# PID Control of a two wheel balancing Robot using ESP32/Node MCU and MATLAB

# Table of Content

# **Introduction**

# 1.1. Project Background

# 1.2. Problem Statement

# 1.3. Objectives

# **Software & Tools**

# **Methodology**

# 3.1. Hardware Development

# 3.1.1. Mechanical (Chassis) Assembly

# 3.1.2. Selection of Electronic Components

# 3.1.3. Placement of Electronic Components on the Robot

# 3.1.4. Hardware Connections of the Robot

# 3.1.5. Power Management of the Robot

# 3.2. Software Development

# 3.2.1. Simulink 3D Modeling

# 3.2.2. Generation of Transfer Function

# 3.3. Implementation & Results

# 3.3.1. Calibration of MPU6050

# 3.3.2. Testing of DC Motors

# 3.3.3. Programming the Self-Balancing Robot

# **Conclusion**

# **References**

# **BOM**

# **Annexure**

# Arduino IDE Simple Code for Balancing Self-Balancing Robot

# Code to Generate Transfer Function of Robot

# DC Motors Testing Code

# MPU6050 Calibration Code

# Introduction

## Project Background

The field of robotics has advanced significantly in recent years, with a growing focus on creating autonomous systems that are able to carry out challenging tasks. Self-balancing robot design and implementation is one such area of study. These robots have drawn attention for their potential uses in industry, healthcare, and entertainment, all of which are influenced by the human capacity to maintain balance.

## Problem Statement

# The problem statement of this project is to design and build a self-balancing robot that can transmit sensor data to a MATLAB/Simulink interface for real-time monitoring and control. The primary challenges in keeping stable balance are the deployment of strong control algorithms, designing a user interface for smooth interaction with the robot, and selecting and integrating the right hardware components.

# Objective

This project's main goal is to design, construct, and demonstrate a self-balancing robot.. In order to achieve this goal, the following goals have been determined:

• Chassis build & design

• Balanced Self Balancing Robot

• Wireless Communication of MATLAB GUI & Self Balancing Robot (esp32)

# Software & Tools

# • MATLAB & Simulink

# • Arduino IDE

# Methadology

# The whole project is divided into two main parts, one is software and the other one is hardware part.

# 

# 3.1. **Hardware Development**

#### Mechanical (Chassis) Assembly

# First of all, chassis of Self Balancing Robot is assembled. All the parts of Self balancing Robot are assembled one by one, that can be seen in vlog video. The robot chassis consists of three acrylic plates, nuts etc.

# 

# 

# Selection of Electronic Components

# Electronic Components selection is the important part of any project. The selection of components depends on the desired application. The components we carefully chosen for Self Balancing Robot were according to the desired objectives of this project. The table of components below, shows component name, it’s picture and gives brief description.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr#** | **Components** | **Picture** | **Description** | **Quantity** |
| 1 | ESP32 |  | ESP32 is a powerful micro-controller with Wi-Fi and Bluetooth features. | 1 |
| 2 | MPU6050 |  | It’s a sensor that includes 3-axis accelerometer and 3 axis gyroscope. | 1 |
| 3 | L298N Motor Driver |  | It’s a Dual H-Bridge motor driver that’s compatible with BO DC motors. | 1 |
| 4 | DC motor |  | It’s a commonly used DC motor in robotics, hobby projects etc. | 1 |
| 5 | Rocker SPST Switch |  | It’s a Single Pole Single Throw Switch commonly used in various applications. | 1 |
| 6 | 18650 Cells |  | These are rechargeable lithium-ion cells (3.7V each). | 2 |
| 7 | Miscellaneous Items |  | Wires, soldering iron, screwdrivers etc. | 1 |

# The choice of components can vary from person to person and on the desired objectives/results of the project. In this case, we chosen the BO DC motors instead of stepper motors (that are commonly used for Self Balancing Robot) because DC motor consume less power, provide continuous rotation and are easy to be controlled as compared to stepper motor.

# 

# Placement of Electronic Components on Self Balancing Robot

# All electronic components are mounted on chassis of Self Balancing Robot after careful analysis of weight distribution by placing components in different arragement/configuration on chassis of robot.

# 

# 

# The MPU6050 sensor is mounted on the top of the robot on breadboard. The battery holder is fixed on bottom side of the base of robot.

# 

# The esp32 is placed in the second floor/plate of robot. And the motor driver & spst switch are fixed on the top side of base of chassis.

# These placements can be visualized in pictures and in video (attached).

# Hardware Connections of Self Balancing Robot

# 

# **ESP32:**

# The Vin, GND, SDA (GPIO21) and SCL (GPIO 22) pins of esp32 is connected to the 5V output of the motor driver, common ground, SDA pin of MPU6050 and SCL pin of MPU6050 respectively.

# **L298N Motor Driver:**

# The IN1, IN2, IN3, IN4, ENA (Enable pin of Motor1) and ENB (Enable pin of Motor2) pins of motor driver are connected to the GPIO25, GPIO26, GPIO27, GPIO14, GPIO12, GPIO13 pins of MPU6050 respectively.

# The Motor Driver is getting power from battery (consists of Two 18650 cells 3.7V of each).

# **MPU6050:**

# Vcc, GND, SDA and SCL pins of MPU6050 are connected to 3.3V, common ground, GPIO21 and GPIO22 pin of esp32 respectively.

# **SPST Rocker Switch:**

# It’s one terminal is connected to the Vin terminal of motor driver and the other is connected to the positive terminal of battery.

# Power Management of Robot

# The battery (consists of Two 18650 cells 3.7V of each) gives power to the motor driver. The motor driver 5V output pin is providing power to the esp32 as I can’t provide power directly from battery to esp32. The mpu6050 is powered by esp32 3.3V pin.

# 3.2. Software Part

# 3.2.1. Simulink 3D Modelling

# In Simulink, the Simscape MultibodyTM environment is used to represent the self-balancing robot in three dimensions. we demonstrated the dynamic behaviour of the self-balancing robot by building a rigid body model with the right proportions and mass distribution, and by including revolute joints for the wheels and a prismatic joint for the chassis.

# 

# 3.2.2. Generation of Transfer Function of Robot

# The transfer function is generated from the general state space equations of inverted pendulum after putting my Self Balancing Robot specifications. The general state space equations of inverted pendulum are:

# 

# 

# Here,

# M = Mass of the Cart

# m = Mass of the Pendulum

# l = Length of the Pendulum

# g = Gravity = 9.81ms-1

# The Self Balancing Robot specifications (Dimensions etc) are:

# Length of each chassis plate/floor: 12 cm

# Length of robot (ground chassis including tires): 17.3 cm

# Width of each tire: 2.5 cm

# Diameter of each tire: 6 cm

# Length of robot from ground: 9.5 cm

# Length between each two plates: 3.5 cm

# Width of chassis: 7.9 cm

# Weight of robot: 400 grams

# These are calculated and measured accurately.

# By using the robot specifications and general state space equations of inverted pendulum, a MATLAB code was written. This MATLAB code generates a transfer function using “tf” command. The code snippet is pasted below and the code is pasted in code folder.

# 

# 3.3. Implementation & Results

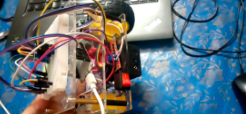
# 3.3.1. Calibration of MPU6050

# MPU6050 is calibrated for accurate results. The code is uploaded by Arduino IDE to esp32 for MPU6050 calibration. The calibration code is attached in the annexure of report. MPU6060 is calibrated by keeping the robot in upright position for accurate calibration because the balancing of Self Balancing Robot heavily depends on MPU6050 sensor.



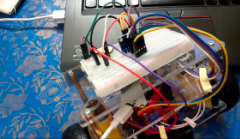
# 3.3.2. Testing of DC Motors

DC motors were tested. Both DC motors were operating at same speed correctly. The code that was used for testing DC motors is attached below in annexure of report. Motor testing part can be seen in video.



# 3.3.2. Programming Self Balancing Robot

The Self Balancing Robot is programmed by Arduino IDE. The code for balancing Self Balancing Robot is uploaded to esp32.



**Code Explantion:**

* Pin Assignments, PID Parameters, Sample Time, and Motor Speed Limits

Pin assignments for motor control and sensor data

PID parameters for pitch and yaw control (these values are adjusted by hidden trial method)

Sample time for sensor data and motor control

Motor speed limits for safe operation

* Libraries for PID and MPU6050 Sensor

PID library for pitch and yaw control

MPU6050 library for sensor data

* Global Variables for Interrupt Handling, PID, Motor, and Sensor Data

Global variables for interrupt handling

Global variables for PID control

Global variables for motor control

Global variables for sensor data

* dmpDataReady() - ISR for New Data from MPU6050 Sensor

Interrupt Service Routine for new data from MPU6050 sensor

* setupMotors() - Sets Up Motor Pins

Sets up motor pins for control

* setupPID() - Sets Up PID for Pitch and Yaw

Sets up PID for pitch and yaw control

* setupMPU() - Initializes MPU6050, Configures Offsets, and Enables DMP

Initializes MPU6050 sensor

Configures offsets for sensor data

Enables DMP for quaternion calculations

* PID, Sensor, and Motor Setup Functions Called by Setup Function

PID setup function

Sensor setup function

Motor setup function

* Main Loop: Computes Control Outputs, Reads Sensor Data, Translates Quaternion to Euler Angles, and Modifies Motor Speeds

Computes control outputs for pitch and yaw

Reads sensor data from MPU6050

Translates quaternion to Euler angles

Modifies motor speeds based on PID outputs

* rotateMotor() - Sets Motor Direction and Speed

Sets motor direction and speed based on input speeds of both motors

This is the brief description of code. The complete code can be viewed in annexure of report.

# 4. Conclusion

This project completed successfully.

**5. References**

* <https://chatgpt.com/>
* <https://youtu.be/UTfhymg9We8?si=uomPcAwZwuxt7ygb>
* <https://youtu.be/nV-S86kX8dY?si=UpatMg52diFdvT9H>
* <https://youtu.be/AitCKcyjHuQ?si=Q4ejN0gVhqrHxySI>
* <https://youtu.be/vxw1DjIAfzI?si=2gU_lA16s_XMSJNO>
* [MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink](https://www.mathworks.com/)
* [Set up and Configure ESP32 Hardware - MATLAB & Simulink (mathworks.com)](https://www.mathworks.com/help/matlab/supportpkg/configure-setup-for-esp32-hardware.html)
* [ESP32 WiFi Tutorial & Library Examples (Arduino IDE) (deepbluembedded.com)](https://deepbluembedded.com/esp32-wifi-library-examples-tutorial-arduino/)
* [ESP32 WiFiMulti Connect to the Strongest Wi-Fi Network | Random Nerd Tutorials](https://randomnerdtutorials.com/esp32-wifimulti/)
* <https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html>
* <https://espressif.github.io/arduino-esp32/package_esp32_index.json>
* <https://forum.arduino.cc/t/esp-32-board-not-installing-libraries-are-not-downloading/976859/2>
* <https://dl.espressif.com/dl/package_esp32_index.json>

**6. BOM**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **BOM level** | **BOM Level** | | **Part name** | **Description** | **Quantity** | **Cost**  **(pkr)** |
| 1 | 2 | | ESP32 | Microcontroller with Wi-Fi and Bluetooth | 1 | 1100 |
| 2 | 2 | | MPU6050 | 6-axis accelerometer and gyroscope | 1 | 300 |
| 3 | 2 | | L298N Motor Driver | Dual H-Bridge motor driver | 1 | 400 |
| 4 | 1 | | Chassis+ DC motor | Three-floor design+ 6V 150RPM DC motors | 1 | 2000 |
| 5 | 2 | | |  | | --- | | Rocker SPST Switch |  |  | | --- | |  | | Single pole single throw switch | 1 | 50 |
| 6 | 2 | | 18650 Cells | Rechargeable lithium-ion batteries | 2 | 600 |
| 7 | 2 | | Miscellaneous Items | Wires, soldering iron, screwdrivers, etc. | 1 | 500 |
|  | | Total Cost | | | 4950 | |

**7. Annexure**

**7.1. Arduino IDE simple code for balancing Self Balancing Robot**

#include <PID\_v1.h>

#include "I2Cdev.h"

#include "MPU6050\_6Axis\_MotionApps20.h"

#if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE

    #include "Wire.h"

#endif

MPU6050 mpu;

#define INTERRUPT\_PIN 2

bool dmpReady = false;

uint8\_t mpuIntStatus;

uint8\_t devStatus;

uint16\_t packetSize;

uint16\_t fifoCount;

uint8\_t fifoBuffer[64];

Quaternion q;

VectorFloat gravity;

float ypr[3];

VectorInt16 gy;

volatile bool mpuInterrupt = false;

void dmpDataReady() {

    mpuInterrupt = true;

}

#define PID\_MIN\_LIMIT -255

#define PID\_MAX\_LIMIT 255

#define PID\_SAMPLE\_TIME\_IN\_MILLI 10

#define SETPOINT\_PITCH\_ANGLE\_OFFSET -2.2

#define PITCH\_ANGLE\_TOLERANCE 3

#define MIN\_ABSOLUTE\_SPEED 10

#define MOTOR\_SPEED 100

double setpointPitchAngle = SETPOINT\_PITCH\_ANGLE\_OFFSET;

double pitchGyroAngle = 0;

double pitchPIDOutput = 0;

double setpointYawRate = 0;

double yawGyroRate = 0;

double yawPIDOutput = 0;

#define PID\_PITCH\_KP 30

#define PID\_PITCH\_KI 100

#define PID\_PITCH\_KD 0.1

#define PID\_YAW\_KP 0.5

#define PID\_YAW\_KI 0.5

#define PID\_YAW\_KD 0

PID pitchPID(&pitchGyroAngle, &pitchPIDOutput, &setpointPitchAngle, PID\_PITCH\_KP, PID\_PITCH\_KI, PID\_PITCH\_KD, DIRECT);

PID yawPID(&yawGyroRate, &yawPIDOutput, &setpointYawRate, PID\_YAW\_KP, PID\_YAW\_KI, PID\_YAW\_KD, DIRECT);

int enableMotor1 = 12;

int motor1Pin1 = 25;

int motor1Pin2 = 26;

int motor2Pin1 = 27;

int motor2Pin2 = 14;

int enableMotor2 = 13;

void setupPID()

{

  pitchPID.SetOutputLimits(PID\_MIN\_LIMIT, PID\_MAX\_LIMIT);

  pitchPID.SetMode(AUTOMATIC);

  pitchPID.SetSampleTime(PID\_SAMPLE\_TIME\_IN\_MILLI);

  yawPID.SetOutputLimits(PID\_MIN\_LIMIT, PID\_MAX\_LIMIT);

  yawPID.SetMode(AUTOMATIC);

  yawPID.SetSampleTime(PID\_SAMPLE\_TIME\_IN\_MILLI);

}

void setupMotors()

{

  pinMode(enableMotor1, OUTPUT);

  pinMode(motor1Pin1, OUTPUT);

  pinMode(motor1Pin2, OUTPUT);

  pinMode(enableMotor2, OUTPUT);

  pinMode(motor2Pin1, OUTPUT);

  pinMode(motor2Pin2, OUTPUT);

  rotateMotor(0, 0);

}

void setupMPU()

{

  Wire.begin();

  Wire.setClock(400000);

  mpu.initialize();

  pinMode(INTERRUPT\_PIN, INPUT);

  devStatus = mpu.dmpInitialize();

  mpu.setXAccelOffset(1708);

  mpu.setYAccelOffset(-5781);

  mpu.setZAccelOffset(1768);

  mpu.setXGyroOffset(72);

  mpu.setYGyroOffset(-6);

  mpu.setZGyroOffset(-37);

  if (devStatus == 0)

  {

      mpu.setDMPEnabled(true);

      mpuIntStatus = mpu.getIntStatus();

      dmpReady = true;

      packetSize = mpu.dmpGetFIFOPacketSize();

  }

}

void setup()

{

  setupMotors();

  setupMPU();

  setupPID();

}

void loop()

{

  if (!dmpReady) return;

  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer))

  {

    mpu.dmpGetQuaternion(&q, fifoBuffer);

    mpu.dmpGetGravity(&gravity, &q);

    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    mpu.dmpGetGyro(&gy, fifoBuffer);

    yawGyroRate = gy.z;

    pitchGyroAngle = ypr[1] \* 180/M\_PI;

    pitchPID.Compute();

    yawPID.Compute();

    if (abs(pitchPIDOutput) > PITCH\_ANGLE\_TOLERANCE) {

      rotateMotor(pitchPIDOutput, pitchPIDOutput);

    } else {

      rotateMotor(0, 0); // Stop motors

    }

  }

}

void rotateMotor(int speed1, int speed2)

{

  if (speed1 < 0)

  {

    digitalWrite(motor1Pin1, LOW);

    digitalWrite(motor1Pin2, HIGH);

  }

  else if (speed1 >= 0)

  {

    digitalWrite(motor1Pin1, HIGH);

    digitalWrite(motor1Pin2, LOW);

  }

  if (speed2 < 0)

  {

    digitalWrite(motor2Pin1, LOW);

    digitalWrite(motor2Pin2, HIGH);

  }

  else if (speed2 >= 0)

  {

    digitalWrite(motor2Pin1, HIGH);

    digitalWrite(motor2Pin2, LOW);

  }

  analogWrite(enableMotor1, MOTOR\_SPEED);

  analogWrite(enableMotor2, MOTOR\_SPEED);

}

**7.2. Code to generate Transfer Function of Robot**

M = 0.4;

m = 0.4/3;

l = 0.245;

g = 9.81;

A = [0 1 0 0;

((M+m)\*g)/(M\*l) 0 0 0;

0 0 0 1;

-(m\*g)/M 0 0 0];

B = [0 ;

-1/(M\*l);

0;

1/M];

C = [l 0 1 0];

D = 0;

StateSpace = ss(A, B, C, D);

TransferFunction = tf(StateSpace)

**7.3. DC Motors Testing Code**

const int IN1 = 25;

const int IN2 = 26;

const int IN3 = 27;

const int IN4 = 14;

const int EN1 = 12;

const int EN2 = 13;

void setup() {

  pinMode(EN1, OUTPUT);

  pinMode(IN1, OUTPUT);

  pinMode(IN2, OUTPUT);

  pinMode(EN2, OUTPUT);

  pinMode(IN3, OUTPUT);

  pinMode(IN4, OUTPUT);

  digitalWrite(IN1,HIGH);

  digitalWrite(IN2,LOW);

  analogWrite(EN1, 255);

  digitalWrite(IN3,HIGH);

  digitalWrite(IN4,LOW);

  analogWrite(EN2, 255);

  delay(2000);

 digitalWrite(IN1,LOW);

  digitalWrite(IN2,LOW);

  digitalWrite(IN3,LOW);

  digitalWrite(IN4,LOW);

  delay(2000);

  digitalWrite(IN1,LOW);

  digitalWrite(IN2,HIGH);

  analogWrite(EN1, 255);

  digitalWrite(IN3,LOW);

  digitalWrite(IN4,HIGH);

  analogWrite(EN2, 255);

  delay(2000);

  digitalWrite(IN1,LOW);

  digitalWrite(IN2,LOW);

  digitalWrite(IN3,LOW);

  digitalWrite(IN4,LOW);

  delay(2000);

  digitalWrite(IN1,HIGH);

  digitalWrite(IN2,LOW);

  analogWrite(EN1, 127);

  digitalWrite(IN3,HIGH);

  digitalWrite(IN4,LOW);

  analogWrite(EN2, 127);

  delay(2000);

  digitalWrite(IN1,LOW);

  digitalWrite(IN2,LOW);

  digitalWrite(IN3,LOW);

  digitalWrite(IN4,LOW);

  delay(2000);

  digitalWrite(IN1,LOW);

  digitalWrite(IN2,HIGH);

  analogWrite(EN1, 127);

  digitalWrite(IN3,LOW);

  digitalWrite(IN4,HIGH);

  analogWrite(en2, 127);

  delay(2000);

  digitalWrite(IN1,LOW);

  digitalWrite(IN2,LOW);

  digitalWrite(IN3,LOW);

  digitalWrite(IN4,LOW);

  delay(2000);

}

void loop() {

}

**7.4. MPU6050 Caliberation Code**

This MPU6050 Caliberation code is taken from a youtube video. The link of that video is provided in references.

#include <Wire.h>

#include <MPU6050\_tockn.h>

MPU6050 SENSOR(Wire);

void setup()

{

  Serial.begin(9600);

  Wire.begin();

  SENSOR.begin();

  Serial.println("MPU6050 SENSOR IS INITIALIZED.");

  calibrateMPU6050();

}

void loop()

{

}

void calibrateMPU6050()

{

  const int NoOfSamples = 1000;

  int32\_t Gyroscope\_Xaxis\_Offset = 0, Gyroscope\_Yaxis\_Offset = 0, Gyroscope\_Zaxis\_Offset = 0;

  int32\_t Accelerometer\_Xaxis\_Offset = 0, Accelerometer\_Yaxis\_Offset = 0, Accelerometer\_Zaxis\_Offset = 0;

  Serial.println("MPU6050 CALIBERATION IS STARTED:");

  for (int i = 0; i < NoOfSamples; i++)

  {

    SENSOR.update();

    Gyroscope\_Xaxis\_Offset += SENSOR.getGyroX();

    Gyroscope\_Yaxis\_Offset += SENSOR.getGyroY();

    Gyroscope\_Zaxis\_Offset += SENSOR.getGyroZ();

    Accelerometer\_Xaxis\_Offset += SENSOR.getAccX();

    Accelerometer\_Yaxis\_Offset += SENSOR.getAccY();

    Accelerometer\_Zaxis\_Offset += SENSOR.getAccZ();

    delay(10);

  }

  Gyroscope\_Xaxis\_Offset /= NoOfSamples;

  Gyroscope\_Yaxis\_Offset /= NoOfSamples;

  Gyroscope\_Zaxis\_Offset /= NoOfSamples;

  Accelerometer\_Xaxis\_Offset /= NoOfSamples;

  Accelerometer\_Yaxis\_Offset /= NoOfSamples;

  Accelerometer\_Zaxis\_Offset /= NoOfSamples;

  Serial.println("MPU6050 SENSOR CALIBERATION IS COMPLETED");

  Serial.print("GYROSCOPE X-axis OFFSET: ");

  Serial.println(Gyroscope\_Xaxis\_Offset);

  Serial.print("GYROSCOPE Y-axis OFFSET: ");

  Serial.println(Gyroscope\_Yaxis\_Offset);

  Serial.print("GYROSCOPE Z-axis OFFSET: ");

  Serial.println(Gyroscope\_Zaxis\_Offset);

  Serial.print("ACCELEROMETER X-axis OFFSET: ");

  Serial.println(Accelerometer\_Xaxis\_Offset);

  Serial.print("ACCELEROMETER Y-axis OFFSET: ");

  Serial.println(Accelerometer\_Yaxis\_Offset);

  Serial.print("ACCELEROMETER Z-axis OFFSET: ");

  Serial.println(Accelerometer\_Zaxis\_Offset);

}