CS3684 - Introduction to Robotics
Continuous Internal Assessment (CIA) – 2

MAZE SOLVING BOT

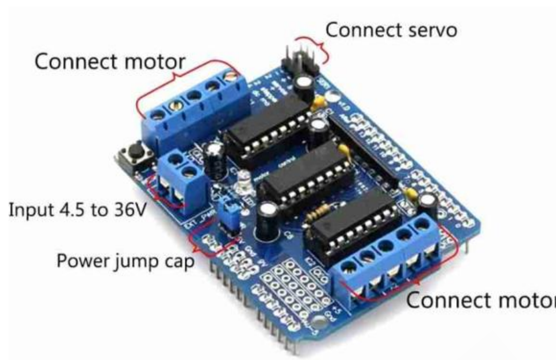
Achyuth Mukund (22110435 / 22011102005)
Haneef Ahmad (22110433 / 22011102019)
3rd Year, 6th Semester - B. Tech CSE (IoT) 'A'

I. Introduction: The Maze Solving Bot is a robotic system designed to autonomously navigate through a maze by using sensors and motors. This project aimed to demonstrate the implementation of algorithms for pathfinding, specifically to solve mazes efficiently. The bot uses an ultrasonic sensor for obstacle detection, an ESP32 microcontroller for processing, and motor drivers to control movement.

II. Components Required/Used:

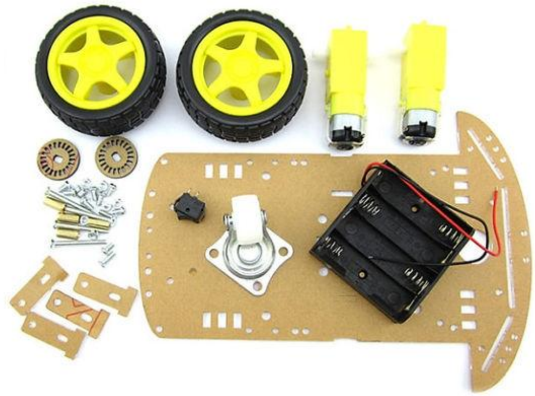
(i) ESP32 WROOM Microcontroller

(ii) HC-SR04 Ultrasonic Sensor



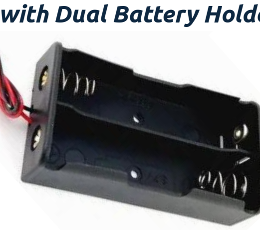
(iii) L293D Motor Driver

(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.
- **HC-SR04 Ultrasonic Sensor –** Used for detecting obstacles in front of the bot.
- **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:

The self-balancing bot uses an ultrasonic sensor to maintain a safe distance from obstacles. The system includes a PID (Proportional-Integral-Derivative) controller to adjust motor speeds based on distance feedback. The MPU6050 sensor is employed for tilt stabilization, and a servo motor is used to rotate the ultrasonic sensor for scanning the environment. By adjusting motor speeds through the PID controller, the bot can avoid obstacles and maintain a balanced position while navigating.

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).**
- **Ultrasonic Sensor to ESP32:**
  - **Trig Pin → GPIO18 (ESP32)**
  - **Echo Pin → GPIO19 (ESP32)**

Servo to ESP32:

Servo Pin → GPIO13 (ESP32)

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.

VI. Behavior and Movement Logic

The bot scans its environment using the servo-controlled ultrasonic sensor. The main logic revolves around maintaining an optimal distance (set to 40 cm in this case). The bot performs the following steps:

1. **Center Scan:** The ultrasonic sensor is positioned in the center, and the bot reads the distance. If the distance is too large or too small, the bot adjusts its movement based on the PID output.
2. **Obstacle Detection:** If the sensor detects an object within 30 cm, the bot evaluates the environment further by rotating the sensor to check the left and right directions.
3. **Side Scan:** The bot performs a side scan by rotating the ultrasonic sensor to 165° for the left and 25° for the right to determine which direction has a clear path.
4. **Movement Decision:** Based on the results of the side scan, the bot chooses to either turn left, turn right, or move forward, ensuring continuous navigation without hitting obstacles.

VII. Additional Features

- **Averaging Sensor Readings:** To improve accuracy, multiple readings from the ultrasonic sensor are averaged to minimize noise and outliers in the data.
- **PID Output Constraints:** The PID output is constrained to a range of -255 to 255 to ensure the motor commands are within the appropriate limits.
- **Timeout Mechanism:** In case the sensor fails or provides invalid readings, the bot assumes a safe path and continues moving forward.

VIII. Challenges & Improvements

Challenges Faced:

- Noise in Sensor Readings: The ultrasonic sensor can occasionally give noisy data, especially when measuring distances at an angle. This could lead to instability.
- Incorrect PID Tuning: In the initial stages, the PID values were improperly tuned, leading to oscillations or delayed corrections.
- Power Supply Fluctuations: Voltage dips from the battery can affect motor performance, causing erratic behavior.

Possible Improvements:

- Implementing a Kalman Filter to improve sensor fusion and filter out noise from the ultrasonic sensor.
- Using higher torque motors or more powerful battery sources for smoother movement and better stability.
- Adding a Bluetooth module for remote tuning of the PID parameters and real-time control.

IX. Conclusion:

This project successfully demonstrates the implementation of a self-balancing bot with obstacle detection and avoidance using an ultrasonic sensor and PID control. The bot is able to navigate autonomously by maintaining a set distance from obstacles and adjusting its movement based on real-time feedback. The integration of the MPU6050 sensor for balance stabilization and the servo-controlled ultrasonic sensor for scanning ensures robust performance. Future improvements could focus on sensor fusion, motor upgrades, and enhanced control features for better adaptability in complex environments.