Programming for AI

Project: Color Courier

24i-0039	Hamna Daud
24i-0112	Ritaj Suleman
24i-0025	Tahreem Fatima
24i-2573	Mehza Ayesh

Abstract

The project incorporates the use of line follower robot based on ESP32, two ESP32-CAM, image processing written in Python and MQTT communication. Where one down-firing camera will confirm the color of the line on the ground under the robot, there will be a front-facing camera to determine the color of boxes on the top. When the two colors identified by the robot are identical the robot halts and the box falls. Otherwise it continues drifting along the line.

Problem Statement

The issue of this project is that an automated sorting device needs to be developed to detect the item in terms of color and move on its own. We s/he also wished to develop a mobile robot that would be able to detect color on the floor, in the case of warehouses or labs, where material must move between the work stations, and make sorting decision in real time.

Objectives

This projects goal was to:

- Construct a following line mobile robot.
- Detecting color Two ESP32-CAM modules.

- Python image stream Process images with OpenCV.
- MQTT wireless communication.
- Break the mechanism of the robot when the color conditions are equal.
- Activates and deactivates servo platform on reception of match.

Features and System

- Real-time streaming of two ESP32-CAMs.
- Color detection using Python OpenCV.
- Two-colour contrast (front and bottom) of the camera.
- MQTT Python-ESP32 communication.
- Line following on an IR sensor array based on ES32.
- The automatic cessation of matching with colors.
- WiFi-enabled control
- Servo-driven sorting platform (90 degree on the match)

Colour Courier robot recognizes color based on two ESP32-CAM video streams which are processed by python. The Python program identifies colors with the help of the OpenCV and compares the two streams and sends the MQTT message to the ESP32. On the ESP32, a 5-IR-sensor line follower algorithm is also running. In a scenario whereby the Python notices a color match in the front cameras as to that in the bottom cameras, it sends a command to the robot via MQTT to stop the motor and rotate the servo motor to drop the box. Otherwise, the bot carries on with autonomous navigation.

Team Members & Roles

· Tahreem – MQTT Python communication & control

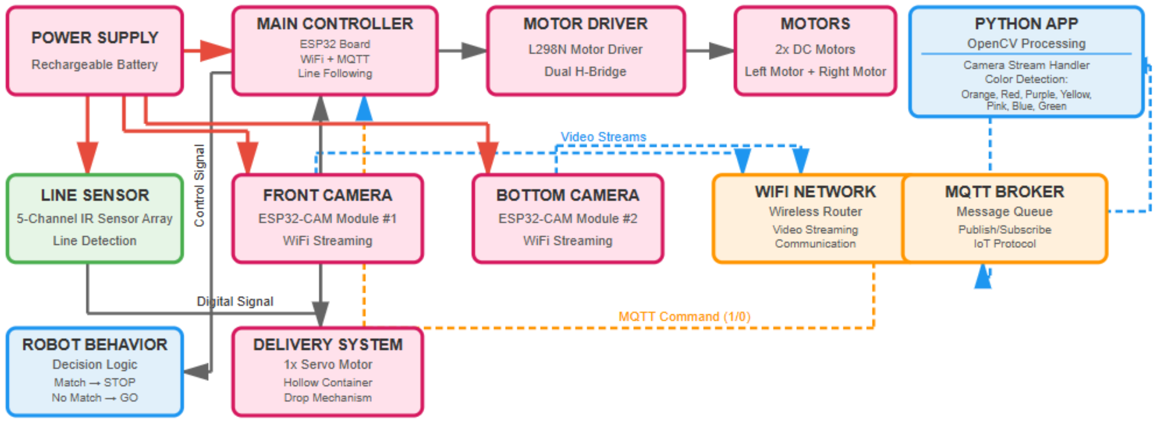
· Ritaj – MQTT ESP32 code + Line Follower logic

- Mehza – ESP32-CAM streaming & camera module setup
- Hamna – Image processing using OpenCV

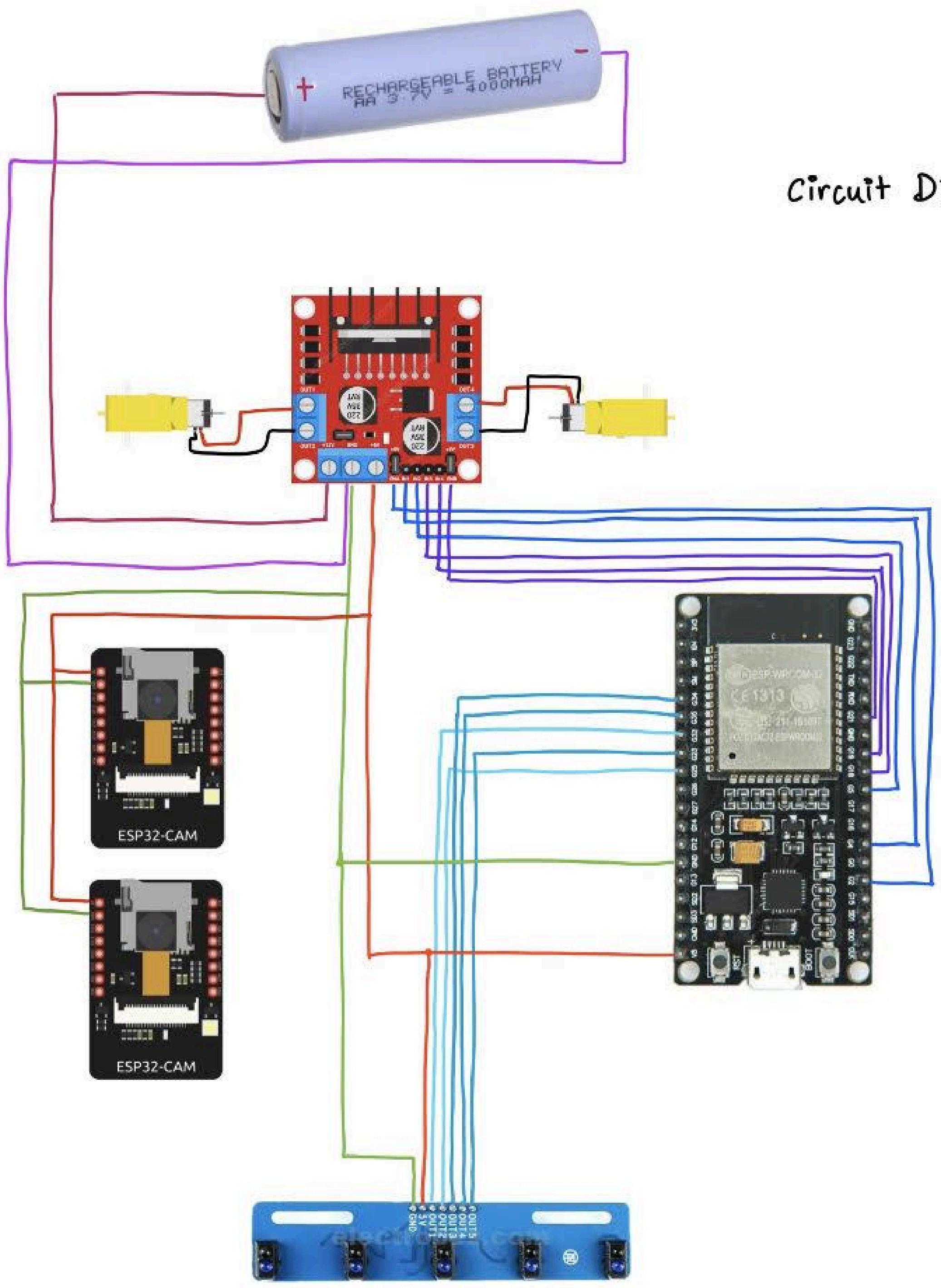
Hardware Components

- ESP32-CAM’s
- ESP32 Dev Board (robot controller)
- IR Sensor Array (5-sensor line follower)
- Motor Driver (L298N or similar)
- DC Motors with wheels
- Servo Motor (for platform rotation)
- Power Supply
- Chassis
- Computer running Python OpenCV

Block Diagram



Circuit Explanation



Circuit Diagram

Challenges Faced & Lessons Learned

Several challenges were encountered during the design process of Color courier, which altered the end-release architecture of the system. These tests taught us the way to consider various alternatives and make engineering choices in terms of utility and what is efficient.

1. Switching Away From Arduino: Bluetooth Limitations

We initially planned to use an Arduino Uno complete board together with a Bluetooth module (HC-05/HC-06) so that we could transmit our commands wirelessly.

Nonetheless, the following issues arose during the testing time:

- Bluetooth connections were not very reliable and had to be redone repeatedly.
- There was not an efficient communication.
- Bluetooth range was limited
- The Bluetooth debugging was difficult.

These problems rendered the robot unreliable and slow.

Due to this, we have resolved to break completely with Arduino + Bluetooth and switched to ESP32, which provided:

- Built-in WiFi
- Faster processing
- Multiple GPIOs
- Compatibility with MQTT
- A more stable communication

This modification enhanced the responsiveness of the robot greatly.

2. Choosing MQTT Instead of HTTP

Early designs discussed the use of Python to send commands to the robot by using HTTP requests.

However, when we tested it, we found that HTTP was a significant disadvantage to our particular applications:

Why HTTP Was Not Suitable

- The robot is not well able to accept running commands.
- It was very ineffective due to headers and reconnecting processes.
- Not very suitable with rapid, frequent signals such as stop, move, turn, etc.
- A delay in the processing of the requests was equal to a late response of the robot.
- More difficult to keep in real time colour scan.

Why MQTT Was the Better Choice

- Easy to carry, very fast messages.
- Continuous communication
- Designed for IoT devices
- It is wholly supported by ESP32 libraries.
- Facilitates instant updates of Python.
- More robust in respect to stop immediately conditions.
- MQTT eventually enabled us to come as close to flawless operation as possible.

3. ESP32-CAM Lessons Learned

With the help of two ESP32-CAM, we learned a number of important things:

- They are power-sensitive, and inadequate amount of current makes them break.
- Our greatest issue was stable WiFi streaming.
- The HSV color detection is affected a lot by light conditions.

4. Integration Was the Hardest Part

Individually, each module worked well.

• Cameras streamed correctly.

• Python detected color.

• MQTT sent messages.

• The robot followed the line.

Nonetheless, the incorporation of all that turned out to cause certain issues:

- Latency was experienced when the two cameras were streamed simultaneously.
- MQTT messages were not synchronized with frame detection.
- It would cause parts to cease to work even though everything is correct.
- Most times there would be no WiFi connection.

Code

The following sections include complete source code.

Python – Dual Camera Color Detection + MQTT

```python
import cv2
import numpy as np

import paho.mqtt.client as mqtt

import time


# ===== CAMERA STREAM URLS =====

CAMERA_URL_1 = "http://192.168.8.101:81/stream"   # FRONT CAMERA

CAMERA_URL_2 = "http://192.168.8.102:81/stream"   # BOTTOM CAMERA


# ===== MQTT CONFIGURATION =====

MQTT_BROKER = "broker.hivemq.com"

MQTT_PORT = 1883

MQTT_TOPIC = "linefollow/colormatch"

MQTT_CLIENT_ID = "PythonColorDetector_CamVersion"


# ===== VALID COLORS =====

VALID_COLORS = ["red", "green", "blue", "yellow", "pink", "orange",
"purple"]


# ===== MQTT SETUP WITH ERROR HANDLING =====

mqtt_client = mqtt.Client(client_id=MQTT_CLIENT_ID)

mqtt_connected = False


def on_connect(client, userdata, flags, rc):

    global mqtt_connected
    if rc == 0:

        print("✓ MQTT Connected successfully!")

        mqtt_connected = True

    else:

        print(f"✗ MQTT Connection failed with code: {rc}")

        mqtt_connected = False


def on_disconnect(client, userdata, rc):

    global mqtt_connected

    mqtt_connected = False

    print("✗ MQTT Disconnected!")


mqtt_client.on_connect = on_connect

mqtt_client.on_disconnect = on_disconnect


# Try to connect with error handling

try:

    print("Attempting MQTT connection...")

    mqtt_client.connect(MQTT_BROKER, MQTT_PORT, 60)

    mqtt_client.loop_start()

    time.sleep(2)  # Wait for connection


    if not mqtt_connected:
        print("⚠ WARNING: MQTT not connected. Will work in offline
mode.")

except Exception as e:

    print(f"⚠ MQTT Connection Error: {e}")

    print("⚠ Running in OFFLINE mode (no MQTT messages will be sent)")

    mqtt_connected = False


# ===== COLOR DETECTION =====

def detect_color(frame):

    """Detect main color in the center of the frame"""

    if frame is None:

        return None


    h, w, _ = frame.shape

    cx, cy = w // 2, h // 2


    # Take small region in center

    region = frame[cy-20:cy+20, cx-20:cx+20]

    hsv = cv2.cvtColor(region, cv2.COLOR_BGR2HSV)


    # Color ranges in HSV (sensitive version)

    color_ranges = {

        "red": [

            [(0, 70, 40), (10, 255, 255)],
            [(170, 70, 40), (180, 255, 255)]

        ],



        "orange": [

            [(11, 80, 80), (25, 255, 255)]

        ],



        "yellow": [

            [(26, 80, 80), (35, 255, 255)]

        ],



        "green": [

            [(36, 40, 40), (85, 255, 255)]

        ],



        "blue": [

            [(86, 80, 40), (130, 255, 255)]

        ],



        "purple": [

            [(131, 40, 40), (160, 255, 255)]

        ],
        "pink": [

            [(161, 20, 80), (169, 200, 255)],

            [(0, 20, 80), (10, 200, 255)]

        ]

    }


    dominant = None

    max_pixels = 0


    for color, ranges in color_ranges.items():

        mask = None


        for color_range in ranges:

            lower = np.array(color_range[0])

            upper = np.array(color_range[1])

            temp_mask = cv2.inRange(hsv, lower, upper)


            if mask is None:

                mask = temp_mask

            else:

                mask = cv2.bitwise_or(mask, temp_mask)


        count = cv2.countNonZero(mask)
```

```python
        if count > max_pixels:
            dominant = color
            max_pixels = count

    return dominant

# ===== MQTT SEND WITH ERROR HANDLING =====
def send_match(match):
    if mqtt_connected:
        try:
            mqtt_client.publish(MQTT_TOPIC, "1" if match else "0", qos=1)
            print(f"📤 MQTT SENT: {'MATCH (1)' if match else 'NO MATCH
(0)'}")
        except Exception as e:
            print(f"⚠ MQTT Send Error: {e}")
    else:
        print(f"⚠ OFFLINE: Would send {'MATCH (1)' if match else 'NO
MATCH (0)'}")


# ===== MAIN LOOP =====
def main():
    print("\n=== Starting Dual Camera Color Detection ===")
    print(f"MQTT Status: {'Connected ✓' if mqtt_connected else 'Offline
✗'}\n")
```

```
cap_front = cv2.VideoCapture(CAMERA_URL_1, cv2.CAP_FFMPEG)

cap_bottom = cv2.VideoCapture(CAMERA_URL_2, cv2.CAP_FFMPEG)


if not cap_front.isOpened():

    print("ERROR: Cannot open front camera stream")

    return


if not cap_bottom.isOpened():

    print("ERROR: Cannot open bottom camera stream")

    return


print("✓ Both cameras connected!")


while True:

    ret1, frame1 = cap_front.read()

    ret2, frame2 = cap_bottom.read()


    if not ret1 or not ret2:

        print("⚠ Camera read error!")

        continue


    front_color = detect_color(frame1)
```

```python
        bottom_color = detect_color(frame2)


        print(f"Front: {front_color} | Bottom: {bottom_color}")


        if front_color and bottom_color:

            match = (front_color == bottom_color)

            send_match(match)


        # Draw detection info on frames

        cv2.putText(frame1, f"Front: {front_color}", (10, 30),

                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        cv2.putText(frame2, f"Bottom: {bottom_color}", (10, 30),

                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)


        # Show windows for debugging

        cv2.imshow("Front Camera", frame1)

        cv2.imshow("Bottom Camera", frame2)


        # Exit by pressing 'q'

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

            break


    cap_front.release()
    cap_bottom.release()

    cv2.destroyAllWindows()


    if mqtt_connected:

        mqtt_client.loop_stop()

        mqtt_client.disconnect()

        print("\n✓ MQTT Disconnected cleanly")


if __name__ == "__main__":

    main()
```

ESP32 – MQTT + Line Follower

```cpp
#include <WiFi.h>

#include <PubSubClient.h>

#include <ESP32Servo.h>


// ===== WiFi Credentials =====

const char* ssid = "FAST Faculty";

const char* password = "";
```

```
// ===== MQTT Broker Settings =====

const char* mqtt_server = "broker.hivemq.com";

const int mqtt_port = 1883;

const char* mqtt_topic = "linefollow/colormatch";

const char* mqtt_client_id = "ESP32_LineFollower";


WiFiClient espClient;

PubSubClient mqtt(espClient);


// ===== SERVO SETTINGS =====

Servo myServo;

#define SERVO_PIN 13

#define SERVO_REST_POS 0

#define SERVO_ACTIVE_POS 90

#define SERVO_ACTIVE_TIME 3000  // 3 seconds in milliseconds


bool servoActive = false;

unsigned long servoActivatedAt = 0;


// ===== IR SENSOR PINS =====

#define ir1 32

#define ir2 25

#define ir3 34
```

```
#define ir4 35

#define ir5 23


// ===== MOTOR DRIVER PINS =====

#define IN1  4

#define IN2  5

#define ENA  2

#define IN3  18

#define IN4  19

#define ENB  21


int speed = 200;

bool colorMatch = false;

bool botStopped = false;

unsigned long lastMqttMessage = 0;


const uint8_t SAMPLES = 5;

const uint8_t SAMPLE_DELAY_MS = 2;


void mqttCallback(char* topic, byte* payload, unsigned int length) {

  Serial.print("Message arrived [");

  Serial.print(topic);

  Serial.print("]: ");
  String message;

  for (int i = 0; i < length; i++) {

    message += (char)payload[i];

  }

  Serial.println(message);



  if (message == "1") {

    colorMatch = true;

    Serial.println("✓ COLOR MATCH - Bot stopping & Servo activating for 3
seconds");

    activateServo();

  } else if (message == "0") {

    colorMatch = false;

    Serial.println("✓ NO MATCH - Bot continuing");

  }



  lastMqttMessage = millis();

}



void activateServo() {

  myServo.write(SERVO_ACTIVE_POS);

  servoActive = true;

  servoActivatedAt = millis();
  Serial.print("🔄 Servo moved to: ");

  Serial.print(SERVO_ACTIVE_POS);

  Serial.println("° for 3 seconds");

}



void checkServoTimer() {

  if (servoActive && (millis() - servoActivatedAt >= SERVO_ACTIVE_TIME)) {

    myServo.write(SERVO_REST_POS);

    servoActive = false;

    Serial.print("🔄 Servo reset to: ");

    Serial.print(SERVO_REST_POS);

    Serial.println("° (3 seconds elapsed)");

  }

}



void reconnectMQTT() {

  while (!mqtt.connected()) {

    Serial.print("Attempting MQTT connection...");



    if (mqtt.connect(mqtt_client_id)) {

      Serial.println("connected!");

      mqtt.subscribe(mqtt_topic);

      Serial.print("Subscribed to: ");
      Serial.println(mqtt_topic);

    } else {

      Serial.print("failed, rc=");

      Serial.print(mqtt.state());

      Serial.println(" trying again in 5 seconds");

      delay(5000);

    }

  }

}



void setup() {

  Serial.begin(115200);

  delay(1000);



  Serial.println("\n=== Line Follower Bot (MQTT + Timed Servo) Starting
===");



  // Setup Servo

  myServo.attach(SERVO_PIN);

  myServo.write(SERVO_REST_POS);

  Serial.print("✓ Servo initialized on pin ");

  Serial.print(SERVO_PIN);

  Serial.print(" at ");

  Serial.print(SERVO_REST_POS);
  Serial.println("°");


  // Connect to WiFi

  WiFi.mode(WIFI_STA);

  WiFi.begin(ssid, password);

  Serial.print("Connecting to WiFi");


  int attempts = 0;

  while (WiFi.status() != WL_CONNECTED && attempts < 20) {

    delay(500);

    Serial.print(".");

    attempts++;

  }


  if (WiFi.status() == WL_CONNECTED) {

    Serial.println("\n✓ WiFi Connected!");

    Serial.print("IP Address: ");

    Serial.println(WiFi.localIP());

  } else {

    Serial.println("\n✗ WiFi Connection Failed!");

  }


  // Setup MQTT
```

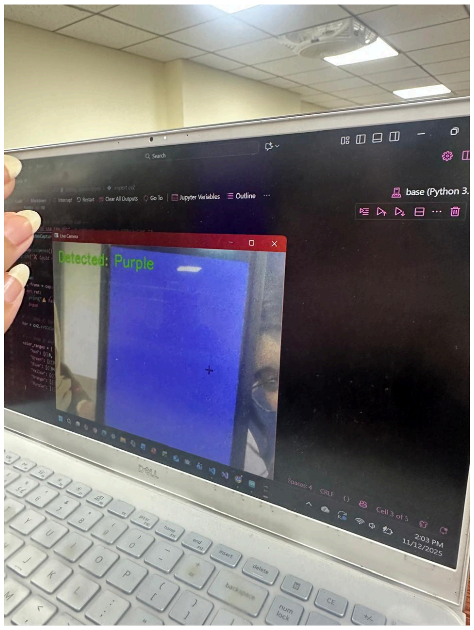
```
  mqtt.setServer(mqtt_server, mqtt_port);

  mqtt.setCallback(mqttCallback);



  // Motor Direction Pins

  pinMode(IN1, OUTPUT);

  pinMode(IN2, OUTPUT);

  pinMode(IN3, OUTPUT);

  pinMode(IN4, OUTPUT);

  pinMode(ENA, OUTPUT);

  pinMode(ENB, OUTPUT);



  // Sensor Pins

  pinMode(ir1, INPUT_PULLUP);

  pinMode(ir2, INPUT_PULLUP);

  pinMode(ir3, INPUT_PULLUP);

  pinMode(ir4, INPUT_PULLUP);

  pinMode(ir5, INPUT_PULLUP);

}



void loop() {

  // Maintain MQTT connection

  if (!mqtt.connected()) {

    reconnectMQTT();
  }

  mqtt.loop();


  // Check if servo timer has elapsed

  checkServoTimer();


  // If color matches, stop the bot

  if (colorMatch) {

    if (!botStopped) {

      stopMotors();

      botStopped = true;

      Serial.println("⏸️ COLOR MATCH! Bot stopped.");

    }

    return;

  }


  botStopped = false;


  // Read sensors

  int s1 = readSensor(ir1);

  int s2 = readSensor(ir2);

  int s3 = readSensor(ir3);

  int s4 = readSensor(ir4);
  int s5 = readSensor(ir5);


  static unsigned long lastPrint = 0;

  if (millis() - lastPrint > 1000) {

    Serial.printf("Sensors: [%d][%d][%d][%d][%d]\n", s1, s2, s3, s4, s5);

    lastPrint = millis();

  }


  if (s3 && !s2 && !s4) {

    moveForward();

  }

  else if (s2 && !s4) {

    turnLeft();

  }

  else if (s4 && !s2) {

    turnRight();

  }

  else if (s2 && s3) {

    turnLeft();

  }

  else if (s3 && s4) {

    turnRight();

  }
```

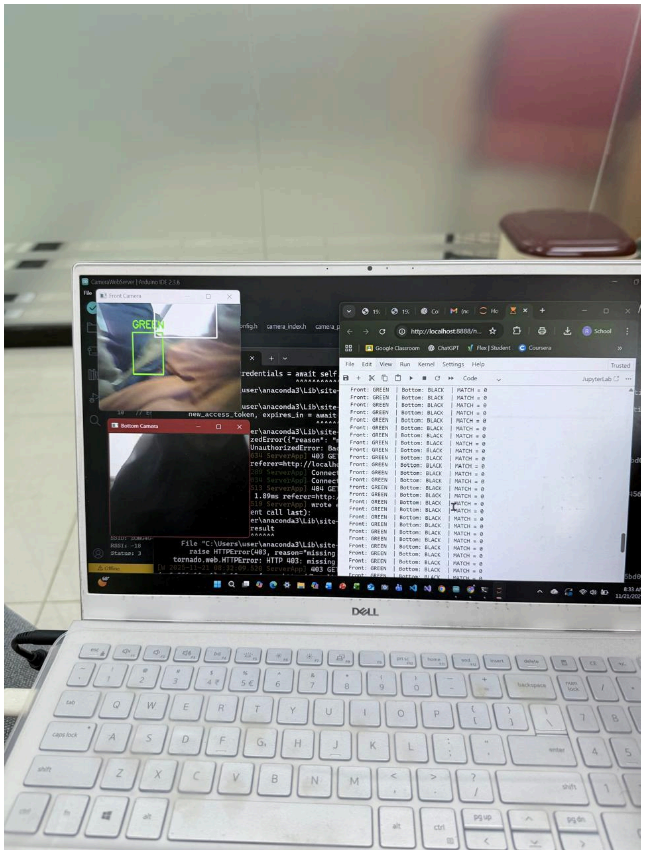
```
  else if (s1) {

    sharpLeft();

  }

  else if (s5) {

    sharpRight();

  }

  else {

    stopMotors();

  }



  delay(50);

}



int readSensor(uint8_t pin) {

  uint8_t hits = 0;

  for (uint8_t i = 0; i < SAMPLES; ++i) {

    if (digitalRead(pin) == LOW) hits++;

    delay(SAMPLE_DELAY_MS);

  }

  return (hits > (SAMPLES / 2)) ? 1 : 0;

}



void moveForward() {
  analogWrite(ENA, speed);

  analogWrite(ENB, speed);

  digitalWrite(IN1, HIGH);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, HIGH);

  digitalWrite(IN4, LOW);

}


void turnLeft() {

  analogWrite(ENA, 140);

  analogWrite(ENB, speed);

  digitalWrite(IN1, HIGH);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, HIGH);

  digitalWrite(IN4, LOW);

}


void turnRight() {

  analogWrite(ENA, speed);

  analogWrite(ENB, 140);

  digitalWrite(IN1, HIGH);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);

}



void sharpLeft() {

  analogWrite(ENA, 0);

  analogWrite(ENB, speed);

  digitalWrite(IN1, LOW);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, HIGH);

  digitalWrite(IN4, LOW);

}



void sharpRight() {

  analogWrite(ENA, speed);

  analogWrite(ENB, 0);

  digitalWrite(IN1, HIGH);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, LOW);

  digitalWrite(IN4, LOW);

}



void stopMotors() {

  analogWrite(ENA, 0);
  analogWrite(ENB, 0);

  digitalWrite(IN1, LOW);

  digitalWrite(IN2, LOW);

  digitalWrite(IN3, LOW);

  digitalWrite(IN4, LOW);

}
```

ESP32-CAM – WebServer Stream

The ESP32-CAM modules use the standard Arduino example 'CameraWebServer'. This initializes WiFi, configures the camera pins, and launches a local web stream accessible via http://<IP>:81/stream.

Behind the scenes




GREEN