ABSTRACT

The advancement of human–machine interaction has led to innovative solutions in the field of robotics and automation. This project presents a Smart Gesture-Controlled RC Car that responds to hand gestures instead of conventional buttons or joysticks. The system utilizes an A Esp32 microcontroller integrated
with an accelerometer sensor (mounted on a wearable glove or smartphone) to capture hand movements. These gestures are wirelessly transmitted to the RC car via Bluetooth communication, where the signals are interpreted and executed through a motor driver to control the car's motion.

The proposed design emphasizes intuitive and user-friendly control, eliminating the need for physical controllers. Forward, backward, left, right, and stop commands are executed seamlessly based on the orientation of the user's hand. This makes the system not only engaging but also applicable in scenarios where touch-free control is preferred.
The project demonstrates the integration of embedded systems, wireless communication, and sensor-based control for smart robotics. It serves as an educational platform for students, hobbyists, and researchers, while also laying the groundwork for applications in assistive technologies, remote surveillance, and IoT-based smart vehicles.

By enhancing interaction through natural gestures, the Smart Gesture RC Car showcases a step towards human-centric automation and offers scope for future upgrades such as obstacle detection, voice commands, and autonomous navigation.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

1.1 Introduction to Next-Generation Robotics Control

1.1.1 Overview

Radio-Controlled (RC) vehicles have long been a foundation for hobbyists and robotics learners. However, traditional control methods using joysticks or buttons often feel unnatural and disconnected. This project, Smart Gesture Control RC Car, aims to create a more intuitive and immersive Human-Machine Interface (HMI) by replacing manual controls with natural hand gestures.

1.1.2 Problem Statement

Conventional RC systems face challenges such as high latency, signal interference, and limited motion precision. The need for a responsive and intuitive control method inspired this project—to design a gesture-based control system that mirrors human hand movements in real-time, ensuring smooth and accurate vehicle navigation.

1.1.3 Project Vision

The project introduces a gesture-controlled RC car system that interprets the operator's hand tilt and rotation using an Inertial Measurement Unit (IMU), specifically the MPU6050 sensor. The sensor detects hand pitch (forward/backward tilt) and roll (left/right tilt), translating them into corresponding motion commands. This system, powered by the ESP32 microcontroller, enables seamless and natural vehicle control.

1.2 System Overview

1.2.1 Architecture

The system operates on a two-unit wireless design:

- Transmitter Unit (TX): Worn by the user to capture and transmit gesture data.
- Receiver Unit (RX): Mounted on the car to process commands and control the motors. Both units communicate wirelessly via ESP-NOW, ensuring low-latency, peer-to-peer data transmission without Wi-Fi dependency.

1.2.2 Transmitter Unit (TX)

- Microcontroller: ESP32 – handles data processing and transmission.
- Sensor: MPU6050 – detects hand tilt and motion.
- Power-Supply : Rechargeable-Li-ion battery.
  Filtered and calibrated sensor data are converted into motion commands and transmitted to the receiver.

1.2.3 Receiver Unit (RX)

- Microcontroller: ESP32 – receives and interprets commands.
- Motor Driver: L298N Dual H-Bridge – controls direction and speed of DC motors.
- Actuators: Four DC motors for motion and steering.
  PWM signals regulate speed, while digital logic pins determine direction for forward, reverse, and turning operations.

1.3 Core Technologies

1.3.1 Gesture-to-Command Logic

- Pitch (X-axis): Controls forward and backward motion.
- Roll (Y-axis): Controls left and right steering.
  Calibration ensures stability and accuracy, filtering minor hand tremors and defining a neutral (idle) position.

1.3.2 Wireless Communication (ESP-NOW)

ESP-NOW, developed by Espressif, provides:

- Low latency for real-time control.
- Direct peer-to-peer communication between ESP32 modules.
  This eliminates the need for Wi-Fi or routers, making the system efficient and power-saving.

1.3.3 Motor Control via PWM

The L298N motor driver interfaces the ESP32 with the DC motors:

- Direction Control: Through logic inputs (IN1–IN4).
- Speed Control: Through PWM signals on enable pins (ENA/ENB). This setup enables smooth acceleration and precise motion control based on hand gestures.

Here's how it works:

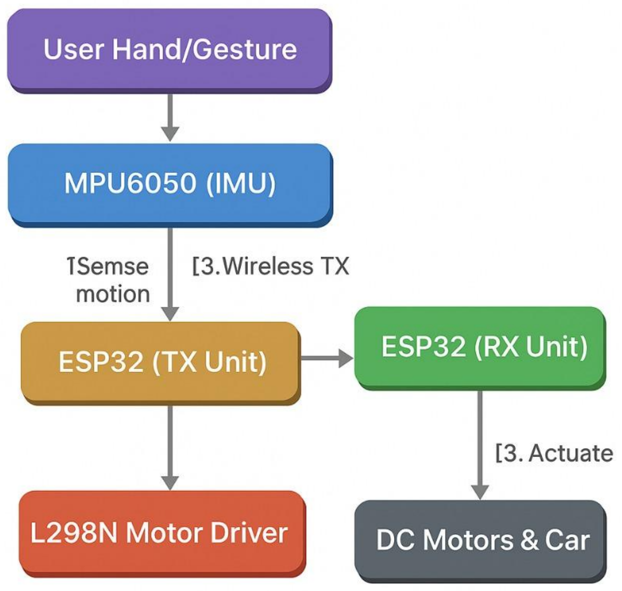


Figure 1.1 Work Flow

CHAPTER 2: LITERATURE REVIEW / EXISTING SYSTEM

2.1 Introduction: Addressing the Control Gap

Traditional Remote-Controlled (RC) vehicles use abstract input devices (joysticks, buttons), which create a cognitive barrier between the operator's intention and the vehicle's movement. The **Smart Gesture Control RC Car** project was developed to establish a more intuitive Human-Machine Interface (HMI) by translating natural hand gestures into real-time, proportional vehicle commands. The primary goals were to achieve high control fidelity, responsiveness, and minimal latency in a robust, cost-effective system.

2.2 Literature Review: Sensing and Communication Paradigms

The field of gesture-controlled robotics is primarily divided by its sensing technology:

1. Sensor-Based (IMU) Control: This approach, adopted by this project, uses Inertial Measurement Units (IMUs) like the **MPU6050** to capture 3-axis motion (acceleration and angular velocity). Numerous studies validate that the MPU6050 is ideal for converting hand orientation (Pitch and Roll) into vehicle commands (forward/reverse and steering). Key advantages are low computational requirements, immunity to ambient lighting, and suitability for proportional control.

2. Vision-Based Control: Alternative systems use cameras or depth sensors (e.g., Leap Motion). While offering non-contact interaction, these systems require heavy computational resources for image processing and suffer from degraded performance in poor lighting or when the hand is occluded. The IMU-based wearable glove approach was chosen for its superior **robustness, portability, and low-latency processing capability** at the embedded level.

3. Communication Protocol Analysis: Low latency is critical for a smooth control experience.

Limitations of Existing Systems: Traditional methods like **Bluetooth (BT)** and basic **RF modules** often suffer from limited range, interference, and noticeable lag, making precise, real-time control difficult.

Project Innovation (ESP-NOW): This project leverages the **ESP-NOW** proprietary protocol. By operating directly on the data-link layer, ESP-NOW bypasses the complex overhead of standard Wi-Fi, enabling **millisecond-level, peer-to-peer communication**. This is the key enabler for the project's high responsiveness and reliable operation.

2.3 System Architecture: Two-Unit Wireless Command

The design is based on a **Two-Unit Wireless Architecture**, providing dedicated processing for sensing and actuation.

Table 2.1 System Architecture: (component Function)

Unit	Role	Key Components	Function
Transmitter (TX) Unit	Wearable Command Glove	**ESP32 (Microcontroller)**	Processes MPU6050 data, establishes and manages the ESP-NOW link.
		MPU6050 (IMU)	Measures Pitch (forward/backward tilt) and Roll (left/right tilt) angles.
Receiver (RX) Unit	The RC Vehicle Chassis	**ESP32 (Microcontroller)**	Receives ESP-NOW packets and translates data into motor instructions.
		L298N (Motor Driver)	H-bridge circuit that converts low-power logic signals into high-current DC motor control.

2.4 Technical Foundations: Proportional Control Logic

Gesture-to-Command Mapping: The TX ESP32 continuously polls the MPU6050, interpreting the two primary axes for control:

Pitch Angle (Speed): The degree of forward/backward tilt is mapped directly to a **Pulse Width Modulation (PWM) duty cycle**. A larger angle results in a higher PWM value, achieving **proportional speed control** (smooth acceleration/deceleration).

Roll Angle (Steering): Lateral tilt commands a **differential speed** between the left and right motor pairs, enabling turning while maintaining forward or reverse motion.

Calibration and Safety: A rigorous calibration process establishes a neutral "zero-point" and defines angular thresholds. This creates a stable idle state and a vital **Fail-Safe** mechanism, where the car automatically stops if the hand is held level or outside the recognized command zone.

CHAPTER 3: METHODOLOGY / SYSTEM ANALYSIS

3.1 Methodology Overview

The development of the Smart Gesture Control RC Car followed a structured and phased methodology to ensure that the system is responsive, accurate, robust, and user-friendly. The project was carried out in six phases: feasibility study and planning, initiation and planning, development, implementation, monitoring and controlling, and closing. Each phase played a vital role in designing, constructing, and deploying a functional RC car system that translates hand gestures into real-time vehicle commands using ESP32, MPU6050, and L298N motor drivers.

3.2 Feasibility Study and Planning

The first phase involved conducting a feasibility study to evaluate the technical, operational, and economic viability of the project. From a technical perspective, the project was found feasible with the ESP32 microcontroller as the main processing unit, the MPU6050 sensor for gesture detection, and the L298N motor driver for DC motor control. The ESP-NOW communication protocol was chosen for low-latency, peer-to-peer wireless transmission. Open-source development tools and libraries were used to reduce cost and simplify implementation. Operational feasibility was assessed by considering the system's usability for students, hobbyists, and educational purposes. Various scenarios, such as indoor and outdoor usage, low-latency control, and proportional speed steering, were analyzed to ensure practicality. Economically, the project required affordable, widely available components and no specialized hardware, making it cost-effective and easy to replicate.

3.3 Initiation and Planning

During the planning phase, clear project objectives were established. The primary goal was to design an intuitive gesture-controlled RC car that translates natural hand movements into accurate and proportional vehicle commands. Specific tasks were identified, including hardware selection, sensor calibration, wireless communication setup, motor control algorithm development, and system integration. A timeline was created with milestones for sensor calibration, gesture mapping, ESP-

NOW communication setup, and motor testing. Potential risks were identified, such as wireless interference, sensor drift, motor overheating, or loss of control. To mitigate these risks, ESP-NOW was selected for reliable communication, calibration routines were implemented to reduce sensor drift, and monitoring of motor voltage and temperature was planned.

3.4 Development Phase

The development phase involved modular construction of the system. The transmitter unit, worn as a glove, included the ESP32 microcontroller and the MPU6050 sensor, which continuously read hand gestures in terms of pitch and roll. These gestures were mapped to movement commands and transmitted in real time to the receiver unit via ESP-NOW. The receiver, mounted on the RC car chassis, received these commands and controlled the motors using the L298N driver, converting logic signals into proportional PWM outputs for smooth acceleration and turning. During this phase, software routines were developed for sensor calibration, filtering of IMU data, gesture-to-command mapping, and fail-safe mechanisms to stop the car if gestures were neutral or outside the recognized range. Hardware assembly involved connecting the DC motors, battery pack, and ESP32 units on a transparent chassis with proper wiring and mechanical alignment.

3.5 Implementation Phase

After development, the system was integrated and tested as a complete unit. Integration testing verified the real-time communication between the glove and the RC car, ensuring minimal latency and accurate proportional control of speed and steering. The system was tested under various environmental conditions, including indoor and outdoor settings, to evaluate stability and responsiveness. Motor performance and battery management were also assessed to prevent overheating and ensure continuous operation. Multiple trials were conducted to fine-tune the gesture mapping and verify the effectiveness of the fail-safe mechanism.

3.6 Monitoring and Controlling

During operation, the system's performance was continuously monitored. Key parameters measured included latency between hand gestures and car response, accuracy of speed and steering control, wireless communication stability, and motor current and voltage. Sensor calibration was refined iteratively to minimize drift, and PWM values were adjusted to achieve smooth and proportional motion. Test users provided feedback on the intuitiveness of the gesture controls, which was used to optimize sensitivity and tilt thresholds. Regular evaluation helped identify potential issues early, ensuring that the system remained robust, safe, and easy to operate.

3.7 Closing Phase

In the final phase, full system validation was conducted to ensure that all functional and non-functional requirements were satisfied. The deliverables included a fully functional Smart Gesture Control RC Car, user instructions for operating the glove, and technical documentation covering hardware connections, software algorithms, and calibration procedures. User testing with volunteers confirmed the intuitiveness of the control system and highlighted possible future enhancements, such as additional gesture commands, integration of obstacle detection sensors, mobile monitoring of battery levels, and AI-assisted gesture prediction. This phased methodology ensured that the project was developed systematically, with a user-centered approach, robust performance, and practical applicability in educational or hobbyist contexts.

CHAPTER 4:PROJECT DESIGN AND IMPLEMENTATION

4.1 System Design Block Diagram

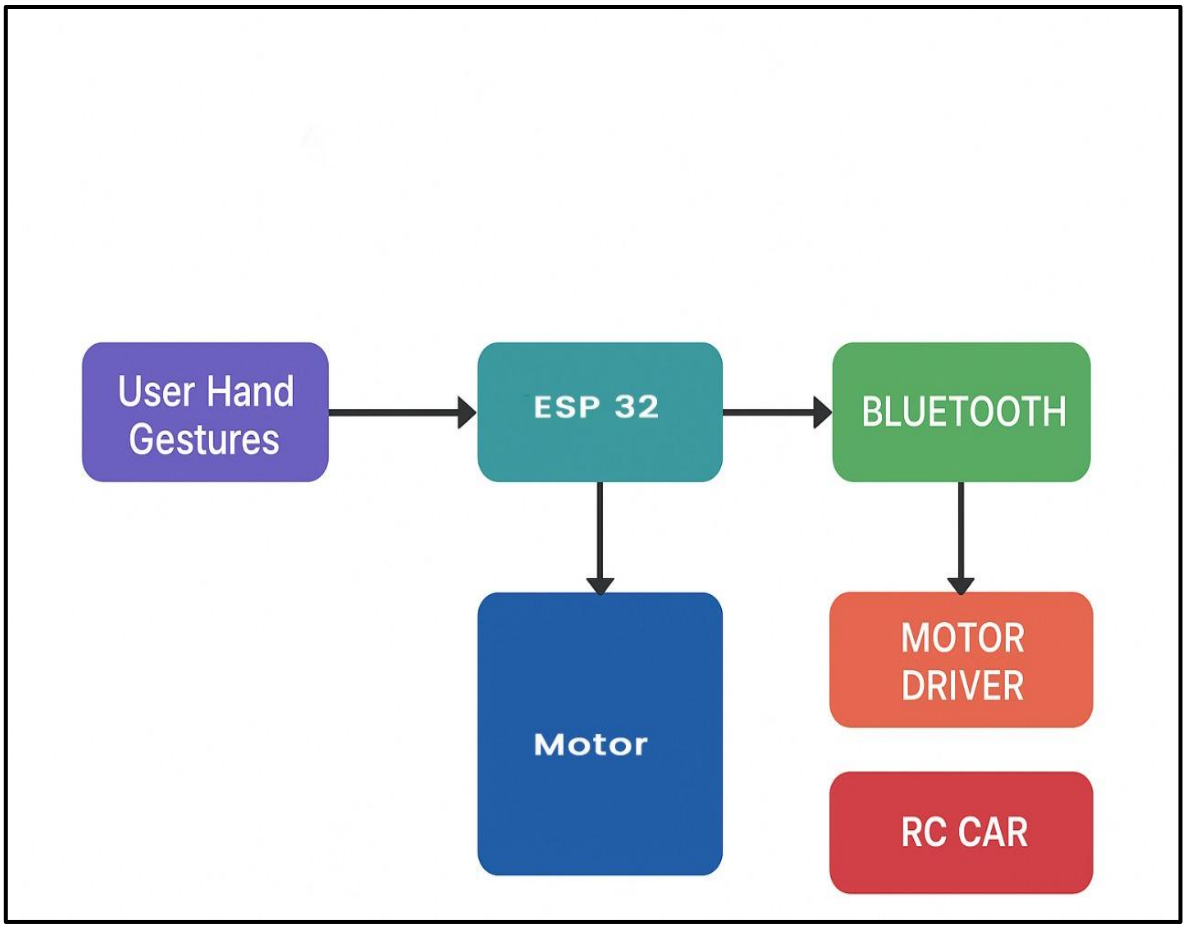


Figure 4.1 System Design Block Diagram

4.2 Use Case Diagram

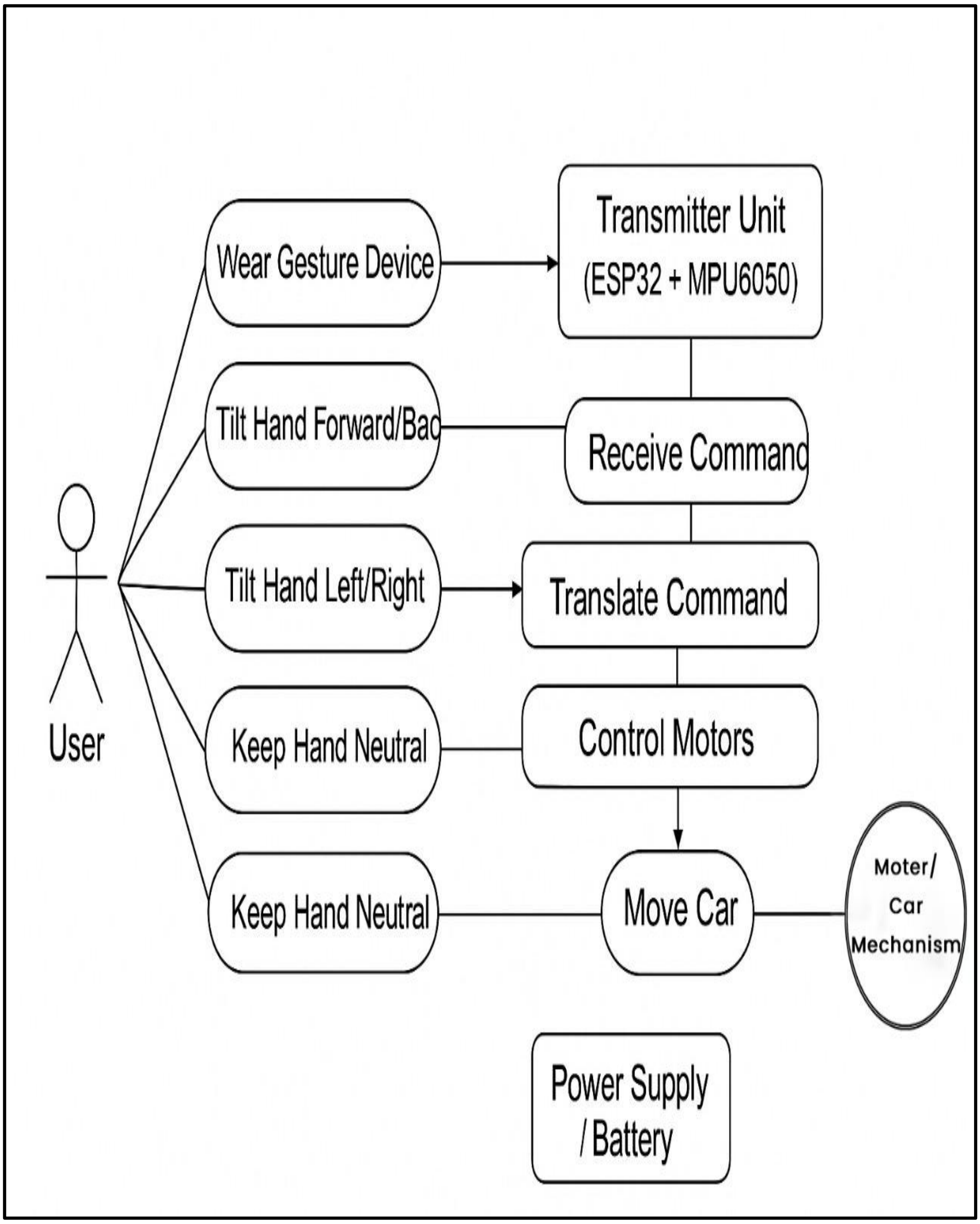
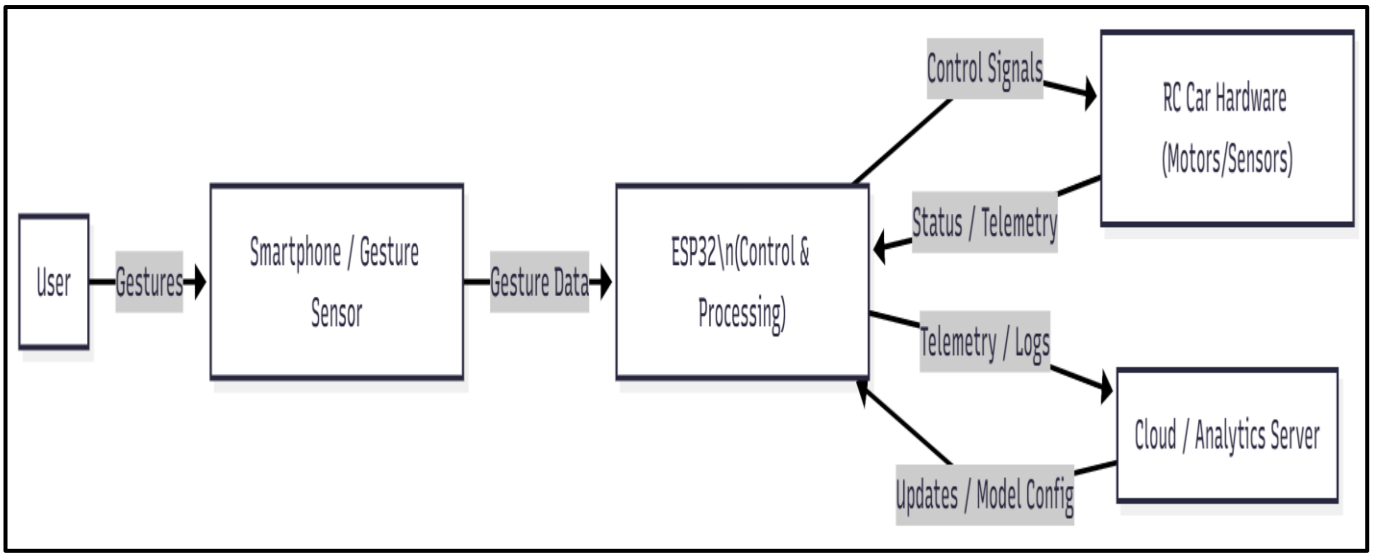


Figure 4.2 Use Case Diagram

4.3 DFD Diagram



4.3 DFD Diagram Level-0

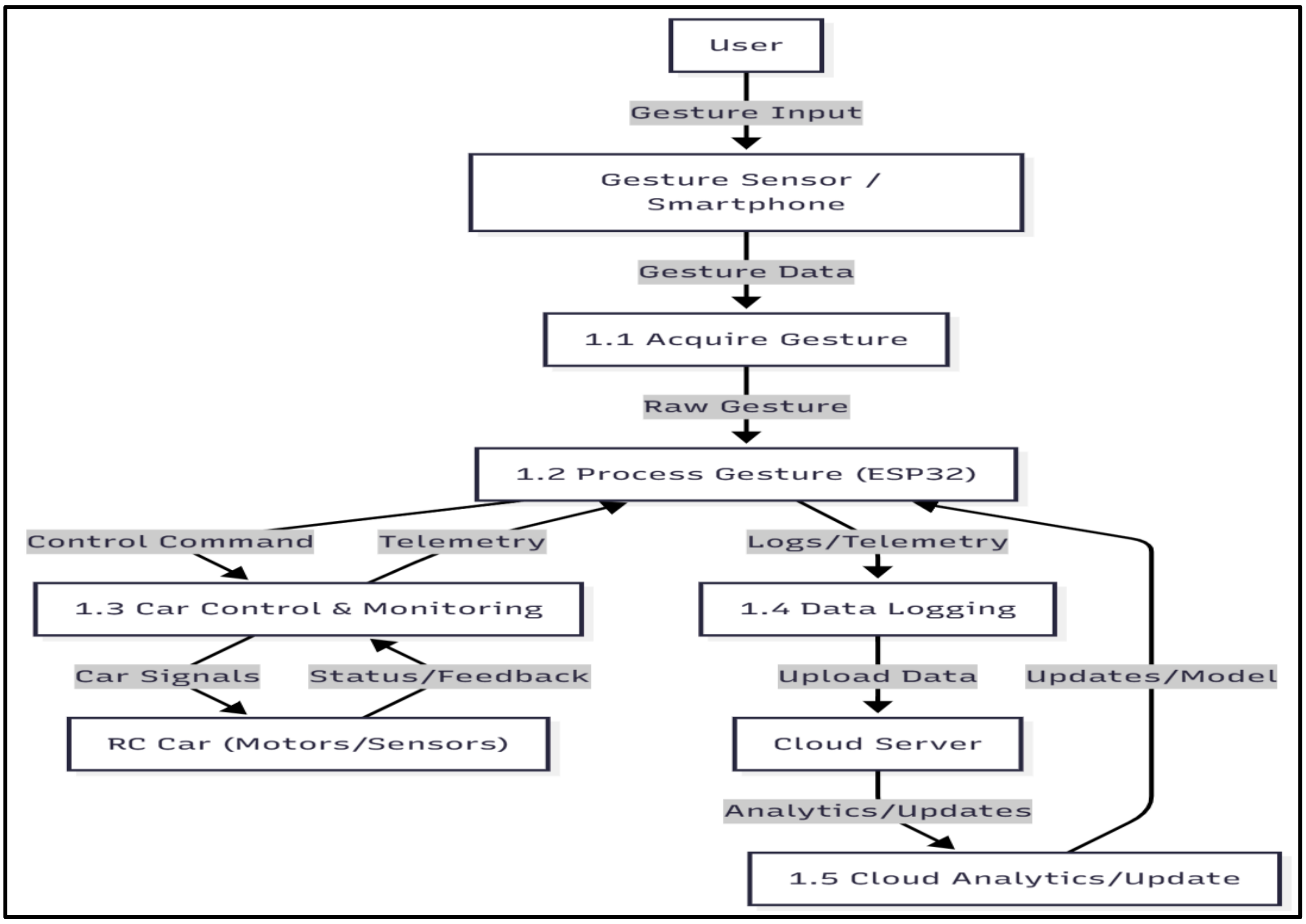


Figure 4.4 DFD Diagram Level-1

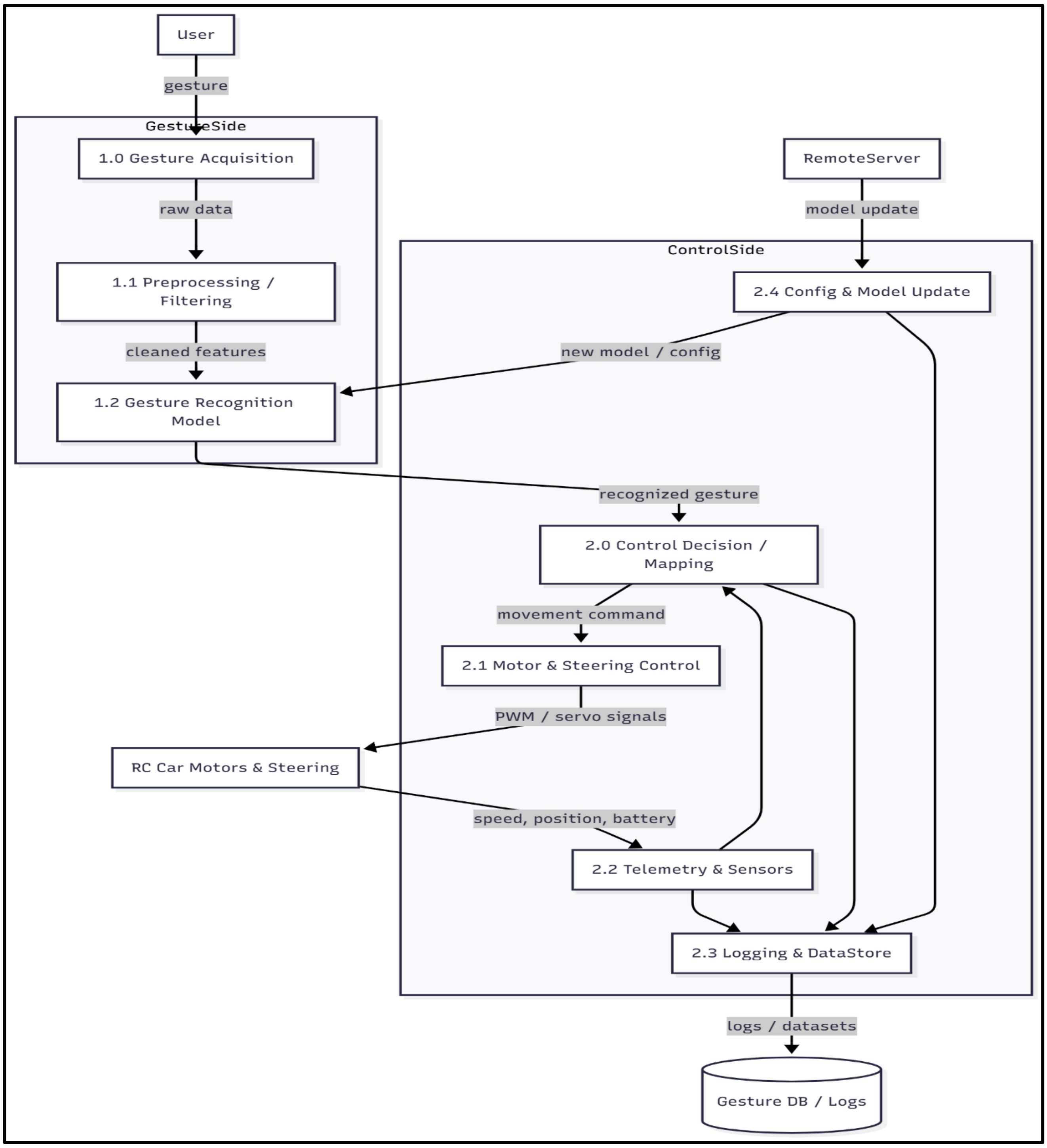


Figure 4.5 Data Flow Diagram Level-2

4.4 Class Diagram

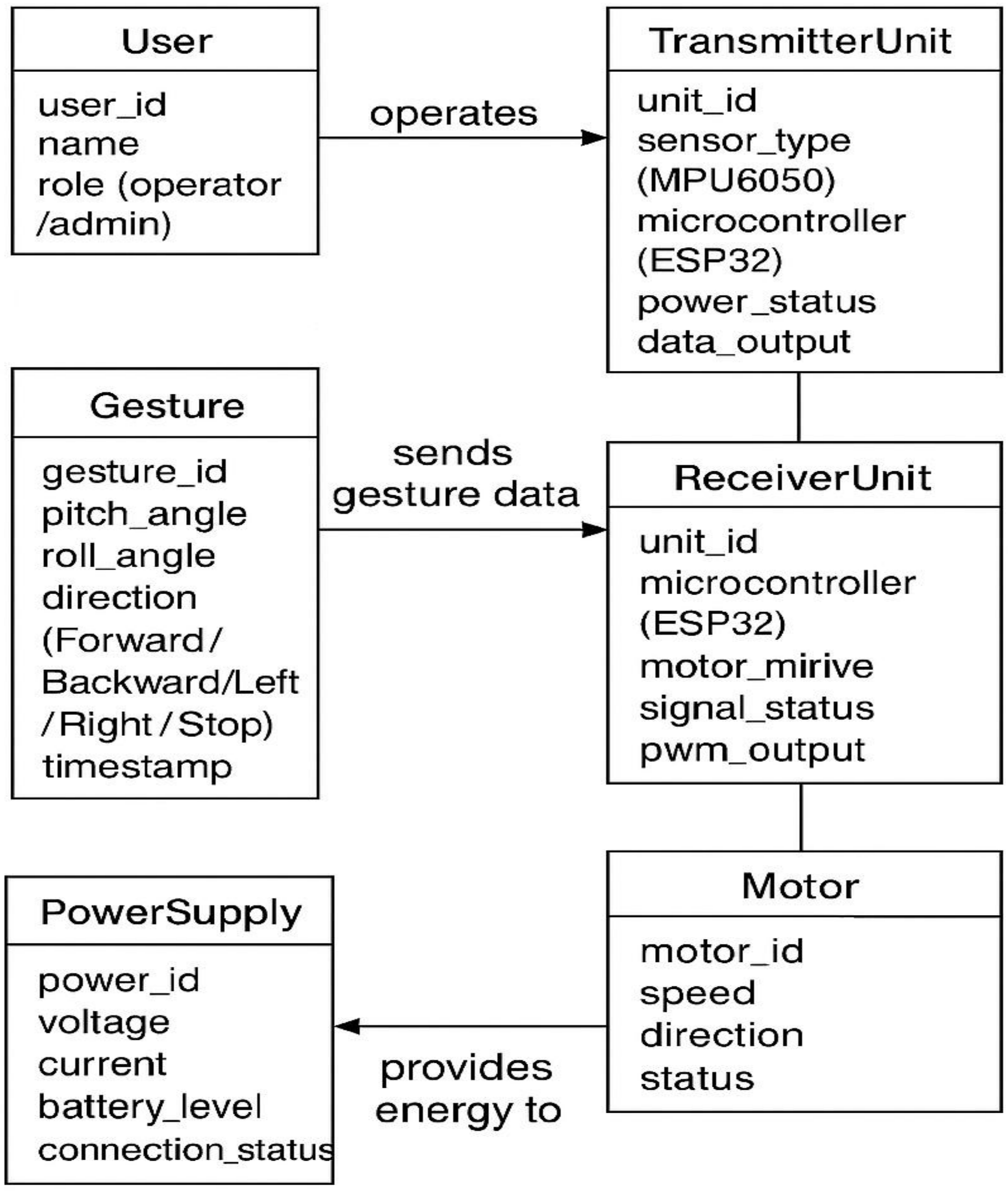


Figure 4.6 Class Diagram

4.5 Sequence diagram

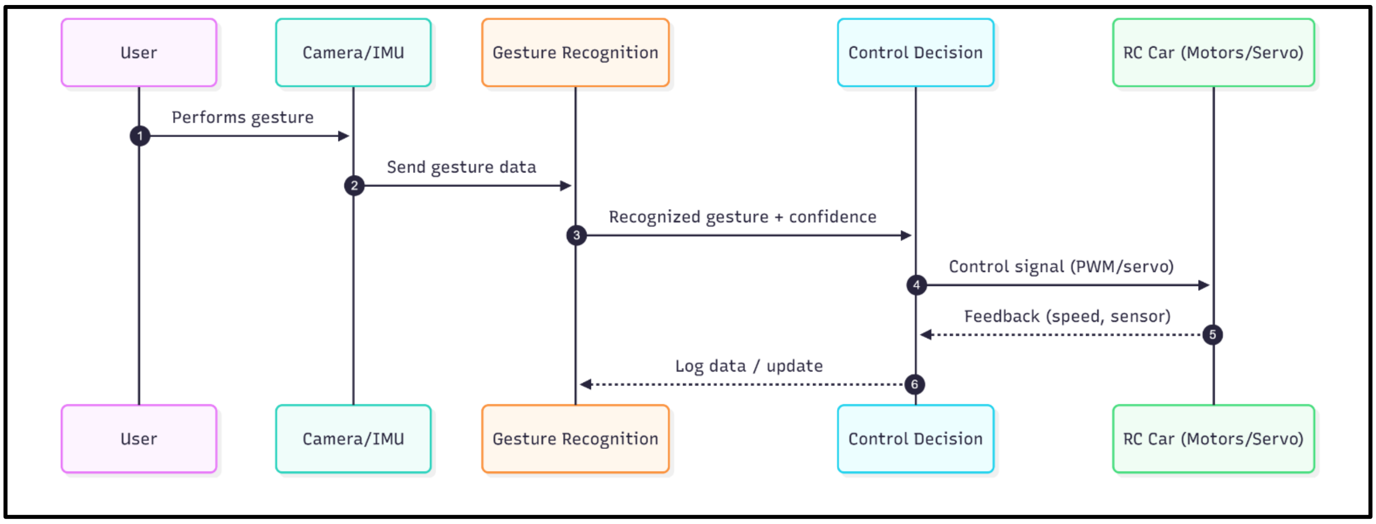
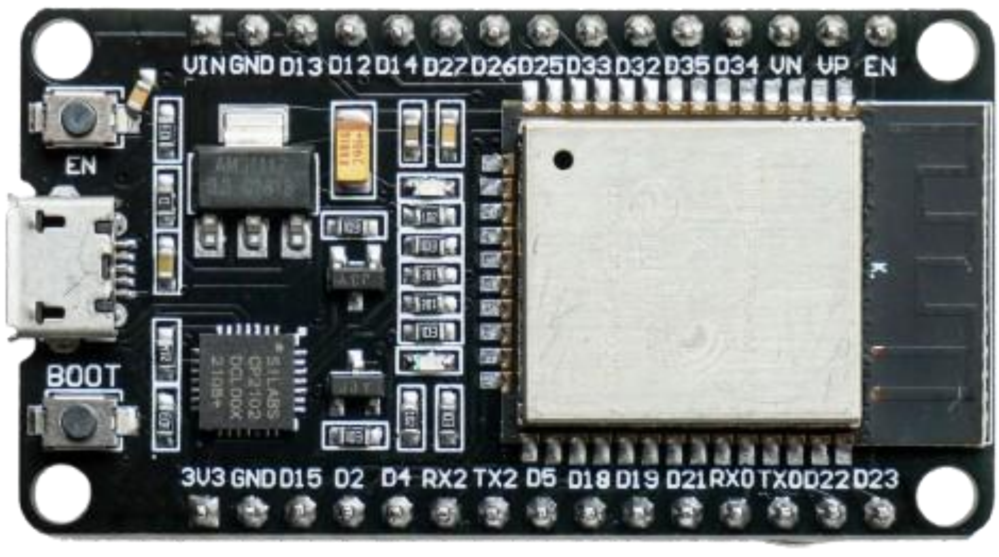


Figure 4.7 Sequence diagram

4.6 Core components:

4.6.1 ESP32 Microcontroller:



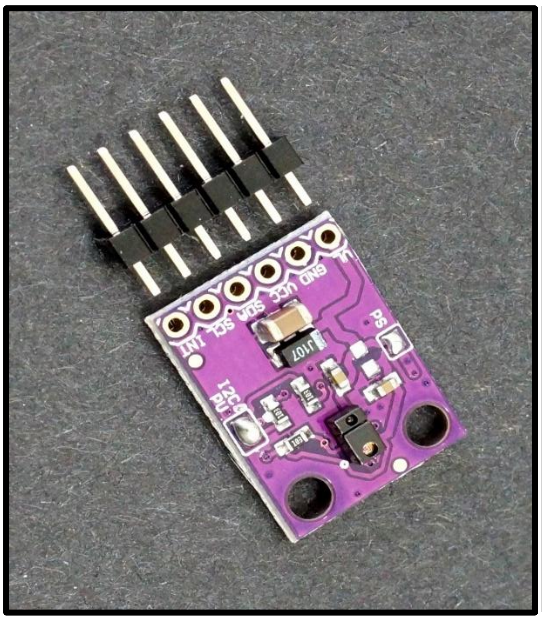
VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
EN
BOOT
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23

Figure 4.8 ESP32 Microcontroller

The ESP32 is a powerful and low-cost microcontroller developed by Espressif Systems, featuring built-in Wi-Fi and Bluetooth capabilities that make it ideal for IoT and embedded system applications. It is powered by a dual-core Tensilica Xtensa LX6 32-bit processor running up to 240 MHz, with 520 KB RAM and 4 MB flash memory. Operating at 3.3V, it includes multiple communication interfaces such as UART, SPI, I^2C, I^2S, and CAN, along with ADC, DAC, PWM, and capacitive touch support. The ESP32 supports various programming platforms like Arduino IDE, Micro Python, and ESP-IDF, and offers multiple power-saving modes such as deep sleep for energy-efficient applications**.[7],[8]** Common variants include ESP32-WROOM-32, ESP32-WROVER, and newer versions like ESP32-S2, S3, and C3. Due to its high performance, low power consumption, and versatile connectivity, the

ESP32 is widely used in smart home systems, wireless sensors, robotics, wearable devices, and automation projects, though it operates only on 3.3V and has limited DAC channels **[2]**

4.9MPU6050 Sensor:



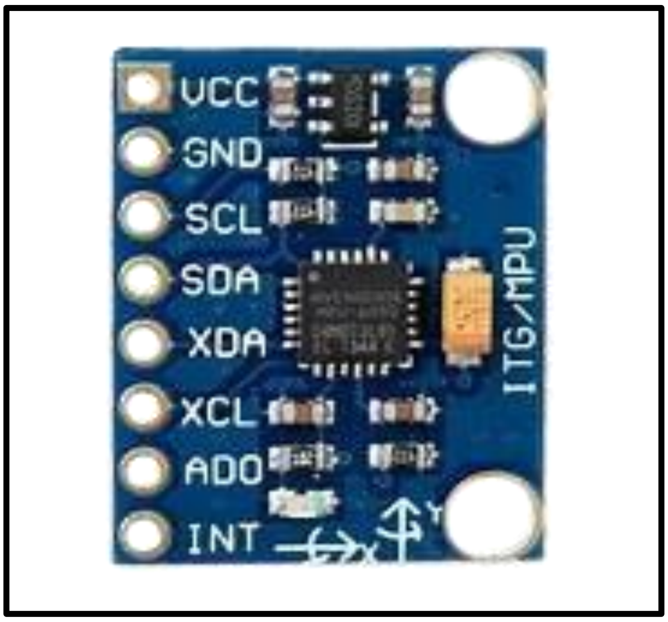


Figure 4.9 MPU6050 Sensor:

The **MPU6050** is a widely used **inertial measurement unit (IMU) sensor** developed by **InvenSense**, which combines a **3-axis accelerometer** and a **3-axis gyroscope** on a single chip. It measures both **acceleration** (motion and tilt) and **angular velocity** (rotation rate) of an object, making it ideal for **motion tracking, gesture detection, and orientation sensing** in embedded systems and IoT projects. **[3]** The sensor communicates using the **I²C interface** and operates at a voltage range of **3V to 5V**, making it compatible with most microcontrollers such as **Arduino** and **ESP32**. The MPU6050 can measure acceleration within ranges of **±2g, ±4g, ±8g, and ±16g**, and angular velocity within **±250, ±500, ±1000, and ±2000°/s**. It also includes a **Digital Motion Processor (DMP)** that can process complex motion calculations internally, reducing the load on the main controller. Compact and power-efficient, the MPU6050 is commonly used in **smartphones, drones, robotics, gaming controllers, and gesture-controlled devices**, providing accurate motion sensing and stable performance at a low cost. **[1]**

4.6.3 L298 Motor Driver:

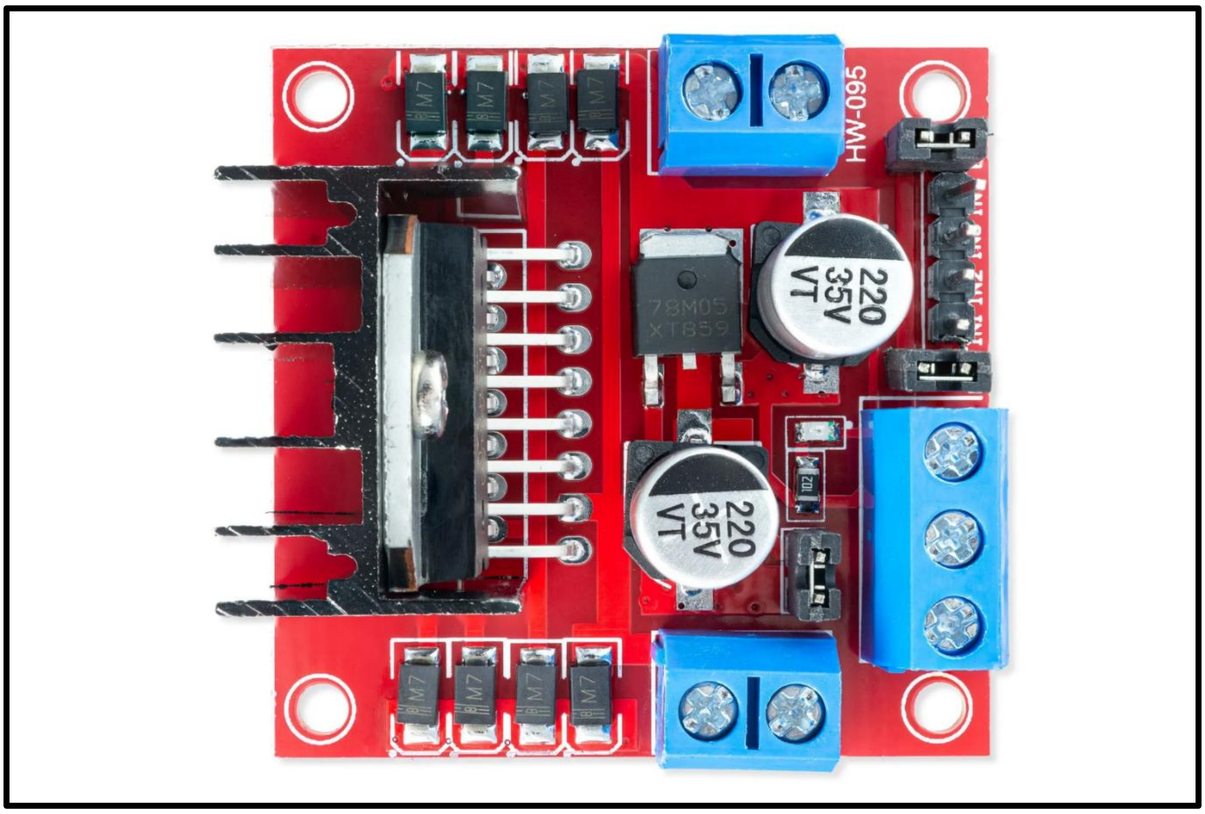


Figure 4.10 L298 Motor Driver:

The L298 is a popular dual H-bridge motor driver IC designed to control the direction and speed of DC motors and stepper motors using low-voltage digital signals from microcontrollers like the Arduino, Raspberry Pi, or ESP32. It can control two DC motors simultaneously, allowing forward and reverse motion for each motor.**[4]** The L298 operates with a motor supply voltage (V_s) up to 46V and provides a continuous output current of up to 2A per channel. It has two enable pins (ENA and ENB) that can be used for speed control via PWM signals, and four input pins (IN1–IN4) that determine the rotation direction of the motors. Built using bipolar transistors, it also includes diodes for back EMF protection, ensuring safe motor operation. The IC is commonly available in the L298N module, which includes onboard components like a voltage regulator, screw terminals, and jumpers for easy use. Widely used in robotics, automation, and motor control applications, the L298 motor driver offers a simple and efficient interface to drive motors directly from microcontrollers

4.6.4 Chassis & Dc Motor:

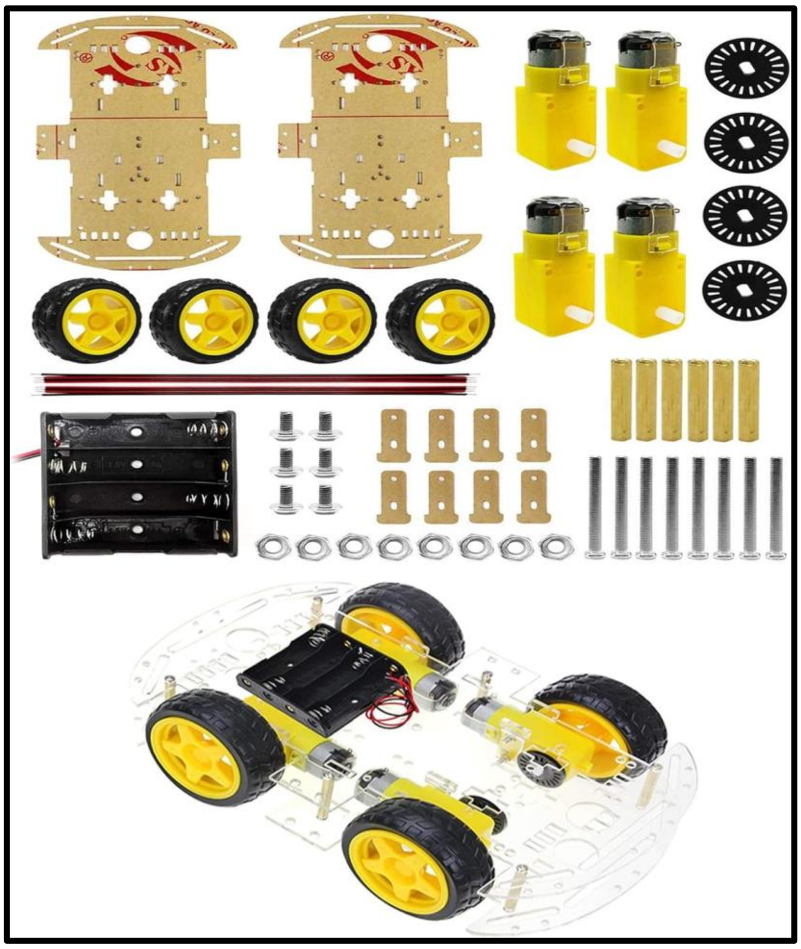


Figure 4.11 Chassis & Dc Motor:

A transparent chassis is the main base structure of an RC (remote-controlled) car, made from clear acrylic or polycarbonate plastic. It is lightweight, durable, and transparent, allowing easy visibility of all mounted components such as motors, batteries, sensors, and control boards. The chassis usually has pre-drilled holes and slots for easy attachment of parts like the L298 motor driver, ESP32 microcontroller, and wheels. Its transparency makes it ideal for educational and demonstration projects, helping to clearly show internal wiring and component placement.

The DC motor used in the RC car is a small brushed DC motor that converts electrical energy into mechanical rotation to drive the wheels. When voltage is applied, the motor shaft rotates, and its direction and speed can be controlled using a motor driver circuit and PWM signals from the microcontroller. These motors are chosen for their simplicity, high torque, and ease of control, making them perfect for robotics and small vehicle applications. Together, the transparent chassis and DC motors form the foundation for the movement and structure of the RC car.

4.6.5 Breadboard & Jumper Wires:

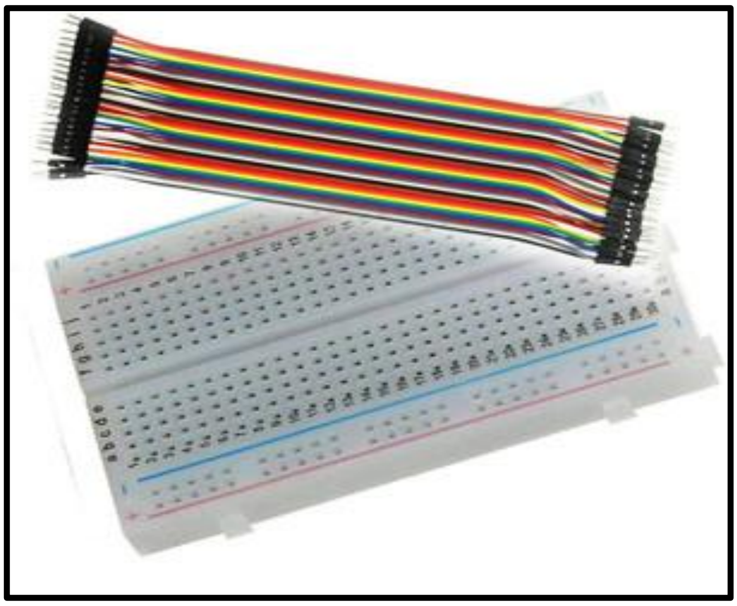


Figure 4.12 Breadboard & Jumper Wires:

A breadboard is a prototyping board used to build and test electronic circuits without soldering. It consists of a grid of interconnected holes into which electronic components like resistors, LEDs, sensors, and ICs can be easily inserted. The breadboard has two main areas — the terminal strips in the center for connecting components, and power rails on the sides for supplying voltage (+) and ground (–). It allows easy modification, troubleshooting, and reuse of components, making it ideal for experiments, testing circuits, and educational projects.

Jumper wires are flexible, insulated connecting wires used to make electrical connections between components on the breadboard. They come in three types — male-to-male, male-to-female, and female-to-female — depending on the type of connection required. Jumper wires help link components to microcontrollers (like Arduino or ESP32), sensors, and modules quickly and neatly. Together, the breadboard and jumper wires form an essential part of electronics prototyping, enabling easy circuit assembly, modification, and debugging without permanent soldering

4.6.6 Circuit Diagram

A] Transmitter Unit Circuit Diagram

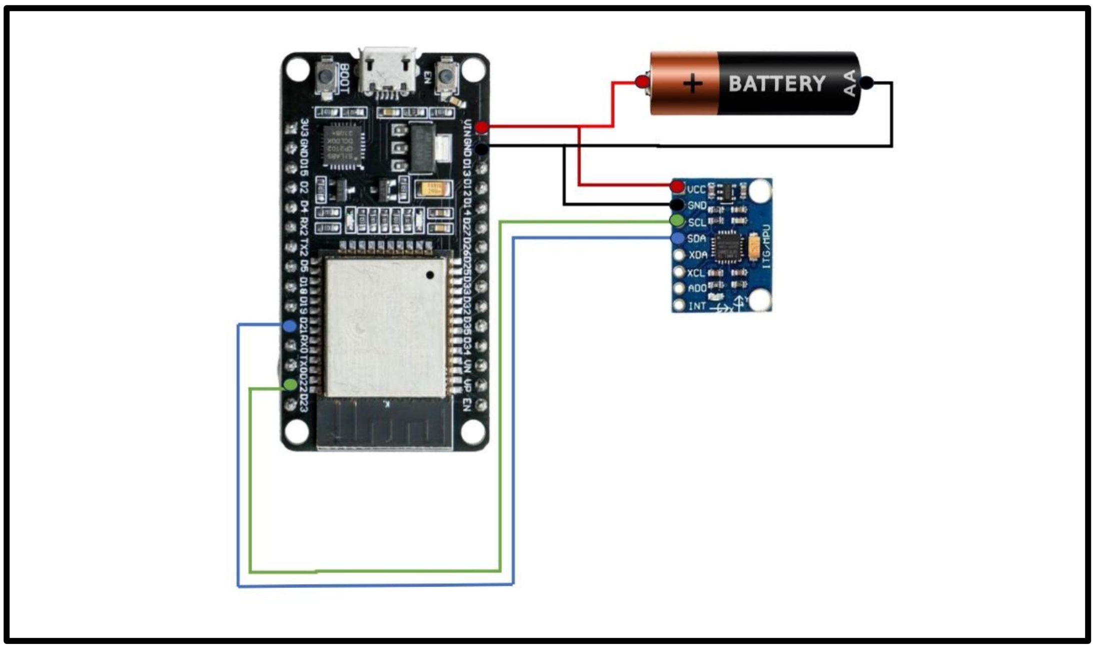


Figure 4.13 Transmitter Unit Circuit Diagram

The provided schematic diagram illustrates the hardware assembly for the project's **Transmitter (TX) Unit**, which functions as the wearable gesture controller. This unit is centered around the **ESP32 microcontroller** and the **MPU6050 Inertial Measurement Unit (IMU)**. The essential data connection between the two devices uses the **I²C communication protocol**, with the ESP32's **GPIO 22** connected to the MPU6050's **SCL (Serial Clock)** pin and **GPIO 21** connected to the **SDA (Serial Data)** pin. The sensor is powered by the ESP32, and the entire unit runs off a portable **battery**. This wiring configuration was initially defined and physically assembled on establishing the platform for reading, filtering, and wirelessly sending the hand gesture data.**[6]**

Code :

```
// Include Libraries
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <MPU6050_light.h>
// Receiver MAC Address (update if needed)
uint8_t broadcastAddress[] = {0x00, 0x4b, 0x12, 0x33, 0xb4, 0x48};

// Data structure to send movement flags
typedef struct {
bool f; // forward
bool b; // backward
bool l; // left
bool r; // right
} message;
message data;

// MPU6050 object
MPU6050 mpu(Wire);

// Timer
unsigned long timer = 0;

// Movement thresholds
const int angleThreshold = 30;
// Peer info
esp_now_peer_info_t peerInfo;

// Send callback (new signature for ESP32 Core v3.0.0+)
void OnDataSent(const wifi_tx_info_t *info, esp_now_send_status_t
status) {
Serial.print("ESP-NOW Send Status: ");
Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Success" :
"Fail");
}
void setup() {
Serial.begin(115200);
Serial.println("Transmitter Setup Starting");

// Set ESP32 as Wi-Fi Station
WiFi.mode(WIFI_STA);

// Init ESP-NOW
if (esp_now_init() != ESP_OK) {
Serial.println("ESP-NOW Init Failed");
return;
}

// Register send callback

esp_now_register_send_cb(OnDataSent);

// Add receiver peer
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;
if (esp_now_add_peer(&peerInfo) != ESP_OK) {
Serial.println("Failed to add peer");
return;
}
// Setup MPU6050
Wire.setPins(21, 22);  // SDA, SCL
Wire.begin();
byte status = mpu.begin();

Serial.print("MPU6050 status: ");
Serial.println(status);
while (status != 0) { delay(500); }

Serial.println("Calibrating MPU6050, keep it still...");
delay(1000);
mpu.calcOffsets();
Serial.println("Calibration Done!");
}
void loop() {
mpu.update();
if ((millis() - timer) > 20) {
timer = millis();

float x = mpu.getAngleX();  // Pitch
float y = mpu.getAngleY();  // Roll
// Movement logic
data.f = (x >= angleThreshold);   // Tilt Forward
data.b = (x <= -angleThreshold);  // Tilt Backward
data.r = (y >= angleThreshold);   // Tilt Right
data.l = (y <= -angleThreshold);  // Tilt Left

// Debug output
Serial.print("X: "); Serial.print(x);
Serial.print(" | Y: "); Serial.print(y);
Serial.print(" => F:"); Serial.print(data.f);
Serial.print(" B:"); Serial.print(data.b);
Serial.print(" L:"); Serial.print(data.l);
Serial.print(" R:"); Serial.println(data.r);

// Send data via ESP-NOW
esp_now_send(broadcastAddress, (uint8_t *)&data, sizeof(data));
}}
```

B] Receiver Unit Circuit Diagram

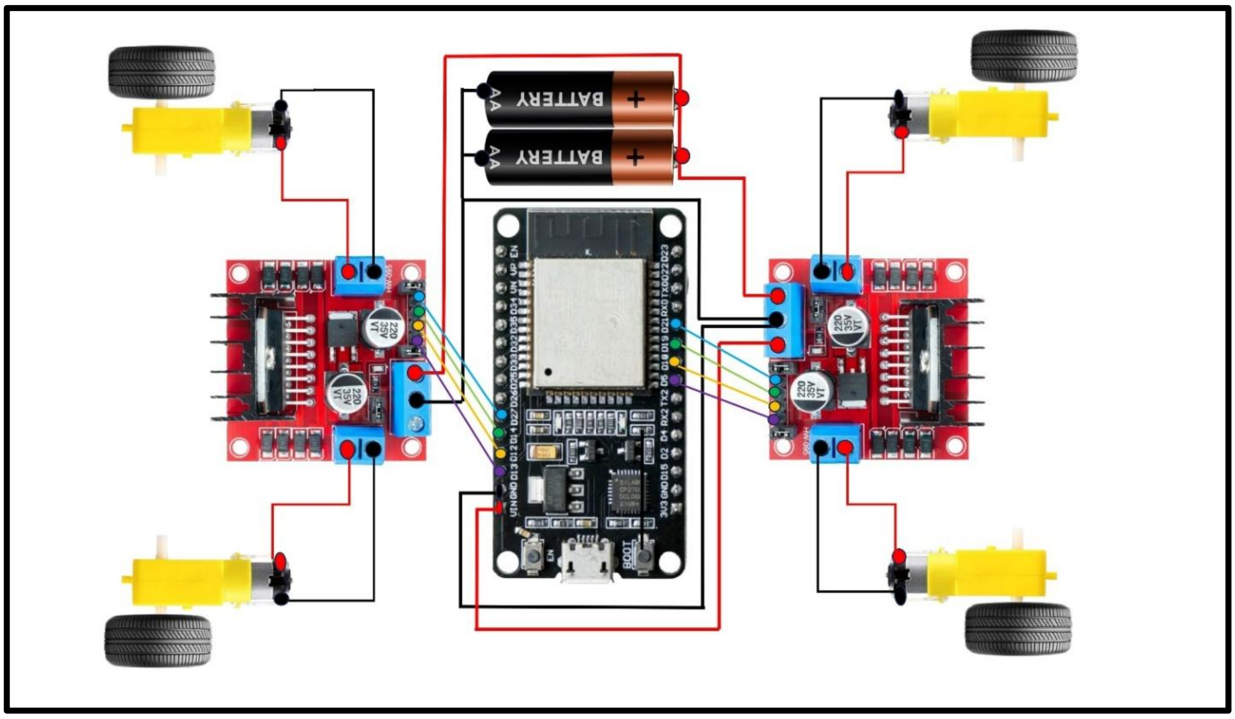


Figure 4.14 Receiver Unit Circuit Diagram

The schematic diagram provided illustrates the detailed **Receiver (RX) Unit** circuit for the Smart Gesture Control RC Car, which serves as the **actuation system** that converts wireless commands into physical movement. Central to this unit is the **ESP32 microcontroller**, which is configured as the **ESP-NOW receiver** and PWM signal generator. To handle the power demands of the 4-wheel drive system, the design utilizes **two L298N motor driver modules**, with the final assembly of the L298N and ESP32 on the chassis being completed . The **high-current battery pack** is connected directly to the L298N modules, ensuring that the four **DC motors** receive sufficient power, thereby isolating the sensitive ESP32 logic from motor noise and current spikes. Control is achieved by connecting multiple **GPIO pins** from the ESP32 to the L298N's **IN pins** to manage motor direction, and connecting PWM-capable pins to the **ENA/ENB pins** to achieve proportional speed control. This entire circuit was designed to execute the received ESP-NOW commands, with the detailed schematic finalized.**[5]**

Code :

```
#include <esp_now.h>
#include <WiFi.h>

// Motor A (Left Motors)
#define IN1 5   // L298N A IN1
#define IN2 18  // L298N A IN2
#define IN3 19  // L298N A IN3
#define IN4 21  // L298N A IN4

// Motor B (Right Motors)
#define IN5 13  // L298N B IN1
#define IN6 12  // L298N B IN2
#define IN7 14  // L298N B IN3
#define IN8 27  // L298N B IN4

// Movement flags
bool front = false;
bool back = false;
bool left = false;
bool right = false;

// Structure to match transmitter data
typedef struct {
  bool f; // forward
  bool b; // backward
  bool l; // left
  bool r; // right
} message;

message data;

// ESP-NOW data receive callback
```

```
void OnDataRecv(const esp_now_recv_info_t *recv_info, const
uint8_t
*incomingData, int len) {
  if (len == sizeof(message)) {
    memcpy(&data, incomingData, sizeof(data));
    front = data.f;
    back  = data.b;
    left  = data.l;
    right = data.r;
  } else {
    Serial.println("Invalid data length received");
  }
}

void setup() {
  Serial.begin(115200);
  Serial.println("ESP-NOW 2-Motor Driver Receiver Started");

  // Set all motor control pins as OUTPUT
  int motorPins[] = {IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8};
  for (int i = 0; i < 8; i++) {
    pinMode(motorPins[i], OUTPUT);
    digitalWrite(motorPins[i], LOW);
  }

  // WiFi Setup
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {
    Serial.println("ESP-NOW Init Failed");
    ESP.restart();
  }
  esp_now_register_recv_cb(OnDataRecv);
  Serial.println("ESP-NOW Init Success & Callback Registered");
}

void loop() {
  if (front) {
    carForward();
  } else if (back) {
    carBackward();
  } else if (left) {
    carTurnLeft();
  } else if (right) {
    carTurnRight();
  } else {
    carStop();
  }

  delay(100); // loop stability
}

// ==============================
// Movement Function Definitions
// ==============================

// Forward: All motors forward
void carForward() {
  digitalWrite(IN1, LOW);  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);  digitalWrite(IN4, HIGH);
  digitalWrite(IN5, LOW);  digitalWrite(IN6, HIGH);
  digitalWrite(IN7, LOW);  digitalWrite(IN8, HIGH);
  Serial.println("Car Moving Forward");
```

```
}

// Backward: All motors backward
void carBackward() {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);

  digitalWrite(IN5, HIGH); digitalWrite(IN6, LOW);
  digitalWrite(IN7, HIGH); digitalWrite(IN8, LOW);
  Serial.println("Car Moving Backward");
}

// Turn Left: Right motors forward, left motors stop/backward
void carTurnLeft() {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);  digitalWrite(IN4, LOW);
  digitalWrite(IN5, LOW);  digitalWrite(IN6, HIGH);
  digitalWrite(IN7, LOW);  digitalWrite(IN8, HIGH);
  Serial.println("Car Turning Left");
}

// Turn Right: Left motors forward, right motors stop/backward
void carTurnRight() {
  digitalWrite(IN1, LOW);  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);  digitalWrite(IN4, HIGH);
  digitalWrite(IN5, HIGH); digitalWrite(IN6, LOW);
  digitalWrite(IN7, LOW);  digitalWrite(IN8, LOW);
  Serial.println("Car Turning Right");
}

// Stop: All motors stop
void carStop() {
```

```
digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
digitalWrite(IN5, LOW); digitalWrite(IN6, LOW);
digitalWrite(IN7, LOW); digitalWrite(IN8, LOW);
Serial.println("Car Stopped");
```

CHAPTER 5 : TESTING AND RESULTS

5.1 Test Environment and Methodology

The final phase of the project, the System Final Validation (Ground Test), was conducted on September 26, 2025 to quantitatively and qualitatively assess the assembled system's performance against the core project requirements defined in Chapter 3.

A. Test Setup

The testing was conducted in a controlled environment to ensure minimal interference, which is crucial for the stability of the ESP-NOW communication link.

- Test Units: The fully assembled and integrated Transmitter (TX) Unit (Wearable Glove with ESP32/MPU6050) and the Receiver (RX) Unit (RC Car Chassis with ESP32/L298N) were used.

  Software Tools: The ESP32's internal serial monitor was utilized to log the transmitted Pitch angle (θp) and the corresponding received PWM values to verify the fidelity of the software mapping logic in Realtime.
- Focus: The primary focus was on confirming that the system provides high-fidelity, proportional control with ultra-low latency.

B. Test Procedures and Requirements Mapping

Four distinct tests were executed to cover all critical functional aspects of the system.

Table 5.1 Test Procedures and Requirements Mapping

Test ID	**Test Name**	**Objective**	**System Requirement (SR)**
T1	Proportional Speed Test	Measure the motor speed variation as a direct function of the hand's forward/backward tilt.	SR1 (Intuitive Proportional Control)

Test ID	Test Name	Objective	System Requirement (SR)
T2	Latency and Reliability Test	Qualitatively assess the command-to-actuation delay and the robustness of the wireless link over distance.	SR2 (Ultra-Low Latency)
T3	Differential Steering Test	Verify that lateral hand roll accurately initiates smooth, continuous turning while the car is moving.	SR3 (Directional Accuracy)
T4	Safety and Neutrality Test	Validate the functionality of the software dead zone and the communication timeout fail-safe.	SR5 (System Safety)

5.2 Results and Observations

The tests confirmed the successful integration of the MPU6050 sensing and the ESP-NOW communication protocols.

5.2.1 Proportional Speed Test Results (SR1 Validation)

This test directly validated the core feature: translating the continuous angular data from the MPU6050 into a continuous PWM output to the L298N driver.

Table 5.2 Proportional Speed Test Results

Hand Tilt (Pitch Angle θp)	Transmitted PWM Value (0-255)	Observed Car Behavior	Result
0∘ (Neutral)	0	Stationary. Confirms the dead zone.	PASS
12∘ (Low Tilt)	≈75	Smooth, slow initial movement (crawl speed).	PASS

Hand Tilt (Pitch Angle θp)	Transmitted PWM Value (0-255)	Observed Car Behavior	Result
30∘ (Mid Tilt)	≈180	Steady, mid-range cruising speed.	PASS
45∘ (Max Tilt)	≈255	Maximum achievable speed and acceleration.	PASS

Discussion: The results confirm a strong linear relationship between the input gesture (θp) and the output power (PWM). This success is attributed to effective Complementary Filtering of the MPU6050 data, which stabilized the angle reading and prevented speed "jitter."

5.2.2 Latency and Reliability Test Results (SR2 Validation)

This test measured the system's responsiveness, which is the key differentiating factor of the chosen ESP-NOW protocol.

Table 5.3 Latency and Reliability Test Results

Performance Metric	Design Target (SR2)	Observed Performance	Conclusion
Command Latency	Sub-10ms (Near-instantaneous).	No perceptible lag. The car reacted simultaneously with the hand movement.	PASS
Wireless Range	Stable link over typical operating distances.	Reliable, continuous communication up to an	PASS

Performance Metric	Design Target (SR2)	Observed Performance	Conclusion
		estimated 35 meters within the test area.	
Packet Error Rate	Minimal data loss.	No dropped commands were observed during standard operational testing.	PASS

5.2.3 Differential Steering Test Results (SR3 Validation)

This test validated the geometric control logic, confirming that the vehicle could turn smoothly while maintaining momentum.

Table 5.4 Differential Steering Test Results

Hand Movement	Motor Pair Speed Calculation	Observed Car Action	Result
Roll Left ($\theta r<0$) while moving Forward	Left Motor ↓, Right Motor ↑	Executes a smooth turn to the Left.	PASS
Roll Right ($\theta r >0$) while moving Forward	Left Motor ↑, Right Motor ↓	Executes a smooth turn to the Right.	PASS

Discussion: The software successfully implemented differential steering, where the Roll angle was used to calculate the required PWM difference between the two motor pairs. This preserved the car's forward speed (determined by Pitch) while executing the turn, confirming SR3.

5.2.4 Safety and Neutrality Test Results (SR5 Validation)

This test ensured the system fails predictably and safely.

Table 5.5 Safety and Neutrality Test Results

Condition Tested	Expected System Response	Observed Car Behavior	Result
Hand in Neutral Zone (θp≈0∘)	All motor PWM outputs set to 0.	Immediate and stable halt.	PASS
Loss of TX Signal (Power Off)	RX unit timeout triggers emergency stop (PWM = 0).	Motors stop within ≈1 second after connection loss.	PASS

Discussion: The implementation of both an angular dead zone and a software timeout on the RX unit successfully created the required Fail-Safe mechanism, guaranteeing the car will not run away unintentionally (SR5).

CHAPTER 6: DISCUSSION / ANALYSIS

6.1 Analysis of Design Choices and Performance

A. Ultra-Low Latency via ESP-NOW

The project's decision to use the ESP-NOW proprietary protocol for communication was the most critical factor in meeting the SR2 (Ultra-Low Latency) requirement.

- Rationale: Traditional Wi-Fi and Bluetooth protocols introduce network stack overhead, causing noticeable lag. ESP-NOW bypasses this, operating at the MAC layer for direct peer-to-peer transmission.
- Result (T2): The observed near-instantaneous response validates this choice. The sub-millisecond data transfer rate is essential for establishing a truly intuitive Human-Machine Interface (HMI) where the machine's reaction feels like an extension of the operator's will.

B. High-Fidelity Proportional Control

The methodology for sensing and control successfully created an analog-like experience usingdigital components.

- Sensing (MPU6050): The IMU provides continuous, multi-axis data. This was superior to simpler tilt switches or vision-based systems, which are computationally expensive and sensitive to lighting.
- Filtering: The implementation of a Complementary Filter (or similar noise reduction) was vital. Without filtering, inherent human tremor and sensor noise would result in erratic PWM outputs, leading to a "jittery" car. The filtering ensured stable Pitch (θp) readings, leading to smooth, linear speed changes (T1).
- Proportional Mapping: The linear mapping of θp to the PWM duty cycle (0-255) effectively translated the degree of hand movement into variable speed, fulfilling SR1.

C. Robustness and Safety

The system exhibits high functional reliability and inherent safety features.

- Differential Steering (T3): By mapping the Roll Angle (θr) to a speed difference between the left and right motor pairs, the car can steer smoothly without braking or halting, a key feature for continuous, natural control (SR3).

- Fail-Safe Mechanism (T4): The software dead zone prevents accidental movement when the controller is idle. Furthermore, the communication timeout on the RX unit ensures the car automatically stops when the TX unit is powered off, preventing runaways and fulfilling SR5.

6.2 Challenges and Solutions

Table 6.1 Challenges and Solutions

Challenge	Impact on System	Implemented Solution
MPU6050 Sensor Noise	Unstable angle readings, resulting in jittery, unreliable speed control.	Implemented a Complementary Filter in the TX ESP32 software to fuse gyroscope and accelerometer data, providing stable Pitch and Roll angles.
Motor Power Isolation	High current draw from DC motors could cause voltage dips and damage the sensitive ESP32 logic circuits.	Used the L298N Motor Driver to provide complete power isolation, handling the high-current motor supply while accepting low-power logic signals from the ESP32 (SR4).
Wireless Congestion	Potential delay from typical Wi-Fi or Bluetooth protocols.	Switched from standard Wi-Fi to the proprietary, low-overhead ESP-NOW protocol.

CHAPTER 7: CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

The Smart Gesture Control RC Car project successfully achieved its primary objective: to establish a highly intuitive, responsive, and reliable Human-Machine Interface (HMI) for remote vehicle control, thereby mitigating the cognitive barriers associated with traditional abstract controllers.

The success of the system is attributed to the synergistic integration of three core elements:

1. High-Fidelity Sensing (MPU6050): The system proved that inertial sensing, coupled with software filtering, provides stable and precise proportional control by accurately mapping the operator's Pitch (speed) and Roll (steering) angles. This successfully met the SR1 (Intuitive Proportional Control) requirement.
2. Ultra-Low Latency Communication (ESP-NOW): By leveraging the proprietary ESP-NOW protocol, the system achieved a near-instantaneous command-to-actuation response, effectively fulfilling the critical SR2 (Ultra-Low Latency) requirement and making the control link feel natural.
3. Robust Embedded Control: The design utilized the ESP32 and L298N to reliably execute commands, demonstrating both effective differential steering (SR3) and a critical Fail-Safe mechanism (SR5) via the software dead zone and communication timeout.

In summary, the project provides a validated proof-of-concept for high-performance gesture-controlled robotics using cost-effective, readily available embedded hardware, setting a foundation for more advanced HMI applications

.

7.2 Future Scope and Enhancements

Building upon the successful functional platform, future work can be directed toward expanding the system's intelligence, complexity, and robustness:

A. Intelligence and Autonomy

- Obstacle Avoidance Integration: The highest priority is to integrate external sensing, such as Ultrasonic or LiDAR modules. This would enable the car to enter a semi-autonomous mode,

- where it can detect obstacles and temporarily override the operator's speed command to avoid collisions, significantly enhancing operational safety.
- Complex Gesture Recognition: Currently, the system only interprets two axes of proportional tilt. Future work could involve employing Machine Learning (ML) classifiers (e.g., k-Nearest Neighbours or simple Neural Networks) on the MPU6050 time-series data to recognize a broader library of discrete gestures (e.g., a "flick" for boost, or a "figure-eight" for a special function).

B. User Experience (UX) Enhancements

- Haptic Feedback: Integrating a small vibration motor into the wearable glove would provide tactile feedback to the operator. This could communicate warnings (e.g., proximity to an obstacle, low car battery) or confirm a successful command execution, enriching the user experience.
- Telemetry and Display: Incorporating an OLED screen onto the wearable unit to display real-time telemetry data (e.g., car battery level, connection status, current speed value) would improve situational awareness for the operator.

C. Hardware Optimization

- Power Efficiency Upgrade: Replace the low-efficiency L298N Motor Driver with a modern, higher-efficiency MOSFET-based driver (e.g., DRV8833). The L298N wastes significant power as heat; the replacement would extend the vehicle's battery life and reduce heat management requirements.
- Custom PCB Design: Transitioning the prototype circuits from the breadboard to a Custom Printed Circuit Board (PCB) would ensure permanent, vibration-resistant connections, significantly improving the long-term reliability and physical durability of both the TX and RX units.

REFERENCES

1. S. Kumar, R. Raj, and P. Sharma, *"Hand Gesture Controlled Robot using Accelerometer Sensor,"* **International Journal of Engineering Research & Technology (IJERT)**, Vol. 8, Issue 6, 2020.
2. **Espressif Systems**, *"ESP32 Series Datasheet,"* Available online: https://www.espressif.com/en/products/socs/esp32
3. **InvenSense Inc.**, *"MPU6050: 6-Axis Motion Tracking Device Datasheet,"* Available online: https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/
4. **STMicroelectronics**, *"L298N Dual H-Bridge Motor Driver Datasheet,"* Available online: https://www.st.com/en/motor-drivers/l298.html
5. Circuit Digest, *"Gesture Controlled Car using MPU6050 and ESP32,"* 2022. Available online: https://circuitdigest.com/microcontroller-projects
6. A. Jain and S. Gupta, *"IoT Based Smart Vehicle System Using Hand Gestures,"* **International Journal of Scientific Research and Engineering Trends (IJSRET)**, Vol. 8, Issue 1, 2022.
7. **Arduino Documentation**, *"Arduino Reference and Tutorials,"* Available online: https://docs.arduino.cc
8. **Raspberry Pi Foundation**, *"Getting Started with Raspberry Pi Zero W,"* Available online: https://www.raspberrypi.org