23](#_Toc188819741)

[Figure 5.7 Partially assembled platform 24](#_Toc188819742)

[Figure 5.8 Assembled 3RRS PM 24](#_Toc188819743)

[Figure 6.1 Three-dimensional system represented as a pair of two-dimensional system [4] 25](#_Toc188819744)

[Figure 6.2 Free body diagram of the ball in two dimension [4] 26](#_Toc188819745)

[Figure 6.3 Figure describing the angle , GE is the platform plane, and HF is the imaginary plane parallel to the base. 28](#_Toc188819746)

[Figure 6.4 Simulink Model of the system 29](#_Toc188819747)

[Figure 6.5 Ball motion model block based on the equation 13 derived in section 0 29](#_Toc188819748)

[Figure 6.6 Angle calculation block based on the equation 16 derived in section 0 29](#_Toc188819749)

[Figure 6.7 Simulink PID block settings 30](#_Toc188819750)

[Figure 7.1 Control Algorithm describing the process. 31](#_Toc188819751)

[Figure 8.1 X axes step responses comparison: Real system (top) and simulated system (bottom) 35](#_Toc188819752)

[Figure 8.2 Y axes step responses comparison: Real system (top) and simulated system (bottom) 36](#_Toc188819753)

[Figure 8.3 Frequency response of the real system 37](#_Toc188819754)

[Figure 8.4 Disturbance rejection of the real system 38](#_Toc188819755)

[Figure 8.5 Circle trajectory traced by the ball motion 39](#_Toc188819756)

[Figure 8.6 Plot of delay and error – Circle trajectory 39](#_Toc188819757)

[Figure 8.7 Infinity trajectory traced by the ball motion 40](#_Toc188819758)

[Figure 8.8 Plot showing delay and error – Infinity trajectory 40](#_Toc188819759)

[Figure 8.9 Square trajectory traced by the ball motion 41](#_Toc188819760)

[Figure 8.10 Plot showing delay and error - square trajectory 41](#_Toc188819761)

****List of Tables****

[Table 4.1 Description of symbols in Figure 4.7 15](#_Toc188819762)

[Table 4.2 Inverse Kinematics parameters 15](#_Toc188819763)

[Table 5.1 Specific Requirements Table 17](#_Toc188819764)

[Table 5.2 Maker UNO important technical specification. 18](#_Toc188819765)

[Table 5.3 TMC2209 Micro stepping Configuration. 19](#_Toc188819766)

[Table 5.4 TMC2209 technical specifications. 19](#_Toc188819767)

[Table 5.5 Cross-referencing of Arduino pins 20](#_Toc188819768)

[Table 5.6 Nema 17 technical specifications 21](#_Toc188819769)

[Table 6.1 Ball motion model parameters 26](#_Toc188819770)

[Table 8.1 PID Gain Values 35](#_Toc188819771)

[Table 8.2 Step Response Analysis 36](#_Toc188819772)

****Code – Arduino Controller****

1. // Forward declarations

2. void setup();

3. void loop();

4. void calculateIKPositions(double hz, double nx, double ny, long\* positions);

5. void moveTo(long\* positions);

6. void PID(double setpointX, double setpointY);

7.

8. // Stewart Platform Control Code

9.

10. // Libraries

11. #include <AccelStepper.h>

12. #include <InverseKinematics.h>

13. #include <MultiStepper.h>

14. #include <stdint.h>

15. #include <TouchScreen.h>

16. #include <math.h>

17.

18. // Machine Parameters

19. Machine machine(2, 3.125, 1.75, 3.669291339);     // (d, e, f, g) lengths of the machine

20. TouchScreen ts = TouchScreen(A1, A0, A3, A2, 0);  // Touchscreen pins (XGND, YGND, X5V, Y5V)

21.

22. // Stepper Motor Definitions

23. AccelStepper stepperA(1, 2, 5);  // (driver type, STEP, DIR) Driver A

24. AccelStepper stepperB(1, 3, 6);  // (driver type, STEP, DIR) Driver B

25. AccelStepper stepperC(1, 4, 7);  // (driver type, STEP, DIR) Driver C

26. MultiStepper steppers;           // Multi-stepper control instance

27.

28. // Stepper Motor Variables

29. long pos[3];                     // Target positions for each stepper motor

30. int ENA = 8;                     // Enable pin for the drivers

31. double angOrig = 206.662752199;  // Original angle for each leg

32.

33. // Speed Control Variables

34. double speed[3] = {0, 0, 0};     // Current speed of stepper motors

35. double speedPrev[3];             // Previous speed of stepper motors

36.

37. // Touchscreen Variables

38. double Xoffset = 500;  // X offset for touchscreen center

39. double Yoffset = 500;  // Y offset for touchscreen center

40.

41. // PID Control Variables

42. double kp = 0.1670, ki = 0.00085, kd = 0.1700;  // PID constants

43. double error[2] = {0, 0};                  // Current error for X and Y directions

44. double errorPrev[2];                       // Previous error for X and Y directions

45. double integr[2] = {0, 0};                 // Integral terms for X and Y

46. double deriv[2] = {0, 0};                  // Derivative terms for X and Y

47. double out[2];                             // PID output for X and Y

48. double rawErrorX, rawErrorY;               // Variables to store raw error values before normalization

49.

50. // Miscellaneous Variables

51. double angToStep = 3200.0 / 360;  // Angle-to-step conversion factor (steps per degree)

52. bool detected = false;            // Ball detection flag

53. long timeI;                       // Timing variable for delay

54. unsigned long lastTime = 0;       // Timing for PID updates

55.

56. void setup() {

57.   Serial.begin(115200);

58.

59.   // Configure MultiStepper

60.   steppers.addStepper(stepperA);

61.   steppers.addStepper(stepperB);

62.   steppers.addStepper(stepperC);

63.

64.   // Enable Pin Configuration

65.   pinMode(ENA, OUTPUT);

66.   digitalWrite(ENA, HIGH);  // Disable drivers initially

67.   delay(1000);              // Allow user to reset the platform

68.   digitalWrite(ENA, LOW);   // Enable drivers

69.

70.   // Set initial speeds for homing

71.   stepperA.setMaxSpeed(600);

72.   stepperB.setMaxSpeed(600);

73.   stepperC.setMaxSpeed(600);

74.   stepperA.setAcceleration(300);

75.   stepperB.setAcceleration(300);

76.   stepperC.setAcceleration(300);

77.

78.   // Calculate home position

79.   long homePos[3];

80.   calculateIKPositions(4.25, 0, 0, homePos);

81.

82.   // Move to home position

83.   steppers.moveTo(homePos);

84.   steppers.runSpeedToPosition();  // This blocks until all steppers reach position

85. }

86.

87. void loop() {

88.   PID(0, 0);  // Run PID control with setpoints (X: 0, Y: 0)

89. }

90.

91. // Function to calculate inverse kinematics positions

92. void calculateIKPositions(double hz, double nx, double ny, long\* positions) {

93.   for (int i = 0; i < 3; i++) {

94.     positions[i] = round((angOrig - machine.theta(i, hz, nx, ny)) \* angToStep);

95.   }

96. }

97.

98. // Function to Move the Platform

99. void moveTo(long\* positions) {

100.   stepperA.moveTo(positions[0]);

101.   stepperB.moveTo(positions[1]);

102.   stepperC.moveTo(positions[2]);

103.

104.   // Run Steppers Incrementally

105.   stepperA.run();

106.   stepperB.run();

107.   stepperC.run();

108. }

109.

110. void PID(double setpointX, double setpointY) {

111.   TSPoint p = ts.getPoint();  // Read touchscreen position

112.   long positions[3];          // Local array for positions

113.   static unsigned long lastUpdateTime = millis();  // Last update time

114.   unsigned long currentTime = millis();            // Current time

115.   double deltaTime = (currentTime - lastUpdateTime) / 1000.0;  // Time difference in seconds

116.

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

118.     detected = true;

119.

120.     // Calculate Errors

121.     rawErrorX = Xoffset - p.x - setpointX;

122.     rawErrorY = Yoffset - p.y - setpointY;

123.     double errorZ = 4.25;

125.     // Normalize Error Vector

126.     double magnitudeX = 1760;

127.     double magnitudeY = 1520;

128.     double normX = rawErrorX / magnitudeX;

129.     double normY = rawErrorY / magnitudeY;

130.

131.     // Compute error magnitude for dynamic speed adjustment

132.     double errorMagnitude = sqrt(rawErrorX \* rawErrorX + rawErrorY \* rawErrorY);

133.

134.     // Compute PID Terms for X and Y

135.     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. }

****Code – Graph plotter****

****Attachments****

1. https://en.wikipedia.org/wiki/Inverted\_pendulum [↑](#footnote-ref-1)
2. https://www.aaedmusa.com/projects/project-three-sng7y-gaslp [↑](#footnote-ref-2)
3. https://en.wikipedia.org/wiki/Parallel\_manipulator [↑](#footnote-ref-3)
4. https://www.aaedmusa.com/ [↑](#footnote-ref-4)
5. https://www.sparkfun.com/datasheets/LCD/HOW%20DOES%20IT%20WORK.pdf [↑](#footnote-ref-5)
6. https://github.com/adafruit/Adafruit\_TouchScreen [↑](#footnote-ref-6)