Bahçeşehir University

Faculty of Engineering and Natural Sciences

Department of Artificial Intelligence

ESP32-CAM AI Follow-Me Robot

with Wireless Control, Obstacle Avoidance, and Web UI

Project Members:

Abdullah Hani Abdellatif Al-Shobaki	2284612
Mohamed Aiman Mohamed Alkozendar	2283149
Mohammed Ahmed Mohammed Al-labani	2105990

Course: CMP3010 – Embedded Systems Programming

Supervisor: Prof. Mahmut Ağan

Submission Date: May 23, 2025

Contents

Abstract

This project presents the development of an AI-powered autonomous mobile robot that leverages computer vision, embedded control, and web technologies to track and follow a human hand in real time. Built using an ESP32-CAM module and an Arduino Mega, the robot integrates real-time video streaming, MediaPipe-based hand tracking, and serial command transmission for dynamic motor control. An ultrasonic sensor enables real-time obstacle avoidance, overriding visual tracking when needed for safety. Additional behaviors such as random personality mode and a dance routine enhance interactivity. A browser-based Gradio interface allows manual control and live video monitoring. The system demonstrates a complete, modular, and scalable embedded AI application suitable for both educational and practical deployments.

1 Problem Statement and Objectives

Problem Statement:

Affordable robotics platforms often lack advanced perception and control features found in high-end systems. This project aims to bridge that gap by designing an embedded robot that can visually track a user's hand, avoid obstacles, and offer wireless control—using only accessible, low-cost components.

Project Objectives:

- Build a robot that can follow a human hand using real-time computer vision
- Enable autonomous obstacle detection and avoidance
- Support wireless manual control and live camera monitoring via a web interface
- Implement multiple robot behavior modes (follow, idle/random, Wi-Fi manual)
- Develop a clean, modular system using embedded and AI technologies

2 Background Information

In recent years, the fusion of embedded systems and artificial intelligence has led to the development of intelligent robotic platforms capable of interacting with humans in real time. Commercial systems like the MIT Gesture-Controlled Robot Arm and Google's AI-based smart assistants demonstrate the growing relevance of computer vision and real-time control. However, many of these solutions are either costly, cloud-dependent, or too complex for beginners.

Our project leverages the ESP32-CAM for wireless video streaming and MediaPipe for lightweight, accurate hand tracking — both of which enable low-cost, locally-processed interaction. MediaPipe, developed by Google Research [1], is particularly suitable for real-time gesture detection without the need for high-end GPUs or cloud infrastructure. Combined with the open-source flexibility of the Arduino platform, this architecture supports autonomous behavior such as obstacle avoidance, idle mode switching, and live web-based control.

This approach democratizes AI-powered robotics by enabling functionality typically found in much more expensive platforms. It supports hands-on learning and experimentation, making it highly suitable for STEM education, prototyping, and interactive demonstrations. By providing responsive, modular, and wireless robot behavior using accessible components, the system serves as an affordable foundation for future innovation in embedded AI applications.

3 Introduction

The convergence of embedded systems and artificial intelligence has enabled the development of responsive, intelligent robotic platforms capable of interacting with their environment in real time. This project explores this intersection by building a smart mobile robot that autonomously follows a user based on hand tracking, while maintaining safety through obstacle detection and offering manual control via a web interface.

At the heart of the system is the ESP32-CAM module, which streams live video to a Python-based control application running on a laptop. The application uses OpenCV and MediaPipe to detect the user's hand and infer direction (forward, left, right, or stop). Com-

mands are sent via HTTP to the ESP32-CAM, which relays them via serial communication to the Arduino Mega. The Arduino, in turn, drives the motors and controls additional peripherals such as RGB LEDs and ultrasonic sensors.

The robot prioritizes safety through continuous distance sensing, performing automatic avoidance maneuvers if an object is detected within a 20 cm range. When hand tracking is lost for more than 30 seconds, the robot enters a random personality mode, executing arbitrary movements until visual contact is re-established. A separate dance mode is available via the web interface, adding a playful element to the system.

This project demonstrates a fully functional, interactive, AI-assisted robot with a modular design and rich functionality. It combines embedded hardware, real-time computer vision, network communication, and human–robot interaction to build a robust platform applicable to educational, experimental, and entertainment domains.

4 Methodology

4.1 System Architecture

The ESP32-CAM AI Follow-Me Robot is architected around three distinct but interconnected modules: the ESP32-CAM for vision and communication, the Arduino Mega for actuation and sensing, and the Python-based laptop controller for AI-driven command logic. These modules communicate over Wi-Fi and serial connections in real time.

- **ESP32-CAM:** Streams MJPEG video over Wi-Fi and hosts an HTTP server to receive movement commands. It forwards these commands via serial to the Arduino Mega.
- **Arduino Mega:** Executes motor control using the L298N driver, interprets ultrasonic sensor data for obstacle avoidance, and controls RGB LED indicators.
- **Laptop (Python):** Processes the video stream with MediaPipe and OpenCV to detect hand gestures. Sends direction commands (F, L, R, S) via HTTP to the ESP32-CAM. Hosts a Gradio web UI for manual override and demo modes.

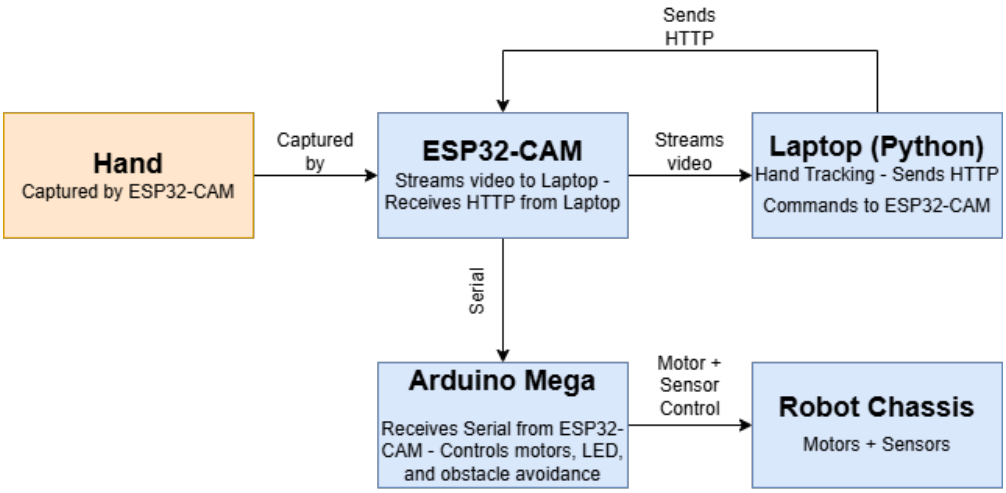

Figure 1: System Architecture and Data Flow of the ESP32-CAM AI Follow-Me Robot

4.2 Hardware Components

The physical system integrates multiple components for movement, sensing, feedback, and communication.

- **ESP32-CAM:** A compact Wi-Fi camera module used for video streaming and command reception [2].
- **Arduino Mega:** Central controller handling all motor logic, LED status, and sensor inputs.
- **L298N Motor Driver:** Dual H-bridge motor controller interfacing between the Arduino and 4 DC motors.
- **4x DC Motors:** Enable directional movement (forward, backward, turning).
- **Ultrasonic Sensor (HC-SR04):** Detects obstacles within 20 cm to trigger avoidance maneuvers [3].
- **RGB LED:** Visual status indicator — green for tracking, red for idle.

- **Battery Packs:** 12V pack for motors, 6V pack for Arduino Mega, and USB power bank for logic components.
- **Switches:** Used to separately control motor and Arduino power circuits.
- **Breadboard:** Used for modular wiring of sensor and LED circuits.

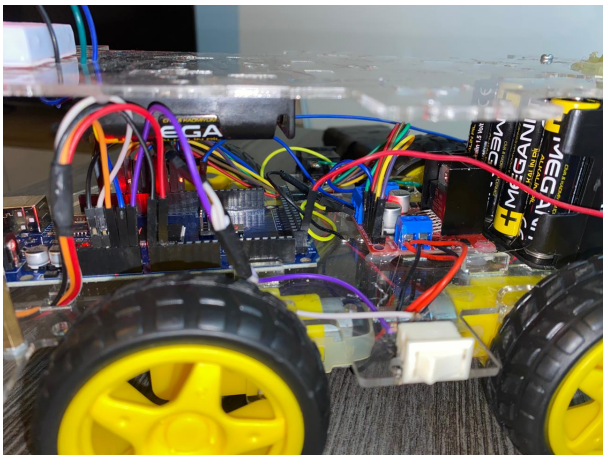

Figure 2: *
Left: Full side view of the robot showing wheel chassis, wiring, and power components

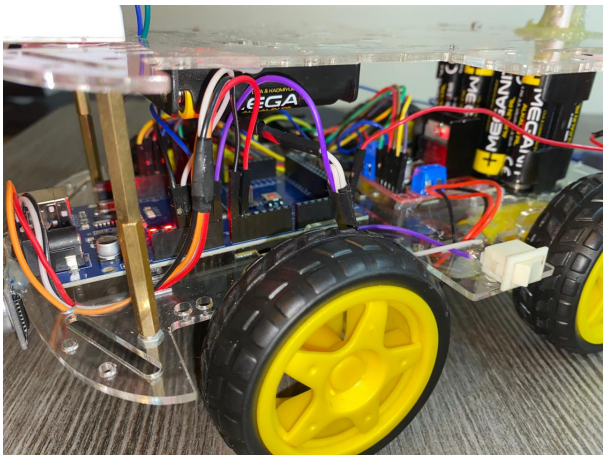

Figure 3: *
Right: Close-up of the Arduino Mega and L298N driver mounted on the lower platform

Figure 4: Hardware configuration showing structural layout and core electronic modules

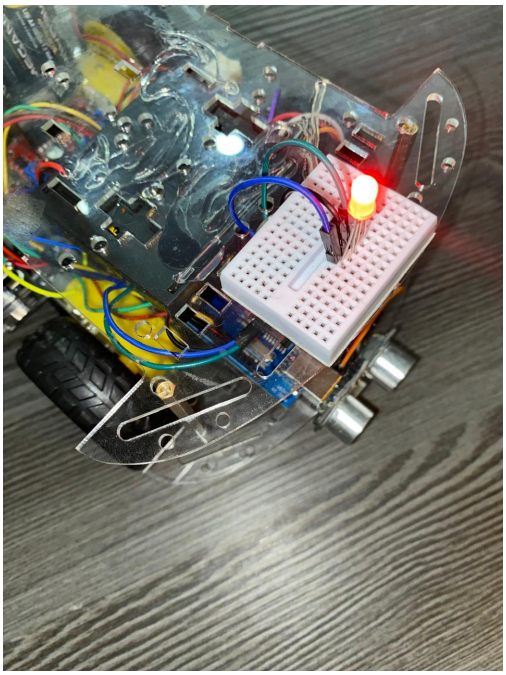

Figure 5: Breadboard wiring: LED and logic power connections

4.3 Electrical Schematic

Figure 6 shows the complete electrical schematic for the ESP32-CAM AI Follow-Me Robot. It includes all the wiring and component-level connections such as motor driver, RGB LED, HC-SR04 ultrasonic sensor, and serial communication lines between the ESP32-CAM and Arduino Mega.

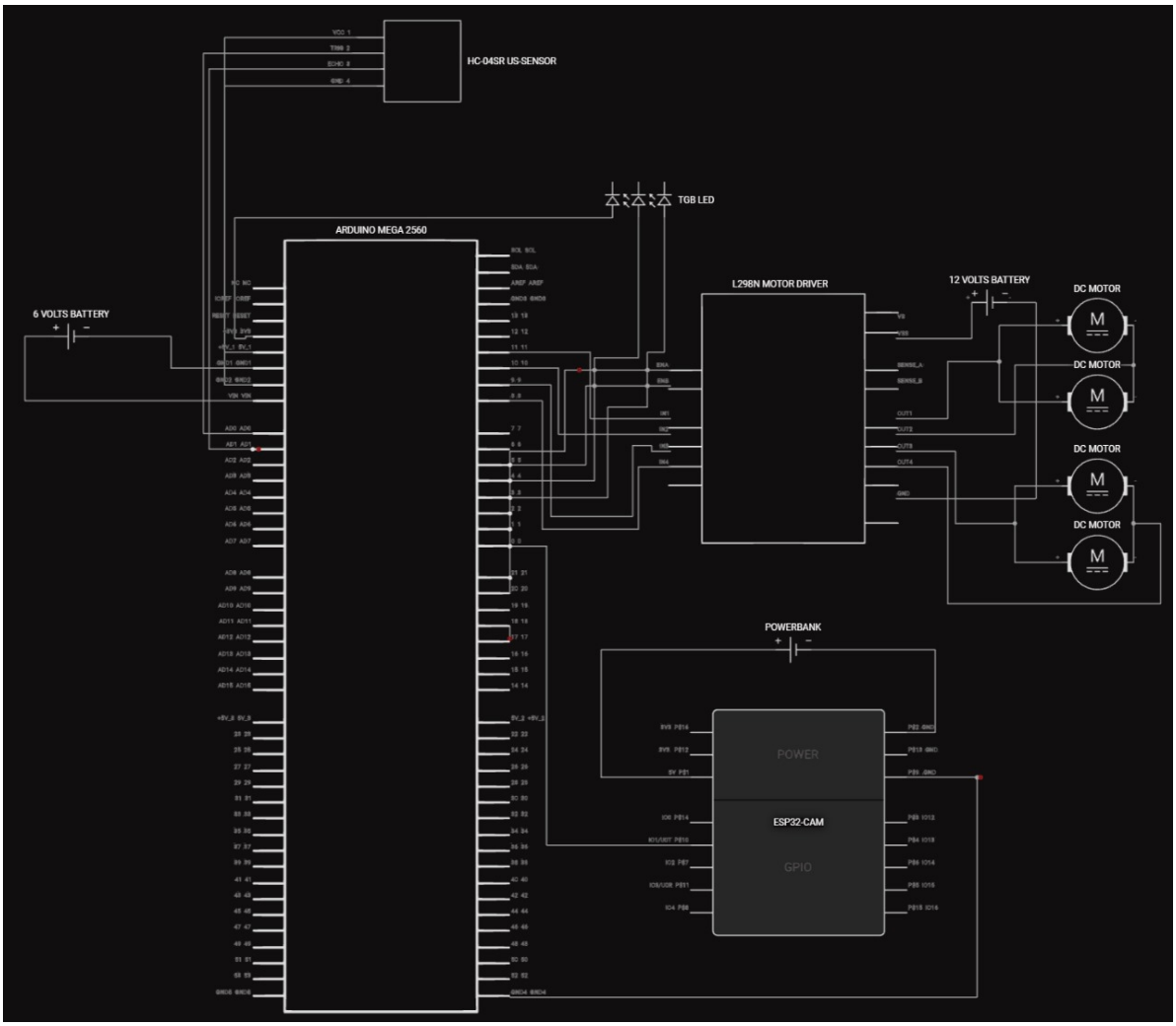

Figure 6: Full electrical schematic of the robot system, including power supplies, control logic, and peripheral wiring

4.4 Software Components

The software side of the system is split between embedded firmware and AI-driven control logic running on the laptop.

- **Arduino IDE:** Used to program ESP32-CAM and Arduino Mega with control logic

and serial protocols.

- **PyCharm:** Development environment for hand tracking and web interface scripts.
- **Python Libraries:**
  - **OpenCV:** Processes the live video stream for region of interest [4].
  - **MediaPipe:** Detects and tracks hand landmarks in real time [1].
  - **Requests:** Sends HTTP commands to ESP32-CAM.
  - **Gradio:** Creates a local web UI for control and monitoring [5].
- **ESP32-CAM Firmware:** Hosts HTTP endpoints for movement commands and live video (port 80/81).
- **Python Scripts:**
  - `hand_tracking.py` — Processes the video and sends commands.
  - `robot_ui.py` — Gradio interface with live preview and control buttons.

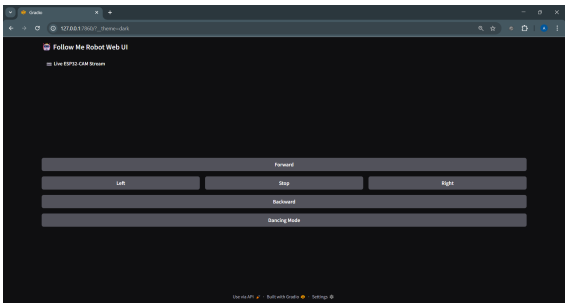

Figure 7: *
Left: Gradio web interface with ESP32-CAM live stream and control buttons

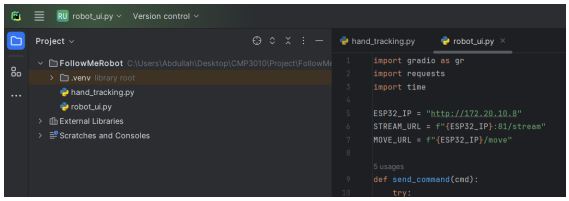

Figure 8: *
Right: Python project in PyCharm showing `robot_ui.py` control script

Figure 9: Control interface and Python backend used to operate the follow-me robot

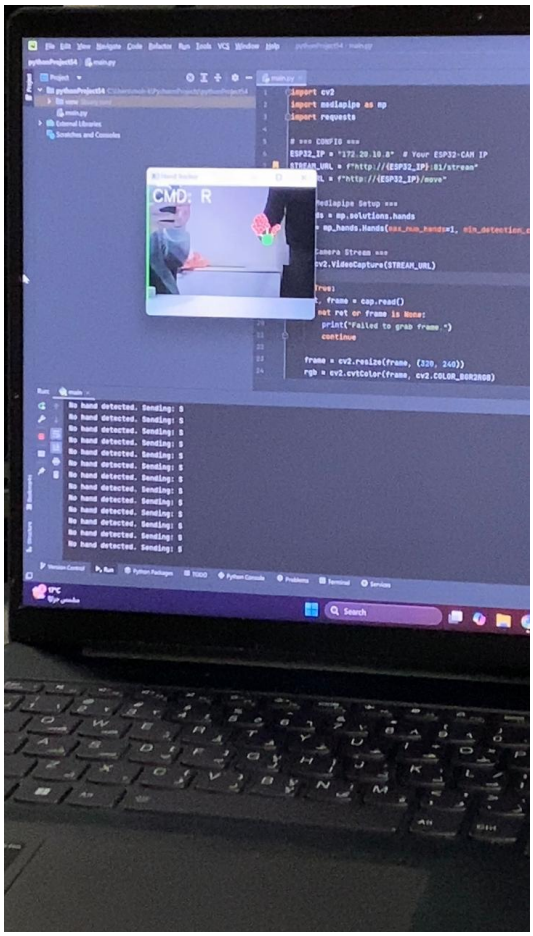

Figure 10: MediaPipe hand tracking visualized on ESP32-CAM stream

4.5 Key Functionalities

The robot supports multiple operational modes and features:

- **Hand-Tracking Follow Mode:** Tracks user hand and maps its position to directional movement (F, L, R, S).
- **Obstacle Avoidance:** Robot performs a right turn if object detected at distance < 20 cm.

- **Random Personality Mode:** If no hand detected for 30+ seconds, moves randomly every 5 seconds.
- **Web Interface:** Gradio-based GUI for full manual control and live monitoring.
- **LED Feedback:** RGB LED shows state: green = tracking, red = idle.

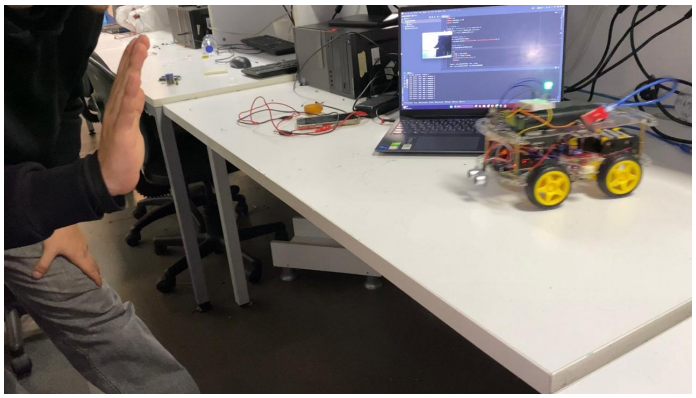

Figure 11: Robot actively following user's hand in follow-me mode

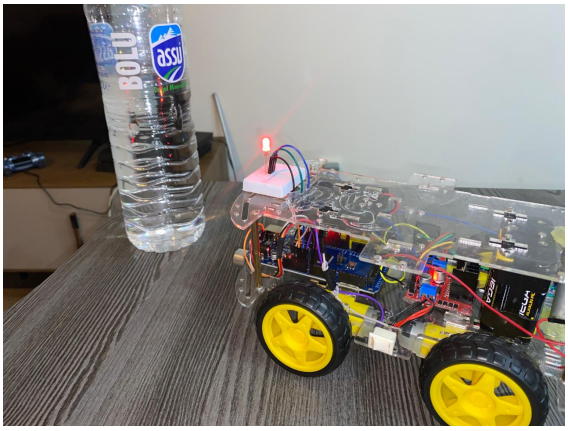

Figure 12: *
Before detection: obstacle in front

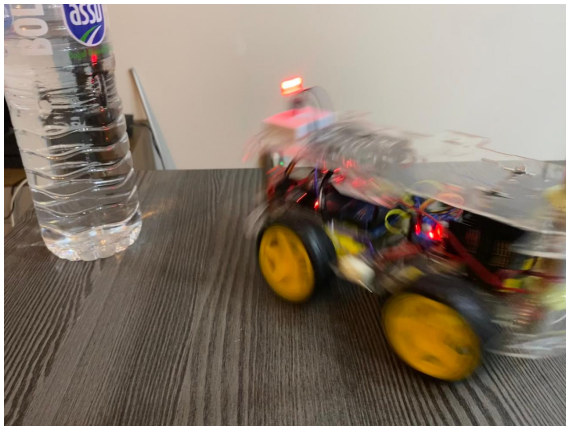

Figure 13: *
During avoidance: robot turning right

Figure 14: Robot avoiding obstacle using HC-SR04 detection logic

5 Work Plan and Budget

5.1 Timeline and Task Division

The project was a collaborative team effort, with responsibilities distributed according to each member's strengths:

- **Abdullah Al-Shobaki (2284612):** Led AI development, implemented the Python hand tracking pipeline using MediaPipe and OpenCV, and built the Gradio web interface for control and streaming.
- **Mohamed Alkozendar (2283149):** Assembled and wired the robot chassis, connected components, and programmed the ESP32-CAM streaming and HTTP interface.
- **Mohammed Al-Labani (2105990):** Programmed the Arduino Mega for motor control, sensor integration, and LED feedback, and ensured system stability and safety logic.

All team members contributed to testing, debugging, documentation, and preparing the final report and presentation.

The following Gantt chart shows the breakdown of major tasks across 11 weeks:

Task List	**W1**	**W2**	**W3**	**W4**	**W5**	**W6**	**W7**	**W8**	**W9**	**W10**	**W11**
1. Research and Design											
1.1 Define goals, do literature search	X	X									
1.2 Design system, select components		X	X								
2. Construction and Coding											
2.1 Assemble system A (motors + base)			X	X							
2.2 Assemble system B (ESP32 + logic)				X	X						
2.3 Develop software (CV, UI, Arduino)			X	X	X	X					
2.4 Integration and full wiring					X	X					
3. Testing and Documentation											
3.1 Prepare project proposal	X	X									
3.2 Test functionality + fix bugs					X	X	X				
3.3 Prepare report and poster							X	X	X		
3.4 Final demo and presentation									X	X	X

Table 1: Table 2. Gantt chart showing task breakdown across weeks

5.2 Component Costs

All required components were supplied by the course instructor, with the exception of the ESP32-CAM module and the FTDI USB programmer, which were purchased by the team.

Component	**Quantity**	**Cost (TL)**
ESP32-CAM Module	1	350
FTDI USB-to-Serial Programmer	1	70
Total Cost		**420 TL**

Table 2: Purchased Component Costs

6 Challenges and Solutions

The development of the ESP32-CAM AI Follow-Me Robot presented several technical and practical challenges that were successfully addressed throughout the project lifecycle:

- **Serial Communication Stability:** Early communication between the ESP32-CAM and Arduino Mega was inconsistent. This was resolved by configuring the serial interface with a stable baud rate of 115200 and optimizing the command protocol to avoid buffer overflows.
- **Power Supply Segmentation:** Voltage mismatches between the motor driver and microcontrollers caused system resets. This issue was mitigated by using two separate switches to independently control the power supply to the logic and motor components, ensuring electrical isolation and safety.
- **Wi-Fi Streaming Latency:** The ESP32-CAM introduced frame delay over the wireless video stream, impacting real-time responsiveness. The Python-side processing was optimized by downscaling the video resolution and reducing the detection frequency to maintain smooth hand tracking.

- **MediaPipe Performance Overhead:** MediaPipe's hand detection pipeline can be computationally intensive. To ensure responsiveness, the frame size and detection interval were fine-tuned to balance accuracy with performance on mid-range laptops.
- **Hardware Wiring Complexity:** Managing connections for multiple components—including motors, the ultrasonic sensor, RGB LED, and ESP32-CAM—initially led to cluttered and unstable wiring. This challenge was solved by utilizing the extended I/O capability of the Arduino Mega, which allowed each component to be connected directly to a dedicated pin, eliminating the need for a breadboard and improving overall system reliability.
- **Camera Placement Optimization:** Initially, the ESP32-CAM was mounted at chassis level, which limited its field of view and made it difficult to reliably detect the user's hand. To resolve this, a vertical metal rod was attached to the robot, and the camera was repositioned to the top of the rod, aligning it with the user's hand level and significantly improving tracking accuracy.

7 Conclusion

This project demonstrates the successful integration of embedded hardware, real-time computer vision, and network communication into an interactive robotic system. The ESP32-CAM Follow-Me Robot can autonomously follow a user using hand tracking, avoid obstacles in real time, and respond to manual control through a browser-based interface.

The system's modular architecture separates vision, control, and actuation responsibilities, ensuring scalability and ease of debugging. Furthermore, the addition of behavior modes such as random movement and dance enhance the robot's versatility and user engagement. This project serves as a proof of concept for practical, low-cost AI-enhanced robotics and reflects strong embedded systems design and interdisciplinary development skills.

Acknowledgements

We express our deep gratitude to our supervisor, Prof. Mahmut Ağan, for his continuous guidance, encouragement, and valuable feedback throughout the course of this project. This work was also developed with extensive support from ChatGPT-4o [6], which assisted across all stages — from component selection and wiring schematics to code development, debugging, and report documentation.

8 Future Improvements

Although the current system functions effectively, several enhancements could improve performance, autonomy, and user experience:

- **Onboard AI Processing:** To eliminate the dependency on a laptop, future versions of the system could integrate a more capable microcontroller such as the Raspberry Pi Zero 2 W or an AI-focused edge processor like the NVIDIA Jetson Nano. These platforms offer sufficient computational power to run real-time hand tracking models directly on-board, enabling a fully autonomous robot without compromising performance.
- **Battery Health Monitoring:** Adding voltage sensors and feedback logic would allow the system to alert the user or shut down safely under low power conditions.
- **Advanced Gesture Recognition:** Supporting additional hand gestures (e.g., open palm = start, fist = stop) could enable more nuanced and intuitive human–robot interaction.
- **Autonomous Navigation:** Incorporating SLAM (Simultaneous Localization and Mapping) would allow the robot to localize itself, build maps, and plan paths in unknown environments.
- **Mobile App Integration:** Developing a cross-platform mobile app for control and monitoring could extend usability beyond the local network.

9 References

[1] Google Research, "Mediapipe." `https://github.com/google/mediapipe`, 2023.

[2] E. Systems, "Esp32-cam development board." `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-cam.html`, 2023.

[3] ELEGOO Inc., "Hc-sr04 ultrasonic sensor datasheet." `https://components101.com/sensors/hc-sr04-ultrasonic-sensor`, 2022.

[4] OpenCV Team, "Opencv: Open source computer vision library." `https://opencv.org`, 2023.

[5] G. Team, "Gradio." `https://www.gradio.app`, 2023.

[6] OpenAI, "Chatgpt by openai." `https://chat.openai.com`, 2025.

10 Appendix

Arduino Mega Code (arduino_megabot.ino)

```
// === Movement command tracking ===
char command = 'S';       // Current movement command (default: Stop)
char lastCommand = 'S';   // Stores the last executed command

// === Motor driver pins (L298N) ===
const int in1 = 8;
const int in2 = 9;
const int in3 = 10;
const int in4 = 11;
const int ena = 5;        // PWM pin for left motor speed
const int enb = 6;        // PWM pin for right motor speed

// === Ultrasonic sensor pins ===
const int trigPin = A0;
const int echoPin = A1;

// === RGB LED pins ===
const int redLED = 4;
const int greenLED = 3;

// === Setup function ===
void setup() {
  Serial.begin(9600);  // Start serial communication

  // Set motor pins as output
  pinMode(in1, OUTPUT); pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT); pinMode(in4, OUTPUT);
  pinMode(ena, OUTPUT); pinMode(enb, OUTPUT);

  // Set motor speed to 50% (0{255 PWM range)
  analogWrite(ena, 200);
  analogWrite(enb, 200);

  // Set ultrasonic sensor pins
  pinMode(trigPin, OUTPUT);
```

```
  pinMode(echoPin, INPUT);

  // Set LED pins
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);

  updateLEDs();  // Show initial LED status
}

// === Main loop function ===
void loop() {
  float distance = measureDistance();  // Read distance from ultrasonic sensor

  // If an obstacle is detected within 20cm
  if (distance > 0 && distance < 20) {
    stopMoving();      // Stop robot
    delay(200);
    turnRight90();     // Perform a 90° right turn
    delay(500);
    stopMoving();
    delay(300);
  } else {
    // Read command sent from ESP32 over Serial
    if (Serial.available()) {
      char incoming = Serial.read();
      if (incoming == 'F' || incoming == 'B' || incoming == 'L' || incoming == 'R' || incoming == 'S') {
        command = incoming;

        // Only update LEDs when the command changes
        if (command != lastCommand) {
          updateLEDs();
          lastCommand = command;
        }
      }
    }

    // Execute the appropriate movement
    if (command == 'F') moveForward();
    else if (command == 'B') moveBackward();
```

```
    else if (command == 'L') moveLeft();
    else if (command == 'R') moveRight();
    else stopMoving();  // Default: stop
  }
}

// === Movement Functions ===
void moveForward() {
  digitalWrite(in1, LOW);  digitalWrite(in2, HIGH);  // Left motor forward
  digitalWrite(in3, HIGH); digitalWrite(in4, LOW);   // Right motor forward
}

void moveBackward() {
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW);   // Left motor backward
  digitalWrite(in3, LOW);  digitalWrite(in4, HIGH);  // Right motor backward
}

void moveLeft() {
  digitalWrite(in1, LOW);  digitalWrite(in2, HIGH);  // Left motor forward
  digitalWrite(in3, LOW);  digitalWrite(in4, LOW);   // Right motor stopped
}

void moveRight() {
  digitalWrite(in1, LOW);  digitalWrite(in2, LOW);   // Left motor stopped
  digitalWrite(in3, HIGH); digitalWrite(in4, LOW);   // Right motor forward
}

void stopMoving() {
  digitalWrite(in1, LOW); digitalWrite(in2, LOW);    // Both motors stopped
  digitalWrite(in3, LOW); digitalWrite(in4, LOW);
}

// === Turn the robot 90 degrees right ===
void turnRight90() {
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW);   // Left motor backward
  digitalWrite(in3, HIGH); digitalWrite(in4, LOW);   // Right motor forward
}

// === Measure distance using HC-SR04 ===
```

```
float measureDistance() {
  digitalWrite(trigPin, LOW); delayMicroseconds(2);
  digitalWrite(trigPin, HIGH); delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 30000);  // 30ms timeout
  if (duration == 0) return -1;                   // No echo received
  return duration * 0.034 / 2;                    // Convert to cm
}

// === Update LED status based on robot state ===
void updateLEDs() {
  if (command == 'S') {
    digitalWrite(greenLED, LOW);
    digitalWrite(redLED, HIGH);   // Red LED = idle
  } else {
    digitalWrite(greenLED, HIGH); // Green LED = moving
    digitalWrite(redLED, LOW);
  }
}
```

ESP32-CAM Code (esp32_stream_and_control.ino)

```
// === Include Required Libraries ===
#include "esp_camera.h"       // Enables camera functionality
#include <WiFi.h>             // Connects ESP32 to Wi-Fi
#include <WebServer.h>        // Handles incoming HTTP requests on port 80
#include "esp_http_server.h"  // For video stream server on port 81

// === Camera Pin Configuration for AI Thinker ESP32-CAM ===
// These are the hardware pins connected to the camera module
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
```

```
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

// === Wi-Fi Network Credentials ===
// Replace with the name and password of the Wi-Fi your ESP32 will connect to
const char* ssid = "iPhonea";
const char* password = "Aa123124";

// === Create HTTP Server on Port 80 ===
// This handles incoming movement commands like /move?dir=F
WebServer server(80);

// === Function: Handle Movement Commands from HTTP ===
// If a GET request is received like /move?dir=F, this function will run
void handleMove() {
  if (server.hasArg("dir")) { // Check if "dir" argument is present in URL
    String dir = server.arg("dir"); // Extract the direction value

    // Valid directions: F (forward), B (backward), L (left), R (right), S (stop)
    if (dir == "F" || dir == "L" || dir == "R" || dir == "S" || dir == "B") {
      Serial.write(dir.charAt(0));  // Send the command to Arduino Mega via Serial
      Serial.println("Sent: " + dir);  // Print to Serial Monitor
      server.send(200, "text/plain", "Sent " + dir);  // Respond to browser
      return;
    }
  }
  // If no valid "dir" received
  server.send(400, "text/plain", "Invalid");
}

// === Function: Start Camera Stream Server on Port 81 ===
// This function launches a stream at /stream that shows live MJPEG video
void startCameraServer() {
```

```
static httpd_handle_t stream_httpd = NULL;
httpd_config_t config = HTTPD_DEFAULT_CONFIG();
config.server_port = 81;

// Define streaming behavior
httpd_uri_t stream_uri = {
  .uri = "/stream",
  .method = HTTP_GET,
  .handler = [](httpd_req_t *req) {
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    size_t _jpg_buf_len;
    uint8_t * _jpg_buf;

    // Set HTTP header type for streaming
    res = httpd_resp_set_type(req, "multipart/x-mixed-replace; boundary=frame");
    if (res != ESP_OK) return res;

    // Stream loop: keeps capturing and sending frames forever
    while (true) {
      fb = esp_camera_fb_get();  // Capture a frame
      if (!fb) {
        Serial.println("Camera capture failed");
        continue;
      }

      _jpg_buf_len = fb->len;
      _jpg_buf = fb->buf;

      // Format the HTTP frame header
      char part_buf[64];
      size_t hlen = snprintf(part_buf, 64,
        "--frame\r\nContent-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n", _jpg_buf_len);

      // Send the frame to client
      res = httpd_resp_send_chunk(req, part_buf, hlen);
      res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
      res = httpd_resp_send_chunk(req, "\r\n", 2);
```

```
        esp_camera_fb_return(fb);  // Release the camera buffer
        if (res != ESP_OK) break;
      }
      return res;
    },
    .user_ctx = NULL
  };

  // Start streaming service
  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
    Serial.println("Stream server started on port 81");
  } else {
    Serial.println("Failed to start stream server!");
  }
}

// === Function: Initialize Camera with Config ===
void startCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;       // 20 MHz clock
```

```
  config.pixel_format = PIXFORMAT_JPEG; // MJPEG streaming
  config.frame_size = FRAMESIZE_QVGA;   // 320x240 resolution
  config.jpeg_quality = 10;             // 0 = best quality, 63 = worst
  config.fb_count = 1;                  // Use single frame buffer

  // Initialize the camera
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
}

// === Arduino Setup Function ===
void setup() {
  delay(3000);  // Wait for voltage to stabilize after power on
  Serial.begin(115200);  // Start Serial communication with Arduino Mega
  Serial.println("Booting...");

  startCamera();  // Initialize the camera

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi connected!");
  Serial.print("Stream URL: http://");
  Serial.print(WiFi.localIP());     // Show your local IP address
  Serial.println(":81/stream");

  // Setup endpoint for /move
  server.on("/move", handleMove);
  server.begin();
  Serial.println("HTTP server started.");
```

```
  startCameraServer();  // Launch the stream server
}

// === Arduino Loop Function ===
// Keeps listening for HTTP requests (e.g. move commands)
void loop() {
  server.handleClient();  // Handle incoming client connections
}
```

Python Code: Hand Tracking (hand_tracking.py)

```
# === Import required libraries ===
import cv2                      # OpenCV for image processing and display
import mediapipe as mp          # MediaPipe for hand tracking
import requests                 # To send HTTP GET requests to ESP32
import time                     # For timers and delays
import random                   # To generate random commands in idle mode

# === ESP32-CAM Configuration ===
ESP32_IP = "172.20.10.8"       # Replace with the actual IP address of your ESP32
MOVE_URL = f"http://{ESP32_IP}/move"         # Endpoint to send movement commands
STREAM_URL = f"http://{ESP32_IP}:81/stream"  # ESP32 camera stream URL

# === Initialize MediaPipe for hand tracking ===
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.7)

# === Open camera stream from ESP32 ===
cap = cv2.VideoCapture(STREAM_URL)

# === State and control variables ===
last_seen_time = time.time()       # Timestamp of last hand detection
in_random_mode = False             # Whether robot is in random movement mode
random_timer = 0                   # Timer for random command interval
random_command = 'S'               # Current random command (F, L, or R)
last_command = 'S'                 # Last sent command
last_send_time = time.time()       # Last time a command was sent
send_interval = 0.3                # Minimum time between command sends (seconds)
steer_timer = 0                    # Timer to control short L/R turns
```

```
steer_duration = 0.2              # Duration of L/R before stopping

# === Main loop ===
while True:
    # Read frame from the ESP32 video stream
    ret, frame = cap.read()
    if not ret or frame is None:
        print("No frame from ESP32")
        continue

    # Resize and convert frame to RGB for MediaPipe
    frame = cv2.resize(frame, (320, 240))
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(rgb)  # Detect hand landmarks

    command = 'S'  # Default command = Stop

    # === If hand is detected ===
    if results.multi_hand_landmarks:
        in_random_mode = False
        last_seen_time = time.time()

        # Get x-position of wrist landmark
        lm = results.multi_hand_landmarks[0].landmark
        cx = int(lm[mp_hands.HandLandmark.WRIST].x * 320)

        # Decide command based on horizontal wrist position
        if cx < 110:
            command = 'L'
        elif cx > 210:
            command = 'R'
        else:
            command = 'F'

        print(f"[Tracking] Hand at {cx}, Sending: {command}")
        steer_timer = time.time()  # Reset L/R turn timer

    # === If no hand detected ===
    else:
```

```
        time_since = time.time() - last_seen_time

        # If hand not seen for more than 10s → enter random mode
        if time_since > 10:
            if not in_random_mode:
                print("[!] Entering Random Mode")
                in_random_mode = True
                random_timer = 0

            # Every 5s → pick a new random direction
            if time.time() - random_timer > 5:
                random_command = random.choice(['F', 'L', 'R'])
                random_timer = time.time()
                print(f"[Random] New command: {random_command}")

            command = random_command
        else:
            print("[No Hand] Sending Stop")
            command = 'S'

    # === Shorten L/R turns to 0.2s then stop ===
    if command in ['L', 'R'] and time.time() - steer_timer > steer_duration:
        command = 'S'

    # === Send command to ESP32 if it's new or periodic ===
    now = time.time()
    if command != last_command or (now - last_send_time > send_interval):
        try:
            requests.get(f"{MOVE_URL}?dir={command}", timeout=0.05)
            last_command = command
            last_send_time = now
        except Exception as e:
            print("Failed to send command:", e)

    # === Display the video frame with command overlay ===
    cv2.putText(frame, f"CMD: {command}", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow("Follow Me", frame)
```

```
    # === Exit if 'q' is pressed ===
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# === Release resources ===
cap.release()
cv2.destroyAllWindows()
```

Python Code: Web Interface (`robot_ui.py`)

```
# === Import required libraries ===
import gradio as gr         # Gradio for building the web interface
import requests             # For sending HTTP requests to ESP32
import time                 # For delays in dance sequence

# === Define IP addresses and endpoints ===
ESP32_IP = "http://172.20.10.8"          # ESP32-CAM IP address on the Wi-Fi network
STREAM_URL = f"{ESP32_IP}:81/stream"    # MJPEG video stream from ESP32-CAM
MOVE_URL = f"{ESP32_IP}/move"           # Endpoint for sending movement commands

# === Function to send a basic movement command ===
def send_command(cmd):
    try:
        r = requests.get(f"{MOVE_URL}?dir={cmd}", timeout=0.3)
        return f"Sent: {cmd} | Response: {r.status_code}"
    except Exception as e:
        return f"Failed to send: {cmd} | Error: {e}"

# === Function to perform the dance sequence ===
def dance_mode():
    try:
        # Perform a full 360° spin using 4 right turns
        for _ in range(4):
            requests.get(f"{MOVE_URL}?dir=R", timeout=0.3)
            time.sleep(0.5)  # Pause for each 90° turn
            requests.get(f"{MOVE_URL}?dir=S", timeout=0.3)
            time.sleep(0.2)

        # Move backward for 2 seconds
```

```
        requests.get(f"{MOVE_URL}?dir=B", timeout=0.3)
        time.sleep(2)

        # Move forward for 2 seconds
        requests.get(f"{MOVE_URL}?dir=F", timeout=0.3)
        time.sleep(2)

        # Stop movement
        requests.get(f"{MOVE_URL}?dir=S", timeout=0.3)
        return "Dance complete!"
    except Exception as e:
        return f"Dance command failed: {e}"

# === Define Gradio interface layout ===
with gr.Blocks() as demo:
    gr.Markdown("##  Follow Me Robot Web UI")  # Title of the app

    # Display live camera stream
    gr.HTML(f"""
        <h4> Live ESP32-CAM Stream</h4>
        <img src="{STREAM_URL}" width="320" height="240" />
    """)

    # Directional buttons for manual control
    with gr.Row():
        gr.Button("Forward").click(lambda: send_command("F"))
    with gr.Row():
        gr.Button("Left").click(lambda: send_command("L"))
        gr.Button("Stop").click(lambda: send_command("S"))
        gr.Button("Right").click(lambda: send_command("R"))
    with gr.Row():
        gr.Button("Backward").click(lambda: send_command("B"))

    # Extra button for dance sequence
    with gr.Row():
        gr.Button("Dancing Mode").click(dance_mode)

# === Launch the web app ===
demo.launch()
```