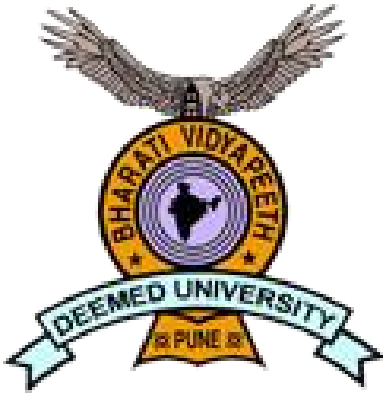
BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY),

COLLEGE OF ENGINEERING,

PUNE-411043

Department of Information Technology



A

Project Based Learning

On

"Traffic Control Monitoring System With Camera"

Submitted By

Roll No.	PRN No.	Name
1	2214110233	Harsh Kumar Agrawal
6	2214110238	Aryan Kumar
22	2214110255	Kumar Sarthak
27	2214110260	R Gokul

Under the Guidance of

Prof. Prakash Devale

DEPARTMENT OF INFORMATION TECHNOLOGY

Certificate

This is to certify that the work under Project Based Learning (PBL) for the topic **Traffic Monitoring System with Camera"** is carried out by Harsh Agarwal, Aryan Kumar, R Gokul and Kumar Sarthak under the guidance of **Prof. Prakash Devale** in partial fulfillment of the requirement for the degree of **Bachelor of Technology in Information Technology Semester-V** of **Bharati Vidyapeeth (Deemed to be University), Pune** during the academic year **2024-2025**.

Date:- 16/10/2024

Prof. Prakash Devale

GUIDE

ACKNOWLEDGEMENT

We would like to express deepest appreciation towards Dr Vidula Sohoni, Principal, Bharati Vidyapeeth (Deemed to Be University) College of Engineering, Pune, India and Dr.Sandeep Vanjale Head of the Department (IT), who invaluable supported us in completing this project.

We are profoundly grateful to Prof. Prakash Devale, Project guide for his expert guidance and continuous encouragement throughout to see that this project rights, its target since its commencement to its completion.

At last we must express our sincere heartfelt gratitude to all the staff members of IT Department who helped us directly or indirectly during this course of work.

Harsh Agarwal

R Gokul

Aryan Kumar

Kumar Sarthak

BTech (IT) Sem-v

INDEX

INTRODUCTION

Traffic monitoring is a crucial component of urban development and road safety. As cities grow and vehicle numbers increase, monitoring and managing traffic in real-time becomes essential to avoid congestion and accidents. Traditional systems rely on expensive infrastructure and extensive manual intervention. However, with advancements in technology, it is now possible to develop more cost-effective and automated solutions.

Introduction to Traffic Monitoring:

A **Traffic Monitoring System using ESPCAM-32** is a project designed to track and analyze traffic patterns using the ESP32-CAM, a low-cost development board with an integrated camera. This system can capture real-time images or videos of road traffic, detect congestion, and monitor vehicle movement. The data collected by the ESP32-CAM can be processed either on the device itself or sent to a server for further analysis.

The **Traffic Monitoring System using ESPCAM-32** takes advantage of the small form factor and wireless capabilities of the ESP32-CAM to provide an efficient, low-cost solution for real-time traffic surveillance. The system is capable of:

- **Capturing real-time images and videos** of traffic.
- **Identifying traffic congestion** or monitoring vehicle counts.
- **Communicating over Wi-Fi** to send data to a server or cloud-based platform for further analysis.
- **Potential integration with AI** for detecting vehicles or recognizing patterns in traffic flow.

ESPCAM-32:

The **ESP32-CAM** (often referred to as ESPCAM-32) is a small, cost-effective development board that combines the power of the ESP32 microcontroller with a built-in camera. It's widely used in IoT, security, and video streaming projects due to its capabilities and compact size. Below are the key details about the ESP32-CAM:

Key Features:

1. **Microcontroller**:
    - **ESP32-S module**: A dual-core microcontroller with Wi-Fi and Bluetooth (BLE) capabilities.
    - Operating at up to 240 MHz with 520 KB SRAM and external flash memory support.
2. **Camera**:
    - Comes with an **OV2640 camera module** (2MP, with resolutions up to 1600x1200).
    - Capable of capturing still images or video.
    - Supports various image formats including JPEG, BMP, and grayscale.
3. **Storage**:

- Supports **MicroSD cards** for data logging or local storage of images and video.

4. **Connectivity**:
   - Built-in **Wi-Fi** for wireless data transmission or remote access.
   - **Bluetooth** is available for additional communication needs.
5. **GPIO Pins**:
   - Several General Purpose Input/Output (GPIO) pins for connecting additional sensors, actuators, or external hardware.
   - Some GPIOs are shared with camera functionality, so pin usage must be planned carefully.
6. **Power**:
   - Can be powered via a 5V supply (USB or external).
   - Low-power modes for energy-efficient applications.
7. **Programming**:
   - Supports programming via the **Arduino IDE** or **ESP-IDF (Espressif IoT Development Framework)**.
   - Can be flashed using a USB-to-TTL converter (the ESP32-CAM doesn't have a built-in USB port).

Project Overview:

To develop a cost-effective and efficient traffic monitoring system that captures real-time images and data of road traffic using the ESP32-CAM. The system will analyze traffic patterns, detect congestion, and provide data visualization through a web interface.

Key Features:

- **Real-time Image Capture**: Capture images and stream video of traffic using the OV2640 camera module.
- **Traffic Analysis**: Identify and analyze traffic flow, density, and vehicle counts.
- **Wireless Communication**: Use Wi-Fi to transmit data to a server for further analysis and visualization.
- **User Interface**: Provide a web-based interface for monitoring traffic data in real time.
- **Data Logging**: Optionally log traffic data for future analysis, enabling studies of traffic patterns over time.

CIRCUIT DIAGRAM:

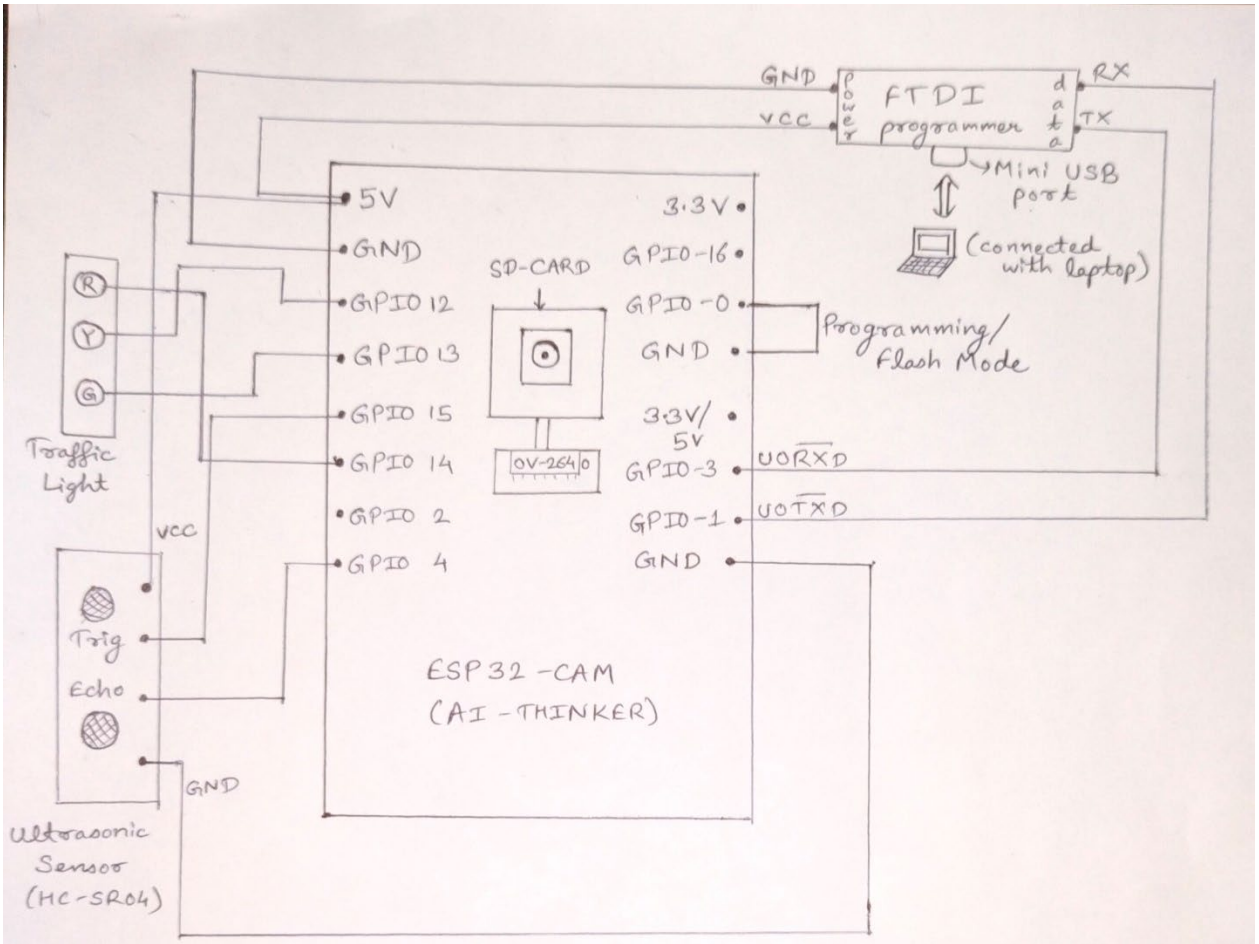


Fig:- Circuit Diagram

Components:-

1.) ESP-32 CAM (AI-THINKER)
2.) 3 LED's (Red, Yellow, Green)
3.) 3 resistors (220 Ω)
4.) Jumper Wires (M-M, M-F, F-F)
5.) 1 - Ultrasonic Sensor (HC-SR04)
6.) FTDI - Programmer
7.) Breadboard
8.) Model Cars

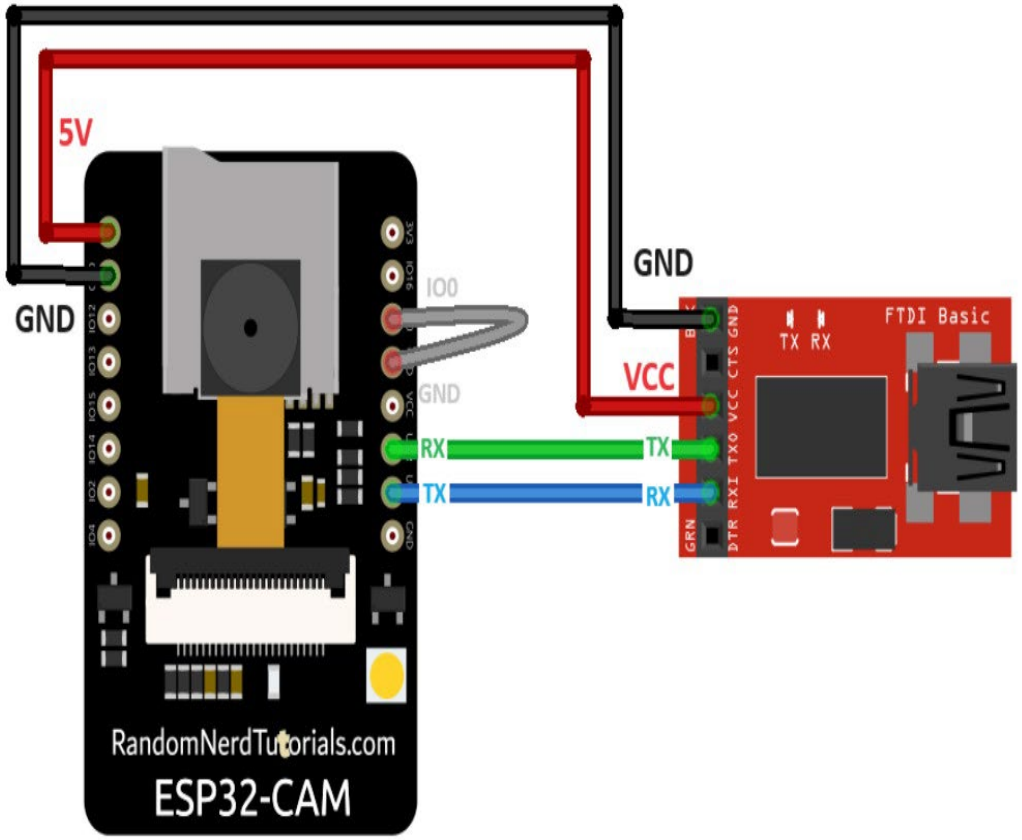
Component and Software Used:

Components:

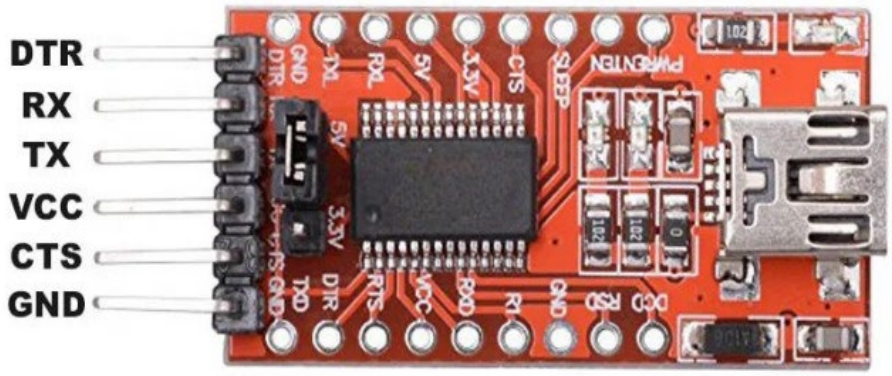
- **Hardware**:
  - ESP32-CAM module
  - Power supply (5V)
  - USB-to-TTL converter (for programming)
  - Optional: MicroSD card (for data logging)
  - Additional sensors (IR, ultrasonic) for enhanced data collection (optional)

- **Software**:
  - Arduino IDE or ESP-IDF for programming the ESP32-CAM.
  - Libraries for camera functionality and web server setup (e.g., ESP32 Camera Library, ESPAsyncWebServer).
  - A server-side application for data processing and visualization (e.g., Flask or Node.js).

- ESP32-CAM module:

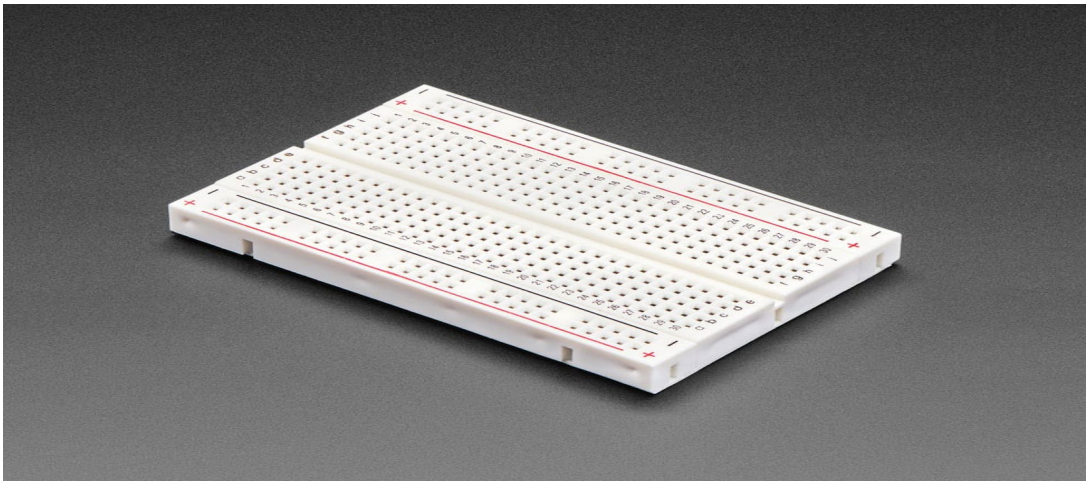


- FTDI Programmer:



- Ultrasonic Sensor:

- Breadboard:



- LEDs:

- Resistor:

- Jumper Wires:

Applications:

- **Smart Parking Management**:
  - Monitor parking lot occupancy in real time and guide drivers to available spaces.
  - Capture images of vehicles entering and leaving to help manage parking fees.
- **Public Transportation Monitoring**:
  - Monitor bus and other public transportation vehicles to provide real-time updates on arrival times and delays.
  - Analyze ridership patterns to improve scheduling and routing.
- **Traffic Violation Detection**:
  - Use the system to detect traffic violations such as running red lights, speeding, or illegal parking.
  - Capture evidence for issuing fines or alerts to traffic enforcement agencies.
- **Event Traffic Management**:
  - Monitor traffic patterns during large events (concerts, festivals) to manage crowd control and parking.
  - Provide real-time traffic updates to attendees to facilitate smoother access to the venue.
- **Pedestrian Safety Monitoring**:
  - Analyze pedestrian traffic at crosswalks and busy intersections to improve safety measures.
  - Use data to inform city planners about necessary infrastructure changes (e.g., new crosswalks or traffic lights).
- **Vehicle Classification**:
  - Classify vehicles into categories (cars, trucks, motorcycles) for better traffic management and road planning.
  - Collect data for environmental studies by monitoring vehicle types and emissions.
- **Remote Monitoring for Law Enforcement**:
  - Allow police and emergency services to monitor specific areas for traffic flow or emergencies.
  - Provide live video feeds to first responders for better situational awareness during incidents.
- **Infrastructure Health Monitoring**:
  - Monitor the condition of roadways and traffic signals based on vehicle flow and stop durations.
  - Collect data to assess wear and tear, helping with maintenance schedules and resource allocation.

- **Traffic Simulation and Modeling**:
    - Use collected data for traffic simulation studies to predict future traffic patterns and impacts of infrastructure changes.
    - Assist urban planners in designing better road systems and reducing congestion.
- **Environmental Impact Studies**:
    - Monitor vehicle emissions and noise levels in specific areas.
    - Collect data to analyze the impact of traffic on air quality and propose mitigation strategies.
- **Data Sharing for Smart City Initiatives**:
    - Integrate with other smart city applications to provide comprehensive urban analytics.
    - Share data with transportation agencies and urban planners for collaborative decision-making.
- **Automated Traffic Reports**:
    - Generate periodic traffic reports based on collected data for city officials, providing insights into traffic trends.
    - Offer public access to traffic data through a web portal or mobile app.
- **Integration with Emergency Response Systems**:
    - Alert traffic management systems of accidents or emergencies to optimize response times and reroute traffic.
    - Provide first responders with live traffic conditions to assist in planning routes to emergencies.

Execution Steps:

Here's how to connect your components for the ESP32-CAM traffic signal project with the ultrasonic sensor HC-SR04 and the FTDI program

1. ESP32-CAM Setup

- FTDI Programmer: This will be used to upload code to the ESP32-CAM. Ensure that the FTDI programmer is set to 5V.
- Connections for Programming:
- Connect the GND pin on the FTDI programmer to GND on the ESP32-CAM.
- Connect the VCC (5V) pin on the FTDI to the 5V pin on the ESP32-CAM.
- Connect the TX pin on the FTDI to the U0R (RX) pin on the ESP32-CAM.
- Connect the RX pin on the FTDI to the U0T (TX) pin on the ESP32-CAM.
- To put the ESP32-CAM into programming mode, connect the IO0 pin to GND before uploading code. Once the code is uploaded, disconnect IO0 from GND.

2. Connecting the Traffic Lights (LEDs)

- Red Light (LED): Connect the anode (longer leg) of the red LED to GPIO 14 on the ESP32-CAM via a 220Ω resistor, and the cathode (shorter leg) to GND.
- Yellow Light (LED): Connect the anode of the yellow LED to GPIO 12 via a 220Ω resistor, and the cathode to GND.
- Green Light (LED): Connect the anode of the green LED to GPIO 13 via a 220Ω resistor, and the cathode to GND.

3. Connecting the Ultrasonic Sensor HC-SR04 for Vehicle Detection

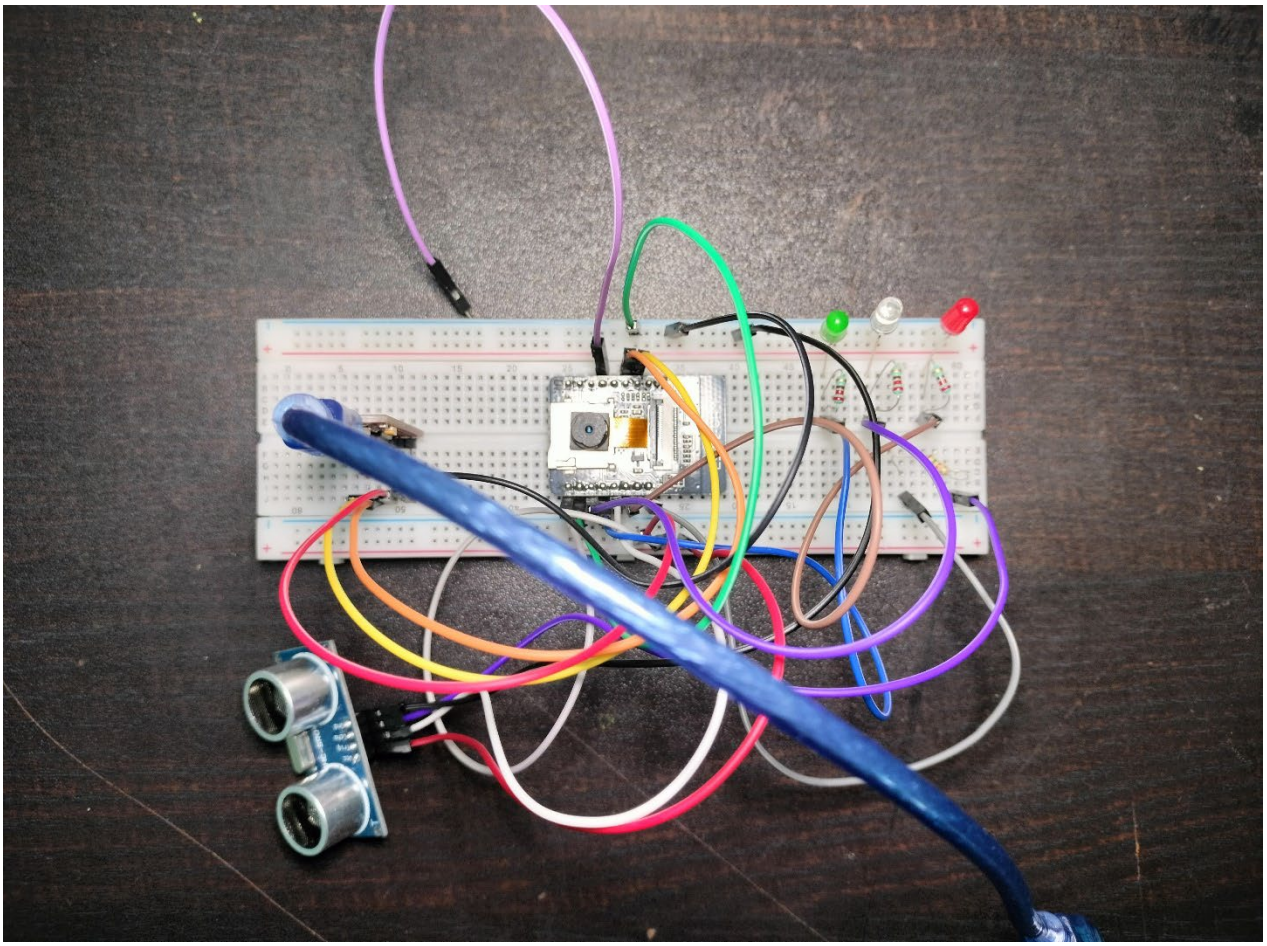
- Power: Connect the VCC pin of the ultrasonic sensor to the 5V pin on the ESP32-CAM, and GND to the common ground.
- Trigger Pin: Connect the Trig pin of the HC-SR04 to GPIO 15 on the ESP32-CAM.
- Echo Pin: Connect the Echo pin of the HC-SR04 to GPIO 4 on the ESP32-CAM.
- Note: Since the Echo pin outputs 5V, use a voltage divider (e.g., 1kΩ and 2kΩ resistors) to drop it to a safe 3.3V for the ESP32-CAM's GPIO.

4. Powering the ESP32-CAM

- During development, the ESP32-CAM can be powered through the FTDI programmer, which connects to your computer via USB.
- For standalone use, you can connect a 5V power supply directly to the ESP32-CAM's 5V and GND pins.

5. General Wiring Notes

- Breadboard and Jumper Wires: Use the breadboard for stable connections, especially for the LEDs and resistors.
- Ensure all components share a common GND connection to avoid issues with grounding and ensure reliable operation.
- This setup allows the ESP32-CAM to control traffic lights and capture images based on vehicle detection using the ultrasonic sensor. After code uploading, remove the IO0 pin from GND and reset the ESP32-CAM to switch from programming mode to normal operation mode.



Code:

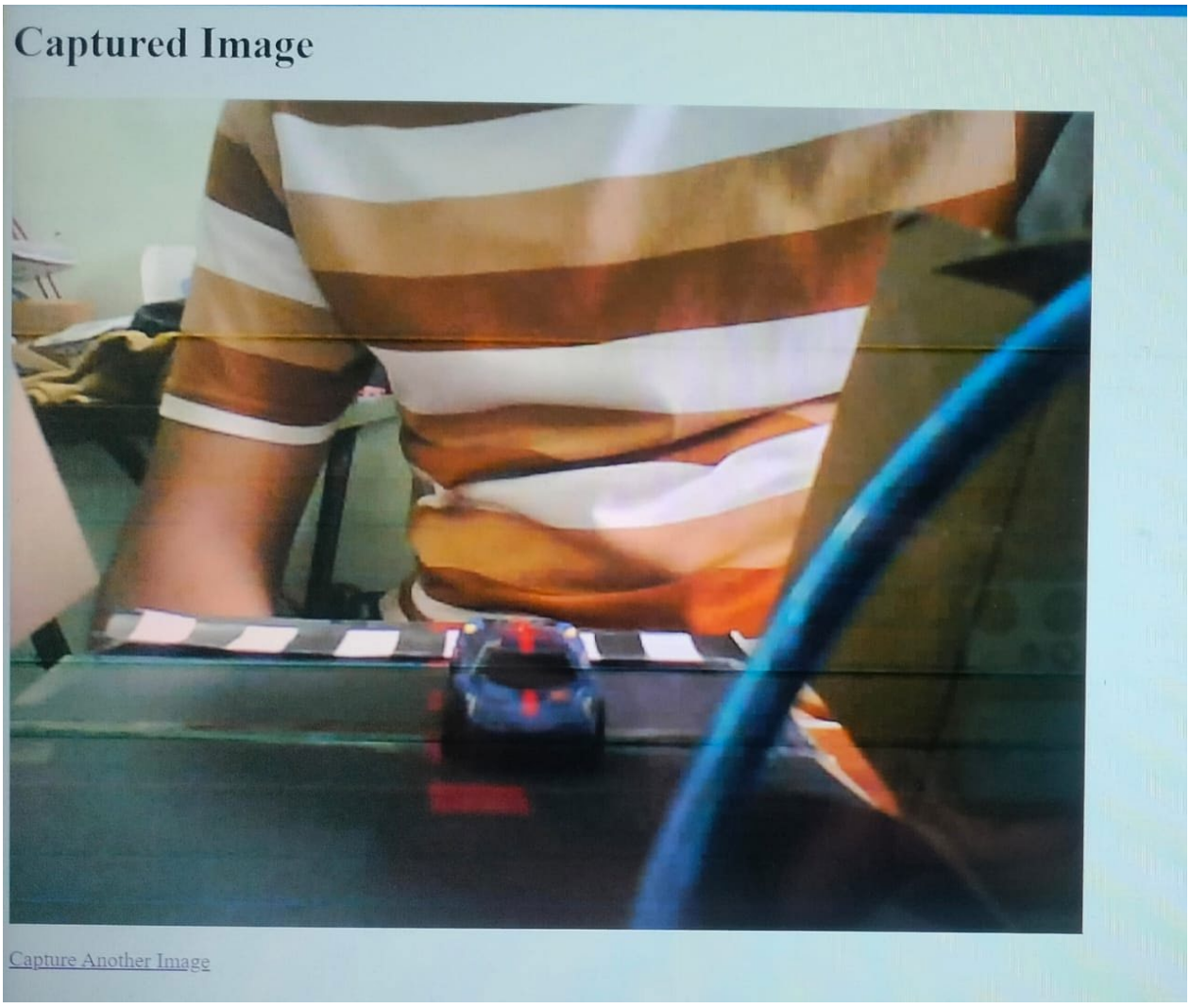
```
#include "esp_camera.h"
#include <WiFi.h>
#include <WebServer.h>
#include "Arduino.h"
#include "Base64.h"

// Camera Pin Definitions for ESP32-CAM AI-Thinker
#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

// Traffic Light Pins
const int redLight = 14;
const int yellowLight = 12;
const int greenLight = 13;

// Ultrasonic Sensor Pins
const int trigPin = 15;
const int echoPin = 4;

// WiFi credentials
const char* ssid = "Flat 15";
const char* password = "patna@123";

// Web server on port 80
WebServer server(80);

int lightState = 0; // 0=Green, 1=Yellow, 2=Red

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

  // Configure traffic light pins
  pinMode(redLight, OUTPUT);
  pinMode(yellowLight, OUTPUT);
  pinMode(greenLight, OUTPUT);

  // Configure ultrasonic sensor pins
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
```

```
  // Start with green light
  digitalWrite(greenLight, HIGH);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to WiFi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());

  // Start camera
  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;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 10;
  config.fb_count = 1;
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  // Start server and define routes
  server.on("/", HTTP_GET, handleRoot);
  server.on("/capture", HTTP_GET, captureAndServeImage);
  server.begin();
}

void loop() {
  manageTrafficLights();
  int distance = getDistance();
  if (distance > 0 && distance < 20 && (lightState == 1 || lightState == 2)) {
    captureAndServeImage();
  }
  server.handleClient();
```

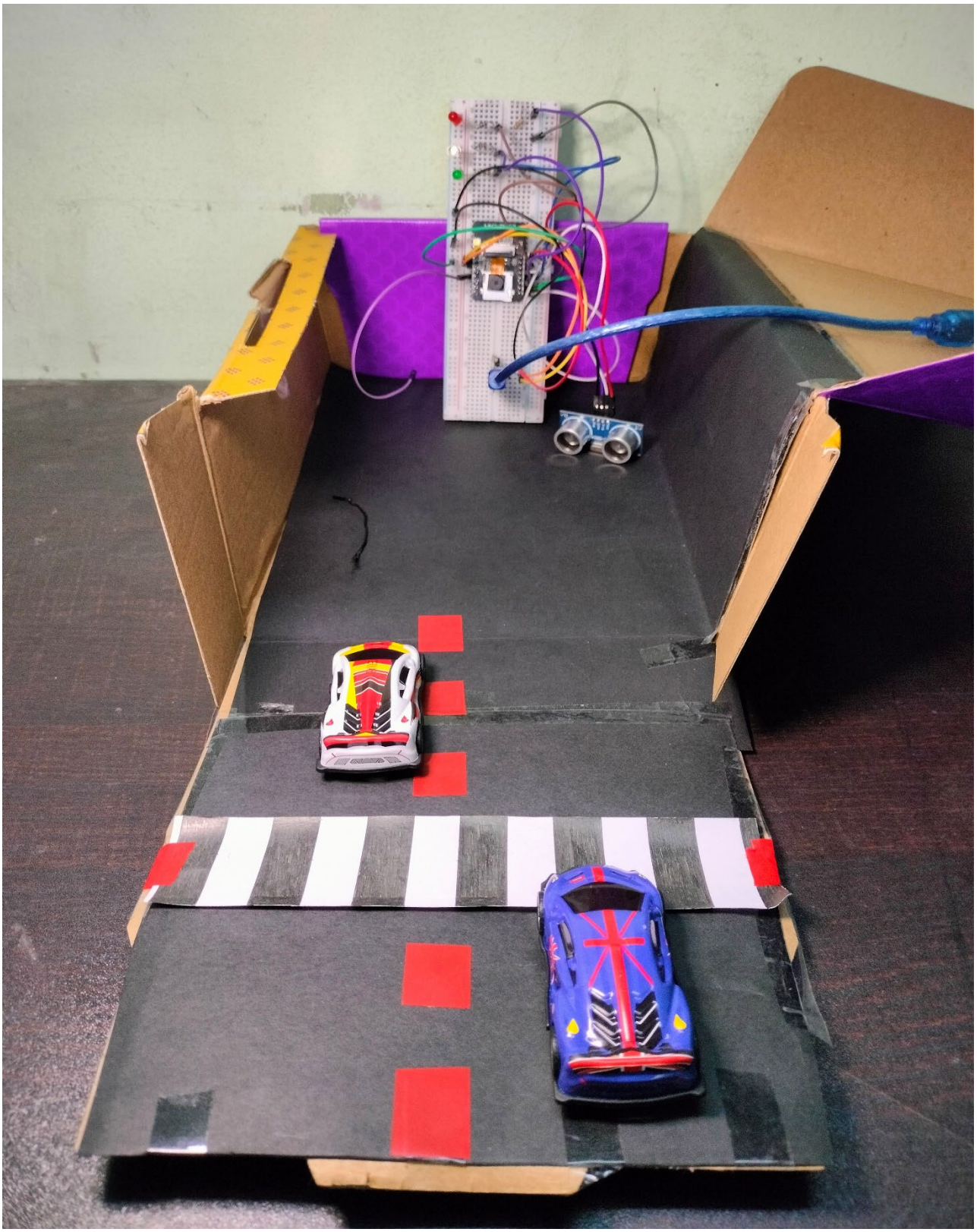
```
  delay(1000);
}

void manageTrafficLights() {
  switch (lightState) {
    case 0: // Green Light
      digitalWrite(greenLight, HIGH);
      delay(10000);
      digitalWrite(greenLight, LOW);
      lightState = 2;
      break;
    case 1: // Yellow Light
      digitalWrite(yellowLight, HIGH);
      delay(6000);
      digitalWrite(yellowLight, LOW);
      lightState = 0;
      break;
    case 2: // Red Light
      digitalWrite(redLight, HIGH);
      delay(10000);
      digitalWrite(redLight, LOW);
      lightState = 1;
      break;
  }
}

int getDistance() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH);
  return duration * 0.034 / 2;
}

void captureAndServeImage() {
  camera_fb_t* fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    server.send(500, "text/plain", "Camera capture failed");
    return;
  }

  // Encode image as base64
  String imageBase64 = base64::encode((uint8_t*)fb->buf, fb->len);
  esp_camera_fb_return(fb);

  // Create HTML page with embedded image
  String html = "<html><body style='background-color:powderblue;'>";
  html += "<h1>Captured Image</h1>";
  html += "<img src=\"" + imageBase64 + "\"/>";
  html += "<p><a href=\"/capture\">Capture Another Image</a></p>";
  html += "</body></html>";

  server.send(200, "text/html", html);
}
```

```
void handleRoot() {
  String html = "<h1>Traffic Signal Monitoring System</h1><h2>Trimurti Chowk</h2><p><a
href=\"/capture\">Capture Image</a></p>";
  server.send(200, "text/html", html);
}
```

Output:



Model:



Conclusion:

The ESP32-CAM-based traffic monitoring system has versatile applications beyond just traffic analysis, contributing to improved urban mobility, safety, and efficiency. By leveraging real-time data and connectivity, this system can play a significant role in developing smarter, more responsive urban environments.

The **Traffic Monitoring System using ESP32-CAM** represents a significant advancement in urban traffic management and surveillance technologies. By leveraging the capabilities of the ESP32-CAM—a compact, cost-effective microcontroller with integrated camera and Wi-Fi features—this project addresses critical challenges faced by modern cities regarding traffic flow, safety, and infrastructure planning.

In summary, the **Traffic Monitoring System using ESP32-CAM** offers a practical, innovative, and scalable solution to some of the pressing challenges in urban traffic management. Its ability to provide real-time data and insights will not only help cities enhance mobility and safety but also contribute to the development of smarter, more efficient urban environments. With ongoing advancements in technology and data analytics, the potential applications and benefits of such systems will continue to expand, positioning them as vital components of future smart city initiatives.