DESIGN AND CONTROL OF A TWO-WHEEL SELF-BALANCING ROBOT

AZUHA SYUHADA BINTI NORMAN ZAIRI

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

DESIGN AND CONTROL OF A TWO-WHEEL SELF-BALANCING ROBOT

AZUHA SYUHADA BINTI NORMAN ZAIRI

This report is submitted in partial fulfilment of the requirements for the degree of Bachelor of Electronic Engineering with Honours

Faculty of Electronics and Computer Technology and Engineering
Universiti Teknikal Malaysia Melaka

2024

UNIVERSITI TEKNIKAL MALAYSIA MELAKA
FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KOMPUTER

BORANG PENGESAHAN STATUS LAPORAN
PROJEK SARJANA MUDA II

Tajuk Projek : Design and Control of a Two-wheel Self-balancing Robot
Sesi Pengajian : 2023/2024

Saya AZUHA SYUHADA BINTI NORMAN ZAIRI mengaku membenarkan laporan Projek Sarjana Muda ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Laporan adalah hakmilik Universiti Teknikal Malaysia Melaka.
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan laporan ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. Sila tandakan (✓):

☐	**SULIT***	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
☐	**TERHAD***	(Mengandungi maklumat terhad yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan.)
✓	**TIDAK TERHAD**	

Disahkan oleh:

(TANDATANGAN PENULIS)

Alamat Tetap: No. 38,Jln Wawasan 1, Tmn Wawasan, 43100 Hulu Langat, Selangor

Tarikh : 12 Januari 2024

(COP DAN TANDATANGAN PENYELIA)

DR AMAT AMIR BIN BASARI
Penyarah Kanan
Fakulti Teknologi Dan Kejuruteraan Elektronik Dan Komputer (FTKEK)
Universiti Teknikal Malaysia Melaka (UTeM)

Tarikh : 12 Januari 2024

DECLARATION

I declare that this report entitled “Design and Control of a Two-wheel Self-balancing Robot” is the result of my own work except for quotes as cited in the references.

Signature : .....

Author : Azuha Syuhada Binti Norman Zairi

Date : 12 Januari 2024

APPROVAL

I hereby declare that I have read this thesis and in my opinion this thesis is sufficient in terms of scope and quality for the award of Bachelor of Electronic Engineering with Honours.

Signature : .....

Supervisor Name : Dr. Amat Amir Bin Basari

Date : 12 Januari 2024

DEDICATION

I dedicate this thesis to my parents, Norman Zairi bin Suleiman and Norma Binti Daud, whose unwavering belief in my abilities and constant encouragement have been a tremendous source of inspiration throughout my journey. Their support has played an invaluable role in shaping both my academic and personal growth. I am profoundly grateful for their unwavering support and the profound impact they have had on my life. I would also like to express my heartfelt dedication to my friends whose unwavering support and understanding have been a constant source of strength. Their love, patience, and encouragement have been instrumental in keeping me motivated. This thesis stands as a testament to their unwavering belief in me, and I am forever grateful for their presence in my life. Furthermore, I would like to extend my dedication to Dr. Amat Amir Bin Basari, whose guidance and expertise have played a pivotal role in shaping my research and academic pursuits. Their dedication to my intellectual growth, insightful feedback, and unwavering support have made this journey possible. Lastly, I dedicate this thesis to all individuals who strive to make a positive impact in their respective fields. May this work contribute, albeit in a small way, to the broader body of knowledge and serve as inspiration for others to pursue their passions with unwavering determination and enthusiasm.

ABSTRACT

Two-wheel self-balancing robots typically have one degree of freedom (1-DOF), limiting their real-world usability, particularly in navigating obstacles and uneven surfaces. A study confirmed this limitation, noting sluggish disturbance rejection and significant oscillations in tilt response. This project aims to design and develop a prototype of 2-DOF two-wheel self-balancing robot using a PID controller. The proposed system focuses on the design and control of a 2-DOF self-balancing robot using a PID controller. The ESP32 microcontroller reads data from the MPU6050 sensor, and NEMA17 stepper motors drive the robot's movement. The ESP32 sends sensor data to Blynk, enabling remote PID control and system tuning without the need to modify the Arduino IDE code directly. Simultaneously, the ESP32 actively sends data to MATLAB for in-depth analysis of PID tuning. Results indicate overall robot performance with varied PID settings and disturbances. The closed-loop control system enhances the robot's real-time balance and trajectory control, effectively adapting to PID parameter changes and external disturbances. This project successfully implemented a PID controller and analysed the impact of PID tuning on the self-balancing robot's response. However, integrating a camera onto the robot could improve the robot’s usefulness for surveillance and monitoring purposes.

ABSTRAK

Robot penyeimbang dua roda biasanya mempunyai satu darjah kebebasan (1-DOF), terhad dalam mengatasi halangan dan permukaan yang tidak rata. Satu kajian disahkan dengan mencatatkan penolakan gangguan yang lambat dan ayunan yang signifikan dalam respons kemiringan. Projek ini bertujuan untuk merekabentuk dan membangunkan prototaip robot penyeimbang dua roda dengan 2-DOF menggunakan pengawal PID. Sistem tertumpu pada reka bentuk dan kawalan robot penyeimbang 2-DOF menggunakan pengawal PID. Mikropengawal ESP32 membaca data dari sensor MPU6050, dan motor langkah NEMA17 menggerakkan pergerakan robot. ESP32 menghantar data sensor ke Blynk, membolehkan kawalan PID dari jauh tanpa mengubah kod Arduino IDE secara langsung. ESP32 menghantar data ke MATLAB untuk analisis mendalam penalaan PID. Hasilnya menunjukkan prestasi keseluruhan robot dengan tetapan PID yang bervariasi dan gangguan. Sistem kawalan gelung tertutup meningkatkan keseimbangan dan kawalan trajektori robot, menyesuaikan diri dengan efektif kepada perubahan parameter PID dan gangguan luaran. Projek ini berjaya melaksanakan pengawal PID dan menganalisis impak penalaan PID terhadap respons robot. Walau bagaimanapun, penggabungan kamera pada robot boleh meningkatkan kegunaan robot untuk tujuan pengawasan dan pemantauan.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude and appreciation for those who has played an important role in assisting me through the completion of this Final Year Project. I would like to thank the supreme power the Almighty God who is obviously the one has always guided me to work on the right path of life. I believe without His grace, this project could not become a reality. Next to Him are my beloved parents, whom I greatly indebted for me brought up with love and encouragement to this stage of life and special thanks for sponsoring me to buy the materials and equipment in order to make this project a success. I am feeling oblige in taking the opportunity to sincerely gratitude to my supervisor, Dr. Amat Amir Bin Basari who helped me with the completion of this project. Last but not least, I would like to thank my precious lecturers from Faculty of Electronic Engineering and Computer Engineering and friends who have been always helping and encouraging me throughout the process of doing this project. I have no valuable words to express my thanks, but my heart is still full of the favours received from each of the person.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

PID	:	Proportional-Integral-Derivative
DOF	:	Degree of Freedom
DC	:	Direct Current
GPIO	:	General Purpose Input/Output
SDA	:	Serial Data Line
SCL	:	Serial Clock Line
Kp	:	Proportional Gain
Ki	:	Proportional Integral
Kd	:	Proportional Derivative
IMU	:	Inertial Measurement Unit
ESP	:	Expressive Systems' Platform
MPU	:	Microprocessor Unit
NEMA	:	National Electrical Manufacturers Association
IoT	:	Internet of Things
PWM	:	Pulse Width Modulation
GUI	:	Graphical User Interface
IDE	:	Integrated Development Environment

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Chapter Overview

This chapter provides a roadmap of the project, laying out the problem statement, objectives, and the scope which converges on designing a 2-DOF self-balancing robot. Additionally, this chapter also includes the outline of each chapter in this thesis.

1.2 Project Background

A two-wheeled self-balancing robot is an important and notable type of mobile robot. It refers to a robot's capability to maintain its balance on two wheels without toppling over. Unlike many other control systems, the inverted pendulum system inherent in these robots is naturally unstable. As a result, it requires control mechanisms to achieve stability in this precarious state. Essentially, a two-wheeled

balancing robot functions as an inverted pendulum system that remains upright on its two wheels.

Compared to other mobile robots, self-balancing robots offer several advantages, including their compact size, versatility, and affordability. These qualities contribute to their popularity in many events and settings. As a distinctive example of an inverted pendulum, the two-wheeled self-balancing robot exhibits characteristics of instability, complex and nonlinear dynamics, and multivariable behavior. The research in the field of two-wheeled balancing robots has gained significant momentum in recent years, primarily driven by the introduction of the Segway, which revolutionized personal transportation.

This project focuses on the modelling of the robot, the design of a Proportional-Integral-Derivative (PID) controller, and the implementation of this controller on the two-wheeled robot. The chosen controller for this project is the PID controller due to its practicality and ease of implementation. It only requires adjustment of three parameters, which can be determined using various techniques. Previous studies have demonstrated that properly tuned PID controllers yield favorable outcomes in terms of response time and accuracy. These parameters, namely Kp, Ki, and Kd, play a crucial role in achieving optimal performance.

1.3 Problem Statement

The two-wheel self-balancing robot is inherently unstable and without external control it would roll around the wheels rotation axis and eventually fall[1]. Most of the produced self-balancing robots have only one degree of freedom which also provides some difficulties for users in the real-world. A PID (Proportional-Integral-

Derivative) control algorithm needs to be implemented to adjust the motor's speed based on the sensor data.

One degree of freedom limits the robot's ability to perform complex tasks that require multi-axis motion or manipulation. It may be challenging to extend the capabilities of the robot beyond basic balance control to more sophisticated behaviors. [2]. In practical situations, the environment is often intricate and constantly changing. The ability of a 1-DOF self-balancing robot to navigate obstacles or uneven surfaces can be hindered due to its limited degrees of freedom. This limitation was confirmed in a study, where the disturbance rejection capability of the control system was sluggish, and the tilt response had significant oscillations [2]. As a result, it lacks the necessary flexibility for successful adaptation and efficient motion planning in such complex and dynamic environments.

Thus, this project aims to develop a two-wheel self-balancing robot with 2-DOF that uses PID control algorithms in order to provide a stable and efficient method for controlling the robot's balance in real-time.

1.4 Objectives

This research aims to create a two-wheel robot using the inverted pendulum concept and a PID controller. The specific objectives formulated to achieve this aim are as follows:

i. To design a two-wheel self-balancing robot with two degrees of freedom (2-DOF).
ii. To develop a prototype for the two-wheel self-balancing robot with a PID controller.

iii. To analyze the performance of the developed self-balancing robot through PID tuning for improved balance control.

1.5 Scope of Project

The scope of this thesis is focused on the design and control of a two-degree-of-freedom (2-DOF) self-balancing robot using a PID controller. NodeMCU ESP32 is the microcontroller used to read the sensor data from MPU6050 sensor. The MPU 6050 sensor is used to accurately measure motion and orientation, including acceleration and rotational movement. On the wheels of the robot, there are two stepper motors NEMA 17, that serve as the driving force for the robot's movement. Its precise control allows for accurate rotation of the wheels, enabling the robot to maintain balance. NodeMCU ESP32 Wi-Fi module is used for internet connectivity, PID tuning, and system tuning can be remotely controlled by using Blynk application.

1.6 Chapter Outline

The two-wheel self-balancing robot that uses IMU sensor was described as an intelligent way to reduce human efforts in their daily activities. All the details about this project were defined in every chapter as shown below.

CHAPTER 1: This chapter will give a brief introduction to the project including the project background, problem statement, objectives, scope of project and the chapter outline for the whole project are clearly explained in this chapter.

CHAPTER 2: This chapter will discuss about the articles or sources that are related to the project. This project is known by the sources and research that has been done before. Literature review provides a background of this project and also gives and

direction in this research. The details of the project's background are briefly explained in this section.

CHAPTER 3: This chapter focuses on the research methodology employed in this project, providing a comprehensive outline of the specific approach utilized. This chapter presents a selection of materials essential for hardware development. Additionally, it delves into the theory and practical application of the PID controller within the context of this project.

CHAPTER 4: This chapter deals with the results and discussion. It will highlight the results obtained in design and development of the hardware. Besides, it also discusses the tuning method of PID in self-balancing robot. All of the obtained results are briefly explained in this section.

CHAPTER 5: The final chapter will explain the conclusion and future recommendation of the project which also includes the project achievement, project problem and limitation, and future recommendation in order to improvise the project.

1.7 Chapter Summary

In conclusion, the following outline of this thesis will cover several different parts and aspects of this project. Then, Chapter 2, will cover the literature review or background studies of related past articles or journal. Next, Chapter 3 will cover the methodology of this project. All the methods and components used in this project will be discussed. Furthermore, in Chapter 4 will be discussing the results of this project including the analysis of the PID tuning method. Finally, the conclusion of this project including the future recommendations of this project will be discussed in the last chapter which is Chapter 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Chapter Overview

This chapter reviews relevant literature to enhance understanding of the project, focusing on presenting theoretical background and summarizing key findings and contributions from previous research.

2.2 Fundamental of Inverted Pendulum

The inverted pendulum is a classic automation problem that has numerous theoretical approaches as well as a multitude of practical applications [3]. A typical design for a robot that performs the inverted pendulum task involves a tower-like structure supported by two wheels. The robot is capable of independently controlling its motors to maintain an upright position while moving in response to user commands.

Stabilizing an inverted pendulum is a fundamental challenge in control systems, involving key components like a DC motor, a cart, a pendulum, and a driving mechanism for the cart. According to the system dynamics the system has two degree of freedom the one is for cart movement and the other one is for Pendulums rotational motion [4]. Figure 2.1 shows the inverted pendulum parametric presentation.

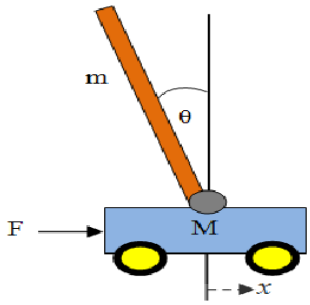


Figure 2.1: Inverted pendulum parametric presentation

The list of parameters of the inverted pendulum system is depicted in Table 2.1 below.

Table 2.1: Parameters of the inverted pendulum system

Parameter	Description	Unit
m	Mass of the pendulum	kg
M	Mass of the cart	kg
F	Force applied to the cart	kg.m/s²
b	Friction of the cart co-efficient	Ns/m
l	Length of the pendulum	m
I	Moment of inertia	kg-m²
g	Gravitational Force	9.8 m/s²
x	Cart position co-ordinate	-
θ	Vertical pendulum angle	In degree, °

In order to obtain the system dynamics of the inverted pendulum, several assumptions are typically made. These assumptions help in enabling the analysis and design of control strategies for the inverted pendulum system.

i. Initial equilibrium state: The system starts from a balanced state with zero initial conditions, simplifying analysis to focus on deviations from equilibrium.

ii. Small angular deviation: The pendulum is modeled with small angular displacements for linearity, allowing the use of linear control techniques as its dynamics remain linear within this range.

iii. Step input: The system is tested with a step input, represented as a sudden change in the pendulum angle (θ), simplifying analysis by examining the response to a consistent, abrupt change in input.

2.3 Types of controllers to balance two-wheel robot

Several researchers have developed various types of controllers, each with its unique design approach and associated pros and cons.

The authors in [5] states that balancing of the robot can be done with help of feedback and a correction element. The feedback element in the system is represented by the MPU6050 board, which communicates the current orientation of the robot to the Arduino. The objective of the study is to achieve the capability of the robot to maintain an upright position and balance effectively. As the result of the study, it has been concluded that PID algorithms can be used to stabilize an unsteady robot. To achieve balance in the robot, two control strategies are utilized which are MPU6050 calibration and PID tuning. The calibration process involves obtaining six sets of offset values from the accelerometer and gyroscope by aligning the MPU6050 with the ground plane. PID tuning is then performed using a trial-and-error method to attain a stationary position for the robot.

Meanwhile, the study in [6] presents two control methods for a WIP-based self-balancing robot: PID for vertical angle stabilization and LQR for motion trajectory, tested in MATLAB Simulink. They employed two techniques: PID control for vertical angle stabilisation and LQR control with state observer to follow a desired motion trajectory and stabilise the pendulum's vertical angle. In the project, just PID control for vertical angle stabilization is used where LQR control will be incorporated in the future. The final version of the WIP self-balancing robot was tested on carpet and stoneware surfaces with different friction levels. Results indicate that the robot, using only PID control for vertical angle stabilization, effectively maintains balance on these surfaces with a maximum error of 4°.

In addition, linear controllers have gained popularity among researchers involved in the design of similar balancing robots, such as JOE: A mobile, inverted pendulum. The most widely used control systems are the Pole Placement controller and the Linear Quadratic Regulators (LQR), both of which are based on linear state space models. These controllers have been extensively implemented and studied due to their effectiveness in achieving desired balancing and control outcomes for such robotic systems. While the authors in [6] mentioned that PID controller based on output feedback may not achieve satisfactory control results for a high-order and multi-variable system.

In other research paper, the author [7] mentioned that fuzzy logic is able to enhance the robot’s ability, particularly when subjected to external forces. In the study, the PD and PID controllers are designed using the pole placement method, and their parameters are optimized using the Genetic algorithm. The PD controller exhibits vibrations and instability, leading to the robot's fall. By adding a pole and transforming

it into a PID controller, stability improves but external forces still cause the robot to fall. To enhance PID performance, the parameters are fine-tuned using Fuzzy logic. As a result, the Fuzzy-PID controller reduces vibrations, improves stability, and mitigates the impact of external forces.

2.4 Two-wheel self-balancing robot using PID controller

One of the fundamental challenges in designing two-wheel self-balancing robots is achieving stable balance and control. These robots typically employ a combination of sensors, actuators, and control algorithms to sense their orientation and make adjustments to keep themselves balanced. Based on the sensor readings, a control algorithm, Proportional-Integral-Derivative (PID) controller, then calculates the appropriate control signals to drive the motors and maintain balance.

PID control is known for its past success, simplicity in the implementation and broad availability [8]. In recent studies, there has been a notable emphasis on augmenting the capabilities of self-balancing robots through the integration of additional degrees of freedom. The inclusion of a 2-DOF system enhances the robot's flexibility, maneuverability, and ability to navigate around obstacles.

The author discusses the utilization of a two-stage proportional-integral-derivative (PID) controller in conjunction with a microcontroller, position sensor, and DC motor for the hardware system design [9]. The authors also present the symbolic representation of the state space dynamic model and highlight the use of MATLAB simulation, fuzzy algorithms, and sophisticated mathematical modeling strategies in previous studies. Furthermore, this paper offers valuable observations on the performance evaluation of the self-balancing robot, demonstrating the effectiveness of a 2-DOF PID controller in reducing settling time and enhancing stability. The

applications of this research encompass educational experiments, robotics and control courses, and potential integration with Bluetooth or wireless modules for improved movement control and balance maintenance, indicating promising future scope in diverse fields such as product containment systems and flying machines.

Next, the authors utilized MATLAB and Arduino IDE for the design and control of the robot, while the main electrical components included Arduino Uno microcontroller, IMU MPU-6050 sensor, 10 in-series 1.2V AA NiMH batteries, Pololu Dual G2 High-Power motor driver, and 2 Pololu 30:1 Metal Gearmotor motors [10]. They applied Lagrange equations for mathematical modeling and used two cascading PID loops for controlling speed and tilt angle. The experimental findings and the robot's actual performance are also included in the article. The robot demonstrated the ability to transport objects of different sizes and weights while maintaining balance and navigating tight spaces. It's shown to be versatile, with potential applications in surveillance, rescue, hazardous environment cleaning, military, and transportation.

Furthermore, this paper focuses on the design, construction, and control of a two-wheeled balancing robot using Linear-Quadratic Regulator (LQR) controller [5]. The robot consists of an MPU 6050, a microprocessor, a frame, two wheels driven by DC gear motors, and a battery. The MPU-6050 sensor detects the angle of tilt or inclination along the X, Y, and Z axes as well as the rotational velocity along the X, Y, and Z axes by combining a 3-axis accelerometer and a 3-axis gyroscope with Micro Electromechanical System (MEMS). Data from the accelerometer and gyroscope are combined using a complementary filter. The LQR controller is employed for stabilization and balance. Experimental results show the robot's ability to maintain

balance, illustrating the principle used in transportation like the Segway PT and garnering interest in control engineering research.

Moreover, the next paper discusses the development of a Two Wheeled Robot (TWR) using a single stepper driver and PID control loop [11]. The TWR is designed [12]using an Arduino Nano microcontroller, Inertial Mass Unit (IMU) sensor, and stepper motors with stepper driver. The PID control loop is implemented to balance the robot's vertical position, and the PID coefficients are tuned for stability and responsiveness, with Kp adjusted for oscillations, Ki for reduction, and Kd for quick response. A complementary filter is used for smoother sensor measurements, and potential enhancements include a Kalman filter, PID with fuzzy logic, LQR controller, and wireless communication. The TWR is applicable in robotics, automation, and education, where precise control and balance are essential.

Besides, the author [11] presents a comprehensive exploration of the design and analysis of a TWABR system, utilizing components such as the MPU6050, DRV8825 motor driver, NEMA17 Bipolar stepper motor, and an ESP32 Microcontroller powered by an 11.1V Lithium Polymer battery. MATLAB is used for designing and simulating classic PID and optimal LQR controllers, focusing on control performance and stabilization. The TWABR's dynamic model is based on nonlinear differential equations and Lagrange dynamics, linearized for analysis in state space and frequency domain. The study underscores the LQR controller's superior performance and suggests exploring speed control and variable tilt angles. This work lays a foundation for low-cost TWABR systems in control theory education and further control strategy research.

Moreover, the author investigates the design and execution of a self-balancing robot with the use of an MPU6050 sensor, a 12V DC motor with encoders, and SolidWorks for design and Arduino Uno for control [12]. A key aspect was creating a state-space model for the DC motor, clarifying the relationship between input voltage, control torque, and deriving state space equations. Experimental results highlighted successful hardware-software integration, particularly in balance control with an inverted pendulum model. Real-time testing employed advanced PID and LQR controllers, with the PID effectively managing the robot's tilt. Future enhancements include improving trajectory tracking, stabilizing on uneven surfaces, and adding manual remote control via Bluetooth. This system outperforms current benchmarks in the field.

Other than that, the author discusses the design and implementation of a two-wheel self-balancing inspection robot, leveraging advanced technologies such as Silan RPLIDAR A2 LiDAR and a particle filter [13]. The research's core is developing a self-balancing control algorithm and a kinematic model, underpinned by a double-closed-loop PID strategy. SolidWorks and finite element statics are used for the robot's design, which features autonomous decision-making, environmental sensing, and control execution using the Gmapping SLAM algorithm for localization and map generation The study highlights the significance of particle number in Gmapping for map accuracy, supported by high-accuracy tests. Both in real-world experiments and in simulation settings like Gazebo, the self-balancing algorithm's robustness and anti-jamming skills were thoroughly tested. The result of these efforts is an inspection robot based on laser SLAM that proves its capacity to perform inspection tasks effectively in test conditions, verifying the efficacy and rationale of the system.

Furthermore, the author in [14] explores the design and implementation of a two-wheeled self-balancing robot (TWSBR), integrating key components such as the MPU6050 sensor, DC motors, a 12V battery, and a chassis made from plastic baseboard and metallic racks. The study validates the PD control theory using MATLAB's SIMULINK, demonstrating the robot's ability to maintain balance over time and respond effectively to disturbances, reaching saturation. Robotic toolbox simulations further confirm the working principle. Motion equations are deduced from this modeling, and a stability analysis is conducted based on the system's poles. The implementation of PD-PI navigational control and the use of a Kalman filter algorithm underscore the robot's stability. The robot's ability to avoid obstacles is maximized using ultrasonic waves for detection, and its communication with IoT devices is facilitated via Bluetooth technology.

Next, the author [15], discusses the construction and operational capabilities of a self-balancing robot, a prime example of a cyber-physical system, utilizing components such as a NEMA17 stepper motor, A4988 stepper motor driver, HC-06 Bluetooth module, LM2596 current limiter, MPU6050 sensor, and an Arduino Uno, all powered by a 5100 mAh lithium polymer battery. The robot successfully achieves balance and recovers from external disturbances, although it faces limitations in sustaining balance against larger forces due to its small form factor. The robot's control algorithm, intricately designed in the Arduino IDE and visualized through UML diagrams, employs a PID controller with finely tuned values (Kp 1150.0, Kd 157.5, Ki 0.12) to maintain its orientation, a process rigorously monitored through sensor data. Additionally, the robot's circuitry using Fritzing software, facilitates controlled movement in four directions which are forward, backward, left, and right maneuvered through a bespoke Android app via Bluetooth.

Last but not least, the author presented the development and control of three-level self-balancing robots highlighting the integration of components such as Arduino Nano, DC motors, L298N motor drivers, MPU6050 Sensor, 16x2 LCD display, push buttons, and lithium rechargeable batteries [16]. The robots, each featuring different chassis materials, employ a combination of PID control and Kalman filter for stability and movement control. The system modeling was achieved using the Lagrange equation, with the model subsequently converted into a transfer function for effective PID implementation. The PID controller was meticulously designed and tuned following the Ziegler Nichols method, while the Kalman filter was developed and integrated to enhance system performance. Both PID control and Kalman filter were rigorously tested through simulations and actual robot implementation, demonstrating the robots' ability to maintain balance for extended periods (up to one hour) and robustly reject disturbances. The paper also encompasses extensive simulations, hardware design, and experimental examinations to gather and analyze research data, ultimately resulting in a well-evaluated, stable, and efficient control system for self-balancing robots.

As a whole, the literature reviewed for this thesis highlights a key fact about robotics and control systems which every project is a special mosaic of invention, with each publication or contributing author offering their own technique and viewpoint. This reflects the PID controller's fundamental role in robotics and automation, adaptable to a wide range of applications, from basic self-balancing robots to complex, multi-level control systems.

Table 2.2: Comparative Analysis of Reviewed and Proposed of Two-wheel Self-balancing Robot using PID Controller

Title	Software / Equipment used	Results	Method Used	Conclusion
A Smart Approach to control a two-Wheeled Self-Balancing Robot using a PID Controller with Two Degree of Freedom (2022)	• Arduino Uno • L298N motor driver • DC motor • GY521 Sensor	• The settling time decreased to 0.0861 seconds. • The 2-DOF controller was found to be more effective than the PID controller.	• Inverted pendulum control theory. • Conventional 2-DOF proportional-integral-derivative (PID) controllers. • Readings of acceleration, distance traveled, and inclination from sensors.	• Presents a two-stage PID controller for controlling a two-wheeled self-balancing robot. • The PID controller is shown to be superior to traditional PID controllers.
Design and Control of Two Wheeled Self Balancing Robot (TWSBR) (2022)	• Arduino Uno • MPU 6050 sensor • 10 in-series 1.2V AA NiMH batteries • Pololu Dual G2 High-Power motor driver • 2 Pololu 30:1 Metal Gearmotor motors	• Design, model, and control of Two-wheeled Self-balancing Robot. • PID gains obtained in simulation were used in practical tests. • Desirable response was achieved in the practical test using one manipulated input (voltage).	• Two PID control loops cascade for controlling outputs. • Tuning PID gains using MATLAB. • Maintaining balance while moving and carrying objects. • Traveling at different desired speeds specified by the user.	• Two PID control loops cascade to control robot speed and tilt angle. • Robots can maintain balance while moving and carrying objects. • Robot can travel at different desired speeds specified by the user.

Controlling of Two Wheeled Self Balancing Robot using PID (2018)	• DC gear motors • Battery • MPU 6050 Sensor • Arduino Uno	• Use of Linear-Quadratic Regulator (LQR) controller for balancing. • Measurement of robot angle using MPU 6050 and filtering of data. • Experimental results show the robot can maintain balance.	• Linear Quadratic Regulator (LQR) controller • State feedback control system • Complementary filter • Encoder feedback • MPU 6050 for angle measurement	• Self-balancing robot was successfully constructed. • The control method provided stable and reliable balance condition for motion control. • Robots can balance upright positions using only two wheels.
Development of Two Wheeled Robot (TWR) by Single Stepper Driver using PID controller (2021)	• Arduino Nano • DRV 8825 stepper motor driver • 7805 IC Voltage regulator • Li-Po Battery • MPU 6050 • Stepper motor • CAD software (Autodesk Inventor Professional) • MATLAB software	• Discusses the combination of accelerometer and gyroscope data. • Use of a complementary filter to sift noise and drift. • Calculation of angles and their addition to former inclination. • Rotation direction of motor 2 based on voltage applied. • Provides a sequence of voltage applied on leads.	• Trial and error method used to tune the PID parameters.	• Two-wheeled robot was successfully developed. • Capable of balancing on its two wheels using PID controller. • Single stepper driver used to drive both stepper motors simultaneously. • Tuned PID coefficients (Kp, Ki, Kd) were found to improve the robot's performance.

Modeling and Control of a Two Wheeled Auto Balancing Robot: A didactic platform for control engineering education (2019)	• MPU6050 • DRV8825 motor driver • ESP32 Microcontroller • NEMA17 stepper motor • 11.1V Lithium Polymer battery (Li-Po) • MATLAB software	• Design and analysis of a Two Wheeled Automatic Balancing Robot (TWABR). • Analysis of TWABR's dynamic model through nonlinear differential equations. • Design and simulation of classic PID controller and optimal LQR controller.	• Design and simulation of classic PID controller • Design and simulation of optimal LQR controller • Experimental comparison of the performance of the implemented controllers. • Two controllers implemented: PID and LQR.	• Design and construction of a low-cost TWABR system for control theory education. • TWABR system modeled using Lagrange dynamics equations. • LQR controller showed better dynamic and static response.
Research on Self-balancing Two Wheels Mobile Robot Control System Analysis (2022)	• SolidWorks software • 12 V DC motors with encoders • MPU6050 Sensor • Arduino Uno	• Experimental results based on hardware implementation and overall system implementation were presented. • Outcomes of research were mentioned based on discussions on several analyses.	• State-space model of the DC motor • Relationship between input voltage and control torque • Rearrangement of equations to obtain the state space equation. • Torque applied on the chassis from the motor and linear transformation.	• Successfully achieved balance control for self-balancing robot. • PID controller used for controlling single axis robot tilting angle. • Recommended further extension for stabilizing robot on sloped and rough surfaces.

Research on Two-Round Self-Balancing Robot SLAM Based on the Gmapping Algorithm (2023)	• Silan RPLIDAR A2 LiDAR • Particle Filter	• Design and analysis of a two-wheel self-balancing inspection robot. • Establishment of a kinematics model and design of a self-balancing control algorithm. • Use of the Gmapping SLAM algorithm for robot localization and map construction. • Importance of particle number selection for improving map accuracy.	• SolidWorks for designing the mechanical structure of the robot. • Multi-closed-loop PID controller for designing the self-balancing control algorithm. • 2D LiDAR-based Gmapping algorithm for robot localization and map construction. • Self-balancing test and anti-jamming test for verifying the algorithm's performance. • Actual test results showing high map accuracy.	• Designs and implements a laser SLAM-based inspection robot. • The double-closed-loop PID algorithm is used as the self-balancing control algorithm. • The system's rationality is verified through simulation and actual testing. • The robot successfully completes the inspection task in the test environment.
Robust Navigational Control of a Two-Wheeled Self-Balancing Robot in a Sensed Environment (2019)	• MPU6050 • DC Motors • 12V Battery • Plastic Baseboard and metallic racks • Transparent plastic board	• PD Control theory is verified using SIMULINK in MATLAB. • The robot can balance itself with two wheels over time. • The robot responds to disturbances and reaches saturation.	• Mathematical modeling • Analysis of the relationship between forces and motors' voltage. • Motion equations deduced from modeling. • Stability analysis based on the poles of the system.	• PD-PI navigational control successfully implemented. • Maximum obstacle avoidance achieved. • Integration with IoT devices established through Bluetooth technology.

Simple Two-wheel Self-Balancing Robot Implementation (2023)	• NEMA17 stepper motor • A4988 stepper motor driver • HC-06 Bluetooth module • LM2596 Current limiter • MPU6050 Sensor • Arduino Uno • 5100 mAh Li-Po Battery • Arduino IDE	• The robot was fully constructed and powered up successfully. • The robot was able to balance and recover from tilts caused by external forces and able to move in four different directions using the mobile app. • The small form factor of the robot caused it to eventually fall off when a large amount of force was applied.	• The structure design of the robot is based on the concept of a reverse pendulum. • The robot's control algorithm uses sensor data to monitor its orientation. • The control loop of the robot uses a PID (proportional-integral-derivative) controller. • UML diagrams were created to model the robot.	• The reverse pendulum concept was used. • Robots can balance on two wheels and recover from external forces. • Control information sent through an android app via Bluetooth. • Robot can move in four basic directions: forward, backward, left, and right.
Using a Combination of PID Control and Kalman Filter to Design of IoT-based Telepresence Self-balancing Robots during COVID-19 Pandemic (2021)	• Arduino Nano • DC Motors • L298N motor drivers • MPU6050 Sensor • 16x2 LCD display • Push Button • Lithium rechargeable battery	• PID control and Kalman filter. • The robots used two DC motor drives and had different chassis materials. • PID control was tested using simulations and actual robot implementation.	• Conversion of the model to a transfer function for PID implementation • PID controller design and Ziegler Nichols tuning • Design of the Kalman filter • Simulation and hardware design	• A robust and stable disturbance rejection control was successfully designed. • The robot maintain balance for approximately one hour before re-balancing.

Proposed System	• **MPU6050 Sensor** • **ESP32 Microcontroller** • **A4988 Stepper Motor Driver** • **NEMA17 Stepper Motor** • **18650 Lithium-ion Battery (12V)**	• **2-DOF self-balancing robot is produced.** • **Robot's performance varied when PID parameters varied.** • **The PID tuning for the robot can be applied to all types of surfaces.** • **Robots are subject to external disturbance while balancing themselves.** • **Analysis of PID tuning parameters is applied through MATLAB from data obtained in serial (ESP32)**	• **Trial and error method for PID tuning.** • **MATLAB software to analyze the real-time data from ESP32.** • **Schematic circuits are built in Fritzing software.** • **Blynk application is used to remotely tune the PID parameters.** • **Automatic Calibration through Arduino IDE library.**	• **2-DOF self-balancing robot obtained self-control in balancing itself.** • **Performance for each PID tuning parameter is analyzed through pitch angle vs time plot in MATLAB.** • **A robust and stable disturbance rejection control is successfully designed.**

2.5 Internet of Things (IoT)

According to Bansal et al. (2019)[17], the globe has a tremendous demand for IoT technologies due to the rapid advancement of technology. The current era witnesses the remarkable capabilities of the Internet of Things (IoT) in bridging the gap between the virtual realm and tangible reality. This transformative technology encompasses a network infrastructure that seamlessly connects various electronic sensors, devices, and objects, enabling them to establish meaningful correlations between their internal states and the external environment. Therefore, IoT has a lot to serve in various aspects of life and technology [18]. It holds immense potential in terms of technological advancements and its ability to significantly benefit humanity.

2.6 Gaps and Challenges

When delving into the extensive amount of literature on PID controllers and robotic systems, it is essential to identify and address the gaps and challenges that continue to affect current research. This subtopic aims to discover the gaps and the complexity in previous research, illuminating unexplored areas and possible roadblocks that may serve as inspiration for next innovations and advances in the field. Table 2.3 below shows the gaps and challenges from previous research and the proposed system.

Table 2.3: Gaps and Challenges of Past Research and Proposed System

Gaps and Challenges	Past Research	Proposed System
Design	• Most of the designs predominantly focus on 1-DOF systems. • Offer limited scope in terms of movement and functionality.	• Advances to a 2-DOF design, expanding the capabilities and potential application of the system. • Open new avenues for exploration and innovation in robotic system design.

PID parameters tuning	• PID tuning involves adjustments within the coding environment and the use of MATLAB Simulink. • Limits the flexibility and real-time adaptability of the system.	• Utilize Blynk application to tune PID parameters. • Allow for more dynamic and on-the-fly adjustments. • Offer a more user-friendly and accessible way to optimize system performance in real-world scenarios.
Wheel speed and acceleration	• LQR is tuned to improved handling of stepper motors. • LQR requires a precise mathematical model of the system, including its dynamics. • LQR controllers involve solving matrix equations, which are computationally intensive.	• Optimize the performance of the stepper motor using Blynk application. • Allow for precise adjustments in real-time, directly impacting and enhancing the robot's wheel speed and acceleration. • Allow for greater flexibility and customization in control system design.
Blynk for control	• Utilized Android apps and Bluetooth devices for control and communication purposes. • Lacked deeper integration with the broader Internet of Things (IoT) ecosystem	• Blynk offers more sophisticated monitoring features including real-time data visualization, which can be critical for understanding the system's performance and behavior over time. • Allow for remote interactions with the robot from virtually any location.

Based on Table 2.3, it compares the approaches and advancements of past research with the proposed system in four key areas: design, PID parameter tuning, wheel speed

and acceleration, and control using Blynk. As for the design part, past research mainly focused on 1-DOF systems, limiting movement and functionality. The proposed system advances to a 2-DOF design, offering expanded capabilities and new possibilities in robotic system design. Then, traditional methods used coding environments or MATLAB Simulink, restricting real-time adaptability [19]. The proposed system uses the Blynk app for dynamic and user-friendly PID tuning, enhancing real-world performance.

Next, previous studies relied on LQR tuning, requiring complex models and high computational power [20]. The proposed system optimizes stepper motor performance using Blynk, allowing for precise, real-time adjustments and increased flexibility. Lastly, earlier approaches used Android apps and Bluetooth for basic control, lacking IoT integration. The proposed system employs Blynk for sophisticated monitoring, real-time data visualization, and remote interaction, significantly enhancing control and monitoring capabilities.

2.7 Chapter Summary

To sum up, multiple existing techniques that are related to the proposed technique are reviewed within this chapter. The reviews are done based on their system design and method as well as the strengths and weaknesses.

CHAPTER 3

METHODOLOGY

3.1 Chapter Overview

The methodology is composed of several related ideas, such as method and algorithm. It outlines the methods that have been used and, in general, defines how the research has been carried out. This chapter describes the project's implementation, including how the two-wheel self-balancing robot operates theoretically with PID controller and how to build the system from the ground up, starting with the software and ending with the hardware.

3.2 Flowcharts

The methodology employed in this project comprises two main components: mechanical design and software algorithm. In this chapter, the approach utilized to accomplish the desired objectives will be outlined. The project will be executed based

on a well-defined flowchart that delineates all the essential activities that need to be undertaken.

Figure 3.1 depicts the project's flowchart, which commences with a comprehensive literature review to gather information on topics relevant to the two-wheeled balancing robot. After thoroughly examining the available resources, the subsequent step involves modeling the inverted pendulum, as it forms the foundational concept for this project.

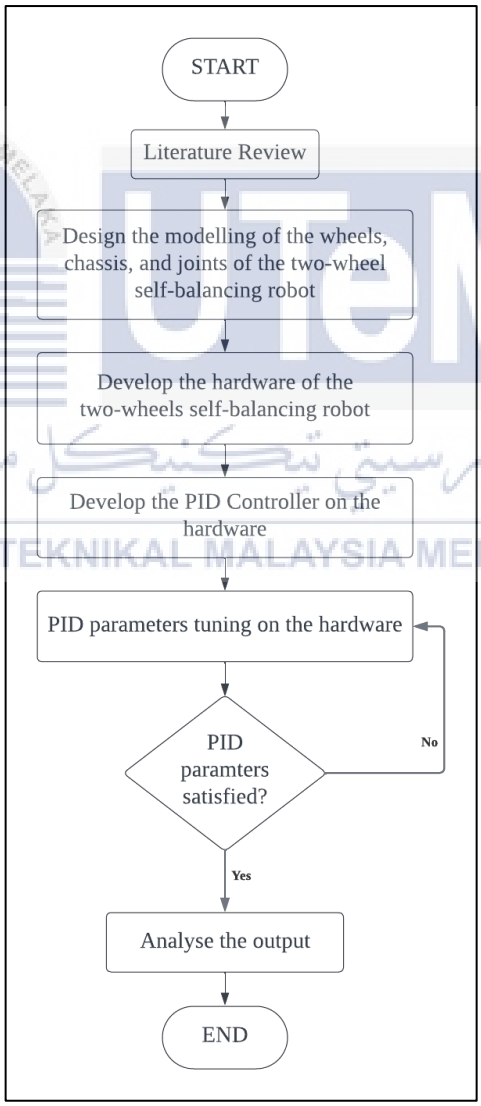


Figure 3.1: Flowchart of the project

Figure 3.2 shows the flowchart for the PID tuning process. It begins by setting the initial values for the proportional, integral, and derivative gains (Kp, Ki, and Kd). The control algorithm is then implemented using these gains to regulate the robot's balance. Following that, the gains are adjusted based on the observed system response. This iterative process continues until the system achieves stability. Once stability is achieved, the system's performance is analyzed.

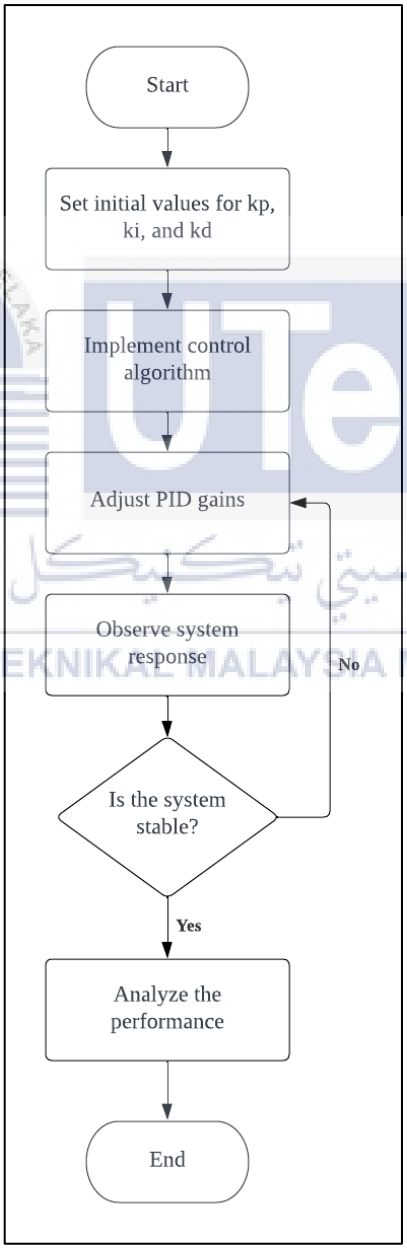


Figure 3.2: Flowchart for PID tuning

The project aims to analyze the variables of rise time, settling time and overshoot, steady-state error reduction, and stability by systematically adjusting the proportional constant (Kp), integral constant (Ki), and derivative constant (Kd). Through experimental observations of the robot's behavior and data analysis, the relationship between these variables and the robot's performance can be examined. The response speed will be assessed by measuring the robot's reaction time to different reference inputs, while steady-state error reduction will be analyzed by observing the error signal for varying Ki values. Additionally, the stability of the robot's motion will be evaluated by gradually increasing Kd and monitoring any signs of instability. By conducting these analyses, valuable insights can be gained to optimize the robot's control system and enhance its overall performance.

3.3 Block Diagram

The effective integration of mechanical and software components relies heavily on the design of the hardware system. The circuitry for the self-balancing robot shown in Figure 3.3 encompasses key elements, including the MPU6050 sensor—an inertial measurement unit (IMU), the ESP32 microcontroller, stepper motor and the stepper motor driver. Additionally, MATLAB is used for analysis of data for PID tuning.

NodeMCU ESP32 will act as the brain of the system which will connect the MPU6050, MATLAB, stepper motor driver and Blynk application as the IoT platform. The MPU6050 sensor is employed to gauge the robot's acceleration and angular rate, converting the analog output into digital data. The raw input from the IMU sensor is subjected to further processing to determine the inclination angle of the robot. This angle is then fed into the PID controller algorithm, which computes the optimal speed for the stepper motor to uphold the robot's balance. Besides, NodeMCU ESP32 will

connect to WIFI and send the data to IoT device server which is Blynk Apps to remotely tune the PID parameters. MATLAB is then used to analyze the response of the robot at different PID parameter values for analysis.

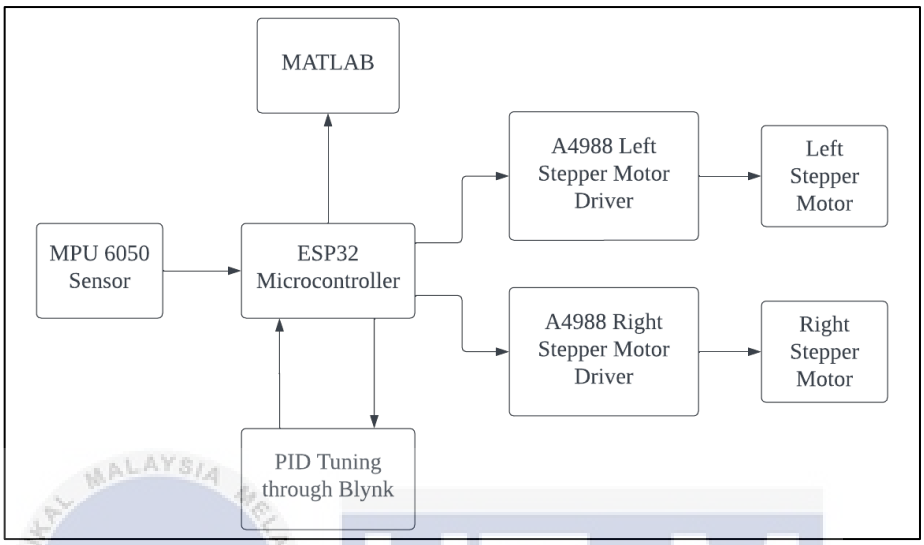


Figure 3.3: Block Diagram of the project

The closed loop control system of the self-balancing robot depicted in Figure 3.4 consists of a PID controller within an ESP32 Microcontroller that processes the tilt angle error signal from a MPU 6050 sensor to generate control commands for a stepper motor. The PID controller adjusts the motor's movements using proportional, integral, and derivative gains to maintain the robot's balance. The stepper motor driver takes the control commands and translates them into electrical signals that drive the stepper motor. The motor responds to the electrical signals by rotating in a precise manner, which adjusts the position of the robot to maintain balance.

Then, a disturbance block, representing external physical forces like pushes or pulls, is shown impacting the system's output, simulating real-world interactions. The closed-loop nature of the system means that any effect of the disturbance on the robot's

balance is continuously fed back into the control system, allowing it to dynamically adjust the motor commands to counteract the disturbance and stabilize the robot.

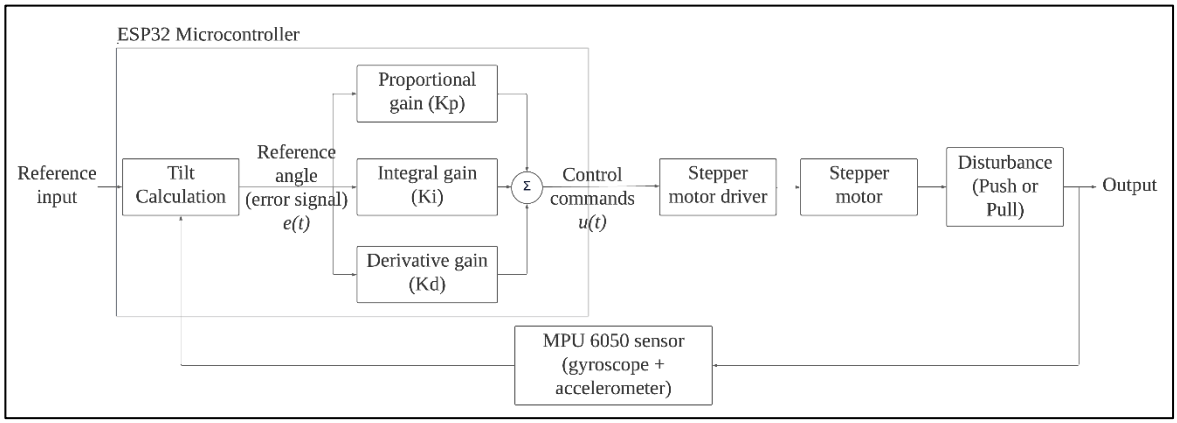


Figure 3.4: Closed loop control system of the self-balancing robot

Based on the closed loop control system shown in Figure 3.4, the PID controller can be mathematically written as:

$$u(t) = K_P e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t) \qquad (1)$$

The controller's response, *u(t)*, depends on three constants: Kp, Ki, and Kd, which represent the proportional, integral, and derivative controllers, respectively. The error signal, *e(t)*, reflects the difference between the y-axis and the actual position of the robot. Kp adjusts the response speed by multiplying with the error, leading to changes in the robot's balance at different set-points. Ki minimizes steady-state error and enhances the robot's motion smoothness, but even a small adjustment can have a significant impact due to its integrating nature. Kd influences the robot's reaction time and should be carefully tuned to prevent instability when excessively increased.

3.4 Main Components

This section introduces the components used in the project, outlining their roles and functions within the system. Each component is vital for the design's success and the system's effective operation.

By carefully selecting and integrating these components, the aim is to create a robust and efficient solution. The model of the self-balancing robot is shown in Figure 3.5. The components have been chosen based on specific criteria, including performance specifications, compatibility, availability, and cost-effectiveness. This ensures that they meet the project requirements and contribute to the successful implementation of the intended functionalities.

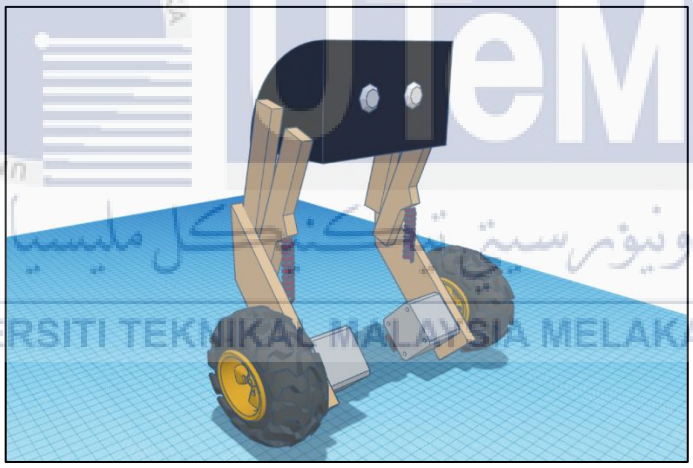


Figure 3.5: Model of the self-balancing robot

The 2-DOF self-balancing robot model is designed with the aid of TinkerCAD software as a virtual representation of a robot intended to keep its balance while moving. TinkerCAD is a computer-aided design (CAD) software that allows users to create 3D models using a simple drag-and-drop interface. It includes a chassis, two wheels, stepper motors, a sensor, and a control system.

3.4.1 MPU 6050 Sensor

The MPU 6050 depicted in Figure 3.6 known as an inertial measurement unit (IMU), is a vital component in the functioning of a self-balancing robot. It integrates both a gyroscope and an accelerometer into a single device. This combination allows the MPU 6050 to measure both angular rate and linear acceleration, which are crucial for the precise control and balance of the robot. The built-in DMP module can convert the angular speed into three angles: pitch angle, roll angle and yaw angle, and can transmit data through I2C and expand temperature sensor or magnetic sensor [21].

The SDA pin of the MPU6050 is connected to the SDA pin of the ESP32, which is often labeled as GPIO (General Purpose Input/Output) pin number 21. Similarly, the SCL pin of the MPU6050 is connected to the SCL pin of the ESP32, usually labeled as GPIO pin number 22. These connections enable the ESP32 to communicate with the MPU6050 sensor and retrieve the accelerometer and gyroscope data for further processing and control in the self-balancing robot system.

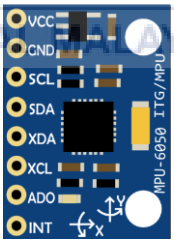


Figure 3.6: MPU 6050 sensor

3.4.2 NodeMCU ESP-32 Microcontroller

The NodeMCU ESP32 serves as the main control unit of the self-balancing robot as shown in Figure 3.7. It handles the overall coordination, computation, and decision-making processes. The NodeMCU ESP32 acts as the micro-controller, in which the RL model or PID mechanism is manually uploaded [22]. The NodeMCU ESP32

receives sensor data from the MPU-6050 sensor, performs calculations based on the PID algorithm, and generates appropriate control signals for the motors.

Figure 3.7: NodeMCU ESP32 Microcontroller

The MPU-6050 sensor, integral to the project, combines a gyroscope and accelerometer to measure the robot's tilt angle, angular velocity, and linear acceleration. It continuously tracks the robot's motion, relaying data to the ESP32 for processing. The NodeMCU ESP32, using I2C protocol, communicates with the MPU-6050, issuing commands for measurements and receiving sensor data. This allows real-time monitoring of the robot's orientation and movement.

Upon receiving raw data from the MPU-6050, the NodeMCU ESP32 processes it to determine crucial metrics like the robot's tilt angle, utilizing the gyroscope's angular velocity and the accelerometer's linear acceleration readings. This tilt angle is then fed into the ESP32's PID controller, which compares it with the desired tilt angle (set point) to generate control signals. These signals regulate motor speed and direction, crucial for maintaining the balance of the two-wheeled robot.

3.4.3 Stepper Motor NEMA 17

The NEMA 17 stepper motor, essential for robotics and automation, features in Figure 3.8. Adhering to NEMA standards with a flange size of 1.7 x 1.7 inches, its operation is based on electromagnetic induction, involving a rotor, stator, and coils

wrapped around the stator poles. By energizing these coils in sequence, the motor achieves precise control of position and speed through stepwise motion. Its holding torque is crucial for tasks requiring stability, particularly in a self-balancing robot, as it helps maintain equilibrium and resist external disturbances.

Figure 3.8: Stepper Motor NEMA 17

Stepper motors provide the highest precision wheel positioning: 200 steps per revolution; in addition, they can be controlled in the range from 1/2 step to 1/16 step, which makes it possible to turn the wheel to the desired angle with the highest accuracy [23]. Moreover, when connected to the ESP32, the microcontroller can send instructions to the motor based on sensor feedback and the PID controller's output, enhancing control and responsiveness in applications like robotics.

3.4.4 Stepper Motor Driver A4988

Figure 3.9 shows the A4988 stepper motor driver that controls motor movement and rotation by regulating current through motor windings and using pulse width modulation (PWM). It receives step and direction signals from a microcontroller like the ESP32, allowing for adjustable rotation and speed by varying the step pulses' duration and frequency.

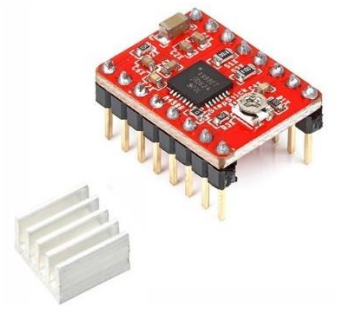


Figure 3.9: A4988 Stepper motor driver

The A4988 driver, enhancing motor control, features customizable current limitation, micro-stepping, thermal shutdown, and overcurrent protection, ensuring efficient and safe operation of stepper motors. It supports a logic control voltage of up to 5.5 V and a motor driving voltage of up to 35 V, capable of delivering up to 2 Ampere per coil [25]. Its ability to manage and regulate current to motor windings allows for precise adjustment of torque and performance, tailored to specific application needs. Users can optimize current flow through motor coils by adjusting the reference voltage or using current limit resistors, thus ensuring optimal motor operation.

3.4.5 A4988 Module Breakout Board

Figure 3.10 illustrates a critical component in stepper motor control systems. It functions as a driver, facilitating precise control over the movement of stepper motors. Operating by interpreting electrical signals from a microcontroller, the A4988 regulates the current sent to the stepper motor windings, ensuring accurate steps and smooth motion. This module employs a chopper drive technique to manage the current flow, minimizing power consumption and heat generation. Key features include adjustable micro-stepping for finer motor control, thermal protection, and overcurrent detection. Its role in converting digital signals into precise motor movements makes it

an indispensable component in various applications requiring precise and controlled motion.

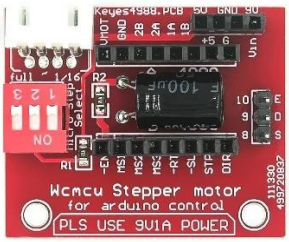


Figure 3.10: A4988 Module Breakout Board

3.4.6 18650 Lithium Ion Battery

Figure 3.11 displays the 18650 Lithium-ion battery, notable for its high energy density, which allows it to store a significant amount of energy relative to its size and weight. This rechargeable battery, measuring 18mm in diameter and 65mm in length, typically operates at around 3.7 volts, with voltage varying from about 4.2 volts when fully charged to 3.0 volts when depleted.

Figure 3.11: 18650 Lithium Ion Battery

For effective and safe use, it requires a compatible charger and circuitry to prevent overcharging or excessive discharge. Despite advanced battery management systems, overcharging can still occur due to battery pack inconsistencies, leading to accelerated deterioration, malfunction, and potentially dangerous situations like thermal runaway or explosions [26].

3.5 Software and Application Used

3.5.1 Arduino IDE

Arduino IDE is a popular software tool used for programming microcontrollers, including Arduino boards as shown in Figure 3.12. It is a user-friendly platform for programming microcontrollers like Arduino boards and ESP32. It is widely used, open-source software compatible with multiple operating systems (Windows, Linux, Unix, Macintosh) and supports languages such as C and C++. It is ideal for various projects, from web to gaming applications which is also essential for writing and uploading code, particularly in embedded systems development. For this project, Arduino IDE is utilized on an ESP32 microcontroller to implement and execute the programming tasks.

Figure 3.12: Arduino IDE Software

3.5.2 Blynk

Figure 3.13 illustrates Blynk Apps, an Internet of Things (IoT) platform equipped with iOS and Android applications that enable control of NodeMCU ESP32 devices over the internet using smartphones. Blynk is a platform for creating GUIs for IoT applications, available on the Google Play Store for Android and the App Store for iOS. Users can sign up and set up projects within the app. A key step is selecting the option to receive an Authenticate ID via email, which is essential for coding sensor data upload to the Blynk Application.

Figure 3.13: Blynk Application

3.5.3 MATLAB

Figure 3.14 shows the MATLAB software which is widely used numerical computing environment that serves as an essential tool for engineers, scientists, and researchers across various disciplines. MATLAB helps them solve all sorts of math problems, analyze data, and create algorithms. It's kind of like a digital toolbox full of tools for working with numbers and making sense of complicated information. Developed by MathWorks, MATLAB provides a versatile platform for mathematical modeling, data analysis, algorithm development, and visualization. It's a handy tool that makes complex tasks easier and faster for professionals in different fields.

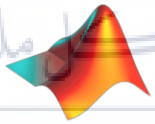


Figure 3.14: MATLAB Software

3.6 Project Operation Development

This proposed system combines hardware implementation and software implementation. The hardware of the system uses PID controller as the controller which needs to be tuned and calibrated in order to get the accurate reading of the sensor.

3.6.1 Project Implementation Procedure

This subtopic will include a brief explanation of this project procedure. It involves Hardware Preparation, which will focus on constructing and configuring hardware components; Software Implementation, which will explain the installation and procedure to use the software, and Hardware and Software Implementation, which will discuss the integration of both hardware and software for this system.

3.6.1.1 Hardware Preparation

The project is organized into three distinct stages. The initial stage focuses on hardware preparation, encompassing the setup of hardware components, the configuration of the Integrated Development Environment (IDE), and the establishment of necessary hardware connections. This stage also includes the programming of the hardware components to ensure they operate correctly. The subsection below will elaborate on the specific detail of each stage:

1. Install Arduino IDE with ESP32 add-ons and all necessary libraries.
2. Define PID parameters and terms in the code.
3. Verify code for errors.
4. Upload code to ESP32.
5. Connect hardware to correct pins.
6. Verify connections for accuracy.
7. Connect ESP32 to Wi-Fi/hotspot.
8. Monitor MPU6050 sensor pitch angle in Serial Monitor.
9. Calibrate if offset exists.
10. Set current limit for NEMA17 motor in a4988 driver.
11. Analyze self-balancing robot's performance.

3.6.1.2 Software Implementation

There are multiple software platforms that have been used in this project which are Blynk as the IoT platform, Arduino IDE as the hardware setup and MATLAB software as the tuning parameters analysis implementation. Thus, the process involved will be discussed in this subtopic. The full codes for every software used are presented in the appendix.

3.6.1.2.1 Blynk

Blynk application is a very powerful IoT platform which allows users to easily create mobile applications to control and monitor hardware projects [24]. The Blynk Console is a central hub for managing devices, templates, datastreams, and application settings. First and foremost, users need to add and configure the IoT devices used in the Blynk Console. After logging in, navigate to the "Devices" section and click on "New Device" as shown in Figure 3.15. To choose the hardware type, authentication method, and connection type, which is Wi-Fi, simply follow the prompts.

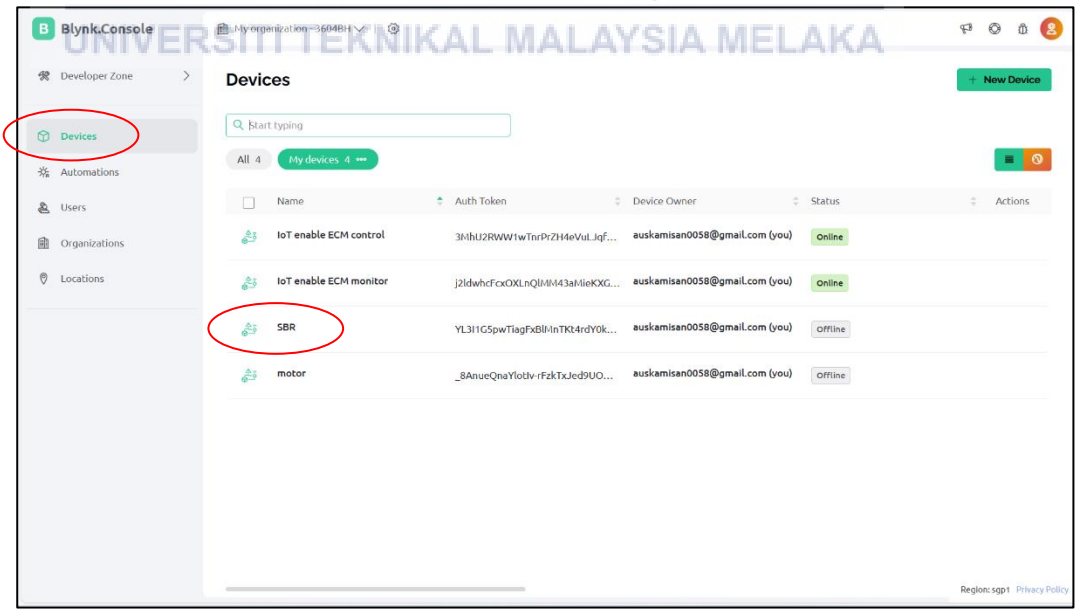


Figure 3.15: Device Layout in Blynk Console

Users will obtain authentication credentials, such as an Auth Token, when the device is connected. These credentials are needed to link the hardware to the Blynk platform. Next, Blynk offers a variety of templates as depicted in Figure 3.16 that simplify the process of creating an application interface for any IoT project. Select a template from the "Templates" section that fits the specifications of the project.

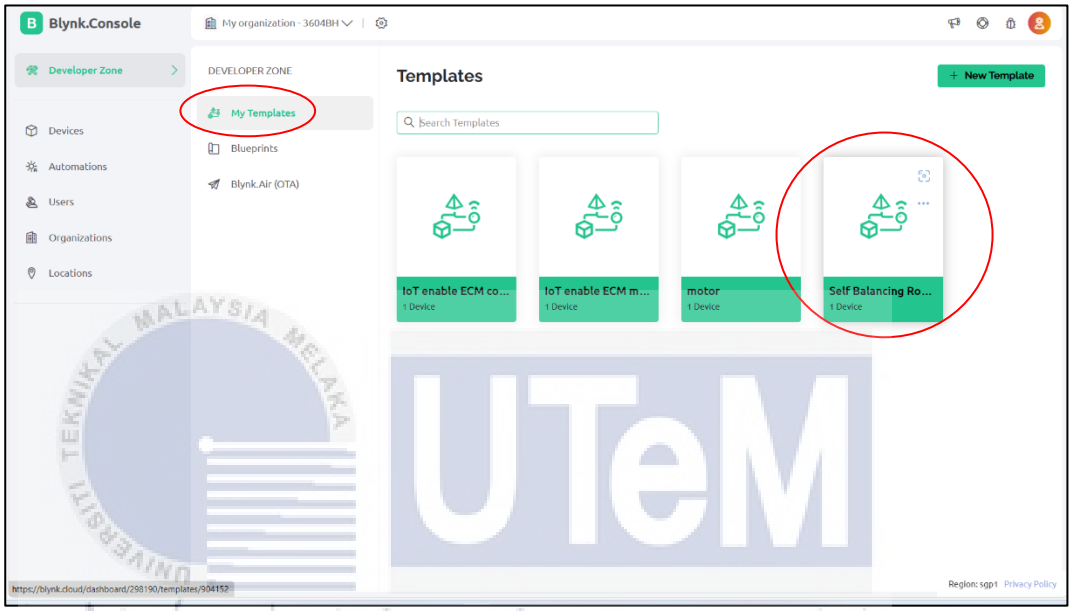


Figure 3.16: Template Layout in Blynk Console

Pre-configured widgets such as graphs, sliders, buttons, and displays are part of templates. The selected template automatically configures the device's interface within the Blynk mobile app, simplifying the setup process.

Furthermore, when the device and template are configured, the next step is to set up “Datastreams” to easily communicate between the hardware and the Blynk app. For linking virtual pins to certain hardware pins on the device, navigate to the "Datastreams" section and create them. This can be seen in Figure 3.17. Transmitting data is accomplished using these virtual pins. Then, set up each virtual pin according to the system’s specifications by selecting the direction (input or output) and data type.

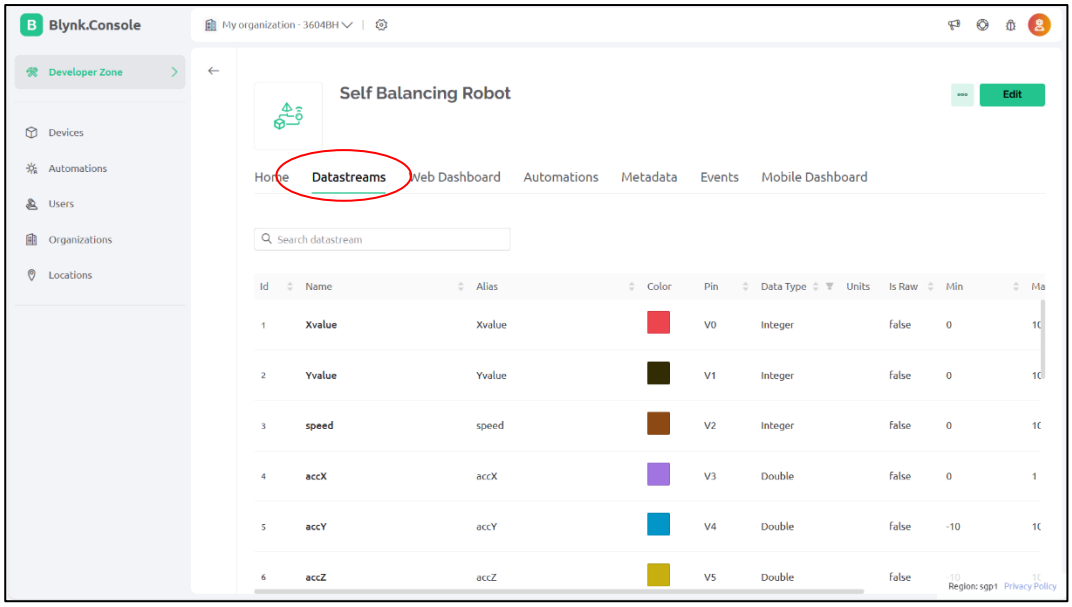


Figure 3.17: Datastreams Layout in Blynk Console

Finally, users can go to the "Applications" section to customize the appearance and behavior of the mobile app. By adding and modifying widgets, the user may specify the user interface's design, theme, and functionality. In order to connect the app to the hardware, link these widgets to the virtual pins that are already generated. After saving the application's settings, the device is ready to launch the project. Thus, the Blynk mobile app may now be used to access and operate the system.

3.6.1.2.2 MATLAB

As for the MATLAB software, the process of analysis begins by installing MATLAB on the computer or laptop. The installation instructions are provided by MATLAB to ease the user to do the installation procedure. The next step involves launching the software. After opening MATLAB, users can initiate the process analysis by creating a new Script, as illustrated in Figure 3.18. Once the MATLAB environment is set up, real-time data integration from the ESP32 begins. This entails establishing a communication link between the ESP32 microcontroller and MATLAB.

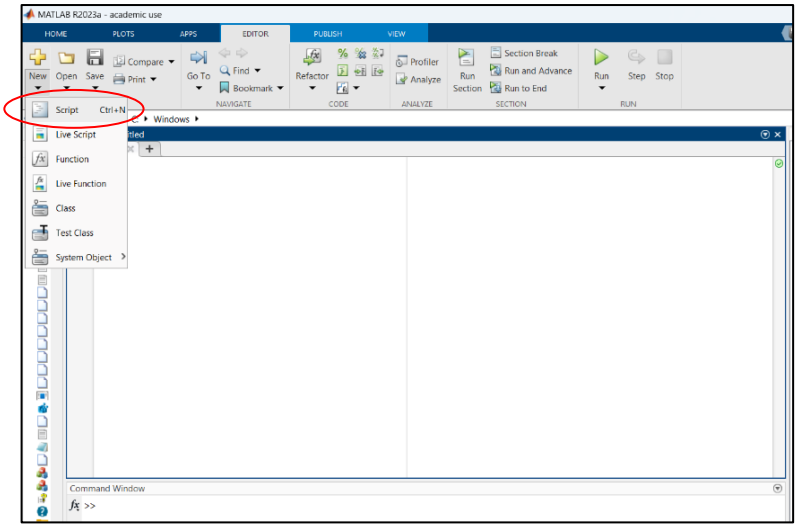


Figure 3.18: New script in MATLAB

MATLAB offers different communication interfaces, including Serial Communication and Wi-Fi, depending on the ESP32's connectivity options. For Serial Communication, MATLAB's 'serial' functions can be utilized, specifying parameters like baud rate. Furthermore, MATLAB offers tools for data visualization and analysis, allowing users to create plots, graphs, and dashboards to interpret and monitor real-time data from the ESP32 and it can be updated dynamically as new data is received, providing a comprehensive and interactive view of the system's performance.

3.6.1.3 Hardware and Software Integration

The primary focus of this subtopic is to establish a connection between the previously developed hardware and software. Figure 3.19 shows the block diagram for hardware and software integration.

Figure 3.19: Block Diagram for Hardware and Software Integration.

The integration process starts with the ESP32 microcontroller sending sensor data to the Blynk platform. Blynk then manages PID tuning and system tuning data, which is relayed back to the ESP32 for effective control and optimization. Blynk acts as a central control hub, dynamically adjusting PID tuning and system parameters based on real-time sensor feedback to maintain system responsiveness. Simultaneously, the ESP32 continuously transmits data to MATLAB for in-depth analysis. This two-way communication establishes a comprehensive feedback loop, allowing MATLAB to process real-time hardware data and offer insights for ongoing system refinement. Overall, the completion of this proposed system requires multidirectional communication link between the hardware and the IoT platform.

3.7 Project Parameter of Analysis

This project develops a two-wheel self-balancing robot to study PID controller tuning parameters, crucial for the robot's balance. Key aspects include analyzing parameters affecting rise time, settling time, overshoot, and steady-state error, as well as the robot's stability and performance under disturbances. The upcoming sub-chapter will detail the methods for analyzing these parameters.

3.7.1 Parameter of Analysis and Data Acquisition

The two parameters that are analyzed in this project are the tuning parameters that affect rise time, settling time and overshoot and steady state error as well as stability and performance of the self-balancing robot when involved with multiple disturbances. To clarify this method of analysis, the real-time sensor data is transmitted to ESP32 which then relays serial data to MATLAB. In return, MATLAB reads this serial data to derive PID parameters (Kp, Ki, and Kd) and pitch angle as input and output.

Once MATLAB receives the serial data, it generates an input (pitch angle) vs. time graph, enabling the evaluation of rise time, settling time, overshoot, and steady-state error for various PID gain settings. From the graphs, the parameters need to be calculated manually. Firstly, overshoot (%) needs to be calculated to evaluate a system's transient performance, providing insight into how much the system's output exceeds the desired value before settling. The overshoot data often used to optimize control parameters, ensuring a balance between response speed and stability. The overshoot (%) can be mathematically written as:

$$Overshoot\ (\%) = \frac{Peak\ Value - Desired\ Value}{Desired\ Value}\ x\ 100 \qquad (2)$$

Next, rise time is a measure of the time it takes for a system to reach a certain percentage of its final value. In context of self-balancing robot, rise time is calculated by finding the difference of time from pitch angle before it recovers to pitch angle after it recover from imbalance. The difference is then measured in seconds. This is because the library used for sensor calibration did not have a fixed offset during calibration process. Therefore, the new offsets post-calibration considers a stable pitch angle, while the stable time refers to how long it takes for a self-balancing robot to regain equilibrium after an imbalance.

3.8 Chapter Summary

Overall, the methodology for this project encompasses the selection of components, detailed explanations of the project and system flowcharts, and thorough descriptions of hardware connections to the IoT server (Blynk) and MATLAB. Additionally, the analysis parameters, focusing on the PID tuning method, are extensively covered.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Chapter Overview

This chapter focuses on establishing the proposed system discussed earlier, highlighting key features, hardware and software implementation, and their integration. Additionally, critical aspects of the code for functionality will be explored. The chapter concludes with an analysis of the PID tuning's impact on the self-balancing robot.

4.2 Hardware Prototype Implementation

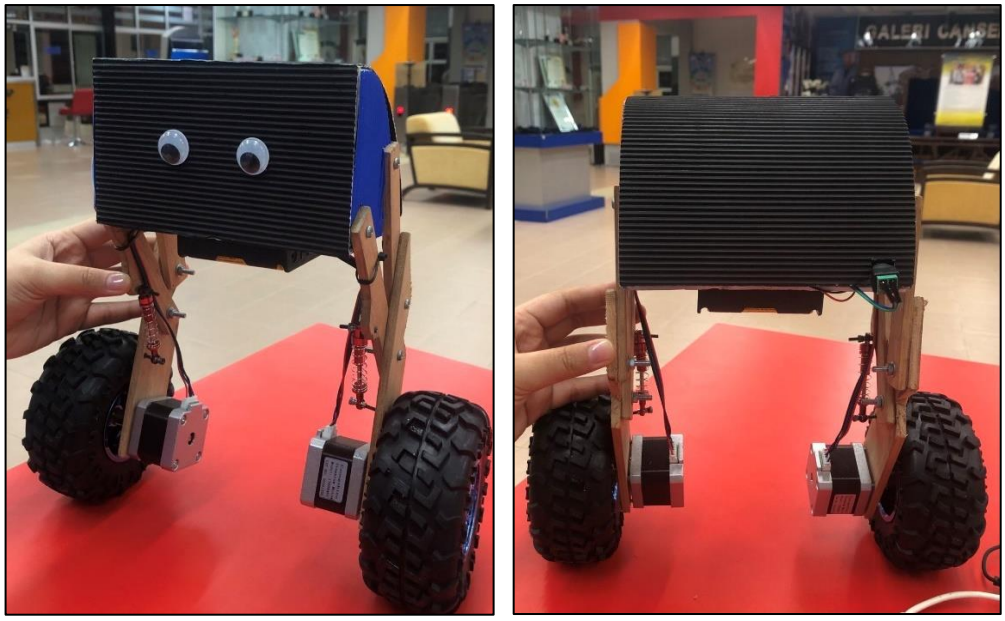
This section explores how the project will be implemented, starting with a thorough explanation of the hardware design. It navigates through the selection and specifications of components, emphasizing the rationale behind each choice. Next, the wiring layout and interconnections, revealing the electrical pathways through detailed

schematics. The calibration and testing procedures section is an important part that covers the methodical steps taken to confirm the prototype works. These sections cover how the hardware prototype is set up, connected, and tested in the project, providing a full understanding of its architecture and functionality.

4.2.1 Hardware Design Overview

This section delves into the physical aspects of the project, explaining how the hardware prototype is designed and built, including the system's overall structure and component choices. This section helps to understand why certain design decisions were made for the hardware prototype, laying the foundation for upcoming discussions. To develop a system, having a prototype is essential as it allows for the systematic progression through the various stages of the development process. A carefully planned prototype helps prevent future problems, streamlining development and reducing challenges in the final product stages.

In this project, the body and chassis of the robot are crafted from MDF board and soft PVC foam. Suspension is achieved by inserting springs between the two joints of each leg or joint of the robot. To enhance stability, 130x60mm hollow rubber wheels were selected, introducing an additional dampening factor. Subsequently, all wheels, joints, and chassis components are assembled to create the self-balancing robot. The circuit is then transferred to the prototype for performance testing. The wiring is arranged within the robot's chassis to prevent potential short-circuits. The final prototype, inclusive of the circuit, is depicted in Figure 4.1 (a) and (b).



(a) (b)

Figure 4.1: Front and rear view of the prototype

The process of building the prototype begins with creating the joint for the two-wheel self-balancing robot started with precise measurements of the required size. MDF board was selected for this purpose as it gives a strong base to withstand any pressure given to the robot. The dimensions were marked before cutting the pieces using a wood cutter as shown in Appendix A.

Next, the process continues with the smoothing process using sandpaper to achieve a clean and polished finish. Following this preparation, the joint pieces were securely attached to the stepper motors as shown in Appendix A, which serve as the robot's legs, and then connected to the wheels. This joint assembly plays a crucial role in the robot's structure, enabling it to maintain balance and mobility while carrying out its functions effectively.

Furthermore, the chassis was designed with careful consideration of the weight distribution of the robot, which plays a critical role in ensuring its balance.

Subsequently, the chassis components were connected to the robot's joints, leading to the development of a functional prototype as shown in Figure 4.1 (a) and (b).

Moreover, the circuit connections are situated within the robot's chassis to facilitate optimal placement of the sensor at the center of gravity. This strategic positioning of the sensor is essential for accurate balance and stability control of the robot. This is to ensure it can effectively respond to its environment and maintain equilibrium during operation. The circuit connections of the prototype will further be discussed in the next sub-topic.

4.2.2 Wiring and Interconnections

Figure 4.2 shows the schematic circuit of a two-wheel self-balancing robot. Schematic diagram is a representation of the elements of a system which uses graphic symbols rather than realistic pictures.

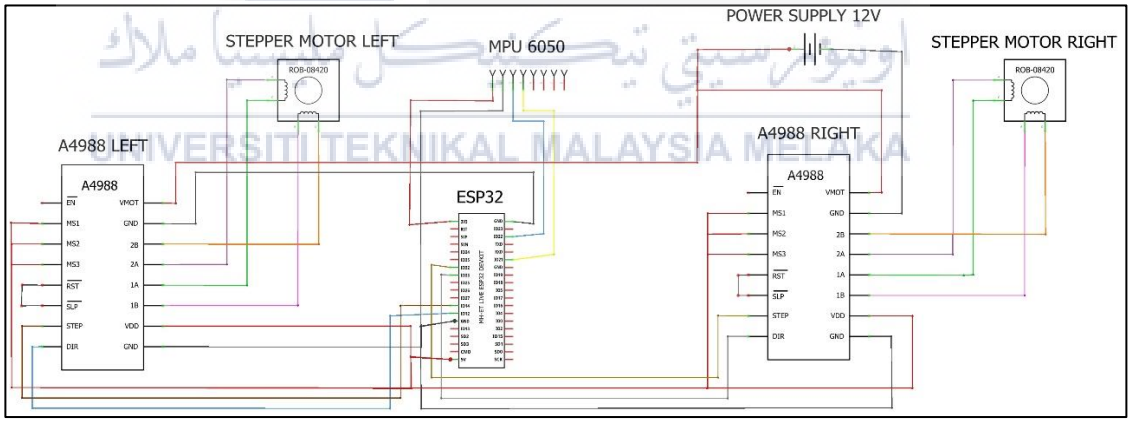


Figure 4.2: Schematic Circuit of the system

The schematic circuit facilitates tracing the circuit and its functions without regard to the actual physical size, shape or location of the component devices or parts. The connection of the circuit can be clearly seen in the schematic diagram shown above which is made by using Fritzing software.

In this project, the functionality of the product is crucial for its success. Therefore, a circuit that exhibits no errors is being contemplated to ensure the project operates successfully. The circuit depicted above is utilized for the two-wheel self-balancing robot, which functions to control the movement of the wheels through stepper motors based on the readings obtained from the MPU6050 sensor. The NEMA17 stepper motor operates by energizing coils in its stationary stator, creating magnetic fields that attract its permanent magnet rotor and induce controlled rotational movement in discrete steps. Figure 4.3 illustrates the completed circuit connections. Before placing the finalized circuit on the prototype, a thorough rechecking of the connections is conducted to prevent any potential issues related to loosen wiring.

Figure 4.3: Completed circuit connection of the system

A 12V power source is applied to the circuit through the VCC pin on the ESP32 and the VMOT pin on both A4988 stepper motor drivers. The MPU6050 sensor is connected to the ESP32, with the SCL (Serial Clock) pin connected to D22 and the

SDA (Serial Data) pin connected to D21. These connections enable the ESP32 to establish communication with the MPU6050 sensor, allowing for the bidirectional exchange of data. This capability enables the microcontroller to read sensor data, including acceleration and gyroscopic information, and exert control over the sensor's configuration.

In this project, two A4988 stepper motor drivers are mounted on breakout boards, simplifying integration with features like voltage regulation and easy-to-use headers. Similarly, an ESP32 expansion board is employed, offering enhanced versatility and user-friendliness compared to the standalone ESP32. The built-in power regulation, convenient headers, and expanded GPIO options are very efficient to be used.

4.3 Software Implementation

In this project, various software platforms have been utilized, including the Arduino IDE for hardware setup, Blynk as the IoT platform, and MATLAB for tuning parameter analysis. In this subtopic, the key features and functionality of certain codes will be discussed. The complete code for each of the software platforms can be found in the appendix.

4.3.1 Arduino IDE Platform

The Arduino IDE platform is very crucial to set up the hardware that has been developed. It helps users to write and upload code in C++ language to control Arduino microcontrollers, enabling users to create and control various electronic projects. The sub-topic below will discuss some of the important codes based on different configurations.

4.3.1.1 Motor Configuration

The motor configuration in the Arduino IDE is to set up and control the two stepper motors responsible for the movement of the self-balancing robot. From Figure 4.4 below, it shows one of the codes of the motor configuration that can be observed.

```
26  // Motor control pins
27  int leftStepPin = 32;
28  int leftDirPin = 33;
29  int rightStepPin = 14;
30  int rightDirPin = 12;
31  int enablePin = 19;
```

Figure 4.4: Motor Pin Configuration

The section of the code assigns specific GPIO pins on the ESP32 microcontroller to control the left and right stepper motors. The ‘leftStepPin’ and ‘rightStepPin’ represent the pins responsible for sending step signals to the motors, ‘leftDirPin’ and ‘rightDirPin’ control the motor direction, and ‘enablePin’ enables or disables the motors as needed during operation.

Next, the code in Figure 4.5 initializes the ‘FastAccelStepperEngine’, which is a library used for controlling stepper motors with smooth acceleration and deceleration. It connects the defined pins for the left and right stepper motors to the engine. Two pointers, "leftMotor" and "rightMotor," are initialized as NULL, indicating that they will later point to instances of the ‘FastAccelStepper’ class to control the left and right motors respectively.

```
33  // Create FastAccelStepper Engine
34  FastAccelStepperEngine engine = FastAccelStepperEngine();
35  FastAccelStepper *leftMotor = NULL;
36  FastAccelStepper *rightMotor = NULL;
```

Figure 4.5: Initialization of ‘FastAccelStepperEngine’ and Motor Pointers

In the ‘setup()’ function which can be seen in Figure 4.6 and Appendix B, it begins by configuring the serial communication for debugging via ‘Serial.begin(9600)’. Then, it sets up the connection to the Blynk IoT platform, WiFi via ‘Blynk.begin(auth, ssid, pass)’, and initializes the I2C communication with the MPU6050 gyroscope and accelerometer sensor using ‘Wire.begin()’ and ‘mpu6050.begin()’. Gyroscope offsets are calculated for calibration with ‘mpu6050.calcGyroOffsets(true)’.

```
73  void setup() {
74    Serial.begin(9600);
75    Blynk.begin(auth, ssid, pass);
76    Wire.begin();
77    mpu6050.begin();
78    mpu6050.calcGyroOffsets(true);
```

Figure 4.6: Initialization and Configuration in the setup() Function

Then, in the 'setup()' function, the 'FastAccelStepper' engine initializes for stepper motor control. Motors are connected to their pins, and parameters like direction, enablement, and speed are set. The PID controller for balance and stability is also configured. The value "2000" denotes the stepper motors' speed in steps per second, with a higher Hz indicating faster motor movement.

4.3.1.2 PID Configuration

PID configuration is crucial for the self-balancing robot's control, maintaining balance by adjusting motor speeds according to sensor inputs and setpoints. It begins with specific code lines in the 'setup()' function, setting output limits of "-255" and "255" as depicted in Figure 4.7. These limits define the maximum speed range for motor rotation in both clockwise and counterclockwise directions. Next, the following line sets the PID's sampling time. In this code, it is set to 1 millisecond (1 ms), meaning that the PID controller calculates motor speed adjustments every 1 ms, ensuring rapid response to changes in the robot's orientation. Then, the PID controller is set to

automatic mode which means it continuously calculates and adjusts the motor speeds to maintain the robot's balance automatically. In automatic mode, the PID controller operates in real-time, making it well-suited for applications like self-balancing robots.

```
98    myPID.SetOutputLimits(-255, 255);
99    myPID.SetSampleTime(1);
100   myPID.SetMode(AUTOMATIC);
```

Figure 4.7: Output Limit in PID Configuration

In the 'loop()' function, the PID controller calculations and adjustments are executed, as shown in Figure 4.8 Here, 'Input' is the angle from the MPU6050 sensor, adjusted by 'Inputoffset', representing the robot's current angle for PID feedback. Next, ‘myPID.SetTunings(Kp, Ki, Kd)’ updates the PID controller's tuning parameters (proportional, integral, and derivative gains) based on the values received from the Blynk app (V9, V10, V11). In addition, ‘myPID.Compute()’ calculates the PID control output (‘Output’) based on the difference between the current angle (‘Input’) and the desired setpoint (‘Setpoint’). The PID controller adjusts ‘Output’ to control the motor speeds, ultimately maintaining the robot's balance.


```
108   Input = mpu6050.getAngleY() - Inputoffset;
109   myPID.SetTunings(Kp, Ki, Kd);
110   myPID.Compute();
```

Figure 4.8: PID Controller Calculations and Adjustments

Thus, the PID configuration in this code is responsible for fine-tuning the robot's motor control to achieve and maintain balance. It continuously adjusts motor speeds based on sensor feedback and tuning parameters to ensure that the robot remains upright and stable during operation.

4.3.1.3 Sensor Configuration

As for the sensor configuration, it is primarily located within the 'setup()' function as illustrated in Figure 4.9. The 'Wire.begin()' line initializes the I2C communication, which is essential for communicating with the MPU6050 sensor and 'mpu6050.begin()' means that the sensor is ready to provide data. Then, the next following line calculates and sets gyro offsets for calibration. The 'true' argument indicates that gyro offsets should be calculated and saved. Calibration ensures that the sensor provides accurate data, which is crucial for self-balancing operation.

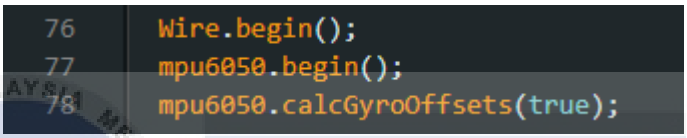


Figure 4.9: MPU6050 Configuration

Therefore, this sensor configuration is very crucial as it needs to be properly set up and calibrated to provide accurate gyroscopic data which is used in the self-balancing control algorithm.

4.3.1.4 Blynk Configuration

Blynk app is used to communicate between the software and hardware. Figure 4.10 shows the Blynk configuration that is in the 'setup()' function as well. 'Serial.begin(9600)' line initializes the serial communication for debugging purposes, allowing users to send and receive data between the ESP32 and the computer at a baud rate of 9600. The subsequent line in the code initiates the connection to Blynk, using the provided 'auth' token, Wi-Fi 'ssid', and 'pass' (password). Blynk, an IoT platform, enables remote control and monitoring of devices through its cloud platform.

```
74    Serial.begin(9600);
75    Blynk.begin(auth, ssid, pass);
```

Figure 4.10: Code that configures Blynk

In addition, this configuration includes Blynk widget configuration settings that involve writing to specific virtual pins. For instance, in Figure 4.11, when a value is changed on virtual pin V9 through the Blynk app, it triggers the 'BLYNK_WRITE' function, which updates the Kp (proportional gain) variable with the new value received from the app. These settings are used to update parameters such as PID tuning constants and other control variables from the Blynk app. These parts of the code are within the 'BLYNK_WRITE' functions for various virtual pins such as V9, V10, V11, and others.

```
42    BLYNK_WRITE(V9) {
43      Kp = param.asDouble();
44    }
```

Figure 4.11: 'BLYNK_WRITE' function in Blynk widget configuration settings

4.3.2 Blynk Platform

This project utilizes the Blynk app for tuning PID control and stepper motor parameters. Users can adjust parameters using 'virtual sliders' in the app, avoiding code changes in the Arduino IDE. PID tuning maintains stability by altering motor output. Kp affects response strength to errors, with higher values for more aggressive corrections. Ki addresses accumulated past errors, aiding in steady-state error elimination by adjusting the control signal over time.

Moreover, Kd looks at the rate of change of error. It helps dampen oscillations and overshoot by anticipating how quickly the error is changing. Figure 4.12 shows the Blynk GUI Layout which contains the widget used for the tuning purpose. Apart from PID tuning, Blynk also allows users to fine-tune various parameters related to the stepper motors, which directly influence the robot's motion and balance. There are six parameters which are involved in system tuning or stepper motor tuning. Firstly, the scale factor adjusts the overall speed of the motors. Users can control how quickly the robot responds to balance corrections by increasing or decreasing this factor.

Next, scale factor plus can be adjusted to control the speed of the motor. This is because it provides extra control over the robot's movement. Besides, scale factor (reverse) and scale factor (forward) enable users to independently control motor speeds when moving in reverse and forward directions. As for acceleration, it determines how quickly the motors change speed. Higher acceleration values result in faster changes, while lower values provide smoother and more gradual acceleration. Lastly, the dead zone parameter is a critical parameter that defines a range within which the robot remains stationary. It helps prevent unnecessary movement and ensures the robot remains stable when the error (tilt) is within this range.

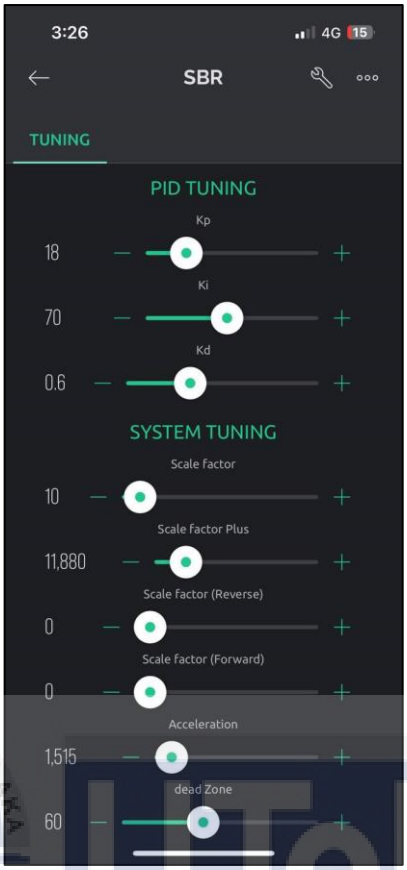


Figure 4.12: Blynk GUI Layout of self-balancing robot

Thus, users can easily adjust these settings to optimize the self-balancing robot's performance in order to ensure it responds accurately to changes or disturbances in its orientation and motion.

4.3.3 MATLAB Platform

As mentioned earlier, MATLAB software is used for analysis purposes. MATLAB can communicate with the ESP32 using the Serial Communication Toolbox or by using functions like serial to configure the serial port settings. In this project, MATLAB is used to produce pitch angle data vs time. Figure 4.13 shows the graph plotted when Kp = 18 after running the code in Editor.

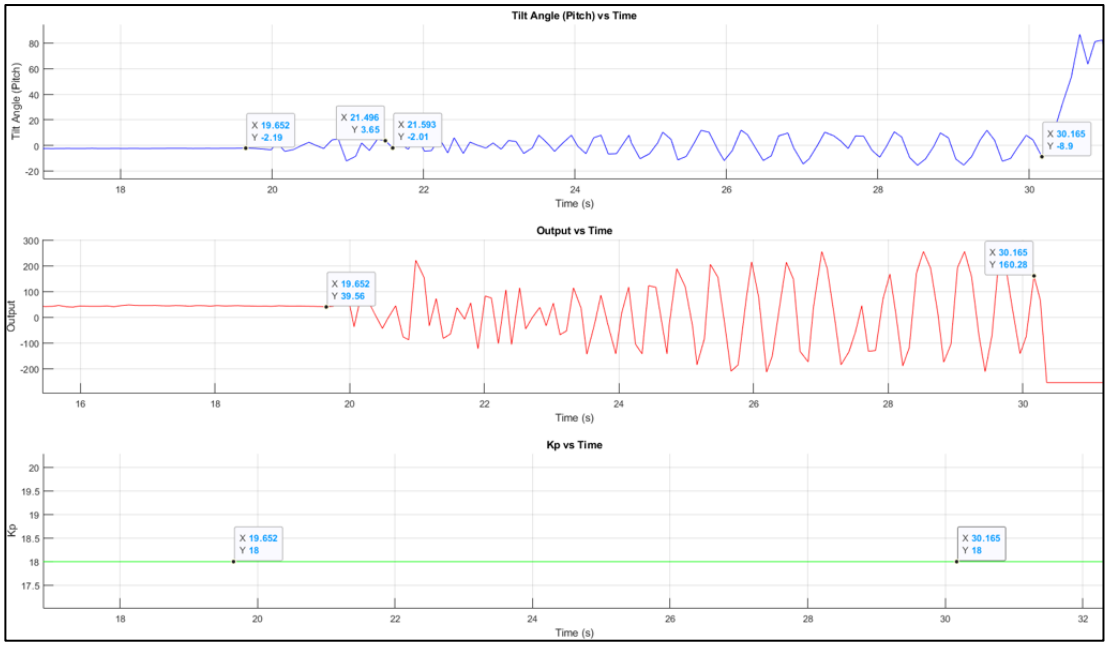


Figure 4.13: Graph of Tilt Angle vs Time when Kp = 18

From the code generated in MATLAB, the plot of input vs time for all the tuning parameters can be analyzed. The rise time, settling time, overshoot and steady state error analysis are performed by analysing the tilt angle (pitch) vs time plot. The analysis for the tuning parameters will be analyzed and discussed in-depth in the next sub-topics.

4.4 Analysis Result

This sub-topic analyzes key parameters, starting with the impact of varying Kp values on rise time. Rise time is the duration for a process variable to reach its setpoint within a tolerance band after control input. A shorter rise time in a PID controller means a quicker response to system changes or disturbances. Increasing the Kp value can shorten rise time, resulting in a more aggressive system response.

4.4.1 Tuning Parameters and Comparison Discussion

This sub-topic focuses on three crucial parameters for optimizing control systems, examining their impact on system performance through graphs and tables. It includes

evaluations of the system's response to disturbances, aiming to enhance stability and disturbance management.

4.4.1.1 Kp Comparison Analysis

This sub-topic will cover a detailed Kp comparison analysis, including graphical interpretations using MATLAB plots and a comparative table contrasting different parameter values. It also involves performance testing of the robot under various disturbances.

4.4.1.1.1 Graphical Interpretation

The Kp analysis in the project involved evaluating five different Kp value ranges to find the ideal balance between stability and achieving the fastest rise time in the control system. A quicker rise time allows the system to respond rapidly to disturbances, but it must be done cautiously to avoid making the robot shaky or unstable. Overly aggressive tuning of Kp values leads to instability and oscillations in the control system of the robot.

For Kp value set to Kp = 4.5, three graphs are depicted in Figure 4.14 as part of the analysis process. To determine the robot's rise time at Kp = 4.5, the tilt angle (pitch) vs. Time graph is examined. Ki and Kd for all the Kp values are initially set to 0. This analysis reveals that the robot attempts to recover from imbalance, transitioning from a tilt angle of -23.95° to 2.95° with a rise time of 0.194 seconds. However, the robot experiences a fall due to its slow response, as evident in the Output vs. Time graph. Nevertheless, the robot achieves stability in less than 1 second, precisely at 0.928 seconds, before applying any disturbances.

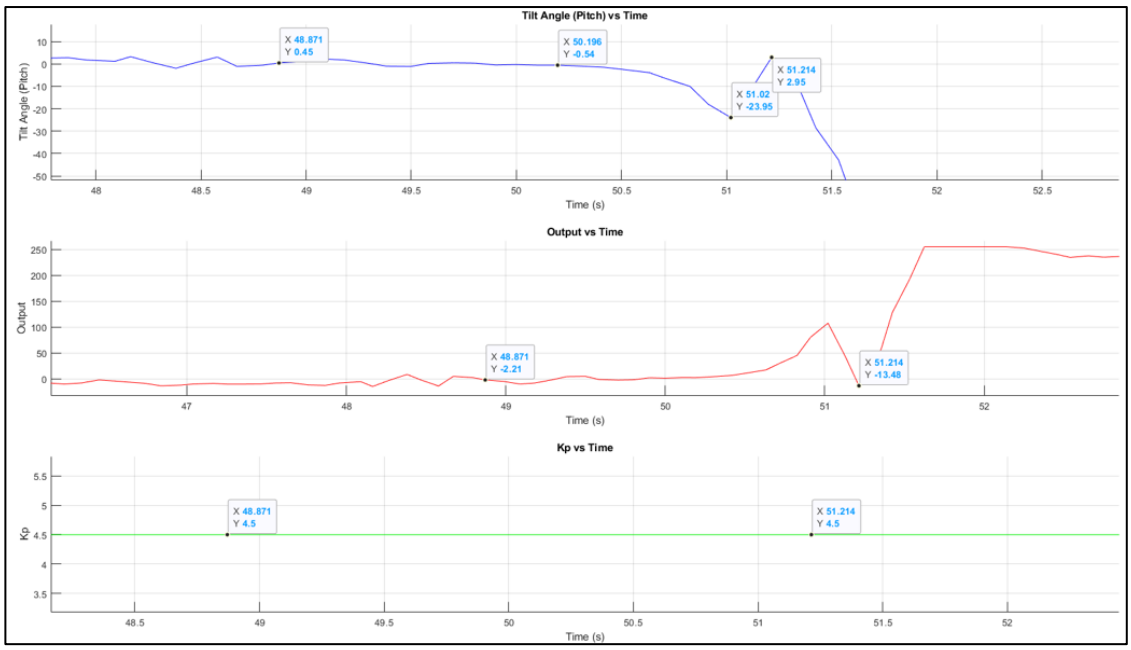


Figure 4.14: Behavioral Response when Kp = 4.5

Next, when the Kp value is configured to Kp = 9.0 as shown in Figure 4.15, the robot's rise time is determined by examining the tilt angle (pitch) vs. Time graph. This examination reveals that the robot endeavors to recover from an imbalance, transitioning from a tilt angle of 5.73° to -7.58° within a rise time of 0.21 seconds. Nonetheless, the robot experiences a fall due to its sluggish response, which is apparent in the Output vs. Time graph. However, the robot attains stability at 1.127 seconds, without any disturbances being applied.

Furthermore, the Kp value is set to Kp = 13.5 which is slightly below the optimal Kp. Figure 4.16 shows the behavioral response of the robot when Kp = 13.5. This analysis reveals that the robot strives to recover from an imbalance, shifting from a tilt angle of -7.54° to 1.13° with a rise time of 0.1 seconds. Despite this, the robot experiences a fall due to its delayed response. Nevertheless, the robot reaches stability after 2.25 seconds which is slightly slower than Kp = 9.0, even without any external disturbances.

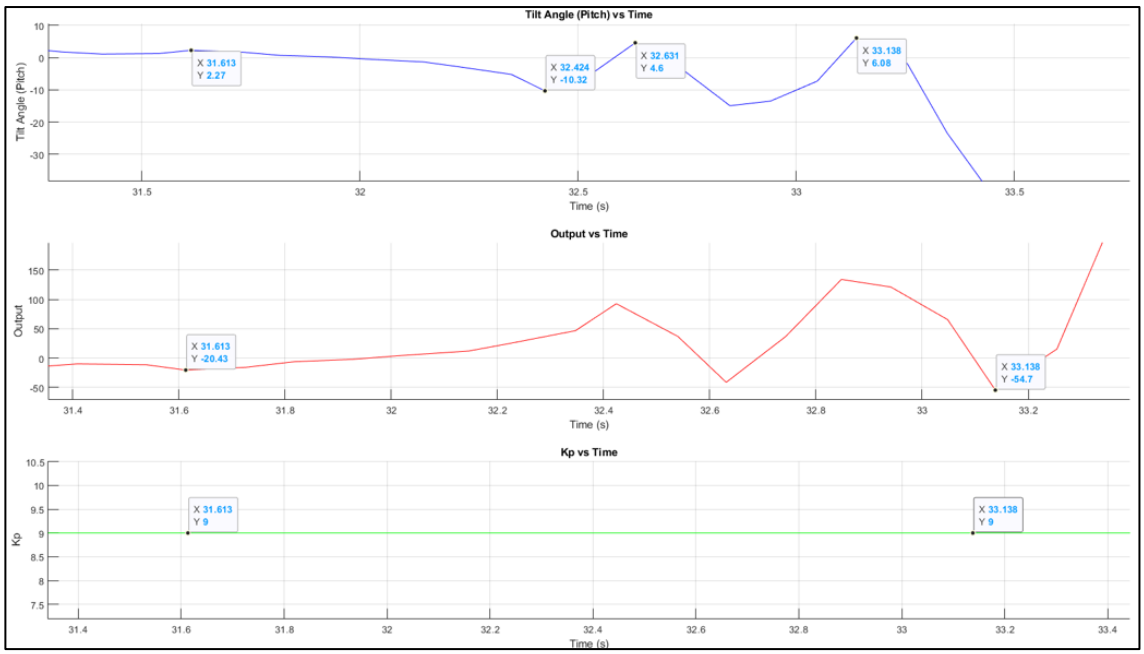


Figure 4.15: Behavioral Response when Kp = 9.0

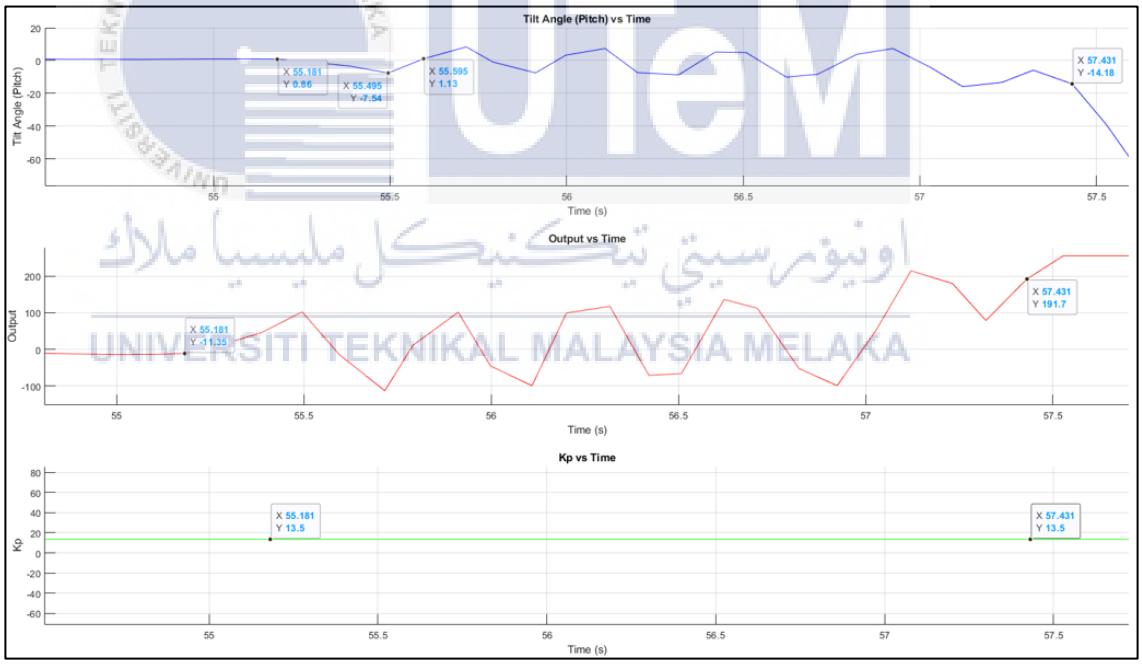


Figure 4.16: Behavioral Response when Kp = 13.5

Besides, the next value of Kp is set to Kp = 18.0 as demonstrated in Figure 4.17, to examine the robot's behavior under these conditions. This analysis unveils that the robot endeavors to regain stability after an imbalance, transitioning from a tilt angle of 3.65° to -2.01° within a rise time of 0.097 seconds, significantly shorter than the

previous rise times. On the other hand, the robot takes more time to experience a decline due to its balanced response. Thus, the robot achieves stability after 10.17 seconds which is the longest compared to previous stable time. In this test analysis, if the stability duration exceeds 10 seconds, the robot is subjected to disturbances to assess its stability.

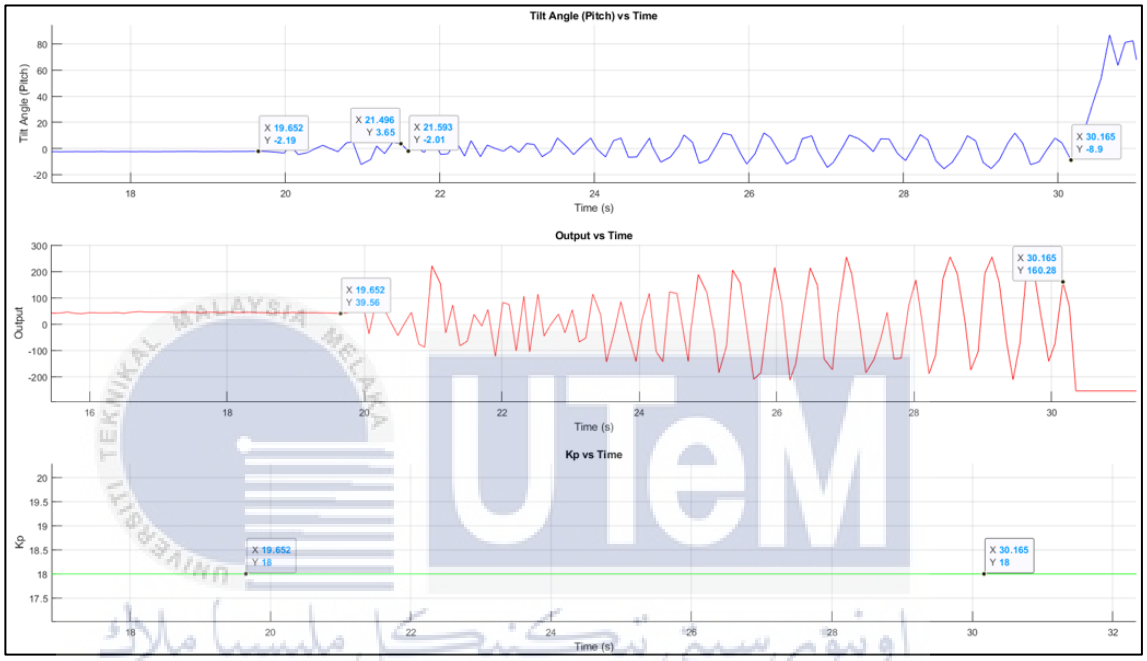


Figure 4.17: Behavioral Response when Kp = 18.0

Moreover, with the Kp value set to Kp = 22.5, the robot's behavioral response is depicted in Figure 4.18. This analysis uncovers that the robot diligently works to regain balance, moving from a tilt angle of -5.88° to 0.98° with a rise time of 0.097 seconds. It takes the robot approximately 1.627 seconds (without disturbance) to undergo a decline due to its jittery response. Nonetheless, the robot attains stability after this duration of 1.627 seconds.

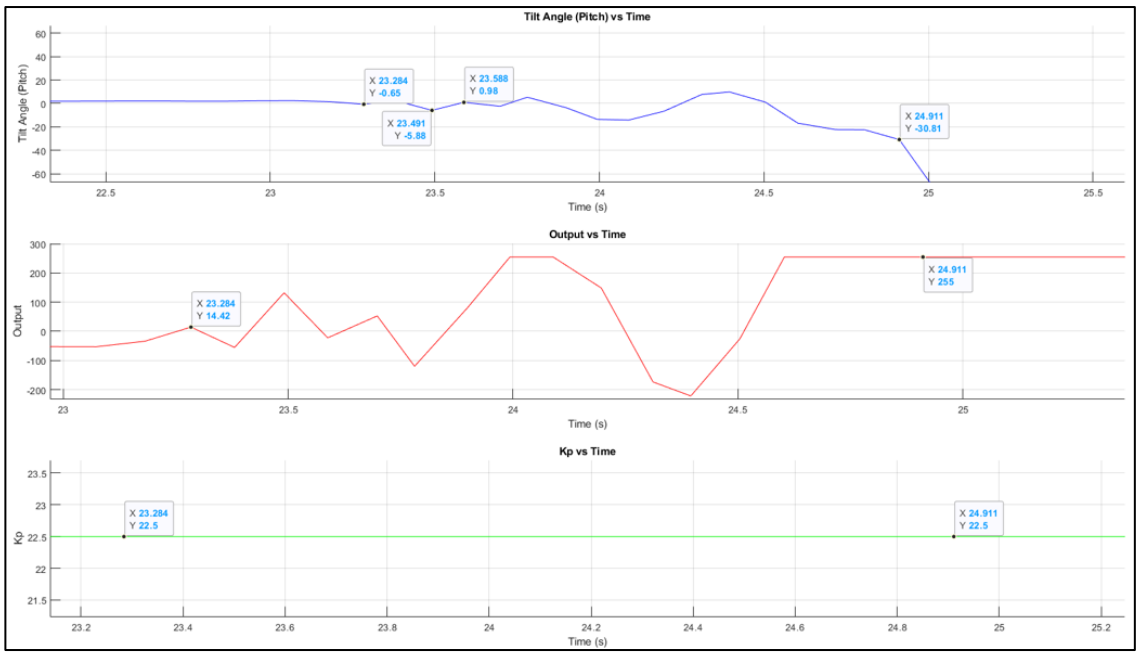


Figure 4.18: Behavioral Response when Kp = 22.5

In conclusion, varying the Kp values has a significant impact on the robot's behavior during its balancing process. This analysis has shown that different Kp values result in distinct rise times and stable times. The choice of Kp value plays a crucial role in determining how quickly the robot can regain balance and maintain stability.

4.4.1.1.2 Comparative Table

Referring to the previously shown real-time data of the robot's behavioral response under varying Kp values, a comparative table is generated to facilitate the comparison and selection of the best performance based on the conducted analysis. Table 4.1 shows a comparative table of robot behavior under varying Kp values. (Assume Ki = 0 and Kd = 0). In Table 4.1, the Kp values are tested, and the corresponding results show a clear trade-off between rise time and stable time. When analyzing the table, it becomes evident why Kp = 18.0 is considered to have the best performance, especially in terms of its rise time and stable time.

Table 4.1: Comparative Table of Robot Behavior under Varying Kp Values

Kp value	Stable Tilt Angle (°)	Rise Time (s)	Stable Time (s)
4.5	-0.54	0.194 (-23.95° to 2.95°)	0.928 (without disturbance)
9.0	-0.44	0.210 (-10.32° to 4.6°)	1.127 (without disturbance)
13.5	0.86	0.100 (-7.54° to 1.13°)	2.250 (without disturbance)
18.0	**-2.19**	**0.097** **(3.65° to -2.01°)**	**10.51** **(with disturbance)**
22.5	-0.65	0.097 (-5.88° to 0.98°)	1.627 (without disturbance)

Kp = 18.0 has one of the shortest rise times among the tested values, at just 0.097 seconds. A shorter rise time means that the robot responds rapidly to deviations from the desired tilt angle. Despite its rapid response, Kp = 18.0 also manages to achieve stability within 10.17 seconds, even in the presence of disturbances. A longer stable time indicates that the robot can sustain its equilibrium without showing any signs of oscillation.

In conclusion, Kp = 18.0 stands out which has the best performance in Table 4.1 due to its remarkable combination of a short rise time of 0.097 seconds and a prolonged stable time of 10.17 seconds, showcasing its ability to respond rapidly while maintaining long-term stability, even in the presence of disturbances. This combination of fast response and long-lasting stability shows that it works very well in the control system.

4.4.1.2 Kd Comparison Analysis

This sub-topic will involve an extensive analysis of Kd to facilitate comparison. This analysis consists of two key elements: graphical interpretation, entailing the creation of graphs utilizing MATLAB, and the formulation of a comparative table designed to systematically differentiate various parameter values for the purpose of comparison.

4.4.1.2.1 Graphical Interpretation

In this section, a visual analysis of the effects of different Kd values on the robot's behavior are discussed. Plotted graphs generated using MATLAB provide insights into how variations in Kd influence the robot's performance, shedding light on its response to disturbances and stability.

Overshoot is closely related to the damping ratio of the system. High overshoot values may indicate underdamped or oscillatory behavior, which can be an indication of instability. Besides, calculating settling time is crucial for assessing how quickly a dynamic system stabilizes around its desired state after a disturbance. Settling time quantifies the system's responsiveness and efficiency in reaching a steady-state condition, aiding in the evaluation and optimization of control mechanisms.

Figure 4.19 illustrates the Pitch angle vs. Time response with Kd set to 0.3 in the self-balancing system. The overshoot, calculated as 2425%, is derived from the formula in (2) where the peak value is the maximum pitch angle reached during the response (5.58º). The desired angle is -0.24º as labelled in the graph. The red-dotted line in Figure 4.23 signifies the steady-state error, representing the deviation between the desired and actual pitch angles after the response has settled. To determine the settling time, it is calculated by the time difference between the disturbance occurrence

(5.738s) and the point at which the system reached steady state (6.454s). The obtained settling time is 0.716 seconds. The Kd vs Time graph proves that Kd value is fixed to Kd = 0.3.

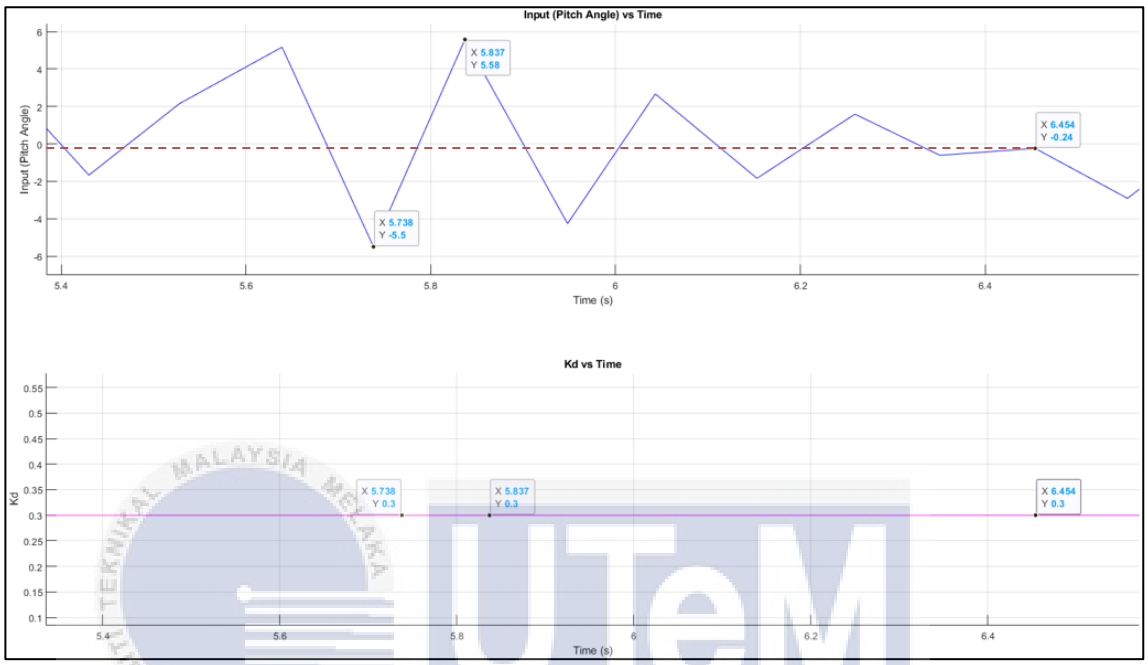


Figure 4.19: Behavioral Response when Kd = 0.3

Moreover, in Figure 4.20, the response of the self-balancing system to Pitch angle vs. Time with Kd set to 0.6 is presented. Calculating the overshoot, which stands at 314%, involves employing the formula in equation (2) with the peak value representing the maximum pitch angle (4.56º) attained during the system's response. For settling time determination, the time difference between the disturbance occurrence and the point where the system achieves stability, yielding a settling time of 1.641 seconds is computed. The time where the system's response is 38.078 seconds whereas the desired value is 39.719 seconds.

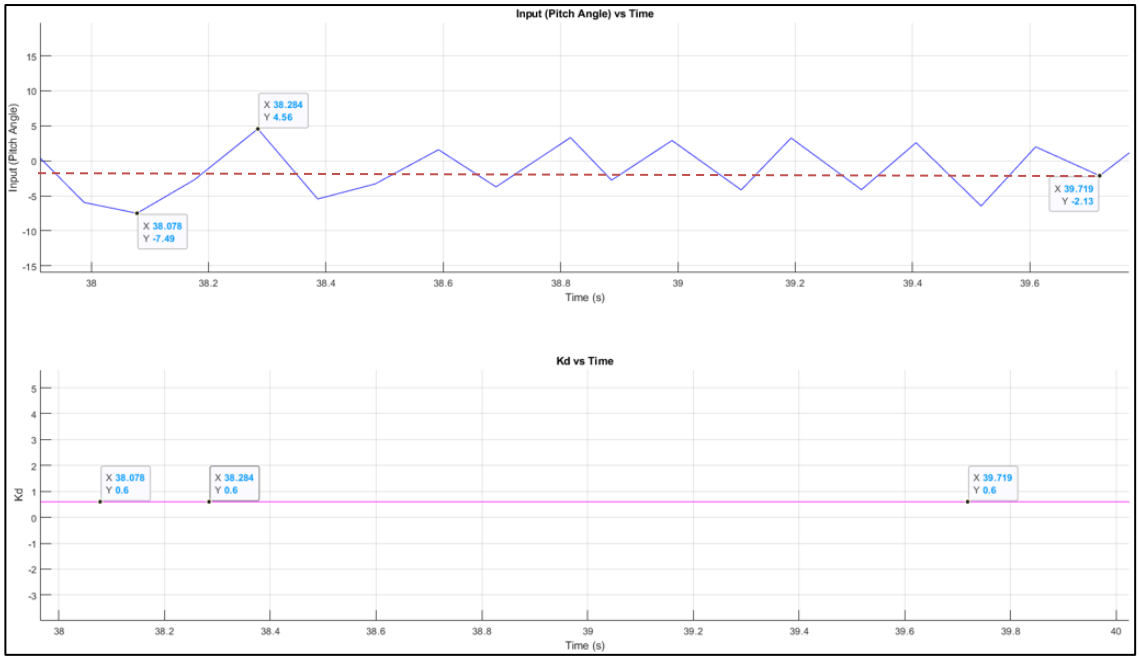


Figure 4.20: Behavioral Response when Kd = 0.6

In addition, as for Kd = 0.9, as shown in Figure 4.21, The calculated overshoot, now at an impressive 10881%, is determined using the formula mentioned in equation (2), where the peak value corresponds to the highest pitch angle achieved during the system's response. 0.36º is the desired value while 39.53º is the peak value as shown in the graph. The system with Kd = 0.9 has no settling time. This means that the system keeps responding to disturbances without eventually stabilizing or reaching a steady state. It indicates a continuous and ongoing reaction to changes without a defined point of rest.

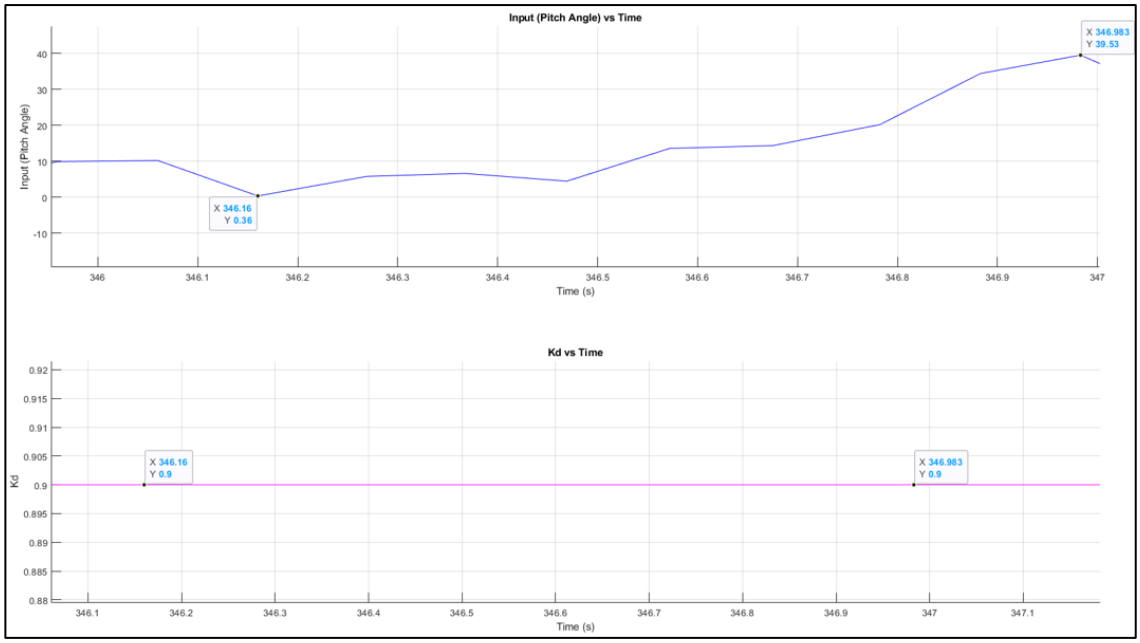


Figure 4.21: Behavioral Response when Kd = 0.9

Thus, altering the Kd values has an influence on the robot's behavior throughout its balancing procedure. This examination illustrates that different Kd values lead to varied overshoots and settling times. The selection of the Kd value is pivotal in shaping how rapidly the robot can recover its balance and sustain stability.

4.4.1.2.2 Comparative Table

Referring to the previously displayed real-time data showcasing the robot's behavioral response under varying Kd values, a comparative table, identified as Table 4.2, has been generated to simplify the comparison and facilitate the selection of the best performance based on the conducted analysis. (Assume Kp = 18 and Ki = 0).

Table 4.2: Comparative Table of Robot Behavior under Varying Kd Values

Kd	Overshoot (%)	Settling Time (s)
0.3	2425	0.716
0.6	**314**	**1.641**
0.9	10881	None

Therefore, by referring to the analysis from the previous sub-topic, it is evident that Kd = 0.6 exhibits the best performance among the three values. In Figure 4.20, where Kd = 0.6, the calculated overshoot is 314%. Comparatively, in Figure 4.19 (Kd = 0.3), the overshoot is 2425%, and in Figure 4.21 (Kd = 0.9), the overshoot is an impressive 10881%. Lower overshoot values are generally desirable as they indicate less oscillation and a smoother response. In this context, Kd = 0.6 demonstrates a more controlled and stable response compared to both Kd = 0.3 and Kd = 0.9.

Additionally, settling time is another critical parameter for evaluating system performance. In Figure 4.24 (Kd = 0.6), the settling time is 1.641 seconds. In Figure 4.19 (Kd = 0.3), the settling time is 0.716 seconds, and for Kd = 0.9 (Figure 4.21), there is no settling time determined, indicating continuous response without stabilization. Ideally, a shorter settling time is desirable as it signifies a quicker recovery to a steady-state condition. However, a system with no settling time, as in Kd = 0.9, indicates continuous oscillation without achieving stability.

To sum up, Kd = 0.6 emerges as the optimal choice as it demonstrates a controlled response with a relatively low overshoot and a reasonable settling time, offering a balanced and stable performance for the self-balancing robot.

4.4.1.3 Ki Comparison Analysis

This sub-topic will encompass a thorough examination of Ki to facilitate comparison. The analysis will comprise two essential components: the graphical interpretation, involving the generation of graphs using MATLAB, and the development of a comparative table designed to systematically distinguish different parameter values for the purpose of Ki comparison.

4.4.1.3.1 Graphical Interpretation

This section examines the impact of different Ki values on robot behavior using MATLAB-generated graphs. Figure 4.22 shows the behavioral response of robot when Ki = 35. The input (pitch angle) vs Time graph shows a time series data of some oscillations in pitch angle. The setpoint vs Time graph displays a constant setpoint value over the same time period. This indicates that the setpoint does not change. From the observations made, when Ki = 35, there has no overshoot and settling time since the robot does not recover from the imbalances.

Then, the average steady state is calculated by taking all the points from the jittery state of the robot which resulting in 1.67º value. The steady state error is calculated by finding the differences between the average steady state and the setpoint. In this case, the average steady state is similar to the steady state error as the setpoint value is 0.

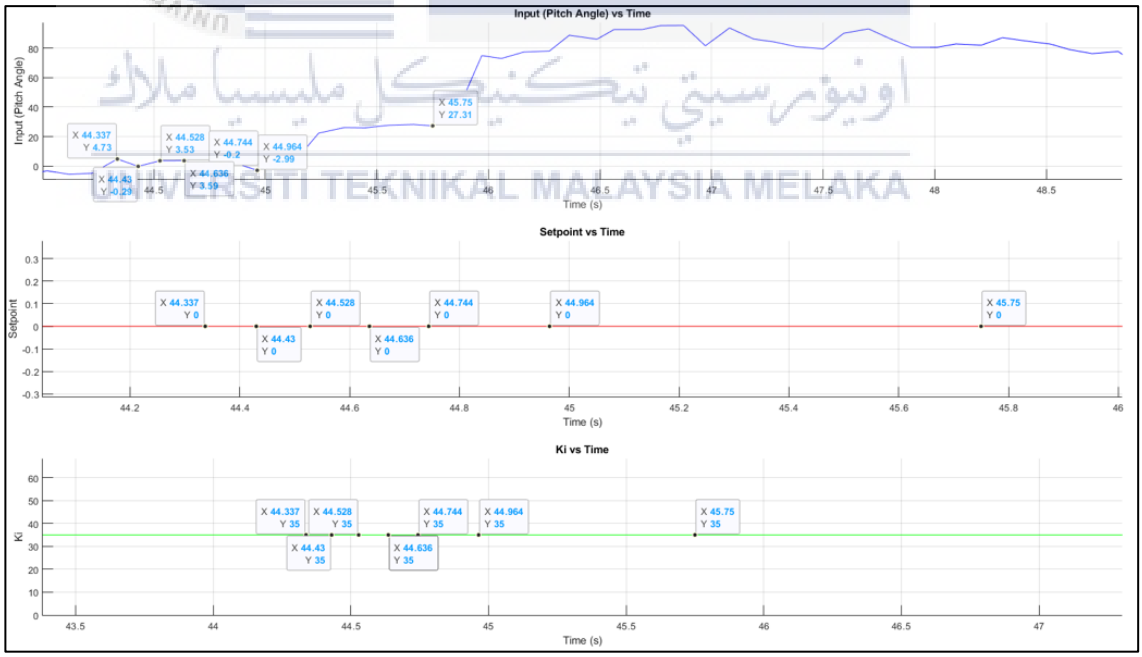


Figure 4.22: Behavioral Response when Ki = 35

As for Ki = 70, there is overshoot and settling time measured as the robot strives to regain balances resulting in 1092% overshoot and 2.041 seconds setting time. The input (pitch angle) vs. time graph presented in Figure 4.23 further elucidates the system's resilience, showcasing that the robot manages to maintain stability despite the presence of disturbances. From the graph, a steady state towards the end indicates that the self-balancing robot has effectively stabilized and reached a point where its pitch angle remains relatively constant over time. The average steady state when Ki = 70 is calculated to 1.92º.

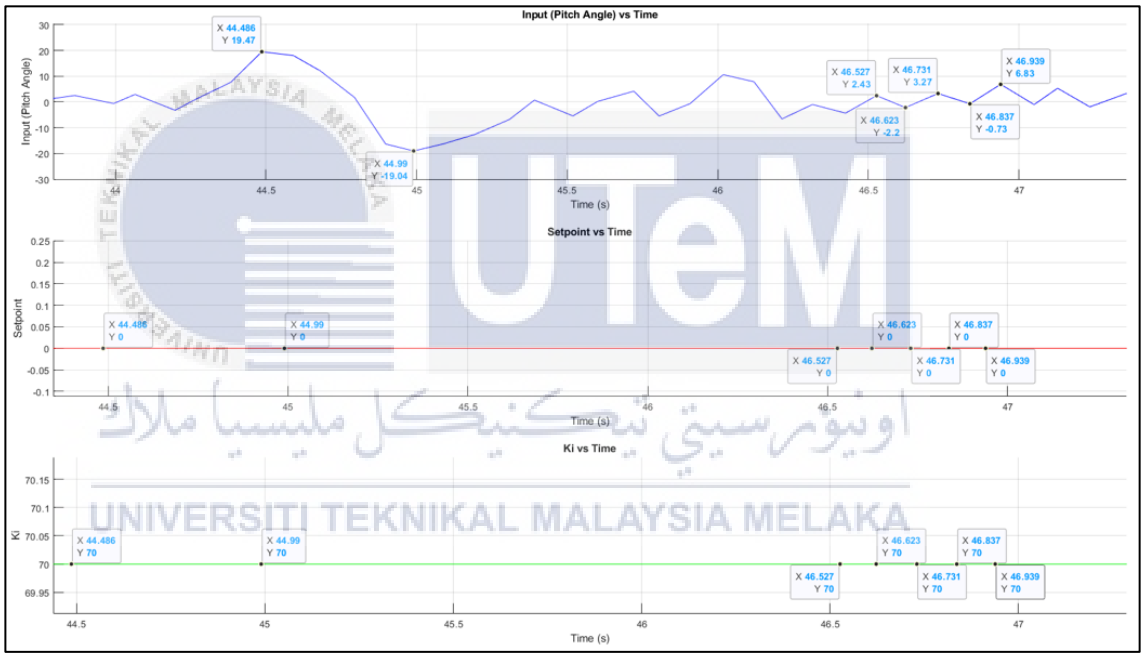


Figure 4.23: Behavioral Response when Ki = 70

Next, Figure 4.24 illustrates the behavioral response when Ki is set to Ki = 105. The graph has an overshoot of 14516% which is quite high and there has no settling time recorded. Since the value of Ki is set too aggressively, it can lead to overshooting and difficulties in reaching a stable state. The instability of the robot can be seen as there has been an absence of settling time. This is because, if the control parameters are not properly tuned, the response may not converge to a steady state, leading to

continuous oscillations. Since the system may be constantly adjusting and readjusting, preventing it from settling completely, there are continuous disturbances on the system where it might not fully reach a stable state. Thus, the calculated value of the average steady state is -1.33º.

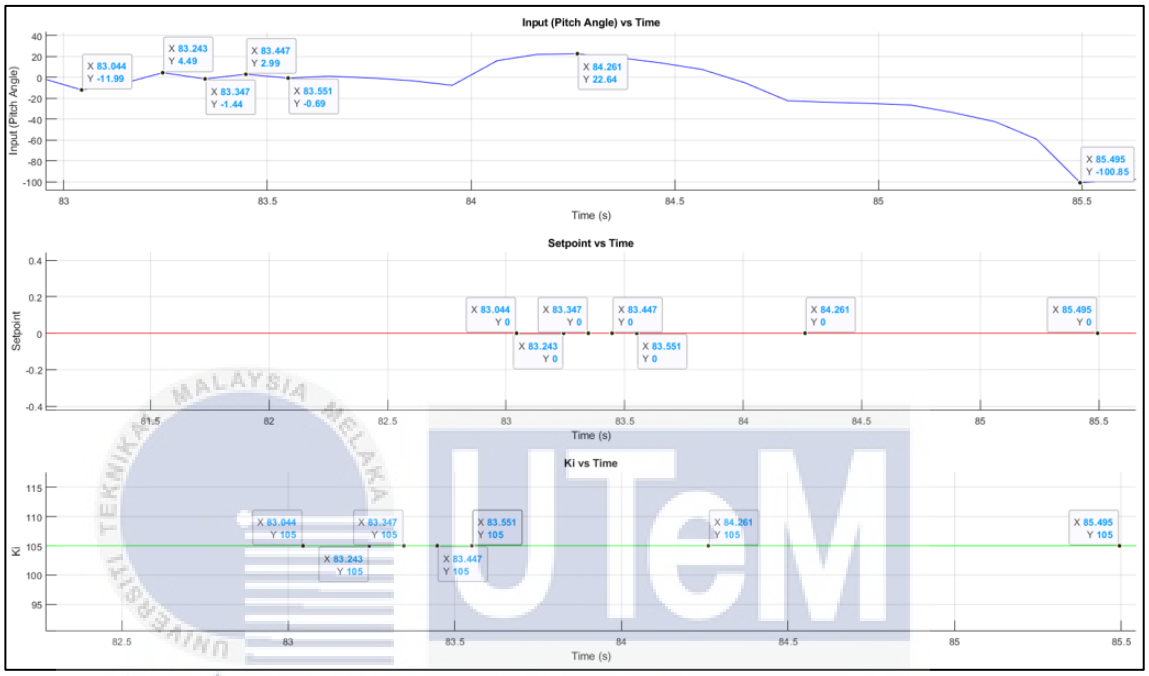


Figure 4.24: Behavioral Response when Ki = 105

To conclude, modifying the Ki values significantly affects the self-balancing robot's performance during its stabilization process. Thus, careful selection of Ki values is crucial for achieving optimal and reliable performance in the system.

4.4.1.3.2 Comparative Table

Table 4.3 shows a comparative table which simplifies the comparison and assists in choosing the most favorable performance based on the analysis conducted with various Ki values. (Assume Kp = 18, Kd = 0.6 and Setpoint = 0).

Table 4.3: Comparative Table of Robot Behavior under Varying Ki Values

Ki	Overshoot (%)	Settling Time (s)	Average Steady-state (°)	Steady State Error
35	Not Recover	Not Recover	1.670	1.670
70	**1092**	**2.246**	**1.920**	**1.920**
105	14516	Not Recover	-1.330	1.330

From Table 4.3, it appears that Ki = 70 exhibits a relatively balanced and effective performance compared to Ki = 35 and Ki = 105. At Ki = 70, there is an overshoot of 1092% and a settling time of 2.041 seconds. This indicates that the system responds to disturbances with a notable overshoot but eventually settles within a reasonable time frame. In addition, the input (pitch angle) vs. time graph in Figure 4.23 demonstrates the system's resilience, showcasing that the robot manages to maintain stability despite disturbances. This resilience is indicative of a well-balanced control system.

Furthermore, the presence of a steady state towards the end of the response, coupled with an average steady state of 1.92°, suggests that the system stabilizes effectively. The steadiness in the pitch angle over time indicates a controlled and stable behavior. The combination of overshoot, settling time, and the absence of continuous disturbances at Ki = 70 suggests a balanced response. It neither overshoots excessively nor exhibits prolonged settling times, indicating a well-tuned and stable control system.

To sum up, Ki = 70 appears to offer a good trade-off between overshoot, settling time, and system stability. It strikes a balance in responsiveness without introducing excessive oscillations or settling difficulties. The careful selection of Ki values is

indeed crucial, and in this case, Ki = 70 demonstrates a performance that aligns well with achieving optimal and reliable self-balancing in the robot.

4.4.2 Performance with Multiple Disturbance Evaluation

As for this section, the self-balancing robot is tested with multiple disturbances to evaluate the performance of the control system under varying conditions. The introduction of diverse disturbances provides insights into the system's ability to respond dynamically and maintain balance in the face of external influences. Figure 4.25 illustrates the tilt angle (pitch) vs time of the self-balancing robot. The robot was tested for its performance when multiple disturbances were applied to it. Even though several disturbances were applied, the robot still managed to stay upright, indicating that it has good stability and balance control.

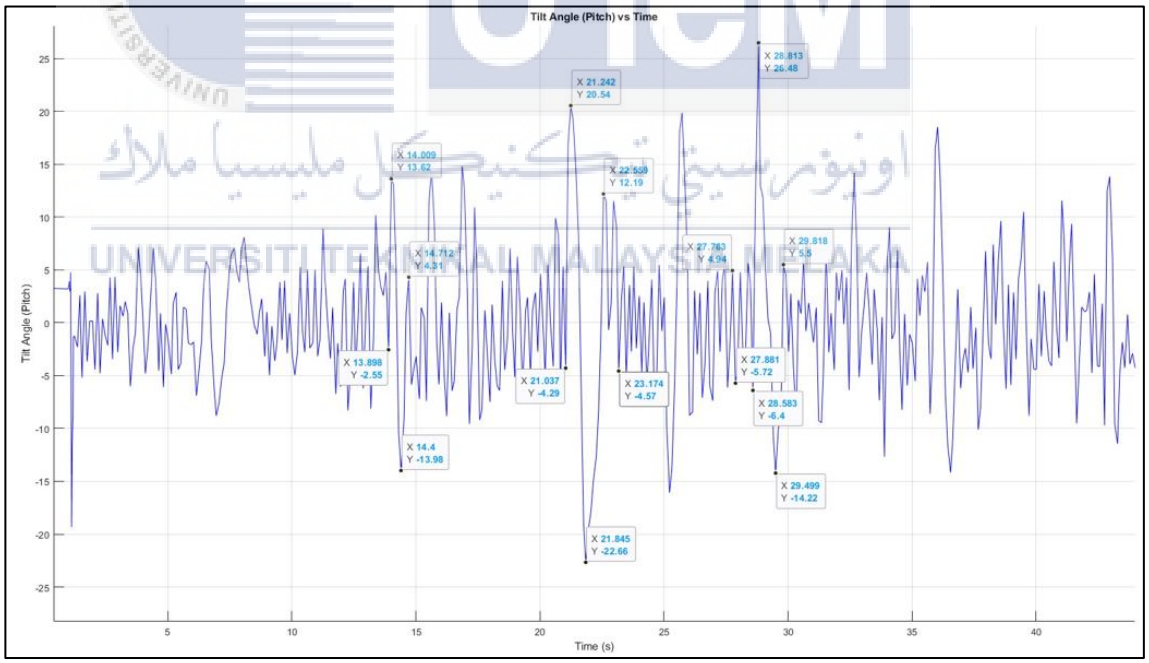


Figure 4.25: Overall Performance of the Robot with Multiple Disturbance

From the graph, it can be seen that the values of pitch angle are mostly near to 0°. This indicates that the self-balancing robot consistently maintains an upright position

and balanced orientation. In the context of a self-balancing robot, the pitch angle represents the deviation from the vertical, with 0º corresponding to the perfectly upright position. The stability of the robot is achieved through the control system's ability to detect and respond to disturbances, continually adjusting the motor inputs to counteract any deviations from the desired orientation.

The effectiveness of the PID controller, with tuned gains (Kp = 18, Ki = 70, Kd = 0.6), contributes to the system's stability. The proportional, integral, and derivative components work together to respond appropriately to changes in the pitch angle, allowing the robot to correct for disturbances swiftly and accurately. As a result, the pitch angle remains close to 0º, indicating that the self-balancing robot maintains stability by actively counteracting external forces and disturbances, providing a robust and reliable performance.

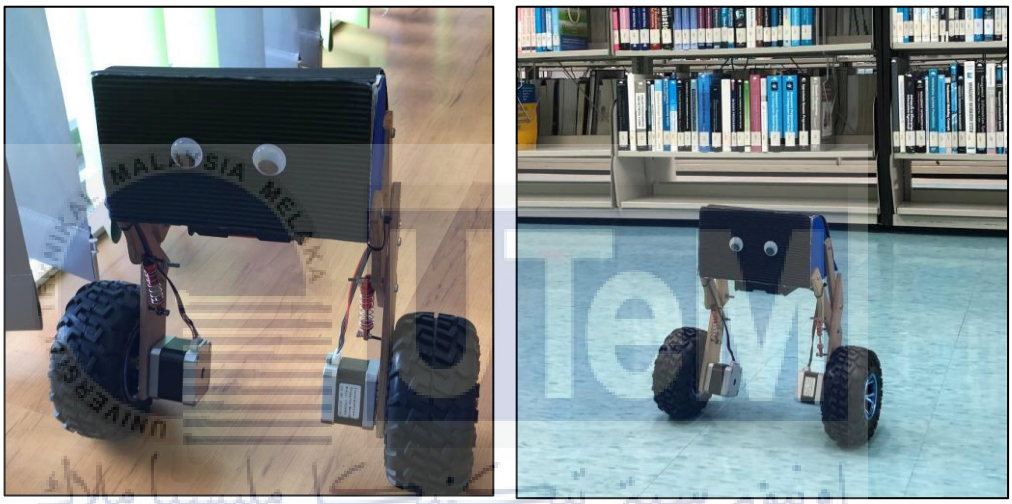
Table 4.4 shows the parameters that are calculated based on the data in Figure 4.25. To assess a robot's performance and stability, key parameters include a shorter rise time for fast response, lower overshoot for controlled reaction, shorter settling time for quick stabilization, smaller average steady-state value for consistent positioning, and lower steady-state error for precise maintenance of the desired position. These factors collectively determine the robot's efficiency and control effectiveness.

Table 4.4: Parameters of the overall performance of the robot

Kp	Ki	Kd	Rise Time (s)	Overshoot (%)	Settling Time (s)	Average Steady State	Steady State Error
18	70	0.6	0.391	225.9	1.932	24.96	24.96

4.4.3 Prototype Performance Testing

This section details testing the prototype to verify it meets performance criteria, mainly balancing and withstanding disturbances. Tests were conducted on various surfaces, as shown in Figures 4.26 (a), (b) and (c), representing wooden, rubber, and rough surfaces, respectively. Despite these differences, the robot consistently maintained balance without tipping over, demonstrating that it can effectively balance itself on diverse surfaces with the same tuning parameters.



(a) (b)

(c)

Figure 4.26: (a) Prototype testing on wooden surface, (b) Prototype testing on rubber surface, (c) Prototype testing on rough surface

After conducting numerous tests, it was observed that the robot demonstrated remarkable stability across various types of surfaces including flat terrain, uneven surfaces and inclines as shown in Figure 4.27 (a) and (b).

(a) (b)

Figure 4.27: (a) Testing on uneven surface, (b) Testing on inclined surfaces

The prototype performance testing also incorporates disturbance scenarios to verify the robot's effectiveness in real-world conditions with external factors. This thorough testing identifies potential design and algorithm weaknesses, enhancing the robot's performance and reliability in dynamic environments. Figure 4.28 (a) shows the presence of disturbance during the prototype performance testing to simulate real-world conditions. Then, the robot is subjected to a physical force simulating disturbance as shown in Figure 4.28 (b).

Then, the PID controller responds to the imbalance as shown in Figure 4.28 (c). The PID controller adjusts the robot's actuators to counteract the external force and bring the system back to a stable state. After the disturbance, and with the corrective actions taken by the PID controller, the robot ideally returns to its original state which maintains stability and balance as illustrated in Figure 4.28 (d).

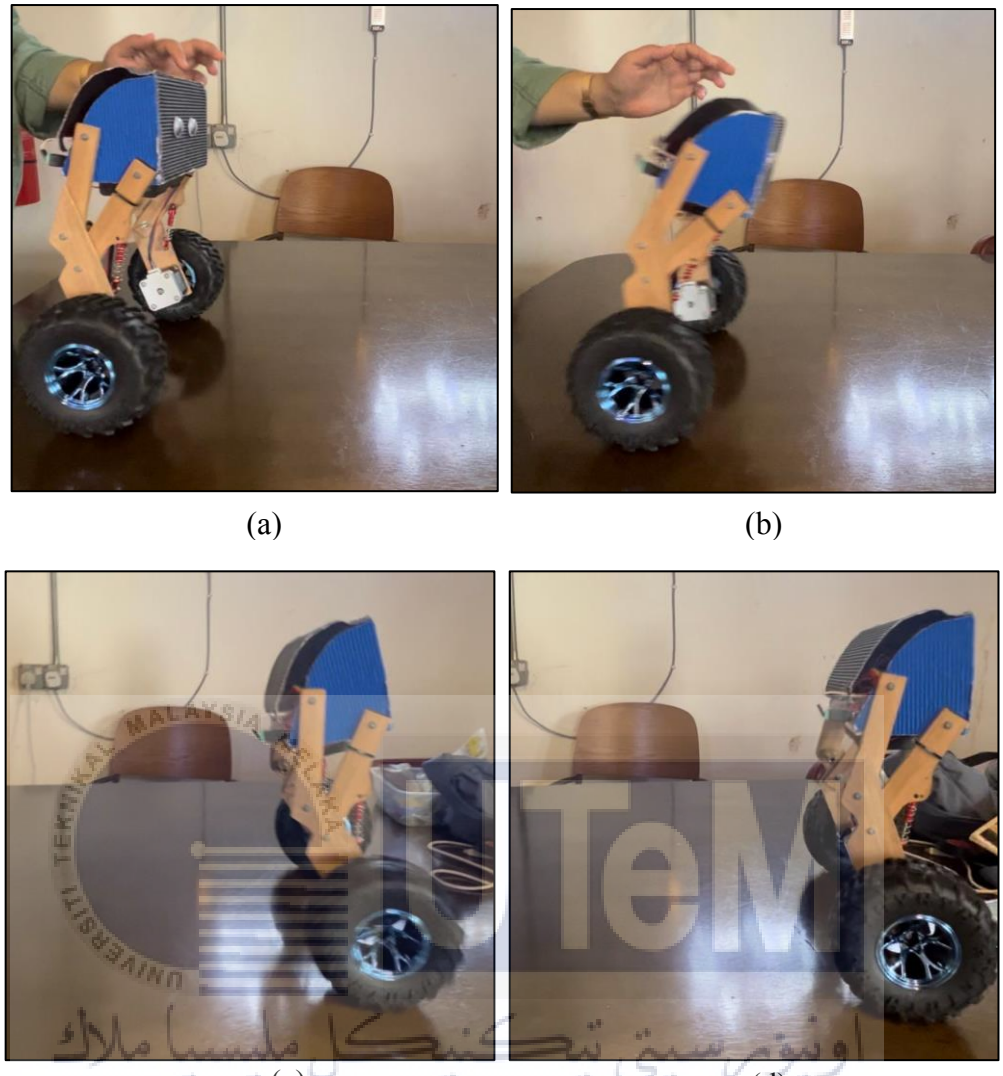


Figure 4.28: (a) Presence of disturbance, (b) Robot is subjected to a physical force, (c) Robot is trying to recover the imbalance, (d) Robot back at its original state

Therefore, this outcome demonstrates the effectiveness of the control algorithm and the tuning parameters in quickly and accurately responding to disturbances.

4.5 Chapter Summary

Thus, this chapter covers the implementation of the proposed system, detailing the software and hardware setup, features, interfaces, and concludes with analytical results and performance analysis for PID parameter tuning.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 Chapter Overview

This chapter explains the project's contributions to society with a conclusion which includes the objectives of the project. The problems which occur during conducting the project with its limitations will also be explained in order to recommend solutions which can enhance the system's performance in the future.

5.2 Project Achievement

This development of a two-wheel self-balancing robot using PID controller was created for the Final Year Project as a proof of concept (POC) of this system implementation for research in PID controller. This project begins with the development of hardware which uses ESP32 as the microcontroller, MPU6050 as the

sensor and NEMA17 stepper motor as well as a4988 stepper motor driver. This is to investigate the tuning parameters of PID gains affect the performance of the robot.

In addition, Blynk application is used as an IoT platform that could control the PID gains in order to tune the parameters to achieve the best performance. Furthermore, MATLAB software is used to produce the analysis of the tuning parameters which produce the Rise Time, Settling Time and Overshoot as well as Steady State Error. The range of PID gains varied and its performance was recorded.

5.3 Project Problem and Limitation

During the development and implementation phase of this project, several issues and limitations were encountered. Firstly, the chosen MPU6050 sensor is sensitive to external vibrations, leading to potential noise in sensor readings. Proper placement of the sensor at the center of gravity (CoG) is crucial for stable robot performance. The center of gravity is the point where the entire weight of the robot can be considered to act, and it plays a crucial role in maintaining balance.

Secondly, the 12V lithium-ion battery initially used to power the hardware fell short in terms of battery life. It required frequent recharging, taking about 2-3 hours for a full charge before reuse. An alternative, a 12V DC rechargeable polymer lithium-ion battery, offered longer-lasting power but added weight (approximately 0.35kg), impacting the robot's weight distribution.

5.4 Future Work and Recommendations

This study is suggested for inclusion in the Control Principle and Systems course at FTKEK. While the existing syllabus covers the PID controller, it currently lacks information on the crucial PID tuning method. Besides, here are further

recommendations to enhance the self-balancing robot, particularly in its control aspects:

i. Suggest a novel control method allowing remote management of the robot's movements using Blynk. This may include adding control interfaces that users could customize via the Blynk app, enabling them to change things like direction, speed, and even preset moves.

ii. Integrate a camera onto the robot, enabling users to monitor its motions and whereabouts. By adding this, the robot's capabilities may be increased, and users will be able to see a live video stream via the Blynk app. This capability improves the robot's usefulness for surveillance to enabling users to see their surroundings.

5.5 Conclusion

In a nutshell, it can be concluded that this final year project took around two semesters worth of time to complete, starting from the proposal of the system, development, and documentation. After evaluation, it can be concluded that the prototype for this project is considered successful. Although there are some limitations in this system, several suggestions are provided for the system quality enhancement. All the objectives are achieved using the required hardware and software mentioned in Chapter 3. There are several improvements that can be made for this project to provide a more efficient way for the users to use the system.

5.6 Chapter Summary

Thus, this chapter summarizes the whole project which includes the achievements, problem and limitations, future works and commercialization relevancy of this project. All the objectives are achieved using the methodology mentioned in Chapter 3.

REFERENCES

[1] B. K. Bhawmick, "DESIGN AND IMPLEMENTATION OF A SELF-BALANCING ROBOT Review study on different dental implant materials and their characterization based on their behavior in different condition of oral saliva View project DESIGN AND IMPLEMENTATION OF A SELF-BALANCING ROBOT View project." [Online]. Available: https://www.researchgate.net/publication/323258475

[2] O. A. Choudhry, M. Wasim, A. Ali, M. A. Choudhry, and J. Iqbal, "Modelling and robust controller design for an underactuated self-balancing robot with uncertain parameter estimation," *PLoS One*, vol. 18, no. 8 August, Aug. 2023, doi: 10.1371/journal.pone.0285495.

[3] I. Mateșică, M. Nicolae, L. Bărbulescu, and A.-M. Mărgerușeanu, "Self-balancing robot implementing the inverted pendulum concept."

[4] B. Subudhi and S. Ghosh, "Adaptive Iterative Learning Control of a Single-Link Flexible Manipulator Based on an Identified Adaptive NARX Model Master of Technology in Control and Automation."

[5] E. P. Kunnel, D. Johny, A. Jacob, and E. Paul, "International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering Controlling of Two Wheeled Self Balancing Robot using PID," 2018, doi: 10.15662/IJAREEIE.2018.0703046.

[6] Y. Gong, X. Wu, and H. Ma, "Research on Control Strategy of Two-Wheeled Self-Balancing Robot," in *Proceedings - 2015 International Conference on Computer Science and Mechanical Automation, CSMA 2015*, 2016. doi: 10.1109/CSMA.2015.63.

[7] R. Sadeghian and M. T. Masoule, "An experimental study on the PID and Fuzzy-PID controllers on a designed two-wheeled self-balancing autonomous robot," in *2016 4th International Conference on Control, Instrumentation, and Automation, ICCIA 2016*, 2016. doi: 10.1109/ICCIAutom.2016.7483180.

[8] M. A. Johnson *et al.*, *PID control: New identification and design methods*. 2005. doi: 10.1007/1-84628-148-2.

[9] S. M. Peash *et al.*, "A Smart Approach to Control a Two-Wheeled Self Balancing Robot Using a PID Controller with Two Degree of Freedom." [Online]. Available: https://www.researchgate.net/publication/367510694

[10] O. M. Mohamed Gad, S. Z. M. Saleh, M. A. Bulbul, and S. Khadraoui, "Design and Control of Two Wheeled Self Balancing Robot (TWSBR)," in *2022 Advances in Science and Engineering Technology International Conferences, ASET 2022*, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/ASET53988.2022.9735004.

[11] V. A. Maheshbhai, D. Kumar, and R. Sinha, “Development of Two Wheeled Robot (TWR) by Single Stepper Driver using PID controller.”

[12] H. M. Tun, M. S. Nwe, Z. M. Naing, M. M. Latt, D. Pradhan, and P. K. Sahu, “Research on Self-balancing Two Wheels Mobile Robot Control System Analysis,” *Electrical Science & Engineering*, vol. 4, no. 1, p. 7, Apr. 2022, doi: 10.30564/ese.v4i1.4398.

[13] J. Zhao, J. Li, and J. Zhou, “Research on Two-Round Self-Balancing Robot SLAM Based on the Gmapping Algorithm,” *Sensors*, vol. 23, no. 5, Mar. 2023, doi: 10.3390/s23052489.

[14] C. Iwendi, M. A. Alqarni, J. H. Anajemba, A. S. Alfakeeh, Z. Zhang, and A. K. Bashir, “Robust Navigational Control of a Two-Wheeled Self-Balancing Robot in a Sensed Environment,” *IEEE Access*, vol. 7, pp. 82337–82348, 2019, doi: 10.1109/ACCESS.2019.2923916.

[15] S. Erfan Arefin, “Simple Two-wheel Self-Balancing Robot Implementation.”

[16] I. Iswanto, A. Ma’arif, N. Maharani Raharja, T. K. Hariadi, and M. A. Shomad, “Using a Combination of PID Control and Kalman Filter to Design of IoT-based Telepresence Self-balancing Robots during COVID-19 Pandemic,” *Emerging Science Journal*, vol. 4, no. Special issue, pp. 241–261, 2020, doi: 10.28991/esj-2021-SP1-016.

[17] S. K. Vishwakarma, P. Upadhyaya, B. Kumari, and A. K. Mishra, “Smart Energy Efficient Home Automation System Using IoT,” in *Proceedings - 2019*

4th International Conference on Internet of Things: Smart Innovation and Usages, IoT-SIU 2019, 2019. doi: 10.1109/IoT-SIU.2019.8777607.

[18] S. Kumar, P. Tiwari, and M. Zymbler, "Internet of Things is a revolutionary approach for future technology enhancement: a review," *J Big Data*, vol. 6, no. 1, 2019, doi: 10.1186/s40537-019-0268-2.

[19] A. J. Moshayedi *et al.*, "Simulation and Validation of Optimized PID Controller in AGV (Automated Guided Vehicles) Model Using PSO and BAS Algorithms," *Comput Intell Neurosci*, vol. 2022, p. 7799654, 2022, doi: 10.1155/2022/7799654.

[20] L.-F. Bărbulescu, C. and E. Universitatea din Craiova. Faculty of Automation, Institute of Electrical and Electronics Engineers, and IEEE Control Systems Society, *2020 24th International Conference on System Theory, Control and Computing (ICSTCC) : proceedings : October 8-10, 2020, Sinaia, Romania*.

[21] P. Xie, W. Wu, and Y. Zhang, "Design of human-computer interaction gesture operation based on STM32 and MPU6050," in *2022 IEEE 4th International Conference on Power, Intelligent Computing and Systems, ICPICS 2022*, 2022. doi: 10.1109/ICPICS55264.2022.9873595.

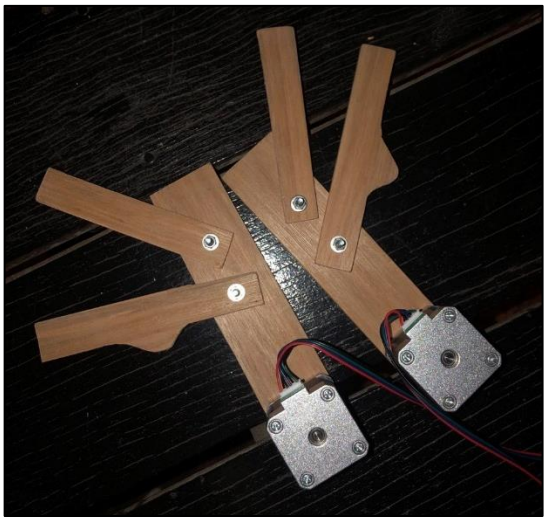
[22] G. S. Krishna, D. Sumith, and G. Akshay, "Epersist: A Two-Wheeled Self Balancing Robot Using PID Controller And Deep Reinforcement Learning," in *International Conference on Control, Automation and Systems*, 2022. doi: 10.23919/ICCAS55662.2022.10003940.

[23] B. Sergey, “Ultrasonic navigation to control the movement of a mobile robot,” in *Moscow Workshop on Electronic and Networking Technologies, MWENT 2020 - Proceedings*, 2020. doi: 10.1109/MWENT47943.2020.9067482.

[24] S. D. Yusuf, S.-L. D. Comfort, I. Umar, and A. Z. Loko, “Simulation and Construction of a Solar Powered Smart Irrigation System Using Internet of Things (IoT), Blynk Mobile App,” *Asian Journal of Agricultural and Horticultural Research*, pp. 136–147, Oct. 2022, doi: 10.9734/ajahr/2022/v9i4202.

APPENDICES

APPENDIX A: Process of making the joints of the self-balancing robot



APPENDIX B: Coding in Arduino IDE

```
// Blynk and WiFi settings
#define BLYNK_TEMPLATE_ID "TMPL6_NeSrwZR"
#define BLYNK_TEMPLATE_NAME "Self Balancing Robot"
#define BLYNK_AUTH_TOKEN "YL3I1G5pwTiagFxBlMnTKt4rdY0kmbwl"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <Wire.h>
#include <MPU6050_tockn.h>
#include <BlynkSimpleEsp32.h>
#include <PID_v1.h>
#include <FastAccelStepper.h>
MPU6050 mpu6050(Wire);
double Setpoint = 0; //Setpoint for balancing
double Input, Output;
double Kp, Ki, Kd, SF, SFREV, SFP, SFFOR, acceleration, deadZone,
Inputoffset;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
// Motor control pins
int leftStepPin = 32;
int leftDirPin = 33;
int rightStepPin = 14;
int rightDirPin = 12;
int enablePin = 19;
// FastAccelStepper Engine
FastAccelStepperEngine engine = FastAccelStepperEngine();
FastAccelStepper *leftMotor = NULL;
FastAccelStepper *rightMotor = NULL;
double alpha;
float filteredPitch = 0;
unsigned long lastUpdateTime = 0;
BLYNK_WRITE(V9) {
Kp = param.asDouble();
}
BLYNK_WRITE(V10) {
Ki = param.asDouble();
}
BLYNK_WRITE(V11) {
Kd = param.asDouble();
}
BLYNK_WRITE(V12) {
SF = param.asDouble();
}
BLYNK_WRITE(V13) {
Inputoffset = param.asDouble();
}
BLYNK_WRITE(V14) {
SFP = param.asDouble();
}
BLYNK_WRITE(V15) {
    SFREV = param.asDouble();
```

```
}
BLYNK_WRITE(V16) {
SFFOR = param.asDouble();
}
BLYNK_WRITE(V17) {
acceleration = param.asDouble();
}
BLYNK_WRITE(V18) {
deadZone = param.asDouble();
}
void setup() {
Serial.begin(9600);
Blynk.begin(auth, ssid, pass);
Wire.begin();
mpu6050.begin();
mpu6050.calcGyroOffsets(true);
engine.init();
leftMotor = engine.stepperConnectToPin(leftStepPin);
rightMotor = engine.stepperConnectToPin(rightStepPin);
if (leftMotor) {
leftMotor->setDirectionPin(leftDirPin);
leftMotor->setEnablePin(enablePin);
leftMotor->setAutoEnable(true);
leftMotor->setSpeedInHz(2000); // Speed in Hz
}
if (rightMotor) {
rightMotor->setDirectionPin(rightDirPin);
rightMotor->setEnablePin(enablePin);
rightMotor->setAutoEnable(true);
rightMotor->setSpeedInHz(2000); // Speed in Hz
}
myPID.SetOutputLimits(-255, 255);
myPID.SetSampleTime(1);
myPID.SetMode(AUTOMATIC);
}
void loop() {
Blynk.run();
mpu6050.update();
Input = mpu6050.getAngleY() - Inputoffset;
myPID.SetTunings(Kp, Ki, Kd);
myPID.Compute();
float motorSpeed = abs(Output * SF) + SFP; // Scale factor for speed
float motorSpeedFor = motorSpeed + SFFOR; // Scale factor for speed
float motorSpeedRev = motorSpeed + SFREV; // Scale factor for speed
if (leftMotor && rightMotor) {
leftMotor->setAcceleration(acceleration * 16);
rightMotor->setAcceleration(acceleration * 16);
if (Output > deadZone) {
// Output positive: left motor counterclockwise, right motor
clockwise
leftMotor->setSpeedInHz(motorSpeedFor); // Scale Output
rightMotor->setSpeedInHz(motorSpeedFor); // Scale Output
leftMotor->move(3 * 16);
rightMotor->move(3 * 16);
```