

TWO WHEEL SELF-BALANCING WITH DYNAMIC STABILIZING



UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2021



UNIVERSITI TEKNIKAL MALAYSIA MELAKA

TWO WHEEL SELF-BALANCING WITH DYNAMIC STABILIZING

This report is submitted in accordance with the requirement of the Universiti Teknikal Malaysia Melaka (UTeM) for the Bachelor of Electronics Engineering Technology with Honours.



AINA FAKHIRA BINTI AHMAD

FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING
TECHNOLOGY

2021

BORANG PENGESAHAN STATUS LAPORAN PROJEK SARJANA MUDA

Tajuk: TWO WHEEL SELF-BALANCING WITH DYNAMIC STABILIZING

Sesi Pengajian: 2019

Saya **AINA FAKHIRA BINTI AHMAD** mengaku membenarkan Laporan PSM ini disimpan di Perpustakaan Universiti Teknikal Malaysia Melaka (UTeM) dengan syarat-syarat kegunaan seperti berikut:

1. Laporan PSM adalah hak milik Universiti Teknikal Malaysia Melaka dan penulis.
2. Perpustakaan Universiti Teknikal Malaysia Melaka dibenarkan membuat salinan untuk tujuan pengajian sahaja dengan izin penulis.
3. Perpustakaan dibenarkan membuat salinan laporan PSM ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. **Sila tandakan (X)

SULIT*

Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia sebagaimana yang termaktub dalam AKTA RAHSIA RASMI 1972.

TERHAD* Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan.

TIDAK TERHAD

Yang benar,

Disahkan oleh penyelia:



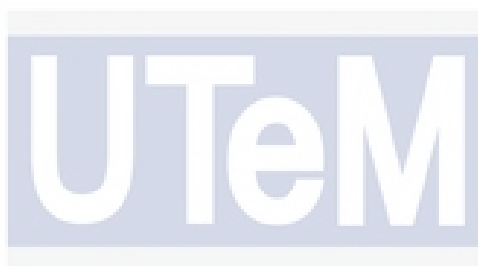
.....
AINA FAKHIRA BINTI AHMAD

.....

IZADORA BINTI MUSTAFFA

Alamat Tetap:

Cop Rasmi Penyelia



اونيورسيتي تيكنيكل مليسيا ملاك

Tarikh: 13/2/2021

Tarikh:

*Jika Laporan PSM ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali sebab dan tempoh laporan PSM ini

DECLARATION

I hereby, declared this report entitled TWO WHEEL SELF-BALANCING WITH DYNAMIC STABILIZING is the results of my own research except as cited in references.



Signature:

Author : AINA FAKHIRA BINTI AHMAD

Date: 13/2/2021



اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

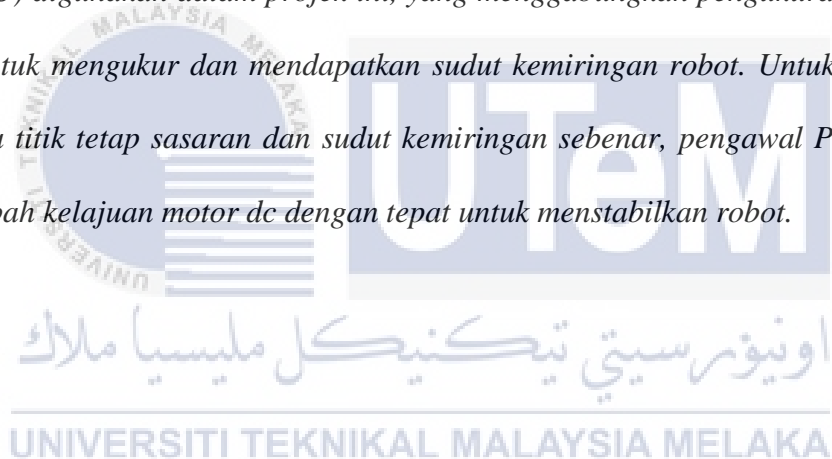
APPROVAL

This report is submitted to the Faculty of Mechanical and Manufacturing Engineering Technology of Universiti Teknikal Malaysia Melaka (UTeM) as a partial fulfilment of the requirements for the degree of Bachelor of Electronics Engineering Technology with Honours. The member of the supervisory is as follow:



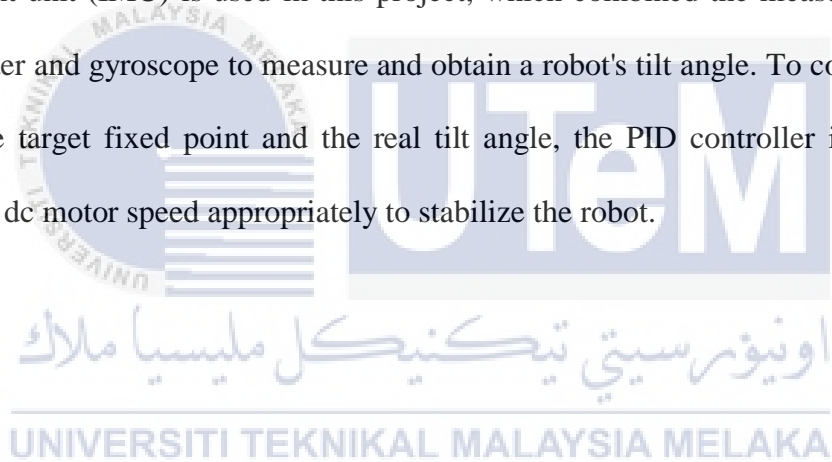
ABSTRAK

Robot pengimbang diri dua roda stabil secara dinamik, tetapi tidak stabil secara statik, berdasarkan sistem pendulum terbalik. Pelbagai teori mekanik dan kawalan terlibat dalam robot. Projek ini menerangkan model robot pengimbang diri dua roda, menggunakan PID untuk merancang pengawal robot, dan menerapkan pengawal robot. Unit pengukuran inersia (IMU) digunakan dalam projek ini, yang menggabungkan pengukuran pecutan dan giroskop untuk mengukur dan mendapatkan sudut kemiringan robot. Untuk membetulkan ralat antara titik tetap sasaran dan sudut kemiringan sebenar, pengawal PID diterapkan dan mengubah kelajuan motor dc dengan tepat untuk menstabilkan robot.



ABSTRACT

The two-wheeled self-balancing robot is dynamically stable, but statically unstable, based on an inverted pendulum system. Various mechanics and control theories are involved in the robot. This project describes the model of the two wheeled self-balancing robot, uses PID to design the robot controller, and implements the robot controller. An inertial measurement unit (IMU) is used in this project, which combined the measurement of the accelerometer and gyroscope to measure and obtain a robot's tilt angle. To correct the error between the target fixed point and the real tilt angle, the PID controller is applied and changes the dc motor speed appropriately to stabilize the robot.



DEDICATION

To my supportive and beloved parents Ahmad Bin Taha and Maslona Binti Halpi.



ACKNOWLEDGEMENTS

In the Name of Allah, the Most Gracious, the Most Merciful

First and foremost, I would like to thank and praise Allah the Almighty, my Creator, my Sustainer, for everything I received since the beginning of my life. I would like to extend my appreciation to Universiti Teknikal Malaysia Melaka (UTeM) for providing the research platform.

Secondly, thanks to my beloved supervisor, Mrs. Izadora Mustaffa for all the knowledge, advice, guidance, ideas, critics, and feedback as well as for supporting me in doing this project. All the guidance was very helpful to complete my project.

I also would like to express my gratitude to my parents, Ahmad Bin Taha and Maslona Binti Halpi for motivating me along with the progress of this project. Not forgetting, my beloved friends Mohd Asyraf Bin Aman, Nurfarinah Binti Wilfred, Nur Fatin Nabila, Hind Binti Khalil, and others who have given me moral support and advice to help me throughout the project including the lecture

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

TABLE OF CONTENTS

	PAGE
LIST OF TABLES	xii
LIST OF FIGURES	xiii
LIST OF SYMBOLS AND ABBREVIATIONS	xvi
LIST OF APPENDICES	xv
CHAPTER 1	1
1.1 Introduction	1
1.2 Problem Statement	2
1.3 Objective	3
1.4 Scope of work	3
1.5 Thesis Outline	3
1.6 Conclusion	4
CHAPTER 2	5
2.1 Introduction	5
2.2 Two wheel self-balancing	5
2.2.1 Two-wheeled balancing robot controller designed using PID	5
2.2.2 Modelling, Simulation, and Optimal Control for Two-Wheeled Self-Balancing Robot	6

2.2.3	Control System for a Self-Balancing Robot	6
2.2.4	Movement Control of Two Wheels Balancing Robot using Cascaded PID Controller	7
2.2.5	Nonlinear dynamics modelling and simulation of two-wheeled Self- balancing vehicle	7
2.2.6	The prototype of self-balancing two-wheeler	8
2.2.7	Design and implementation of a self-balancing robot	8
2.2.8	Robust navigational control of a two-wheeled self-balancing robot in a sensed environment.	8
2.2.9	Control system in open-source FPGA for a self-balancing robot Juan	9
2.2.10	Adaptive observer-based output feedback control for a two-wheeled self-balancing robot	9
2.3	Conclusion	10
CHAPTER 3		12
3.1	Introduction	12
3.2	Hardware Implementation	14
3.2.1	Arduino UNO	15
3.2.2	MPU6050 Sensor	17
3.2.3	MPU6050 Sensor	19
3.2.4	Motor driver L298N	20

3.3	Software Implementation	21
3.3.1	Integrated Development Environment (IDE)	21
3.3.2	Processing	22
3.3.3	PID controller	23
3.3.4	Coding Implementation	24
3.4	Conclusion	29
CHAPTER 4		30
4.1	Introduction	30
4.2	Hardware overview	30
4.3	Result	32
4.4	Project analysis	33
4.5	Discussion	37
4.6	Conclusion	38
CHAPTER 5		39
5.1	Conclusion	39
5.2	Future Project Recommendations	40
REFERENCES		41
APPENDICES		43

LIST OF TABLES

TABLE	TITLE	PAGE
Table 2.1	Summary of the literature review	12



LIST OF FIGURES

FIGURE	TITLE	PAGE
Figure 3.1	Flow chart of the project	13
Figure 3.2	Block diagram of the hardware	14
Figure 3.3	Circuit diagram of the hardware	15
Figure 3.4	Arduino Uno	16
Figure 3.5	Arduino Uno to ATmega328 Pin Mapping	16
Figure 3.6	MPU6050 sensor	17
Figure 3.7	The connection between the MPU6050 sensor and Arduino	18
Figure 3.8	Servo motor	19
Figure 3.9	The connection between the DC motor and motor driver	19
Figure 3.10	Motor driver L298N	20
Figure 3.11	Connection between the Arduino and motor driver	20
Figure 3.12	processing	22
Figure 3.13	Setpoint and actual tilt angle	23
Figure 3.14	Flow chart of the coding	24
Figure 4.1	Side view of two wheel self-balancing vehicle	31
Figure 4.2	Top view of two wheel self-balancing vehicle	31
Figure 4.3	The two wheel vehicle balance itself without stand	32
Figure 4.4	Yaw pitch and roll reading. Left collum: yaw value.	33

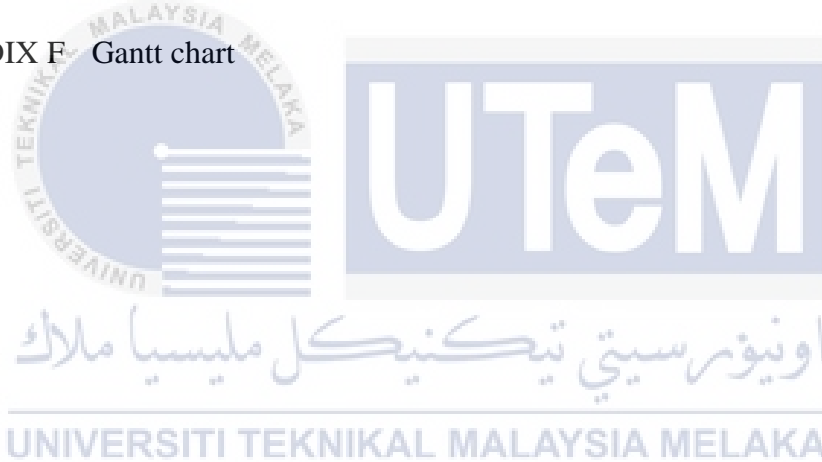
Middle column: pitch value. Right column: roll value

Figure 4.5	Reading standing upright condition to falling forward	34
Figure 4.6	Reading of falling forward condition	35
Figure 4.7	Reading of falling backward condition	36



LIST OF APPENDICES

APPENDIX	TITLE	PAGE
APPENDIX A	Project coding	47
APPENDIX B	Schematic of Arduino Uno Microcontroller Board	52
APPENDIX C	Schematic MPU6050 Board	53
APPENDIX D	Block diagram of project	54
APPENDIX E	Circuit diagram of the project	55
APPENDIX F	Gantt chart	56



LIST OF SYMBOLS AND ABBREVIATIONS

PID	-	Proportional Integral Derivative
LQR	-	Linear Quadratic Regulator
FLC	-	Fuzzy Logic Controller
SMC	-	Sliding Mode Controller
IMU	-	Inertial Measurement Unit
DC	-	Direct Current
TWSBR	-	Two-wheeled self-balancing robot
SE	-	Sensed Environment
FPGA	-	Field-programmable gate array
PWM	-	Pulse width modulation
USB	-	Universal Serial Bus
ICSP	-	In Circuit Serial Programming
V	-	Voltage
GND	-	Ground
INT	-	Interrupt
SDA	-	Serial Data
SCL	-	Serial Clock
IC	-	Integrated circuit
IDE	-	Integrated Development Environment
2D	-	Two-dimensional
3D	-	Three-dimensional
Ypr	-	Yaw, pitch, roll
Kp	-	proportional gain
Kd	-	derivative gain
Ki	-	integral gain
DMPP	-	Digital Motion Processor
GSM	-	Global system for mobile communication

CHAPTER 1

INTRODUCTION

1.1 Introduction

A significant type of mobile robot is a two-wheeled self-balancing robot. Adjusting robots infers the capacity of the robot without falling to adjust on its two wheels. Unlike many other control systems, the inverted pendulum mechanism is naturally unstable. Therefore to achieve equilibrium in this unstable state, the system has to be controlled. A two-wheeled balancing robot is simply an inverted pendulum system that standing upright upon two wheels.

The self-balancing robot has the benefits of being lightweight, compact, and low-cost, unlike other mobile robots, and has been used extensively for various events. As a special case of the inverted pendulum, the two-wheeled self-balancing robot has complex, multivariable, unstable, and nonlinear characteristics. The research of two wheeled balancing robots has increased in recent years to the invention of the Segway human transporter program. This unique project consists of robot modeling, developing a Proportional-Integral-Derivative (PID) controller, and implementing a two-wheeled robot controller.

1.2 Problem Statement

Mobile robots are now rapidly deployed that are used in a wide range of applications, which include discovery, search and rescue, dangerous area industrial equipment, and entertainment. While robots can move over obstacles, they are more complicated to construct and control because of the higher number of degrees of freedom. Wheeled robots are much more energy consuming, tend to have simple mechanical parts and a simple dynamic relative to the ground contact, and to have a driving force for wheeled robots. Static equilibrium can be accomplished by robots with at least three axes, making dynamics simpler.

Naturally, the inverted pendulum system is unstable. Therefore to monitor the system, an effective control system methodology and procedure need to be investigated. An application of the inverted pendulum that involves a controller to maintain its upright position is the two wheel balancing robot. To do this, it is important to build and implement a controller on the robot to stabilize the inverted pendulum.

The robot is fundamentally unstable and it will roll around the wheels' rotation axis without external control and ultimately fall. Various types of controls have been implemented on two wheeled balancing robots, including Linear Quadratic Regulator (LQR), Pole-Placement Controller and Fuzzy Logic Controller (FLC), and Sliding Mode Controller (SMC). Although several publications have simulated results, experimental results are notably lacking for both linear and nonlinear controllers.

1.3 Objective

To achieve this goal, the objectives are formulated as follows;

1. To design a prototype of two wheel self-balancing robot.
2. To develop a prototype of two wheel self-balancing robot.
3. To analyze the performance of the chosen self-balancing method.

1.4 Scope of work

The scope of the project focuses on the development of a balancing system for a two-wheel vehicle prototype, the robot's creations using PID, and applies the interface of the robot. The system would be able to support itself without having to stand. An inertial measurement unit (IMU) is used to combines gyroscope and acceleration sensor to assess and acquire the tilt angle of the device. The PID controller corrects the mistake between the specified reference point and the true inclination angle.

1.5 Thesis Outline

Chapter 1, which is the overview, discusses the project context, problem statement, and project goals, the complexity of the project, and the importance of this project.

Chapter 2 which is the literature review represented the literature review and quotation from any references regarding any information related to this project. The reference for equipment performance is also included in this segment.

A more detailed explanation of this project was covered in Chapter 3 which is the methodology. This chapter also provides information on process flow within this project.

Because the outcome of this experiment, followed by many reviews, is explained in Chapter 4.

The conclusion of Chapter 5 was based on the cumulative phase that took place in this project from the beginning to the completion of this project, followed by potential guidance from this project.

1.6 Conclusion

This project focuses on developing the balancing system for a two-wheel vehicle prototype. This chapter explains the introduction, the statement of the problem, the objective, and the scope. This is an important chapter where we define and consider a solution to the problem. Other than that, a goal that could direct us to smoothly complete the project. Finally, the scope is relevant when we list the basic component and project feature which set the project limit. The next chapter will be a summary of literature consisting of 10 articles as sources for our project.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

A significant type of mobile robot is a two-wheel self-balancing robot. Robot stability ensures a robot may balance its two wheels without falling. In this chapter, the literature review related to self-balancing, existing two-wheel self-balancing, and the hardware that fits to run the two-wheel self-balancing.

2.2 Two wheel self-balancing

2.2.1 Two-wheeled balancing robot controller designed using PID

Two-wheeled self-balancing robot is dynamically stable but physically unstable, based on an inverted pendulum design. The robot incorporates numerous theories of physics and power. An inertial measuring unit (IMU) is used in this project which combines accelerometer and gyroscope measurement to estimate and obtain the robot's tilt angle.

To fix the error between the target fixed point and the real tilt angle, the PID controller is used to adjust the dc motor speed to stabilize the robot accordingly. The result shows that the Control system can manage the robot appropriately with certain limitations. The model's simulation result is compared with the hardware developed and controller output is evaluated and discussed. (Suhardi, 2015).

2.2.2 Modelling, Simulation, and Optimal Control for Two-Wheeled Self-Balancing Robot

This paper deals with the two-wheeled self-balancing robot system as the object of research that uses the method of the Newtonian mechanic's equation for deriving the dynamic equation.

The linear state-space model approximates the non-linear system in the region of operation to the degree that the system operates only around the operating point and the signals involved are small. Based on the mathematical model of the system, the LQR controller is programmed to control the angle of inclination of the system and the angle of direction so that the system can be managed to shift to the desired location. The output of the control strategy shall be checked and presented using the Matlab / Simulink software.

(Asali et al., 2017)

2.2.3 Control System for a Self-Balancing Robot

The purpose of this project is to investigate the efficacy of various control algorithms in a home-made robot called Bimbo, such as Proportional, Integral, and Derivative (PID), pole placement, adaptive control, among others. The algorithm, applied to the location, was also evaluated. The robot was built with motion and position control modules. A Kalman filter was used to acquire a roll angle from the Inertial Measurement Unit (IMU).

Other than that, the system was introduced to read encoders and to control the two motors. A device variable control mechanism was created by Bluetooth communication, which allows any robot variable to be continuously monitored, enabling the device and the

control variables to be checked in real-time. A PID is implemented by the chosen solution, continuously updating the relation by a "location algorithm." To order the path of Bimbo navigation, a human control interface has been developed.(Martins & Nunes, 2017)

2.2.4 Movement Control of Two Wheels Balancing Robot using Cascaded PID Controller

The movement of control balancing robots is the subject of this paper. The robot is designed to move from the android device according to remote input. Three controls will be designed, the first to control the balance of the robot, the second to control the distance controller for the control of forwarding or backward movement, and the last to navigate or steer.

To make the robot remain balanced when moving, both balancing and distance control will be cascaded. This cascaded PID control will correct the angle error of the balancing controller by using the distance controller speed error. (Pratama et al., 2016)

2.2.5 Nonlinear dynamics modelling and simulation of two-wheeled Self-balancing vehicle

In this project, the cycle trajectory is used to analyze the device dynamics. The Lyapunov exponent trajectory and the system cycle can further explain the process model. Finally, simulation results ensure that the PID controller can maintain stability effectively and have a certain effect on the self-balancing two-wheeled vehicle framework. (Liu et al., 2016)

2.2.6 The prototype of self-balancing two-wheeler

The project concentrated on a two-wheeler bike development concept which is validated with the aid of a prototype. The project deals with a prototype research project to get a gyroscopic effect. The concept is a two-wheel vehicle in which the imparted revolving disks function as a gyroscope to create a counterbalancing force, i.e. gyroscopic effect when either side of the vehicle's configuration loses equilibrium. So, the vehicle stabilizes itself. (Patil *et al.*, 2017)

2.2.7 Design and implementation of a self-balancing robot

With a fixed tilt error of 5 degrees, a self-balancing robot was able to balance smoothly. The robot will carry some 0.3 kg of payloads. Diverse studies have calculated the optimal angle of tilt for balancing. The robot will shift forward and backward continuously within a spectrum of -3 cm and +3 cm across the equilibrium area to align itself on the flat surfaces. Used as a robot actuator, DC motor with higher torque. The angle of inclination fed manually into the machine or using intelligence. (Shaon, Bhowmik and Bhawmick, 2018)

2.2.8 Robust navigational control of a two-wheeled self-balancing robot in a sensed environment.

This research upgraded handheld pendulum robot, named a two-wheeled self-balancing robot (TWSBR), utilizing a compact 32-bit microcontroller in a sensed environment (SE) dependent Proportional-Derivative Proportional-Integral (PD-PI) control device.

The robot retains a balance with two wheels and a Kalman filter algorithm-based PD-PI controller during the navigation process and can stabilize when avoiding acute and complex obstacles in the sensed area. With ultrasonic waves, the Proportional (P) control is used to apply turn control in SE to avoid obstacles. (Iwendi *et al.*, 2019)

2.2.9 Control system in open-source FPGA for a self-balancing robot Juan

This paper introduces a robotics framework that was developed entirely using free FPGA tools. Developed, built, and programmed an inverted pendulum robot that integrates the iCE40HX4K-TQ144 Lattice using open FPGA tools such as Ice Studio and the IceZum Alhambra board. A PD control algorithm that directs two DC motors uses perception from an ultrasonic sensor. All the modules had been synthesized as proof of concept in an FPGA. Their experimental work shows positive results and conduct. (Cerezo, Morales, and Plaza, 2019).



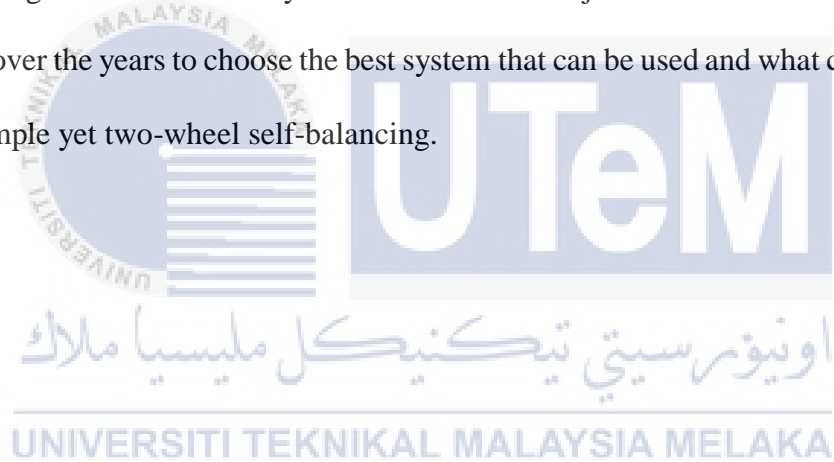
2.2.10 Adaptive observer-based output feedback control for a two-wheeled self-balancing robot

This paper provides for the two-wheeled self-balancing robot with an adaptive high gain control system. Adaptive observer use guarantees estimation of the joint state and the unknown parameter (actual body weight). Lyapunov research has carried out the convergence of the adaptive observer-dependent performance input controller. The simulation findings demonstrate both the performance of the suggested observer and the monitoring control scheme added to the self-balancing robot on two-wheelers. In future

research, we must bear in mind that the robot's self-balancing fundamentals are also compromised by unknown time-varying disturbances, and will make the control method more robust fur (Jmel *et al.*, 2020)

2.3 Conclusion

The overview of two-wheel self-balancing had been studied in this chapter including the past research on the two-wheel self-balancing system and the hardware related to this project. The development and design from several authors used different types of ideas and designs to simulate the system to achieve the objective. Other than that, it also had been made over the years to choose the best system that can be used and what can be improve to create simple yet two-wheel self-balancing.



No	Author	Year	Title	Microcontroller	Sensor	Function
1.	Suhardi	2015	Two-wheeled balancing robot controller designed using PID	PID controller	IMU sensor	Two-wheeled self-balancing
2.	Asali et al	2017	Modeling, Simulation, and Optimal Control for Two-Wheeled Self-Balancing Robot	1.LQR controller 2.PID controller	Pendulum	Two-wheeled self-balancing
3.	Martins & Nunes	2017	Control System for a Self-Balancing Robot	PID controller	IMU sensor	Two-wheeled self-balancing
4.	Pratama	2016	Movement Control of Two Wheels Balancing Robot using Cascaded PID Controller	PID Controller	IMU sensor	Two-wheeled self-balancing
5.	Yunping Liu	2016	Nonlinear dynamics modeling and simulation of two-wheeled self-balancing vehicle	PID controller	accelerometer sensor	Two-wheel self-balancing
6.	Ching Lung Chang	2017	Using Reinforcement Learning to Achieve Two-Wheeled Self Balancing Control	Arduino Uno	-gyroscope sensor - accelerometer sensor	Two-wheel self-balancing
7.	Bhowmik and Bhawmik	2018	Design and implementation of a self-balancing robot	Arduino Uno	-gyroscope sensor - accelerometer sensor	self-balancing robot
8.	Ji-Hyun Park	2018	Development of a self-balancing robot with a control moment gyroscope	Arduino Uno	-gyroscope sensor -inclinometer sensor	Two-wheel self-balancing
9.	Sergio Tamayo-León	2018	Self-Stabilization of a Riderless Bicycle With a Control Moment Gyroscope via Model-based Active Disturbance Rejection Control	Arduino Uno	-gyroscope sensor	Two-wheel self-balancing
10.	Mr. Aneesh Kulkarni	2019	Self-Balancing Robot	Arduino Uno	-Inertial Measurement Units (IMU)	- Robot balancing

Table 2.1: summary of the literature review

CHAPTER 3

METHODOLOGY

3.1 Introduction

As several types of research had been made based on previous studies, the hardware to use is chosen to run and complete this project. In the methodology section, the flow of the project, how the device runs, and what hardware that uses are stated. This methodology is important to plan the project flow until the project runs successfully. Also, the progress on completing the report documentation can be planned.

The flow chart of the project, which starts with a literature review of the data set was included in subjects related to two-wheeled self-balancing. The next step is to model the inverted pendulum after evaluating all the materials, as it presents a basic concept for this initiative. The next thing is to build the hardware and manufacture it. The method involves constructing the robot's foundation and body. This is accompanied by the implementation and hardware integration of software algorithms. Finally, for performance enhancement, the robot will be checked and fine-tuned.

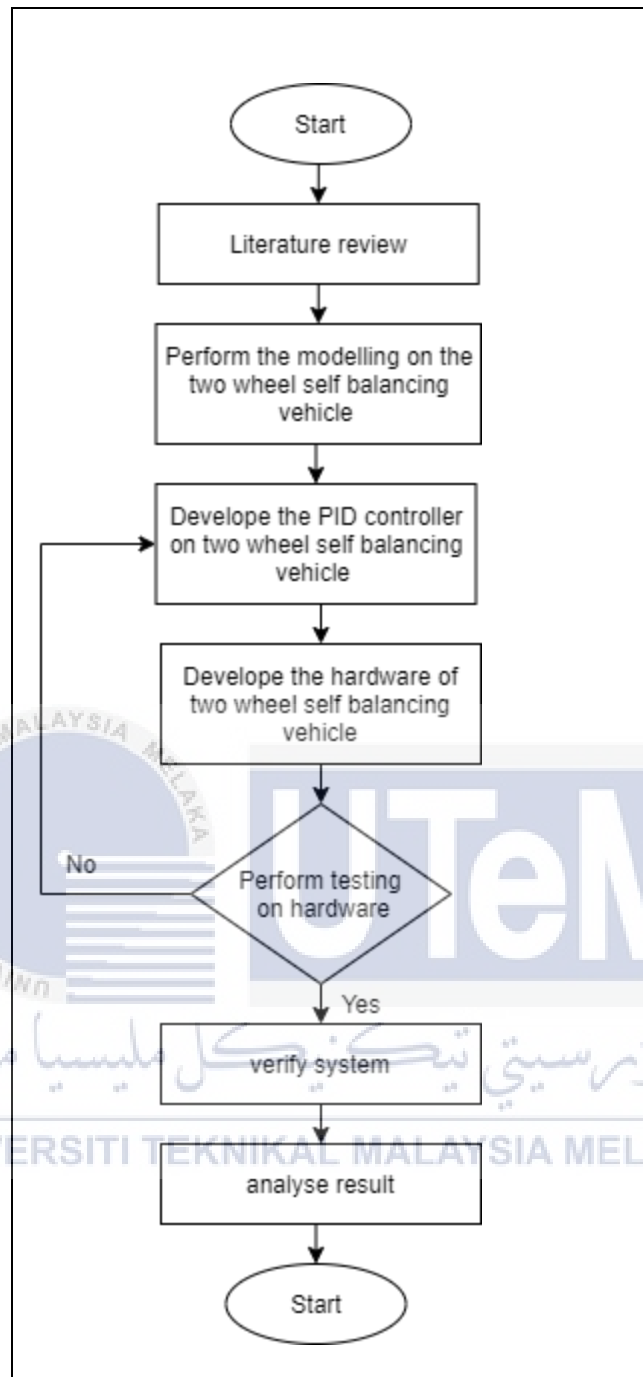


Figure 3.1: Flow chart of the project

3.2 Hardware Implementation

The Figure below is a description of two-wheel Self-Balancing Robot components and activity using dynamic stabilization techniques. The key components of this two-wheel self-balancing robot's circuit are inertial. The MPU6050 sensor, Arduino controller, and the servo motor DC.

The total block diagram of the electronic balancing robot system is seen in Figure 3.2. The IMU is used to calculate the angular rate and acceleration of the two-wheel and transform the result into the form of digital. Further analysis of the raw inputs from the IMU is to achieve the tilt angle of the robot. To balance the two wheels, this angle of inclination is then fed into the PID algorithm controller to produce the correct velocity for the DC motor.

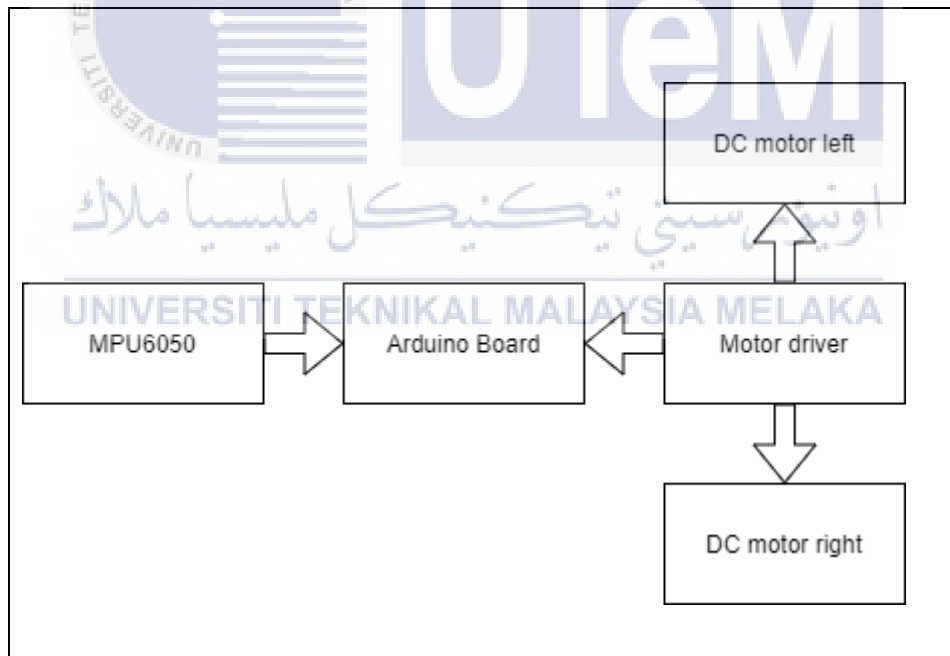


Figure 3.2: Block diagram of the hardware

This two-wheel self-balancing robot system uses Arduino and MPU6050 to connect the Arduino with MPU6050 and link the motors to Driver Motor Board. The complete circuit is powered by a 7.4V li-ion battery. Below shown the circuit diagram.

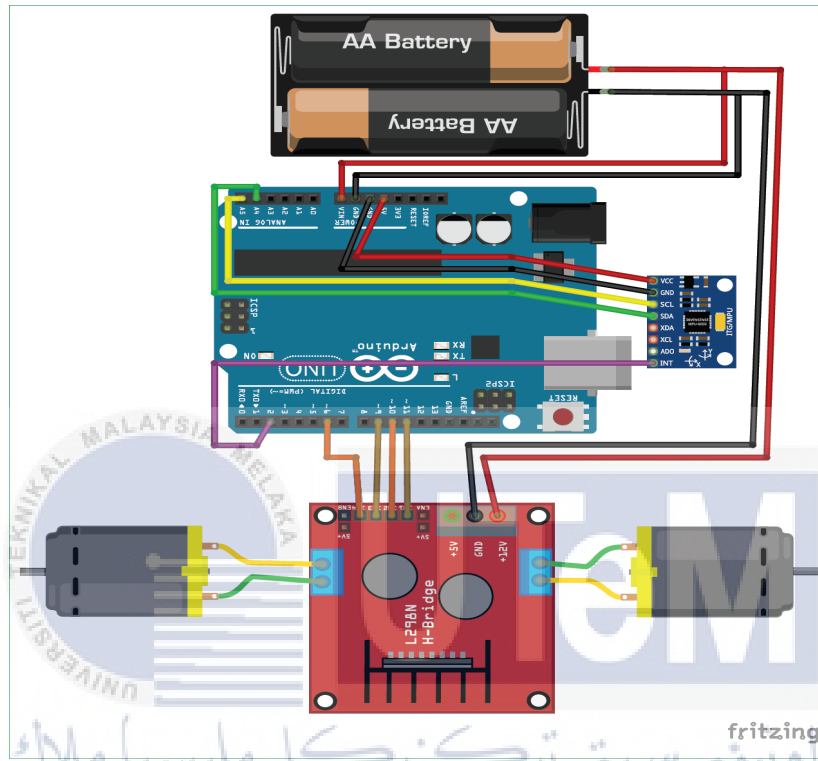


Figure 3.3: Circuit diagram of the hardware

3.2.1 Arduino UNO

Microcontroller Board, Arduino Uno is based on an 8-bit ATmega328P microcontroller. In addition to ATmega328P, it is composed of other microcontroller support components such as serial communication, voltage regulator, crystal oscillator, etc. Arduino Uno has input/output pins with 14 optical (6 and is used as a PWM output), 6 analog entry pins, USB interface, ICSP header, reset button, and Power barrel jack.



Figure 3.4: Arduino Uno

Using pin Mode, (digital Read) (and digital Write) (functions in the programming of Arduino, The 14 optical input/output pins may be used as output pins or input pins. Each pin shall operate at 5V and is capable of supplying or receiving has a maximum current of 40mA and has a detached internal pull-up resistor of 20-50 K Ohms.

Arduino function	ATmega328P Pin	Arduino function	ATmega328P Pin
reset	(PCINT14/RESET) PC6	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	GND	GND
GND	GND	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Figure 3.5: Arduino Uno to ATmega328 Pin Mapping

3.2.2 MPU6050 Sensor

To resolve the problem of measuring sounds and the drawbacks of measures from either the accelerometer or gyroscope alone, we would need to integrate the measures from both the accelerometer and the gyroscope in a practical manner so that we can use the strengths of both sensors to produce a more precise result than either test alone.

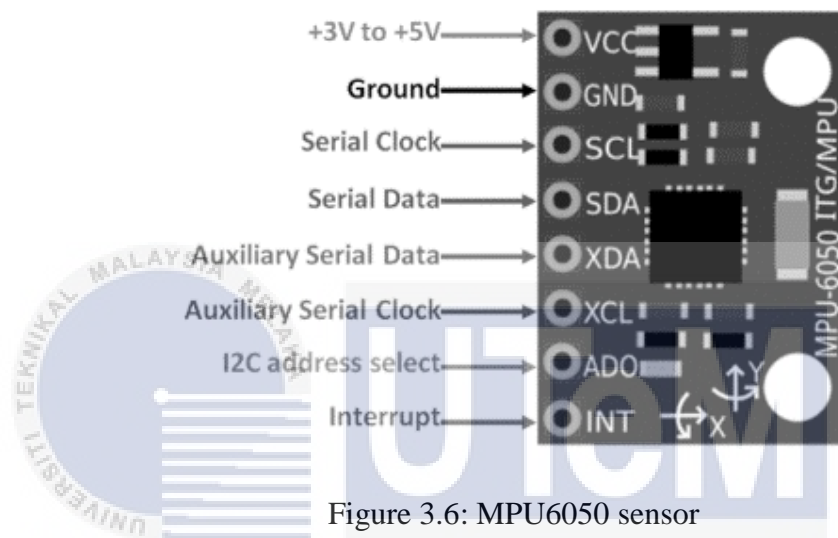


Figure 3.6: MPU6050 sensor

Through the I2C protocol, MPU6050 communicates with Arduino. The MPU6050 connects with Arduino as shown in the diagram below. If you have a 5V pin in the MPU 6050 module then can connect it to the 5V pin in Arduino. If not, then need to attach it to the 3.3V pin. Next, the Arduino's GND is attached to MPU6050's GND.

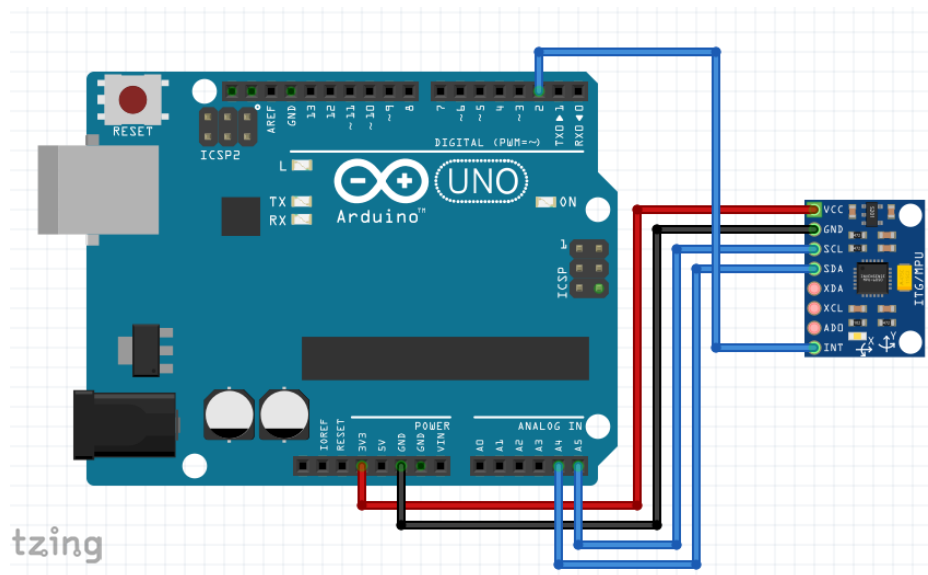


Figure 3.7: The connection between the MPU6050 sensor and Arduino

The software that we are going to run here also benefits from the interrupt button of the Arduino. Connect Arduino's digital pin 2 (interrupt pin 0) to the pin marked with an INT on the MPU 6050. Last, set up I2C lines. To do so, connect the pin marked as SDA on the MPU 6050 to the Arduino analog pin 4 (SDA), and the pin marked as SCL on the Arduino analog pin 5 (SCL) on the MPU 6050.

3.2.3 MPU6050 Sensor



Figure 3.8: Servo motor

The servo motor is used in this self-balancing robot as a component that detects the tilt to resist movement so as not to fall. The IMU sensor detects falls even and will send to the microcontroller and subsequently, the servo motor will be able to resist the current of the falling event at an approximate angle. So that the situation is stable again.

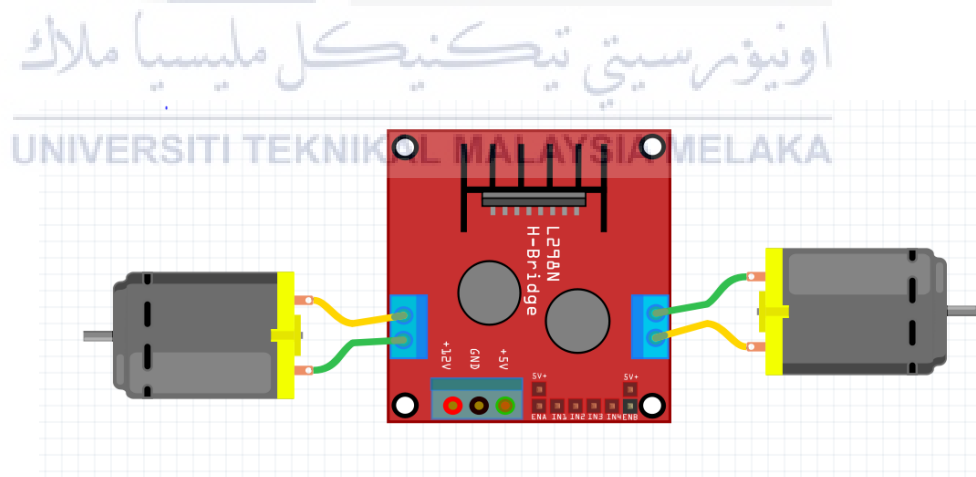


Figure 3.9: The connection between the DC motor and motor driver

3.2.4 Motor driver L298N

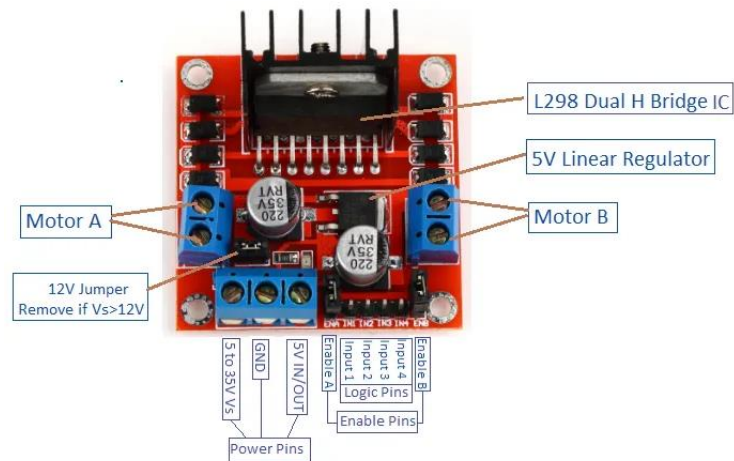


Figure 3.10: Motor driver L298N

In this project, the L298N module is used where it has a very well-known L298 Motor driver IC, which is the main part of this module. To control the speed of DC motors, this module uses the PWM system. The module will allow the speed and direction of two DC motors to be controlled. It can power motors that are between 5 to 35V and up to 2A in operation. The module has an on-board regulator that helps to provide a 5V production. The Arduino or external power supply will power the module from 5 to 35V.

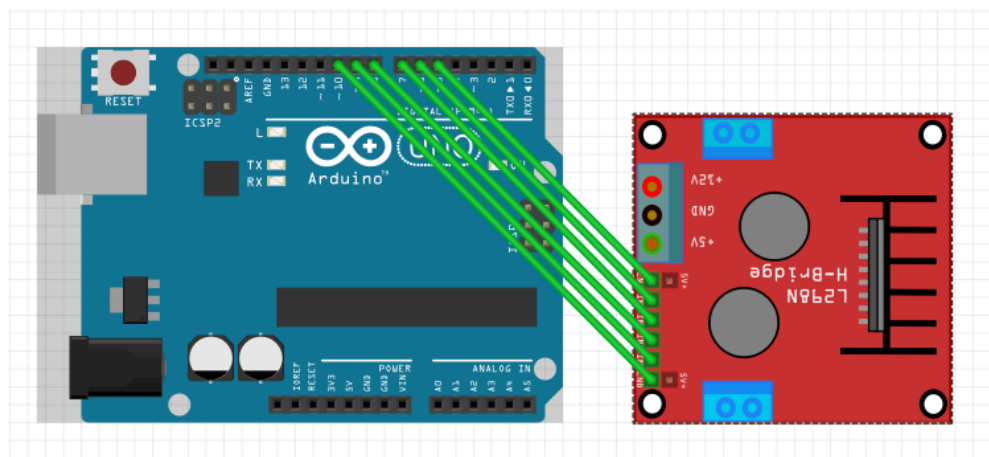


Figure 3.11: Connection between the Arduino and motor driver

The input and activate pins (ENA, IN1, IN2, IN3, IN4, and ENB) of the L298N module are connected to six Arduino digital output pins (5, 6, 7, 8, 9, and 10) for the connexion between the Arduino and this L298N module. Note that output pins 5 and 10 of the Arduino are both PWM-enabled.

3.3 Software Implementation

This section discusses the equilibrium of the two wheel self-balancing robot and the nature of the PID controller algorithm. Software development is the toughest part of the project and takes the most time.

3.3.1 Integrated Development Environment (IDE)

The open-source Arduino Software (IDE) encourages the writing and transfer of code to the computer. It is operating on all Windows, Mac OS X, and Linux. The framework is written in Java and based on other open-source tools and production. It can be used with any Arduino board using this software. This software can be used to program any operation that needs to be performed. The threshold level of the IMU sensor is set in this software to determine normal events and fall events.

3.3.2 Processing

Processing software is being used to display the sensor's 3d image, keep on reading, and display the 3D version of the MPU6050 data. Processing, but for a couple of features, is close to Arduino. Processing is mostly used in 2D/3D designs for visualizing details and producing them. The processing code (MPUTEapot) is performed by pressing the 'Run' symbol button. A thin, plane-like entity is about to emerge. Wait for around 10 seconds to stabilize the value of the MPU 6050. The 3D model of the MPU 6050 could be seen after that moves in compliance with the sensor.

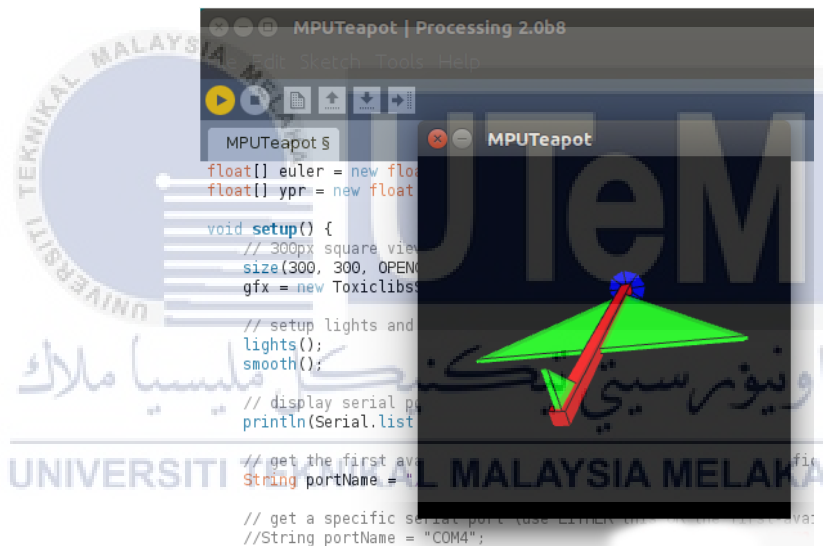


Figure 3.12: processing

To achieve an accurate location value from the sensor, the value of both the accelerometer and the gyroscope must be used, since the values of the accelerometer have noise issues and the values of the gyroscope continue to drift across time. So we're going to have to mix that to get the value of yaw pitch and roll of our robot, which we're going to use just the value of yaw.

3.3.3 PID controller

The control algorithm used to keep its equilibrium point on the self-balancing two-wheel robot was the PID controller. The Additive, PID controller was known as the 3-word controller. The PID controller can be a control loop feedback mechanism that is commonly used in the industry. The controller helps to adjust and correct the error between the calculated process and thus the target process and the output correction steps to manage the system accordingly. This controller must be operated regularly enough and at the same time within the controllable range of the machine.

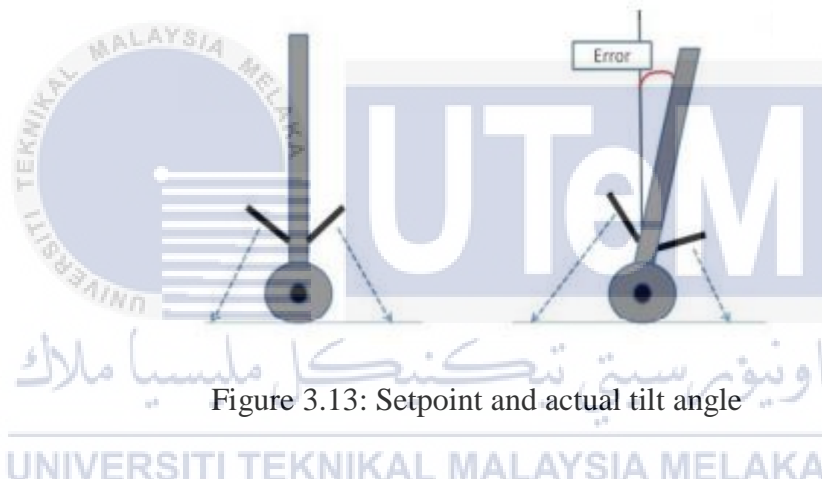


Figure 3.13: Setpoint and actual tilt angle

The setpoint and real tilt angle of the two-wheeled robot are seen in the diagram. The error is that the true tilt angle varies from the ideal tilt angle (setpoint). The PID controller includes, as its name implies, three parts, which are the proportional term, the integral term, and the derivative term. These words have various effects on the DC motor's reaction. The robot's setpoint must be 0° to stabilize the robot.

The real angle is the time-to-time instantaneous angle of the robot. The Inertial Measurement Unit (IMU), which generates dc output, determines this actual angle. We obtained the error by comparing it with the target setpoint, obtaining the difference between

the expected setpoint and the real angle. The error is then fed into the controller of the PID. To drive the DC motor, the PID controller can process, measure, and produce the exact output to reach equilibrium in the right way.

3.3.4 Coding Implementation

Below is the flow chart of the coding, is as below, and will explain each function in this coding. The full coding has been attach at appendix.

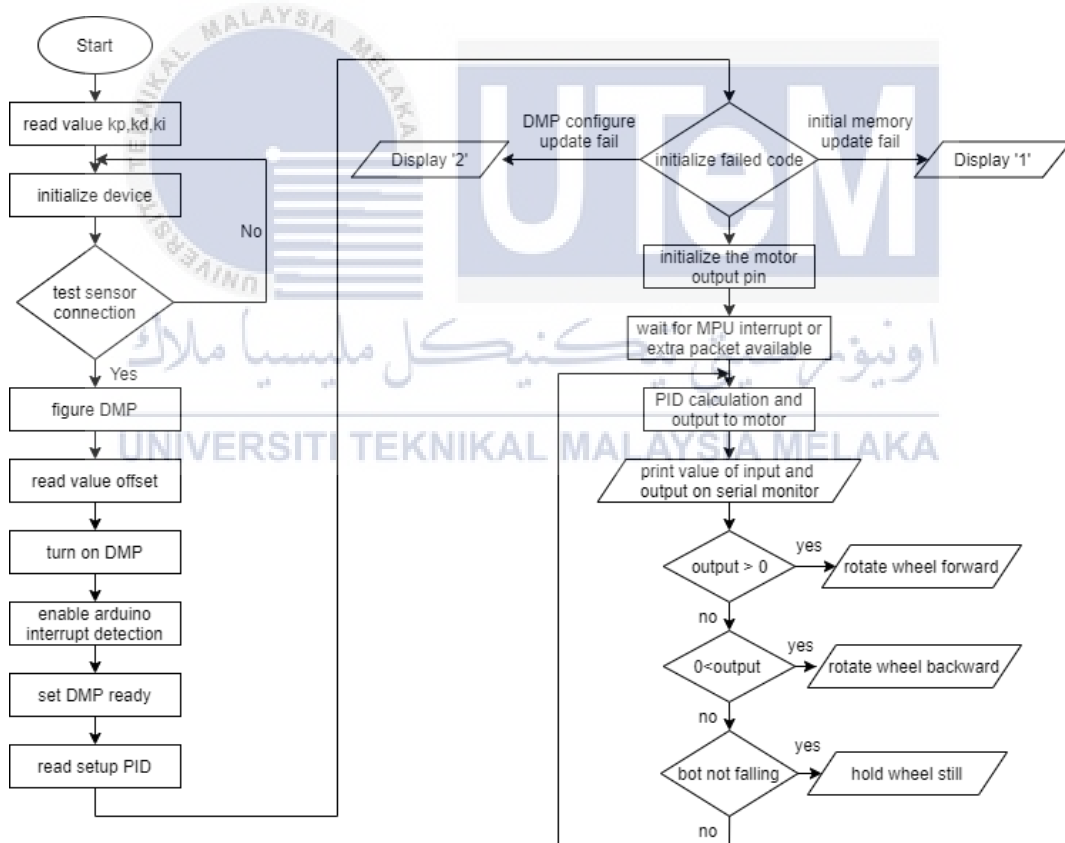


Figure 3.14: Flow chart of the coding

First, the libraries needed for the execution of this program are included. The built-in I2C library, PID library, and MPU6050 library are included, which can be downloaded from the GitHub website.

```
// Download The Libraries from those links and add to arduino library folder
#include "I2Cdev.h" // https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
#include <PID_v1.h> //From https://github.com/br3ttb/Arduino-PID-Library/blob/master/PID_v1.h
#include "MPU6050_6Axis_MotionApps20.h" //https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
```

The variables needed to get the data from the sensor MPU6050 are then declared. Analyze all gravity vector and quaternion values, then calculate the yaw pitch and roll weight of the bot. The ypr [3] float array will carry the final answer.

```
// MPU control/status vars
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorFloat gravity; // [x, y, z] gravity vector
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector
```

Next comes the most critical section of the code, and this is where those spend a long time tuning for an accurate set of values. If the robot is built with a very high center of gravity and the modules are arranged symmetrically (which is not, in most cases), the set-point value will be 180. Otherwise, attach the robot to Arduino Serial Monitor and tilt until it finds the right balance spot, read the value on the serial monitor, so this is the value of the fixed point. The value of Kp, Kd, and Ki should be adjusted as per the robot. There would be no two identical bots with the same values of Kp, Kd, and Ki, and there would be no relief from it.

```

/*****Tune these 4 values for your BOT*****/
double setpoint= 179.55; //set the value when the bot is perpendicular to ground using serial monitor.

#####
//Change those Values according to your Design
#####
double Kp = 22; //Set this first
double Kd = 0.8; //Set this second
double Ki = 180; //Finally set this
#####

```

In the next row, configure the PID algorithm by adding input, output, setpoint, Kp, Ki, and Kd parameters to the input variables. Fixed the set-point Kp, Ki, and Kd values in the above code snippet from these. The input value will become the actual yaw value read from the MPU6050 sensor and the output value will become the value calculated by the PID algorithm. The PID algorithm gives the final output that can be used to adjust the value of the input to be close to the point range.

```

PID pid(input, output, setpoint, Kp, Ki, Kd, DIRECT);

```

Initializing the MPU6050 inside of the void configuration feature by configuring the DMPP (Digital Motion Processor). It will allow us to integrate the data from the Accelerometer with data from the Gyroscope to provide Yaw, Pitch, and Roll with a consistent value. Each MPU6050 sensor does have its offset values and can measure the offset value of the sensor using this Arduino schematic and change the following lines in the software accordingly.

```

// supply your own gyro offsets here, scaled for min sensitivity
mpu.setXGyroOffset(63);
mpu.setYGyroOffset(-18);
mpu.setZGyroOffset(39);
mpu.setZAccelOffset(1013);

```

Then, enable the Digital PWM pins used to attach the motors to them. There are D6, D9, D10, and D11 in this case. So, enable these pins by default, as the output pins render them LOW.

```
//Initialise the Motor outpu pins
pinMode (6, OUTPUT);
pinMode (9, OUTPUT);
pinMode (10, OUTPUT);
pinMode (11, OUTPUT);

//By default turn off both the motors
analogWrite (6,LOW);
analogWrite (9,LOW);
analogWrite (10,LOW);
analogWrite (11,LOW);
```

Check if the data from the MPU6050 is ready for reading within the main loop feature. If yes, then use it to measure the PID value and then display on the serial monitor the PID input and output value only to test how the PID reacts. Decide if the bot needs to step forward or backward or stand still depending on the performance value.

Since we expect that when the bot is upright, the MPU6050 will return 180. When the bot drops to the front, we will get positive correction values and if the bot drops to the back, we will get negative values. So, to move the bot forward or backward, test for this state and call the necessary functions.

```

while (!mpuInterrupt && fifoCount < packetSize)
{
    //no mpu data - performing PID calculations and output to motors
    pid.Compute();

    //Print the value of Input and Output on serial monitor to check how it is working.
    Serial.print(input); Serial.print(" =>"); Serial.println(output);

    if (input>150 && input<215){//If the Bot is falling

    if (output>0) //Falling towards front
    Forward(); //Rotate the wheels forward
    else if (output<0) //Falling towards back
    Reverse(); //Rotate the wheels backward
    }
    else //If Bot not falling
    Stop(); //Hold the wheels still

}

```

The output vector of the PID also affects how fast the motor needs to be turned. If the bot may be about to fall, make a slight adjustment by slowly spinning the wheel. We increase the speed of the motor if these slight correction dint work and even if the bot is falling. By the PI algorithm, the importance of how quickly the wheels spin can be determined.

```

void Reverse() //Code to rotate the wheel forward
{
    analogWrite(6,0);
    analogWrite(9,output*-1);
    analogWrite(10,output*-1);
    analogWrite(11,0);
    Serial.print("F"); //Debugging information
}

void Forward() //Code to rotate the wheel Backward
{
    analogWrite(6,output);
    analogWrite(9,0);
    analogWrite(10,0);
    analogWrite(11,output);
    Serial.print("R");
}

void Stop() //Code to stop both the wheels
{
    analogWrite(6,0);
    analogWrite(9,0);
    analogWrite(10,0);
    analogWrite(11,0);
    Serial.print("S");
}

```

3.4 Conclusion

To conclude, the correct method must be used to make the project a success. Hardware must also be selected wisely to ensure that the circuit or system developed is functioning properly. Other than that, software interfaces are often essential to ensure that the hardware is working as programmed correctly. So the entire process of this project has already been identified and seen well. There is still scope for improvement, however, to make this project more efficient and efficient. The team needs to plan a structured time table to ensure that each operation is performed in due time.



CHAPTER 4

RESULT AND ANALISYS

4.1 Introduction

In this chapter, the results of this project will be evaluated and debated. Other than that, to achieve the purpose or objectives of this project, which is to build a two-wheel self-balancing robot using a dynamic stabilization technique, the outcome of a failing event will be evaluated. This segment will also clarify the flow of the project method.

4.2 Hardware overview

This project is made to balance the two-wheeled self-balancing robot, which is the robot's ability to balance two wheels without falling. The hardware used is included Arduino Uno, sensor MPU6050, motor, and motor driver. Each component involved in the detection of falling action of the robot and then act to balance the two wheels under certain conditions.

Finished hardware with casing

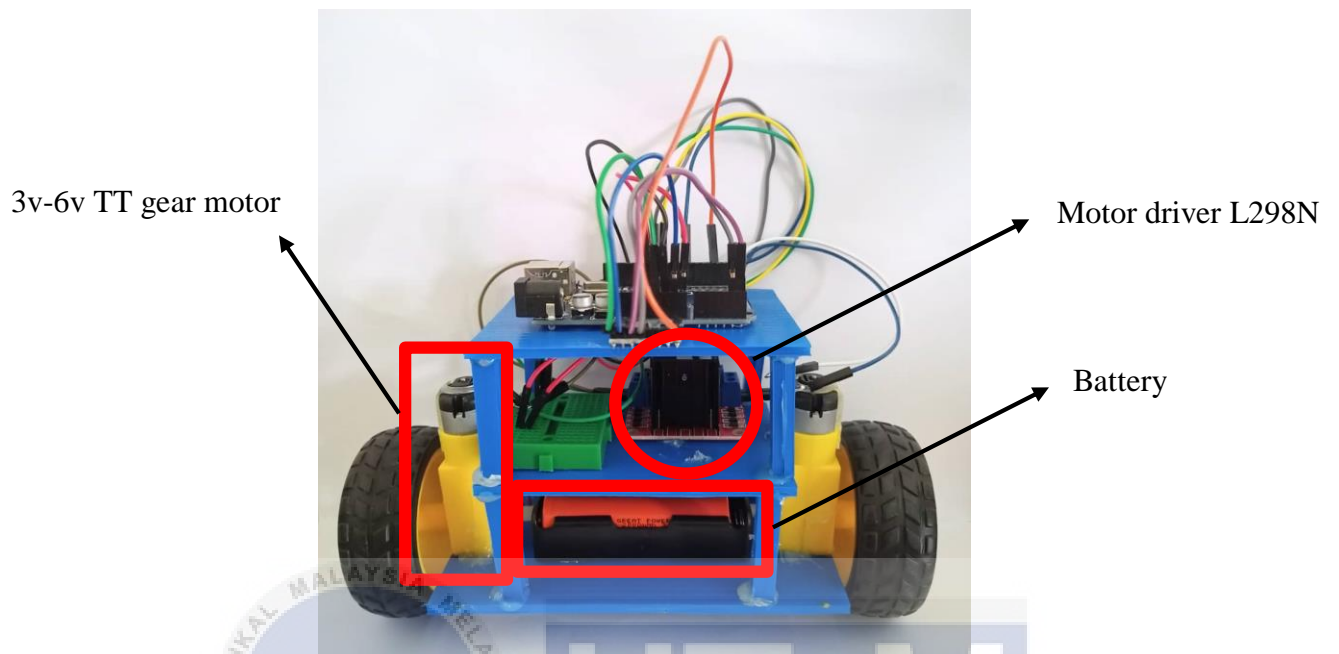


Figure 4.1: Side view of two wheel self-balancing vehicle

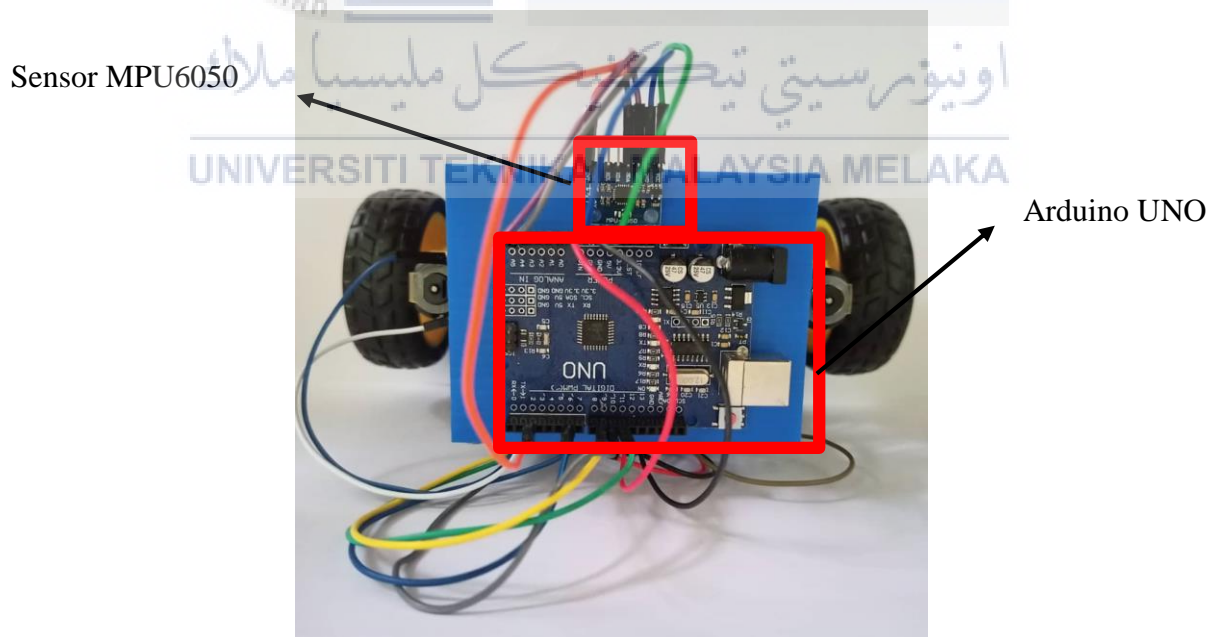


Figure 4.2: Top view of two wheel self-balancing vehicle

4.3 Result

The result is as the Figure below:

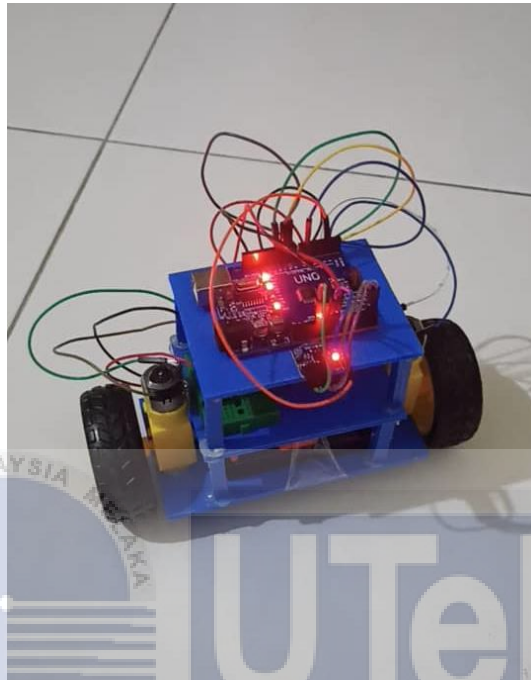


Figure 4.3: The two wheel vehicle balance itself without stand

Based on the Figure above, the two-wheel self-balancing robot balances itself on two wheels being able to move around without overturning. A "closed-loop feedback control" system is used by self-balancing robots; this shows that real-time motion sensor data is used to control the motors and quickly compensate for any tilting motion to hold the robot upright.

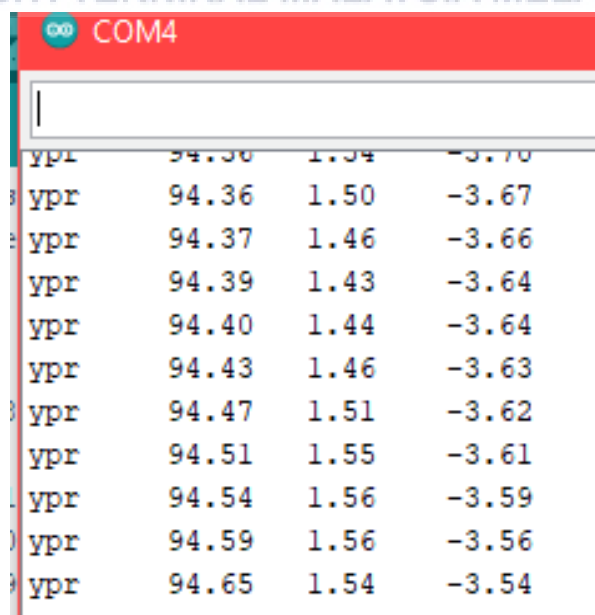
The control algorithm was used to maintain the balance of the PID controller on the autonomous two-wheel self-balancing robot. A three-term controller is well known as the proportional, integral, and derivative (PID) controller. An error from the device is an input

to the controller. The additive, integral, and derivative constants of K_p , K_i , and K_d are referred to as (the three terms get multiplied by these constants respectively).

When the robot is about to fall, the wheels rotate with a speed corresponding to the angle of fall in the inclined direction to correct the inclination. It is possible to do this by calculating the tilt angle. This is accomplished by a gyroscope and an accelerometer that shows the robot's correct orientation. This is fed back to the Arduino with the support of the feedback element, and the motor and the wheels together prevent the robot from collapsing with a corrective element. This ensures an upright robot stands.

4.4 Project analysis

The serial monitor on the software IDE will display the output value of the MPU6050 sensor which depends on the position of the sensor. The first column is the yaw output value, followed by the pitch output value and then the output roll value. roll is a horizontal axis, the pitch is a vertical axis and yaw is a perpendicular axis to the pitch and roll axis that allows the sensor to detect 3-dimensional states.



Label	Yaw	Pitch	Roll
ypr	94.36	1.54	-3.70
ypr	94.36	1.50	-3.67
ypr	94.37	1.46	-3.66
ypr	94.39	1.43	-3.64
ypr	94.40	1.44	-3.64
ypr	94.43	1.46	-3.63
ypr	94.47	1.51	-3.62
ypr	94.51	1.55	-3.61
ypr	94.54	1.56	-3.59
ypr	94.59	1.56	-3.56
ypr	94.65	1.54	-3.54

Figure 4.4: Yaw pitch and roll reading. Left collum: yaw value. Middle column: pitch value. Right column: roll value

The serial monitor of the IDE software display the input and output values for the PID algorithm are shown in the input => output format. The input value are the current yaw value read from the MPU6050 sensor and the output value are the value determined by the PID algorithm. The 'F' alphabet means that the bot is falling forward, and 'R' means that the bot is falling backward and moving in reverse.



Figure 4.5: Reading standing upright condition to falling forward

The Figure above shows the value when the robot in an upright position. the position of the two wheels changes from standing upright to leaning forward. The serial monitor shows that both wheels are moving forward by displaying the letter "F". The input value is at 179.62 while the offset key in the encoding is 179.55. The offset value means that

the value of two wheels perpendicular to the ground means that it is a stable value of two wheels.

Next is the falling forward condition where the input and output value is as below in the serial monitor:

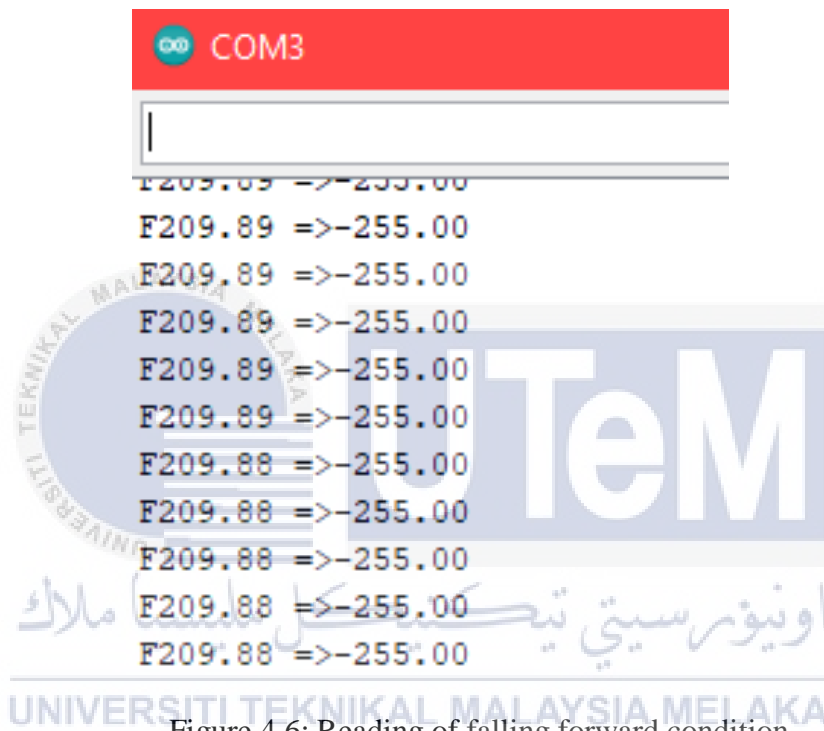


Figure 4.6: Reading of falling forward condition

The Figure above shows the reading value when the robot falls forward. what we can observe is that when the situation falls forward, the serial monitor shows that both wheels are moving forward by displaying the letter "F". The input value is at 209 which is greater than the setpoint value which is 179.55. when the input value is greater than the setpoint value, it means the robot falls forward. motors and wheels also respond by moving forward when receiving a signal from a sensor to prevent the robot from falling with a correction element from the PID. This is because to ensure the robot is upright and does not fall.

Next is the falling backward condition where the input and output value is as below in the serial monitor:

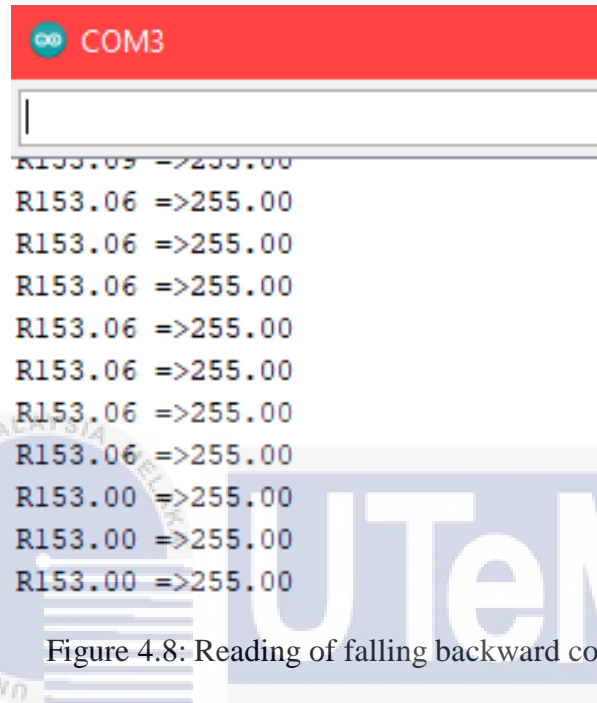


Figure 4.8: Reading of falling backward condition

The Figure above shows the reading value when the robot falls backward. what we can observe is that when the backward state falls, the serial monitor shows that both wheels are pointing forward by displaying the letter "R". The input value is 152 which is less than the setpoint value which is 179.55. When the input value is less than the setpoint value, it means the robot falls backward. motors and wheels also respond by moving backward when receiving a signal from a sensor to prevent the robot from falling with a correction element from the PID. This is because to ensure the robot is upright and does not fall.

Data from the MPU6050 can be read in the main loop function in encoding. Then it is used to calculate the PID value and then show the input and output values of the PID on

the serial monitor just to confirm how the PID responds. After that, it is determined based on the output value, whether the two wheels should step forward or backward or stop.

This is when the robot is upright, the MPU6050 will return 179.55. When the robot falls forward, we get a value greater than 179.55 and if the ship falls backward, we get a value smaller than 179.55. Therefore, to move the robot forward or backward, it detects this condition and calls the required function.

4.5 Discussion

From the analysis, the input value which is the value of yaw plays a major role in determining whether the two wheels are falling state or not. In the IDE software, the input reading can be observed on the serial monitor. In the serial monitor, there are two columns which are input and output, which is input is the value of yaw and output is the value determined by the PID algorithm. These values of yaw will be read by Arduino Uno microcontroller which when it exceeds the specific threshold of the sensor it will transmit the information to move the wheel either forward or backward.

The reading of value three axis yaw, pitch, and raw depend on how the sensor was placed. So, If the direction of the sensing axis matches the value offset (perpendicular to the ground) which is 179.55, the two wheels are balanced well. Then if the value of yaw is below 179.55, it is mean the two wheels are falling backward. The serial monitor will display the alphabet "R" to show that the two wheels are falling backward. Then, the motor and wheel will rotate backward together to prevent the two wheels from collapsing with a corrective element to ensure the two wheel to stay balance.

as well as forward conditions when the value of yaw is exceeded 179.55, it is mean the two wheels are falling forward. The serial monitor will display the alphabet "F" to show

that the two wheels are falling backward. Then, the motor and wheel will rotate forward together to prevent the two wheels from collapsing with a corrective element to ensure the two wheel to stay balance.

The Arduino microcontroller act as the medium to detect input and generates output for the system. The IDE software is used to sketch the code to program the board to run the system. In the project, the function used to make the falling event more precise is from the value of yaw from the sensor. Sox this system can detect the real falling event and false falling event.

4.6 Conclusion

The outcome of that project performance was evaluated and discussed at the end of this chapter. The value output for the PID algorithm was the parameter evaluated in this section when standing upright, falling forward, falling backward, and stopping. However, to make this project more effective and realistic, there are still improvements that can be made.

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

This two wheel self-balancing vehicle is balancing itself on two wheels without a stand. A "closed-loop feedback control" system is used for self-balancing vehicles; this ensures that the Arduino UNO as a microcontroller and sensor MPU6050 is used to control the engines in real-time data from the balancing sensor and easily compensate for any tilting movement to hold the two wheels upright. This project is using sensor MPU6050 to design and develop a system that can detect falling action.

The IDE software was also used to draw the code for this project, where it was important to select the precise function and the counter function used for this project. The goal of this project was successfully achieved during the project progress time, where the performance electronics used for this project were studied and the design and production of this project was a complete success.

5.2 Future Project Recommendations

Several improvements can be made to this project to make it more usable and sophisticate. First is by connecting the GSM module with the Arduino microcontroller wirelessly. It can be achieved by using a Bluetooth device that consumes low power and short distances that consume low power and short distances. By adding Bluetooth, this two-wheel can be controlled by using the application which is MIT application. So, this two-wheel can be used at the restaurant as a waiter to serve food for example.

Then, a two-wheel self-balancing system modelling and simulation with a different control principle is strongly recommended. A low noise reference voltage circuit board can be used to monitor the conversion by supplying an exact reference voltage to the microcontroller. Reduces erratic reference voltage and noise in the data conversion process.

Dc motor with encoder is preferred as the encoder motor usually has a high torque and speed requirement compared to the dc motor. The encoder can be used to receive inputs to evaluate and control the speed of the engine very precisely. Other than that, the diameter of the wheel can be expanded to improve the moment of friction and traction on the floor. However, improved balancing efficiency can be predicted for future work on the update and advice mentioned above.

REFERENCES

Martins, R. S., and Nunes, F. (2017) 'Control System for a Self-Balancing Robot', (Project I), pp. 297–302.

(Martins and Nunes, 2017) Martins, R. S., and Nunes, F. (2017) 'Control System for a SelfBalancing Robot', (Project I), pp. 297–302.

(Pratama, Binugroho and Ardilla, 2016) Pratama, D., Binugroho, E. H. and Ardilla, F. (2016) 'Movement control of two wheels balancing robot using cascaded PID controller', Proceedings - 2015 International Electronics Symposium: Emerging Technology in Electronic and Information, IES 2015, pp. 94–99. DOI: 10.1109/ELECSYM.2015.7380821.

(Liu et al., 2016) Liu, Y. et al. (2016) 'Nonlinear dynamics modeling and simulation of two wheeled self-balancing vehicle', International Journal of Advanced Robotic Systems, 13(6), pp. 1–9. DOI: 10.1177/1729881416673725. (Patil et al., 2017)

Patil, R. et al. (2017) 'Prototype of Self-Balancing Two Wheeler', 7th International Conference on Recent Trends in Engineering Science and Management (ICRTE SM-17), 1, pp. 466–471.

(Shaon, Bhowmik and Bhawmick, 2018) Shaon, A. K. M. A. S., Bhowmik, S. and Bhawmick, B. K. (2018) 'DESIGN AND IMPLEMENTATION OF A SELF-BALANCING ROBOT ICMERE2017-PI-294 DESIGN AND IMPLEMENTATION OF A SELF-BALANCING ROBOT', (February).

(Iwendi et al., 2019) Iwendi, C. et al. (2019) 'Robust Navigational Control of a Two-Wheeled Self-Balancing Robot in a Sensed Environment', IEEE Access, 7, pp. 82337–82348. DOI: 10.1109/ACCESS.2019.2923916.

(Cerezo, Morales and Plaza, 2019)Cerezo, J. O., Morales, E. C. and Plaza, J. M. C. (2019) 'Control system in open-source FPGA for a self-balancing robot', Electronics (Switzerland), 8(2), pp. 1–19. DOI: 10.3390/electronics8020198. (Jmel et al., 2020)

Jmel, I. et al. (2020) 'Adaptive Observer-Based Output Feedback Control for Two-Wheeled Self-Balancing Robot', 2020. (Son and Anh, 2014)

Son, N. N., and Anh, H. P. H. (2014) 'Adaptive backstepping self-balancing control of a two-wheel electric scooter', International Journal of Advanced Robotic Systems, 11, pp. 1–11. DOI: 10.5772/59100.

(Fang, 2014)Fang, J. (2014) 'The LQR controller design of two-wheeled self-balancing robot based on the particle swarm optimization algorithm', Mathematical Problems in Engineering, 2014. DOI: 10.1155/2014/729095.

Maker Pro. 2021. How To Interface Arduino And The MPU 6050 Sensor | Arduino. [online] Available at: [Accessed 5 January 2021]. Electronicwings.com. 2021. MPU6050 Interfacing With Arduino UNO | Arduino. [online] Available at: [Accessed 5 January 2021].

Robots, P., 2021. The MPU6050 Explained. [online] Programming Robots. Available at: [Accessed 5 January 2021]. Robots, I., 2021. Introduction To Self-Balancing Robots | Chillibasket. [online] chillibasket. Available at: [Accessed 5 January 2021].

Robots, P., 2021. The MPU6050 Explained. [online] Programming Robots. Available at: [Accessed 5 January 2021]. Elektor. 2021. Balbot: A Self-Balancing Robot. [online] Available at: [Accessed 6 January 2021]

APPENDICES

APPENDIX A: Project coding

```
#include "I2Cdev.h" //
https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
#include <PID_v1.h> //From https://github.com/br3ttb/Arduino-PID-Library/blob/master/PID\_v1.h
#include "MPU6050_6Axis_MotionApps20.h"
//https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
MPU6050 mpu;
// MPU control/status vars
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, != 0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
```

```

Quaternion q; // [w, x, y, z] quaternion container

VectorFloat gravity; // [x, y, z] gravity vector

float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

/*****Tune these 4 values for your BOT*****/

double setpoint= 179.55; //set the value when the bot is perpendicular to ground using
serial monitor.

#####
//Change those Values according to your Design
#####
double Kp = 22; //Set this first
double Kd = 0.8; //Set this second
double Ki = 180; //Finally set this
#####

double input, output;

PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

```

```

volatile bool mpuInterrupt = false; // indicates whether MPU interrupt pin has gone
high

void dmpDataReady()

{
    mpuInterrupt = true;
}

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

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // verify connection
    Serial.println(F("Testing device connections...!"));

    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") :
F("MPU6050 connection failed"));

    // load and configure the DMP

    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity

```

```

mpu.setXGyroOffset(63);

mpu.setYGyroOffset(-18);

mpu.setZGyroOffset(39);

mpu.setZAccelOffset(1013);

// make sure it worked (returns 0 if so)
if (devStatus == 0)
{
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
    attachInterrupt(0, dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's okay to use it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();

```



```

//setup PID

pid.SetMode(AUTOMATIC);

pid.SetSampleTime(10);

pid.SetOutputLimits(-255, 255);

}

Else

{

// ERROR!

// 1 = initial memory load failed

// 2 = DMP configuration updates failed

// (if it's going to break, usually the code will be 1)

Serial.print(F("DMP Initialization failed (code ");

Serial.print(devStatus);

Serial.println(F(")"));

}

//Initialise the Motor output pins

pinMode (6, OUTPUT);

pinMode (9, OUTPUT);

pinMode (10, OUTPUT);

pinMode (11, OUTPUT);

```

```

//By default turn off both the motors

analogWrite(6,LOW);

analogWrite(9,LOW);

analogWrite(10,LOW);

analogWrite(11,LOW);

void loop() {

// if programming failed, don't try to do anything
if (!dmpReady) return;

// wait for MPU interrupt or extra packet(s) available
while (!mpuInterrupt && fifoCount < packetSize)
{
//no mpu data - performing PID calculations and output to motors
pid.Compute();

//Print the value of Input and Output on the serial monitor to check how it is
working.

Serial.print(input); Serial.print(" =>"); Serial.println(output);

if (input>150 && input<215){//If the Bot is falling

if (output>0) //Falling towards front

```

```

Forward(); //Rotate the wheels forward

else if (output<0) //Falling towards back

Reverse(); //Rotate the wheels backward

}

else //If Bot not falling

Stop(); //Hold the wheels still

}

// reset interrupt flag and get INT_STATUS byte

mpuInterrupt = false;

mpuIntStatus = mpu.getIntStatus();

// get current FIFO count

fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)

if ((mpuIntStatus & 0x10) || fifoCount == 1024)

{

// reset so we can continue cleanly

mpu.resetFIFO();

Serial.println(F("FIFO overflow!"));

// otherwise, check for DMP data ready interrupt (this should happen frequently)

}

else if (mpuIntStatus & 0x02)

```

```

{
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;

    mpu.dmpGetQuaternion(&q, fifoBuffer); //get value for q
    mpu.dmpGetGravity(&gravity, &q); //get value for gravity
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity); //get value for ypr
    input = ypr[1] * 180/M_PI + 180;
}
}

```

```

void Reverse() //Code to rotate the wheel forward

```

```

{
    analogWrite(6,0);
    analogWrite(9,output*-1);
    analogWrite(10,output*-1);
}

```

```

analogWrite(11,0);

Serial.print("F"); //Debugging information
}

void Forward() //Code to rotate the wheel Backward
{
    analogWrite(6,output);

    analogWrite(9,0);

    analogWrite(10,0);

    analogWrite(11,output);

    Serial.print("R");
}

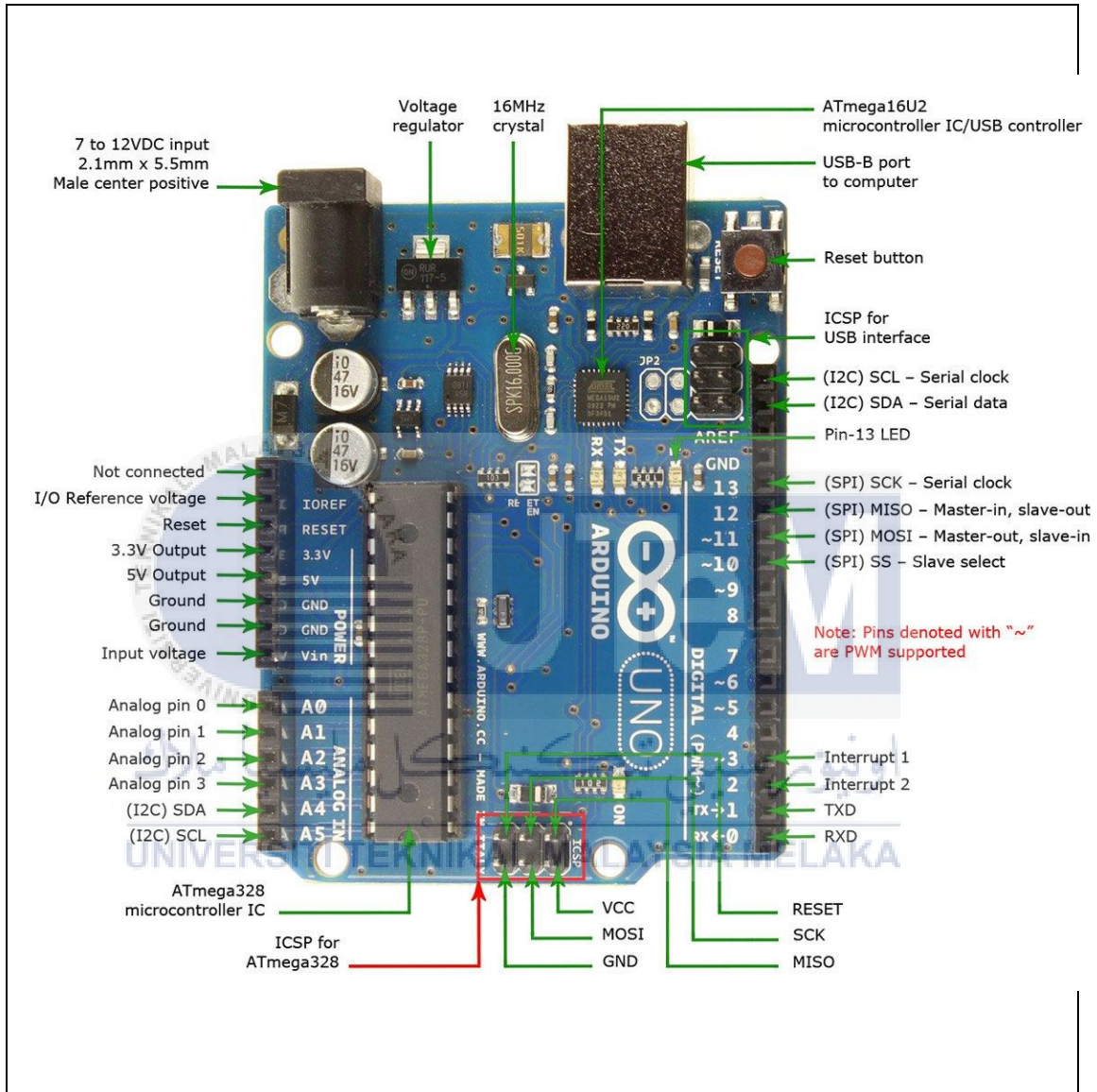
void Stop() //Code to stop both the wheels
{
    analogWrite(6,0);
    analogWrite(9,0);
    analogWrite(10,0);
    analogWrite(11,0);

    Serial.print("S");
}

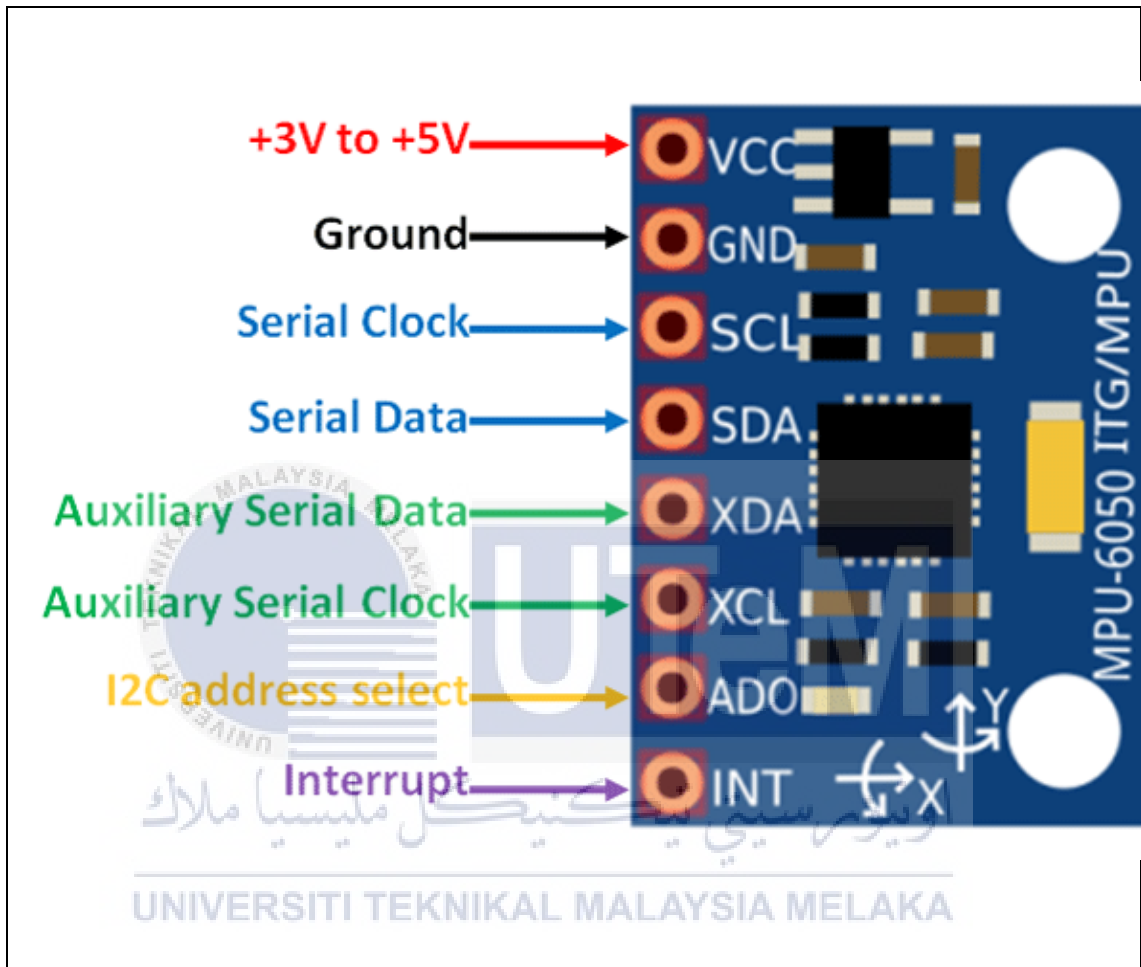
```



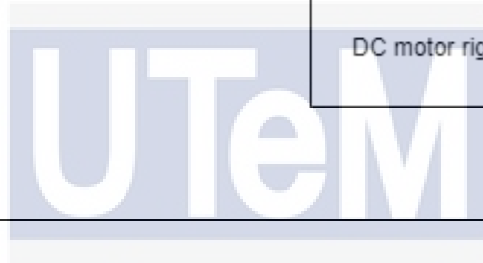
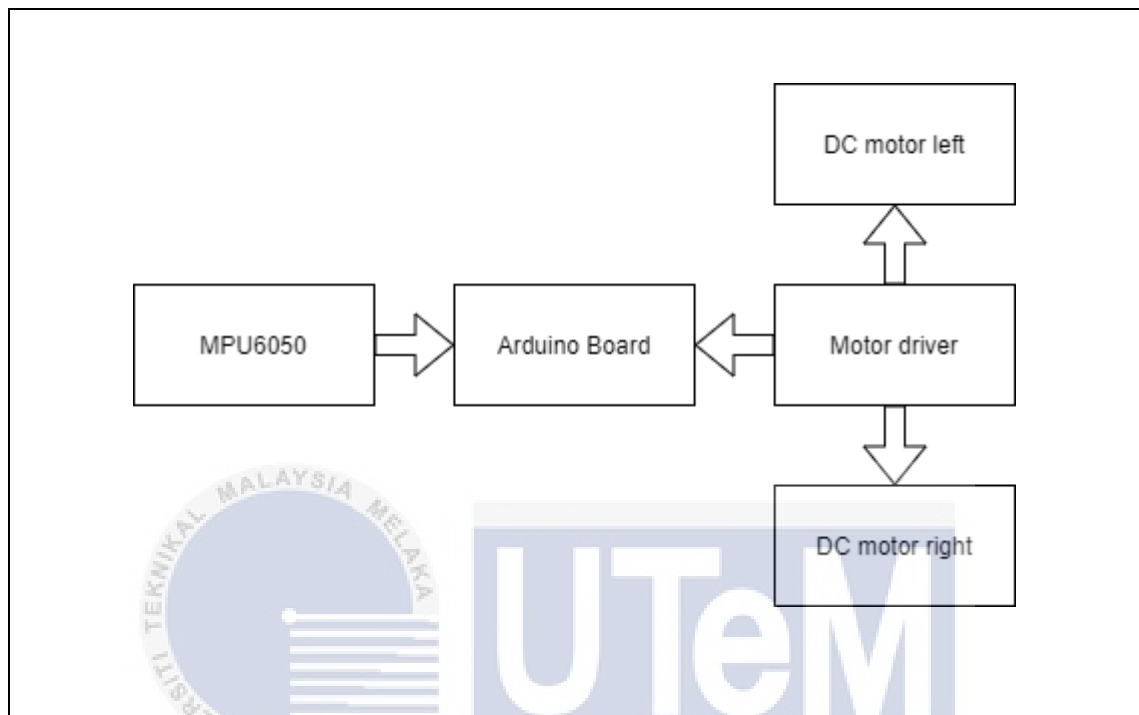
APPENDIX B: Schematic of Arduino Uno Microcontroller Board



APPENDIX C: Schematic MPU6050 Board



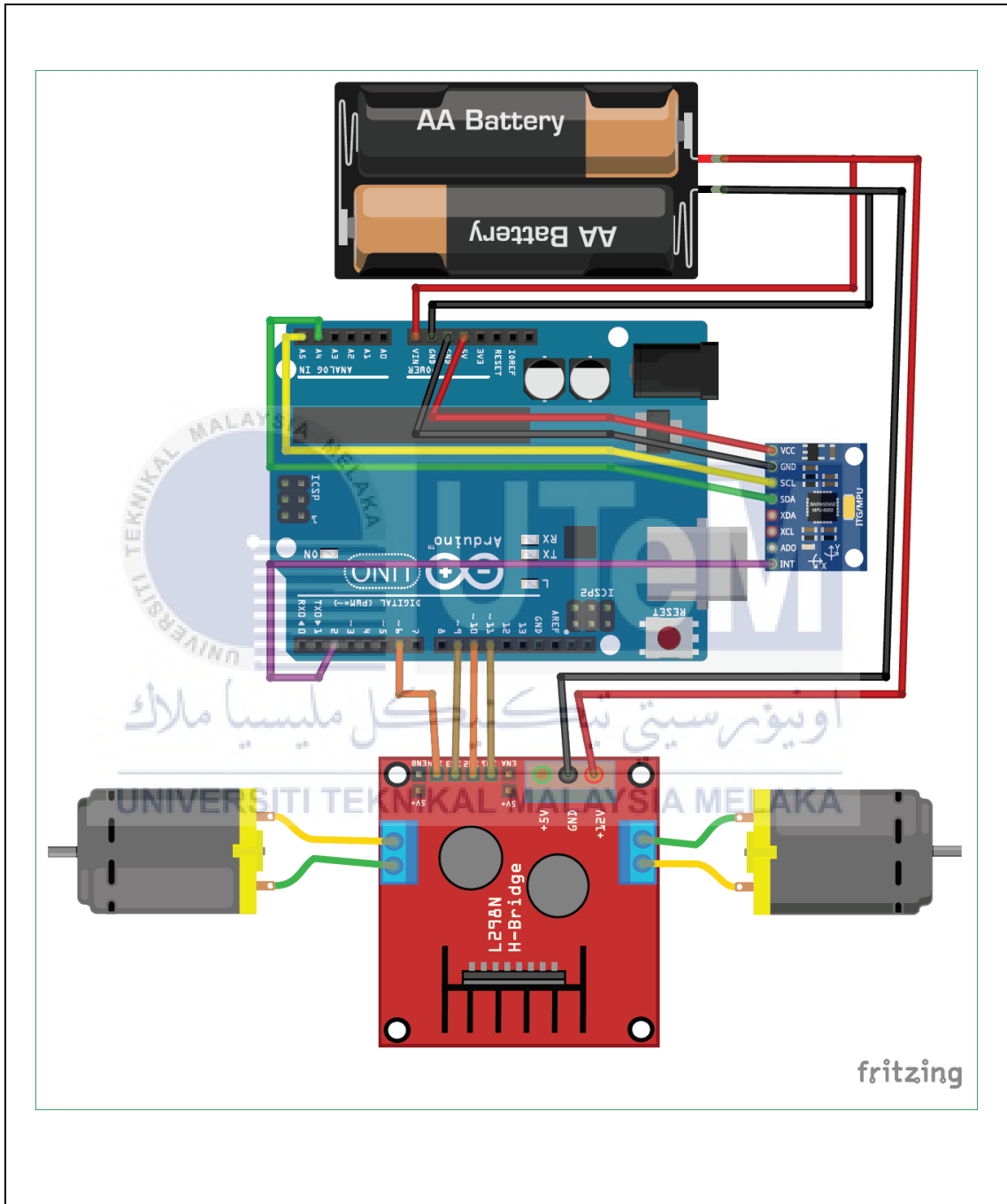
APPENDIX D: Block diagram of the project



اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

APPENDIX E: Circuit diagram of the project





اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA