University of Tripoli
Faculty of Engineering
Department of Electrical and Electronic Engineering

Modeling, Control, and Experimental Validation of a Ball and Plate System

A project submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Electrical and Electronic Engineering

Prepared by:
Mohammed S. Baayou

Supervised by:
Dr. Ali S. ElMelhi

Spring 2023

Dedication

To my father, may Allah have mercy on him,
and to my mother, may Allah protect her

Acknowledgements

First of all, thanks to Allah for giving me the ability, strength and aptitude to complete this project.

I would like to extend my thanks and gratitude to my supervisor **Dr.Ali Elmelhi**, for his advises, suggestions, patience and encouragement throughout this project.

I am deeply and forever obliged to thank my great family for their love, caring, support and encouragement throughout my entire life.

Finally, I would like to thank to all the good friends that I have for their support and help, with special appreciation for **Mahmoud Karah** and **Salman Ajweeli**.

Abstract

This study presents an exploration of a real-time ball-on-plate system, encompassing the implementation, modeling, analysis, and control aspects. The primary objectives are to build a physical system, develop a dynamic model, design a PID controller, and compare simulation results with experimental outcomes. The initial phase involves the mechanical and electonical parts of the ball-on-plate system, discussing resistive touchscreen technology for ball position detection,filtering and sending.The modeling stage delves into the development of a dynamic model for the ball-on-plate system. The main parameters that affect the behavior of the system are determined using Euler's langrangian equation. The model is then simplified and linearized.The study analyzes the model, leading to the design and implementation of a PID controller. The manual tuning process is documented, and the resulting con-

troller parameters are established. Challenges encountered during this phase, including unsimilarity between simulation and real-world responses, are discussed. A comparative analysis between simulated and experimental outcomes provides valuable insights into the system's performance and the effectiveness of the designed PID controller.

Outline and objectives

Outline

The Outline of this Project is structured as follows:

- **Chapter 1** is an introduction contains abrief history of control theory the motivation to select our system, an literature review of previous works and overall project objectives.
- **Chapter 2** describes the mechanical and electrical setup of the BPS selected and are used in this project.
- **Chapter 3** presents the mathematical modeling work, the linearization and simplifcation process, open loop validation and geometrical analysis to the linkage and servomotor in overall system.
- **Chapter 4** presents the design process of linear controllers including the theory of these controllers.
- In **Chapter 5**, simulations and experimental results of each approach that have been raised in are presented then discussed.
- Conclusions and Future Work of the project are presented in **Chapter 6**.

Objectives

The main objective of the project is to develop a dynamical model for 2-DOF Ball and plate system, study and develop a full mathematical model of the overall system, design PID controller, and finally implement and fabricate a lightweight BPS based on available parts in Libya and build it to test the PID controller actual performance

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and overview

Control theory is a multidisciplinary field that plays a pivotal role in engineering and technology, providing a systematic framework for understanding, analyzing, and designing systems to achieve desired behaviors. At its core, control theory aims to regulate the output of a system, ensuring it conforms to a specified reference or trajectory. This field has far-reaching applications, ranging from industrial processes and aerospace systems to biological and economic systems.

The origins of control theory can be traced back to the early 19th century, with the advent of steam engines and the industrial revolution. Engineers faced challenges in maintaining the stability and efficiency of these complex mechanical systems. James Clerk Maxwell, a Scottish physicist, made early contributions with his Paper On governor—a mechanical device designed to regulate the speed of steam engines [1]. This marked the beginning of a systematic approach to controlling systems.

In the early 20th century, Norbert Wiener, a mathematician, and engineer, introduced the concept of cybernetics—a term derived from the Greek word for "steersman." Wiener's work focused on feedback systems and communication in both biological and mechanical systems. His seminal work, "Cybernetics: Or Control and Communication in the Animal and the Machine" (1948) [2], provided a theoretical foundation for control theory.

Simultaneously, Aleksandr Lyapunov, a Russian mathematician, developed the theory of stability, which became a cornerstone in the analysis of dynamic systems. Lyapunov's stability theory laid the groundwork for understanding when a system would remain stable over time, a critical aspect of control system design [3].

The mid-20th century witnessed a surge in the formalization of control theory. Engineers and mathematicians began developing mathematical models and techniques to analyze and design control systems. Notable contributors during this period include Rudolf Kalman, who introduced the Kalman filter for state estimation [4], and Richard Bellman, who pioneered dynamic programming as a method for solving optimization problems in control systems [5]. The advent of computers in the 1950s and 1960s revolutionized control theory by enabling the implementation of complex algorithms for real-time control. This era also saw the rise of the field of optimal control.

In the latter half of the 20th century, control theory continued to evolve with the introduction of robust control techniques, adaptive control strategies, and the application of control theory to non-linear systems. The field expanded its reach into aerospace engineering, telecommunications, and other emerging technologies.

Today, control theory remains a dynamic and evolving field, with the ongoing researches into advanced control methodologies, researchers are in dire need of a real system on which to test advanced control methods and thesis. The implementation of control systems in unstable physical systems is a critical aspect of engineering, as it addresses the need to stabilize systems that are prone to catastrophic failures. Textbooks on control systems often feature example problems involving such unstable systems, providing students with opportunities to study their behavior and explore various methods for designing and implementing control algorithms.

Two well-known examples of unstable systems used experimentally are the inverted pendulum and ball balancing systems on beams that shown in figure 1.1 or ball and plate. These systems serve as valuable educational tools, allowing students to understand the challenges associated with instability and the importance of con-

Figure 1.1: Ball and Beam system

trol in maintaining stability. The study of these systems provides a foundation for exploring different control strategies and algorithms.

The ball and plate system (BPS) represents a extended and generalized version of the traditional ball and beam. In this system, a ball can freely roll on a rigid platform, and the inclination of the platform can be independently manipulated in two directions. The rigid plate, often equipped with a resistive touch panel or a camera, measures the position of the ball with respect to the center or a predetermined point on the plate. This system, by its nature, is inherently unstable, as even a minor disturbance can cause the ball to deviate significantly from its stationary position. Moreover, it lacks the ability to naturally return to equilibrium without external force intervention, which necessitating control methods for stability.

Everything we mentioned previously, in addition to the fact that the system is easily transportable, affordable, safe, has few malfunctions, scalable and controllable makes the BPS an ideal practical test environment for evaluating control methods in real-world scenarios. The main goal of the system is to keep the ball balanced on the plate within set boundaries or specific points. When the ball goes beyond these

limits, a controller steps in, adjusting the plate's tilt in two directions to bring the ball back to its intended position.

To work effectively with the BPS, creating a precise nonlinear model is crucial for implementing model-based controllers. The Lagrange-Euler method is commonly used to derive the system's nonlinear equations by considering the kinetic and potential energies. Kinetic energy accounts for both linear and angular motions, while potential energies are calculated for all rigid bodies in the system. This method ensures a thorough understanding of the system's dynamics, facilitating the design and optimization of controllers tailored to its specific behavior. In our thesis we will work to design three different controllers to control the system theoretically and we will implement a PID controller to control the system experimentally then we will make a performance response comparison between these controllers.

1.2 Literature review

In general the previous works on ball-on-plate system can be classified based on several categories, Based on the type of the sensor used, based on actuation and linkage prototype or based on the controllers used to stabalize the system.

In [6], [7] Awtar and Ham introduced a prototype with a resistive touch screen and servo-motors for plate deflection. Another method in[8], [9], [10] and [11] used a top-positioned camera for ball location extraction, while Debono [12] employed a camera and computer algorithm for faster acquisition. Dual pneumatic rotary cylinders in [13] provided a low-cost solution with good control accuracy, though with increased design complexity. Designs using metallic balls were common, but Das and Roy [14] introduced a hollow plastic ball, automatically decoupling input parameters.

In the realm of control strategies for ball-on-plate systems, various approaches have been explored. Different controllers offer unique advantages and challenges in achieving system stability. Proportional-Integral-Derivative (PID) controllers, being a common choice, have been implemented in several studies [15], [16], [17], [18] and

[19]. Optimal control techniques have also been investigated. In [20], Hooshang Jafari. explored LQG control strategies, considering the kalman filter estimator in the system. Model Predictive Control (MPC) has been another area of interest, as showcased by Krzysztof Zarzycki and Maciej Ławryńczuk in [21], where MPC was employed for a comprehensive comparison with PID LQR controllers.

Chapter 2

System Setup and implementation

This chapter describes the details of a mechanical BPS including a description of its mechanical parts, electrical and software that is used to control the system.

2.1 Mechanical structure

In order to balance a ball on a flat platform, there were several options and several designs that served this purpose, and each design had advantages and disadvantages that must be chosen between. Based on degree of freedom of the plate, Figure 2.1 demonstrate a different types of designs. to reduce the number of actuators and reduce the level of complexity in the platform, we chose the 2-DOF design since 2 is so enough to control the both axis of the plate. To achieve this design six main parts was needed, the base, the joint, rods, servo holder, the plate and the ball.

- **Base:** For the base a black rectangular piece of wood was selected with dimensions of 45 x 19.5 x 2 cm, this base serves as a main base on which the rest of the mechanical and electronic parts stand.
- **Joint:** the pivotal connection allowing the plate to move around two axes is crucial.Two types of joints were considered shown in figure 2.2: the universal joint and the ball joint. The ball joint was chosen for its superior flexibility, enabling smooth and responsive movements of the plate. This decision

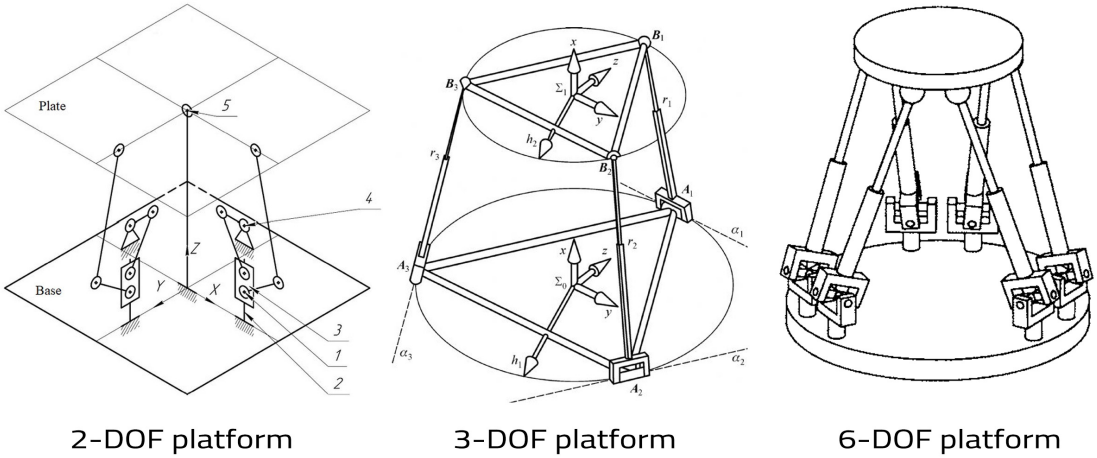


Figure 2.1: A variety of Ball and Plate System platform types based on the degree of freedom

contributes to the overall adaptability and performance of the ball-on-plate system.

Figure 2.2: Two types of joints considered in the design - universal joint and ball joint

- **Rods:** A linkage part between the servomotor and the plate that converts tho rotational movement into a linear motion, 20 degree at most of inclination is considered, so the angle relation with the dimension is analyzed and calculated in chapter 3

- **Servo Holder:** A simple aluminum structure that holds the servos in place, facilitating controlled movements and reduce the servomotor vibration, shown in figure 2.3.

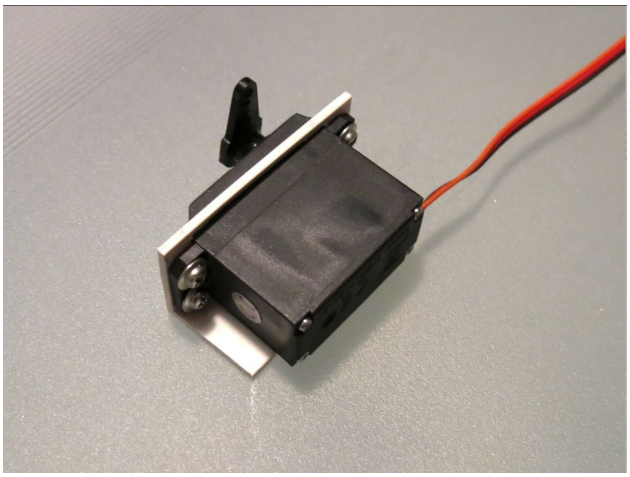


Figure 2.3: The holder structure for the servomotor, ensuring controlled movements and reduced vibration

- **Plate:** Transparent plastic flat platform where the ball rests.
- **Ball:** The object most be spherical symmetrical solid to be balanced on the plate , The ball's mass should be not lower than 70 g. If the force is too small, two different effects can be observed: no touch is detected and the sensor returns Not a Number (NaN) signal or the ball is temporarily detected in a different place

2.2 Electrical parts

The circuit used and all the electrical parts are shown in figure 2.4

2.2.1 Actuators

For our 2-DOF plant, where the plate tilts around its center, two actuators are required to control tilt along the X and Y axes. Accuracy is crucial for achieving zero steady-state error. The Servo motor and stepper motor shown in figure 2.5 are the most suitable choices.

Stepper Motor:

- Comes without an encoder or drive, requiring separate purchase, installation,

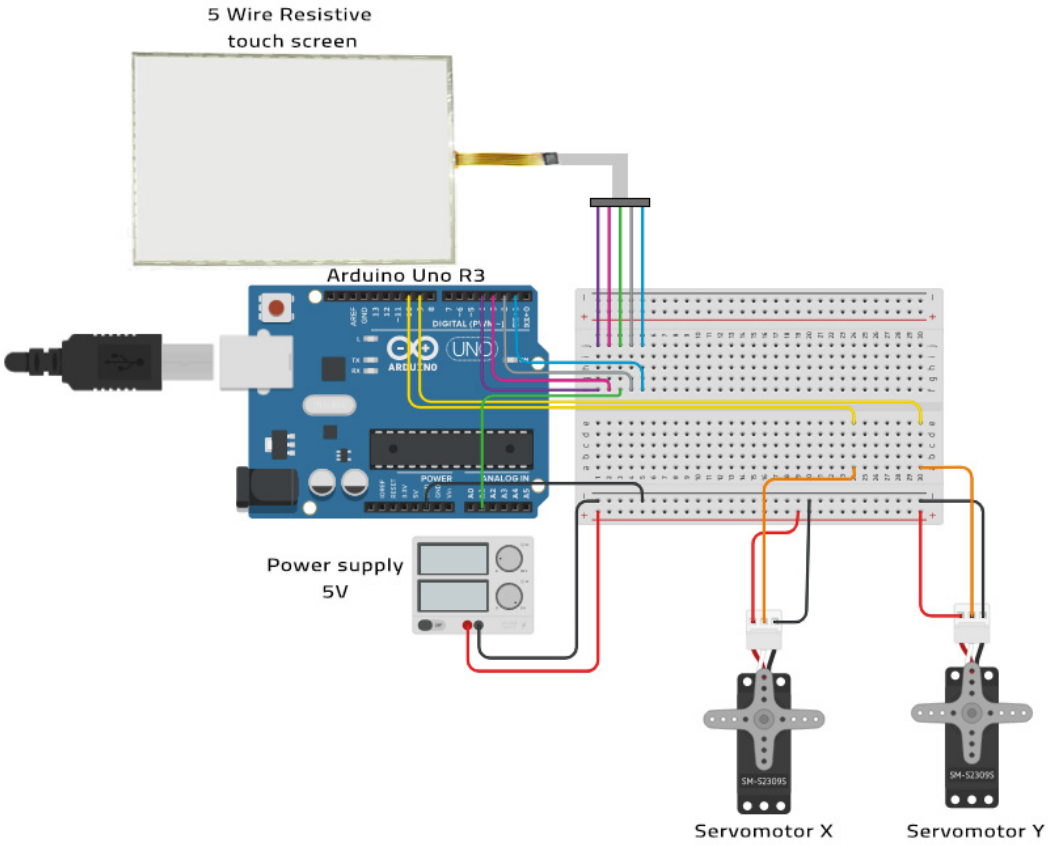


Figure 2.4: Basic schematic illustrating the electrical components and circuitry of the System

and control.

- Relatively more expansive
- Encoder feedback is used to eliminate missed steps.
- Torque decreases with speed, and steppers consume continuous current unless controlled not to.

Servo Motor:

- DC motor with a gearbox for speed reduction and higher torque.
- Internal circuitry includes an encoder/potentiometer and controller.
- Powered by two cables and commanded via Pulse-width modulation(PWM).
- Consumes current only when needed, running cooler.

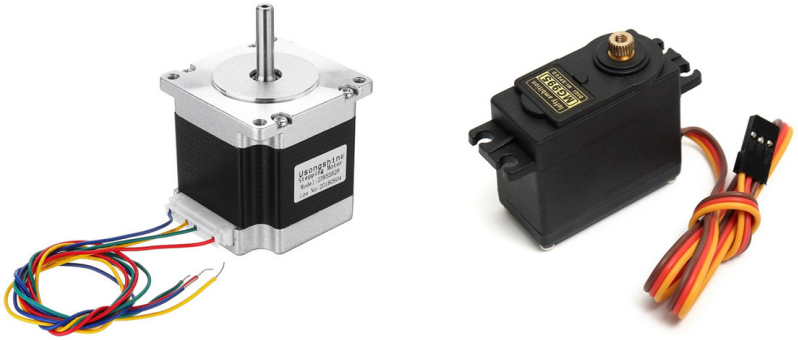


Figure 2.5: Types of actuators explored for the 2-DOF system - Stepper Motor and Servo Motor

Servomotors are used as the actuators. The choice has been motivated by the fact that servos do not require the use of an additional control loop for engine rotation. Additionally, their relatively low cost is also an essential advantage. The servomotors are equipped with a proportional controller. The Pulse Width Modulation (PWM) signal is used to control them, the pulse cycle is equal to 20 ms and pulse width is in the 1000–2000 μs range. Their rotational range is 60 deg, the maximum speed is 0.5 s/60 deg.

2.2.2 Sensor

In the BPS, achieving accurate ball location detection is fundamental for effective control. For this purpose, we explored various sensor options. While alternatives like cameras, infrared sensors, and capacitive systems are available, we went for resistive touch screens due to their versatility and cost-effectiveness. In particular, we have chosen the 17" 5-wire resistive touch screen pad (Width of the panel was 355 mm, its height was 290 mm)

A five-wire resistive touchscreen, much like its four-wire counterpart, comprises two transparent resistive plates separated by insulating spacers. In this configuration, the top plate features a metalized contact, serving as the voltage sensing node. The bottom plate's four corners generate voltage gradients in both the x and y direc-

tions (refer to Figure 2.6). The way the touchscreen works is by setting up specific

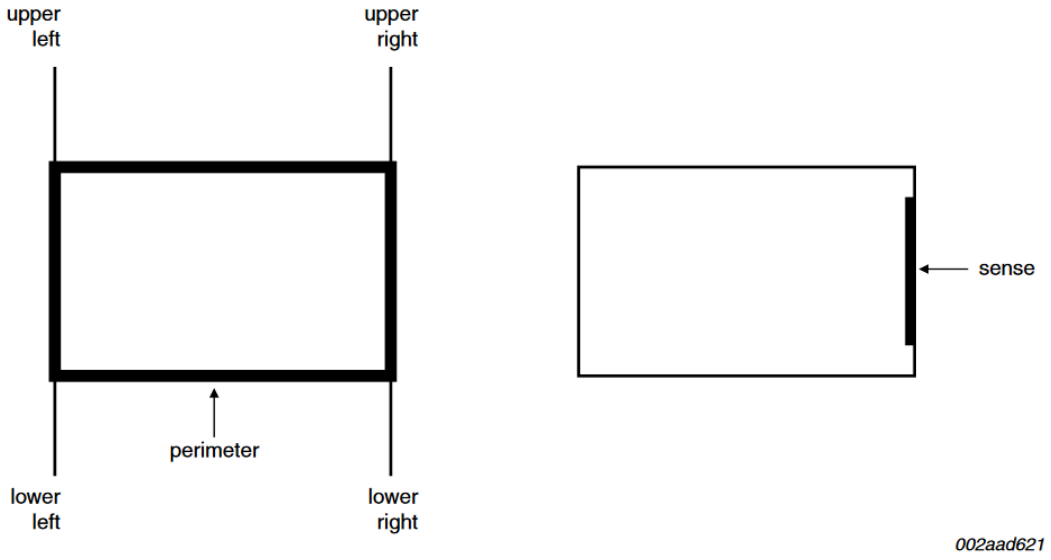


Figure 2.6: Structure of 5-wire resistive touch screen

ways for measuring both sideways (x-direction) and up-down (y-direction). When you press the screen with enough force, the top part bends and touches the bottom part. Figuring out the resistance in a 5-wire touchscreen can be a bit complex, but the addition of circuits along the edges of the touchscreen makes it simpler to think of it like a tool that divides voltage at the spot where it's touched. This, of course, depends on using the right setup called biasing.

Biasing the upper left and right corners to Vss and biasing the lower corners to Vdd enables measurement of the y-coordinate. Conversely, biasing the left side to Vss and the right side to Vdd facilitates measurement of the x-coordinate. Simultaneously biasing all four corners to Vss serves to detect when the screen is touched, triggering an interrupt (see Figure 2.7). In the touch condition, the sense signal from the screen connects to a port pin programmed as an input with a high resistance pullup. All corners of the touchscreen are driven to a logic zero. When touched, a voltage divider is established between the internal pullup of the port pin and the resistance of the touch screen. Notably, the resistance of the touch screen is significantly lower than the pullup connected to the sense signal. Consequently, when a touch occurs, the voltage at the sense signal pin approaches zero, initiating

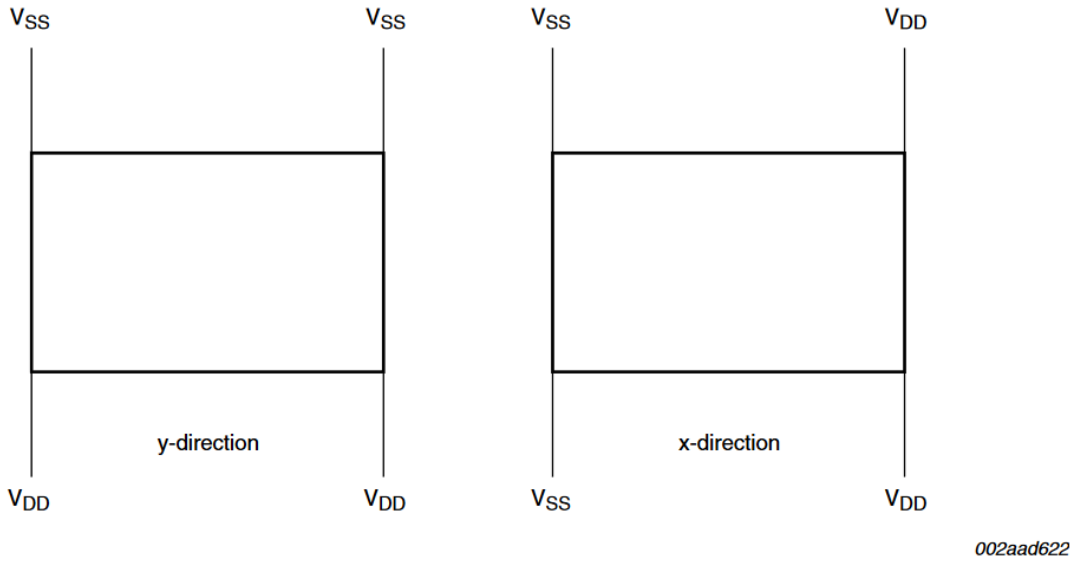


Figure 2.7: Biasing setup for reading x and y directions of a 5-wire resistive touch screen

an interrupt.the biasing and measurement requirements for each of the four wires of the touchscreen are summarized in Table 2.1 .

Table 2.1: Touchscreen interface requirements

Function	**Upper left**	**Lower left**	**Upper right**	**Lower right**	**Sense**
Hardware touch detection	Vss	Vss	Vss	Vss	Logic zero interrupt
Read x-position	Vss	Vss	V_{DD}	V_{DD}	Voltage measurement
Read y-position	Vss	V_{DD}	Vss	V_{DD}	Voltage measurement

The touch panel, while effective, has some disadvantages. It's highly sensitive to disturbances, requiring sufficient pressure into the plate, If the force of the touch is too small, two issues can arise: either no touch is detected, resulting in a NaN signal, or the touch is temporarily detected in a different location than its actual position. While these problems are rare when the ball is stationary, constant rolling can introduce measurement errors. Additionally, tilting the plate can reduce the gravity force component perpendicular to the plate, making the pressure force on the screen insufficient.

To mitigate these issues, several filters are employed:

1. A filter detects when the sensor stops detecting the ball, and the touch panel returns invalid position values (zero, NaN or etc). When identified, the filter ignores these values, using the last meaningful number as the current position.

2. lowpass filter that identifies high magnitude spikes when the sensor detects an incorrect ball position. If the current position change exceeds a certain threshold, it is ignored.

3. An arithmetic filter collects n measurements of the ball position and calculates it as the sum of the measurements divided by n.(An average and median filters was considered to select in between).

2.2.3 Microcontroller

In this work, two approaches to control the system was implemented, in the first approach Arduino Uno R3 was used as a microcontroller,Through it, the sensor data is retrieved and filtered, then through the same Arduino the PID signal is calculated and given to the servomotor as an output. in the second approach, the Arduino only played the role of data acquisition from the sensor to send it to the PC through a serial communication. The computer, using Simulink, processes the data, filters it, controls it, and sends the control signal to the Arduino again, which in turn sends it to the servo motor. Arduino Uno is a microcontroller board based on 8-bit the ATmega 328P. It has 14 digital input/output terminals, where six of them can be used as a pulse width modulation (PWM) outputs. It has six analog inputs, a 16 MHz quartz crystal,a USB connection, a power jack, and a reset button

2.3 Software

This section provides a concise overview of the software design methodology. For the full code implementation, please refer to the appendix A The Arduino software

Figure 2.8: microcontroller: Arduino Uno R3

consists of three primary components. The first component handles the acquisition of coordinate positions from the sensor. The second component deals with serial communication, enabling the exchange of data between the Arduino and the PC. Finally, the third component is devoted to the control section, managing the operational control of the system.

2.3.1 Acquisition of coordinate positions

The implementation of the algorithm for acquiring coordinate positions involves utilizing a 5-wire resistive touch screen, as illustrated in the figure 2.9. The flowchart illustrates the data retrieval process from the sensor, which progresses through several stages. Initially, we set the sensor to a "Stand By Mode". Upon detecting a touch on the screen, if the touch happens we instruct the sensor to capture the signal along both the X-axis and Y-axis. Subsequently, we filter the obtained readings via a lowpass filter. The figure 2.10 below demonstrates the signal's shape before and after passing through the Low-pass filter. Following the filtering, we convert the coordinates of the readings into centimeters.

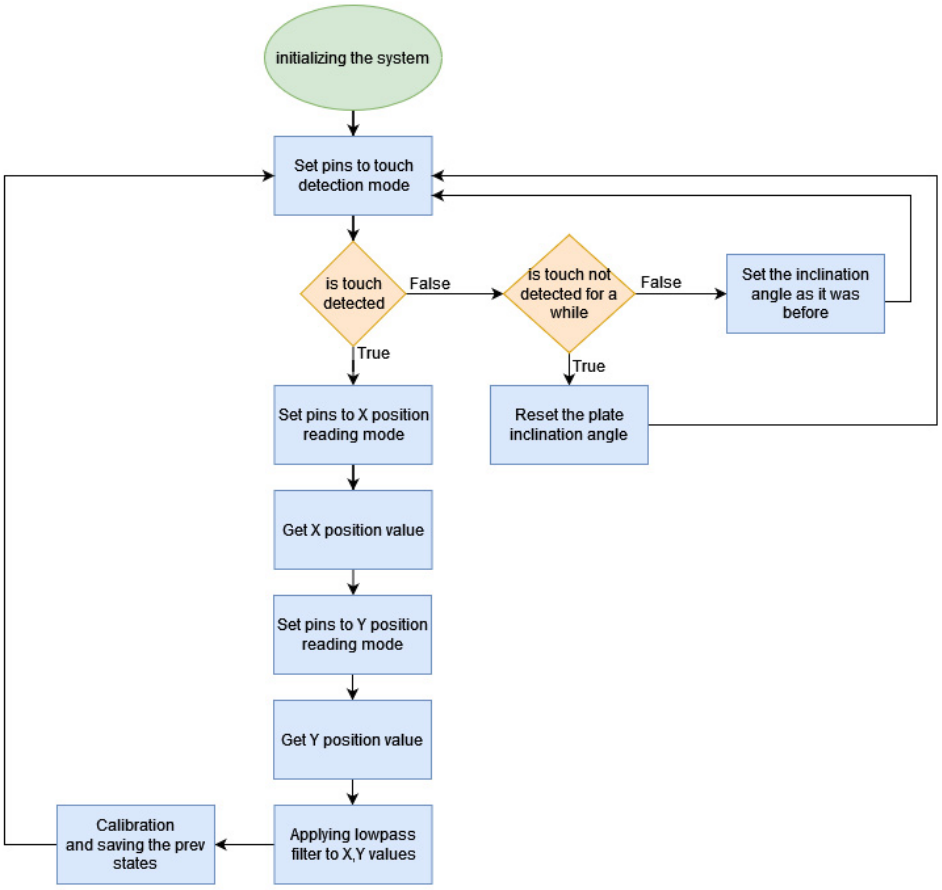


Figure 2.9: Flowchart outlining the algorithm for acquiring coordinate positions from the resistive touch screen

2.3.2 Data transmission via serial communication

Once we figure out where the ball is on plate through the Arduino, we want to share that information with the PC (Simulink). We use something called the "serial communication protocol" to send this data from the Arduino to the computer through a USB connection. Additionally, we want the computer to send back calculated output about the servomotor angle to the Arduino.

Using Instrument Control Toolbox in Simulink we could send and receive bytes in Arduino and interpret it as floats numbers, in the figure 2.11 we used Serial Configuration, Serial Send, Serial Receive blocks, cast to double block and byte pack which converts single data type to 4 packing bytes using byte alignment 4 with a header and terminator

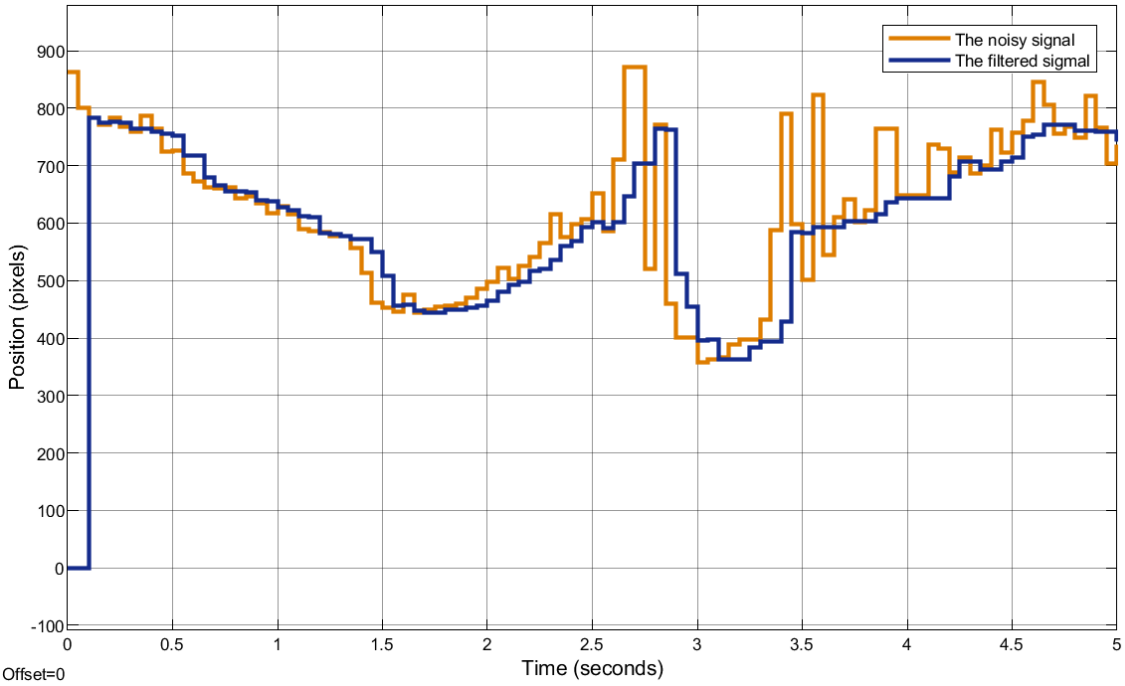


Figure 2.10: Signal comparison before and after applying a low-pass filter to the resistive touch screen readings

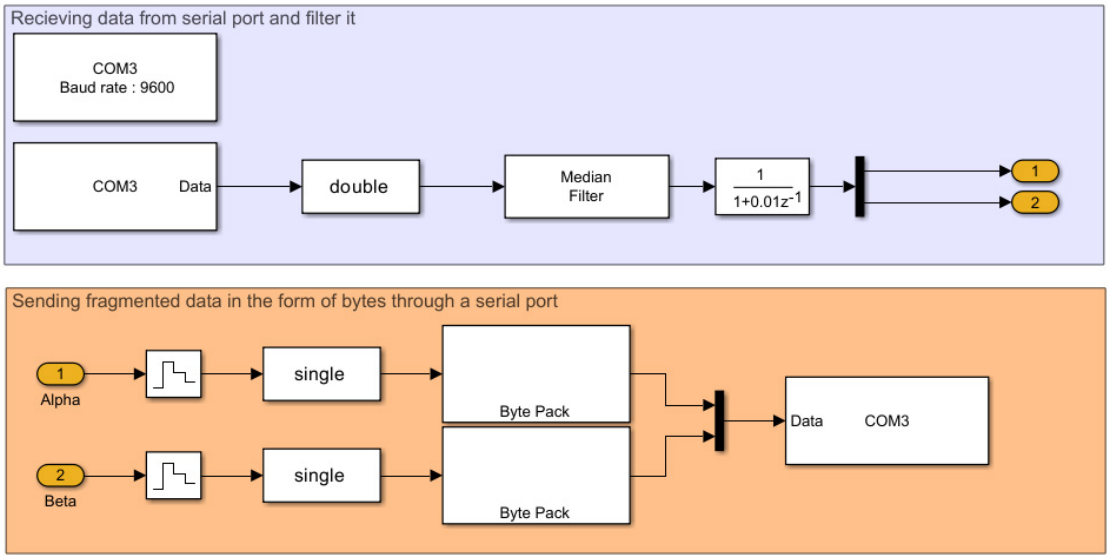


Figure 2.11: Serial communication setup using Simulink for data transmission between Arduino and PC

On the Arduino side, we used a union type to convert a float into an array of bytes (uint8). To receive the bytes from matlab, a function named getFloat() was defined that converts each four bytes into float, as outlined in the code snippet presented in Figure 2.12.

```
typedef union{
  float number;
  uint8_t bytes[4];
} FLOATUNION_t;

FLOATUNION_t sendX1;
FLOATUNION_t sendY2;
FLOATUNION_t xValue;
FLOATUNION_t yValue;
void setup(){
  ....
}
void loop(){
  ....
  //here to send and receive the data required
}
float getFloat(){
    int cont = 0;
    FLOATUNION_t f;
    while (cont < 4 ){
        f.bytes[cont] = Serial.read() ;
        cont = cont +1;
    }
    return f.number;
}
```

Figure 2.12: Code snippet showcasing Arduino-side implementation for receiving and sending data via the serial port

2.3.3 the operational control of the system

To calculate the appropriate servomotor angle we used two methods one By by using serial communication and simulink control toolbox to control the system and the other by using the Arduino PID library designed by Brett Beauregard, in the first algorithm, the code running on the workstation communicates with the Arduino by a serial COM, therefore, the speed at which the Arduino respond is defined by the baud-rate, however the latter method is to be working in lower sampling time because there is no delay that occurs during data transfer

Chapter 3

Mathematical Modeling and Linearization

3.1 System Schematics

The Ball and Plate (BPS) system is unstable, nonlinear, multi-variable, and under-actuated, The BPS model is an extension of the classical Ball and Beam system. It consists of a rectangular flat plate fixed movable at its center by a universal joint. The control target is the plate which could rotate about 2 mutually perpendicular axes The mechanical parts of the BPS are illustrated in Fig. 2.1, which contains;

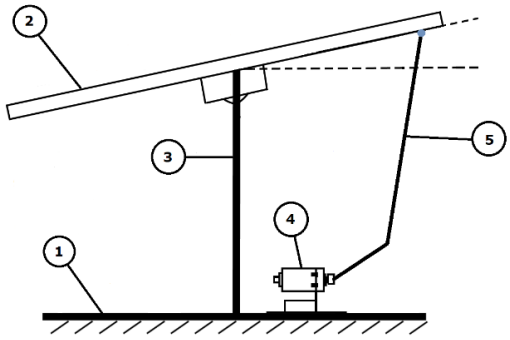


Figure 3.1: Side projection of a BPS

1. The BPS wooden base carries all the other components of the system.

2. Plate is a plastic holder and a resistive touch screen sensor.

3. The central shaft is used between the structure base and the middle of the plate holder. Its job is to hold the plate from the center point by a connected universal joint.

4. The servo motor holder which fits the main base.

5. Two independent two-linkage mechanism which is used to convert rotation motion from servo motor to linear motion in the plate

3.2 Plant modeling

The mathematical model of any mechanical system can be derived using three Methods;

1. The classical Newton method

2. Modern Euler-Lagrange method

3. Hamiltonian mechanics method

In our study, we will use the second method to derive the system's equations of motion and we will compare them with simcape to verify their validity

3.2.1 Requirements and assumptions

Before deriving the mathematical model of BPS, it's essential to begin by setting a standing point to take into account during the mathematical analysis. First, the model as shown in Fig. 2.2 will be divided into two models, a ball and plate model and a servo motor model therefore the following assumptions are made

1. The ball and the plate are in continuous contact.

2. All friction forces and rotational moments are neglected.

3. The ball is completely symmetric and rounded.

4. The ball is not sliding.

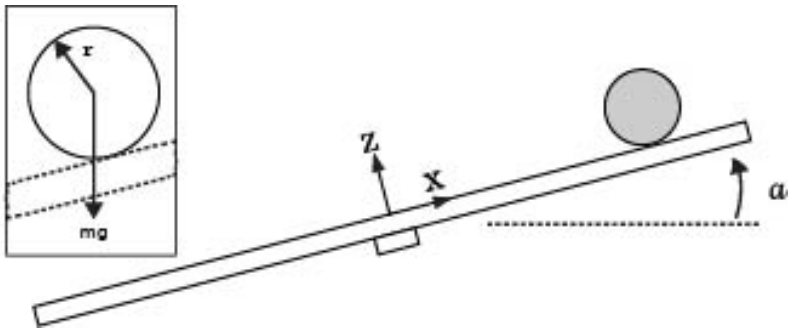


Figure 3.2: Schematic representation of the Ball and Plate System (BPS).

3.2.2 Euler-Lagrange approach

In this section, the mathematical model of BPS is developed based on the Euler–Lagrange equation [22]. The relation between the kinetic and the potential energy possessed by the mechanical system is described by the following equation.

$$L(q_i, \dot{q}_i, t) = T(\dot{q}_i, t) - V(q_i, t) \tag{3.1}$$

where L is the Lagrangian function which represents the difference between the overall kinetic energy T and the overall potential energy V. Then, the general Euler–Lagrange's equation is defined as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{q}_i} - \frac{\partial T}{\partial q_i} + \frac{\partial V}{\partial q_i} = Q_i \tag{3.2}$$

Equation (3.2) can express the motion of a wide variety of mechanical systems where Qi represents the i-th generalized composite force acting on the system and qi is i-th generalized coordinate.

The system has 4 degrees of freedom therefore four generalized coordinates; two in ball motion $[x, y]$ and two in the inclination of the plate$[\alpha, \beta]$. Here we assume the generalized coordinates of the system to be x and y ball's position on the plate and α and β the inclination of the plate

$$q_i \in \{x, y, \alpha, \beta\}$$

The kinetic energy of the system consists of the kinetic energy of the ball and the kinetic energy of the plate

$$T = T_b + T_p \tag{3.3}$$

While the kinetic energy of the ball consists of rotational with respect to its center of mass and translational parts:

$$T_b = T_{trans} + T_{rot} \tag{3.4}$$

The translational energy of the ball can be described by:

$$T_{trans} = \frac{1}{2} m_b (v^2) = \frac{1}{2} m_b (\dot{x}^2 + \dot{y}^2) \tag{3.5}$$

The rotational energy of the ball :

$$T_{rot} = \frac{1}{2} J_b \omega^2 = \frac{1}{2} J_b \frac{v^2}{r^2} = \frac{1}{2} \frac{J_b}{r^2} (\dot{x}^2 + \dot{y}^2) \tag{3.6}$$

Where m_b is mass of the ball and J_b is moment of inertia of the ball; $\dot{x}$ and $\dot{y}$ are ball's translational velocities along x-axis and y-axis; ω_x and ω_y are ball's rotational velocities along x-axis and y-axis that

$$\omega_y = \frac{\dot{y}}{r_b} \quad , \quad \omega_x = \frac{\dot{x}}{r_b} \tag{3.7}$$

By substituting (3.6) and (3.5) into equations (3.4) we will have:

$$T_b = \frac{1}{2} m_b (\dot{x}^2 + \dot{y}^2) + \frac{1}{2} \frac{J_b}{r^2} (\dot{x}^2 + \dot{y}^2) = \frac{1}{2} \left(m_b + \frac{J_b}{r_b^2} \right) (\dot{x}^2 + \dot{y}^2) \tag{3.8}$$

the kinetic energy of the plate, by considering the ball as a point mass which is placed in (x, y) can be expressed as:

$$T_p = \frac{1}{2} (J_b + J_p) \left(\dot{\alpha}^2 + \dot{\beta}^2 \right) + \frac{1}{2} m_b \left(x\dot{\alpha} + y\dot{\beta} \right)^2 \tag{3.9}$$

From T_b (3.8) and T_p (3.9) we can calculate the total kinetic energy by substituting into Eqs (3.3):

$$T = \frac{1}{2}\left(m_b + \frac{J_b}{r_b^2}\right)(\dot{x}^2 + \dot{y}^2) + \frac{1}{2}(J_p + J_b)\left(\dot{\alpha}^2 + \dot{\beta}^2\right) + \frac{1}{2}m_b\left(x\dot{\alpha} + y\dot{\beta}\right)^2 \quad (3.10)$$

The potential energy V of the ball according to the plate center can be calculated as:

$$V_b = m_b g h = m_b g(x\sin\alpha + y\sin\beta) \quad (3.11)$$

Now the Lagrangian function can be expressed using equation (3.1):

$$L = \frac{1}{2}\left(m_b + \frac{J_b}{r_b^2}\right)(\dot{x}^2 + \dot{y}^2) + \frac{1}{2}(J_p + J_b)\left(\dot{\alpha}^2 + \dot{\beta}^2\right) + \frac{1}{2}m_b\left(x\dot{\alpha} + y\dot{\beta}\right)^2 - m_b g(x\sin\alpha + y\sin\beta) \quad (3.12)$$

Then we derive the system's equations

$$\frac{\partial T}{\partial \dot{\alpha}} = (J_p + J_b)\dot{\alpha} + m_b\left(x^2\dot{\alpha} + yx\dot{\beta}\right) \quad (3.13)$$

$$\frac{\partial T}{\partial \dot{\beta}} = (J_p + J_b)\dot{\beta} + m_b\left(xy\dot{\alpha} + y^2\dot{\beta}\right) \quad (3.14)$$

$$\frac{\partial T}{\partial \dot{x}} = \left(m_b + \frac{J_b}{r^2}\right)\dot{x} \quad (3.15)$$

$$\frac{\partial T}{\partial \dot{y}} = \left(m_b + \frac{J_b}{r^2}\right)\dot{y} \quad (3.16)$$

$$\frac{\partial T}{\partial x} = m_b\left(x\dot{\alpha} + y\dot{\beta}\right)\dot{\alpha}, \qquad \frac{\partial T}{\partial y} = m_b\left(x\dot{\alpha} + y\dot{\beta}\right)\dot{\beta} \quad (3.17)$$

$$\frac{\partial T}{\partial \alpha} = 0, \quad \frac{\partial T}{\partial \beta} = 0 \tag{3.18}$$

$$\frac{\partial V}{\partial \alpha} = m_b g x \cos\alpha, \quad \frac{\partial V}{\partial \beta} = m_b g y \cos\beta, \quad \frac{\partial V}{\partial x} = m_b g x \sin\alpha, \quad \frac{\partial V}{\partial y} = m_b g y \sin\beta \tag{3.19}$$

From the Lagrange-Euler equation of the ball, we can write:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{x}} - \frac{\partial L}{\partial x} = (m_b + \frac{J_b}{r_b^2})\ddot{x} - m_b(x\dot{\alpha} + y\dot{\beta})\dot{\alpha} + m_b g \sin\alpha = 0 \tag{3.20}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{y}} - \frac{\partial L}{\partial y} = (m_b + \frac{J_b}{r_b^2})\ddot{y} - m_b(x\dot{\alpha} + y\dot{\beta})\dot{\alpha} + m_b g \sin\beta = 0 \tag{3.21}$$

It is important to note there are no external forces (except gravity) acting on the ball itself ($Q_x = 0$ and $Q_y = 0$) and there are forces in the form of torque acting on the plate and changing its inclination ($Q_{beta} = \tau_y$ and $Q_{alpha} = \tau_x$).

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{\alpha}} - \frac{\partial L}{\partial \alpha} &= (J_b + J_p)\ddot{\alpha} + m_b x^2\ddot{\alpha} + 2m_b x\dot{x}\dot{\alpha} \\ &+ m_b xy\ddot{\alpha} + m_b\dot{x}y\dot{\beta} + m_b x\dot{y}\dot{\beta} - m_b g\cos\alpha = \tau_x\end{aligned} \tag{3.22}$$

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{\beta}} - \frac{\partial L}{\partial \beta} &= (J_b + J_p)\ddot{\beta} + m_b y^2\ddot{\beta} + 2m_b y\dot{y}\dot{\beta} \\ &+ m_b xy\ddot{\beta} + m_b\dot{y}x\dot{\alpha} + m_b y\dot{x}\dot{\alpha} - m_b g\cos\beta = \tau_x\end{aligned} \tag{3.23}$$

So the 4 differential equations of the BPS are respectively (3.20),(3.21),(3.22),(3.23), finally we rewrite these non-linear equations so we got:

$$0 = \left(m_b + \frac{J_b}{r^2}\right)\ddot{x} - m_b\left(x\dot{\alpha}^2 + y\dot{\alpha}\dot{\beta}\right) + m_b g\sin\alpha \tag{3.24}$$

$$0 = \left(m_b + \frac{J_b}{r^2}\right)\ddot{y} - m_b\left(x\dot{\beta}^2 + x\dot{\alpha}\dot{\beta}\right) + m_b g \sin\beta \tag{3.25}$$

$$\begin{aligned}\tau_x = (J_b + J_p + m_b x^2)\ddot{\alpha} + 2m_b x\dot{x}\dot{\alpha} + m_b xy\ddot{\beta} \\ + m_b\dot{x}y\dot{\beta} + m_b x\dot{y}\dot{\beta} - m_b g x\cos\alpha\end{aligned} \tag{3.26}$$

$$\begin{aligned}\tau_y = (J_b + J_p + m_b y^2)\ddot{\beta} + 2m_b y\dot{y}\dot{\beta} + m_b xy\ddot{\alpha} \\ + m_b\dot{x}y\dot{\alpha} + m_b x\dot{y}\dot{\alpha} - m_b g y\cos\beta\end{aligned} \tag{3.27}$$

It can be seen that equations (3.24) and (3.25) describe the ball motion and how the acceleration of the ball depends on its position on the plate and on angles and angular velocities of the plate. Equations (3.26) and (3.27) show plate dynamics and how it depends on external torques, ball position, velocity, angular velocity and acceleration of the plate.

It is necessary to note that dynamics of the stepper motors are neglected, thus the system equations describe only the ball and plate problem. These dynamics will be added in the geometrical analysis subsection later, however we should know that it's assumed stepper motors don't lose any step and load doesn't affect their performance thus equations (3.26) and (3.27) can be neglected

Then we can transform the model into state space form, defining state variables $X = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (x, \dot{x}, \alpha, \dot{\alpha}, y, \dot{y}, \beta, \dot{\beta})$ and from eqs (3.24), (3.25) we can get the nonlinear BPS's state equations as follows:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \\ \dot{x_6} \\ \dot{x_7} \\ \dot{x_8} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{m_b}{m_b+\frac{J_b}{r^2}}(x_1 x_4^2 + x_4 x_5 x_8 - g \sin x_3) \\ x_4 \\ 0 \\ x_6 \\ \frac{m_b}{m_b+\frac{J_b}{r^2}}(x_5 x_8^2 + x_1 x_4 x_8 - g \sin x_7) \\ x_8 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{3.28}$$

3.2.3 Interpretation of terms in system equations

In the Table no. 3.1 description of the terms in the system equation and there units

Table 3.1: Interpretation of mathematical model terms

parameter	description	unit
m_b	Mass of the ball	kg
r_b	Radius of the ball	m
J_b	Moment of inertia of the ball	kgm2
J_p	Moment of inertia of the plate	kgm2
x, y	Coordinates of the ball from the center of the plate	m
$\dot{x}, \dot{y}$	First time derivatives of coordinates which represents the transitional velocity	ms-1
$\ddot{x}, \ddot{y}$	Second time derivatives of coordinates which represents the transitional acceleration	ms-2
α, β	Plate inclination angles	rad
$\dot{\alpha}, \dot{\beta}$	First-time derivatives of plate angles which represents the angular velocity	rads-1
$\ddot{\alpha}, \ddot{\beta}$	Second-time derivatives of plate angles which represent the angular acceleration	rads-2
τ	Torques acting on the plate	Nm
g	Gravitational acceleration	M/s2
$m_b(x\dot{\alpha}^2 + y\dot{\alpha}\dot{\beta})$	Centrifugal force	N
$(J_b + J_p + m_b x^2)\ddot{\alpha}$	Gravitational acceleration	Nm
$2m_b x\dot{x}\dot{\alpha}$	Coriolis influence	Nm
$m_b g x \cos\alpha$	Gravitational influence	Nm

3.3 Linearization and simplfication

To simplify and linearize the model, four main assumptions was made

1. the stepper motors don't lose any step and load doesn't affect their performance.

2. The ball is assumed to be a homogenous and solid sphere

3. The rate of change of angles is close to zero.

4. The angles of the plate are assumed to change in range ⟨-15; 15⟩ or in other words, $|\alpha| < 1$ and $|\beta| < 1$ in radians.

In the interest of model simplification, the first assumption is made that stepper motors exhibit no step loss, and the impact of the load on their performance is discounted. Consequently, angles $|\alpha|$ and $|\beta|$ are deemed direct inputs to the system. Accordingly, equations (3.26) and (3.27) may be excluded and neglected from consideration as we said before.

Secondly to find the $\frac{m_b}{m_b+\frac{j_b}{r^2}}$ term we assumed the ball to be a solid sphere, to find (J_b) of this sphere the approximate expression for a moment of inertia of a solid ball (J_b) can be derived by summing the moments of fragile disks about z-axis as can be seen in fig. 3 The moment of inertia expression for one disk dJ derived by:

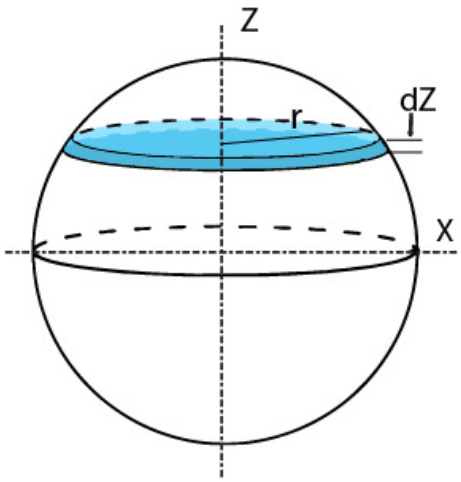


Figure 3.3: Illustration of the moment of inertia calculation for solid ball segments

$$dJ = \frac{1}{2}r^2 dm = \frac{1}{2}r^2 p dV = \frac{1}{2}r^2 p\pi r^2 dz \tag{3.29}$$

the mass of the disk dm can be represented as the density of the ball p times the volume of the disk dV, the volume can be represented as the disk area multiplied by the height dz. To find the sum of the moment of inertia for all disks we integrate both sides of the equation to become:

$$J = \frac{1}{2}p\pi \int_{-R}^{R} r^4\, dz \;= \frac{1}{2}p\pi \int_{-R}^{R} (R^2 - z^2)^2\, dz \;= \frac{8}{15}p\pi R^5 \tag{3.30}$$

Where R is the Radius of the sphere and $y = (R^2 - z^2)$ is derived from the Pythagorean theorem. Substituting the density expression $p = \frac{M}{V} = \frac{M}{\frac{4}{3}\pi R^3}$ gives:

$$J = \frac{8}{15}\left[\frac{M}{\frac{4}{3}\pi R^3}\right]\pi R^5 \tag{3.31}$$

$$J = \frac{2}{5}MR^2 \tag{3.32}$$

So, as has been proven the approximate moment of inertia value for a solid ball is $J_b = \frac{2}{5}MR^2$, therefore by taking the account of all simplfications and substituting (3.32) into the system model we get:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{5}{7}(-g\sin x_3) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} u_x \end{bmatrix} \tag{3.33}$$

$$\begin{bmatrix} \dot{x_5} \\ \dot{x_6} \end{bmatrix} = \begin{bmatrix} x_6 \\ \frac{5}{7}(-g\sin x_7) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} u_y \end{bmatrix} \tag{3.34}$$

It is assumed that the ball is rolling without slipping in continuous contact with the plate keeping in mind the rate of change of angles is low and close to zero, and because of that the angular velocity and the acceleration of the the plate rotation

will be very low and can be negligible,

$$\dot{\alpha} \approx 0$$

$$\dot{\beta} \approx 0$$

the last assumption to finally linearize the model is that the angle of inclination for the plate is relatively small (up to ± 15) which leads $\longrightarrow \sin\alpha \simeq \alpha$, $\sin\beta \simeq \beta$ in rads so the final linearized equation will be

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} x_2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{5}{7}(-g) \end{bmatrix} \begin{bmatrix} \alpha \end{bmatrix} \tag{3.35}$$

$$\begin{bmatrix} \dot{x_5} \\ \dot{x_6} \end{bmatrix} = \begin{bmatrix} x_6 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{5}{7}(-g) \end{bmatrix} \begin{bmatrix} \beta \end{bmatrix} \tag{3.36}$$

Equation No. (3.35) serves as the foundational equation for the subsequent steps in our process. Following these simplifications, the system equations can be expressed in transfer function form:

$$\ddot{x} + \frac{5}{7}g\alpha = 0 \tag{3.37}$$

$$\ddot{y} + \frac{5}{7}g\beta = 0 \tag{3.38}$$

Assuming the angles as inputs to BPS, the linearized transfer function can be obtained as:

$$G(s) = \frac{x(s)}{\alpha(s)} = -\frac{5}{7}\frac{g}{s^2} \tag{3.39}$$

$$G(s) = \frac{y(s)}{\beta(s)} = -\frac{5}{7}\frac{g}{s^2} \tag{3.40}$$

To validate our linearization process, we compared the linear model with nonlinear simulink and Simscape models through an open-loop step response by the Simulink program, using four different inputs, as presented in Figure 3.4. Since the

system is open-loop, the response is unstable, and we can also notice from the figure that the difference between the three models increases as the angle of the plate α increases. This is due to our assumption that the input angle changes in the range $\langle 15; 15 \rangle$.

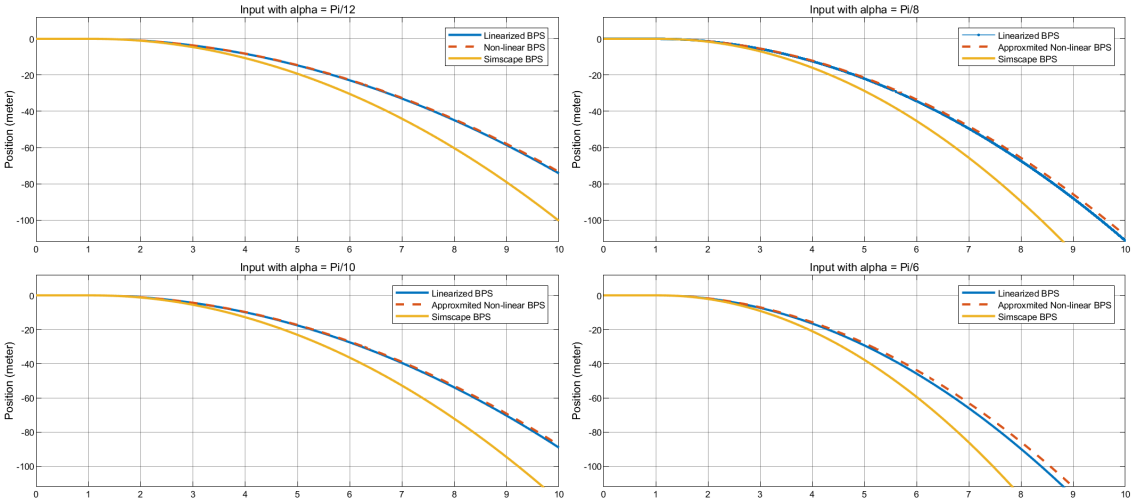


Figure 3.4: Comparison of open-loop responses for linear, non-linear, and Simscape BPS models

3.4 Geometrical analysis

Since our overall system input should be the angle signal provided to the servo motor, not the plate inclination angle itself, we should discover the transfer function of the servo motor and then we should determine the gain or geometrical relationship between the plate inclination angle and servo motor angle

3.4.1 Servo motor modeling

Servomotors, like the MG995 model that we used in our thesis, are widely used for precise linear or angular motion in engineering. In Fig. 3.5, key components of the MG995 are highlighted. The DC motor generates rotational motion from applied voltage, while the gearbox adjusts speed and boosts torque. The electronic control board interprets PWM signals, converts them to angles, and computes control signals by comparing them with the current angle. Unlike professional servos

with encoders, inexpensive hobby servos, including china made MG995's, often lack encoders and instead use a potentiometer for position sensing. The potentiometer, rotating with the output shaft, induces a voltage signal variation sent to the control board. This signal goes as a feedback and helps the internal controller calculate the difference between the desired and current positions, serving as the input for the embedded controller [23]. In the electrical dynamics of Fig. 3.6, key parameters

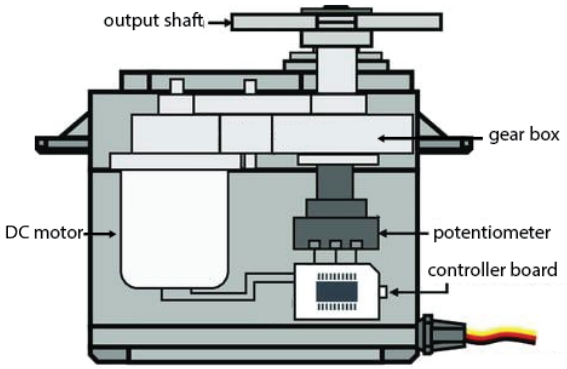


Figure 3.5: Schematic representation of key components in the MG995 servomotor

include u (voltage applied), i (electrical current), R (motor resistance), L (inductance), and e (back electromotive force). On the mechanical side, τ_m denotes motor torque, τ_{jm} is motor moment of inertia, b_m is viscous friction coefficient, τ_f is viscous friction torque, θ_m is motor shaft rotation angle, N is gearbox gear ratio, J_l is load moment of inertia, τ_l is load torque, and θ_l is servo output shaft rotation angle. To

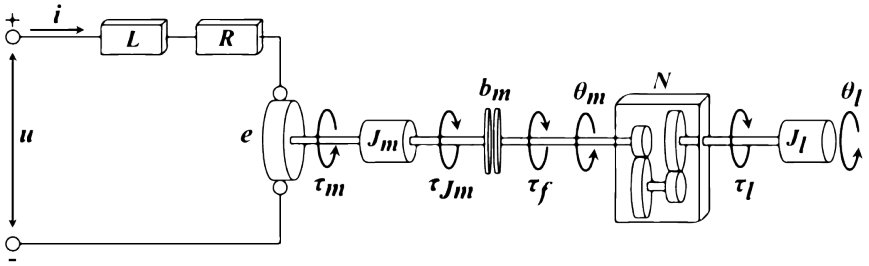


Figure 3.6: The electromechanical diagram of a servomotor system

obtain a transfer function that represents the open-loop dynamics $P(s)$ of a servo motor for a voltage input $U(s)$ and as output the angle of the servo output shaft $\Theta_l(s)$, is given by:

$$P(s) = \frac{\Theta_l(s)}{U(s)} = \frac{K_t K_\omega \eta}{(J_{eq}s^2 + b_{eq}s)} \tag{3.41}$$

where K_t is the motor torque constant, K_ω is the speed constant, and η is the gearbox efficiency factor, $J_{eq} = J_l + J_m\eta N^2$ and $b_{eq} = b_m\eta N^2$. This transfer function characterizes the open-loop behavior of the servo motor.

The closed-loop system is shown in Figure 3.7, where the controller $C(s)$ is assumed

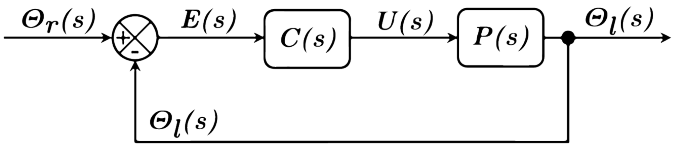


Figure 3.7: Diagram illustrating the key parameters in the dynamical model of the servomotor

to use a proportional control with gain K_P. The input of the closed-loop control is the commanded angle $\Theta_r(s)$, and the feedback signal is the load output shaft angle $\Theta_l(s)$. The error signal $E(s)$ is defined as the difference between the commanded angle and the actual output angle. The open-loop transfer function $P(s)$, is used to obtain the closed-loop transfer function $G(s)$, which relates the output angle $\Theta_l(s)$ to the commanded angle $\Theta_r(s)$:

$$G(s) = \frac{\Theta_l(s)}{\Theta_r(s)} = \frac{\eta K_t K_P}{(RJ_{eq}s^2 + (Rb_{eq} + \eta N^2 K_t^2 K_P)s + \eta N K_t K_P)} \tag{3.42}$$

This transfer function characterizes the closed-loop behavior of the servo motor, taking into account the effects of the proportional control and the feedback loop. Due to the limited time in our thesis we will use the parameters and transfer function identified in[23] which it uses a very unique method to determine the input and output angles of the servo therefore to be able to use the Matlab system identification toolbox:

$$\frac{\Theta_l(s)}{\Theta_r(s)} = \frac{224.8}{s^2 + 22.33s + 225.4} \tag{3.43}$$

can also be represented in state space formula as:

$$\begin{bmatrix} \dot{\Theta}_l \\ \ddot{\Theta}_l \end{bmatrix} = \begin{bmatrix} 1\dot{\Theta}_l \\ 225.4\Theta_l + 22.33\dot{\Theta}_l \end{bmatrix} + \begin{bmatrix} 0 \\ 224.8 \end{bmatrix} \begin{bmatrix} \Theta_r \end{bmatrix} \tag{3.44}$$

3.4.2 Linking Plate Inclination and Servo Motion

There are several different ways to connect the motor to the plate that to convert the rotational move of the servomotor into a simple plate inclination, we have adopted the simplest method, such as the one shown in figure. 3.8, by the requirement of minimum inclination angle and due to the lack of availability of all parts in the market, we were able to obtain and construct the dimension shown in Table. 2

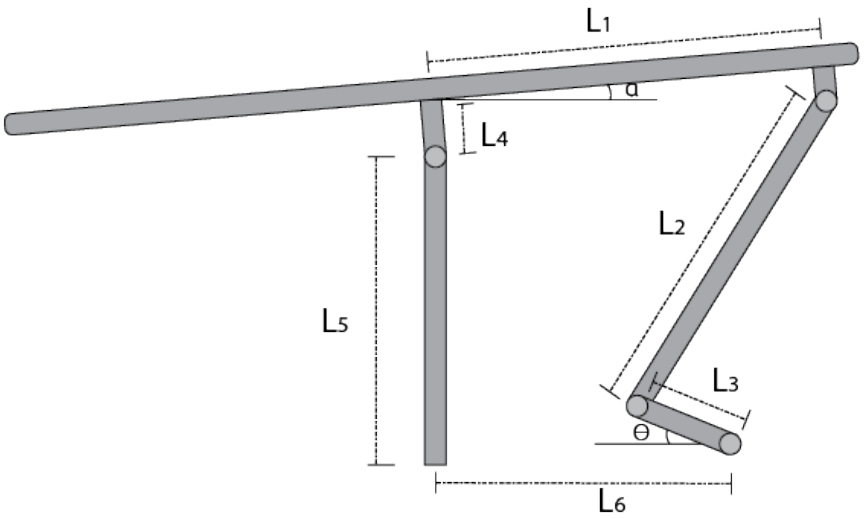


Figure 3.8: geometric illustration of the linkage between actuator and the plate of BPS

Table 3.2: System dimensions in centimeter

X	L1	L2	L3	L4	L5	L6
Value	9.5	24	1.5	5	22	13.4
Y	L1	L2	L3	L4	L5	L6
Value	10	24	1.5	5	22	13

The relation between the angle of rotation of the plate around the x axis α and

the angle of rotation of servomotor arm L3 Θ is:

$$\sin\Theta = \frac{L_3}{L_1}\sin\alpha \tag{3.45}$$

which can be represented as a constant gain $K_l = \frac{L_3}{L_1}$, from Table. 2

$$K_{lx} = 0.158 \; K_{ly} = 0.150 \tag{3.46}$$

but the behaviour of the angles appears a Little different so we decided to map the inclination angle with the servo motor angle and determinee by the linear relationship regression the appropriate gain the full table and details of mapping method found in Appendix C, and the new gain values we get is:

$$K_{lx} = 0.0445 \; K_{ly} = 0.0377 \tag{3.47}$$

The overall system equation taking in account the plant equation Eq. (3.39) (3.40) with the linkage gain Eq. (3.47) and the identified servomotor equation Eq. (3.43) can be represented as follows:

$$\frac{x}{\Theta_r} = \frac{70.1}{s^4 + 22.33s^3 + 225.4s^2} \tag{3.48}$$

Which after state space transformation:

$$\begin{bmatrix} \dot{x_1} \\ \ddot{x_2} \\ \dot{x_3} \\ \ddot{x_4} \end{bmatrix} = \begin{bmatrix} x_1 \\ \frac{-5}{7}gx_3 \\ x_4 \\ -225.4x_x - 22.33x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 70.1 \end{bmatrix} \begin{bmatrix} \Theta_r \end{bmatrix} \tag{3.49}$$

The system in Eq. (3.47)(3.48) is a fourth order transfer function, unstable due to the two poles in the origins as shown fig. 3.9, however it can be considered as a second order system since the other two poles in fig. 3.9 are in the left side far away

from the origin so they have a little impact on the system behavior

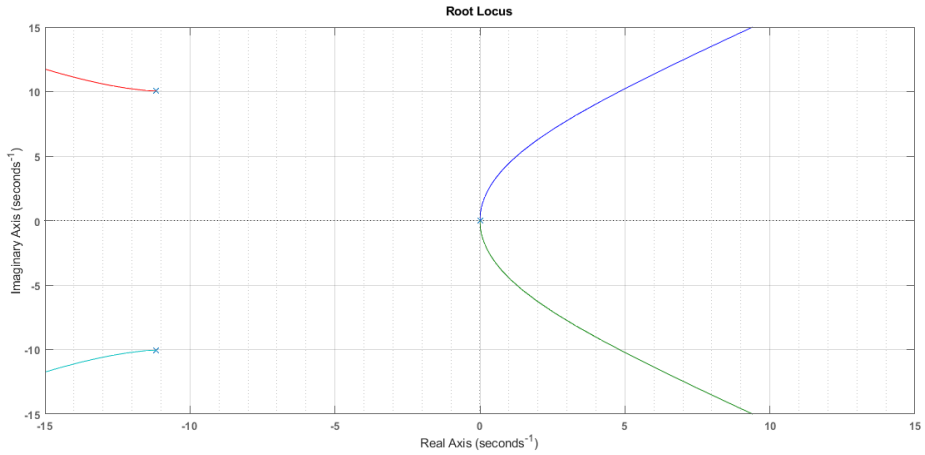


Figure 3.9: Root Locus Analysis: Navigating Stability for BPS

Chapter 4

Controller Design Approaches

In this chapter we will focus on designing an appropriate controller theoretically and experimentally

4.1 System Requirements

Before starting into the evaluation of the controller's performance, certain requirements conditions and specifications have been identified as crucial benchmarks. These prerequisites provide a foundational base to be able to determine if the controller performs well enough:

1. the plate to incline only in 15 deg in both coordinates.
2. Settling time less than 5 seconds ($Ts < 5$)
3. Rise time less or equal to 1 seconds ($T_r < 0.5$).
4. A overshoot less than 20% ($M < 0.20$).
5. The static error should be equal to zero or less than 25 pixels ($e_0 < 0.05$).

The goal is that the controller will be able to fulfill these demands.

4.2 System Controllability and Observerability

The controllability and observability of the system are crucial aspects in ensuring effective control. In order to have full freedom in the placement of the poles, it is necessary to check the controllability of the system. This is done by determining the controllability matrix S [24], which is given by:

$$S = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{4.1}$$

where n is the number of states. If the determinant of the matrix S is non-zero, it indicates that the placement of the poles is not restricted, allowing for the fulfillment of control demands. Using the overall plant transfer function in Eq. (3.48), The determined controllability matrix:

$$S = 1.0 \times 10^5 \times \begin{bmatrix} 0 & 0 & 0 & -0.0158 \\ 0 & 0 & -0.0158 & 0.3527 \\ 0 & 0.0023 & 0.0010 & 0.6159 \\ 0.0023 & -0.0503 & 0.6159 & -2.4073 \end{bmatrix} \tag{4.2}$$

The rank of the controllability matrix Eq.(4.2) for the linear model is 4, which is full rank meaning the system is completely controllable

Observability is equally important, as the controller needs to have all the information to control the system successfully. The observability of the system can be checked using the observability matrix O, defined as:

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{4.3}$$

Just like in controllability, if the observability matrix Eq.(4.4) is full rank then

the system is observable.

$$O = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0 & -7.0071 & 0 \\ 0 & 0 & 0 & -7.0071 \end{bmatrix} \tag{4.4}$$

Since the matrix O is full rank (rank(O) = 4). This means that we can estimate all the states from the displacement measurement and the system is both observable and controllable

4.3 PID Controller design

The Proportional Integral Derivative (PID) controller is one of the most widely used controllers today. The ideal PID controller is described by the equation:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{d}{dt} e(t) \tag{4.5}$$

where u represents the control signal and K_p , K_I and K_D denote the proportional, integral and derivative gains respectively. Notably large value for K_p leads to increased speed of the controller. However, it can compromise stability. K_I eliminates static error, but also decreases stability. On the other hand the derivative gain, K_D, decreases the oscillation and leads to stability enhancement The closed-loop transfer function of a PID-controlled system is given by: Eq.4.2 where s is the Laplace variable

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s \tag{4.6}$$

To enable the PID controller's operation, the error $e(t)$ is essential. This error is derived by taking the difference between the reference signal $r(t)$ and the actual system output $x(t)$), represented as the variance between the desired position and the sensed position (as depicted in Fig. 4.1). In this illustration, two feedback loops

are evident. The internal loop corresponds to the servomotor's internal controller, managing its own dynamics. Simultaneously, and the external loop ensures that the PID controller receives the necessary input after error calculation

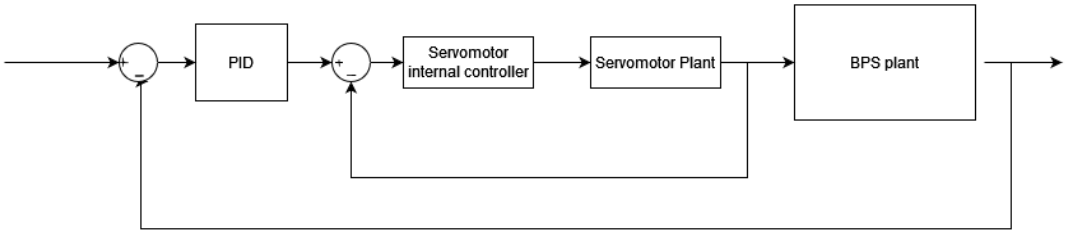


Figure 4.1: A block diagram illustrates BPS's control mechanism using PID

In real-world scenarios, achieving an optimal PID controller often involves considering the inherent dynamics and characteristics of the system. Practical tuning methods, such as frequency response analysis or model-based approaches, provide engineers with tools to fine-tune the controller parameters for specific performance criteria. In order to meet our requirements, we utilized the Simulink PID block tuner tool. The block diagram, as depicted in Fig. 4.2, served as the foundation for our tuning process.

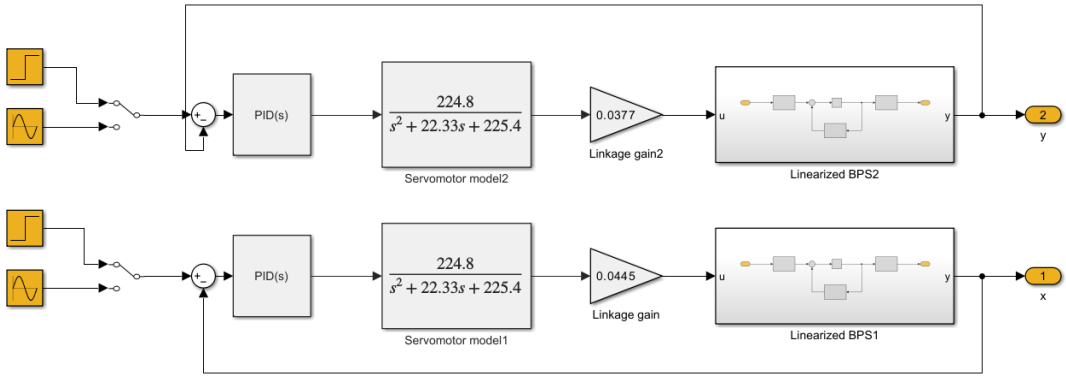


Figure 4.2: the simplified Simulink PID diagram, highlighting its key components and signal pathways

Although our tuning efforts using the Simulink PID block tuner tool yielded a very good response, we faced some challenges that required attention. The main concern was the PID output, went beyond the 0 to 180-degree range. This presented

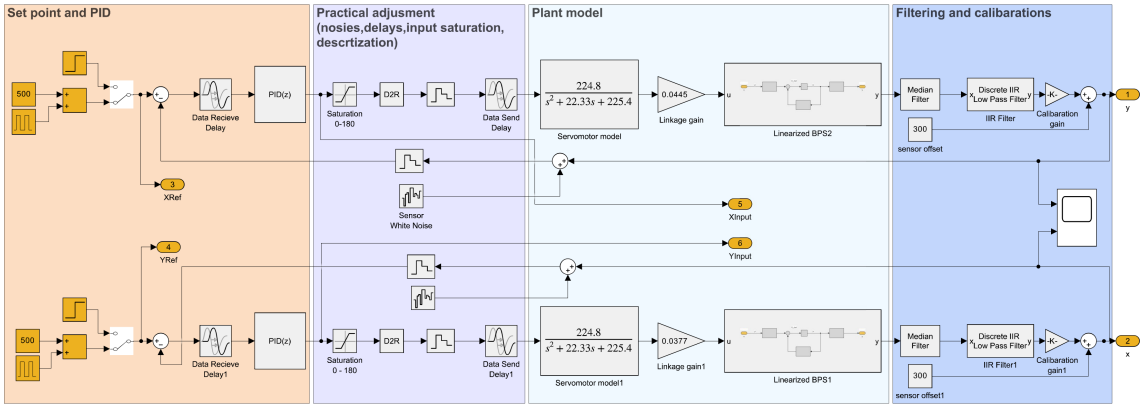


Figure 4.3: Refined Simulink Model Illustrating Practical Adjustments

a practical issue as the servo couldn't handle values outside its normal operational limits.

Moreover, our sensor provided the ball position through an analog signal ranging from 0 to 1024, making it necessary to make adjustments to the model. To resolve these issues, we implemented a saturation function to limit the PID output within a manageable range. Additionally, we introduced a calibration transformation and filter to the sensor signal. To account for calculation and data transmission delays, we incorporated a delay component into the model.

These adjustments and refinements led us to the development of Fig. 4.3, which represents a more practical and realistic model (a higher quality image can be found in appendix B). This modified model takes into account the limitations and characteristics of both the servo and the sensor and considers the calculation and data transport delay from the computer to the arduino and vice versa.

As a result of these adjustments introduced complexity and non-linearity to the system that surpassed the capabilities of SIMULINK tuning tools. Consequently, we shifted to a manual tuning approach, relying on a trial-and-error method to iteratively fine-tune the parameters for optimal system performance. The PID gains, determined through manual tuning using a trial-and-error method, are summarized in Table 1.

Table 4.1: Tuned PID Parameters

	Proportional Gain (K_P)	Integral Gain (K_I)	Derivative Gain (K_D)
PID X 1	0.125	0.00184	0.098
PID Y 1	0.127	0.00084	0.088
PID X 2	0.165	0.00085	0.11
PID Y 2	0.167	0.00084	0.105
PID X 3	0.105	0.00084	0.07
PID Y 3	0.107	0.00184	0.069

4.4 PID Controller Implementation

After the theoretical foundation of the PID controller design, the next crucial step is its practical implementation. This section outlines the real-world application of the PID controller to control the Ball and Plate System (BPS).

4.4.1 Real System Implementation

The PID controller, designed based on the principles discussed earlier, was implemented in the physical BPS. Figure 4.4 showcases the real system setup, highlighting the integration of the PID controller into the overall system.

In this configuration, the PID controller interacts with the servomotors to adjust the position of the plate in response to the sensed ball position. The physical setup involves the Arduino Uno, servomotors, resistive touch screen, and the mechanical structure of the BPS.

4.4.2 Simulink Model for PID Control

To facilitate a seamless interaction between the PID controller and the real system, a Simulink model was developed. Figure 4.5 illustrates the Simulink model responsible for PID control (a higher quality image can be found in appendix B), incorporating the practical adjustments discussed earlier.

The Simulink model integrates the PID block with the plant model, representing the dynamics of the BPS. Additionally, it includes the saturation function, calibration transformation, and filters to ensure compatibility with the real-world limita-

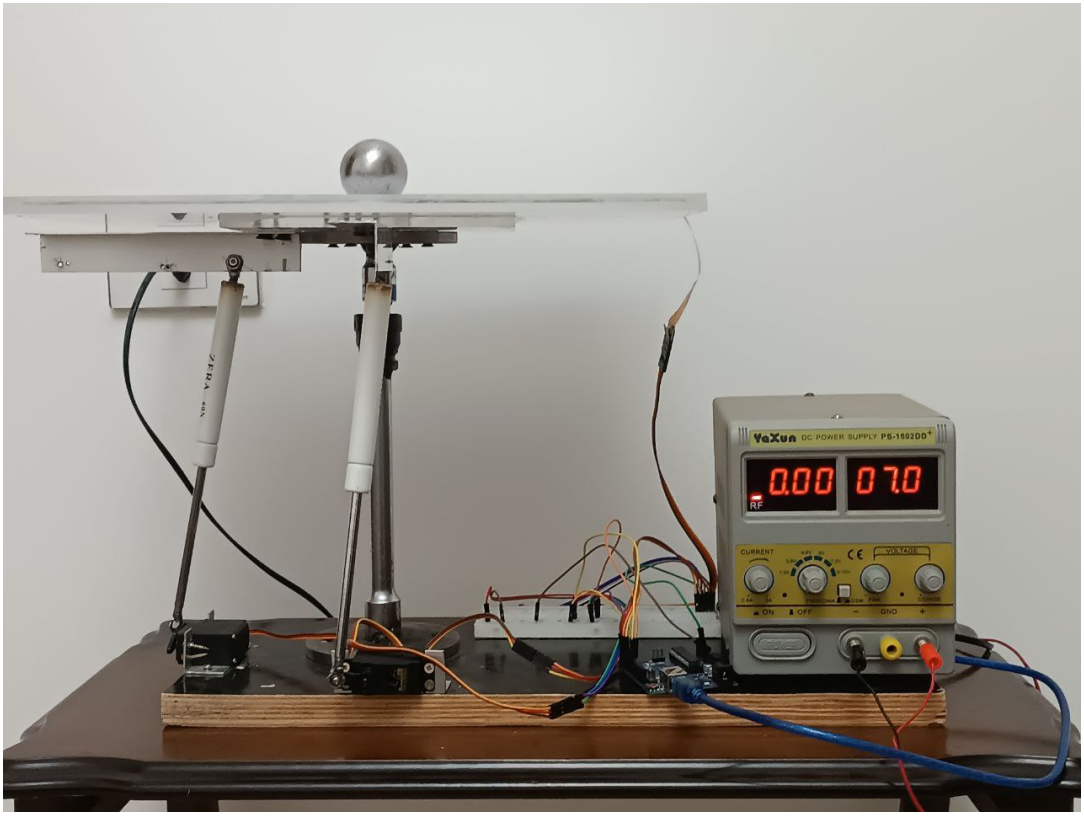


Figure 4.4: Real System Implementation with PID Controller

tions of the servo and sensor.

The communication between the Simulink model and the real system is established through the Arduino via COM3 serial pot, which serves as the intermediary for data exchange. This bidirectional communication allows the Simulink model to receive real-time data from the system and send control signals back for continuous adjustment.

4.4.3 Challenges and Manual Tuning

Despite the theoretical foundation and simulation-based tuning, the real-world implementation presented challenges.

One of the initial challenges was determining the appropriate time sampling for the PID controller to ensure compatibility with the Arduino. An approximate time sample for the Arduino was estimated, considering its processing capabilities, and additional delay factors were incorporated. After an iterative experimentation, it became evident that a time sample of 0.08 seconds provided the best response.the chosen time sampling is crucial for the overall responsiveness of the PID controller

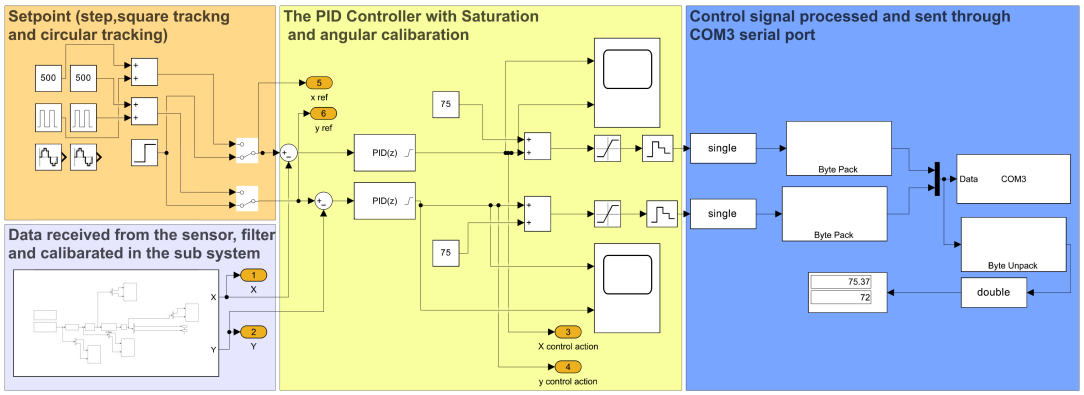


Figure 4.5: Simulink Model for PID Control Implementation

in the real system. It plays a pivotal role in balancing the need for rapid adjustments with the Arduino's processing constraints. This optimization contributes to achieving a smooth and stable control mechanism for the Ball and Plate System. Another significant challenge encountered during the real-time implementation was associated with the differential part of the PID controller. In instances where the ball experienced sudden and significant movements, resulting in rapid changes in its position, the differential component of the controller exhibited a behavior known as "differential kicks" presented in Fig. 4.6. These sudden kicks or spikes in Fig. 4.6 had a detrimental impact on the stability of the ball on the plate.

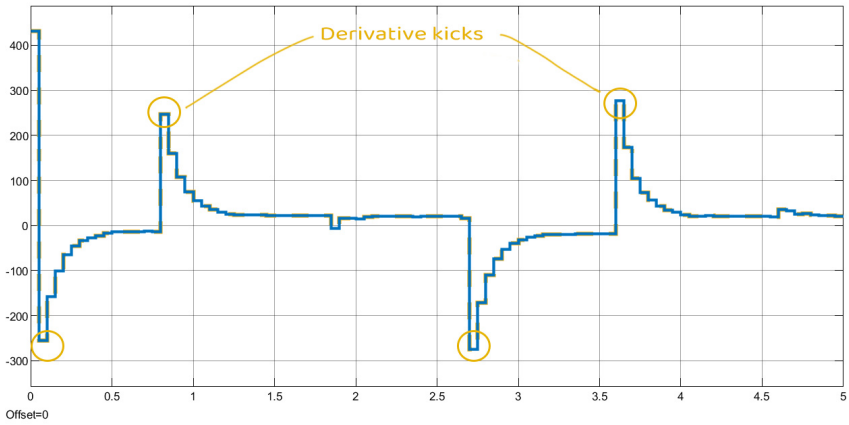


Figure 4.6: Illustratiion of the "Differential Kicks" Phenomenon

To solve this issue an median filter was applied. This filter smoothed out the extreme and abrupt changes in the controller signal, ensuring a more controlled response. By computing the average controller signal over a 3 time window, this approach struck a balance between responsiveness and stability, enhancing the adaptability of the control system.

Chapter 5

Results and Discussion

5.1 Introduction

In this chapter, we'll show and confirm the outcomes of our modeling efforts, examining both the linear and non-linear states of the model. We'll compare these models with the real-world implementation and Simscape simulations. Next, we'll evaluate the performance of the PID controller we designed earlier in the previous chapter by comparing simulated results with our system actual outcomes.

5.2 Open Loop Step Response of Linear and Non-linear Models against Real Implementation

To ensure our model's accuracy, we conducted a step response comparison under Open Loop conditions, as depicted in Figure 5.1. The Figure 5.1 indicate a clear instability in the system, particularly noticeable as the input angle rises. A noticeable difference arises between the linearized and non-linear systems. The observed difference can be traced back to our simplification assumption, limiting the input plate inclination angle to the range of -15 to +15 degrees.Any variations beyond this specified range result in increased disparities between the linear and non-linear responses.

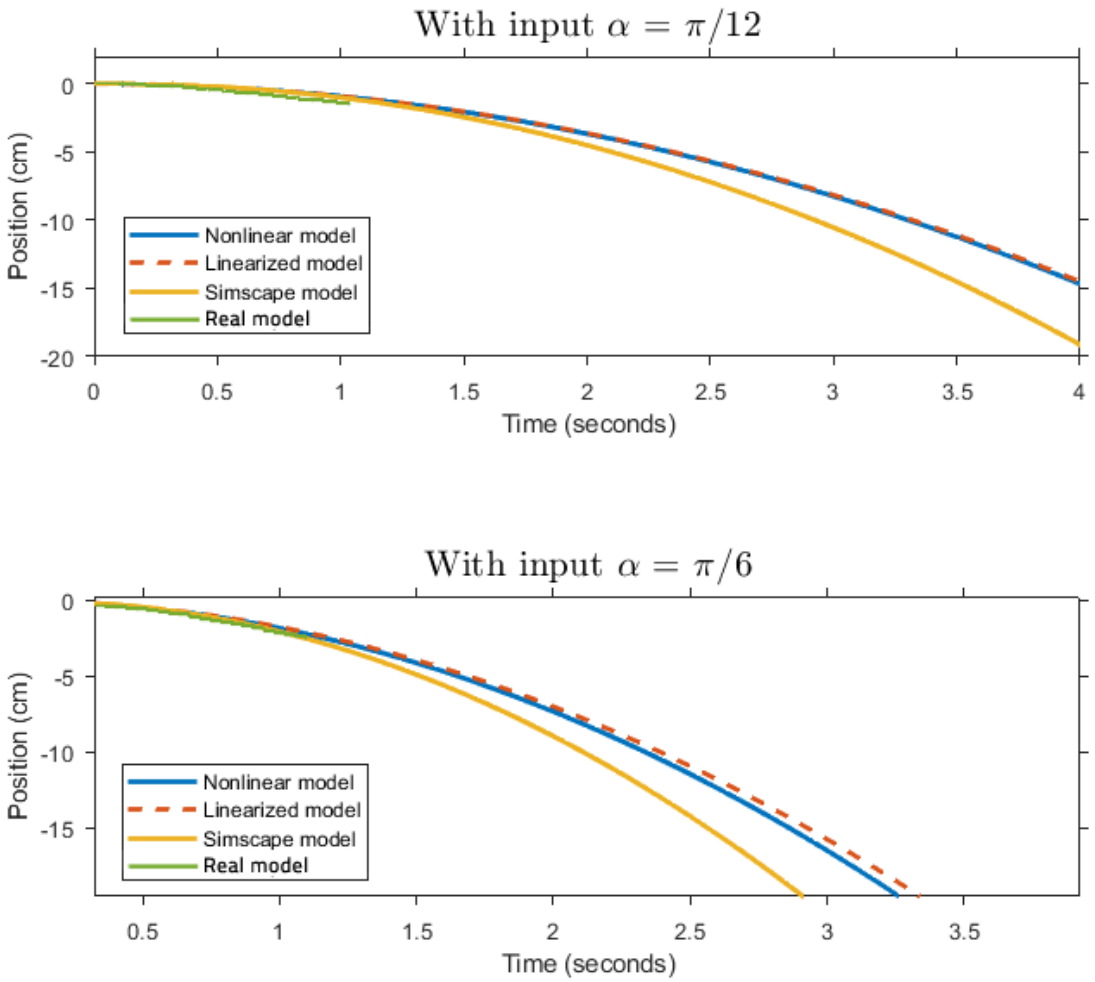


Figure 5.1: open loop response comparison between the real plant, nonlinear model, linearized model, simscape model

Moreover, our model assumes an infinite plate space, depicting a scenario where the ball neither falls nor bounces. This abstraction diverges from reality, introducing challenges and complexities in the practical implementation of the system

As a final test, the response of the real system to an angle of pi/12 is depicted in the figure 5.1. Notably, the response aligns reasonably well with our linear model, suggesting a certain level of reliability in our linearized representation

5.3 Closed Loop Validation of PID Controller Performance

In the tuning stage of our PID controller, achieving system stability was impossible using only a P or PI controllers, The ball exhibited endless rolling or even falling off the plate. therefore tuning a full PID controller was necessary, after manual tuning process characterized by trial and error the parameter derived was as shown in table 4.1. The resulting experimental step responses of the three PIDs tuned parameters are depicted in Figure 5.2, and details of these responses characteristics are presented in Table 5.1, The experimental step responses in Figure 5.2 reveal distinct behaviors under the influence of PID1, PID2, and PID3. PID2, characterized by a 100% overshoot and a relatively shorter rise time of 0.55 seconds, demonstrates a more aggressive response compared to PID1 and PID3.

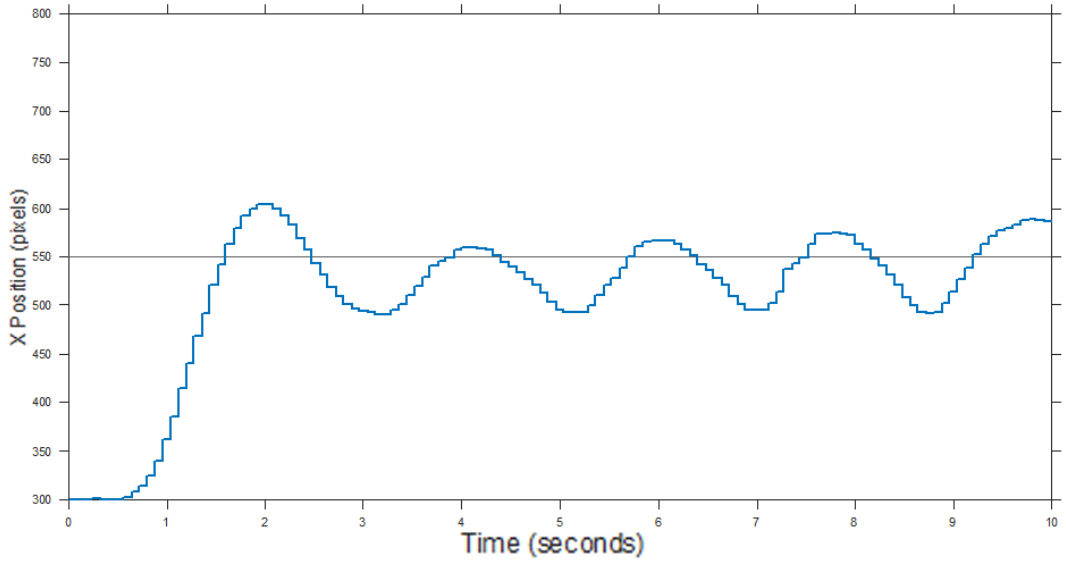
Table 5.1: PID Controller Characteristics

Controller	**Overshoot (%)**	**Rise Time (s)**
PID1	20	0.65
PID2	100	0.55
PID3	28	0.65

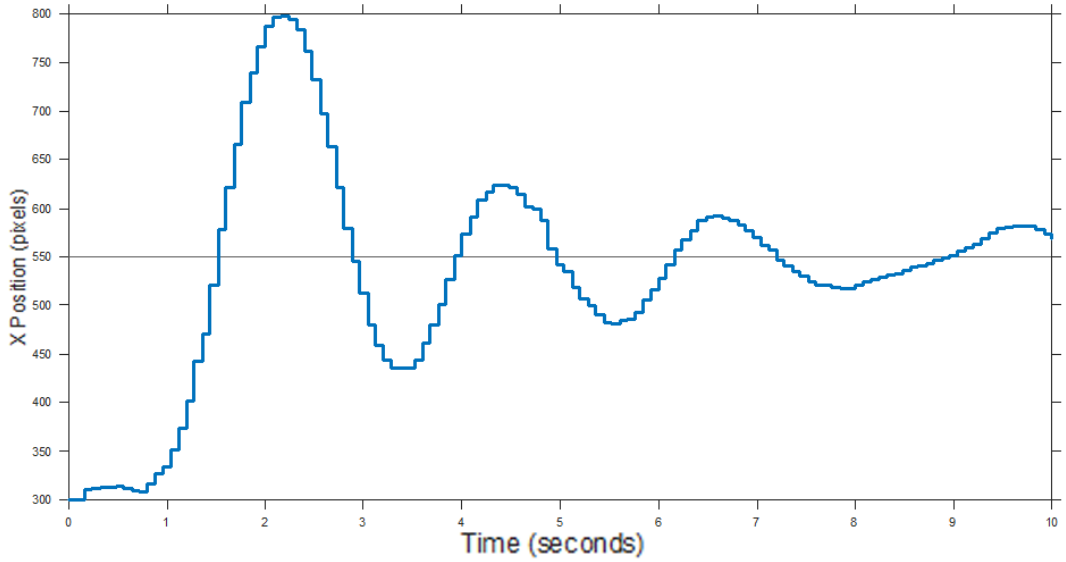
the resulting simulation and experimental response and controller output for all three PIDs is in the figure 5.3, 5.4, and 5.5 a and b respectively, Between the two responses a and b of each figure there are subtle differences. This variance can be caused by factors such as the neglected friction, vibrations from the servomotor or plate, time delays caused by the filtering or data transmission, and the non-symmetry in the shape of the ball. These real-world complexities introduce nuances that impact the system's behavior.

The integral action in our controller can effectively eliminate steady-state errors, but the presence of steady-state error in the response caused by the static friction, When the ball gets close to the desired position, small tilting the plate might not be enough to overcome static friction. This can cause the ball to briefly get stuck, leading to a steady-state error that persists for a short time. The step response is

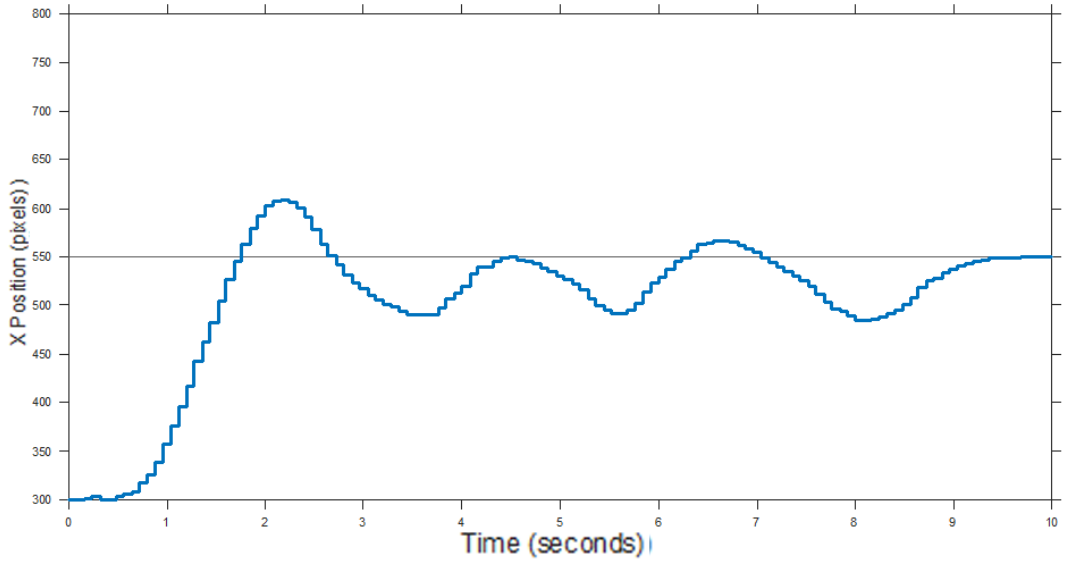
always good to clearly see the quality of the controller's performance, however, the plant should be able to track any reference given to it. The tracking of a square wave reference is shown in fig 5.6, 5.7, 5.8 a and b. The issue of derivative kick is cearly shown in the control output of the figures when the angle of the controller goes extreme due to a sudden change in the input and it represent a major defect in PIDs controllers



(a) PID Controller 1

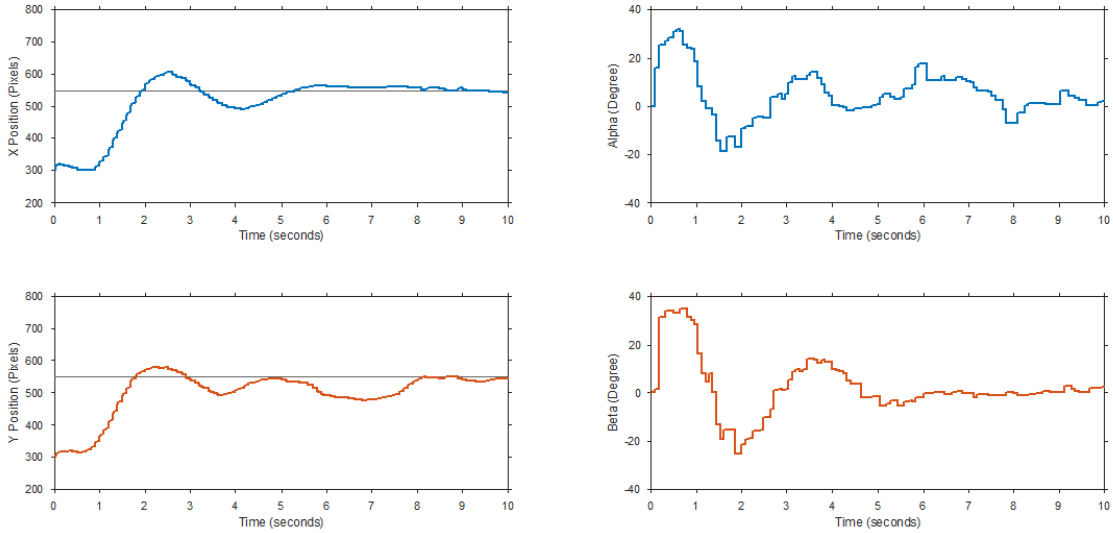


(b) PID Controller 2

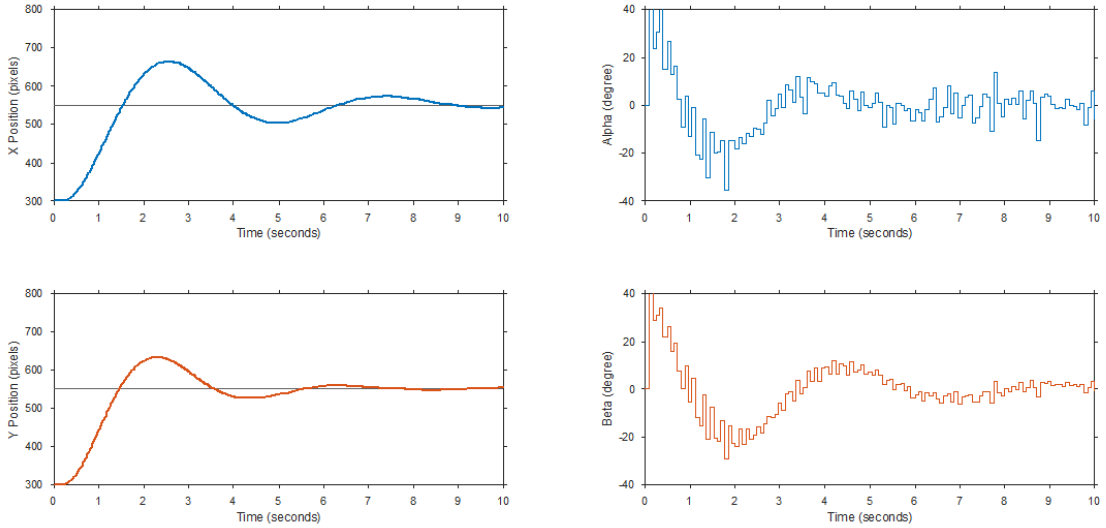


(c) PID Controller 3

Figure 5.2: Experimental Step Response and Control Output of the BPS.

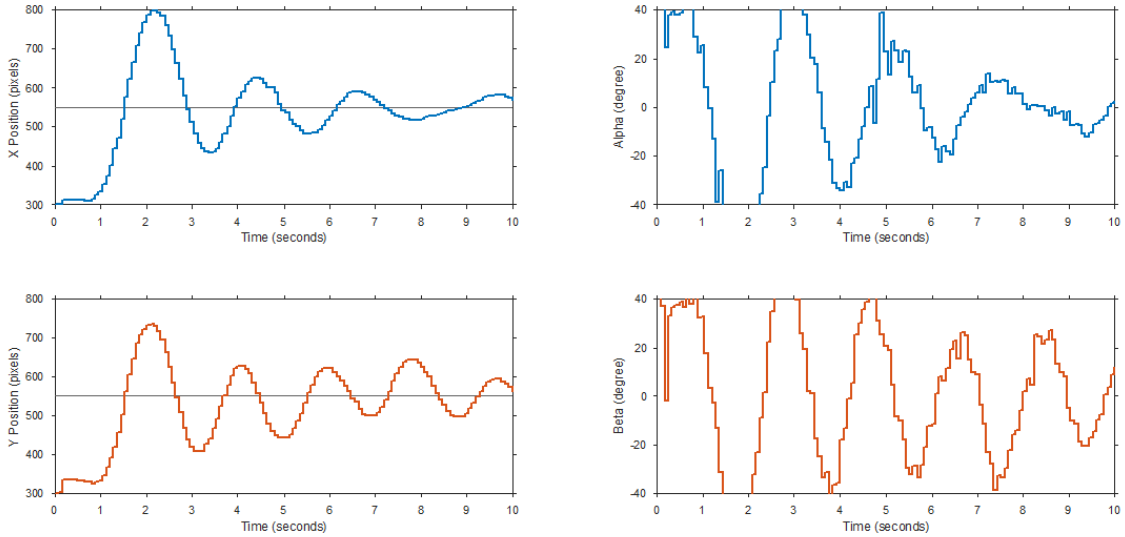


(a) Experimental results

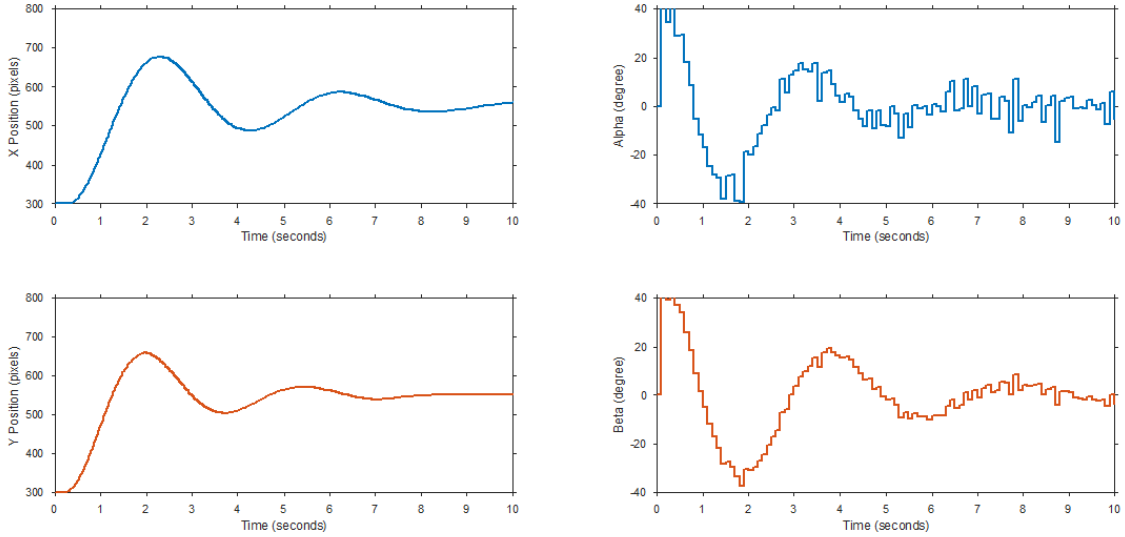


(b) Simulation result

Figure 5.3: Step Response and Control Output of the BPS under parameters of PID 1.

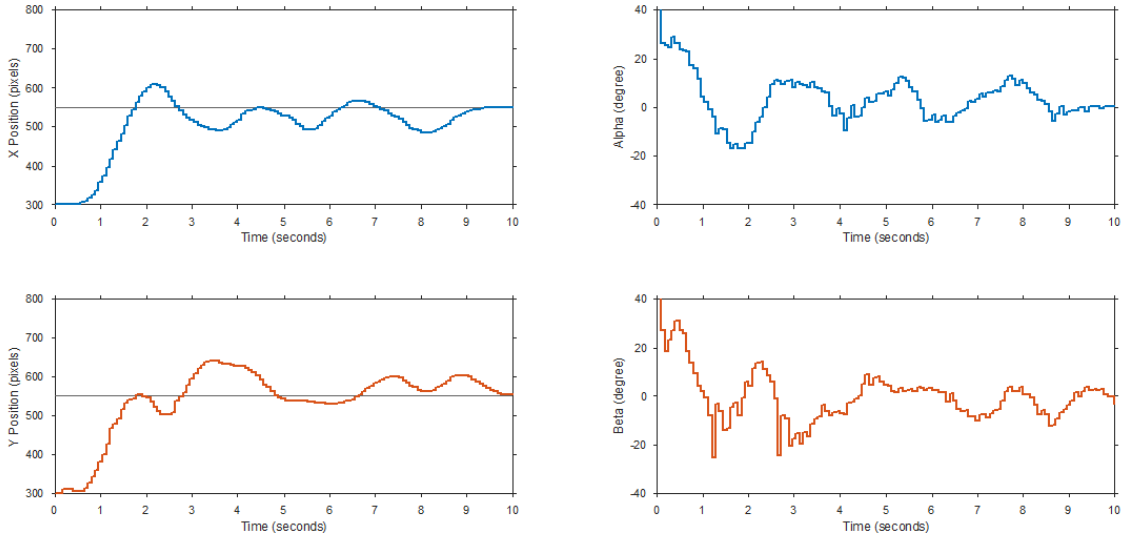


(a) Experimental results

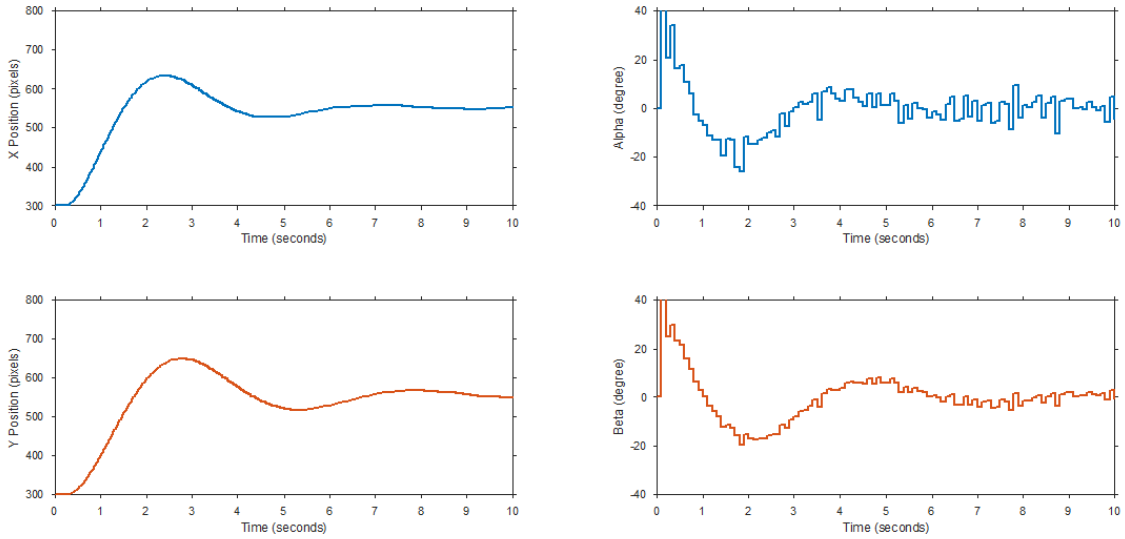


(b) Simulation result

Figure 5.4: Step Response and Control Output of the BPS under parameters of PID 2.

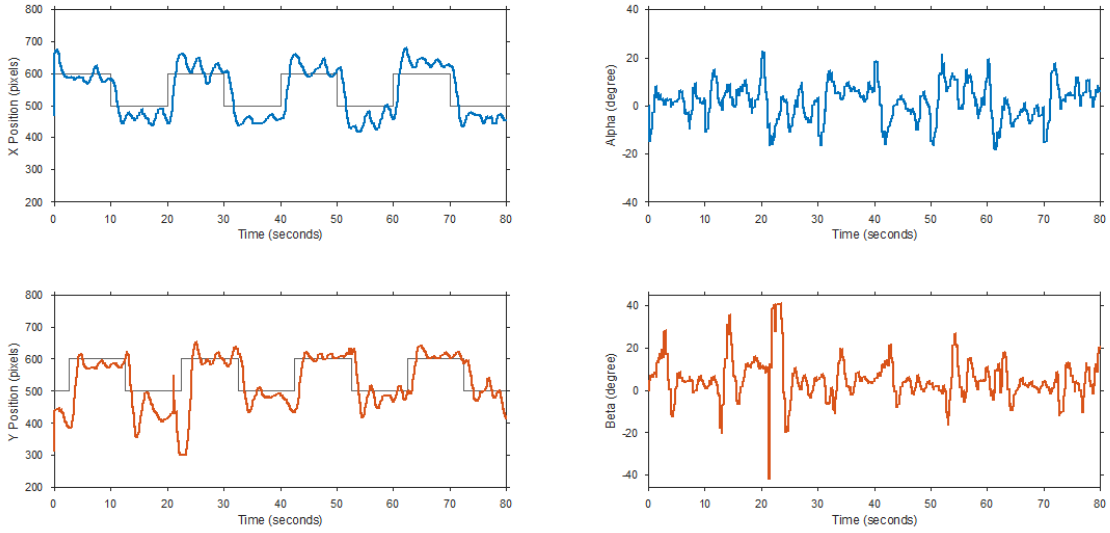


(a) Experimental results

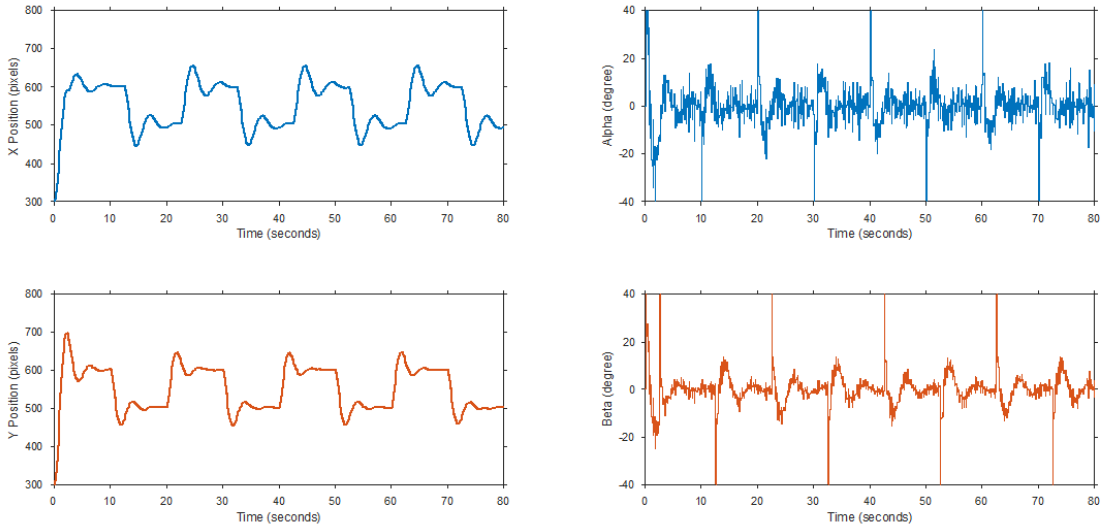


(b) Simulation result

Figure 5.5: Step Response and Control Output of the BPS under parameters of PID 3.

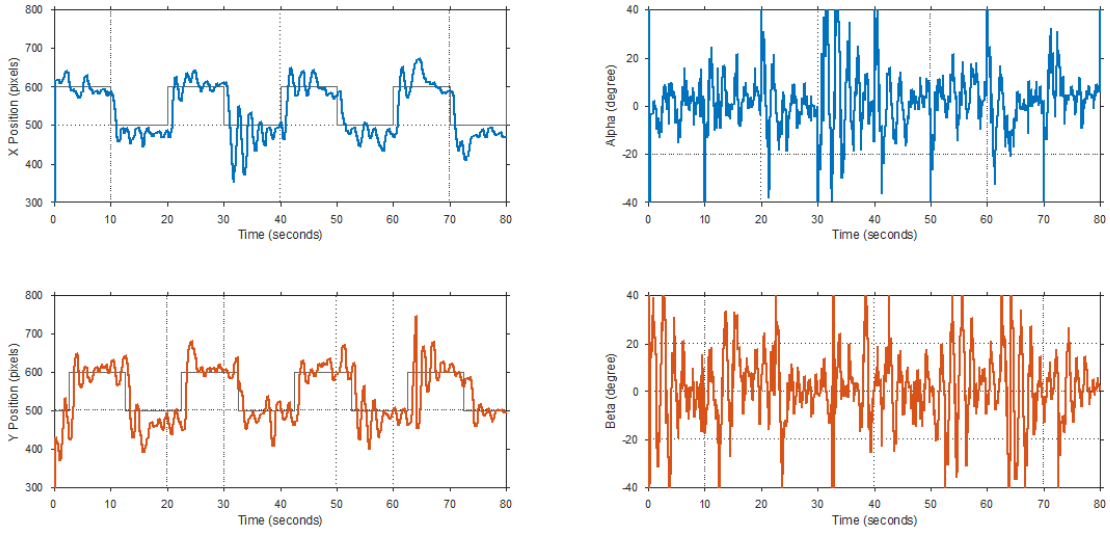


(a) Experimental results

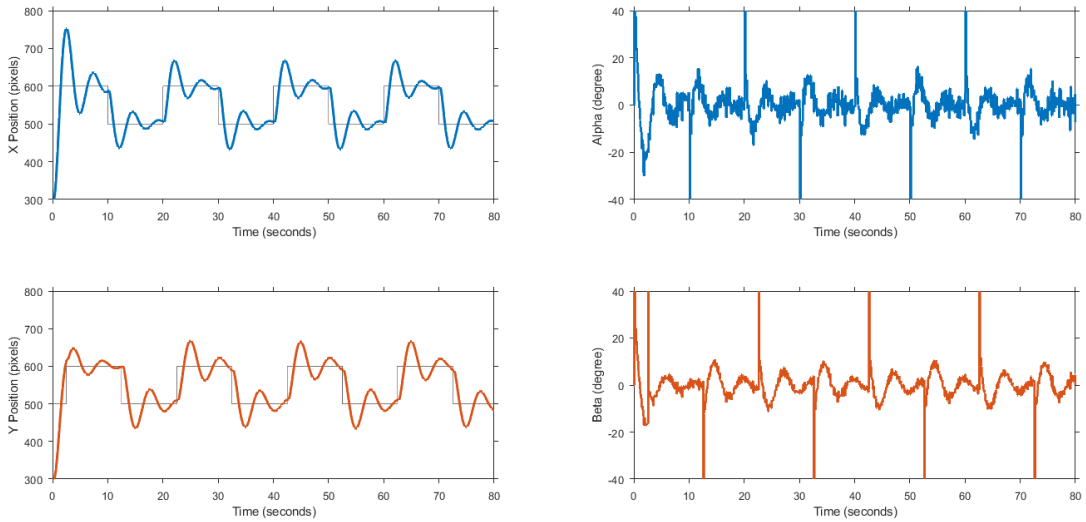


(b) Simulation result

Figure 5.6: Square trajectory Response and Control Output of the BPS under parameters of PID 1.

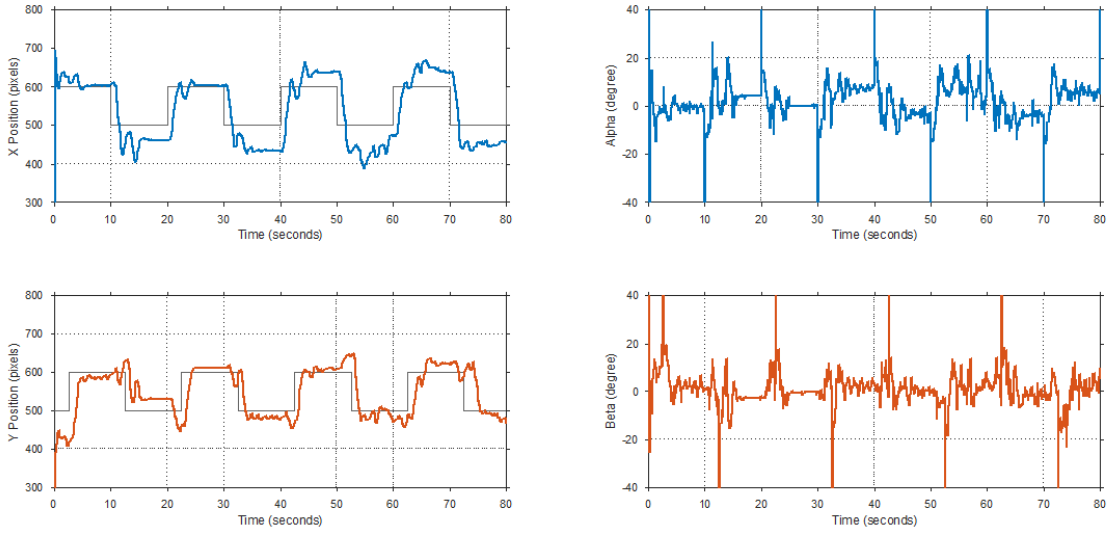


(a) Experimental results

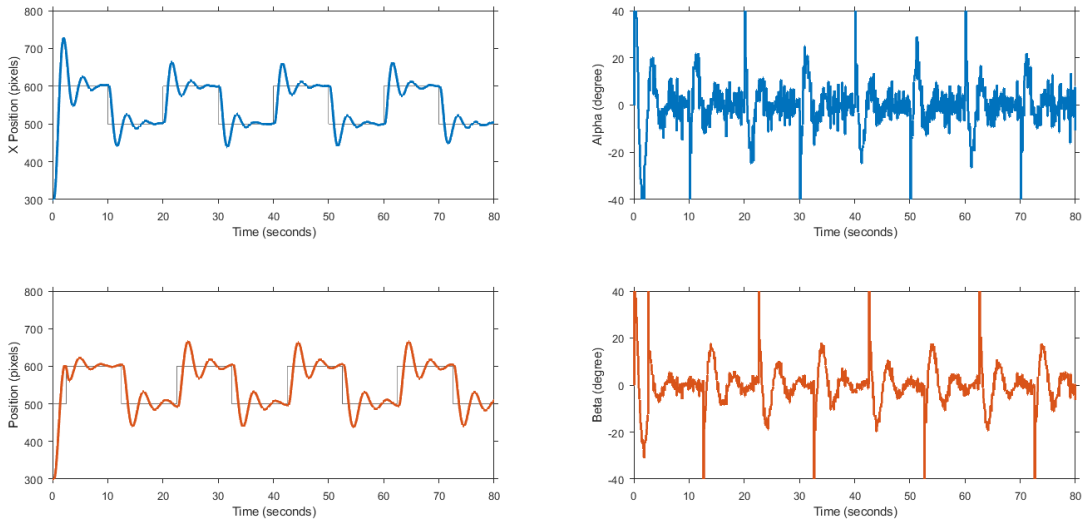


(b) Simulation result

Figure 5.7: Square trajectory Response and Control Output of the BPS under parameters of PID 2.



(a) Experimental results



(b) Simulation result

Figure 5.8: Square trajectory Response and Control Output of the BPS under parameters of PID 3.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

- The theoretical Ball-and-Plate System (BPS) model were established and simulated, they shown promising results. The experimental implementation and testing, were less quality by the lack of adequate hardware, mainly the rods and actuators, however it still do a very good functional work
- The BPS presents an ideal platform for control engineering students to experiment with and validate control methodologies. This is attributed to the system's cost-effectiveness and the ease with which the response can be visually observed, making it a practical and insightful tool for educational purposes.
- Software and programming for controlling any system is as important as theoretical control analysis, which can be an issue when approaching the challenge of building a system from an exclusive theoretical background.
- PID controllers remain one of the most widely employed feedback controllers. They operate by calculating an error value, representing the difference between a measured process variable and a desired set point. The controller aims to minimize this error by making adjustments to the process control inputs. However, it's important to note that PID controllers have limitations when it comes to rapidly stabilizing and mitigating the impact of disturbances

6.2 Future Work

Numerous ways for enhancing both the plant design and controller performance are available for exploration

- While the study relied on the resistive touchscreen for data acquisition, integrating a top camera could offer a more suitable solution. The touch screen, despite filtering efforts, exhibited intolerable noise and discontinuities in readings. Transitioning to a camera-based approach presents a promising alternative for precise ball position determination.
- More advanced control algorithms can be explored such as Model Predictive Control (MPC), Optimal Control (LQR) or Adaptive Control, to potentially achieve superior system stability and performance.

Appendix A

Arduino code

```
/*
 * Program:
 *  5_Wire_Read_servomotor_write_PID_calculate
 * Created by:
 *  Mohammed S. baayou
 * On date:
 *  2023-8-5
 *
 * Description: This program controls a ball and plate system using a 5-wire touch panel,
 *              servo motors, and PID control to maintain the ball's position at the center.
 * Function:
 *      1. Detects ball placement on the plate using a 5-wire resistive touch panel.
 *      2. Reads ball position coordinates using a state machine approach.
 *      3. Filters sensor readings to reduce noise.
 *      4. Calibrates readings for accurate position determination.
 *      5. Implements PID control to calculate precise servo adjustments.
 *      6. Controls servo motors to tilt the plate and keep the ball balanced.
 *      7. Transitions between states to manage system behavior.
 *      8. Enters standby mode when the ball is not present to conserve energy.
 * Plan:
 *  Clean extraneous code
 */

//~~~~~~~~
//~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// LIBRARIES
```

```
29 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
30 #include <Filters.h>
31 #include <Servo.h>
32 #include <PID_v1.h>
33
34 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
35 // Variables
36 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
37 int temp;
38
39 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
40 // State variables
41 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
42 int state;
43 unsigned long state_time;
44 #define STATE_STANDBY 0
45 #define STATE_START 1
46
47
48 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
49 // PID Variables
50 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
51 double SetpointX, InputX, OutputX;
52 double SetpointY, InputY, OutputY;
53
54 float xKp = 4.4;
55 float xKi = 0.01;
56 float xKd = 0.8;
57
58 //stable
59 //float yKp = 2.5;
60 //float yKi = 0.005;
61 //float yKd = 1.260;
62 float yKp = 4.2;
63 float yKi = 0.005;
64 float yKd = 0.80;
65
66 int Ts = 15;
67
68 PID myPIDX(&InputX, &OutputX, &SetpointX, xKp, xKi, xKd,
69 DIRECT);
70 PID myPIDY(&InputY, &OutputY, &SetpointY, yKp, yKi, yKd,
71 DIRECT);
72
73 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
74 // Touch panel variables
75 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
76 #define PIN_TR 2
77 #define PIN_TL 1
78 #define PIN_S  A1
79 #define PIN_SD 5
```

```
80 #define PIN_BL 4
81 #define PIN_BR 3
82 #define SETTLE_TIME 5
83
84
85 float x_pos, y_pos;
86 float pre_pos = 810;
87 unsigned long panel_time;
88 int standByValue = 0 ;
89 unsigned int noTouchCount = 0;
90
91 //Caliboration
92 float Caliboration_Xscale = 800/34;
93 float Caliboration_Yscale = 800/34;
94 float Xoffset = 550;
95 float Yoffset = 550;
96 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
97 // low pass filters variables
98 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
99  float filterFrequencyX= 5;
100  float filterFrequencyY= 4;
101  int InputXfiltered, InputYfiltered;
102
103  // Initializing a one pole (RC) lowpass filter
104  FilterOnePole lowpassFilterX( LOWPASS, filterFrequencyX )
       ;
105  FilterOnePole lowpassFilterY( LOWPASS, filterFrequencyY )
       ;
106
107
108 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
109 // Servo.h Instllation
110 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
111 Servo myservoX;
112 Servo myservoY;
113
114 double angleX;
115 double angleY;
116
117 double flatAngleX = 72;
118 double flatAngleY = 75;
119
120 int stableCount = 0;
121
122 #define PIN_X 9
123 #define PIN_Y 8
124
125 void setup()
126 {
127   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
128   // Establish pin modes
```

```
129   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
130   // Touch panel
131   pinMode(PIN_TR, OUTPUT);
132   pinMode(PIN_TL, OUTPUT);
133   pinMode(PIN_BL, OUTPUT);
134   pinMode(PIN_BR, OUTPUT);
135   digitalWrite(PIN_TR, LOW);
136   digitalWrite(PIN_TL, LOW);
137   digitalWrite(PIN_BL, LOW);
138   digitalWrite(PIN_BR, LOW);
139   pinMode(PIN_S, INPUT);
140
141   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
142   // Initialize variables
143   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
144   // State Machine
145   state = 0;
146
147   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
148   // Servo Motors
149   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
150   // Attach Servo motors to pins
151   myservoX.attach(PIN_X);
152   myservoY.attach(PIN_Y);
153
154   OutputX = flatAngleX;
155   OutputY = flatAngleY;
156
157   myservoX.write(OutputX);
158   myservoY.write(OutputY);
159
160 // Setting starting parameters for PID
161   myPIDX.SetMode(AUTOMATIC);
162   myPIDY.SetMode(AUTOMATIC);
163   InputX = 0;
164   InputY = 0;
165
166   SetpointX = 0.00;
167   SetpointY = 0.00;
168
169   // Setting limiting parameters for servo motors
170   myPIDX.SetOutputLimits(flatAngleX-85, flatAngleX+85);
171   myPIDY.SetOutputLimits(flatAngleY-85, flatAngleY+85);
172
173   myPIDX.SetSampleTime(Ts);
174   myPIDY.SetSampleTime(Ts);
175
176   angleX = flatAngleX;
177   angleY = flatAngleY;
178
179
```

```
180
181
182
183   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
184   // Begin Serial Communication
185   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
186   Serial.begin(9600);
187 }
188
189 void loop()
190 {
191   switch(state)
192   {
193     case 0: //standby mode
194       pinMode(PIN_TR, OUTPUT);
195       pinMode(PIN_TL, OUTPUT);
196       pinMode(PIN_BL, OUTPUT);
197       pinMode(PIN_BR, OUTPUT);
198       digitalWrite(PIN_TR, LOW);
199       digitalWrite(PIN_TL, LOW);
200       digitalWrite(PIN_BL, LOW);
201       digitalWrite(PIN_BR, LOW);
202
203       pinMode(PIN_S, INPUT_PULLUP );
204       delay(10);
205
206       standByValue = analogRead(PIN_S);
207       //Serial.println(noTouchCount);
208       //Serial.print("\t");
209       if(standByValue < 800){
210         state = 1;
211         myservoX.attach(PIN_X);
212         myservoY.attach(PIN_Y);
213         noTouchCount = 0;
214       }
215       else{
216         noTouchCount++;
217
218           myservoX.write(OutputX);
219           myservoY.write(OutputY);
220         if(noTouchCount > 100) //if there is no ball on
    plate longer
221         {
222
223           OutputX=flatAngleX; //make plate flat
224           OutputY=flatAngleY;
225           myservoX.write(OutputX);
226           myservoY.write(OutputY);
227           delay(1000);
228           myservoX.detach(); //detach servos
229           myservoY.detach();
```

```
230           break;
231         }
232
233       }
234       //state=1;
235       break;
236
237
238     case 1: // Set up X read mode
239       //~~~~~~~~~~~~~~~~~
240       // Set up X read
241       //~~~~~~~~~~~~~~~~~
242       // Set TR and BR high
243       digitalWrite(PIN_TR, LOW);
244       digitalWrite(PIN_BR, LOW);
245       // Set TL and BL low
246       digitalWrite(PIN_TL, HIGH);
247       digitalWrite(PIN_BL, HIGH);
248
249       //~~~~~~~~~~~~~~~~~
250       // Move to next state
251       //~~~~~~~~~~~~~~~~~
252       // Record the time
253       panel_time = millis();
254       // Switch to waiting for the panel to settle
255       state = 2;
256       break;
257
258     case 2: // Reading X position mode
259       // How long has it been since the panel changed
    configuration?
260       temp = millis() - panel_time;
261       // Has it been long enough?
262       if (temp >= SETTLE_TIME)
263       { // If it has...
264         pre_pos = x_pos;
265         x_pos = analogRead(PIN_S);
266         //filtering no touch spikes
267         if (x_pos > 900){
268           x_pos = pre_pos;
269         }
270         // Switch to the Read X state
271         state = 3;
272       } // Otherwise, do nothing.
273       break;
274
275     case 3: // Set up Y read mode
276       //~~~~~~~~~~~~~~~~~
277       // Set up Y read
278       //~~~~~~~~~~~~~~~~~
279       // Set TR and TL high
```

```
280       digitalWrite(PIN_TR, HIGH);
281       digitalWrite(PIN_TL, HIGH);
282       // Set BL and BR low
283       digitalWrite(PIN_BL, LOW);
284       digitalWrite(PIN_BR, LOW);
285
286       //~~~~~~~~~~~~~~~~
287       // Move to next state
288       //~~~~~~~~~~~~~~~~
289       // Record the time
290       panel_time = millis();
291       // Switch to waiting for the panel to settle
292       state = 4;
293
294       break;
295
296     case 4: // Reading Y position mode
297       // How long has it been since the panel changed
    configuration?
298       temp = millis() - panel_time;
299
300       // Has it been long enough?
301       if (temp >= SETTLE_TIME)
302       { // If it has...
303         // Read the sensor voltage (Y position)
304         pre_pos = y_pos;
305         y_pos = analogRead(PIN_S);
306         if (y_pos > 800){
307           y_pos = pre_pos;
308         }
309           state = 5;
310       } // Otherwise, do nothing.
311       break;
312
313
314     case 5: // testing & filtering coordinates mode
315       //----------------------------------
316       //Serial.print(x_pos);
317       //Serial.print("\t");
318       //Serial.print(y_pos);
319       //Serial.print("\t");
320
321       // Filtering touchpanel signal
322       x_pos = lowpassFilterX.input(x_pos);
323       y_pos = lowpassFilterY.input(y_pos);
324
325       // Write the coordinates to Serial for debuging
326       //Serial.print("filtered coordinates  ");
327       //Serial.print(x_pos);
328       //Serial.print("\t");
329       //Serial.print(y_pos);
```

```
330       //Serial.print("\n");
331
332       state = 6;
333       //---------------------------------
334       break;
335
336     case 6: // Caliboration & checking if the ball in the
    center
337       InputX = (x_pos - Xoffset) / Caliboration_Xscale;
338       InputY = (y_pos - Yoffset) / Caliboration_Yscale;
339       //Serial.print("Caliborated position  ");
340       Serial.print("\t");
341       Serial.print(InputX);
342       Serial.print("\t");
343       Serial.print(InputY);
344       //Serial.print("\n");
345
346       if(InputX <= SetpointX + 0.85 && InputX >= SetpointX
     - 0.85 && InputY <= SetpointY + 1.25 && InputY >=
    SetpointY - 1.25 ){
347         stableCount++;
348       }
349       else{
350         stableCount =- 2;
351       }
352
353       if(stableCount > 150){
354         myservoX.write(flatAngleX);
355         myservoY.write(flatAngleY);
356         //delay(1000);
357         state = 8;
358         break;
359       }
360       state = 7;
361       break;
362
363     case 7: // setting PID outputs
364
365
366       myPIDX.Compute();  //action control X compute
367       myPIDY.Compute();  //   action control  Y compute
368
369       myservoX.write(OutputX);
370       myservoY.write(OutputY);
371
372       //Serial.print("servos angle  ");
373       //Serial.print("\t");
374       //Serial.print(OutputX);
375       //Serial.print("\t");
376       //Serial.print(OutputY);
377       Serial.println("\n");
```

```
378
379       state = 8;
380       break;
381
382
383     case 8: // Turn off the panel
384       // Turn off the panel
385       digitalWrite(PIN_TR, LOW);
386       digitalWrite(PIN_TL, LOW);
387       digitalWrite(PIN_BL, LOW);
388       digitalWrite(PIN_BR, LOW);
389
390       // Switch to button debounce state
391       state = 9;
392       break;
393
394
395     case 9: // End state
396
397       // Send message through Serial
398       //Serial.println("Panel read finished");
399       delay(20);
400
401       // Switch to the waiting state
402       state = 0;
403       break;
404
405     default: // Error
406       Serial.println("Something has gone terribly wrong.")
    ;
407       break;
408     }
409   }
```

Listing A.1: Arduino full code for controlling the system by internal PID

```
1  /*
2   * Program:
3   *  5_Wire_Read_servomotor_write_PID_calculate
4   * Created by:
5   *  Mohammed S. baayou
6   * On date:
7   *  2023-8-5
8   *
9   * Description: This program controls a ball and plate
      system using a 5-wire touch panel,
10  *              servo motors, and PID control to maintain
      the ball's position at the center.
11  * Function:
12  *      1. Detects ball placement on the plate using a 5-
      wire resistive touch panel.
13  *      2. Reads ball position coordinates using a state
```

```
    machine approach.
 *      3. Filters sensor readings to reduce noise.
 *      4. Calibrates readings for accurate position
    determination.
 *      5. Implements PID control to calculate precise
    servo adjustments.
 *      6. Controls servo motors to tilt the plate and
    keep the ball balanced.
 *      7. Transitions between states to manage system
    behavior.
 *      8. Enters standby mode when the ball is not
    present to conserve energy.
 * Plan:
 *  Clean comments and extraneous code
 */

//~~~~~~~~
//~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// LIBRARIES
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#include <Servo.h>
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
int temp;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// State variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
int state;
unsigned long state_time;
#define STATE_STANDBY 0
#define STATE_START 1


int Ts = 15;



//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Touch panel variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#define PIN_TR 3
#define PIN_TL 2
#define PIN_S  A1
#define PIN_SD 6
#define PIN_BL 5
#define PIN_BR 4
#define SETTLE_TIME 5
```

```
59
60
61 float x_pos, y_pos;
62 float pre_pos = 810;
63 unsigned long panel_time;
64 int standByValue = 0 ;
65 unsigned int noTouchCount = 0;
66 char header;
67
68 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
69 //Caliboration
70 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
71 float Caliboration_Xscale = 800/34;
72 float Caliboration_Yscale = 800/28;
73 float Xoffset = 550;
74 float Yoffset = 550;
75
76 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
77 // Servo.h Instllation
78 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
79 Servo myservoX;
80 Servo myservoY;
81 double angleX;
82 double angleY;
83 double flatAngleX = 72;
84 double flatAngleY = 75;
85 //int outputX = flatAngleX;
86 //int outputY = flatAngleY;
87 #define PIN_X 9
88 #define PIN_Y 8
89
90 int stableCount = 0;
91
92 typedef union{
93   float number;
94   uint8_t bytes[4];
95 } FLOATUNION_t;
96
97 FLOATUNION_t sendX1;
98 FLOATUNION_t sendY2;
99
100 FLOATUNION_t xValue;
101 FLOATUNION_t yValue;
102
103 void setup()
104 {
105   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
106   // Establish pin modes
107   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
108   // Touch panel
109   pinMode(PIN_TR, OUTPUT);
```

```
110   pinMode(PIN_TL, OUTPUT);
111   pinMode(PIN_BL, OUTPUT);
112   pinMode(PIN_BR, OUTPUT);
113   digitalWrite(PIN_TR, LOW);
114   digitalWrite(PIN_TL, LOW);
115   digitalWrite(PIN_BL, LOW);
116   digitalWrite(PIN_BR, LOW);
117   pinMode(PIN_S, INPUT);
118
119   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
120   // Initialize variables
121   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
122   // State Machine
123   state = 0;
124
125
126 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
127   // Servo Motors
128   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
129   // Attach Servo motors to pins
130   myservoX.attach(PIN_X);
131   myservoY.attach(PIN_Y);
132
133   xValue.number = flatAngleX;
134   yValue.number = flatAngleY;
135   angleX = flatAngleX;
136   angleY = flatAngleY;
137
138   myservoX.write(flatAngleX);
139   myservoY.write(flatAngleY);
140
141
142
143   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
144   // Begin Serial Communication
145   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
146   Serial.begin(9600);
147 }
148
149 void loop()
150 {
151   switch(state)
152   {
153     case 0: //standby mode
154       pinMode(PIN_TR, OUTPUT);
155       pinMode(PIN_TL, OUTPUT);
156       pinMode(PIN_BL, OUTPUT);
157       pinMode(PIN_BR, OUTPUT);
158       digitalWrite(PIN_TR, LOW);
159       digitalWrite(PIN_TL, LOW);
160       digitalWrite(PIN_BL, LOW);
```

```
161       digitalWrite(PIN_BR, LOW);
162
163       pinMode(PIN_S, INPUT_PULLUP );
164       delay(10);
165
166       standByValue = analogRead(PIN_S);
167
168       if(standByValue < 800){
169         state = 1;
170         myservoX.attach(PIN_X);
171         myservoY.attach(PIN_Y);
172         noTouchCount = 0;
173       }
174       else{
175         noTouchCount++;
176         myservoX.write(angleX);
177         myservoY.write(angleY);
178         if(noTouchCount > 100) //if there is no ball on
    plate longer
179         {
180           myservoX.write(flatAngleX);
181           myservoY.write(flatAngleY);
182           delay(1000);
183           myservoX.detach(); //detach servos
184           myservoY.detach();
185           break;           }
186
187       }
188       break;
189
190
191      case 1: // Set up X read mode
192       //~~~~~~~~~~~~~~~~~
193       // Set up X read
194       //~~~~~~~~~~~~~~~~~
195       // Set TR and BR high
196       digitalWrite(PIN_TR, LOW);
197       digitalWrite(PIN_BR, LOW);
198       // Set TL and BL low
199       digitalWrite(PIN_TL, HIGH);
200       digitalWrite(PIN_BL, HIGH);
201
202       //~~~~~~~~~~~~~~~~~
203       // Move to next state
204       //~~~~~~~~~~~~~~~~~
205       // Record the time
206       panel_time = millis();
207       // Switch to waiting for the panel to settle
208       state = 2;
209       break;
210
```

```
211      case 2: // Reading X position mode
212        // How long has it been since the panel changed
       configuration?
213        temp = millis() - panel_time;
214        // Has it been long enough?
215        if (temp >= SETTLE_TIME)
216        { // If it has...
217          pre_pos = x_pos;
218          x_pos = analogRead(PIN_S);
219          //filtering no touch spikes
220          if (x_pos > 900){
221            x_pos = pre_pos;
222          }
223          // Switch to the Read X state
224          state = 3;
225        } // Otherwise, do nothing.
226        break;
227
228      case 3: // Set up Y read mode
229        //~~~~~~~~~~~~~~~~~
230        // Set up Y read
231        //~~~~~~~~~~~~~~~~~
232        // Set TR and TL high
233        digitalWrite(PIN_TR, HIGH);
234        digitalWrite(PIN_TL, HIGH);
235        // Set BL and BR low
236        digitalWrite(PIN_BL, LOW);
237        digitalWrite(PIN_BR, LOW);
238
239        //~~~~~~~~~~~~~~~~~
240        // Move to next state
241        //~~~~~~~~~~~~~~~~~
242        // Record the time
243        panel_time = millis();
244        // Switch to waiting for the panel to settle
245        state = 4;
246
247        break;
248
249      case 4: // Reading Y position mode
250        // How long has it been since the panel changed
       configuration?
251        temp = millis() - panel_time;
252
253        // Has it been long enough?
254        if (temp >= SETTLE_TIME)
255        { // If it has...
256          // Read the sensor voltage (Y position)
257          pre_pos = y_pos;
258          y_pos = analogRead(PIN_S);
259          if (y_pos > 800){
```

```
260           y_pos = pre_pos;
261         }
262           state = 5;
263       } // Otherwise, do nothing.
264       break;
265
266
267     case 5: // packeting & sending coordinates via serial
    communication
268
269       //-----------------------------------
270       //x_pos = (x_pos - Xoffset) / Caliboration_Xscale;
271       //y_pos = (y_pos - Yoffset) / Caliboration_Yscale;
272       //-----------------------------------
273       sendX1.number = x_pos;
274       sendY2.number = y_pos;
275       if( x_pos > 0 && y_pos > 0){
276
277         Serial.print("A");
278
279         for (int i=0; i<4; i++){
280           Serial.write(sendX1.bytes[i]);
281         }
282
283         for (int i=0; i<4; i++){
284           Serial.write(sendY2.bytes[i]);
285         }
286
287         Serial.print("\n");
288         delay(50);
289
290         state = 6;
291         //---------------------------------
292         break;
293       }
294       state = 0;
295       break;
296
297
298     case 6: // recieving servo angles
299
300       if (Serial.available() >= 8) {
301         xValue.number = getFloat();
302         angleX = xValue.number;
303         yValue.number = getFloat();
304         angleY = yValue.number;
305       }
306
307       if(isnan(angleX) || isnan(angleY) || angleY == 0 ||
    angleX == 0){
308         //Serial.flush();
```

```
309           state = 7;
310           break;
311         }
312
313         myservoX.write(angleX);
314         myservoY.write(angleY);
315
316         state = 7;
317         break;
318
319
320       case 7: // Turn off the panel
321         // Turn off the panel
322         digitalWrite(PIN_TR, LOW);
323         digitalWrite(PIN_TL, LOW);
324         digitalWrite(PIN_BL, LOW);
325         digitalWrite(PIN_BR, LOW);
326
327         // Switch to button debounce state
328         state = 0;
329         break;
330
331       default: // Error
332         Serial.println("Something has gone terribly wrong.")
      ;
333         break;
334       }
335     }
336
337 float getFloat(){
338       int cont = 0;
339       FLOATUNION_t f;
340       while (cont < 4 ){
341           f.bytes[cont] = Serial.read() ;
342           cont = cont +1;
343       }
344       return f.number;
345 }
```

Listing A.2: Arduino full code for controlling the system by a simulink controller

Appendix B

Simulink Block diagram

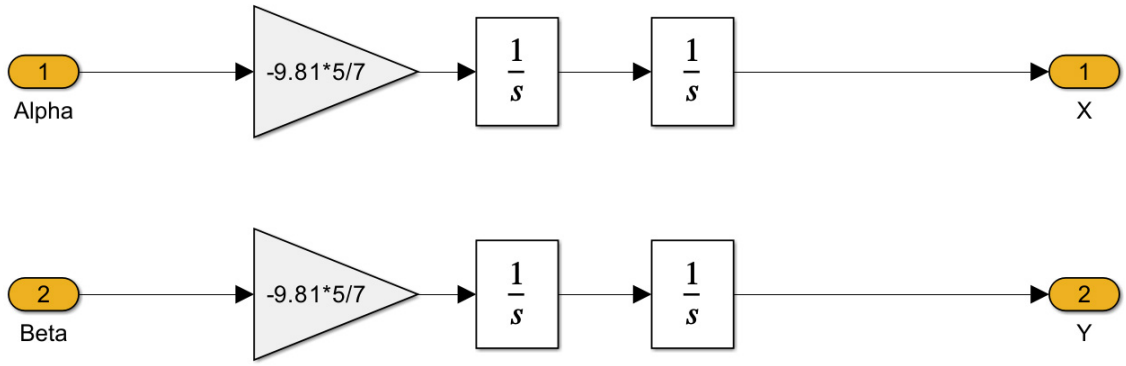


Figure B.1: Linear simulation model of the Ball and Plate System in a transfer function form

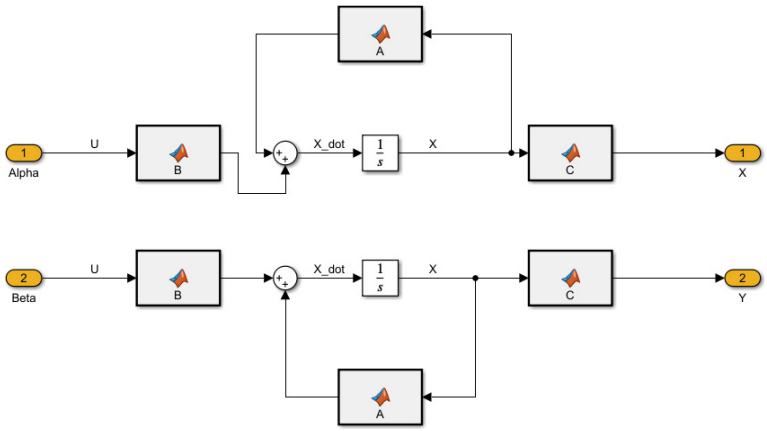


Figure B.2: Linear simulation model of the Ball and Plate System in state-space form

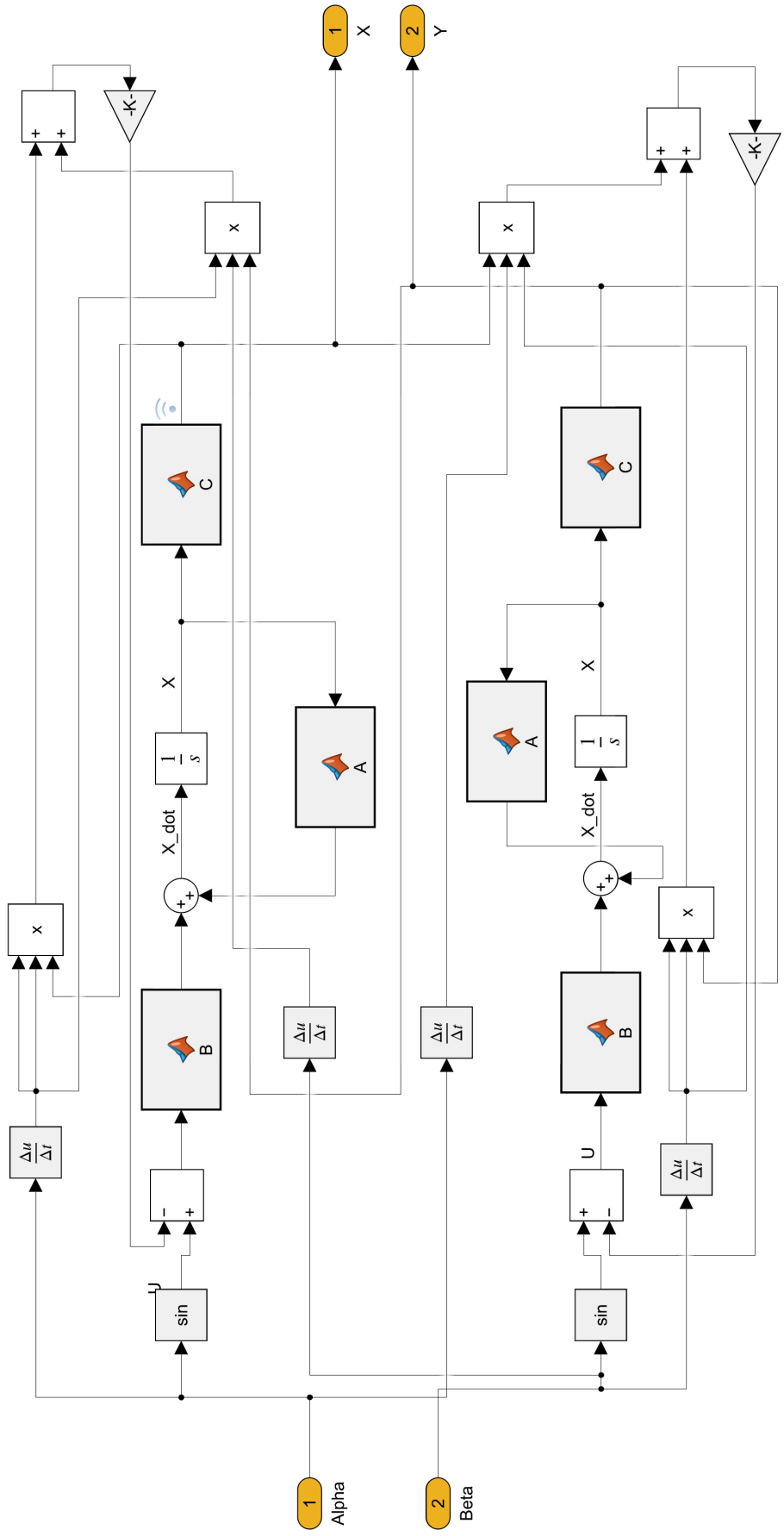


Figure B.3: Approximated representation of the non-linear BPS dynamics in a Simulink model

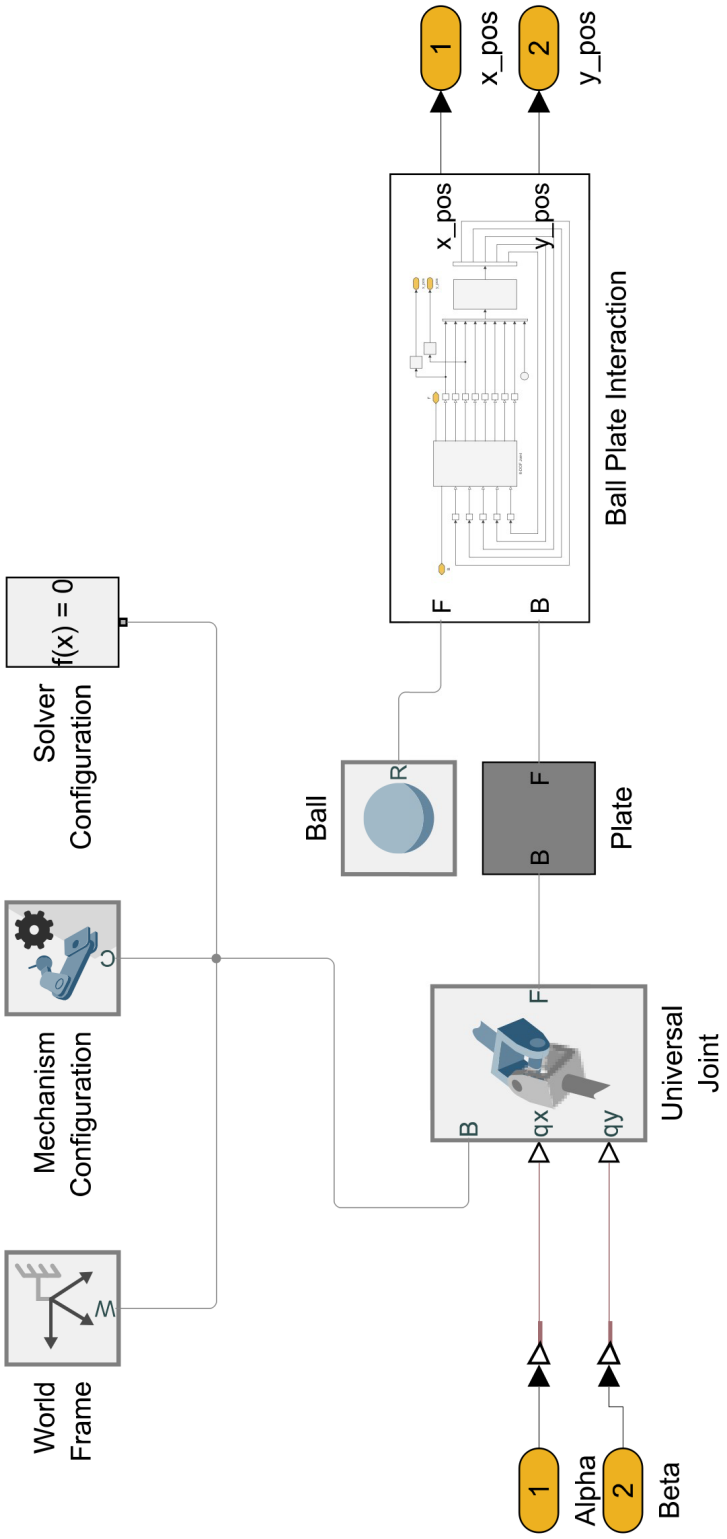


Figure B.4: Dynamic simulation of the non-linear Ball and Plate System using Simscape

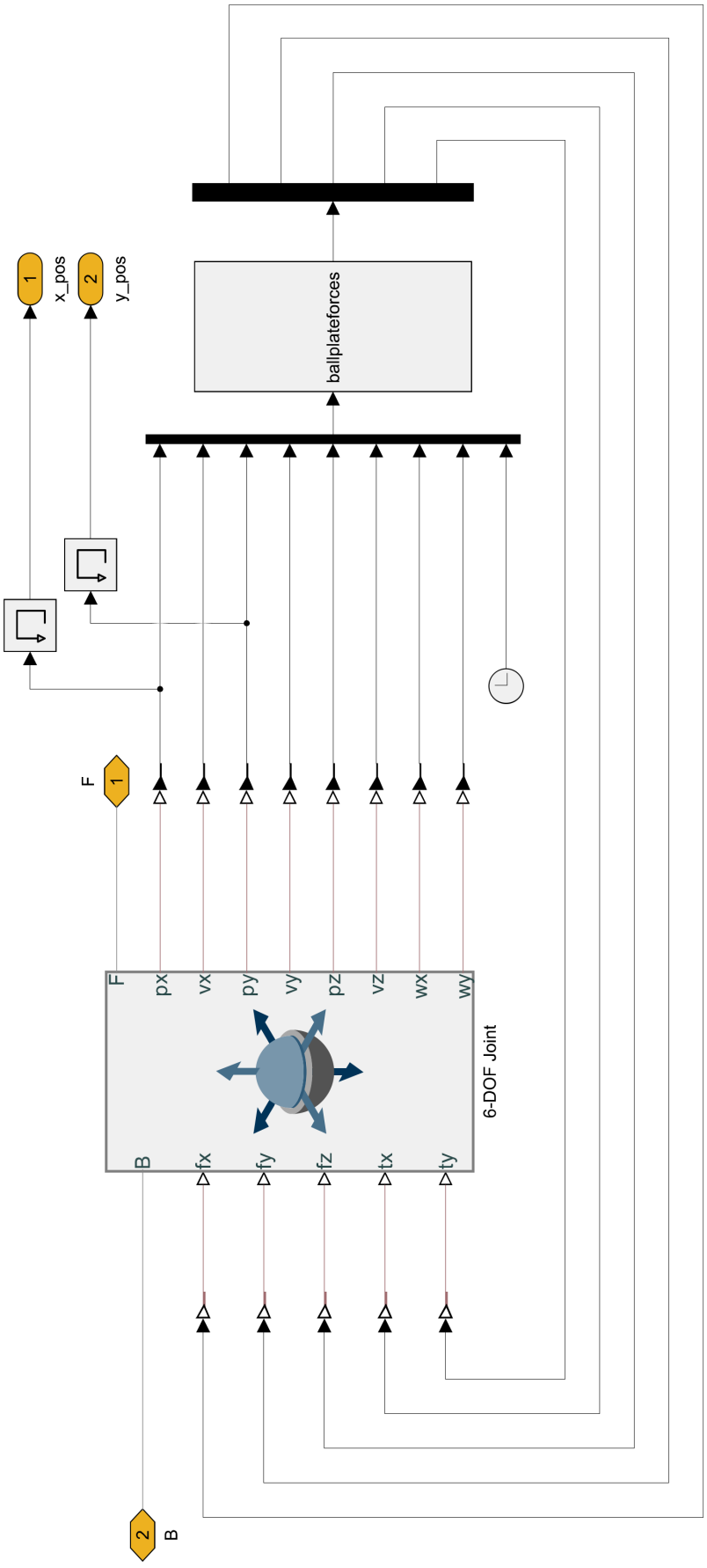


Figure B.5: BPS interaction subsystem

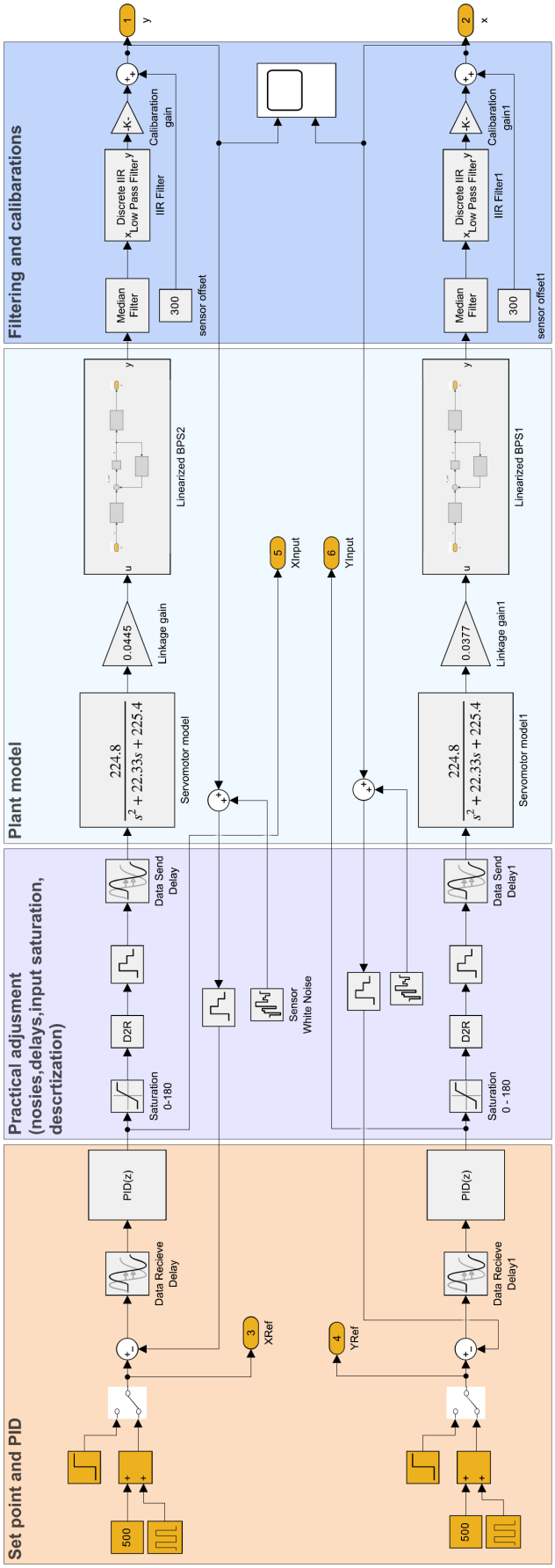


Figure B.6: PID Simulink Model Illustrating Practical Adjustments: saturation, calibration, white noises, communication delays

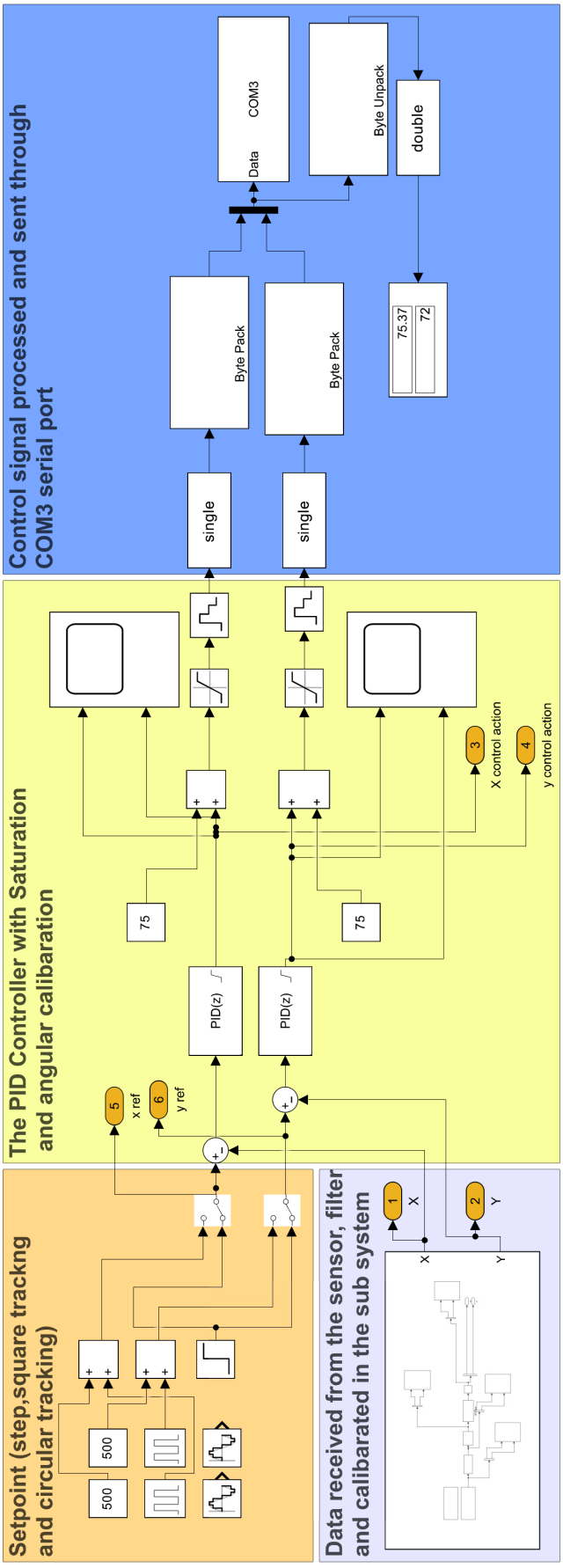


Figure B.7: Simulink Model used for Controlling the real system

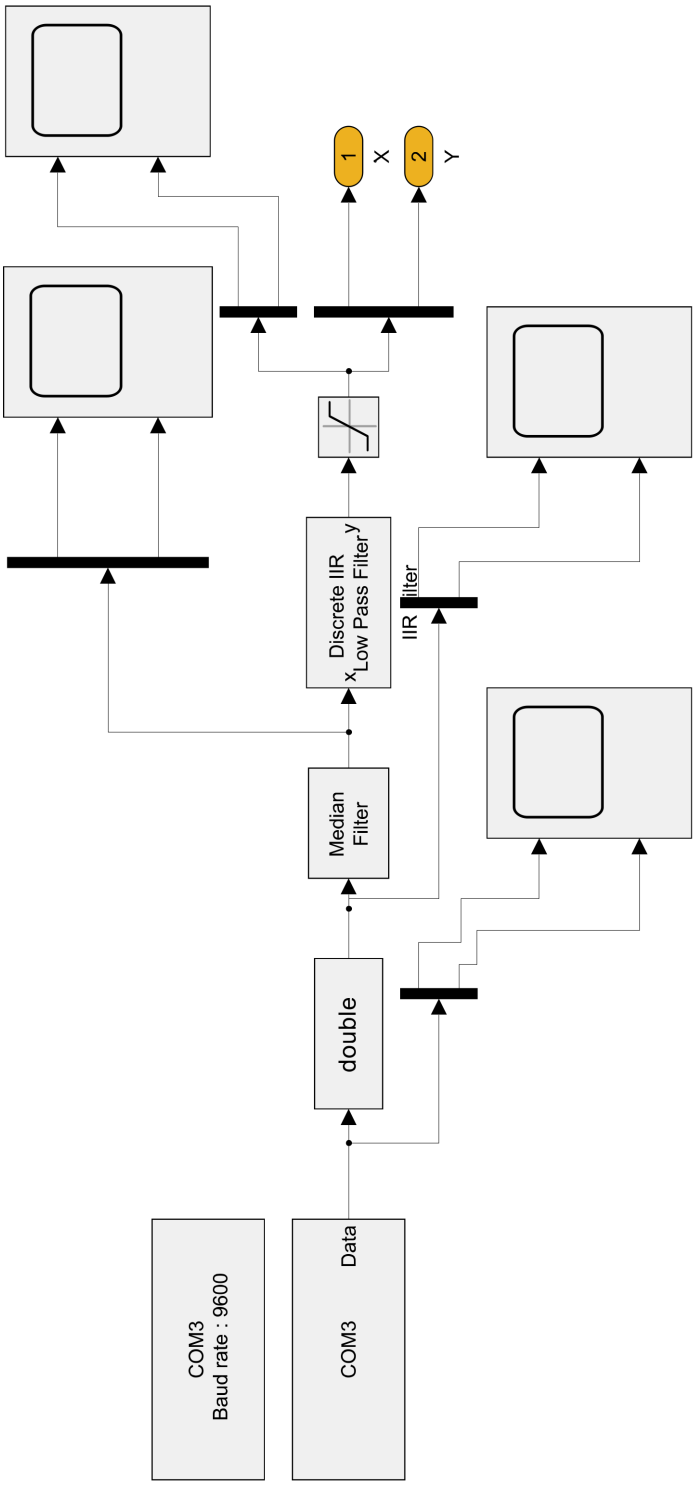


Figure B.8: Simulink Block used for receiving the feedback data

Appendix C

Method of mapping the angles

In this section, the angular mapping method is utilized to establish the relationship between the input angles and the corresponding output angles of the servomotors. The measurements were obtained using a smartphone, and the linear gains for the mapping were determined as $k_x = 0.0445$ and $_y = 0.0377$ by the following matlab code:

```
input = [0,5,10,15,20,25,30,
35,40,45,50,55,60,65,70,75,
80,85,90,95,100,105,110,115,
120,125,130,135,140,145,150,
155,160,165,170,175,180];
output = [-7.9,-7.8,-7.6,-7.2,
-6.9,-6.4,-6,-5.1,-4.6,-3.8,-3,-2.1,-1.4,-0.5,
   0.1,0.1,0.5,0.8,1.3,2,2.8,3.6,4.4,5,5.7,6.4,
7,7.6,8.1,9,9.6,9.8,9.9,10.1,10.3,10.4,10.5];
degree = 1;
% polyfit(input,output,degree);
gx = output/input
inputy = [0,5,10,15,20,25,30,35,40,45,
50,55,60,65,70,75,80,85,90,95,100,
105,110,115,120,125,130,135,140,145,150];
outputy=  [7.3,7,6.8,6.6,6.2,5.8,5.3,
4.6,4,3.4,2.7,2.1,1.5,0.7,0.2,0,
-0.9,-1.7,-2.3,-2.9,-3.8,-4.6,-5.3,
-5.8,-6.5,-7,-7.7,-8,-8.6,-9,-9.1];
gy = outputy/inputy
```

Listing C.1: matlab code for polynomial fits

Mapping Values

The following table presents the measured input and output angles along with the calculated gains.

Servomotor Angle (degrees)	**Inclination Angle X (degrees)**	**Inclination Angle Y (degrees)**
0	-7.9	7.3
5	-7.8	7.0
10	-7.6	6.8
15	-7.2	6.6
20	-6.9	6.2
25	-6.4	5.8
30	-6.0	5.3
35	-5.1	4.6
40	-4.6	4.0
45	-3.8	3.4
50	-3.0	2.7
55	-2.1	2.1
60	-1.4	1.5
65	-0.5	0.7
70	-0.1	0.2
75	0.1	0.0
80	0.5	-0.9
85	0.8	-1.7
90	1.3	-2.3
95	2.0	-2.9
100	2.8	-3.8
105	3.6	-4.6
110	4.4	-5.3
115	5.0	-5.8
120	5.7	-6.5
125	6.4	-7.0
130	7.0	-7.7
135	7.6	-8.0
140	8.1	-8.6
145	9.0	-9.0
150	9.6	-9.1
155	9.8	
160	9.9	
165	10.1	
170	10.3	
175	10.4	
180	10.5	

Table C.1: Mapping values for input and output angles to calculate the gains.

Bibliography

[1] J. C. Maxwell, "I. on governors," *Proceedings of the Royal Society of London*, no. 16, pp. 270–283, 1868.

[2] N. Wiener, *Cybernetics or Control and Communication in the Animal and the Machine*. MIT press, 2019.

[3] A. M. Lyapunov, *General problem of the stability of motion*, vol. 55. CRC Press, 1992.

[4] R. E. Kalman *et al.*, "Contributions to the theory of optimal control," *Bol. soc. mat. mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[5] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[6] S. Awtar, C. Bernard, N. Boklund, A. Master, D. Ueda, and K. Craig, "Mechatronic design of a ball-on-plate balancing system," *Mechatronics*, vol. 12, no. 2, pp. 217–228, 2002.

[7] C. Ham and M. M. Taufiq, "Development of a ball and plate system," in *2015 ASEE Annual Conference & Exposition*, pp. 26–518, 2015.

[8] G. Madhumitha, S. S. Kumar, and A. Gurupriya, "Design and development of a ball-plate balancing system with a smart phone human-machine interface," in *Journal of Physics: Conference Series*, vol. 1969, p. 012057, IOP Publishing, 2021.

[9] B. Y. Zhao and X. L. Li, "On multiple model adaptive control strategy research of ball and plate system," *Advanced Materials Research*, vol. 898, pp. 501–505, 2014.

[10] K. Han, Y. Tian, Y. Kong, J. Li, and Y. Zhang, "Tracking control of ball and plate system using a improved pso on-line training pid neural network," in *2012 IEEE international conference on mechatronics and automation*, pp. 2297–2302, IEEE, 2012.

[11] S. R. Bdoor, O. Ismail, M. R. Roman, and Y. Hendawi, "Design and implementation of a vision-based control for a ball and plate system," in *2016 2nd International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, pp. 1–4, IEEE, 2016.

[12] D. Debono and M. Bugeja, "Application of sliding mode control to the ball and plate problem," in *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 1, pp. 412–419, IEEE, 2015.

[13] D. Yuan and Z. Zhang, "Modelling and control scheme of the ball–plate trajectory-tracking pneumatic system with a touch screen and a rotary cylinder," *IET control theory & applications*, vol. 4, no. 4, pp. 573–589, 2010.

[14] A. Das and P. Roy, "Improved performance of cascaded fractional-order smc over cascaded smc for position control of a ball and plate system," *IETE Journal of Research*, vol. 63, no. 2, pp. 238–247, 2017.

[15] A. Adiprasetya and A. S. Wibowo, "Implementation of pid controller and pre-filter to control non-linear ball and plate system," in *2016 International conference on control, electronics, renewable energy and communications (ICCEREC)*, pp. 174–178, IEEE, 2016.

[16] D. Stander, S. Jiménez-Leudo, and N. Quijano, "Low-cost "ball and plate" design and implementation for learning control systems," in *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*, pp. 1–6, IEEE, 2017.

[17] F. I. R. Betancourt, S. M. B. Alarcon, and L. F. A. Velasquez, "Fuzzy and pid controllers applied to ball and plate system," in *2019 IEEE 4th Colombian Conference on Automatic Control (CCAC)*, pp. 1–6, IEEE, 2019.

[18] A. Kassem, H. Haddad, and C. Albitar, "Commparison between different methods of control of ball and plate system with 6dof stewart platform," *IFAC-PapersOnLine*, vol. 48, no. 11, pp. 47–52, 2015.

[19] E. Fabregas, S. Dormido-Canto, and S. Dormido, "Virtual and remote laboratory with the ball and plate system," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9132–9137, 2017.

[20] H. Jafari, A. Rahimpour, M. Pourrahim, and F. Hashemzadeh, "Linear quadratic gaussian control for ball and plate system," in *2012 international conference on computer, control, education and management*, pp. 1–7, 2012.

[21] K. Zarzycki and M. Ławryńczuk, "Fast real-time model predictive control for a ball-on-plate process," *Sensors*, vol. 21, no. 12, p. 3959, 2021.

[22] M. Nokhbeh, D. Khashabi, and H. Talebi, "Modelling and control of ball-plate system," *Amirkabir University of Technology*, 2011.

[23] L. M. S. Santana, M. R. Maximo, and L. C. S. Góes, "Cob-2021-0636 physical modeling and parameters identification of the mg995 servomotor,"

[24] N. S. Nise, *Control systems engineering*. John Wiley & Sons, 2020.