CONGEST AI

A SUMMER PROJECT REPORT

Submitted by

RUBESH.K (2022105702)

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY, CHENNAI 600025

JULY 2024

ANNA UNIVERSITY

CHENNAI-600 025

BONAFIDE CERTIFICATE

Certified that this project report titled **CONGEST AI** is the bonafide work of **RUBESH.K (2022105702)** of 5th Semester who carried out the project work for **EC5512-Project 1,** in the duration June 2024 – July 2024 under my supervision. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. M.A. BHAGYAVENI
HEAD OF THE DEPARTMENT
Professor
Department of Electronics and
Communication Engineering
College of Engineering, Guindy
Anna University
Chennai-600 025

SIGNATURE

Dr. T. SRIDARSHINI
SUPERVISOR
Assistant Professor
Department of Electronics and
Communication Engineering
College of Engineering, Guindy
Anna University
Chennai – 600 025

ACKNOWLEDGEMENT

First and foremost, we thank God for an abundance of grace and has countless blessings in making this work a great success.

We also express our sincere gratitude to our Head of the Department, **Dr. M.A. BHAGYAVENI**, Professor, Department of Electronics and Communication Engineering for her enthusiastic encouragement and support throughout the project.

We present our sincere thanks and gratitude to our project supervisor, **Dr. T. SRIDARSHINI**, Assistant Professor, Department of Electronics and Communication Engineering for her wholehearted support, patience, valuable guidance, technical expertise and encouragement in our project.

We thank all the teaching and non-teaching staff of Department of Electronics and Communication Engineering, for their kind help and co-operation during the course of our project.

TABLE OF CONTENTS:

ABSTRACT

Congest AI aims to develop an advanced traffic signal time control system using deep learning techniques to reduce traffic congestion awaiting times at traffic signals. Traditional traffic signal systems operate on fixed time schedules, which are often inefficient in managing dynamic traffic conditions. This project proposes an innovative approach to traffic signal control by employing deep learning methods to adapt signal timings based on real-time vehicular flow.

The system leverages Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to process video feeds from traffic cameras. These networks are used to detect and count vehicles at intersections, providing real-time data on traffic conditions. By utilizing these advanced neural networks, the system can accurately assess traffic density and flow, which is critical for making informed decisions on signal timings.

To achieve optimal traffic flow efficiency, the system integrates these deep learning models with a reinforcement learning algorithm. This combination allows the system to continuously learn and optimize traffic signal timings. As a result, Congest AI can dynamically adjust to changing traffic patterns, minimizing waiting times and improving overall traffic flow efficiency.

INTRODUCTION:

Roadway transportation is the most significant transportation in the world contributing social and economic development to a country. Developing countries like India have major industrial and densely populated cities. To commute to any place, roads are the primary way of transportation. Due to increased population and industry-rich cities, the traffic congestion has been a major problem seen in developing urbans and cities. For convenient controlling of traffic at multiple junctions of roads, the traffic signal lights are implemented. Conventional traffic lights run at pre-defined time periods at different times of a day. Though it ensures smoother traffic flow, the waiting time at the signal junction is increased due to the growth in number of vehicles used today. Hence, the conventional traffic controlling may fail at some circumstances. Nowadays, the traffic police remotely control the traffic signal at the junction, which can become tedious for them especially during peak-hours.

By leveraging advanced technology such as machine learning and deep learning, the flow of vehicles at a traffic can be controlled seamlessly. Fetching vehicle density data in real-time, Congest AI can dynamically assign traffic signal timings at a junction to reduce waiting times significantly. In the event of global warming, majority of its causes are from vehicle combustion exhaust. Hence, by reducing waiting times for the vehicle, we can save fuel consumption to some extent. Implementing Deep learning models in traffic control would solve traffic congestion effectively in busy cities without human intervention.

STRUCTURE:

1. **Data collection:**

   Traffic video data is collected from cameras installed at various intersections. This data serves as the foundation for detecting and analyzing traffic patterns.

2. **Vehicle detection and classification:**

   Using Convolutional Neural Networks [1] (CNNs), the system processes the preprocessed video frames to detect vehicles and classify them based on type (e.g., cars, trucks, motorcycles). This classification is crucial for accurately understanding traffic density and flow.[2]

3. **Decision making:**

   Based on the number of detected vehicles, the model controls the time period of the traffic signal. The system dynamically adjusts signal timings to optimize traffic flow and reduce waiting times at intersections.

COMPONENTS USED:

Breadboard:

A breadboard is used to create a temporary prototype of the traffic light control circuit. It allows for easy connections and modifications without the need for soldering.

Connecting wires:

Connecting wires are essential for establishing electrical connections between the various components of the project. They are used to connect the Raspberry Pi, the traffic light module, and other peripheral devices.

LED Traffic light signal module:

The traffic light module consists of red, green, and yellow LEDs, which simulate a real traffic light. This module is controlled by the Raspberry Pi to demonstrate the adaptive signal timing based on vehicle count.

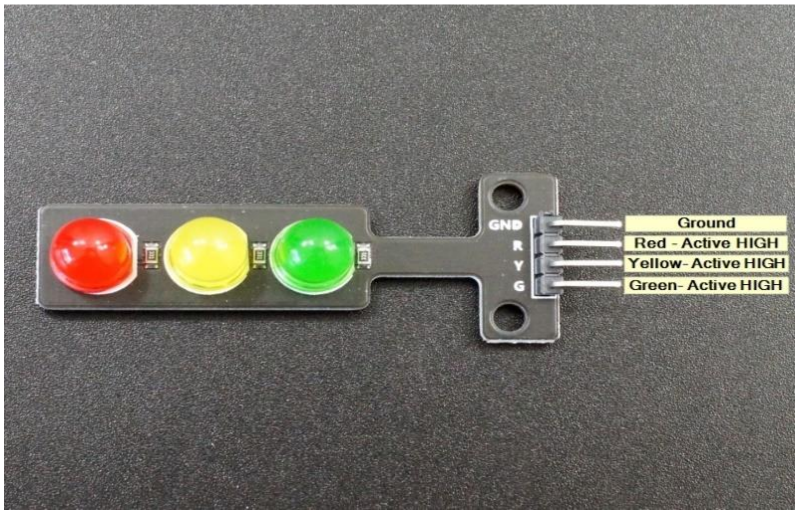


Fig 1: LED Traffic Lights Signal Module

Raspberry pi 4b+:

The Raspberry Pi 4b+ is the central processing unit for the project. It runs the vehicle detection and signal scheduling algorithms, processes video feeds, and controls the traffic light module.

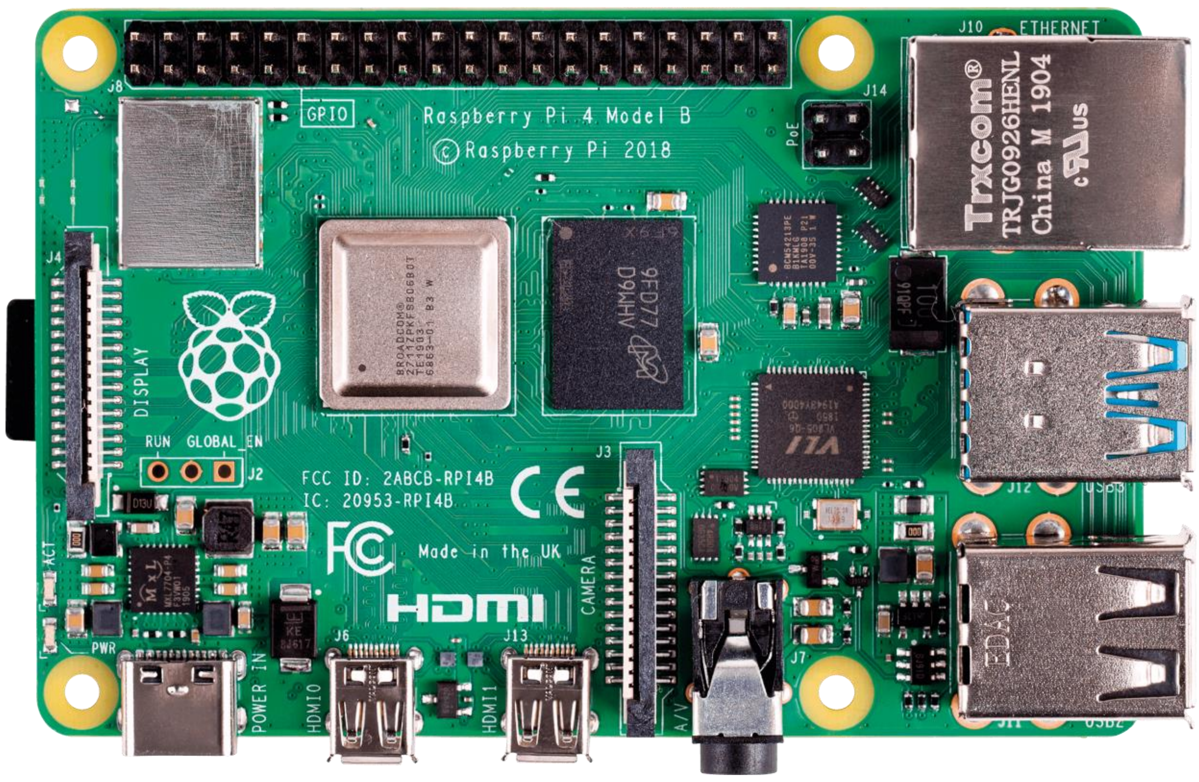


Fig 2: Raspberry pi 4b+

Micro-HDMI to HDMI cable:

A micro-HDMI to HDMI cable is used to connect the Raspberry Pi to a monitor. This allows for display output, making it easier to monitor the system's operation and debug if necessary.

Monitor:

A monitor is used to display the Raspberry Pi's output. It is crucial for visualizing the video feeds from traffic cameras and observing the real-time adjustments in signal timings.

Keyboard:

A keyboard is required for inputting commands and interacting with the Raspberry Pi's operating system. It is essential for setting up the software environment and running the algorithms.

Mouse:

A mouse is used alongside the keyboard to navigate the Raspberry Pi’s interface. It aids in the setup and configuration of the system.

5v DC Adapter (Power Supply for Raspberry Pi)

The 5v DC adapter provides the necessary power supply for the Raspberry Pi 4b+. A stable and sufficient power source is critical for the reliable operation of the Raspberry Pi and its connected components.

64 GB SD card:

A 64 GB SD card is used to store the operating system, software, and data for the Raspberry Pi. It provides ample storage for the video feeds, algorithms, and other project files.

IMPLEMENTATION:

1)Vehicle Detection and Counting Using Deep Learning:

To detect and count vehicles, we utilize the YOLO (You Only Look Once) pre-trained model. YOLO is a state-of-the-art, real-time object detection system renowned for its high accuracy and speed [3]. This model is particularly effective for our application as it can quickly and reliably identify various objects within video frames.

The YOLO model we employ is trained with the COCO (Common Objects in Context) dataset. The COCO dataset is a large-scale object detection, segmentation, and captioning dataset that contains over 200,000 labelled images[4]. By training YOLO on this comprehensive dataset, we ensure that the model can accurately recognize a wide range of vehicle types and other relevant objects in traffic scenes.

By applying the YOLO model to video feeds from traffic cameras, we can accurately detect and count the number of vehicles at intersections. This real-time data is crucial for our adaptive traffic signal control system, enabling it to dynamically adjust signal timings based on the current traffic conditions and optimize the flow of vehicles through intersections[5].

2)Calculating Signal Timings Based on Vehicle Density:

To determine the signal timings based on vehicle density, a Python function has been developed. This function is designed to dynamically adjust traffic signal durations to optimize traffic flow at intersections. By leveraging real-time traffic data, the function can make informed decisions on how long each signal should remain green, yellow, or red.[5]

The Python function utilizes nested for loops to calculate the appropriate signal timings according to the vehicle count. These loops iterate through the collected traffic data, assessing the number of vehicles present at each intersection. Based on this assessment, the function computes the optimal signal duration to minimize waiting times and prevent congestion.

By dynamically adjusting the signal durations based on real-time traffic data, the system aims to enhance traffic flow efficiency. This adaptive approach ensures that signal timings are responsive to the current traffic conditions, reducing overall waiting times for vehicles and improving the overall effectiveness of the traffic management system.

3)Integration with Raspberry Pi and LED Lights:

The vehicle detection algorithm and the signal schedule algorithm are integrated using a Raspberry Pi 4b+. This compact and powerful device serves as the central processing unit for the system, seamlessly coordinating the various components involved. Traffic video samples are fed into the algorithm, which processes the data to detect and count vehicles at intersections.[6]

Based on the calculated vehicle count, the algorithm assigns specific signal timings. These timings are dynamically adjusted to optimize traffic flow, ensuring that the duration of green, yellow, and red lights is appropriate for the current traffic conditions. The goal is to minimize congestion and reduce waiting times for vehicles at each intersection.

The assigned signal timings are then sent to a traffic module equipped with red, green, and yellow LED lights. Each LED is turned on for the duration specified by the algorithm, effectively controlling the traffic signals. This integration of vehicle detection, signal scheduling, and real-time adjustment ensures efficient traffic management and improved flow at intersections.[7]

IMAGES:

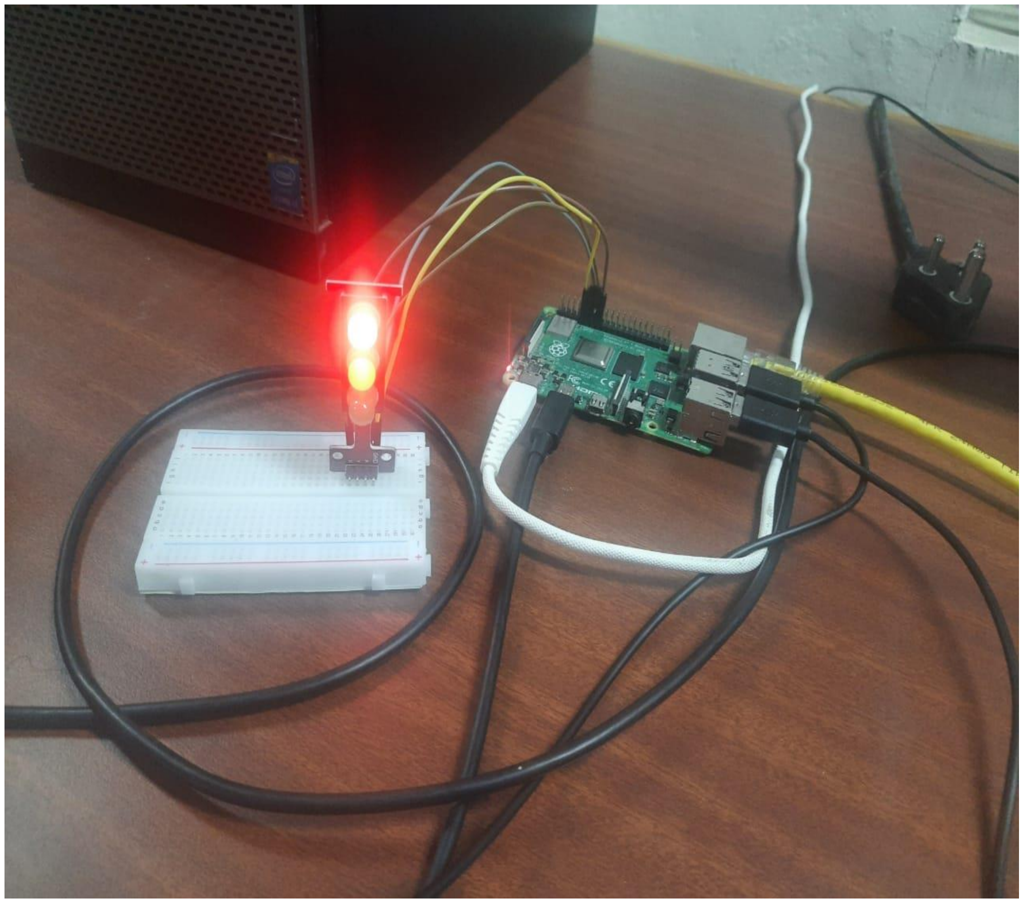


Fig 3: Project setup showcasing traffic light module connected with Raspberry pi 4+

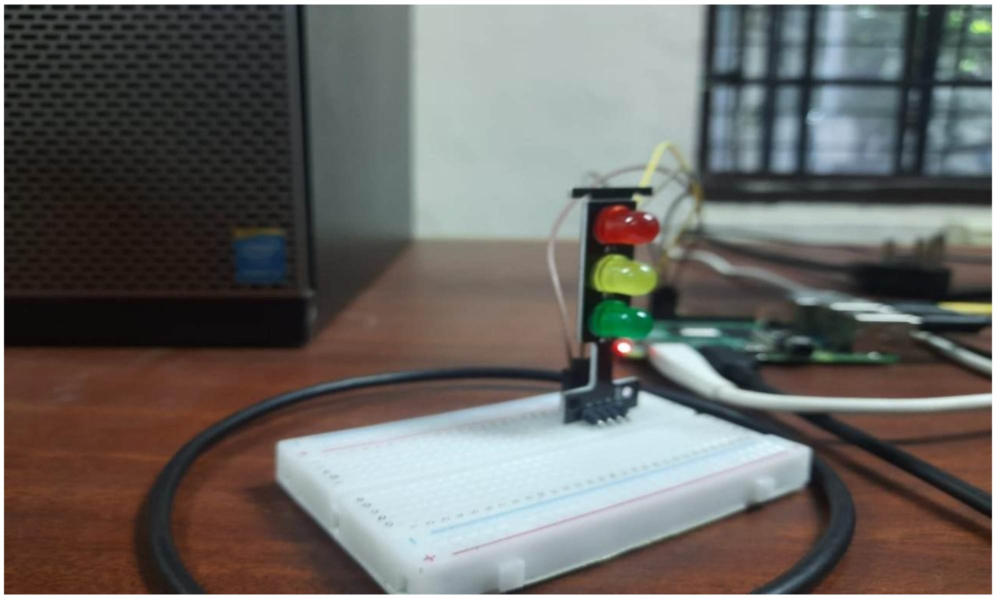


Fig 4: Traffic light module mounted on breadboard

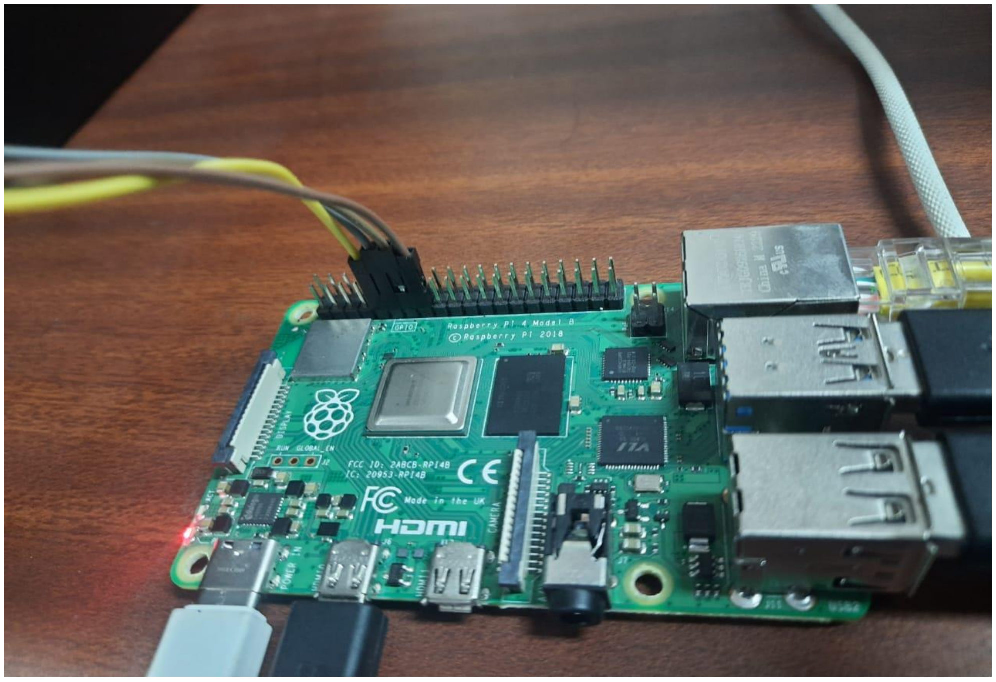


Fig 5: Project setup showcasing Raspberry pi 4+

ISSUED FACED:

Python Library Compatibility:

One of the primary challenges encountered during the project was the compatibility of Python library files with the latest version of Python. Many of the libraries required for the vehicle detection and signal scheduling algorithms had dependencies that were not fully supported in the newer Python versions. This incompatibility led to significant delays as we had to troubleshoot errors and, in some cases, revert to older versions of Python or seek alternative libraries that offered similar functionality.

Raspberry Pi Integration:

Another significant issue arose during the integration of the vehicle detection algorithm and the signal scheduling algorithm with the Raspberry Pi 4b+. Compatibility issues were noted, particularly related to the GPIO (General-Purpose Input/Output) pins and the libraries required to control the traffic light module. These issues required extensive debugging and adjustments to both the hardware connections and the software code to ensure seamless communication between the Raspberry Pi and the traffic light module.

Resolution and Adaptation:

To address these issues, we undertook several steps. For the Python library compatibility problem, we either downgraded to an older, more compatible version of Python or identified and implemented alternative libraries. For the Raspberry Pi integration challenges, we conducted thorough testing and debugging, modifying the code and hardware setup as needed. These adaptations were crucial in ensuring that the project could proceed despite the initial setbacks, ultimately leading to a successful implementation of the adaptive traffic signal control system.

FUTURE IMPLEMENTATION:

Enhanced Data Collection and Analysis:

Future implementations of Congest AI can benefit from enhanced data collection and analysis techniques. By incorporating additional data sources such as GPS data from vehicles, weather conditions, and road work updates, the system can gain a more comprehensive understanding of traffic patterns. Machine learning models can be further trained with these diverse datasets to improve their accuracy and robustness. Additionally, deploying more advanced sensors and high-resolution cameras at intersections can provide more precise and detailed traffic data, enabling better decision-making.[7]

Integration with Smart City Infrastructure:

As cities move towards smarter infrastructure, integrating Congest AI with other smart city systems can significantly enhance its effectiveness. For example, linking the traffic signal control system with public transportation schedules, emergency response routes, and pedestrian crossing signals can create a more holistic approach to traffic management. This integration would allow for coordinated traffic flows, reducing congestion and improving overall urban mobility. Furthermore, the system can be scaled to cover larger areas, incorporating multiple intersections and traffic corridors for a city-wide adaptive traffic control network.

REAL TIME APPLICATION:

To implement Congest AI in a real-time application, the first step involves setting up an infrastructure that includes high-resolution traffic cameras at key intersections to capture continuous video feeds. These cameras should be strategically placed to cover all lanes and directions of traffic. The video feeds will then be transmitted to a central processing unit, which could be a powerful server or a distributed network of edge devices. The Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) [8] will run on this hardware, processing the video feeds to detect and count vehicles in real-time. The detected vehicle data will be used to assess traffic density and flow, forming the basis for adaptive traffic signal control.

The next step involves integrating the processed data with a reinforcement learning model[8] for adjusting traffic signal timings. According to the number of vehicle count, it compares with the other lanes and give the optimise signal timing. The adjusted signal timings will be communicated to traffic light controllers at the intersections. Additionally, the system should include a feedback mechanism to monitor traffic conditions after signal adjustments, allowing for continuous improvement of the model's performance. This real-time implementation ensures that Congest AI can effectively manage dynamic traffic conditions, enhancing overall traffic efficiency and reducing congestion in urban areas.[9]

PROGRAMMING CODES:

The following code is used to detects vehicles, take it's count, get signal timings accordingly, and controls the LED timing.

```
import cv2
import math
import numpy as np
import cvzone
from ultralytics import YOLO
import time
import RPi.GPIO as GPIO

# Initialize GPIO
GPIO.setwarnings(False)  # Disable GPIO warnings
GPIO.setmode(GPIO.BCM)
RED_LED = 17
YELLOW_LED = 27
GREEN_LED = 22

GPIO.setup(RED_LED, GPIO.OUT)
GPIO.setup(YELLOW_LED, GPIO.OUT)
GPIO.setup(GREEN_LED, GPIO.OUT)
# Initialize video capture and model
cap = cv2.VideoCapture('output_video.mp4')
model = YOLO('yolov8n.pt')
classnames = []
with open('classes.txt', 'r') as f:
    classnames = f.read().splitlines()

# Function to calculate signal timings
def calc_timings(density):
    if 0 < density <= 20:
        green_time = 20
        yellow_time = 3
        red_time = 80 - green_time - yellow_time
    elif 21 <= density <= 40:
        green_time = 30
        yellow_time = 3
        red_time = 80 - green_time - yellow_time
    elif 41 <= density <= 60:
        green_time = 40
        yellow_time = 4
        red_time = 80 - green_time - yellow_time
    elif 61 <= density <= 80:
        green_time = 60
        yellow_time = 4
        red_time = 90 - green_time - yellow_time
    elif 81 <= density <= 100:
        green_time = 65
        yellow_time = 5
        red_time = 90 - green_time - yellow_time
    elif density > 100:
        green_time = 70
        yellow_time = 5
        red_time = 90 - green_time - yellow_time
    else:
        green_time=0
        red_time=90
        yellow_time=0
    return green_time, yellow_time, red_time

# Initialize variables
total_vehicles_set = set()
vehicle_counts = []
sample_interval = 5  # seconds
total_samples = 6  # total number of samples needed for 30 seconds
desired_fps = 10  # Adjust this value as needed
frame_time = 1 / desired_fps

# Process the video
start_time = time.time()
while len(vehicle_counts) < total_samples:
    ret, frame = cap.read()
    if not ret:
        break

    frame_start_time = time.time()
    frame_height, frame_width = frame.shape[:2]
    middle_y = frame_height // 2
    line = [0, middle_y, frame_width, middle_y]  # Middle line across the frame

    detections = np.empty((0, 6))  # Updated to 6 columns
    result = model(frame, stream=1)
    for info in result:
        boxes = info.boxes
        for box in boxes:
            x1, y1, x2, y2 = box.xyxy[0]
            conf = box.conf[0]
            classindex = box.cls[0]
            conf = math.ceil(conf * 100)
            classindex = int(classindex)
            objectdetect = classnames[classindex]
            if objectdetect in ['car', 'bus', 'truck', 'motorcycle'] and conf > 49:
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                new_detections = np.array([x1, y1, x2, y2, conf, classindex])  # Added
classindex or some other value
                detections = np.vstack((detections, new_detections))

    cv2.line(frame, (line[0], line[1]), (line[2], line[3]), (0, 255, 255), 6)
    for i, detection in enumerate(detections):
        x1, y1, x2, y2, conf, classindex = detection
        x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
        w, h = x2 - x1, y2 - y1
        cx, cy = x1 + w // 2, y1 + h // 2
        cv2.circle(frame, (cx, cy), 6, (0, 255, 255), -1)
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cvzone.putTextRect(frame, f'{i}', [x1 + 8, y1 - 12], thickness=2, scale=1)

        total_vehicles_set.add(i)

    # Display the total number of unique vehicles
    cvzone.putTextRect(frame, f'Total Vehicles: {len(total_vehicles_set)}', [50, 50],
thickness=2, scale=1, colorR=(0, 255, 0))

    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    # Check if it's time to sample the vehicle count
    elapsed_time = time.time() - start_time
    if elapsed_time >= sample_interval * (len(vehicle_counts) + 1):
        vehicle_counts.append(len(total_vehicles_set))
        total_vehicles_set.clear()  # Reset for the next sample

    # Calculate time taken for processing the frame
    frame_end_time = time.time()
    processing_time = frame_end_time - frame_start_time
    # Sleep to maintain desired FPS
    if processing_time < frame_time:
        time.sleep(frame_time - processing_time)

cap.release()
cv2.destroyAllWindows()

# Calculate total vehicle count
if vehicle_counts:
    total_count = sum(vehicle_counts)

    # Get signal timings based on average vehicle count
    green_time, yellow_time, red_time = calc_timings(total_count)

    print(f"Green Time: {green_time} seconds")
    print(f"Yellow Time: {yellow_time} seconds")
    print(f"Red Time: {red_time} seconds")

    # Control the LEDs
    cycle_count = 0
    max_cycles = 1
    try:
        print("Starting LED control")
        while cycle_count < max_cycles:
            if average_count == 0:
                print("RED LED ON")
                GPIO.output(RED_LED, GPIO.HIGH)
                for _ in range(red_time):  # Blink yellow LED twice
                    GPIO.output(YELLOW_LED, GPIO.HIGH)
                    time.sleep(0.5)
                    GPIO.output(YELLOW_LED, GPIO.LOW)
                    time.sleep(0.5)

                cycle_count += 1  # Increment cycle count
            else:
                print("RED LED ON")
                GPIO.output(RED_LED, GPIO.HIGH)
                GPIO.output(GREEN_LED,GPIO.LOW)
                GPIO.output(YELLOW_LED,GPIO.LOW)
                time.sleep(red_time)

                print("YELLOW LED ON")
                GPIO.output(GREEN_LED, GPIO.LOW)
                GPIO.output(RED_LED, GPIO.LOW)
                GPIO.output(YELLOW_LED, GPIO.HIGH)
                time.sleep(yellow_time)

                print("GREEN LED ON")
                GPIO.output(RED_LED, GPIO.LOW)
                GPIO.output(YELLOW_LED, GPIO.LOW)
                GPIO.output(GREEN_LED, GPIO.HIGH)
                time.sleep(green_time)

                cycle_count += 1  # Increment cycle count

    except KeyboardInterrupt:
        pass
    finally:
        GPIO.cleanup()
        print("GPIO cleanup done")

else:
    print("No vehicle data collected.")
    print("RED LED ON")
    GPIO.output(RED_LED, GPIO.HIGH)
    GPIO.output(YELLOW_LED, GPIO.LOW)
    GPIO.output(GREEN_LED,GPIO.LOW)
```

CONCLUSION:

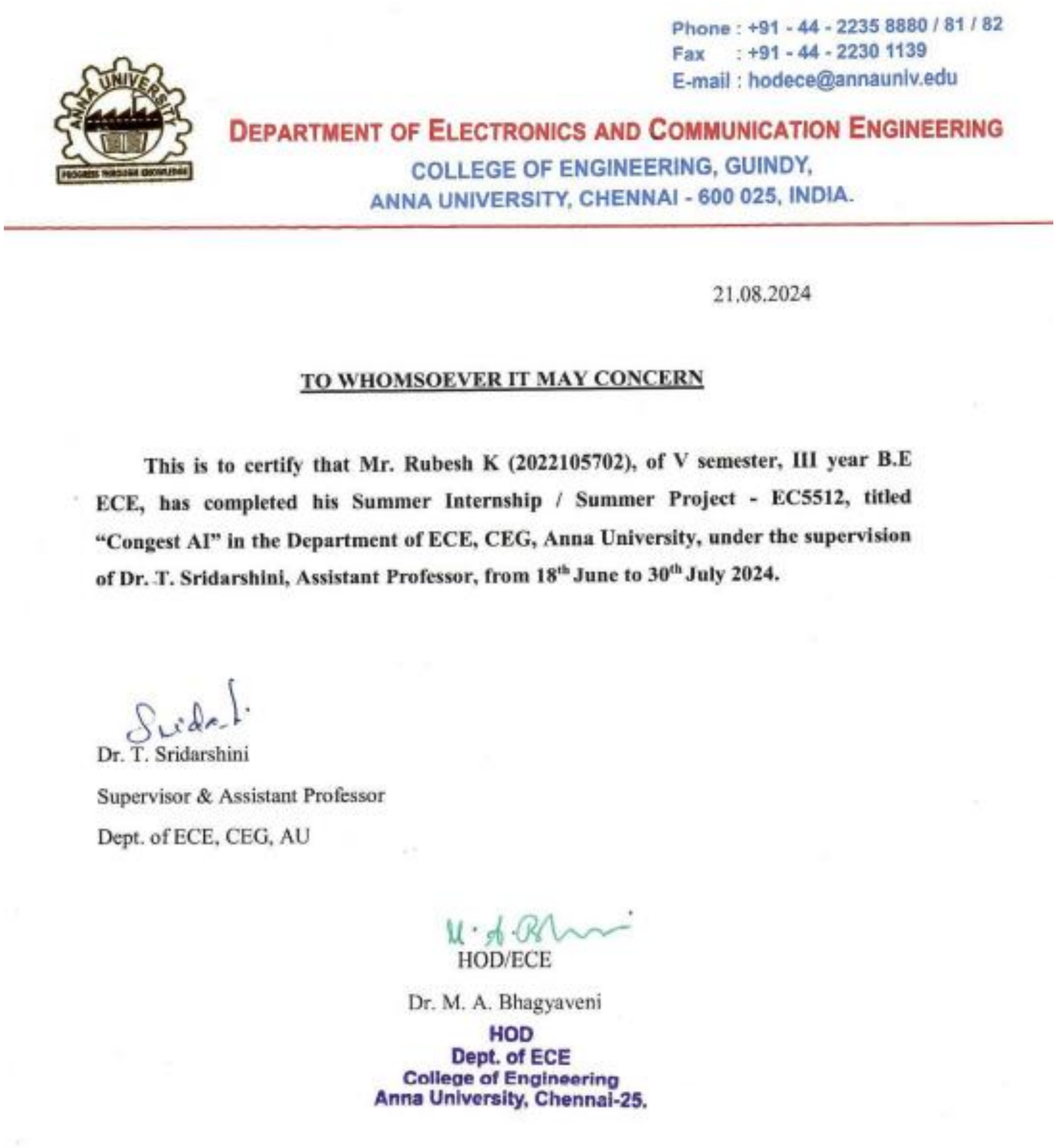
Congest AI represents a significant advancement in traffic signal control systems by utilizing deep learning techniques to reduce traffic congestion and waiting times at intersections. Unlike traditional traffic signal systems that operate on fixed schedules and often fail to adapt to dynamic traffic conditions, Congest AI employs an innovative approach that adapts signal timings based on real-time vehicular flow.

The system leverages Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to process video feeds from traffic cameras, detecting and counting vehicles to provide accurate, real-time data on traffic conditions. By integrating these advanced neural networks with a reinforcement learning algorithm, Congest AI can continuously learn and optimize traffic signal timings.

As a result, Congest AI can dynamically adjust to changing traffic patterns, significantly minimizing waiting times and improving overall traffic flow efficiency. This adaptive approach ensures that traffic signal control is more responsive and effective, leading to smoother and more efficient traffic management in urban environments.

REFERENCES:

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444
2. Chen, L., Englund, C., & Bajcsy, R. (2017). Efficient dynamic traffic control with Adaptive Traffic Signal Control system—A case study in the city of Malmö. In Transportation Research Procedia (Vol. 22, pp. 170-181).
3. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 779-788).
4. COCO Dataset: https://docs.ultralytics.com/datasets/detect/coco/
5. A. Kanungo, A. Sharma and C. Singla, "Smart Traffic Lights Switching and Traffic Density Calculation using Video Processing", *Recent Advances in Engineering and Computational Sciences (RAECS)*, 2014.
6. B. Siripatana, K. Nopchanasuphap and S. Chuai-Aree, "Intelligent Traffic Light System Using Image Processing," *2021 2nd SEA-STEM International Conference (SEA-STEM)*, Hat Yai, Thailand, 2021,
7. Mimbela, L. E. Y., & Klein, L. A. (2007). A Summary of Vehicle Detection and Surveillance Technologies used in Intelligent Transportation Systems. The Vehicle Detector Clearinghouse.
8. N. Kodama, T. Harada and K. Miyazaki, "Traffic Signal Control System Using Deep Reinforcement Learning With Emphasis on Reinforcing Successful Experiences," in *IEEE Access*, vol. 10
9. Arel, I., Liu, C., Urbanik, T., & Kohls, A. G. (2010). Reinforcement Learning-based Multi-agent System for Network Traffic Signal Control. IET Intelligent Transport Systems, 4(2), 128-135.



Phone : +91 - 44 - 2235 8880 / 81 / 82
Fax : +91 - 44 - 2230 1139
E-mail : hodece@annauniv.edu

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
COLLEGE OF ENGINEERING, GUINDY,
ANNA UNIVERSITY, CHENNAI - 600 025, INDIA.

21.08.2024

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Mr. Rubesh K (2022105702), of V semester, III year B.E ECE, has completed his Summer Internship / Summer Project - EC5512, titled "Congest AI" in the Department of ECE, CEG, Anna University, under the supervision of Dr. T. Sridarshini, Assistant Professor, from 18th June to 30th July 2024.

[illegible]
Dr. T. Sridarshini
Supervisor & Assistant Professor
Dept. of ECE, CEG, AU

[illegible]
HOD/ECE
Dr. M. A. Bhagyaveni

HOD
Dept. of ECE
College of Engineering
Anna University, Chennai-25.