SHIV NADAR UNIVERSITY
CHENNAI

CS3684 - Introduction to Robotics
Continuous Internal Assessment (CIA) – 1

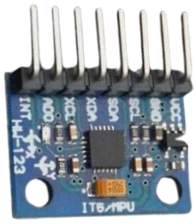
SELF-BALANCING BOT

Achyuth Mukund (22110435 / 22011102005)
Haneef Ahmad (22110433 / 22011102019)
3rd Year, 6th Semester - B. Tech CSE (IoT) 'A'

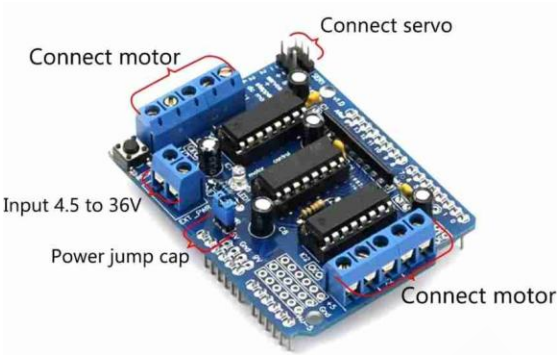
I. Introduction: A self-balancing bot is an autonomous two-wheeled robot that maintains its balance using a control system. This project implements a Proportional-Integral-Derivative (PID) controller to ensure the bot stays upright by making real-time adjustments based on sensor readings. An ESP32 microcontroller is used for processing, while an MPU6050 accelerometer and gyroscope provide tilt measurements.

II. Components Required/Used:

(i) ESP32 WROOM Microcontroller

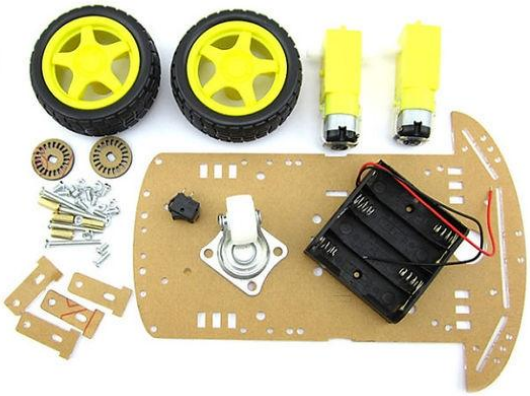


(ii) MPU – 6050 Accelerometer & Gyroscope



(iii) L293D Motor Driver

(iv) 12V DC Servo Motor



(v) 2 Wheeled Chassis Kit with Wheels, Battery Holder & Switch

(vi) 2500 mAh, 2.7V - Lithium Ion 18650 Cell Battery with Dual Battery Holder

- **ESP32 –** Microcontroller used to process sensor data and control motors.
- **MPU6050 –** 6-axis motion sensor providing acceleration and gyroscope readings.
- **Motor Driver –** Controls the direction and speed of the motors.
- **DC Motors –** Drive the bot forward and backward to maintain balance.
- **Battery –** Powers the ESP32, sensors, and motors.
- **Chassis & Wheels –** Provide the physical structure of the bot.
- **Switch –** For better ergonomics and ease of access.

III. Working Principle

This self-balancing bot relies on an **inverted pendulum model**. The MPU6050 sensor continuously provides acceleration and gyroscope data, which the ESP32 processes to determine the bot's tilt angle. A **PID (Proportional-Integral-Derivative) controller** is implemented to adjust the motor speed and direction to maintain balance.

IV. Connections:

- **MPU6050 to ESP32:**
  - VCC → 3.3V (ESP32)
  - GND → GND (ESP32)
  - SDA → GPIO21 (ESP32)
  - SCL → GPIO22 (ESP32)

Motor Driver to ESP32:

IN1, IN2 → GPIO26, GPIO27 (ESP32)
IN3, IN4 → GPIO14, GPIO12 (ESP32)
Enable Pins (PWM) → GPIO33, GPIO32 (ESP32)

- **Motor Driver to Motors & Power:**
  - Output terminals to respective motor leads.
  - External power from battery (preferably 12V for optimal performance).

V. Calibration of MPU6050

Proper calibration of the MPU6050 is essential for accurate tilt readings. Since raw sensor readings contain bias and noise, calibration is performed to correct the offsets.

1. Reading Raw Data

- Place the bot on a perfectly level surface.
- Read raw accelerometer and gyroscope values multiple times.

2. Calculating Offsets

- Compute the average accelerometer values for X, Y, and Z axes.
- Compute the average gyroscope values to find the zero-bias.
- Store these values as offsets. (Implicit Calibration)

3. Applying Offsets

- Subtract the measured offsets from real-time sensor readings.
- Ensures the bot considers zero tilt when placed in a balanced position.

```
mpu.initialize();
mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
```

VI. PID Controller: The PID controller is used to correct the tilt angle:

• Proportional (Kp): Reacts to the tilt error. Higher Kp means faster corrections.
• Integral (Ki): Accounts for accumulated past errors to remove steady-state drift.
• Derivative (Kd): Predicts future errors and dampens oscillations.

PID Formula: Output = (Kp × Error) + (Ki × ∑ Error) + (Kd × dError/dt)

The **output value** controls the **motor speed** and **direction** to restore balance.

Chosen PID Constants:

```
float Kp = 35.0; // Proportional
float Ki = 0.1; // Integral
float Kd = 1.0; // Derivative
```

Tuning the PID values was done by adjusting each term to achieve a stable balance.

VI. Challenges & Improvements

Challenges Faced:

- Noise in sensor readings causing instability.
- Incorrect PID tuning, leading to oscillations.
- Power supply fluctuations affecting motor response.

Possible Improvements:

- Implementing a **Kalman Filter** for better sensor fusion.
- Using a **higher torque motor** for better stability.
- Adding a **Bluetooth module** for remote tuning of PID parameters.

VII. Conclusion

This project successfully demonstrates the implementation of a self-balancing bot using an ESP32 microcontroller and an MPU6050 sensor. The PID controller effectively maintains balance by dynamically adjusting motor speed. Future enhancements can further improve the bot's efficiency and stability.