**DEPARTMENT OF COMPUTER & SOFTWARE**

**ENGINEERING** **COLLEGE OF E&ME, NUST, RAWALPINDI**



# **MICROPROCESSOR AND MICROCONTROLLER BASED DESIGN**

**PROJECT REPORT**

**SUBMITTED TO:**

**Prof. Dr. Ishfaq Hussain**

**SUBMITTED BY (Group Members):**

**NAME**: Dawood Ali Khan **NAME**: Syed Faqih Ali Zamin

**REG#:** 482342 **REG#:** 454081

**NAME**: Kaleem Ullah **NAME**: Muhammad Hassan Ali

**REG#:** 479688 **REG#:** 456543

**CE-** 45(A)

**Submission Date: 6th May, 2025**

**Table of Content**

1. Introduction
2. Project Goals
3. Justification / Why We Chose This Project
4. Components List with Descriptions
5. Why We Use ESP32
6. ESP-NOW Protocol Overview
7. Transmitter Design
8. Receiver Design
9. System Architecture
10. Control interface
11. Limitations / Loopholes
12. Future Advancements
13. Conclusion
14. Code Appendix (Transmitter & Receiver Code)

**SMART Robotic Car with Gesture, Voice, and Obstacle Detection via ESP32 and IoT**

**1. INTRODUCTION**

The integration of natural user interfaces in robotics is redefining how humans interact with machines. This project presents a **multi-mode robotic car** that can be controlled through **hand gestures**, **voice commands using Google Assistant**, and features **obstacle detection** for autonomous navigation safety. Built on the **ESP32 microcontroller**, the system brings together embedded programming, wireless communication, cloud-based IoT integration, and real-time sensing to deliver an intuitive and responsive user experience.

The gesture control system uses an **MPU6050 accelerometer and gyroscope** to detect directional hand movements, which are wirelessly transmitted to the car via Wi-Fi. For voice control, **Google Assistant is integrated through IFTTT and Adafruit IO**, allowing users to control the robot using simple spoken commands from any smart device. An **ultrasonic sensor** mounted on the robot enables obstacle detection to prevent collisions, enhancing both functionality and safety.

The design and implementation of a gesture-controlled car plus voice control plus obstacle avoidance using an ESP32 microcontroller, standards wheels, and an MPU6050 motion sensor are presented. The ESP32, a low-cost, low-power system on a chip with integrated WiFi and Bluetooth capabilities, serves as the central controller. The system achieves movement by interpreting hand gestures, captured by the MPU6050 sensor and processed by the ESP32 to control the motors. The ESP-NOW protocol enables wireless communication between the gesture control unit and the car. Potential applications include assistive robotics, remote-controlled vehicles, and interactive entertainment, showcasing the usefulness of integrating gesture control with advanced wheel mechanisms to enhance mobility in robotic systems.

**2. PROJECT GOALS**

* **Design and implement a dual-control robotic car** using both gesture recognition and voice commands.
* **Integrate obstacle detection** to allow real-time collision avoidance and autonomous behavior.
* Achieve **low-latency response** for gesture control (<200ms) and acceptable delay for cloud-based voice commands.
* Use **ESP32 microcontrollers** for handling wireless communication, sensor inputs, and motor control.
* Employ **Google Assistant via IFTTT and Adafruit IO** for voice command integration using cloud-based services.
* Ensure a **cost-effective, modular, and scalable design** using standard rubber wheels and commonly available components.
* Demonstrate practical implementation of **IoT, sensor fusion, and real-time embedded systems**.

**3. JUSTIFICATION / WHY WE CHOSE THIS PROJECT**

Gesture control offers a natural and intuitive way to operate a robotic system without any physical controller. Voice control through Google Assistant adds a hands-free experience and shows how cloud-based IoT systems can interact with physical hardware. Obstacle detection not only improves the functionality of the robot but also enhances safety and enables future autonomous expansion.

This project is relevant for smart robotics, assistive technologies, and home automation systems, making it both educational and practically significant. It provides hands-on experience in sensor interfacing, IoT protocols, microcontroller programming, real-time control, and user-centered design — all of which are vital skills in the modern technological landscape.

**4. COMPONENTS LIST WITH PIN CONNECTION**

1. **ESP32 Development Board (1x)**

The ESP32 is a powerful microcontroller with integrated Wi-Fi and Bluetooth capabilities, making it ideal for IoT and robotics projects. It features a dual-core processor, multiple GPIO pins, ADCs, PWMs, UARTs, and other peripherals. In this project, it serves as the brain of the robot, handling motor control, sensor data processing, and communication.

**Example**: ESP32-WROOM-32

1. **L298N Motor Driver Module (1x)**

This module allows the microcontroller to control the speed and direction of two DC motors independently using PWM signals. It has built-in protection diodes, an onboard 5V regulator, and screw terminals for easy motor and power connections. It operates using a dual H-Bridge configuration.

**Alternative**: L293D Motor Driver IC with a breakout board (similar functionality but typically lower current handling)

1. **DC Motors (2x or 4x depending on configuration)**

Brushed DC motors are commonly used in mobile robots to provide locomotion. They rotate when a voltage is applied across their terminals. The number of motors used depends on the robot's design (e.g., 2-wheel or 4-wheel drive). Choose motors with appropriate voltage (usually 6V or 12V) and RPM according to your speed and torque requirements.

1. **Motor Power Supply**

Motors require more current than what the ESP32 can supply. A dedicated external power supply ensures the motors operate effectively without affecting the microcontroller. Choose a power source based on motor voltage and current needs.

**Examples**:

* 7.4V Li-ion rechargeable battery (compact and rechargeable)
* 9V battery (for basic applications with low current motors)

1. **Jumper Wires (Male-to-Male, Male-to-Female, Female-to-Female)**

Used for making reliable electrical connections between components like the ESP32, motor driver, and power supply. Different types of jumper wires are used depending on the pin configuration of the components.

1. **Breadboard**

A solderless prototyping board that allows you to build and test circuit designs quickly. It’s especially useful during the development and testing phase for temporary circuit setups.

1. **USB Cable (Micro-USB)**

Used to connect the ESP32 board to a computer. It is essential for uploading code, serial communication (monitoring outputs), and supplying power during development and debugging.

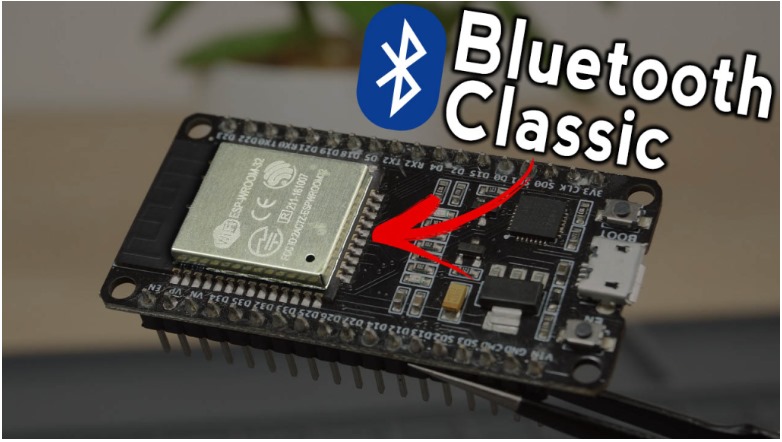
1. **Pin Configuration**

|  |  |  |
| --- | --- | --- |
| **ESP32 Pin** | **Connected To** | **Component Function** |
| GPIO16 | IN1 of L298N | Motor A - Direction Control 1 |
| GPIO17 | IN2 of L298N | Motor A - Direction Control 2 |
| GPIO18 | IN3 of L298N | Motor B - Direction Control 1 |
| GPIO19 | IN4 of L298N | Motor B - Direction Control 2 |
| GPIO4 | ENA of L298N | Motor A - Speed Control (PWM) |
| GPIO5 | ENB of L298N | Motor B - Speed Control (PWM) |
| 3.3V | VCC of L298N (Logic) | Power supply for motor driver logic |
| GND | GND of L298N | Common ground for ESP32 and L298N |

**5. WHY WE USE ESP32?**

The ESP32 microcontroller is a powerful and versatile platform that perfectly aligns with the requirements of this multi-functional robotic car project. It was chosen for the following reasons:

1. **Built-in Wi-Fi and Bluetooth Connectivity**  
   One of the key strengths of the ESP32 is its integrated **Wi-Fi and Bluetooth (BLE)** modules. These features eliminate the need for additional wireless modules, making the ESP32 a compact and all-in-one solution for wireless communication.
2. **High Performance at Low Cost**  
   The ESP32 offers a **dual-core processor**, running at up to 240 MHz with 520 KB SRAM, making it capable of handling multitasking — such as real-time gesture processing, voice command reception, and obstacle detection. Despite these capabilities, it remains **affordable and accessible**, making it ideal for educational and student projects.
3. **Ease of Programming and Community Support**  
   The ESP32 is compatible with the Arduino IDE, and there exists a vast collection of libraries and documentation available through an active developer community. This reduces development time and simplifies debugging, especially when integrating sensors like MPU6050, ultrasonic modules, or interfacing with motor drivers.
4. **Scalability for Future Features**  
   The ESP32 supports advanced protocols and peripherals (e.g., camera modules, capacitive touch, DAC), which makes the system **future-ready**. For example, integrating an **ESP32-CAM** module or **offline voice recognition** could be explored in later versions without changing the main controller.

****

**6. ESP-NOW PROTOCOL OVERVIEW**

ESP-NOW is a fast communication protocol that can be used to exchange small messages (up to 250 bytes) between ESP32 boards(in our Case). Stating Espressif website, ESP-NOW is a “protocol developed by Espressif, which enables multiple devices to communicate with one another without using Wi-Fi. The protocol is similar to the low-power 2.4GHz wireless connectivity (…) . The pairing between devices is needed prior to their communication. After the pairing is done, the connection is safe and peer-to-peer, with no handshake being required.”





This means that after pairing a device with each other, the connection is persistent. In other words, if suddenly one of your boards loses power or resets, when it restarts, it will automatically connect to its peer to continue the communication. ESP-NOW supports the following features:

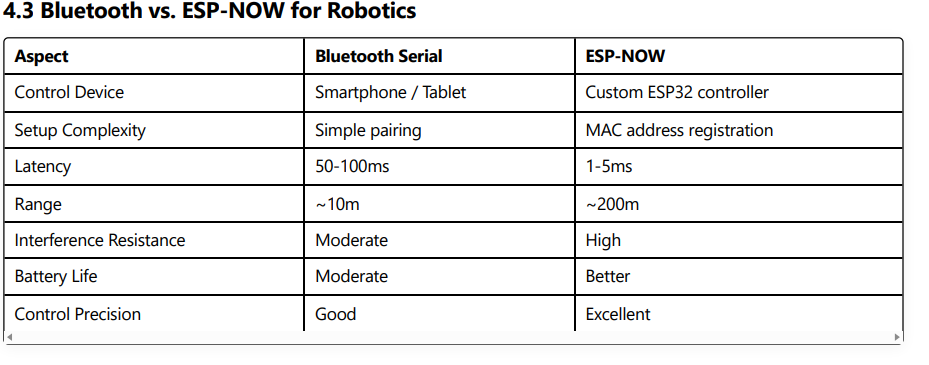
Encrypted and unencrypted unicast communication;

Mixed encrypted and unencrypted peer devices;

Up to 250-byte payload can be carried;

Sending callback function that can be set to inform the application layer of

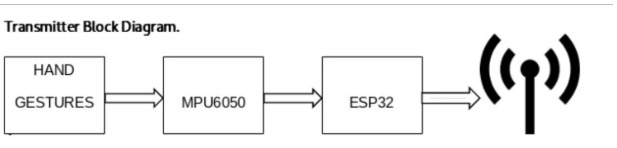
transmission success or failure.

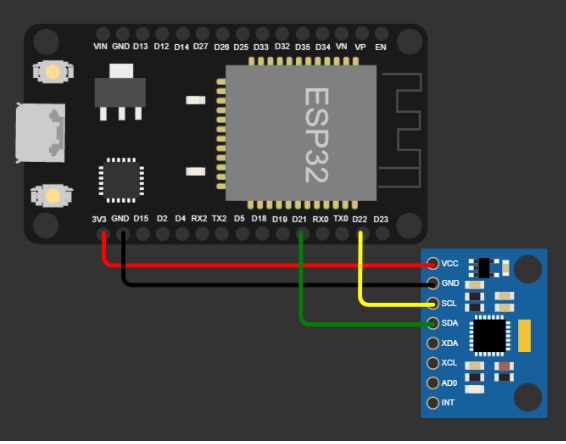


1. **TRANSMITTER**

The **Transmitter** section is designed to capture hand gestures and wirelessly transmit control signals to the receiver. Here’s how it works:

* **Hand Gestures**: Input is provided via hand movements, detected by the **MPU6050** sensor (a 6-axis accelerometer and gyroscope).
* **ESP32 Microcontroller**: Processes the gesture data from the MPU6050 and encodes it into signals.
* **Wireless Transmission**: The ESP32 sends the control signals (e.g., motor directions/speed) to the receiver unit via Wi-Fi/Bluetooth.

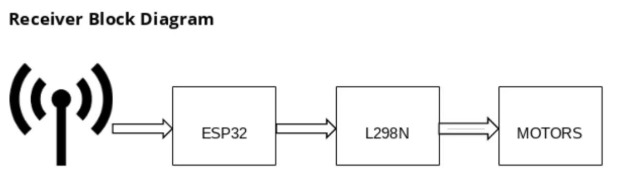


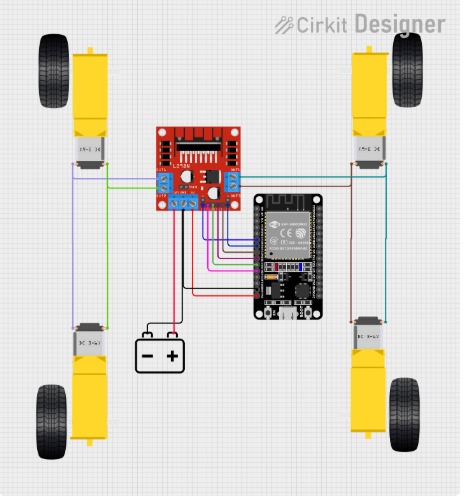


1. **RECEIVER**

The **Receiver** section interprets the transmitted signals and drives the motors accordingly. Key components:

* **ESP32**: Receives wireless signals from the transmitter and decodes them into motor commands.
* **L298N Motor Driver**: Acts as an intermediary between the ESP32 and motors, providing the necessary power and control (direction/speed) to the motors.
* **Motors**: Execute movements (e.g., forward, backward, turn) based on the processed signals.





1. **SYSTEM WORKFLOW**
2. **Transmitter**: Gesture → MPU6050 → ESP32 (encodes) → Wireless signal.
3. **Receiver**: ESP32 (decodes) → L298N → Motors (action).
4. **CONTROL INTERFACE**



A **versatile control interface** designed for remote or embedded systems, offering multiple input options and real-time feedback. The interface includes **arrow keys** for directional navigation, a **terminal** for direct command input or debugging, and **W/S keys** for acceleration control, such as throttle adjustment. Additionally, an **accelerometer** enables motion-based input, allowing users to control the system by tilting or moving the device. Physical or on-screen **buttons and a slider** provide further customization, such as speed adjustments or mode selection.

Real-time feedback is displayed through metrics, including a **distance measurement** (15 cm, likely from an ultrasonic sensor) and other system data such as speed, battery level, or sensor readings. The **"HO"** indicator may represent a hold or override function, ensuring safety during operation. Advanced features like **voice control** enhance usability by enabling hands-free commands, making the system more accessible. The mention of **PCBWay's promotional offer** ($5 for 10 PCBs) suggests that the interface is part of a custom-designed circuit board, emphasizing affordability and practicality for DIY electronics or prototyping projects.

Overall, this control interface combines **traditional inputs (keys, buttons) with modern features (accelerometer, voice control)**, making it suitable for robotics, remote-controlled vehicles, or interactive systems requiring flexible and intuitive operation.

1. **LIMITATIONS / LOOPHOLES**

While the project successfully demonstrates a smart, multi-modal robotic control system, it does have certain limitations:

1. **Limited Mobility**: The robot uses standard rubber wheels, which restrict it to forward, backward, and turning motions only. It lacks omnidirectional or strafing capability.
2. **Voice Command Latency**: Voice control relies on internet connectivity and cloud services (IFTTT and Adafruit IO), which introduces delays ranging from 1 to 3 seconds depending on network conditions.
3. **Gesture Recognition Sensitivity**: The MPU6050 sensor can misinterpret rapid or unintentional hand movements, requiring the user to keep steady posture during use. It may also suffer from gyroscopic drift over time.
4. **No Real-Time Feedback**: The robot does not send any status updates (e.g., battery voltage, distance to obstacle, motor speed) back to the user, which could be valuable for monitoring and debugging.
5. **Obstacle Detection Range**: The ultrasonic sensor detects obstacles only in a limited field of view and within a short range (usually <4 meters), which may not be sufficient for high-speed navigation or complex environments.
6. **Power Management**: Running motors and sensors simultaneously requires careful regulation of voltage and current. Improper power distribution may affect performance or damage components.
7. **Dependency on Internet**: The voice control mode is unusable without an active and stable internet connection.

**12. FUTURE ADVANCEMENTS**

To address the current limitations and enhance the system’s capabilities, several future improvements can be considered:

1. **Omnidirectional Motion**: Replacing standard wheels with **mecanum or omni wheels** will allow the robot to move in all directions, improving agility and enabling complex movement patterns.
2. **Edge Computing for Gesture Recognition**: Implementing **machine learning-based gesture classification** directly on the ESP32 or a companion edge device (e.g., Raspberry Pi) would reduce false gestures and improve accuracy.
3. **Real-Time Feedback System**: Integrating a **bi-directional communication protocol** (like MQTT or WebSockets) would allow the robot to report status (obstacle distance, speed, battery level) to a dashboard or mobile app.
4. **Offline Voice Command Support**: Using offline voice recognition modules (e.g., Elechouse Voice Recognition Module or AI-powered ESP-Skainet) would eliminate dependency on internet for voice control.
5. **Mobile App Integration**: Adding a smartphone app via **Bluetooth or Wi-Fi** could provide a GUI for manual control, parameter tuning, and data logging.
6. **Multi-Sensor Fusion**: Combining **IR sensors, LiDAR, or a camera module (like ESP32-CAM)** can improve environmental awareness, support obstacle avoidance in all directions, and enable visual line-following or object detection.
7. **Solar Charging or Smart Power System**: Adding a **solar panel** or **intelligent battery management system** could extend operational time and improve energy efficiency.
8. **Voice Feedback or Speech Output**: Incorporating a speaker module to provide audio responses (e.g., “Turning Left”, “Obstacle Ahead”) can enhance interactivity and accessibility.

**13. CONCLUSION**

This project successfully demonstrates the design and implementation of a **multi-mode smart robotic car** that integrates **gesture control, voice activation through Google Assistant, and real-time obstacle detection**. Built around the ESP32 microcontroller, the system showcases how embedded systems, IoT communication, and sensor technologies can be combined to create an intuitive and responsive robotic platform.

By utilizing the **MPU6050 sensor**, the robot responds to hand gestures for directional movement, offering a natural human-machine interaction method. Simultaneously, **Google Assistant integration via IFTTT and Adafruit IO** allows voice commands to be executed remotely, proving the potential of cloud-based control. The inclusion of an **ultrasonic sensor** ensures basic collision avoidance, improving safety and usability in dynamic environments.

Although there are limitations such as reliance on internet connectivity for voice control and limited mobility due to standard wheels, the project achieves its key objectives. It demonstrates real-time, wireless control using affordable and accessible hardware, making it a valuable educational tool and a foundation for future development in smart robotics.

Overall, this project represents a significant step toward user-friendly robotic systems that combine **gesture recognition**, **voice control**, and **environmental awareness**. It opens pathways for applications in **assistive technology**, **home automation**, and **human-robot interaction**, and serves as a scalable prototype for more advanced autonomous systems.

**14. CODING APPENDIX**

#include <ESP32Servo.h>

#include "BluetoothSerial.h"

BluetoothSerial SerialBT;

// Motor Connections

#define IN1 16

#define IN2 17

#define IN3 18

#define IN4 19

#define ENA 4

#define ENB 5

// Ultrasonic Sensor Connections

int frontTrigPin = 22;

int frontEchoPin = 23;

int backTrigPin = 12;

int backEchoPin = 14;

Servo myServo;

int motor\_speed = 150;

char t;

void setup() {

  Serial.begin(9600);

  SerialBT.begin("ESP32\_4WD");  // Bluetooth device name

  // Initializing motor pins

  pinMode(IN1, OUTPUT);

  pinMode(IN2, OUTPUT);

  pinMode(IN3, OUTPUT);

  pinMode(IN4, OUTPUT);

  pinMode(ENA, OUTPUT);

  pinMode(ENB, OUTPUT);

  // Initializing Ultrasonic Sensor Pins

  pinMode(backTrigPin, OUTPUT);

  pinMode(backEchoPin, INPUT);

  pinMode(frontTrigPin, OUTPUT);

  pinMode(frontEchoPin, INPUT);

  myServo.attach(13, 1000, 2000);

  myServo.write(90);

}

void loop() {

  if (SerialBT.available()) {

    t = SerialBT.read();

    Serial.println(t);

  }

  switch (t) {

    case 'F':

      {  //move forward(all motors rotate in forward direction)

        myServo.write(90);

        delay(500);  // Wait for servo to stabilize

        checkAndMove("move forward");

      }

      break;

    case 'B':

      {  //move reverse (all motors rotate in reverse direction)+

        if (measureDistanceBack() < 30) {

          stopRobot();

        } else {

          moveBackward();

        }

      }

      break;

    case 'L':

      {  //turn right (left side motors rotate in forward direction, right side motors doesn't rotate)

        myServo.write(160);

        delay(500);  // Wait for servo to stabilize

        checkAndMove("move left");

        delay(300);

      }

      break;

    case 'R':

      {  //turn left (right side motors rotate in forward direction, left side motors doesn't rotate)

        myServo.write(20);

        delay(500);  // Wait for servo to stabilize

        checkAndMove("move right");

        delay(300);

      }

      break;

    case 'S':

      {  //STOP (all motors stop)

        myServo.write(90);

        stopRobot();

      }

  }

}

long measureDistanceAhead() {

  // Send a 10us pulse to the trigger pin to initiate the measurement

  digitalWrite(frontTrigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(frontTrigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(frontTrigPin, LOW);

  // Measure the pulse duration from the echo pin

  long duration = pulseIn(frontEchoPin, HIGH);

  // Calculate distance in centimeters (speed of sound = 343 m/s)

  long distance = (duration / 2) / 29.1;

  Serial.println(distance);

  return distance;

}

int measureDistanceBack() {

  long duration;

  digitalWrite(backTrigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(backTrigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(backTrigPin, LOW);

  duration = pulseIn(backEchoPin, HIGH, 20000);  // Timeout after 20ms

  if (duration == 0) {

    return -1;  // No object detected

  }

  int distance = duration \* 0.034 / 2;

  // Serial.println(distance);

  return distance;

}

void checkAndMove(const String& direction) {

  long distance = measureDistanceAhead();  // Measure the distance to the obstacle

  Serial.print(direction);

  Serial.print(" Distance: ");

  Serial.println(distance);

  if (distance > 30) {  // If no obstacle within 30 cm, move the robot in the desired direction

    if (direction == "move forward") {

      moveForward();

    } else if (direction == "move left") {

      turnLeft();

    } else if (direction == "move right") {

      turnRight();

    }

  } else {

    stopRobot();  // Stop if an obstacle is detected

    Serial.println("Obstacle detected! Moving stopped.");

  }

}

void moveForward() {

  analogWrite(ENA, motor\_speed);

  analogWrite(ENB, motor\_speed);

  digitalWrite(IN1, HIGH);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, HIGH);

  digitalWrite(IN4, LOW);

}

void moveBackward() {

  analogWrite(ENA, motor\_speed);

  analogWrite(ENB, motor\_speed);

  digitalWrite(IN1, LOW);

  digitalWrite(IN2, HIGH);

  digitalWrite(IN3, LOW);

  digitalWrite(IN4, HIGH);

}

void turnRight() {

  analogWrite(ENA, motor\_speed);

  analogWrite(ENB, motor\_speed);

  digitalWrite(IN1, LOW);

  digitalWrite(IN2, HIGH);

  digitalWrite(IN3, HIGH);

  digitalWrite(IN4, LOW);

}

void turnLeft() {

  analogWrite(ENA, motor\_speed);

  analogWrite(ENB, motor\_speed);

  digitalWrite(IN1, HIGH);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, LOW);

  digitalWrite(IN4, HIGH);

}

void stopRobot() {

  analogWrite(ENA, 0);

  analogWrite(ENB, 0);

  digitalWrite(IN1, LOW);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, LOW);

  digitalWrite(IN4, LOW);

}

      {  //STOP (all motors stop)

        myServo.write(90);

        stopRobot();

      }

  }

}

long measureDistanceAhead() {

  // Send a 10us pulse to the trigger pin to initiate the measurement

  digitalWrite(frontTrigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(frontTrigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(frontTrigPin, LOW);

  // Measure the pulse duration from the echo pin

  long duration = pulseIn(frontEchoPin, HIGH);

  // Calculate distance in centimeters (speed of sound = 343 m/s)

  long distance = (duration / 2) / 29.1;

  Serial.println(distance);

  return distance;

}

int measureDistanceBack() {

  long duration;

  digitalWrite(backTrigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(backTrigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(backTrigPin, LOW);

  duration = pulseIn(backEchoPin, HIGH, 20000);  // Timeout after 20ms

  if (duration == 0) {

    return -1;  // No object detected

  }

  int distance = duration \* 0.034 / 2;

  // Serial.println(distance);

  return distance;

}

void checkAndMove(const String& direction) {

  long distance = measureDistanceAhead();  // Measure the distance to the obstacle

  Serial.print(direction);

  Serial.print(" Distance: ");

  Serial.println(distance);

  if (distance > 30) {  // If no obstacle within 30 cm, move the robot in the desired direction

    if (direction == "move forward") {

      moveForward();

    } else if (direction == "move left") {

      turnLeft();

    } else if (direction == "move right") {

      turnRight();

    }

  } else {

    stopRobot();  // Stop if an obstacle is detected

    Serial.println("Obstacle detected! Moving stopped.");

  }

}

void moveForward() {

  analogWrite(ENA, motor\_speed);

  analogWrite(ENB, motor\_speed);

  digitalWrite(IN1, HIGH);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, HIGH);

  digitalWrite(IN4, LOW);

}

void moveBackward() {

  analogWrite(ENA, motor\_speed);

  analogWrite(ENB, motor\_speed);

  digitalWrite(IN1, LOW);

  digitalWrite(IN2, HIGH);

  digitalWrite(IN3, LOW);

  digitalWrite(IN4, HIGH);

}

void turnRight() {

  analogWrite(ENA, motor\_speed);

  analogWrite(ENB, motor\_speed);

  digitalWrite(IN1, LOW);

  digitalWrite(IN2, HIGH);

  digitalWrite(IN3, HIGH);

  digitalWrite(IN4, LOW);

}

void turnLeft() {

  analogWrite(ENA, motor\_speed);

  analogWrite(ENB, motor\_speed);

  digitalWrite(IN1, HIGH);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, LOW);

  digitalWrite(IN4, HIGH);

}

void stopRobot() {

  analogWrite(ENA, 0);

  analogWrite(ENB, 0);

  digitalWrite(IN1, LOW);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, LOW);

  digitalWrite(IN4, LOW);

}

**RECEIVER CODE**

// the code to get the Receiver MAC Address

// which will be entered to transmitter code

// #include <WiFi.h>

// void setup() {

// Serial.begin(115200);

// WiFi.mode(WIFI\_STA);

// Serial.print("Receiver MAC Address: ");

// Serial.println(WiFi.macAddress());

// }

// void loop() {

// }

#include <esp\_now.h>

#include <WiFi.h>

// Motor Pins

#define IN1 16

#define IN2 17

#define IN3 18

#define IN4 19

#define ENA 4

#define ENB 5

typedef struct struct\_message {

char direction;

} struct\_message;

struct\_message incomingMessage;

void onDataRecv(const uint8\_t \*mac, const uint8\_t \*incomingData, int len) {

memcpy(&incomingMessage, incomingData, sizeof(incomingMessage));

char dir = incomingMessage.direction;

// Serial.print("Direction: ");

// Serial.println(dir);

switch (dir) {

case 'F':

moveForward();

break;

case 'B':

moveBackward();

break;

case 'L':

turnLeft();

break;

case 'R':

turnRight();

break;

case 'S':

stopMotors();

break;

}

}

void setup() {

Serial.begin(115200);

WiFi.mode(WIFI\_STA);

if (esp\_now\_init() != ESP\_OK) {

Serial.println("Error initializing ESP-NOW");

return;

}

esp\_now\_register\_recv\_cb(onDataRecv);

pinMode(IN1, OUTPUT);

pinMode(IN2, OUTPUT);

pinMode(IN3, OUTPUT);

pinMode(IN4, OUTPUT);

pinMode(ENA, OUTPUT);

pinMode(ENB, OUTPUT);

}

void loop() {

// Nothing in loop; all work is done in callback

}

void moveForward() {

analogWrite(ENA, 150);

analogWrite(ENB, 150);

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

digitalWrite(IN3, LOW);

digitalWrite(IN4, HIGH);

}

void moveBackward() {

analogWrite(ENA, 150);

analogWrite(ENB, 150);

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

digitalWrite(IN3, HIGH);

digitalWrite(IN4, LOW);

}

void turnRight() {

analogWrite(ENA, 150);

analogWrite(ENB, 150);

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

digitalWrite(IN3, HIGH);

digitalWrite(IN4, LOW);

}

void turnLeft() {

analogWrite(ENA, 150);

analogWrite(ENB, 150);

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

digitalWrite(IN3, LOW);

digitalWrite(IN4, HIGH);

}

void stopMotors() {

analogWrite(ENA, 0);

analogWrite(ENB, 0);

digitalWrite(IN1, LOW);

digitalWrite(IN2, LOW);

digitalWrite(IN3, LOW);

digitalWrite(IN4, LOW);

}

**TRANSMITTER CODE**

// Transmitter code

#include <Wire.h>

#include <esp\_now.h>

#include <WiFi.h>

#include <MPU6050.h>

MPU6050 mpu;

uint8\_t receiverMAC[] = {0xA0, 0xB7, 0x65, 0x04, 0x4D, 0x5C}; // Replace with Receiver MAC Address

typedef struct struct\_message {

char direction;

} struct\_message;

struct\_message message;

void setup() {

Serial.begin(115200);

Wire.begin();

mpu.initialize();

WiFi.mode(WIFI\_STA);

if (esp\_now\_init() != ESP\_OK) {

Serial.println("Error initializing ESP-NOW");

return;

}

esp\_now\_peer\_info\_t peerInfo = {};

memcpy(peerInfo.peer\_addr, receiverMAC, 6);

peerInfo.channel = 0;

peerInfo.encrypt = false;

if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK) {

Serial.println("Failed to add peer");

return;

}

}

void loop() {

int16\_t ax, ay, az;

int16\_t gx, gy, gz;

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

float pitch = atan2(ax, sqrt(ay \* ay + az \* az)) \* 180.0 / PI;

float roll = atan2(ay, sqrt(ax \* ax + az \* az)) \* 180.0 / PI;

// Determine direction

if (pitch > 10) {

message.direction = 'F'; // Forward

} else if (pitch < -10) {

message.direction = 'B'; // Backward

} else if (roll > 10) {

message.direction = 'R'; // Right

} else if (roll < -10) {

message.direction = 'L'; // Left

} else {

message.direction = 'S'; // Stop

}

esp\_now\_send(receiverMAC, (uint8\_t \*)&message, sizeof(message));

delay(200);

}1