Name: Syed Shah
NUID: 002459070

Mid-Cycle Report: Smart Traffic Management System

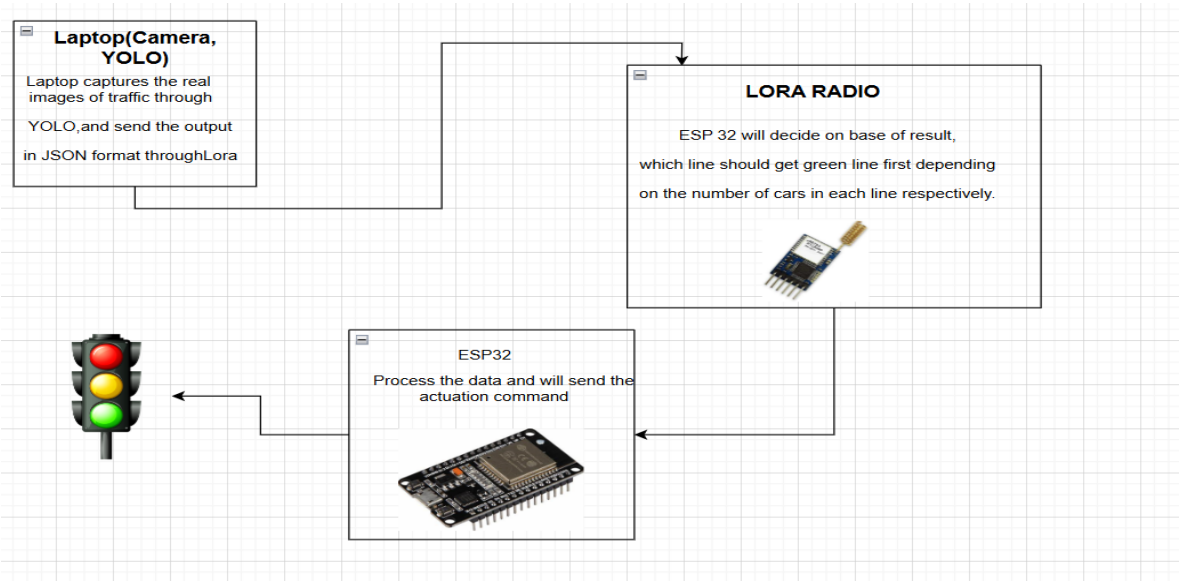
1. Introduction

The objective of this project is to develop a Smart Traffic Management System using computer vision, embedded systems, and LoRa communication. The system detects cars using a camera, identifies their position in different lanes, and transmits the processed information to an ESP32 via LoRa radio. The ESP32 will use the received data to manage traffic flow based on lane congestion.

This report presents the completed work, including vision-based detection, real-time lane assignment, LoRa data transmission, ESP32-based data reception, and emulated decision logic for signal switching.

2. Methodology

The system is implemented in three key phases:
1. Car Detection using YOLOv8 – Identifying cars in real-time using a trained deep-learning model.
2. Frame Distribution for Lane Classification – Differentiating between lanes A and B based on car positions.
3. LoRa Communication for Data Transmission – Sending Lane congestion data to ESP32.



3. Car Detection using YOLOv8

3.1 Why Use a Pretrained Model?

A pretrained model, such as YOLOv8, has already been trained on large datasets, making it more efficient and accurate for object detection tasks. Using a pretrained model allows us to leverage existing feature extraction capabilities, reducing the training time and improving performance, especially when working with small datasets.

3.2 Why COCO Dataset?

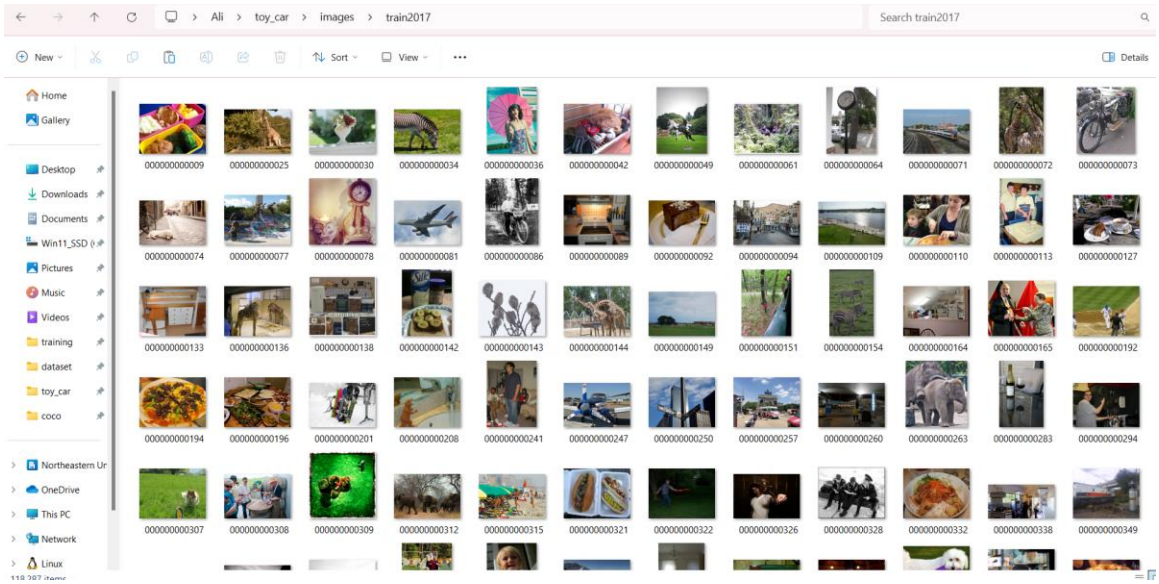
The COCO (Common Objects in Context) dataset is one of the largest and most diverse datasets for object detection. It contains 80 different object classes, including cars, making it ideal for training a robust vehicle detection model. The dataset includes real-world images with different lighting, weather conditions, and environments, ensuring that the model performs well in live traffic scenarios.

3.2 Dataset Preparation

The dataset was obtained from Roboflow and includes images of toy cars in various settings. The dataset is structured into training, validation, and test sets:

- Train: 80% of the dataset
- Validation: 10%
- Test: 10%

The dataset is labeled in YOLO format, with a dataset.yaml file defining class names.



Coco data set car images for training

C: > Users > user > toy_car > ! coco.yaml

```yaml

# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: C:/Users/user/toy_car
train: train2017.txt # train images (relative to 'path') 118287 images
val: val2017.txt # val images (relative to 'path') 5000 images
test: test-dev2017.txt # 20288 of 40670 images, submit to https://competitions.codalab.org/competitions/20794

# Classes
names:
  0: person
  1: bicycle
  2: car
  3: motorcycle
  4: airplane
  5: bus
  6: train
```

Data.yaml

3.3 YOLOv8 Model Training

The YOLOv8 model was trained using Ultralytics YOLOv8 framework with the following command:

C: > Users > user > toy_car > toy_yolo_training.py > ...

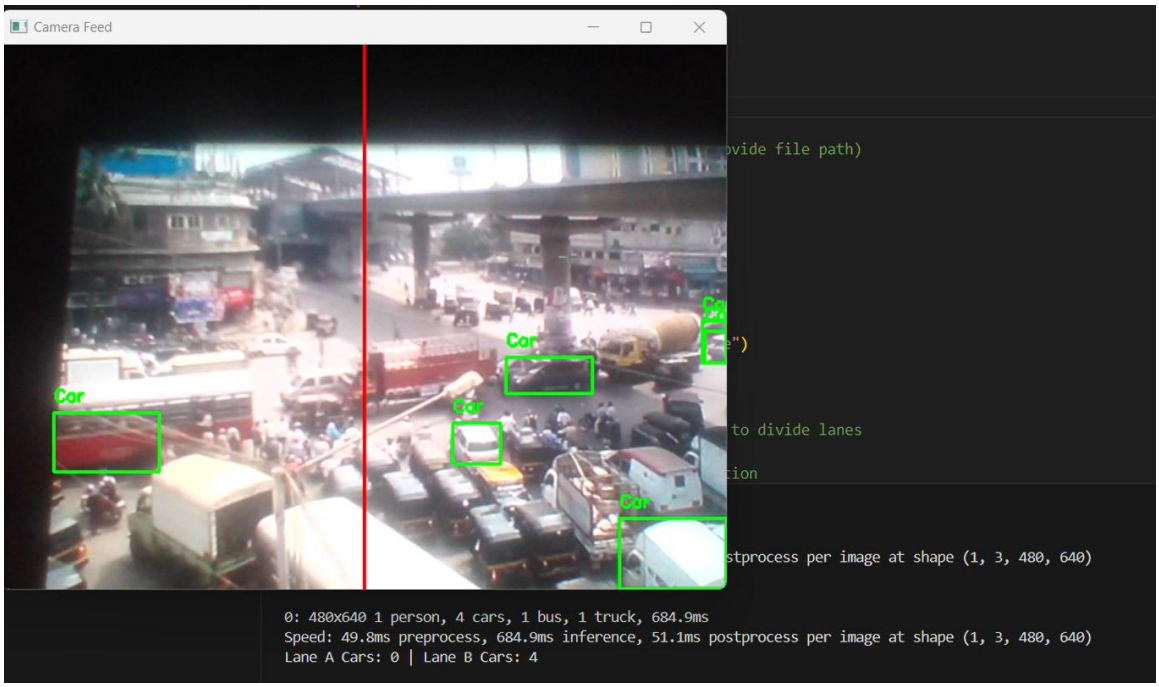
```python
if __name__ == '__main__':
    from ultralytics import YOLO

    # Load a pretrained YOLOv8 model
    model = YOLO("yolov8n.pt")

    # Train the model on your custom dataset
    model.train(data="coco.yaml", epochs=100, imgsz=416,batch=16,
                device="cuda:0",  # Change this if you have a GPU (e.g., "cuda:0")  # Use GPU
                workers=4  # Offload preprocessing to CPU
                )  # Resume from last checkpoint if needed )

    print("✅ Training Complete!")
```

Training of Model using pretrained model

Results:

```
Validating runs\detect\train3\weights\best.pt...
Ultralytics 8.3.84 🚀 Python-3.12.9 torch-2.5.1+cu121 CUDA:0 (NVIDIA GeForce GTX 1050, 2048MiB)
Model summary (fused): 72 layers, 3,006,428 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 8/8 [00:03<00:00,  2.26it/s]
                   all        254        307      0.954      0.991      0.986      0.919
                   Car         66         66      0.929          1      0.979      0.894
                   Car         69         69      0.972      0.996      0.992      0.916
                   Car         81         81      0.947      0.975       0.98      0.929
                   Car         91         91      0.968      0.992      0.992      0.937
Speed: 0.3ms preprocess, 4.3ms inference, 0.0ms loss, 2.9ms postprocess per image
Results saved to runs\detect\train3
✅ Training Complete!
(yolovenv) PS C:\Users\user\toy_car>
```

After training results



Example of output

3.4 Lane Classification

To separate detected cars into Lane A and Lane B, the bounding box X-coordinates are used, That's how our final code will look like:

Code:

```python
import time
import cv2
from ultralytics import YOLO

# Load the YOLO model
model = YOLO("C:/Users/user/training/runs/detect/train4/weights/best.pt")

# Open the video source (0 for webcam, or provide file path)
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Error: Could not open webcam.")
    exit()

lane_split_x = 320  # Adjust based on your video frame size

while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Failed to capture frame")
        break

    results = model(frame)  # Run YOLO detection
    car_count = {"Lane A": 0, "Lane B": 0}
```

```
    for result in results:
        for box in result.boxes:
            if int(box.cls) == 0:  # Class ID 0 is "car"
                x1, y1, x2, y2 = map(int, box.xyxy[0])  # Get bounding box
                x_center = (x1 + x2) // 2
                lane = classify_lane(x_center, lane_split_x)
                car_count[lane] += 1
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(frame, lane, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    print(f"Lane A: {car_count['Lane A']}, Lane B: {car_count['Lane B']}")

    cv2.imshow("Camera Feed", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

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

4. LoRa Communication for Data Transmission

After lane classification, the system transmits the number of cars in each lane to an ESP32 via LoRa radio. The Python script uses the YOLOv8 model to detect cars and classify lanes in real time. It then sends the car counts to an ESP32 via LoRa using AT+SEND commands. The LoRa module is configured on port COM3 at 115200 baud.

This string is embedded in the AT+SEND command to address 10 (ESP32).

4.1 Python Script

```
import time

import cv2

import serial

from ultralytics import YOLO


# Load YOLO model

model = YOLO("yolov8n.pt")


# Open the video source (0 for webcam, or provide file path)

cap = cv2.VideoCapture(0)

if not cap.isOpened():

   print("Error: Could not open webcam.")

   exit()


# Configure LoRa module (Connected via USB-UART)

LORA_PORT = "COM3"  # Change as needed (e.g., "/dev/ttyUSB0")

BAUD_RATE = 115200


try:

   ser = serial.Serial(LORA_PORT, BAUD_RATE, timeout=1)

   time.sleep(2)  # Allow LoRa module to initialize

   if ser.is_open:

      print(f"LoRa module connected on {LORA_PORT}")

   else:
```

```
        print("Error: Serial port not open.")
        exit()
except Exception as e:
    print(f"Error: {e}")
    exit()

# === LORA Configuration Function ===
def configure_lora(serial_port):
    commands = [
        'AT+RESET',
        'AT+MODE=0',
        'AT+ADDRESS=11',
        'AT+NETWORKID=6',
        'AT+BAND=433000000'
    ]
    for cmd in commands:
        serial_port.write((cmd + 4'\r\n').encode())
        time.sleep(0.5)  # Wait for response
        while serial_port.in_waiting:
            response = serial_port.readline().decode().strip()
            print(f"{cmd} -> {response}")

# Call the config function
configure_lora(ser)
```

```
# Start timer for sending updates every 3 seconds
last_sent_time = time.time()

while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Failed to capture frame")
        break

    frame_width = frame.shape[1]
    mid_x = frame_width // 2  # Middle point to divide lanes

    results = model(frame)  # Run YOLO detection
    car_count_a = 0  # Lane A (Left Half)
    car_count_b = 0  # Lane B (Right Half)

    if results and results[0].boxes is not None:
        for box in results[0].boxes:
            if int(box.cls) == 2:  # Class ID 2 is "car"
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                center_x = (x1 + x2) // 2

                if center_x < mid_x:
                    car_count_a += 1
                else:
```

```
            car_count_b += 1


        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(frame, "Car", (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)


# Draw lane division line
cv2.line(frame, (mid_x, 0), (mid_x, frame.shape[0]), (0, 0, 255), 2)


# Show the live video feed
cv2.imshow("Camera Feed", frame)


# Send data every 3 seconds without slowing video
if time.time() - last_sent_time >= 3:
    payload = f"LANE1:{car_count_a},LANE2:{car_count_b}"
    command = f"AT+SEND=10,{len(payload)},{payload}\r\n"
    ser.write(command.encode())
    ser.flush()
    print(f"Sent: {command.strip()}")
    print(f"Lane A Cars: {car_count_a} | Lane B Cars: {car_count_b}")
    last_sent_time = time.time()


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

cap.release()

ser.close()

cv2.destroyAllWindows()

5. ESP32 Logic and Emulation

The ESP32 is configured with the same LoRa settings as the Python side and is set to address 10. It continuously listens for messages from the LoRa module over UART and parses incoming traffic data.

5.1 Real-Time Data Reception

Using FreeRTOS, a dedicated UART task listens to incoming data and updates lane1_count and lane2_count variables globally.

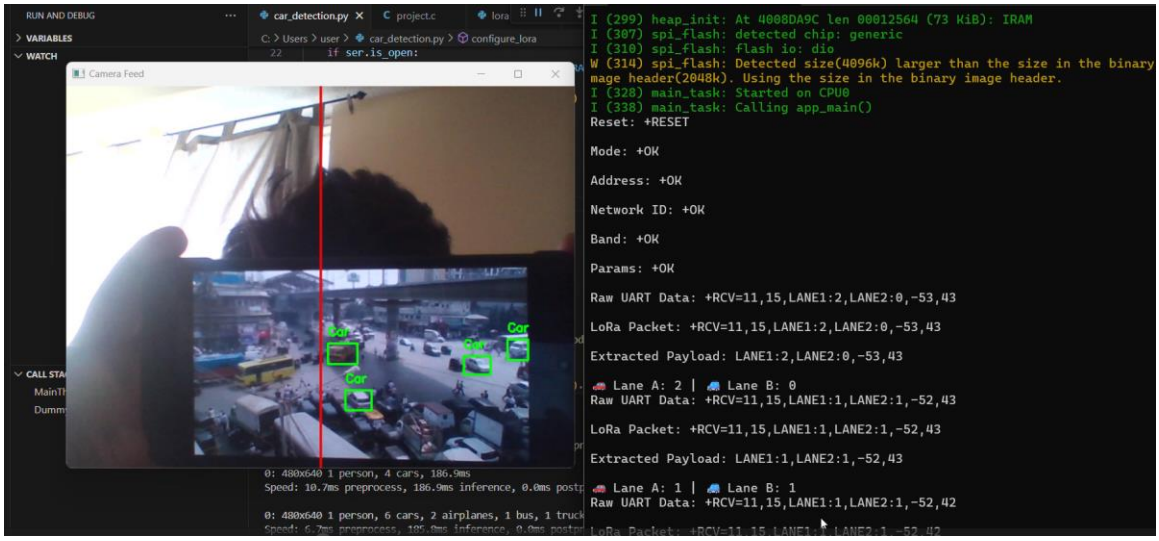
5.2 Dynamic Traffic Light Logic

A signal management task runs a logic loop that:

- Gives green to the lane with more cars.
- If cars <= 5: green for 20 seconds; if > 5: green for 30 seconds.
- 5 seconds before current green ends, it checks the other lane's latest count.
- Switches signal accordingly, with updated duration.

This emulation is printed in the terminal using emojis to indicate RED and GREEN signals for each lane.

Output:



5.3 C code at ESP 32 end:

```
#include <stdint.h>

#include <stdio.h>

#include <string.h>


#include "driver/gpio.h"

#include "driver/uart.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"


#define TXD_PIN GPIO_NUM_19

#define RXD_PIN GPIO_NUM_18

#define UART_NUM UART_NUM_2

#define BUF_SIZE 512


// Global traffic data

volatile int lane1_count = 0;

volatile int lane2_count = 0;


// Function to send an AT command and get the response

size_t command(const char* cmd, char* response) {

   char tmp[BUF_SIZE];

   snprintf(tmp, sizeof(tmp), "%s\r\n", cmd);

   uart_write_bytes(UART_NUM, tmp, strlen(tmp));

   size_t len = uart_read_bytes(UART_NUM, response, (BUF_SIZE - 1), 500 /
portTICK_PERIOD_MS);
```

```
    if (len > 0) response[len] = 0;
    return len;
}

// Emulated red/green signal display
void display_signals(char green_lane) {
    if (green_lane == '1') {
        printf("\n🔴 Lane 2 RED\n🟢 Lane 1 GREEN\n");
    } else {
        printf("\n🔴 Lane 1 RED\n🟢 Lane 2 GREEN\n");
    }
}

// Task to receive UART LoRa data continuously
void uart_receive_task(void *arg) {
    char data[BUF_SIZE];
    while (1) {
        int len = uart_read_bytes(UART_NUM, data, BUF_SIZE - 1, 1000 / portTICK_PERIOD_MS);
        if (len > 0) {
            data[len] = 0;
            char* rcv_start = strstr(data, "+RCV=");
            if (rcv_start) {
                char* payload = strchr(rcv_start, ',');
                if (payload) payload = strchr(payload + 1, ',');
                if (payload) {
```

```
            payload++;
            int l1 = 0, l2 = 0;
            if (sscanf(payload, "LANE1:%d,LANE2:%d", &l1, &l2) == 2) {
              lane1_count = l1;
              lane2_count = l2;
              printf("🚦 Updated Counts -> 🚗 Lane A: %d | 🚙 Lane B: %d\n", lane1_count,
lane2_count);
            }
          }
        }
      }
      vTaskDelay(100 / portTICK_PERIOD_MS);
    }
}

// Main signal logic
void handle_signal_logic() {
  char active_lane;
  int green_time;

  if (lane1_count > lane2_count) {
    active_lane = '1';
    green_time = (lane1_count <= 5) ? 20 : 30;
  } else {
    active_lane = '2';
    green_time = (lane2_count <= 5) ? 20 : 30;
```

```
}

display_signals(active_lane);
printf(" ⏱ Lane %c GREEN for %d seconds\n", active_lane, green_time);

for (int sec = green_time; sec > 0; sec--) {
   printf("Lane %c: %d sec left | 🚗 Lane A: %d | 🚌 Lane B: %d\n",
         active_lane, sec, lane1_count, lane2_count);

   if (sec == 5) {
      char next_lane = (active_lane == '1') ? '2' : '1';
      int other_lane_count = (next_lane == '1') ? lane1_count : lane2_count;
      int next_time = (other_lane_count <= 5) ? 20 : 30;

      printf("⏭ Next Lane %c will get GREEN for %d seconds after this\n",
           next_lane, next_time);

      vTaskDelay(5000 / portTICK_PERIOD_MS); // wait final 5 sec

      display_signals(next_lane);
      printf(" 🚦 Switched to Lane %c GREEN for %d seconds\n", next_lane, next_time);

      for (int s = next_time; s > 0; s--) {
         printf("Lane %c: %d sec left | 🚗 Lane A: %d | 🚌 Lane B: %d\n",
              next_lane, s, lane1_count, lane2_count);
```

```
            vTaskDelay(1000 / portTICK_PERIOD_MS);
        }

        return;
      }

      vTaskDelay(1000 / portTICK_PERIOD_MS);
   }
}

void app_main() {
   const uart_config_t uart_config = {
      .baud_rate = 115200,
      .data_bits = UART_DATA_8_BITS,
      .parity    = UART_PARITY_DISABLE,
      .stop_bits = UART_STOP_BITS_1,
      .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
      .source_clk = UART_SCLK_DEFAULT,
   };

   uart_driver_install(UART_NUM, BUF_SIZE * 2, 0, 0, NULL, 0);
   uart_param_config(UART_NUM, &uart_config);
   uart_set_pin(UART_NUM, TXD_PIN, RXD_PIN, UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE);

   char response[BUF_SIZE];
```

```
  if (command("AT+RESET", response))      printf("Reset: %s\n", response);
  if (command("AT+MODE=0", response))     printf("Mode: %s\n", response);
  if (command("AT+ADDRESS=10", response)) printf("Address: %s\n", response);
  if (command("AT+NETWORKID=6", response))printf("Network ID: %s\n", response);
  if (command("AT+BAND=433000000", response)) printf("Band: %s\n", response);
  if (command("AT+PARAMETER=12,7,1,4", response)) printf("Params: %s\n", response);

  uart_flush_input(UART_NUM);
  vTaskDelay(2000 / portTICK_PERIOD_MS);

  // Create UART listener task
  xTaskCreate(uart_receive_task, "uart_receive_task", 4096, NULL, 10, NULL);

  // Main loop: keep cycling through logic every cycle
  while (1) {
    handle_signal_logic();
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
```

5. Conclusion & Next Steps

The project successfully implements a real-time Smart Traffic Management System prototype using:

- YOLOv8-based car detection
- Lane-based classification
- LoRa communication from laptop to ESP32

- Dynamic traffic logic on ESP32 with red/green light control simulation

This system can be extended by adding multiple intersections, and logging traffic data. The modular design makes it suitable for future enhancements like MQTT integration, cloud dashboards, or edge AI models.

.

6. Reference

Roboflow - Dataset for training YOLOv8 model: https://roboflow.com/

Ultralytics YOLOv8 - Model training and object detection: https://docs.ultralytics.com/

LoRa Communication Guide - Reference for LoRa setup: https://randomnerdtutorials.com/

LoRa Guide (Random Nerd Tutorials):

https://randomnerdtutorials.com/

ESP-IDF UART Driver:

https://docs.espressif.com/

FreeRTOS Task Management:

https://www.freertos.org/