REPORT

HAND GESTURE BASED CONTROLLING CAR

BATCH-02

GROUP MEMBERS:

1. Prajwal Borgave – 26
2. Kanchan Pandore – 36
3. Neha Redekar – 37
4. Swara Patil – 38

INTRODUCTION:

This project focuses on controlling an IoT-powered car using hand gestures, eliminating the need for traditional controllers. The system utilizes computer vision and AI to recognize hand movements and send real-time commands to the car via WebSockets.

Key Components:

1. Login System – Ensures secure access.

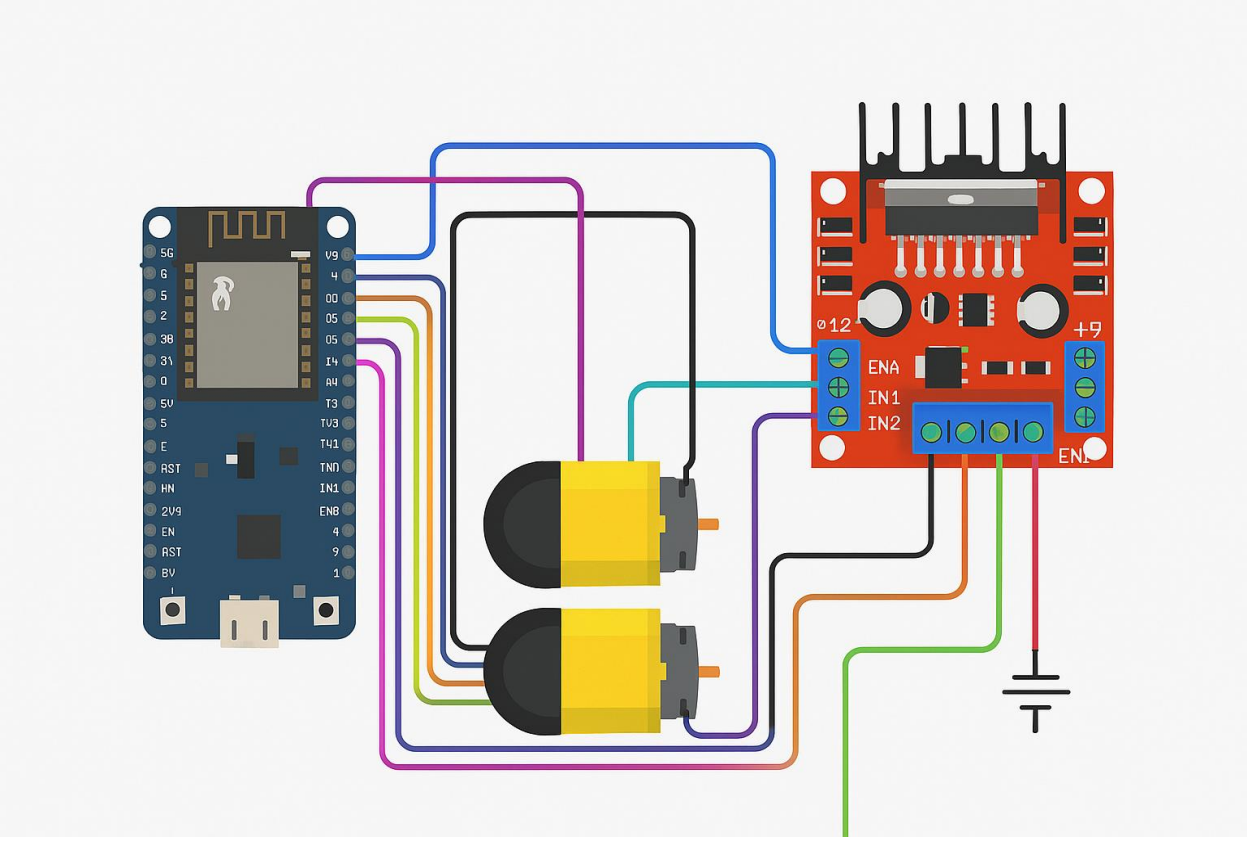
2. Connection Guide – Provides steps to connect the car to the user's device.

3. Gesture Recognition – Uses MediaPipe and OpenCV to detect hand gestures via a webcam.

4. WebSocket Communication – Sends detected gestures (Forward, Backward, Left, Right, Stop) to NodeMCU ESP8266, which controls the car's motors.

This project demonstrates the integration of AI-driven gesture detection with IoT-based control, making car navigation intuitive and hands-free.

CIRCUIT DIAGRAM:



CIRCUIT DIAGRAM EXPLANATION:

This is a circuit diagram showing the connection between an ESP32 microcontroller, an L298N motor driver module, and two DC motors. This setup is commonly used in IoT-based robotic car projects controlled via Wi-Fi or Bluetooth.

Components Used:

1. ESP32 Development Board
2. L298N Motor Driver Module
3. Two DC Geared Motors
4. Power Supply (Battery)
5. Connecting Wires

Wiring Connections:

ESP32 to L298N Motor Driver Module:

- GPIO Pin 2 (D2) → IN1 (L298N)
  Controls motor direction (Motor A)
- GPIO Pin 4 (D4) → IN2 (L298N)
  Controls motor direction (Motor A)
- GPIO Pin 5 (D5) → ENA (L298N)
  Enables/controls speed of Motor A using PWM
- GPIO Pin 18 (D18) → IN3 (L298N)
  Controls motor direction (Motor B)
- GPIO Pin 19 (D19) → IN4 (L298N)
  Controls motor direction (Motor B)
- GPIO Pin 21 (D21) → ENB (L298N)
  Enables/controls speed of Motor B using PWM

The ENA and ENB pins can also be connected to 5V for always ON mode, but PWM control is better for speed control.

Power and Ground:

- ESP32 GND → L298N GND
- Battery GND → L298N GND
- Battery V+ (e.g., 9V or 12V) → L298N 12V pin
- L298N 5V (optional) → Can power external devices if jumper is used

Motors to L298N Motor Driver:

- Motor A:
  - One wire → OUT1
  - Other wire → OUT2
- Motor B:
  - One wire → OUT3
  - Other wire → OUT4

Motor polarity doesn't matter initially, just adjust direction in code.

How It Works:

- The ESP32 sends signals to the L298N inputs (IN1-IN4) to control the direction of the motors.
- The ENA/ENB pins control speed using PWM signals.
- The L298N amplifies the signal and powers the motors using the external power supply.
- This allows you to make a robot move forward, backward, turn left/right, or stop.

Notes:

- Always use external power (battery) to power the motors via L298N.
- Do not power the motors from ESP32 — it cannot provide enough current.
- Keep grounds connected (ESP32, L298N, and battery) to avoid erratic behavior.

CODES:

CODE 1: Hand Gesture Recognition for IoT Car Control

```
import cv2

import mediapipe as mp
import asyncio
import websockets

# Replace with your ESP8266 WebSocket IP
ESP_IP = "ws://ESP ID"

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(min_detection_confidence=0.7, min_tracking_confidence=0.7)
mp_draw = mp.solutions.drawing_utils  # For drawing landmarks

# Open Webcam
cap = cv2.VideoCapture(0)

# Initialize Async Event Loop
loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)

async def send_command(command):
    """Send gesture command to ESP8266 via WebSocket."""
    try:
        async with websockets.connect(ESP_IP) as websocket:
            await websocket.send(command)
            print(f"Sent: {command}")
    except Exception as e:
        print(f"WebSocket Error: {e}")

def recognize_gesture(landmarks):
    """Detects if the palm is open (all fingers extended)"""
```

```
thumb_tip = landmarks[4]
index_tip = landmarks[8]
middle_tip = landmarks[12]
ring_tip = landmarks[16]
pinky_tip = landmarks[20]

index_pip = landmarks[6]
middle_pip = landmarks[10]
ring_pip = landmarks[14]
pinky_pip = landmarks[18]



thumb_base = landmarks[2]

# Check if all fingers are straight (tips above their PIP joints)
if (
    index_tip.y < index_pip.y and
    middle_tip.y < middle_pip.y and
    ring_tip.y < ring_pip.y and
    pinky_tip.y < pinky_pip.y and
    abs(thumb_tip.x - thumb_base.x) > 0.1  # Thumb spread out
):
    return "S"

elif thumb_tip.y < middle_tip.y < pinky_tip.y:
    return "F"

elif thumb_tip.y > middle_tip.y > pinky_tip.y:
    return "B"

elif thumb_tip.x > middle_tip.x > pinky_tip.x:
    return "L"

elif thumb_tip.x > middle_tip.x > pinky_tip.x:
```

```
        return "R"
return None  # No recognized gesture

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Convert frame to RGB (for MediaPipe)
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(rgb_frame)

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_draw.draw_landmarks(frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)

            landmarks = hand_landmarks.landmark
            gesture = recognize_gesture(landmarks)

            if gesture:
                loop.run_until_complete(send_command(gesture))  # ✅ Fix: Prevent Freezing
                cv2.putText(frame, f"Gesture: {gesture}", (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    cv2.imshow("Gesture Control", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Code Explanation:

1. Libraries Used:

- cv2 (OpenCV): Access webcam and process video frames.
- mediapipe: Detect hand landmarks.
- asyncio and websockets: Send commands asynchronously over a WebSocket connection.

2. Setup ESP8266 WebSocket Connection

python

CopyEdit

ESP_IP = "ws://ESP ID"

Replace this with the WebSocket IP of your ESP8266 (e.g., ws://192.168.1.100).

3. MediaPipe Hand Tracking Initialization

python

CopyEdit

mp_hands.Hands(min_detection_confidence=0.7, min_tracking_confidence=0.7)

- Detects and tracks hand landmarks with confidence threshold of 0.7.
- mp_draw: Utility to draw landmarks for visualization.

4. Open Webcam and Start Async Event Loop

python

CopyEdit

cap = cv2.VideoCapture(0)

loop = asyncio.new_event_loop()

- Opens the default webcam.
- Prepares an asynchronous event loop for sending WebSocket data without freezing the main thread.

5. Async Function to Send Command

python

CopyEdit

async def send_command(command):

...

- Connects to ESP8266 via WebSocket.
- Sends the gesture command string (e.g., "F" for forward).

6. Gesture Recognition Logic

python

def recognize_gesture(landmarks):

- Receives landmarks of detected hand.
- Checks conditions to recognize:
    - "S" – **Stop** (all fingers extended)
    - "F" – **Forward**
    - "B" – **Backward**
    - "L" – **Left**
    - "R" – **Right**

These are based on the position of specific hand landmarks like finger tips and joints.

7. Main Loop: Processing Webcam Feed

python

while cap.isOpened():

- Reads frames continuously.
- Converts frame to RGB for MediaPipe.
- If a hand is detected, it:
    - Draws landmarks
    - Passes landmarks to recognize_gesture()
    - Sends command to ESP8266
    - Displays the gesture name on-screen

8. Exit Mechanism

python

if cv2.waitKey(1) & 0xFF == ord('q'):

Press **'q'** to exit the program.

CODE 2: ESP8266 WebSocket-Based Gesture-Controlled Car

```
#include <ESP8266WiFi.h>
#include <WebSocketsServer.h>

// Wi-Fi Credentials
const char* ssid = "IoT_Car";  // Change to your Wi-Fi SSID
const char* password = "12345678";  // Change to your Wi-Fi Password

// Motor Pins
#define IN1 D1
#define IN2 D2
#define IN3 D3
#define IN4 D4

WebSocketsServer webSocket(81); // WebSocket server on port 81

void controlCar(String command) {
    if (command == "F") { // Forward
        digitalWrite(IN1, HIGH);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, HIGH);
        digitalWrite(IN4, LOW);
    } else if (command == "B") { // Backward
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, HIGH);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, HIGH);
    } else if (command == "L") { // Left
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, HIGH);
        digitalWrite(IN3, HIGH);
        digitalWrite(IN4, LOW);
    } else if (command == "R") { // Right
        digitalWrite(IN1, HIGH);
```

```
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, HIGH);
    } else { // Stop
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, LOW);
    }
}

// WebSocket Event Handler
void webSocketEvent(uint8_t num, WStype_t type, uint8_t *payload, size_t length) {
    if (type == WStype_TEXT) {
        String command = String((char *)payload);
        Serial.println("Command Received: " + command);
        controlCar(command);
    }
}

void setup() {
    Serial.begin(115200);

    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);

    // Connect to Wi-Fi
    WiFi.softAP(ssid, password);
    Serial.println("WiFi Started!");
    Serial.print("IP Address: ");
    Serial.println(WiFi.softAPIP());
```

```
  // Start WebSocket Server
  webSocket.begin();
  webSocket.onEvent(webSocketEvent);
}

void loop() {
  webSocket.loop(); // Keep WebSocket running
}
```

Code Explanation:

1. Libraries Used

cpp
CopyEdit

```
#include <ESP8266WiFi.h>
#include <WebSocketsServer.h>
```

- ESP8266WiFi: Manages Wi-Fi operations.
- WebSocketsServer: Sets up a WebSocket server to receive real-time commands from a client (like your PC with the hand gesture script).

2. Wi-Fi Setup

cpp
CopyEdit

```
const char* ssid = "IoT_Car";
const char* password = "12345678";
```

- Sets up the ESP8266 as a **Wi-Fi access point** with the SSID **"IoT_Car"** and password **"12345678"**.
- Your gesture control Python script should connect to this AP.

3. Motor Pin Definitions

cpp
CopyEdit

```
#define IN1 D1
#define IN2 D2
#define IN3 D3
#define IN4 D4
```

- These pins control the motor driver (e.g., L298N or L293D).

- IN1/IN2 control one motor, IN3/IN4 control the other.

4. WebSocket Setup

cpp

CopyEdit

WebSocketsServer webSocket(81);

- Starts a WebSocket server on port 81.
- This port is accessed from your Python script.

5. Car Control Logic

cpp

CopyEdit

void controlCar(String command)

This function decides motor actions based on received gesture commands:

- "F" – **Forward**: Both motors spin forward.
- "B" – **Backward**: Both motors spin backward.
- "L" – **Left**: One motor forward, the other backward.
- "R" – **Right**: Opposite motor movement to turn.
- Default (e.g., "S" or any invalid input) – **Stop**: All pins LOW.

6. Handling Incoming WebSocket Data

cpp

CopyEdit

void webSocketEvent(...)

- Triggered when a new message is received via WebSocket.
- Converts the payload to a string and passes it to controlCar().

7. Setup Function

cpp

CopyEdit

```
void setup() {
   ...
}
```

- Initializes serial communication for debugging.
- Sets motor pins as output.
- Starts the Wi-Fi access point.
- Starts the WebSocket server and registers the event handler.

8. Loop Function

cpp

CopyEdit

```
void loop() {
   webSocket.loop();
}
```

- Keeps checking and responding to WebSocket events.

How It Works Together:

1. ESP8266 starts as a Wi-Fi hotspot.
2. The Python gesture control script connects via WebSocket.
3. Detected gestures are sent as single characters (e.g., "F", "L").
4. ESP8266 receives these and adjusts motor directions.

GUI:

IoT Car Login:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>IoT Car Login</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h2>IoT Car Login</h2>
  <div class="login-box">
    <form>
      <label for="username">Username:</label><br>
      <input type="text" id="username" name="username" placeholder="admin"><br><br>

      <label for="password">Password:</label><br>
      <input type="password" id="password" name="password"><br><br>

      <button type="submit">Login</button>
    </form>
  </div>
</body>
</html>
```

Connection Guide:

connect.html

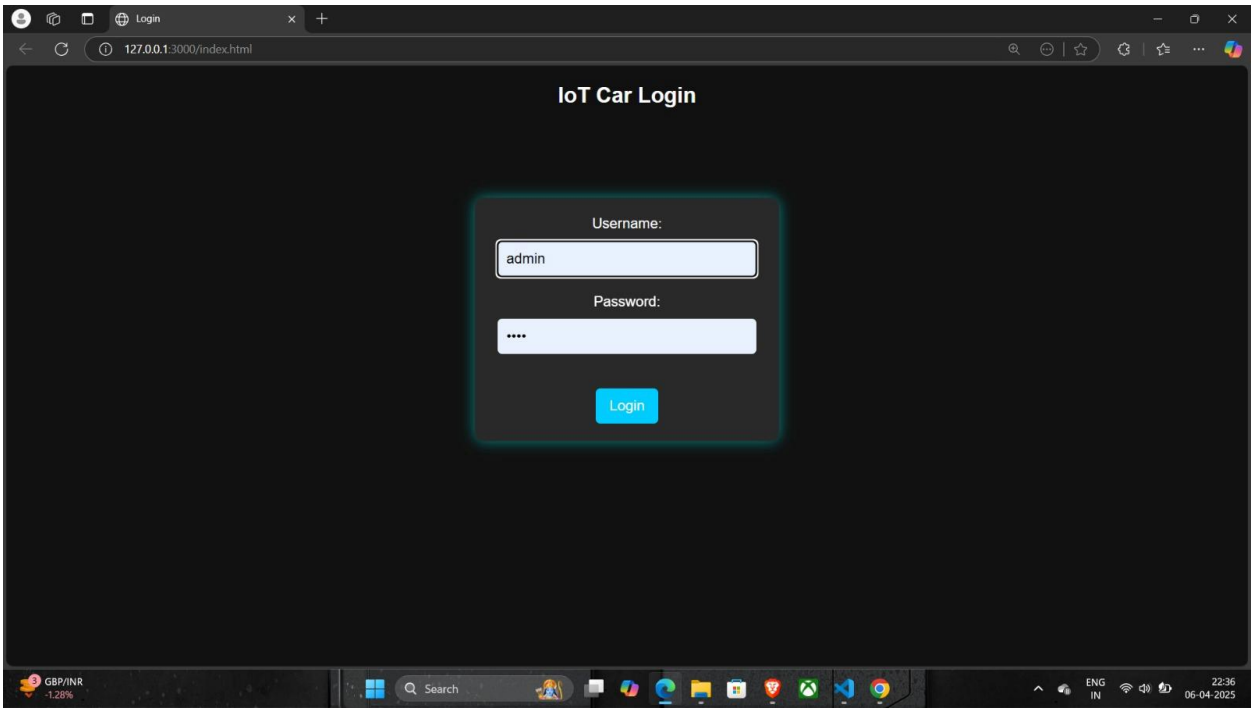
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Connect to IoT Car</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="guide-box">
    <h2>Connection Guide</h2>
    <p>Follow these steps to connect to the car:</p>
    <ol>
      <li>Turn on your NodeMCU.</li>
      <li>Connect your laptop to the NodeMCU hotspot.</li>
      <li>Wifi Name = " IoT_Car "</li>
      <li>Wifi password = " 12345678 "</li>
      <li>Ensure your laptop and NodeMCU are on the same network.</li>
      <li>After this Click the below button.</li>
    </ol>
    <div class="buttons">
      <button onclick="location.href='gesture.html'">Start Gesture Control</button>
      <button onclick="location.href='index.html'">Logout</button>
    </div>
  </div>
</body>
</html>
```

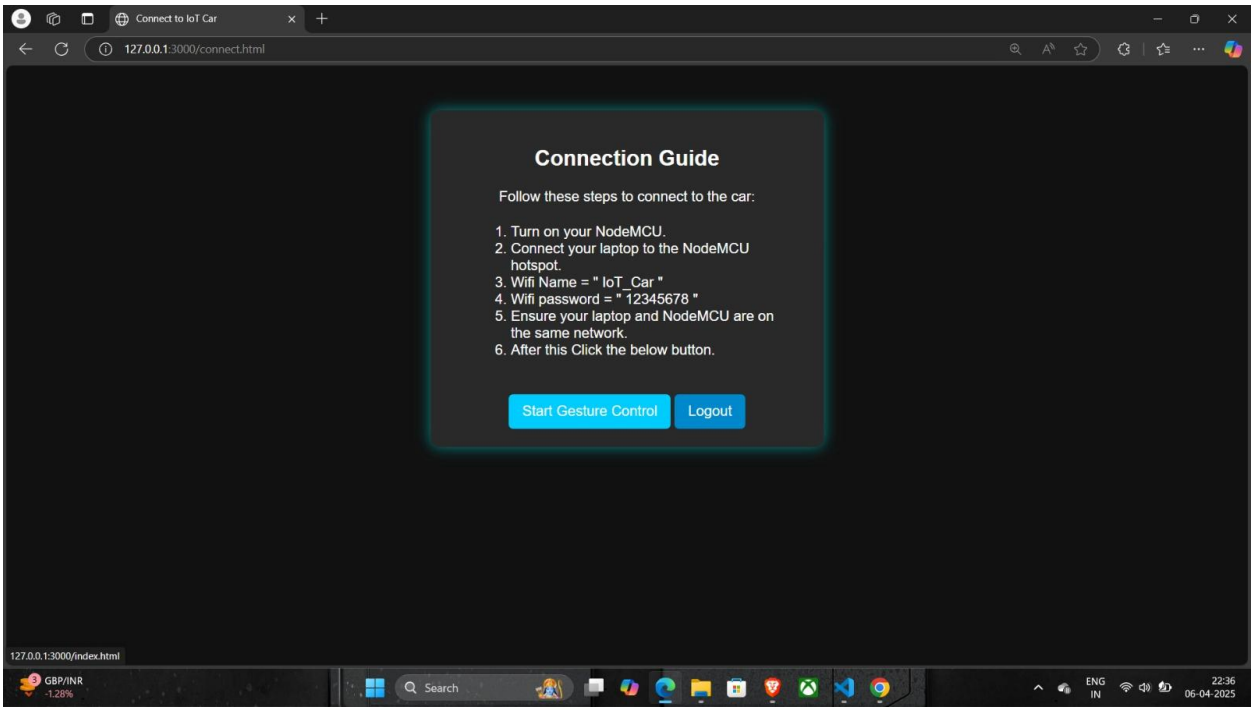
style.css

```
body {
  background-color: #0a0a0a;
  font-family: Arial, sans-serif;
  text-align: center;
  color: #fff;
  margin-top: 50px;
}

h2 {
  color: #fff;
}

/* Login Box Styles */
.login-box {
  background-color: #1c1c1c;
  padding: 30px 40px;
  display: inline-block;
  border-radius: 10px;
  box-shadow: 0 0 15px #00f5c966;
}

input[type="text"],
input[type="password"] {
  width: 250px;
  padding: 10px;
  margin-top: 5px;
  border: 2px solid #ccc;
  border-radius: 5px;
  outline: none;
}

/* Guide Box Styles */
.guide-box {
```

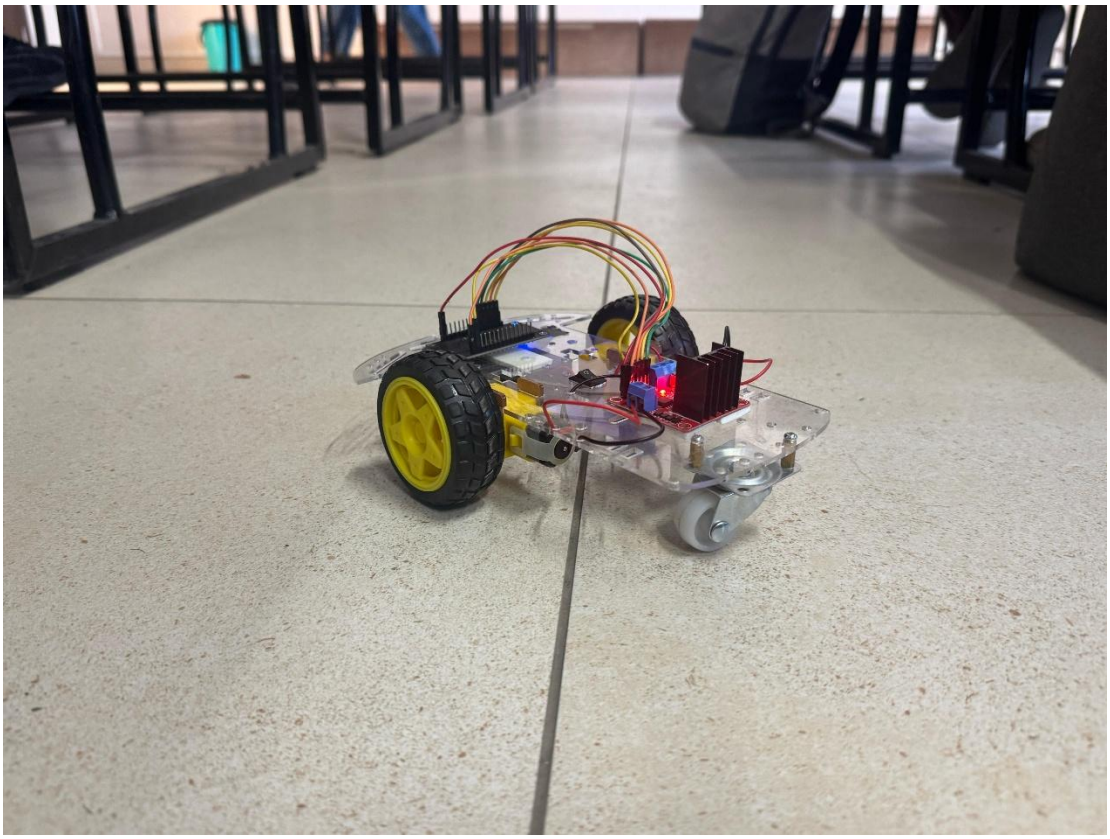
```
  background-color: #1c1c1c;
  padding: 30px 40px;
  width: fit-content;
  margin: auto;
  border-radius: 10px;
  box-shadow: 0 0 15px #00f5c966;
  color: #fff;
  text-align: left;
  margin-top: 60px;
}

.guide-box h2 {
  text-align: center;
  margin-bottom: 15px;
}

.guide-box p {
  font-size: 16px;
}

ol {
  margin-left: 20px;
  font-size: 15px;
}

/* Shared Button Styles */
button,
.buttons button {
  background-color: #00ccff;
  color: white;
  padding: 10px 25px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
```

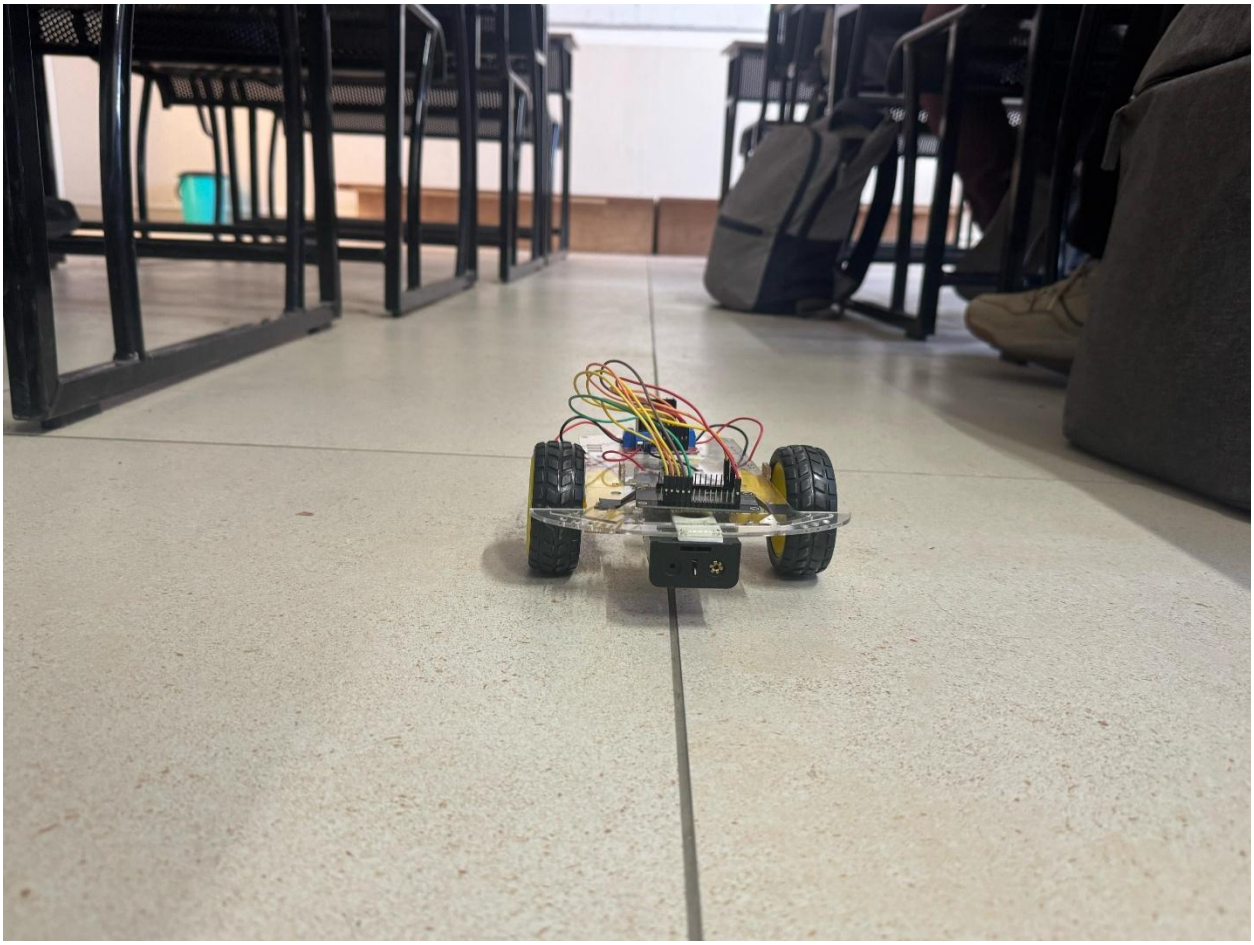
```
  font-weight: bold;
}

button:hover,
.buttons button:hover {
  background-color: #0099cc;
}

/* Buttons Layout for Guide */
.buttons {
  display: flex;
  justify-content: center;
  gap: 15px;
  margin-top: 20px;
}
```




Connect to IoT Car
127.0.0.1:3000/connect.html
Connection Guide
Follow these steps to connect to the car:
1. Turn on your NodeMCU.
2. Connect your laptop to the NodeMCU hotspot.
3. Wifi Name = " IoT_Car "
4. Wifi password = " 12345678 "
5. Ensure your laptop and NodeMCU are on the same network.
6. After this Click the below button.
Start Gesture Control
Logout
127.0.0.1:3000/index.html
GBP/INR
-1.28%
Search
ENG
IN
22:36
06-04-2025

WORKING CONDITION:





CONCLUSION:

The Hand Gesture Controlled Car project successfully integrates AI-based gesture recognition with IoT technology to enable intuitive and hands-free vehicle control. By utilizing MediaPipe and OpenCV for hand tracking and WebSockets for real-time communication, the system effectively translates hand gestures into movement commands for the car.

This project demonstrates the potential of human-computer interaction (HCI) in IoT applications, eliminating the need for traditional remote controllers. It provides a seamless and interactive experience, making it useful in various fields such as assistive technology, robotics, and automation.

The project can be further improved by enhancing gesture accuracy, integrating voice commands, and expanding remote access capabilities. These advancements can make the system more robust, versatile, and applicable to real-world scenarios like smart vehicle navigation and assistive mobility solutions.

REFERENCE:

The following references have been utilized in the development of this project, providing insights into key technologies such as hand gesture recognition, real-time communication using WebSockets, asynchronous programming, and IoT-based vehicle control:

1. MediaPipe Hands: Google's Hand Tracking API. This resource provides an in-depth understanding of hand landmark detection and tracking, which is fundamental for gesture-based control. Available at: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker.
2. WebSockets in Python: This documentation explains the implementation of real-time communication in Python, which is essential for transmitting commands between the gesture recognition system and the IoT-based vehicle. Available at: https://websockets.readthedocs.io/en/stable/.
3. AsyncIO in Python: This resource details asynchronous programming techniques in Python, which facilitate smooth and efficient handling of multiple tasks, including real-time data transmission. Available at: https://docs.python.org/3/library/asyncio.html.
4. Hand Gesture Controlled Car Using Bluetooth Modules and Accelerometer Sensor – *IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)*. This research paper discusses the use of Bluetooth and accelerometer sensors for gesture-controlled vehicle navigation, providing relevant insights into alternative methods of gesture-based control.
5. Hand Gesture Controlled Vehicle – *Academia.edu*. This publication explores various implementations of hand gesture-controlled vehicles, offering valuable theoretical and practical knowledge that has contributed to the design and functionality of this project.

.