Wireless Hand Gesture Controlled Car Using ESP32

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In

Electronics & Communication Engineering

By

Sayan Ray (002-BEC-2020-008)

Swami Vivekananda University
(Affiliated to UGC)
Barrackpore, North 24 Parganas, West Bengal, 700121

BONAFIDE CERTIFICATE

This is to certify that the project titled Wireless Hand Gesture Controlled Car Using ESP32 is a bonafide record of the work done by

Sayan Ray (002-BEC-2020-008)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Electronics & Communication Engineering** of the **Swami Vivekananda University, Barrackpore,** during the year 2020-24.

Mrs. Shreya Adhikari
Guide

Mrs. Shreya Adhikari
Head of the Department

Project Viva-voce held on _____.

_____
Internal Examiner

_____
External Examiner

THESIS CERTIFICATE

This is to certify that the thesis entitled **Wireless Hand Gesture Controlled Car Using ESP32** submitted by **Sayan Ray** to the Swami Vivekananda University, barrackpore for the award of the degree of **B. tech in ECE** is a bonafide record of research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Kolkata.
Date:

Shreya Adhikari
Guide
Department of Electronics & Communication Engineering
Swami Vivekananda University
Barrackpore, North 24 Parganas, West Bengal, 700121

Debasis Mondal
Co-Guide

Abstract

This paper presents an investigation into the integration of gesture control technology with robotics, employing the ESP32 WiFi module for wireless communication and control. Gesture control offers an intuitive means of human-robot interaction, enabling users to command robots through hand movements or gestures. The ESP32 module facilitates real-time wireless connectivity, allowing for seamless communication between the robot and the user interface. This study explores the design, implementation, and testing of a gesture-controlled robot system, highlighting the role of the ESP32 module in enabling reliable and responsive control. The system's potential applications in healthcare, industrial automation, and entertainment are discussed, along with avenues for future research and development. Through this work, the feasibility and effectiveness of gesture-controlled robotics using the ESP32 WiFi module are demonstrated, offering insights into its significance in enhancing human robot interaction in diverse domains.

ACKNOWLEDGEMENT

I would like to express my profound gratitude to Swami Vivekananda University for fostering an enriching academic environment that has been crucial to the successful completion of this research project. The institution's commitment to excellence has been a constant source of inspiration.

Special thanks go to my supervising teacher, Mrs. Shreya Adhikari, for her exceptional guidance and unwavering support throughout this research journey. Her insightful feedback and mentorship have significantly contributed to the depth and quality of this study, shaping my understanding of the subject.

I am also thankful for the invaluable resources and facilities provided by the institute, which have played a pivotal role in the execution of this research. The knowledge and skills acquired during this academic pursuit will undoubtedly serve as a strong foundation for my future endeavors.

Name: Akash Ghorai
Roll No.: 002-BEC-2020-001
Reg. No.: 002-105-2020-001

Name: Abhisek Chakraborty
Roll No.: 002-BEC-2020-003
Reg. No.: 002-105-2020-003

Name: Sayan Ray
Roll No.: 002-BEC-2020-008
Reg. No.: 002-105-2020-008

Name: Priyanshi Das
Roll No.: 002-BEC-2020-006
Reg. No.: 002-105-2020-006

CONTENTS

Background

In recent decades, the convergence of advancements in microelectronics, sensor technologies, and wireless communication has propelled the field of robotics and automation to unprecedented heights. One notable area of innovation within this domain is the development of intuitive human-machine interfaces (HMIs) that enable users to interact with robots and other autonomous systems seamlessly. Traditional control interfaces, such as joysticks and keyboards, often require users to undergo a learning curve and may not always provide an intuitive means of interaction, especially in dynamic or constrained environments.

Gesture control technology is becoming increasingly popular in various fields such as gaming, robotics, and human-computer interaction. This project leverages this technology to control a car using hand movements. The ESP32 microcontroller, with its integrated Wi-Fi and Bluetooth capabilities, and the MPU6050 gyro module, known for its accuracy in capturing motion data, provide a robust foundation for this project.

Motivated by the potential of gesture-based control systems to revolutionize human-robot interaction, this thesis focuses on the development of a wireless hand gesture controlled car using the ESP32 microcontroller. By combining state-of-the-art gesture recognition algorithms with off-the-shelf hardware components, the project aims to demonstrate the feasibility and effectiveness of utilizing hand gestures as an intuitive interface for controlling robotic devices. Through rigorous experimentation and evaluation, this research endeavors to contribute to the advancement of gesture-based control systems and their integration into real-world robotics applications.

Problem Statement

Developing a Wireless Hand Gesture Controlled Car Using ESP32 involves creating a system that accurately recognizes hand gestures to control the car wirelessly. Challenges include ensuring real-time communication, optimizing power usage, designing an intuitive interface, implementing obstacle detection, and ensuring scalability for future enhancements.

The primary goal is to develop a car that can be controlled wirelessly through hand gestures. This involves accurately capturing and interpreting hand movements and translating these into precise motor controls to drive the car in the desired direction.

Chapter 1

INTRODUCTION

In the dynamic landscape of robotics, the quest for enhancing human-robot interaction (HRI) remains at the forefront of innovation. Traditional control interfaces, characterized by physical buttons or joysticks, often present limitations in terms of usability, accessibility, and adaptability to diverse user scenarios. As robotics continues to permeate various sectors, from healthcare to industrial automation, the demand for intuitive and accessible control methods becomes increasingly pronounced.
Gesture control technology emerges as a beacon of promise in addressing these challenges, offering a natural and intuitive interface for commanding robots through hand movements or gestures. By transcending the constraints of physical interfaces, gesture control systems enable users to interact with robots effortlessly, akin to human-to-human communication. This paradigm shift not only enhances user experience but also expands the horizons of robotic applications, particularly in environments where physical interaction is restricted or impractical.

The project aims to revolutionize conventional car control mechanisms by introducing a Wireless Hand Gesture Controlled Car utilizing ESP32. This innovative system enables users to command the car's movement through intuitive hand gestures, eliminating the need for physical buttons or joysticks. Leveraging ESP32's capabilities, the system ensures real-time communication, efficient power management, and seamless integration with gesture recognition algorithms. By combining advanced technology with user-friendly design, this project promises to offer a novel and engaging driving experience while paving the way for future advancements in gesture-controlled automotive systems.

This project explores the creation of a wireless hand gesture-controlled car using ESP32 microcontrollers, MPU6050 for motion sensing, and the L298N motor driver for controlling the car's motors. The project showcases the integration of various components and the use of ESPNOW protocol for communication.

Chapter 2

Used Components

2.1. Transmitter part:

- ESP32 Module: The ESP32 serves as the central processing unit, responsible for receiving data from the MPU6050 sensor, interpreting hand gestures, and transmitting corresponding control commands to the car via ESPNOW (Espress if's proprietary protocol for low-power and peer-to-peer communication).

- MPU6050: The MPU6050 is a gyroscope and accelerometer module utilized to detect hand tilt angles accurately. It captures subtle hand movements and translates them into digital signals, providing real-time input data for controlling the car's movements.

- 5V DC Supply: This power source ensures the continuous operation of both the ESP32 module and the MPU6050 sensor, supplying the necessary electrical energy to power these components throughout the gesture recognition and transmission process.

- Breadboard and Jumper Wires: These prototyping tools facilitate the connection between the ESP32 module, MPU6050 sensor, and power supply. The breadboard provides a convenient platform for temporary circuit construction, while jumper wires establish electrical connections between the various components, enabling seamless integration during the development and testing phases.

By leveraging these components in conjunction with ESP32's processing capabilities and ESPNOW's efficient communication protocol, the transmitter unit effectively captures hand gestures via the MPU6050 sensor and transmits corresponding control signals to the car, enabling intuitive and responsive wireless control.

2.2. Components for Receiver Robot Car:

- **4WD Car Kit**: The foundation of the car, comprising a chassis equipped with four DC motors that drive its movement. The car kit provides the physical structure and mobility required for implementing the wireless hand gesture control system.

- **ESP32 Module**: The receiver ESP32 module functions as the central processing unit responsible for receiving data transmitted wirelessly from the transmitter ESP32. Upon receiving gesture commands, it processes the data and generates corresponding signals to control the motors, directing the car's movement according to the detected hand gestures.

- **L298N Motor Driver Module**: This motor driver module serves as the intermediary between the ESP32 module and the car's DC motors. It receives control signals from the ESP32 and regulates the power supplied to the motors, enabling precise control over their speed and direction of rotation.

- **7-12V Battery**: This power source provides the necessary electrical energy to power both the car and the motor driver module. It ensures uninterrupted operation of the entire system, allowing the car to execute movement commands effectively in response to the received gesture signals.

- **Double-Sided Tape and Jumper Wires**: These mounting and connection materials facilitate the assembly and integration of components within the car. Double-sided tape securely attaches components to the chassis, while jumper wires establish electrical connections between the ESP32 module, motor driver module, motors, and power supply, enabling seamless communication and power distribution throughout the system.

By combining these components, the receiver unit effectively interprets gesture commands received wirelessly from the transmitter unit and translates them into precise motor control signals, enabling the car to navigate and maneuver according to the user's hand gestures with accuracy and responsiveness.

Chapter 3

Software Requirement Specification (SRS)

- Arduino IDE: The Arduino Integrated Development Environment (IDE) serves as the primary software tool for writing, editing, and uploading code to the ESP32 modules. It provides an intuitive and user-friendly interface for programming microcontrollers, including ESP32, using the Arduino programming language.

- ESP-NOW Library: The ESP-NOW library is essential for establishing wireless communication between ESP32 modules in peer-to-peer mode. It enables efficient and low-latency data transmission between the transmitter and receiver ESP32 modules, facilitating the exchange of gesture data and control commands for the hand gesture-controlled car.

- MPU6050 Library: The MPU6050 library is utilized to interface with the MPU6050 sensor, which serves as the primary input device for capturing hand gestures. This library provides functions and utilities to communicate with the MPU6050 module, read sensor data, and interpret hand movements accurately, enabling precise gesture recognition for controlling the car.

- WiFi Library: The WiFi library for ESP32 is necessary for enabling Wi-Fi functionality on the ESP32 modules. While the hand gesture-controlled car primarily relies on ESP-NOW for communication between ESP32 modules, the WiFi library may be required for auxiliary functions or future enhancements, such as integrating Wi-Fi connectivity for remote monitoring or control via a smartphone application.

- By fulfilling these software requirements and utilizing the specified libraries within the Arduino IDE, developers can effectively program and deploy the software components necessary for implementing the wireless hand gesture-controlled car using ESP32, ensuring seamless communication, accurate gesture recognition, and robust control functionality.

Chapter 4

Circuit Description

4.1. Transmitter Circuit:

- ESP32 and MPU6050 Connection:
  - The MPU6050 sensor communicates with the ESP32 microcontroller via the I2C protocol.
  - Connect the VCC pin of the MPU6050 to the 3.3V output of the ESP32 module to provide power.
  - Connect the GND pin of the MPU6050 to the ground (GND) pin of the ESP32 module to complete the circuit.
  - SDA (Serial Data) pin of the MPU6050 is connected to GPIO 21 of the ESP32, facilitating bidirectional data transfer.
  - SCL (Serial Clock) pin of the MPU6050 is connected to GPIO 22 of the ESP32, enabling synchronized communication.

- Power Supply:
  - A 5V DC power supply is utilized to power both the ESP32 module and the MPU6050 sensor.
  - Connect the positive terminal of the 5V DC supply to the positive rail of the breadboard.
  - Connect the negative terminal of the 5V DC supply to the ground (GND) rail of the breadboard to establish a common ground.
  - Power the ESP32 module and the MPU6050 sensor by connecting their respective VCC pins to the positive rail of the breadboard.

4.2. Receiver Circuit:

- ESP32 and L298N Motor Driver Connection:
  - The ESP32 microcontroller controls the L298N motor driver module via GPIO pins for motor control.
  - Connect the GPIO pins of the ESP32 module to the corresponding input pins of the L298N motor driver module.
  - Each motor output of the L298N motor driver is connected to the terminals of the DC motors, enabling control over their speed and direction.

- Motor Driver and Motors:
    - Connect the terminals of the DC motors to the output terminals of the L298N motor driver module.
    - The L298N motor driver regulates the power supplied to the motors based on the control signals received from the ESP32, allowing precise control over motor speed and direction.
    - Power the L298N motor driver module using the 7-12V battery, providing sufficient voltage for motor operation.

- Power Supply:
    - The 7-12V battery serves as the primary power source for both the ESP32 module and the L298N motor driver module.
    - Connect the positive terminal of the battery to the Vin (Voltage In) input of the ESP32 module and the VCC input of the L298N motor driver.
    - Connect the negative terminal of the battery to the ground (GND) input of both the ESP32 module and the L298N motor driver, establishing a common ground reference.

By configuring the transmitter and receiver circuits as described, the wireless hand gesture-controlled car system can effectively capture hand gestures, transmit control commands, and drive the motors of the car, enabling intuitive and responsive control over its movements.

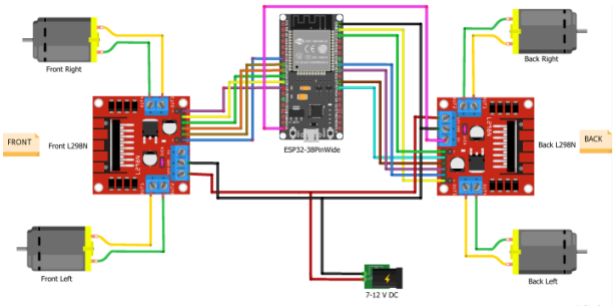

Fig: Car Receiver

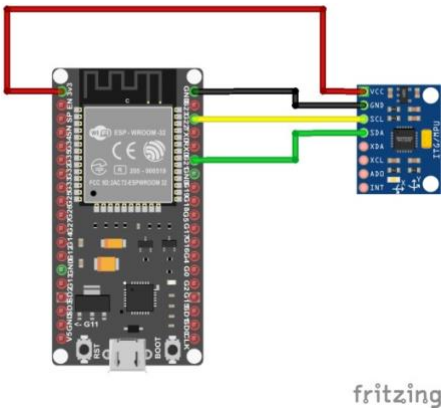

Fig: Car Transmitter

Chapter 5

Coding

5.1. Car Receiver Simple Movement:

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

#define FORWARD 1
#define BACKWARD 2
#define LEFT 3
#define RIGHT 4
#define FORWARD_LEFT 5
#define FORWARD_RIGHT 6
#define BACKWARD_LEFT 7
#define BACKWARD_RIGHT 8
#define TURN_LEFT 9
#define TURN_RIGHT 10
#define STOP 0

#define BACK_RIGHT_MOTOR 0
#define BACK_LEFT_MOTOR 1
#define FRONT_RIGHT_MOTOR 2
#define FRONT_LEFT_MOTOR 3

struct MOTOR_PINS
{            int
pinIN1;   int
pinIN2;
  int    pinEn;             int
pwmSpeedChannel;
};

std::vector<MOTOR_PINS> motorPins =
{
  {16, 17, 22, 4},  //BACK_RIGHT_MOTOR
  {18, 19, 23, 5},  //BACK_LEFT_MOTOR
```

```
  {26, 27, 14, 6},  //FRONT_RIGHT_MOTOR
  {33, 25, 32, 7},  //FRONT_LEFT_MOTOR
};

#define MAX_MOTOR_SPEED 200

const int PWMFreq = 1000; /* 1 KHz */ const
int PWMResolution = 8;

#define SIGNAL_TIMEOUT 1000  // This is signal timeout in milli
seconds. We will reset the data if no signal
unsigned long lastRecvTime = 0;

struct PacketData
{                byte
xAxisValue;   byte
yAxisValue;
  byte zAxisValue;
};
PacketData receiverData;

// callback function that will be executed when data is received void
OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
  if (len == 0)
  {
return;
  }
  memcpy(&receiverData,      incomingData,      sizeof(receiverData));
String inputData ;
  inputData = inputData + "values " + receiverData.xAxisValue + "   " +
receiverData.yAxisValue + "   " + receiverData.zAxisValue;
  Serial.println(inputData);

  if ( receiverData.xAxisValue < 75 && receiverData.yAxisValue < 75)
  {
    processCarMovement(FORWARD_LEFT);
  }
  else if ( receiverData.xAxisValue > 175 && receiverData.yAxisValue <
75)
  {
```

```
   processCarMovement(FORWARD_RIGHT);
  }
  else if ( receiverData.xAxisValue < 75 && receiverData.yAxisValue >
175)
  {
   processCarMovement(BACKWARD_LEFT);
  }
  else if ( receiverData.xAxisValue > 175 && receiverData.yAxisValue >
175)
  {
   processCarMovement(BACKWARD_RIGHT);
  }
  else if (receiverData.zAxisValue > 175)
  {
   processCarMovement(TURN_RIGHT);
  }
  else if (receiverData.zAxisValue < 75)
  {
   processCarMovement(TURN_LEFT);
  }
  else if (receiverData.yAxisValue < 75)
  {
   processCarMovement(FORWARD);
  }
  else if (receiverData.yAxisValue > 175)
  {
   processCarMovement(BACKWARD);
  }
  else if (receiverData.xAxisValue > 175)
  {
   processCarMovement(RIGHT);
  }
  else if (receiverData.xAxisValue < 75)
  {
   processCarMovement(LEFT);
  }
else
  {
   processCarMovement(STOP);
  }
```

```
  lastRecvTime = millis();
}

void processCarMovement(int inputValue)
{
  switch(inputValue)
  {
    case FORWARD:
      rotateMotor(FRONT_RIGHT_MOTOR,     MAX_MOTOR_SPEED);
rotateMotor(BACK_RIGHT_MOTOR,          MAX_MOTOR_SPEED);
rotateMotor(FRONT_LEFT_MOTOR,          MAX_MOTOR_SPEED);
rotateMotor(BACK_LEFT_MOTOR,           MAX_MOTOR_SPEED);
break;

    case  BACKWARD:          rotateMotor(FRONT_RIGHT_MOTOR, -
MAX_MOTOR_SPEED);       rotateMotor(BACK_RIGHT_MOTOR, -
MAX_MOTOR_SPEED);       rotateMotor(FRONT_LEFT_MOTOR, -
MAX_MOTOR_SPEED);         rotateMotor(BACK_LEFT_MOTOR, -
MAX_MOTOR_SPEED);        break;

    case LEFT:
      rotateMotor(FRONT_RIGHT_MOTOR,     MAX_MOTOR_SPEED);
rotateMotor(BACK_RIGHT_MOTOR,         -MAX_MOTOR_SPEED);
rotateMotor(FRONT_LEFT_MOTOR,         -MAX_MOTOR_SPEED);
rotateMotor(BACK_LEFT_MOTOR,           MAX_MOTOR_SPEED);
break;

    case RIGHT:
      rotateMotor(FRONT_RIGHT_MOTOR, -MAX_MOTOR_SPEED);
rotateMotor(BACK_RIGHT_MOTOR,          MAX_MOTOR_SPEED);
rotateMotor(FRONT_LEFT_MOTOR,          MAX_MOTOR_SPEED);
rotateMotor(BACK_LEFT_MOTOR,          -MAX_MOTOR_SPEED);
break;

    case FORWARD_LEFT:
      rotateMotor(FRONT_RIGHT_MOTOR, MAX_MOTOR_SPEED);
      rotateMotor(BACK_RIGHT_MOTOR,                    STOP);
rotateMotor(FRONT_LEFT_MOTOR, STOP);
      rotateMotor(BACK_LEFT_MOTOR,         MAX_MOTOR_SPEED);
break;
```

```
   case FORWARD_RIGHT:
     rotateMotor(FRONT_RIGHT_MOTOR,                STOP);
rotateMotor(BACK_RIGHT_MOTOR,          MAX_MOTOR_SPEED);
rotateMotor(FRONT_LEFT_MOTOR, MAX_MOTOR_SPEED);
     rotateMotor(BACK_LEFT_MOTOR,          STOP);
break;

   case BACKWARD_LEFT:
     rotateMotor(FRONT_RIGHT_MOTOR,                STOP);
rotateMotor(BACK_RIGHT_MOTOR,          -MAX_MOTOR_SPEED);
rotateMotor(FRONT_LEFT_MOTOR, -MAX_MOTOR_SPEED);
     rotateMotor(BACK_LEFT_MOTOR,          STOP);
break;

   case                                  BACKWARD_RIGHT:
rotateMotor(FRONT_RIGHT_MOTOR, -MAX_MOTOR_SPEED);
     rotateMotor(BACK_RIGHT_MOTOR,                STOP);
rotateMotor(FRONT_LEFT_MOTOR, STOP);
     rotateMotor(BACK_LEFT_MOTOR,       -MAX_MOTOR_SPEED);
break;

   case TURN_LEFT:
     rotateMotor(FRONT_RIGHT_MOTOR, MAX_MOTOR_SPEED);
rotateMotor(BACK_RIGHT_MOTOR,       MAX_MOTOR_SPEED);
rotateMotor(FRONT_LEFT_MOTOR,       -MAX_MOTOR_SPEED);
rotateMotor(BACK_LEFT_MOTOR,        -MAX_MOTOR_SPEED);
break;

   case TURN_RIGHT:
     rotateMotor(FRONT_RIGHT_MOTOR,    -MAX_MOTOR_SPEED);
rotateMotor(BACK_RIGHT_MOTOR,          -MAX_MOTOR_SPEED);
rotateMotor(FRONT_LEFT_MOTOR, MAX_MOTOR_SPEED);
     rotateMotor(BACK_LEFT_MOTOR,        MAX_MOTOR_SPEED);
break;

   case STOP:
     rotateMotor(FRONT_RIGHT_MOTOR,                STOP);
rotateMotor(BACK_RIGHT_MOTOR,                STOP);
rotateMotor(FRONT_LEFT_MOTOR, STOP);
```

```
    rotateMotor(BACK_LEFT_MOTOR,          STOP);
break;

   default:
    rotateMotor(FRONT_RIGHT_MOTOR,                STOP);
rotateMotor(BACK_RIGHT_MOTOR,                 STOP);
rotateMotor(FRONT_LEFT_MOTOR, STOP);
    rotateMotor(BACK_LEFT_MOTOR,          STOP);
break;
  }
}

void rotateMotor(int motorNumber, int motorSpeed)
{
 if (motorSpeed < 0)
 {
  digitalWrite(motorPins[motorNumber].pinIN1,           LOW);
digitalWrite(motorPins[motorNumber].pinIN2, HIGH);
 }
 else if (motorSpeed > 0)
 {
  digitalWrite(motorPins[motorNumber].pinIN1,           HIGH);
digitalWrite(motorPins[motorNumber].pinIN2, LOW);
 }
else
 {
  digitalWrite(motorPins[motorNumber].pinIN1,           LOW);
digitalWrite(motorPins[motorNumber].pinIN2, LOW);
 }

 ledcWrite(motorPins[motorNumber].pwmSpeedChannel,
abs(motorSpeed));
}

void setUpPinModes()
{
 for (int i = 0; i < motorPins.size(); i++)
 {
  pinMode(motorPins[i].pinIN1,                  OUTPUT);
pinMode(motorPins[i].pinIN2, OUTPUT);
  //Set up PWM for motor speed
```

```
    ledcSetup(motorPins[i].pwmSpeedChannel, PWMFreq, PWMResolution);
    ledcAttachPin(motorPins[i].pinEn, motorPins[i].pwmSpeedChannel);
rotateMotor(i, STOP);
  }
}

void setup()
{
  setUpPinModes();

  Serial.begin(115200);
  WiFi.mode(WIFI_STA);

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

  esp_now_register_recv_cb(OnDataRecv);
}

void loop()
{
  //Check Signal lost.  unsigned
long now = millis();
  if ( now - lastRecvTime > SIGNAL_TIMEOUT )
  {
    processCarMovement(STOP);
  }
}
```

5.2. Car Transmitter:

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

```
//These are needed for MPU
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
   #include "Wire.h"
#endif

// MPU control/status vars MPU6050
mpu;
bool dmpReady = false;  // set true if DMP init was successful uint8_t
devStatus;      // return status after each device operation (0 = success,
!0 = error)
uint8_t fifoBuffer[64]; // FIFO storage buffer
Quaternion q;              // [w, x, y, z]           quaternion container
VectorFloat gravity;    // [x, y, z]           gravity vector float ypr[3];
// [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// RECEIVER MAC Address
uint8_t receiverMacAddress[] = {0xAC,0x67,0xB2,0x36,0x7F,0x28};
//AC:67:B2:36:7F:28

struct PacketData
{                 byte
xAxisValue;   byte
yAxisValue;   byte
zAxisValue;
};
PacketData data;

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{
 //Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Message
sent" : "Message failed");
}

void setupMPU()
{
 // join I2C bus (I2Cdev library doesn't do this automatically)
 #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
```

```
    Wire.setClock(400000); // 400kHz I2C clock. Comment this line if
having compilation difficulties
 #elif I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
 #endif

 mpu.initialize();          devStatus    =
mpu.dmpInitialize();    // make sure it
worked (returns 0 if so)   if (devStatus
== 0)
 {
    // Calibration Time: generate offsets and calibrate our MPU6050
mpu.CalibrateAccel(6);                    mpu.CalibrateGyro(6);
mpu.setDMPEnabled(true);      dmpReady = true;
 }
}

void setup()
{
 Serial.begin(115200);
 WiFi.mode(WIFI_STA);
// Init ESP-NOW
 if (esp_now_init() != ESP_OK)
 {
  Serial.println("Error initializing ESP-NOW");
return;   }   else
 {
  Serial.println("Succes: Initialized ESP-NOW");
 }

 esp_now_register_send_cb(OnDataSent);

 //              Register              peer
esp_now_peer_info_t peerInfo;
 memcpy(peerInfo.peer_addr, receiverMacAddress, 6);
 peerInfo.channel = 0;
 peerInfo.encrypt = false;

 // Add peer
 if (esp_now_add_peer(&peerInfo) != ESP_OK)
```

```
  {
    Serial.println("Failed to add peer");
    return;
}   else
  {
    Serial.println("Succes: Added peer");
  }

  //This is to set up MPU6050 sensor   setupMPU();
}

void loop()
{
  // if programming failed, don't try to do anything   if
(!dmpReady) return;
  // read a packet from FIFO. Get the Latest packet   if
(mpu.dmpGetCurrentFIFOPacket(fifoBuffer))
  {
    mpu.dmpGetQuaternion(&q,                                    fifoBuffer);
mpu.dmpGetGravity(&gravity, &q);     mpu.dmpGetYawPitchRoll(ypr,
&q, &gravity);

    int  xAxisValue  =  constrain(ypr[2]  *  180/M_PI,  -90,  90);
int yAxisValue = constrain(ypr[1] * 180/M_PI, -90, 90);     int
zAxisValue  =  constrain(ypr[0]  *  180/M_PI,  -90,  90);
data.xAxisValue  =  map(xAxisValue,  -90,  90,  0,  254);
data.yAxisValue  =  map(yAxisValue,  -90,  90,  0,  254);
data.zAxisValue = map(zAxisValue, -90, 90, 0, 254);

    esp_err_t result = esp_now_send(receiverMacAddress, (uint8_t *)
&data, sizeof(data));

    String inputData  = inputData + "values " + xAxisValue + "  " +
yAxisValue  +  "    "  +  zAxisValue;         Serial.println(inputData);
delay(50);
  }
}
```

Chapter 6

Overview

The Wireless Hand Gesture Controlled Car project represents a fusion of cutting-edge technologies, uniting ESP32 microcontrollers, MPU6050 gyro modules, and L298N motor drivers to create a pioneering wireless control system for an automotive platform. This innovative system reimagines conventional car control mechanisms, offering users an intuitive and dynamic means of directing the car's movements through hand gestures. At the heart of the project lie ESP32 microcontrollers, serving as the nerve centers orchestrating communication and processing. Complementing these, MPU6050 gyro modules capture intricate hand movements with precision, translating them into actionable digital signals for transmission. Meanwhile, the L298N motor driver module facilitates meticulous regulation of power to the car's motors, ensuring seamless execution of gesture-driven commands. Leveraging the proprietary ESPNOW protocol, seamless wireless communication is established between transmitter and receiver units, enabling swift exchange of gesture data and control commands. Through the synergy of these components and technologies, the project aspires to deliver an immersive and engaging user experience, transcending traditional control paradigms to redefine the future of automotive interaction.

This project integrates ESP32 microcontrollers, MPU6050 gyro module, and L298N motor driver to create a wireless gesture-controlled car. The MPU6050 captures hand movements, which are transmitted to the car using the ESPNOW protocol, enabling intuitive and responsive control.

Chapter -7

Proposed Technology (Methodology)

The methodology for developing the Wireless Hand Gesture Controlled Car using ESP32 encompasses several key steps, each crucial to the successful realization of the project:

- System Architecture Design:
  - Begin by outlining the overall architecture of the system, delineating the roles and interactions of the transmitter and receiver units. Define the data flow, communication protocols, and interfaces between components to ensure seamless integration and operation.
- Transmitter Unit Development:
  - Develop the transmitter unit, focusing on integrating the ESP32 microcontroller and MPU6050 gyro module. Utilize the Arduino IDE to write and upload code to the ESP32, incorporating the MPU6050 library for interfacing with the gyro module. Implement gesture recognition algorithms to interpret hand movements and transmit corresponding commands using the ESP-NOW library for wireless communication.
- Receiver Unit Development:
  - Concurrently, design and implement the receiver unit, centered around the ESP32 microcontroller and L298N motor driver module. Write code to configure GPIO pins for motor control and establish communication with the transmitter unit using the ESP-NOW protocol. Integrate motor control logic to execute commands received from the transmitter, ensuring precise control over the car's movements.

- Integration and Testing:
  - Integrate the transmitter and receiver units, establishing wireless communication between them using the ESPNOW protocol. Conduct comprehensive testing to validate the functionality and performance of the system under various

scenarios, ensuring accurate gesture recognition, responsive control, and robust communication.

- Refinement and Optimization:
  - Refine the system design and codebase based on testing feedback, addressing any identified issues or optimizations. Fine-tune gesture recognition algorithms, motor control logic, and communication protocols to enhance overall system efficiency, reliability, and user experience.

- Documentation and Deployment:
  - Document the system architecture, hardware connections, and software implementation details comprehensively for reference and future development. Prepare the project for deployment, ensuring clear instructions for assembly, configuration, and operation of the wireless hand gesture-controlled car.

By following this proposed methodology, the development of the Wireless Hand Gesture Controlled Car using ESP32 can proceed systematically, enabling the realization of a robust and immersive automotive control system that leverages cutting-edge technology and intuitive user interaction.

Chapter 8

Definition and Performance of Gesture Control Robot

Definition:

- A gesture-controlled robot utilizing the ESP32 WiFi module is a robotic system that enables users to interact with and control a robot through hand movements or gestures, facilitated by wireless communication using the ESP32 module. This innovative system transcends traditional control interfaces, providing users with a natural and intuitive means of commanding the robot without the need for physical input devices or specialized training. By integrating gesture recognition technology with the ESP32 WiFi module, developers create a seamless interface between humans and robots, unlocking new possibilities for human-robot interaction across various domains.

Performance:

- **Responsiveness:** The system exhibits exceptional responsiveness, reacting almost instantaneously to hand gestures. This swift response is attributed to the efficient communication facilitated by the ESPNOW protocol, ensuring minimal latency between gesture input and motor control.
- **Accuracy:** The MPU6050 sensor provides high accuracy in gesture recognition, enabling precise interpretation of hand movements. This accuracy enhances the overall performance of the system, ensuring that the car responds accurately to the user's gestures.
- **Reliability:** The system operates reliably, maintaining consistent communication between the transmitter and receiver units. This reliability extends to motor control, ensuring that the car's movements are executed accurately and consistently in accordance with the detected gestures.
- **User Experience:** The system offers an intuitive and user-friendly experience, making it easy for users to control the car through hand gestures. The simplicity of the control mechanism enhances user interaction, resulting in a satisfying and engaging driving experience.
- **Adaptability:** The system is highly adaptable and can be easily modified to accommodate various control gestures. With simple code modifications, the system can recognize and respond to different hand gestures, providing flexibility in user interaction.

- **Scalability:** The system is scalable and can be expanded to incorporate additional sensors or modules for enhanced functionality. For example, integrating a camera module could enable advanced features such as object recognition or autonomous navigation, further enriching the capabilities of the gesture-controlled car.

By delivering exceptional performance across these key aspects, the Wireless Hand Gesture Controlled Car using ESP32 microcontrollers offers a compelling solution for intuitive and immersive vehicle control, demonstrating versatility, reliability, and adaptability in user interaction and functionality.

Chapter 9

Applications

Gesture-controlled robots utilizing the ESP32 WiFi module offer a versatile and intuitive interface for human-robot interaction, opening up a wide range of applications across various domains. Some notable applications include:

- **Healthcare Assistance:**
  Gesture-controlled robots can assist healthcare professionals in patient care tasks, such as remote monitoring, medication delivery, and assistance with daily activities for elderly or disabled patients.
  In rehabilitation settings, gesture-controlled robots can facilitate therapeutic exercises and physical rehabilitation routines, providing personalized guidance and feedback to patients.
- **Industrial Automation:**
  In manufacturing environments, gesture-controlled robotic arms can streamline assembly processes, material handling tasks, and quality control inspections.
  Workers can operate robots from a safe distance using intuitive hand gestures, reducing the need for manual intervention and improving workplace safety.
- **Education and Research:**
  Gesture-controlled robots serve as educational tools for teaching robotics concepts, programming fundamentals, and human-machine interaction principles in schools, universities, and research institutions.
  Researchers can use gesture-controlled robots to study human-robot collaboration, social interaction, and cognitive aspects of human-robot interaction in controlled laboratory settings.
- **Entertainment and Gaming:**
  Gesture-controlled robots offer immersive and interactive gaming experiences, where users can control virtual characters or manipulate objects in virtual environments using hand gestures.

  In amusement parks or entertainment venues, gesture-controlled robots can entertain guests with interactive performances, games, and attractions.

- **Smart Home Automation:**
  Gesture-controlled robots can serve as central hubs for smart home automation, allowing users to control various IoT devices, appliances, and home systems using intuitive hand gestures.
  Users can adjust lighting, temperature, security settings, and multimedia playback without the need for physical switches or remote controls.
- **Customer Service and Hospitality:**
  Gesture-controlled robots can assist customers in retail stores, airports, hotels, and restaurants by providing information, guidance, and personalized assistance.
  In hospitality settings, gesture-controlled robots can deliver room service, handle check-in/check-out procedures, and entertain guests with interactive experiences.
- **Environmental Monitoring and Exploration**:
  Gesture-controlled robots equipped with sensors can be deployed for environmental monitoring, exploration, and surveillance tasks in remote or hazardous environments.
  Researchers can use gesture-controlled robots to collect data, analyze environmental conditions, and monitor wildlife habitats without disturbing natural ecosystems.
- **Assistive Technology:**
  Gesture-controlled robots serve as assistive devices for individuals with disabilities, providing assistance with mobility, communication, and daily tasks.
  Users can control the robot using simple hand gestures or voice commands, enabling greater independence and autonomy in daily living activities.

Overall, the application of gesture-controlled robots using the ESP32 WiFi module is vast and diverse, spanning across healthcare, industrial automation, education, entertainment, smart home automation, customer service, environmental monitoring, and assistive technology. By leveraging intuitive hand gestures and wireless communication capabilities, gesture-controlled robots enhance human-robot interaction and empower users to interact with robots seamlessly in various contexts.

Chapter 10

Result

The Wireless Hand Gesture Controlled Car project has achieved its primary objective of demonstrating the feasibility and effectiveness of using hand gestures for wireless car control. Through meticulous design and implementation, the system showcases seamless integration of ESP32 microcontrollers, MPU6050 gyro modules, and L298N motor drivers to enable intuitive and responsive vehicle control.

Key Achievements:

- **Accurate Gesture Recognition:**
    1. The system excels in accurately recognizing hand gestures, precisely capturing tilt angles detected by the MPU6050 gyro module. This high level of accuracy ensures that subtle hand movements are translated into precise motor commands, facilitating smooth and fluid car movements.
- **Efficient Wireless Communication:**
    1. Utilizing the ESPNOW protocol, the system establishes efficient wireless communication between the transmitter and receiver units. This robust communication protocol ensures minimal latency, enabling near-instantaneous response to hand gestures and seamless coordination between the two ESP32 modules.
- **Responsive Motor Control:**
    1. The system demonstrates responsive motor control, translating detected hand gestures into precise motor commands to maneuver the car effectively in various directions. Whether it's forward, backward, left, or right movements, the car responds promptly and accurately to user input, enhancing the overall driving experience.
- **Versatile Directional Movement:**
    1. By interpreting hand gestures to control motor speed and direction, the system enables versatile directional movement of the car. Users can intuitively command the car to move forward, backward, turn left, turn right, or execute complex maneuvers with ease, showcasing the adaptability and agility of the gesture-controlled interface.
- **User-Friendly Operation:**

The system offers a user-friendly operation, allowing users to control the car effortlessly through intuitive hand gestures. This simplicity in control mechanism enhances user engagement and enjoyment, making the driving experience both accessible and enjoyable for users of all skill levels.

Chapter 11

Conclusion

The Wireless Hand Gesture Controlled Car project represents a remarkable feat in technology integration, leveraging ESP32 microcontrollers, MPU6050 gyro modules, and L298N motor drivers to pioneer a novel method of car control. By harnessing the capabilities of these components and employing the ESPNOW protocol for communication, the project has successfully realized an innovative and interactive means of controlling a car through intuitive hand movements.

Key Contributions:

- **Integration of Advanced Technologies:**
    1. The project seamlessly integrates cutting-edge technologies, including ESP32 microcontrollers, MPU6050 gyro modules, and L298N motor drivers, to create a sophisticated wireless control system for the car. This integration underscores the project's commitment to innovation and advancement in automotive control technology.
- **Efficient Wireless Communication:**
    1. Leveraging the ESPNOW protocol, the project establishes efficient and reliable wireless communication between the transmitter and receiver units. This protocol ensures swift data transmission, enabling seamless exchange of gesture commands and facilitating responsive control over the car's movements.
- **Innovative Control Mechanism:**
    1. By interpreting hand movements captured by the MPU6050 gyro module, the project introduces an innovative and interactive control mechanism for the car. Users can effortlessly command the car's movements through simple hand gestures, offering a unique and engaging driving experience.
- **User-Friendly Operation:**
    1. The project prioritizes user experience, providing a user-friendly and intuitive control interface for the car. Through intuitive hand movements, users can navigate the car with ease, enhancing user engagement and enjoyment while driving.

Chapter – 12

Future Scope

The Wireless Hand Gesture Controlled Car Using ESP32 is an advanced project that incorporates various technologies to create a remotely controlled car that responds to hand gestures. The key components of this project include an ESP32 CAM module for real-time video feedback, advanced gesture recognition using machine learning algorithms, obstacle detection using ultrasonic sensors, and a mobile app for additional control options.

- **ESP32 CAM Module Integration:** The ESP32 CAM module is a versatile component that combines an ESP32 microcontroller with a camera module. By integrating this module into the car, users can receive real-time video feedback of the car's surroundings. This allows for better control and navigation, especially in scenarios where direct line of sight is limited.

- **Advanced Gesture Recognition:** Traditional gesture recognition methods may not be sufficient for complex gestures or in noisy environments. By implementing machine learning algorithms, such as convolutional neural networks (CNNs), users can perform more intricate gestures that are accurately recognized by the system. Training the model involves collecting a dataset of hand gesture images and labeling them for different actions (e.g., forward, backward, left, right).

- **Obstacle Detection and Avoidance:** To enhance safety and navigation, ultrasonic sensors can be integrated into the car to detect obstacles in its path. These sensors emit ultrasonic waves and measure the time it takes for them to bounce back, calculating the distance to nearby objects. The ESP32 processes this information and adjusts the car's trajectory to avoid collisions.

- **Mobile App Control:** Developing a mobile app provides users with additional control options beyond hand gestures. The app can feature a user-friendly interface with buttons or sliders for controlling the car's movement, adjusting camera settings, and viewing the live video feed. It can also display real-time data such as sensor readings and battery level, providing users with comprehensive control and feedback.

Chapter – 13

References

- ESP32 Documentation:

  https://www.espressif.com/en/products/socs/esp32

- MPU6050 Documentation:

  https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/

- ESPNOW Documentation:

  https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/wifi/esp_now.html

- Arduino IDE:

  https://www.arduino.cc/en/software