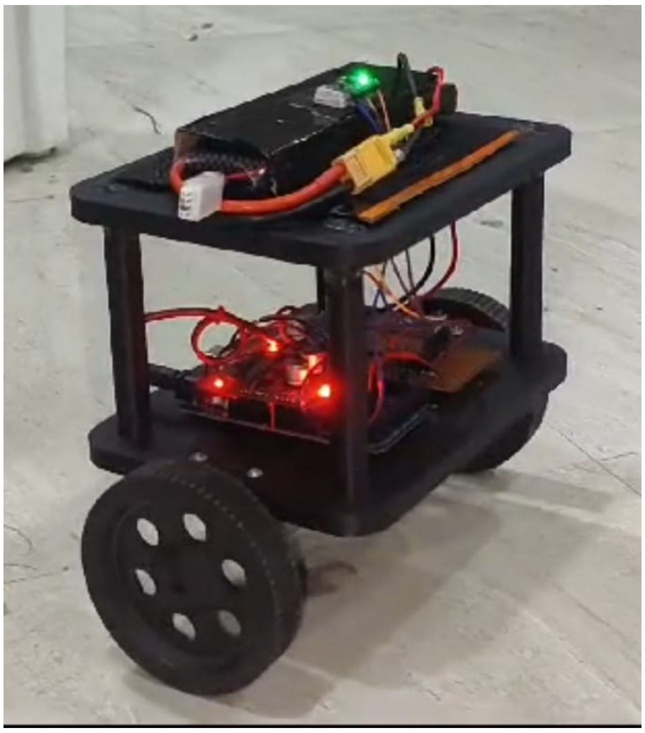
Robotics Club
Summer Project 2024

GYROBOT:
Two Wheeled Self-Balancing Robot



Mentors
Yuval Bansal
Rishi Baghel
Sanskar Yaduka

Mentees
Arbaaz Tanveer
Ayush Kumar
Divyesh Chaudhari
Golden Kumar
Ishita Porwal
Mansi Ghodke
Prutha Vinay
Sanvi Jain
Vedika Taparia
Yashwi Agarwal

Abstract

This report provides a comprehensive overview of our robotics project, which involves the design, implementation, and testing of a self-balancing robot. It is a two-wheel vehicle which is highly susceptible to tip over about one axis, thus its electrical, mechanical and software structures have to be integrated to enable it to balance itself. The robot design uses the inverted pendulum principle. The wheels of the robot are driven by DC motors. Information about the angle of the device relative to the ground is obtained from a 6DOF IMU sensor. Then it uses the feedback from a PID algorithm to generate position control signals to restore the balance and bring it back to its original vertical position.

Contents

1 Introduction 2

2 Literature Review 4

3 Mechanical Design 6

4 Electrical Design 10

5 Software Design 14

6 Testing 21

7 Results and Conclusion 22

Chapter 1

Introduction

In recent times, self-balancing robots have gained significant recognition in various applications, particularly as human transporters, due to their reliance on electronic devices and embedded control systems. Making use of their extensive stability and maneuverability, self-balancing robots can be used to solve various real world problems. In general, two-wheeled robots can offer superior mobility in confined spaces in comparison to humanoid robots. Their speed also makes them one of the best and most popular choices for mobile robot platforms.

This project incorporates a complex electrical and software subsystem along with a finely tuned control system, all in place to implement building a two-wheeled self- balancing robot. These robots function on the basis of the Inverted Pendulum principle that requires precise real-time tilt angle measurements to maintain balance. The Inverted Pendulum is a classic and complex control problem due to its non-linear and unstable nature, with one input signal and multiple output signals. Maintaining the pendulum's balance in an inverted position demands external force application.

Elaborating more on how exactly this works- The inverted pendulum consists of a rod with a mass at the top, hinged at the bottom (the pivot point). What makes this one of the most complex problems in the field of robotics is the fact that it's centre of mass is placed above the pivot point which places the system in a state of unstable equilibrium because any small deviation from the upright position causes the pendulum to fall unless an external force is applied in order to prevent the same. The primary goal is to apply forces at the base (the pivot) to keep the pendulum balanced upright. This requires continuous feedback and adjustment based on the pendulum's position and velocity.

Applying this principle to our bot, in order to stay balanced, the two-wheeled robot relies on the movements of its wheels. When an unexpected disturbance occurs, the robot uses wheel adjustments and body tilting to maintain its balance. However, we need a proper mechanism that can sense even the slightest of disturbances in the state of our bot. For this we employ sensors that measure the angle of our bot and it's angular velocity. Gyroscopes and accelerometers within these sensors measure these parameters and detect deviations from the vertical axis. If the disturbance exceeds the robot's capability to respond, it will inevitably lose it's stability. To help with this, the robot is equipped with a control algorithm such as the Proportional-Integral-Derivative (PID) control algorithm to enhance balance with minimal movement. When a disturbance impacts the robot, a disturbance observer estimates it, and the algorithm acts and compensates for it. This module allows the robot to maintain balance with only slight wheel movements.

In this paper, we report the major components and working of our self-balancing robot and give an insight as to why the components were chosen and reasonings behind the working of the bot. Our bot consists of the Arduino MEGA micro-controller unit based on the ATMega2560 processor, a 6-axis MPU6050 sensor consisting of a gyroscope and an accelerometer, a VNH2SP30 motor driver to control the working of our 2 DC geared motors, a power down bug converter, a power supply, and an HC05 Bluetooth module to help us communicate with the bot.

The following sections provide an in-depth analysis on the various subsystems of our bot including the Mechanical, Electrical and Software sub-systems and an overview of the basic progression and initial testing of the robot.

Chapter 2

Literature Review

Self-balancing robots have the capability to maintain stability on two wheels using advanced control systems. They use the inverted pendulum principle, where a platform with two wheels continuously adjusts its position to prevent falling over. PID (Proportional-Integral-Derivative) algorithm is essential to stabilize the robot by adjusting motor speeds based on sensor feedback. To determine the most suitable design and control systems for the self-balancing robot, current designs and research were studied. The integration of required mechanical, electrical and software components was studied to achieve stable self-balancing. Key ideas for the bot are its mechanical design, sensor integration, like 6DOF IMU (Inertial Measurement Unit), and PID control algorithms.

Inverted Pendulum Systems

The stability and control of self-balancing robots is based on the inverted pendulum principle. Using this principle, these robots can achieve precise balance and maneuverability, making them suitable for a wide range of applications.
The inverted pendulum concept involves maintaining the equilibrium of a pendulum on a moving base. By adjusting the position and velocity of the wheels in response to sensor feedback, these robots dynamically stabilize themselves against external disturbances. Key challenges include minimizing oscillations and optimizing power consumption.

IMU Sensors

Inertial Measurement Units (IMUs) are necessary to get real-time data of the orientation and motion of self-balancing robots. A 6DOF (Degrees of Freedom) IMU sensor uses a gyroscope and accelerometer, to get precise estimation of the robot's orientation in space. Accurate sensor data is essential for effective control

algorithms in autonomous robots. Challenges associated with IMU sensors are its calibration and noise reduction of sensor data.
The MPU6050, a popular 6DOF IMU sensor, integrates a 3-axis accelerometer and a 3-axis gyroscope, which allows for accurate measurement of linear acceleration and angular velocity along multiple axes. The MPU6050's compact size, low power consumption and digital interface makes it suitable for integration into small-scale robotics applications, including self-balancing robots.

PID Control Algorithms

PID (Proportional-Integral-Derivative) control algorithms are widely used in self-balancing robots to maintain stability by adjusting motor speeds based on feedback from sensors. The PID controller calculates an error signal that represents the difference between desired and actual orientation of the robot, then adjusts the motors to minimize this error.
Effective tuning of PID parameters is essential to achieve stable and responsive control in self-balancing robots. The proportional term adjusts the motor speed proportionally to the error, the integral term is required for accumulated errors over time and the derivative term dampens oscillations by predicting future errors. Studies emphasize the iterative process of PID tuning to achieve optimal performance.

Chapter 3

Mechanical Design

The mechanical design of the bot consists of-

Chassis

The primary chassis of the robot is made of 2 3D-printed PLA (Polylactic Acid) sheets that act as platforms for the various components of the bot. The dimensions of these platforms are 15cm x 20cm x 1 cm each. Four 10cm cylindrical rods of 1.5 cm diameter are placed at the corners of the bottom platform so as to mount the upper platform upon. These rods have been 3D-printed along with the bottom platform and hence are attached to it internally. Whereas, the upper platform has been attached to the rods with the help of glue.

Motors

The bot utilises two DC Johnson Geared motors that have been placed on either side on the bottom side of the bottom platform of the bot. The motors are 12V DC geared motors of 300 RPM speed and 10kg-cm torque. To ensure that the motors are in place on the bottom of the platform, the motors have been attached to a mount which in turn has been screwed on to the bottom via 2 M3 screws for each mount.

Wheels

Two wheels are mounted on the shafts of the two motors via threaded screws provided on the internal plastic rim of the wheel on either side of the bot. The wheels are 2cm in width and 11cm in diameter and have an outer lining of ridged rubber.

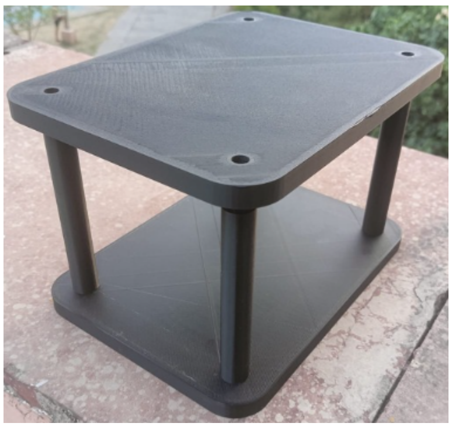


Figure 3.1: Mechanical Components

Electrical Components

The various electrical components of the bot have been placed in the following ways-

1. LiPo Battery: The LiPo battery has been mounted on the topmost platform in order to ensure that the Center of Gravity of our system remains roughly at the middle for better balancing and has been placed inside a cardboard encasing which is taped down with the help of black duct tape.

2. IMU Sensor: The MPU6050, which is the IMU sensor we use in the bot, has been placed on the bottom part of the topmost platform. It is very important to ensure that the sensor is placed at a position nearest to the center of gravity of the model to make it sensitive to even the smallest of deflections.

3. Arduino MEGA: The system utilises the Arduino MEGA micro-controller unit and it's placed on the bottom platform. One of the main reasons of utilizing the Arduino MEGA is because the motor driver can be directly mounted on the unit without the need for excessive soldering.

4. Motor Driver: The motor driver used in this particular robot is the Monster Motor Shield VNH2SP30. This is mounted directly on the Arduino MEGA and hence is placed on the bottom platform.

5. ZeroPCB: The ZeroPCB is placed besides the Arduino MEGA and consists of an HC05 Bluetooth module, a 12V-7V power down Buck Converter and a connector that connects the MPU6050 to the Zero PCB.

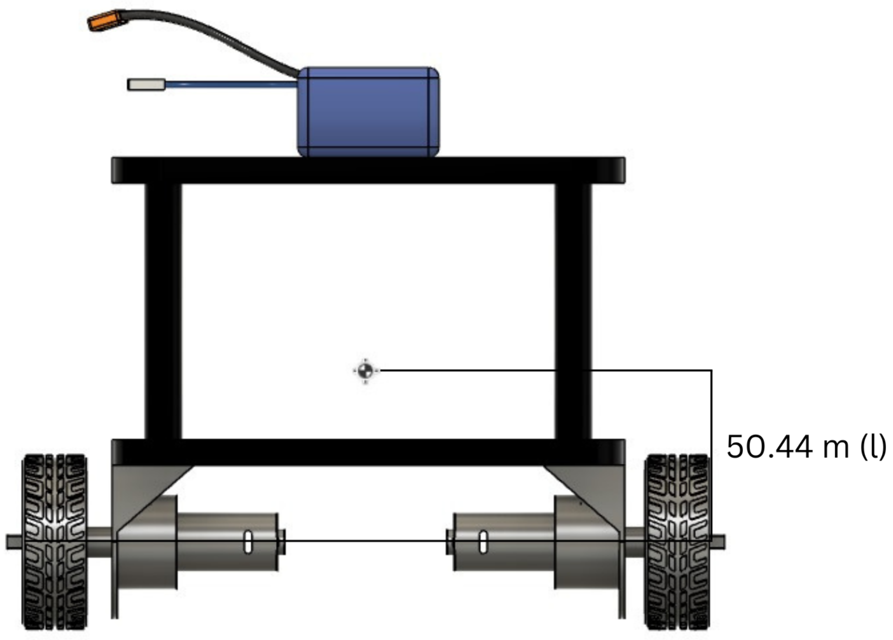


Figure 3.2: AutoCAD Model of the Bot

Torque Calculations

The CAD model of our bot given below shows that the model can be regarded simply as an inverted pendulum system on two wheels. The small pointer lying between the two platforms of the bot represents the center of mass of our bot. The center of mass of the system is naturally desired to be positioned higher than the motor shaft because we wish to study the dynamics of a system in a state of unstable equilibrium. The mass (m) of our bot is 1497.369 g, and the maximum recovery angle (θ for our bot is 15°. The center of mass of our wheels lies at 50.44 mm from the motor shaft. If we consider this distance as the height of center of mass (l), we are able to determine the maximum torque that this system will have by using the following formula for the same-

$$t_{max} = m * g * l * sin\theta$$

Therefore, the maximum torque that can be produced by the motors is 0.1918 N-m. However, since there are two motors, each motor is individually capable of producing 0.0959 N-m.

Considering the target speed of the system is 300 RPM, the power generated by the motors can be considered as-

$$Power = t_{max} * w$$

Therefore, the power generated by the system is 3.0128 W.

For calculating the torque (q) generated by friction on the wheels, if we consider the frictional coefficient $(\mu) between the wheels and the ground to be 0.6, and N to be the normal reaction one$

$$q = \mu * N * r$$

After calculations, this torque comes out to be 0.1763 N-m.

Chapter 4

Electrical Design

The electrical design of the self-balancing robot includes the selection, integration and wiring of the electronic components used so that the robot can sense its orientation, process the sensor data, and actuate the motors.

Microcontroller

The robot uses an Arduino Mega microcontroller, responsible for reading sensor data, executing the control algorithm, and sending the required signals to the motor drivers. Arduino MEGA was preferred over Arduino UNO or NANO since the motor driver could be directly mounted on it without additional wiring or soldering. Arduino Mega has an operating voltage range of 7-12 V.

Inertial Measurement Unit (IMU) Sensor

The robot uses the MPU6050 IMU sensor to measure its tilt and angular velocity. The IMU sensor combines a 3-axis accelerometer and a 3-axis gyroscope to provide a 6 Degrees of Freedom (6DOF) measurement to determine the robot's orientation and movement. It uses the I2C communication protocol to send the data to the Arduino.

Motor Driver

To drive the DC motors, VNH2SP30 motor driver is used. It is capable of handling a high current and provides good control over the robot's motors. It has a dual H-bridge configuration, allowing control of 2 motors. VNH2SP30 can control the speed and direction of the motors based on PWM signals from the Arduino Mega. It has an operating voltage range of 5.5-16V.

DC Motors

The robot uses two DC motors, each connected to one of the wheels. They are controlled by the same motor driver and are responsible for the movement of robot and maintaining its balance by adjusting the position. The maximum speed of the motors is about 300RPM.

Power Supply

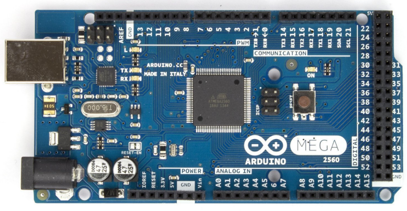
The robot is powered by a rechargeable 12V Li-Po battery pack. It has a capacity of 3500mAh and discharge rate of 35C. The battery supplies power to the Arduino Mega, IMU, motor driver and motors.

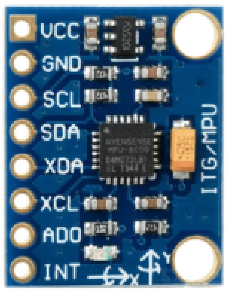
Bluetooth Module

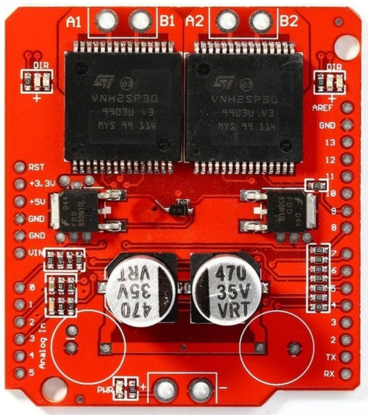
For wireless communication and remote control, an HC-05 Bluetooth module is used, allowing easy pairing with smartphones, tablets or computers, which will be used to send commands to the robot remotely. It uses the UART communication protocol and has a range of upto 10 metres.

Wiring and Connections

The MPU6050 is connected to the Arduino Mega via the I2C bus. The VNH3ASP30 motor driver receives PWM signals from the Arduino Mega's digital pins. The PWM pin controls the motor speed and the DIR pin controls the motor direction. The HC-05 Bluetooth module is connected to the Arduino Mega's UART pins.
A buck converter, HC-05 Bluetooth module and a connector (to connect the MPU6050 fixed on the top platform) are soldered on the zero PCB.


(a) Arduino Mega


(b) MPU6050


(c) VNH2SP30

(d) DC Motor

(e) LiPo Battery

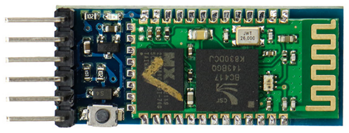

(f) HC-05 Bluetooth Module

Figure 4.1: Electrical Components

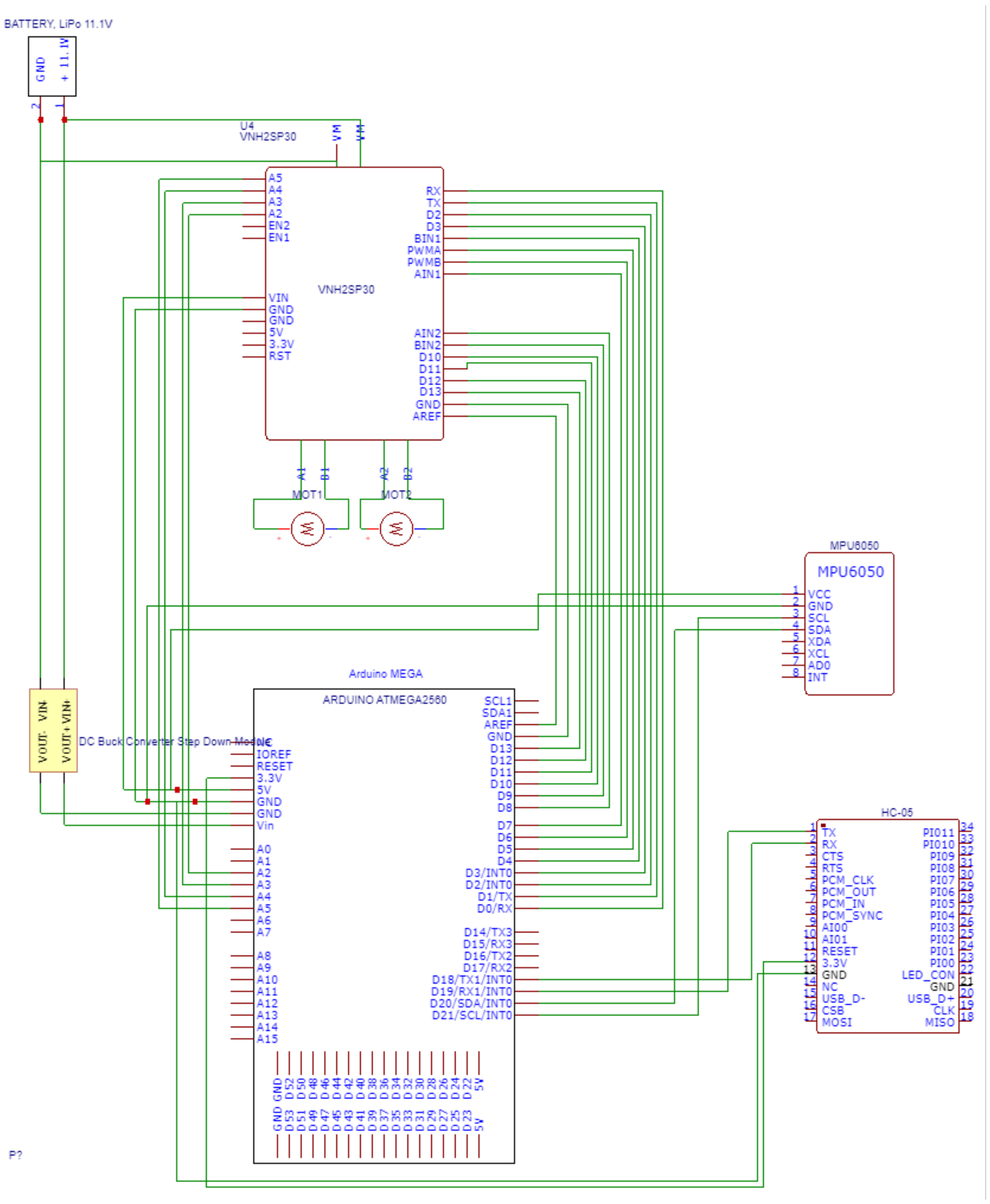


Figure 4.2: Schematic Diagram

Chapter 5

Software Design

The software design of the self-balancing robot includes sensor integration, motor control using a PID control algorithm and a Bluetooth module for communication. The primary components of the software are the MPU6050 sensor to measure orientation, the PID loop to maintain balance and the motor driver to control the motors accordingly.

Sensor Integration

The MPU6050 sensor data is obtained via I2C communication with Arduino Mega. ‘Adafruit MPU6050‘ library is used to interface with the sensor and obtain acceleration and gyroscope data. The Kalman filter is applied to the sensor data to estimate the tilt angle (pitch) of the robot accurately.

PID Control Algorithm

The PID control algorithm is used to calculate the required motor speeds for balancing the robot. The algorithm continuously adjusts the motor speeds based on the error between desired setpoint (upright position) and current tilt angle, and applies the necessary corrections to maintain balance. The PID parameters (K_p, K_i, K_d) are tuned to achieve optimal performance.
The PID controller is initialized with the following parameters: $K_p = 3.0$; $K_i = 0.30$; $K_d = 0.02$
Maximum output speed limits are set from -80 to 80, integrator limits are set from -50 to 50, and the sample time is 0.1 seconds.

Motor Control

The motor driver VNH2SP30 is connected to Arduino Mega and controls the speed of two DC motors using the PWM signals generated by the Arduino. The direction and speed of the motors are controlled based on the output from the PID controller to maintain the robot's balance.

Bluetooth Communication

The Bluetooth module HC-05 is connected to the Arduino for wireless communication. The software initializes the Bluetooth module for communication. Commands received via Bluetooth can be used to adjust the PID parameters or to directly control the robot.

The software design of the self-balancing robot integrates sensor data, a PID control loop, motor control and Bluetooth communication. The MPU6050 sensor provides real-time angle measurements, which are processed by the PID control algorithm to generate motor control signals. The VNH2SP30 motor driver adjusts the motor speeds to maintain balance, while the HC-05 Bluetooth module facilitates remote control and monitoring.

Arduino Code

```
#include <PID_v1.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include <SoftwareSerial.h>

// RX, TX pins for HC-05 Bluetooth module
SoftwareSerial bluetooth(51, 50);

// Variables to store decoded values
int joystickX = 0;
int joystickY = 0;
int forwardBackwardOffset = 0;
int turnOffset = 0;
String receivedString = ""; // For storing incoming
    Bluetooth data

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

#define MIN_ABS_SPEED 30

MPU6050 mpu;
```

```

// MPU control/status vars
bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];

// orientation/motion vars
Quaternion q;
VectorFloat gravity;
float ypr[3];

// PID
double originalSetpoint = 183.5;
double setpoint = originalSetpoint;
double movingAngleOffset = 0.1;
double input, output;

double Kp = 25;
double Kd = 1.7;
double Ki = 270;
PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

double motorSpeedFactorLeft = 0.75;
double motorSpeedFactorRight = 0.48;

// MOTOR CONTROLLER
#define MOTOR1_ENABLE_PIN A0
#define MOTOR2_ENABLE_PIN A1
#define IN1 7
#define IN2 8
#define IN3 9
#define IN4 4
#define ENA 5
#define ENB 6

int currentSpeed = 0;

volatile bool mpuInterrupt = false;
void dmpDataReady() {
    mpuInterrupt = true;
```

```
}

void setup() {
    Serial.begin(115200);
    Serial.println(F("Initializing..."));

    bluetooth.begin(9600);
    Serial.println("Bluetooth receiver ready");

    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        TWBR = 24;
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

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

    devStatus = mpu.dmpInitialize();

    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788);

    if (devStatus == 0) {
        mpu.setDMPEnabled(true);
        attachInterrupt(digitalPinToInterrupt(2),
           dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();
        dmpReady = true;
        packetSize = mpu.dmpGetFIFOPacketSize();
        pid.SetMode(AUTOMATIC);
        pid.SetSampleTime(10);
        pid.SetOutputLimits(-255, 255);
        Serial.println(F("DMP ready!"));
    } else {
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    pinMode(MOTOR1_ENABLE_PIN, OUTPUT);
    pinMode(MOTOR2_ENABLE_PIN, OUTPUT);
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(ENA, OUTPUT);
    pinMode(ENB, OUTPUT);

    digitalWrite(MOTOR1_ENABLE_PIN, HIGH);
    digitalWrite(MOTOR2_ENABLE_PIN, HIGH);
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);

    Serial.println(F("Setup complete."));
}

void processBluetoothData() {
    int commaIndex = receivedString.indexOf(',');
    if (commaIndex > 0) {
        String xString = receivedString.substring(0,
           commaIndex);
        String yString = receivedString.substring(commaIndex
            + 1);

        joystickX = xString.toInt();
        joystickY = yString.toInt();

        movingAngleOffset = (float)joystickY * 1.9/ 255;
        turnOffset = (float)joystickX * 30 / 255;
        setpoint = originalSetpoint + movingAngleOffset;

        Serial.print("Received: X=");
        Serial.print(joystickX);
        Serial.print(", Y=");
        Serial.println(joystickY);
    }
```

```
}

void move(int balanceSpeed, int minAbsSpeed) {
    int leftSpeed = balanceSpeed + turnOffset;
    int rightSpeed = balanceSpeed - turnOffset;

    leftSpeed = constrain(leftSpeed, -255, 255);
    rightSpeed = constrain(rightSpeed, -255, 255);

    digitalWrite(IN1, leftSpeed > 0 ? HIGH : LOW);
    digitalWrite(IN2, leftSpeed > 0 ? LOW : HIGH);
    digitalWrite(IN3, rightSpeed > 0 ? HIGH : LOW);
    digitalWrite(IN4, rightSpeed > 0 ? LOW : HIGH);

    analogWrite(ENA, abs(leftSpeed) * motorSpeedFactorLeft);
    analogWrite(ENB, abs(rightSpeed) * motorSpeedFactorRight
       );
}

void loop() {
    while (bluetooth.available() > 0) {
        char c = bluetooth.read();
        if (c == '.') {
            processBluetoothData();
            receivedString = "";
        } else {
            receivedString += c;
        }
    }

    if (!dmpReady) return;

    if (mpuInterrupt || fifoCount >= packetSize) {
        mpuInterrupt = false;
        mpuIntStatus = mpu.getIntStatus();
        fifoCount = mpu.getFIFOCount();

        if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
            mpu.resetFIFO();
            Serial.println(F("FIFO overflow!"));
        } else if (mpuIntStatus & 0x02) {
            while (fifoCount < packetSize) fifoCount = mpu.
               getFIFOCount();

            mpu.getFIFOBytes(fifoBuffer, packetSize);
            fifoCount -= packetSize;

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

            pid.Compute();
            move(output, MIN_ABS_SPEED);
        }
    }
}
```

Chapter 6

Testing

The various aspects of the bot that required testing and development are as follows-

Calibration of IMU Sensor

The calibration of IMU sensor is done to ensure that the sensor readings are accurate and reflect the true measurements of motion and orientation. Many-a-times the case could be that manufacturing processes can introduce slight differences between individual sensors, and calibration helps compensate for these variances. Calibration aligns the sensor's axes with the real-world axes, ensuring that the measurements correspond accurately to the intended directions. In our case, the sensor calibration was done by setting the position of the bot along the z axis and checking if acceleration is equal to g.

Offset for the balance position

In a self-balancing robot, the equilibrium position (where the robot remains upright) might not align perfectly with the zero reading of the sensors. Setting an offset ensures that the control system recognizes the true balanced position. Additionally, external factors like uneven surfaces or slight inclines can affect the robot's balance. An offset helps the robot adapt to these conditions, improving stability. In the case of our bot, the MPU reading at the balance position was taken and accordingly, an offset was set in the code.

Chapter 7

Results and Conclusion

The final bot was completed with a chassis of PLA, DC geared motors, LiPo battery, MPU6050 IMU sensor, Arduino MEGA micro-controller, VNH2SP30 motor driver and a Zero PCB consisting of HC05 Bluetooth module for communication and a Buck converter. The PID control algorithm ensures the movement of the bot in two basic directions : front and back. The inclusion of the Kalman Filter has allowed successful elimination of gyroscopic drift. The robot is designed to balance itself on two wheels and has the capability to recover from any external forces that may cause it to lose balance. But it is still not perfect - the few issues, including the minor wobble, the asymmetrical motor speeds, and limited recovery angle for which the bot gets upright are small problems that can be fixed in a future update.

We came into this project expecting to build a two-wheeled robot that would balance itself with the help of an IMU. It took a good amount of work, and we encountered significant challenges, but we met our expectations and achieved our goal. After building the chassis, designing and testing the circuits, writing the software, and tuning the PID coefficients, we were able to successfully balance the robot on the two wheels.

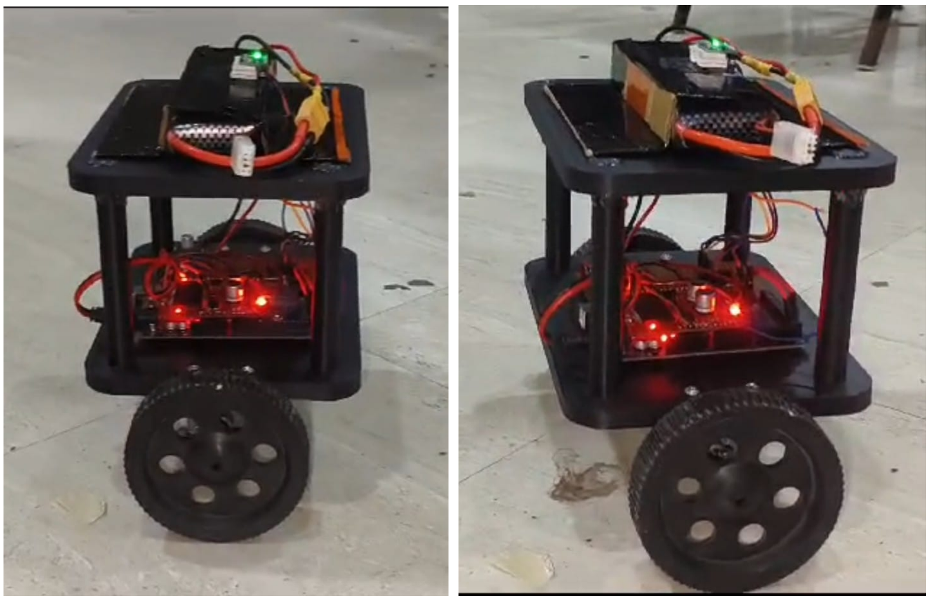


Figure 7.1: Final self-balanced bot