INTELLIGENT TRAFFIC SYSTEM FOR URBAN CONDITION USING REAL-TIME VEHICLE DETECTION

Submitted in partial fulfillment of requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

Submitted by

Bonthu Tulasi (21761A1210)

Prathipati Prasanna(21761A1246)

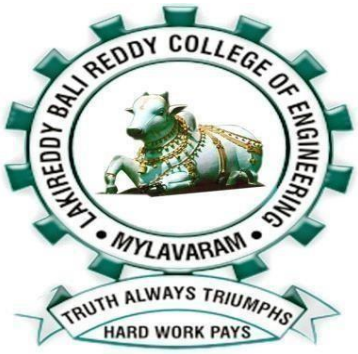
Orugu Jagadeesh (21761A1241)

Under the Esteemed Guidance of

Dr. B. Srinivasa Rao

Professor & HOD

Dept of IT, LBRCE



DEPARTMENT OF INFORMATION TECHNOLOGY

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

L. B. REDDY NAGAR, MYLAVARAM, NTR (District) - 521230 Affiliated to JNTUK Kakinada & Approved by AICTE, New Delhi, NAAC Accredited with "A" grade, Accredited by NBA Tier-I(IT) &An ISO 21001:2018,14001:2015,50001:2018 Certified Institution

2024-2025

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to JNTUK Kakinada & Approved by AICTE, New Delhi, NAAC Accredited with "A" grade, Accredited by NBA Tier-I(IT) &An ISO 21001:2018,14001:2015,50001:2018 Certified Institution

L.B. REDDY NAGAR, MYLAVARAM, NTR Dist.

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that **B. Tulasi (21761A1210), P. Prasanna(21761A1246), O. Jagadeesh (21761A1241)** of IV B. Tech **(Information Technology)** have successfully completed the project titled "INTELLIGENT TRAFFIC SYSTEM FOR URBAN CONDITION USING REAL-TIME VEHICLE DETECTION" at Lakireddy Bali Reddy College of Engineering during the Academic Year 2024-2025. This project report is submitted as partial fulfillment for the award of degree **B. Tech (Information Technology).**

Project Guide		Head of Department
Dr. B. Srinivasa Rao		**Dr. B. Srinivasa Rao**
Professor & HOD		Professor & HOD
Department of IT	External Examiner	Department of IT

ACKNOWLEDGEMENT

Behind every achievement lies an unfathomable sea of gratitude to those who activated it, without whom it would ever have been in existence. To them we lay the words of gratitude imprinted with us.

We express our special gratitude to **Dr. K. AppaRao, Principal of Lakireddy Bali Reddy College of Engineering,** who made this endeavor possible. We would also thank all the teaching and non- teaching staff members of Information Technology department who have extended their full cooperation.

We highly oblige to **Dr. B. Srinivasa Rao, HOD of Information Technology** for his support and encouragement.

It is with utmost pleasure that we avail this opportunity to express heart full gratitude to our beloved guide **Dr. B. Srinivasa Rao, Professor & HOD of Information Technology** for his mastery supervision through all the phases of our main project work with valuable suggestions and support. His friendly and informal talks helped us to work under excellent working conditions.

We are highly thankful to our project coordinator **Dr. A.V.N Reddy, Professor of Information Technology,** for providing the knowledge to deal with the project at every phase in a systematic manner.

Finally, a word of gratitude to our **PARENTS** who have be a constant source of encouragement and love.

PROJECT ASSOCIATES

B. Tulasi (21761A1210)

P. Prasanna (21761A1246)

O. Jagadeesh (21761A141)

DECLARATION

We hereby declare that the project report entitled "INTELLIGENT TRAFFIC SYSTEM FOR URBAN CONDITION USING REAL-TIME VEHICLE DETECTION" submitted to JNTUK is partial fulfillment of the requirement for the award of the degree of Bachelor of Technology (B. Tech) is an original work carried out by us. The matter embodied in this project is a genuine work by the student and has not been submitted earlier to this university or any other university for award of any degree or diploma or prize.

PROJECT ASSOCIATES

B.Tulasi (21761A1246)

P.Prasanna (21761A1246)

O.Jagadeesh (21761A141)

INDEX

TABLE OF CONTENTS	PAGE NO
LIST OF FIGURES	I
ACRONYMS AND ABBREVIATIONS	II
ABSTRACT	1
1.INTRODUCTION	3-5
1.1 Introduction	3-4
1.2 Existing System	5
1.3 Proposed System	5
2.LITERATURE SURVEY	6-7
3.SYSTEM SPECIFICATIONS	8-10
3.1 Software Requirements	8-9
3.1.1 Python	8
3.1.2 Numpy	8
3.1.3 OpenCV	8
3.1.4 PySerial	8
3.1.5 Arduino IDE	8
3.1.6 Ultralytics YoloV8	9
3.2 Hardware Requirements	9-10
3.2.1 Camera (USB/Webcam)	9
3.2.2 NEMA 17 Stepper Motor	9

3.2.3 A4988 Stepper Motor Driver 9

3.2.4 Arduino UNO 9

3.2.5 NodeMCU (ESP8266) 10

3.2.6 Seven Segment Display 10

3.2.7 LEDs (Green) 10

3.2.8 Breadboard and Jumper Wires 10

3.2.9 Breadboard Power Adapter 10

3.2.10 12V DC Adapter 10

4. METHODOLOGY 11-12

4.1 Overview of the Project 11

4.2 Explanation of workflow steps 12

4.7 Performance Analysis 12

5. DESIGN VIEW 13-18

5.1 UML Diagrams 13-18

5.1.1 Use Case Diagram 13-14

5.1.2 Class Diagram 15-16

5.1.3 Activity Diagram 17-18

6. IMPLEMENTATION 19-27

7.TESTING 28

8. RESULTS AND DISCUSSIONS 29-32

9.CONCLUSION 33

10.FUTURE ENHANCEMENT 34

11.REFERENCES 35

LIST OF FIGURES

Fig 1: Workflow of proposed system 10

Fig 2: Use-case Diagram 14

Fig 3: Class diagram 16

Fig 4: Activity diagram 18

Fig 5: Performance analysis 26

Fig 6: F1-curve 26

Fig 7: Confusion matrix normalized 27

Fig 8: Labels correlogram 27

Fig 9: Labels 28

Fig 10: Web page 28

Fig 11: Vehicle Detection 28

Fig 12: Vehicle Detection 29

Fig 13: Hardware implementation 29

ACRONYMS AND ABBREVIATIONS

IOT	Internet Of Things
IDE	Integrated Development Environment
LED	Light Emitting Diode
NEMA	National Electric Manufacturers Association
NodeMCU	Node Micro Controller Unit
CNN	Convolutional Neural Networks

ABSTRACT

This project introduces an intelligent traffic management system specifically designed for urban road conditions, employing real-time vehicle tracking through deep learning and Internet of Things (IoT) integration. The proposed system addresses the limitations of traditional traffic control systems, which operate on static timers, often resulting in inefficient traffic flow and prolonged waiting times. By leveraging the YOLOv8 (You Only Look Once version 8) deep learning algorithm for vehicle detection and classification, the system provides dynamic signal timing based on the real-time density of vehicles.

A rotating camera, mounted on a stepper motor and driven by a NEMA17 driver controlled via NodeMCU, captures traffic images from all four directions at an intersection. These images are processed using the YOLOv8 model trained specifically to detect cars. The detected vehicle count determines the green signal duration for each lane, which is displayed using a seven-segment display. LEDs (red, yellow, and green) represent traffic lights and are controlled via an Arduino microcontroller.

This integration of computer vision with embedded systems ensures automated traffic regulation, reduces human intervention, and promotes efficient vehicle movement. The system is implemented using Python for the detection and processing tasks, while the hardware components are programmed using the Arduino IDE. A dataset of 200 images, with 160 used for training and 40 for validation, ensures accurate model performance in real-time urban traffic scenarios. The system demonstrates significant potential in reducing traffic congestion, enhancing safety, and laying the foundation for smart city traffic solutions.

.

CHAPTER – 1

INTELLIGENT TRAFFIC SYSTEM FOR URBAN CONDITION USING REAL-TIME VEHICLE TRACKING

1. INTRODUCTION

1.1 INTRODUCTION:

The increasing number of vehicles in urban areas has led to significant challenges in traffic management, including road congestion, increased fuel consumption, and pollution. Conventional traffic systems operate based on fixed-time intervals without considering the actual traffic density at intersections. As a result, vehicles often remain idle at red signals unnecessarily, leading to inefficiencies and commuter frustration. Addressing this issue requires the adoption of smart, adaptive systems capable of responding to real-time traffic conditions.

This project proposes an intelligent traffic management system that combines **deep learning-based object detection** with **IoT-enabled hardware components** to provide adaptive and efficient signal control. At the core of the system lies the YOLOv8 object detection algorithm, one of the most powerful real-time models for identifying objects in images. A camera mounted on a stepper motor continuously rotates to monitor traffic from all four directions of a road intersection. The images captured are analyzed to detect the number of cars present in each lane using the pre-trained YOLOv8 model.

The hardware system is built using components such as the **NodeMCU (ESP8266)** for wireless control, the **NEMA17 stepper motor and driver** for camera rotation, and **Arduino** for controlling traffic signals represented by LEDs. The duration of the green light is adjusted dynamically based on the vehicle count, and the countdown timer is displayed on a **seven-segment display**, ensuring that traffic from more congested lanes is cleared efficiently before switching to the next direction.

The entire software pipeline is developed in **Python**, with YOLOv8 utilized for detection and OpenCV for image processing. The hardware logic is implemented through the **Arduino IDE**, ensuring seamless communication between software and physical components. By automating traffic management through intelligent vision systems and microcontrollers, this project offers a scalable and cost-effective solution for **smart city infrastructure**, minimizing manual efforts, enhancing traffic flow, and improving urban mobility.

To enhance the system's responsiveness and reliability, a feedback mechanism is incorporated, allowing the central controller to receive and process real-time data from sensors and cameras. This

ensures timely updates and allows for corrective actions in case of anomalies such as emergency vehicles, pedestrian crossings, or lane blockages. With its ability to scale and integrate with smart city platforms, the project lays the foundation for further enhancements such as priority lanes, emergency management systems, and AI-driven traffic forecasting.

.

1.1 EXISTING SYSTEM:

Conventional traffic control systems in most urban areas operate on fixed-time cycles that do not account for real-time traffic density or vehicle flow patterns. These systems follow predefined signal intervals, switching lights at set durations regardless of whether a lane is heavily congested or completely empty. While this approach is simple and cost-effective to implement, it lacks the intelligence needed to manage the dynamic nature of modern urban traffic. As a result, vehicles are often forced to wait unnecessarily at red signals, leading to prolonged idle times, increased fuel consumption, and higher emission levels, all of which contribute to environmental degradation and driver frustration. Moreover, during peak traffic hours or in the event of emergencies, manual intervention is frequently required to override standard timings and manage congestion—adding to operational inefficiency. Traditional systems also lack the infrastructure for data collection and real-time analysis, which are critical for improving long-term traffic planning. Furthermore, their limited adaptability and outdated architecture make them poorly suited for integration with smart city technologies and IoT-based enhancements, thereby restricting their scalability and effectiveness in future-ready urban environments.

1.2 PROPOSED SYSTEM

The proposed intelligent traffic management system overcomes the limitations of fixed-time signal control by integrating deep learning and IoT for real-time traffic regulation. Using the YOLOv8 object detection model, the system detects and counts cars from images captured by a rotating camera mounted on a NEMA17 stepper motor. This setup replaces the need for multiple static cameras and offers a 360° view of the intersection. The Python-based controller processes the images, calculates signal durations based on traffic density, and sends this data to an Arduino or NodeMCU. These microcontrollers operate the traffic LEDs and a seven-segment display that shows countdown timers. By dynamically adjusting green light durations, the system minimizes idle time, improves traffic flow, and reduces fuel consumption. Its modular design and low-cost components make it scalable and suitable for smart city integration, offering a practical solution to growing urban traffic challenges.

CHAPTER-2

2. LITERATURE SURVEY

Over the past decade, numerous approaches have been proposed to improve traffic signal control systems using real-time sensing, image processing, and artificial intelligence. Traditional systems relied on fixed signal timings or basic sensor-based triggers, which lacked adaptability and failed to respond effectively to dynamic traffic conditions. These methods were often inefficient in managing traffic congestion, particularly in urban areas with varying traffic volumes and unpredictable flow. The emergence of computer vision and deep learning has opened new avenues for real-time traffic analysis, with object detection models such as YOLO (You Only Look Once) becoming central to modern intelligent traffic systems.

Early models utilized background subtraction and motion detection using classical image processing techniques in combination with sensors. While these systems could detect the presence of vehicles, they were highly sensitive to lighting changes, weather conditions, and required frequent recalibration. To overcome these challenges, deep learning models—especially Convolutional Neural Networks (CNNs)—have been increasingly applied for traffic surveillance. The YOLO series, starting from YOLOv1 to the latest YOLOv8, have demonstrated strong performance in real-time object detection with high accuracy and speed. Among these, YOLOv8 offers improved architecture and pre-trained models that are well-suited for detecting vehicles in complex scenes, making it ideal for traffic management applications.

Several studies have focused on using YOLO-based detection for estimating traffic density and adapting signal timings accordingly. These systems typically involve a static camera capturing live traffic feed, with the detected vehicle count directly influencing the duration of green lights. Some research extends this by integrating microcontrollers like Arduino or Raspberry Pi to simulate real-world signal control using LEDs and displays. While these setups show significant improvement over conventional systems, most rely on multiple cameras and are often limited by fixed-angle views, which increases hardware complexity and costs.

Other works have explored IoT-enabled traffic systems, using wireless communication modules such as NodeMCU or ESP8266 to remotely control signals and send real-time data to centralized units. This approach has shown promise in reducing human intervention and improving scalability. However, many such systems lack full integration with intelligent decision-making models and operate on predefined rules rather than live data analysis.

In addition, several hybrid systems have been proposed, combining computer vision, hardware control, and data analytics to optimize signal flow at intersections. For example, rotating cameras controlled via stepper motors have been used to scan multiple directions using a single device, reducing cost while covering a larger area. Some studies also integrated seven-segment displays and traffic light simulation using LEDs to create functional prototypes that demonstrate the viability of smart traffic systems in urban environments.

In conclusion, while deep learning and IoT technologies have significantly enhanced the potential of smart traffic management, many existing systems either focus solely on detection without real-time control or require complex and expensive hardware setups. The proposed system addresses these gaps by combining YOLOv8-based vehicle detection with a rotating camera, stepper motor control, Arduino-based signal operation, and wireless communication via NodeMCU. This integrated solution ensures efficient traffic flow management across four-lane intersections while remaining cost-effective, scalable, and suitable for real-world deployment in smart cities.

CHAPTER-3

3. SYSTEM SPECIFICATIONS

3.1 SOFTWARE REQUIREMENTS:

3.1.1 Python:

Python is a high-level, general-purpose programming language widely used for machine learning, deep learning, and automation tasks. In this project, Python was used to implement YOLOv8 for object detection, manage camera input, count vehicles, and communicate with hardware using serial ports. Its rich ecosystem of libraries and simplicity make it ideal for real-time image processing and control tasks**.**

3.1.2 Numpy:

NumPy is a Python library used for handling arrays and performing mathematical operations efficiently. It was used in this project to manipulate image data and perform numerical operations required during detection and timer calculation.

3.1.3 OpenCV:

OpenCV (Open Source Computer Vision Library) is a powerful open-source computer vision library used in Python to process real-time images and video. It was used for reading camera frames, preprocessing input images for the YOLOv8 model, and displaying detection results during testing.

3.1.4 PySerial:

PySerial is a Python library that allows communication between a computer and a serial device such as Arduino or NodeMCU. It was used in this system to send data (vehicle count and timing values) from Python to the Arduino board via the serial port.

3.1.5 Arduino IDE:

The Arduino IDE was used to write and upload code to the Arduino UNO and NodeMCU microcontrollers. It supports C/C++ syntax and provides libraries to control LEDs, stepper motors, and displays, making it essential for the hardware implementation of the traffic control logic.

3.1.6 Ultralytics YOLOv8:

YOLOv8 (You Only Look Once version 8) is a state-of-the-art deep learning model designed for real-time object detection. It was used to detect vehicles (specifically cars) in the images captured by the rotating camera. It provides high accuracy and fast inference, making it suitable for real-time traffic monitoring.

3.2 HARDWARE REQUIREMENTS:

3.2.1 Camera (USB/Webcam):

A USB camera or webcam was used to capture real-time traffic images from all four lanes of a road intersection. The camera was mounted on a rotating platform to scan each direction sequentially. It plays a critical role in enabling computer vision-based vehicle detection.

3.2.2 NEMA17 Stepper Motor:

NEMA17 is a high-torque stepper motor used to rotate the camera precisely to each side of the intersection. It enables controlled and stepwise rotation with high accuracy, ensuring each direction is correctly scanned. It is controlled through pulse signals from the Arduino.

3.2.3 A4988 Stepper Motor Driver:

The A4988 is a microstepping driver for controlling bipolar stepper motors like NEMA17. It receives signals from the Arduino and powers the stepper motor accordingly. It allows precise control of rotation angles required to scan different lanes.

3.2.4 Arduino UNO:

Arduino UNO is a widely used microcontroller board based on the ATmega328P. In this project, it was used to control LEDs, drive the stepper motor, and manage the seven-segment display. It reads input from Python (via serial) and triggers appropriate hardware actions.

3.2.5 NodeMCU (ESP8266):

NodeMCU is a Wi-Fi-enabled microcontroller based on the ESP8266 chip. It was used for wireless communication between the Python detection system and the hardware unit. It can send or receive data (like vehicle count) without requiring USB connection

3.2.6 Seven Segment Display:

The seven-segment display is used to show a countdown timer based on the green signal duration calculated after vehicle detection. It provides a user-friendly visual indication to drivers waiting at the signal.

3.2.7 LEDs (Green):

Standard 5mm LEDs were used to simulate real traffic lights. Each color is programmed to turn ON/OFF according to traffic control logic based on vehicle density. These LEDs indicate when vehicles should stop, wait, or proceed.

3.2.8 Breadboard and Jumper Wires:

A breadboard was used to build and test the circuit without soldering. Jumper wires provided connections between Arduino, LEDs, displays, and other components. This setup helped in quick prototyping and debugging

3.2.9 Breadboard Power Adapter:

This adapter is used to provide 3.3V and 5V regulated power to the breadboard circuit. It connects to an external power supply and ensures stable voltage for microcontrollers and peripherals.

3.2.10 12V DC Adapter:

The NEMA17 stepper motor requires a higher voltage (typically 12V) to operate smoothly. A separate DC adapter was used to power the motor driver (A4988), enabling proper rotation of the motor with sufficient torque.

CHAPTER-4

4. METHODOLOGY

4.1 Overview of the project:

This project presents a real-time intelligent traffic signal control system for four-way urban intersections, integrating deep learning with IoT-based hardware automation. A rotating camera mounted on a NEMA17 stepper motor captures traffic images from all four directions sequentially. Each image is processed using the YOLOv8 object detection algorithm to count vehicles, specifically cars. Based on vehicle count, the system dynamically assigns green signal durations, giving higher priority to congested lanes. The calculated duration is displayed on a seven-segment display, while traffic lights are simulated using red, yellow, and green LEDs controlled by an Arduino UNO. Communication between the Python detection logic and hardware is established via serial interface or NodeMCU.

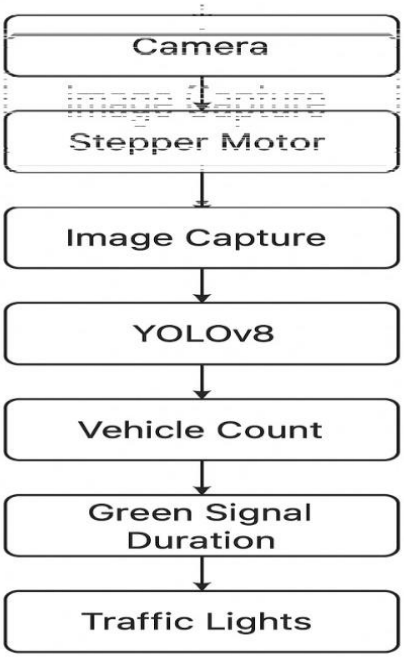


Figure 1 Intelligent Traffic System Using Real-Time Vehicle Tracking and Object Detection

Figure-1: Workflow of proposed system

4.2 Explanation of Workflow Steps

1. Camera & Stepper Motor Control

A camera mounted on a stepper motor rotates to face all four lanes of the intersection. The motor is driven by a NEMA17 motor and A4988 driver, controlled via Arduino or NodeMCU.

2. Image Capture

At each direction, the system captures a still image of the traffic lane using the camera before rotating to the next direction.

3. YOLOv8 Detection

Each image is processed using the YOLOv8 deep learning model, which identifies and counts vehicles—limited to one class: cars.

4. Vehicle Count Analysis

The vehicle count from each direction is stored and compared to determine traffic density in all lanes.

5. Green Signal Duration Calculation

The lane with the highest vehicle count is prioritized. The system calculates how long the green signal should stay active based on vehicle density.

6. Traffic Light Operation

LEDs (green, yellow, and red) are activated to simulate traffic lights. The countdown timer is shown using a seven-segment display, and the signal moves to the next lane after the timer ends.

Transformer-based NLP models for text processing

- DeepSpeech for speech recognition
- By combining these models, the overall performance and accuracy of the system improve .

4.3 Performance Analysis

The system's performance is evaluated based on several metrics:

- Accuracy – Measures the correctness of predictions.
- Precision & Recall – Assess how well the model identifies gestures and speech.
- F1-Score – Balances precision and recall for optimal performance.
- Word Error Rate (WER) – Evaluates the accuracy of speech-to-text conversion.

.

CHAPTER-5

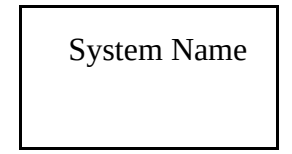
5. DESIGN VIEW

5.1 UML DIAGRAMS:

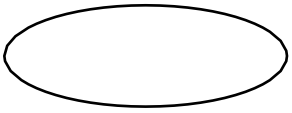
5.1.1 USECASE DIAGRAM:

A Use case diagram is an UML diagram of interactions or behaviors. It represents the user's possible interaction with a system graphically. This diagram shows the users who interact with the system and their actions. The users are shown as actors and their actions are shown as use cases.

Some of the basic symbols and notations used in Use case diagram are:



System



Use case

Actor

<<include>>

Relationships

<<extends>>

The use case diagram of the system is shown in Fig 8. The actors of the model are system, person. Each actor has its own actions to perform. The action of a person is to provide data to test whether autistic or not. The action of system is to capture data provided by the user/person. The system receives data and converts into well-defined arrays to predict the disorder

Relationships:

Illustrate the relationships with a clear line between an actor and a use case. Using arrows marked either "uses" or "extends" for relationships among usage cases. A "uses" relationship means that one usage case is required by another to perform a function. In a specific use case, an "extends" relationship implies alternate choices.

Relationships

<<include>>

<<extends>>

Communication Link:

The role of the character in the mode of use is indicated by connecting the character to the mode of use with a strong link. Actors may be connected to use cases by organizations, indicating that the actor and the use case are communicating to each other using messages.

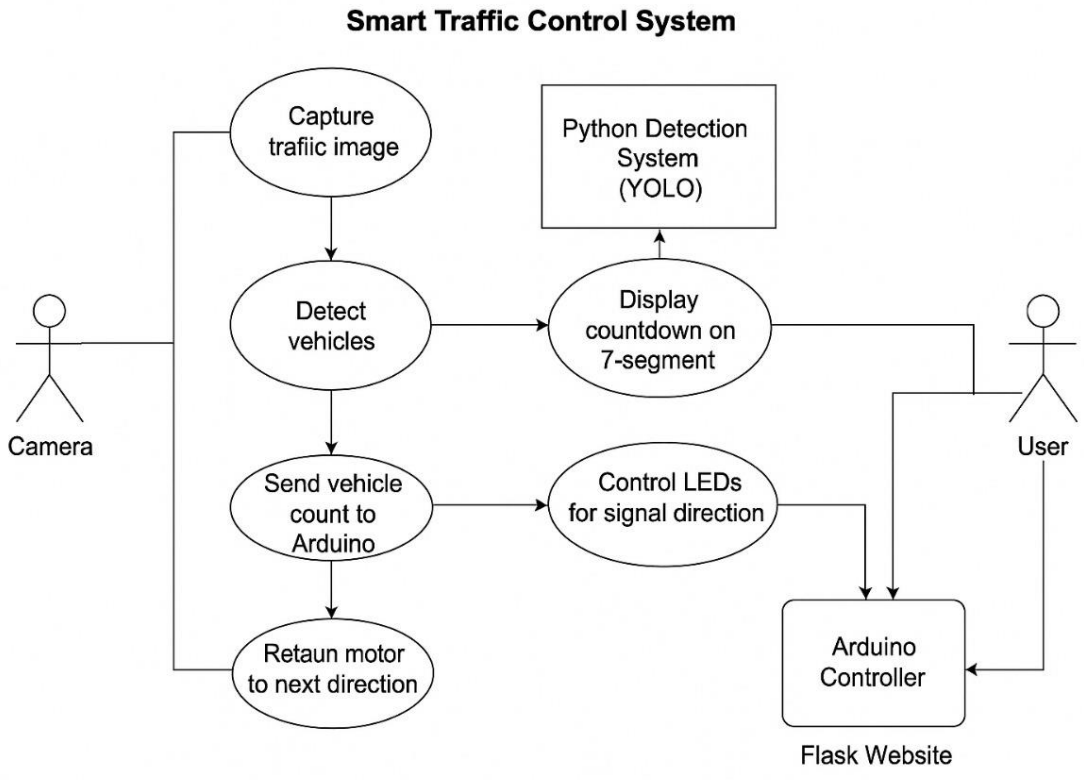


Figure-2: Use Case Diagram

5.1.2 CLASS DIAGRAM:

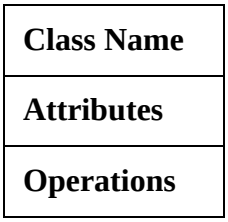
The diagram of class is a static diagram. It reflects an application's Static View. Class diagram is used not only to represent, explain, and log different aspects of a program, but also to create the software application's executable code.Class diagram defines a class's properties and processes, and the system's constraints. Class diagrams are commonly used in object-oriented systems modeling as they are the only UML diagrams that can be mapped with object- oriented languages directly.

Class diagram displays a set of groups, frameworks, partnerships, alliances and constraints. The structural diagram is also known as this.

Basic Class Diagram Symbols:

Class:

Class is an object contains attributes and the operations performed by an object. Class contains Class Name, Class Attributes and Class Operations.



In the class diagram, there are three parts. The top part of the class consists of name of the class. The middle part indicates the attributes of the class, and the lower part indicates the operations to be performed by the class which are the tasks to be performed.

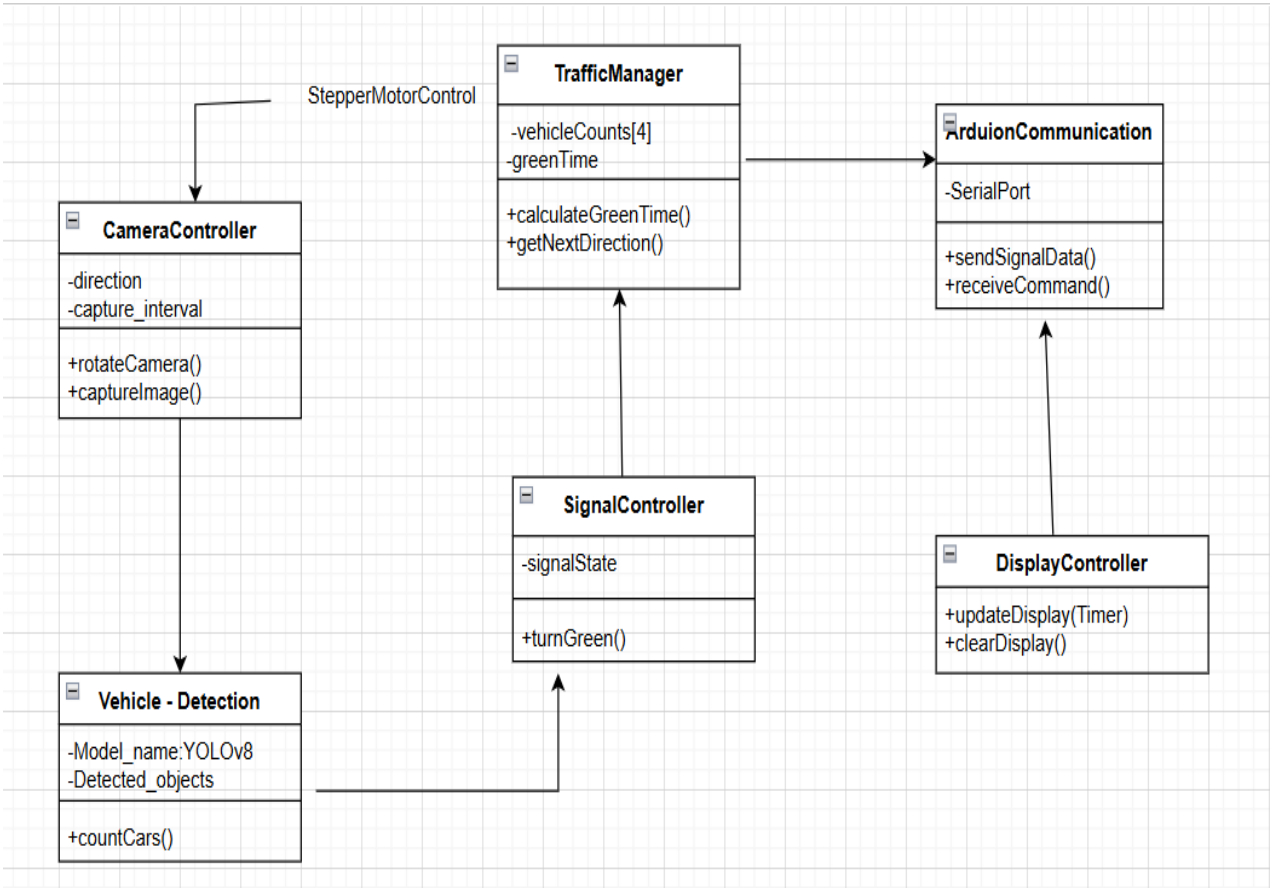


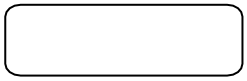
Figure-3: Class diagram

5.1. ACTIVITY DIAGRAM:

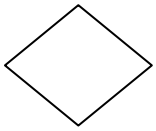
An activity diagram in UML is a behavioral diagram. It illustrate the control flow in a system from a start point to a finish point. It is very similar to a flowchart. It focuses on condition of flow and the sequence in which it happens in the system. In activity diagram, an activity is a function performed by the system. Some of the notations and symbols used in activity diagram are as follows:

Start

Finish



Activity



Decision node

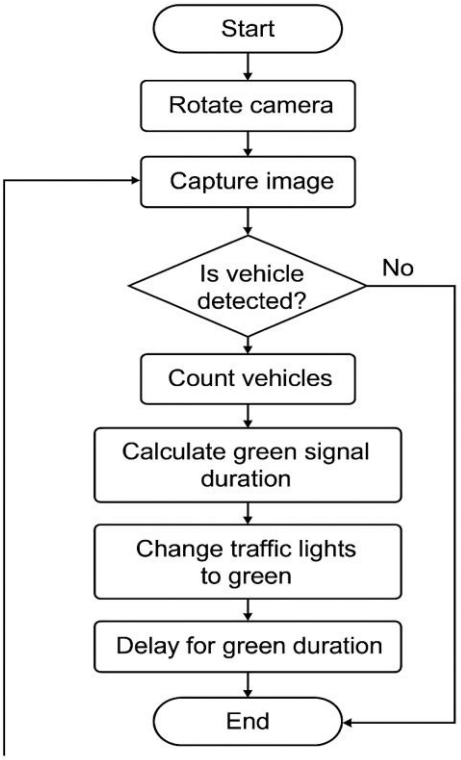


Figure 2 Activity Diagram of Intelligent Traffic Signal System

Figure-4: Activity diagram

CHAPTER-6

6.IMPLEMENTATION

Source Code:

Training.py

```
From ultralytics import YOLO

# Load YOLOv8 model
model = YOLO("yolov8m.pt")  # You can also use yolov8s.pt, yolov8m.pt, etc.

# Train YOLO on custom dataset
model.train(data="datasetfinal/data.yaml", epochs=20, imgsz=416, batch=16)
```

Arduino code

```
#include <TM1637Display.h>

// Pins for TM1637 display
#define CLK D3
#define DIO D4

// Motor driver pins
const int dirPin = D0;
const int stepPin = D1;

// LED Pins for each direction (Green LEDs only)
#define LED_N D5  // Lane 0
#define LED_E D6  // Lane 1
#define LED_S D7  // Lane 2
#define LED_W D8  // Lane 3 (after rotation back)

const int stepsPerRevolution = 50;  // For 90-degree step
const int stepDelay = 4000;         // Microseconds

TM1637Display display(CLK, DIO);
int rotationCount = 0;  // 0 → N, 1 → E, 2 → S, 3 → W (back)

void setup() {
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  Serial.begin(9600);

  display.setBrightness(5);
  display.showNumberDec(0);

  // LED setup
  pinMode(LED_N, OUTPUT);
  pinMode(LED_E, OUTPUT);
  pinMode(LED_S, OUTPUT);
```

```
 pinMode(LED_W, OUTPUT);
  allLEDsOff();
}

void countdown(int value) {
  for (int i = value; i >= 1; i--) {
    display.showNumberDec(i, false);
    delay(1000);
  }
  display.showNumberDec(0, false);
}

void motorRotateClockwise() {
  digitalWrite(dirPin, HIGH);
  for (int i = 0; i < stepsPerRevolution; i++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepDelay);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(stepDelay);
  }
}

void motorRotateAntiClockwise(int steps) {
  digitalWrite(dirPin, LOW);
  for (int i = 0; i < steps; i++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepDelay);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(stepDelay);
  }
}

void setGreenLED(int direction) {
  // Turn on only the green LED for the current direction
  digitalWrite(LED_N, direction == 0 ? HIGH : LOW);
  digitalWrite(LED_E, direction == 1 ? HIGH : LOW);
  digitalWrite(LED_S, direction == 2 ? HIGH : LOW);
  digitalWrite(LED_W, direction == 3 ? HIGH : LOW);
}

void allLEDsOff() {
  digitalWrite(LED_N, LOW);
  digitalWrite(LED_E, LOW);
  digitalWrite(LED_S, LOW);
  digitalWrite(LED_W, LOW);
}

void loop() {
  int latestCount = -1;
```

```
// Receive vehicle count from Python (via serial)
  while (Serial.available()) {
    int incoming = Serial.parseInt();
    if (incoming > 0 && incoming < 100) {
      latestCount = incoming;
    }
    delay(10);
  }

  if (latestCount != -1) {
    int countdown_time = latestCount * 10;
    Serial.print("Received count: ");
    Serial.println(latestCount);

    // Set the green LED based on current rotation
    setGreenLED(rotationCount);
    countdown(countdown_time);

    // If last direction (West, i.e., rotationCount = 3)
    if (rotationCount == 3) {
      Serial.println("Rotating back 270°...");
      motorRotateAntiClockwise(stepsPerRevolution * 3);
      allLEDsOff(); // turn off LEDs
      rotationCount = 0;

      Serial.println("Waiting for next value...");
      delay(5000);  // pause before next cycle
      while (Serial.available() == 0);  // Wait for next count
    } else {
      motorRotateClockwise();
      rotationCount++;
    }

    Serial.println("DONE");
  }
}
```

App.py

```
from flask import Flask, render_template
import threading

app = Flask(_name_)

# Global variable for total vehicle count (you can update this from another thread)
traffic_data = {"total": 0}

@app.route('/')
def index():
    total = traffic_data["total"]
```

```
  # Traffic status logic
    if total == 0:
        status = "No Traffic Detected"
    elif total > 12:
        status = "Heavy Traffic Detected"
    elif total > 6:
        status = "Medium Traffic Detected"
    else:
        status = "Low Traffic Detected"

    return render_template("index.html", total=total, status=status)

def run_app():
    app.run(host="0.0.0.0", port=5000, debug=True, use_reloader=False)
```

No of vehicles.py

```
import cv2
import serial
import time
from ultralytics import YOLO
from app import traffic_data, run_app
import threading

# Start Flask in a separate thread
threading.Thread(target=run_app).start()

# Load YOLOv8 model
model = YOLO("best.pt")
arduino = serial.Serial('COM8', 9600)
time.sleep(2)
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

lane_counter = 0
total_vehicles = 0

while True:
    ret, frame = cap.read()
    if not ret:
        print("Camera not accessible")
        break

    results = model(frame, conf=0.8)
    vehicle_count = 0

    for result in results:
        for box in result.boxes:
            cls_id = int(box.cls[0])
```

```
label = model.names[cls_id]
            if label in ["car", "bus", "truck", "motorcycle"]:
                vehicle_count += 1
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(frame, label, (x1, y1 - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

    cv2.putText(frame, f"Vehicles: {vehicle_count}", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)

    total_vehicles += vehicle_count
    lane_counter += 1

    print(f"[Lane {lane_counter}] Sending: {vehicle_count}")
    arduino.write(f"{vehicle_count}\n".encode())

    frozen_frame = frame.copy()
    wait_time = vehicle_count * 10

    print(f"Waiting {wait_time} sec showing frozen frame...")
    start = time.time()
    while time.time() - start < wait_time:
        cv2.imshow("Vehicle Detection", frozen_frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    if lane_counter == 4:
        print(f"\nTotal vehicles in 4 lanes: {total_vehicles}")
        traffic_data["total"] = total_vehicles  # 👈 Share with Flask
        lane_counter = 0
        total_vehicles = 0
        print("Waiting for next rotation cycle...\n")
        time.sleep(3)

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

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

Testing.py

```
from ultralytics import YOLO
import cv2

# Load the trained YOLOv8 model
model = YOLO("best.pt")  # Replace with your trained model

# Open the webcam
```

```
cap = cv2.VideoCapture(0)  # Use 0 for the default webcam

# Set frame width & height (optional)
cap.set(3, 640)  # Width
cap.set(4, 480)  # Height

while cap.isOpened():
    success, frame = cap.read()
    if not success:
        print("Failed to grab frame")
        break

    # Perform object detection
    results = model(frame, conf=0.6)  # Adjust confidence threshold if needed

    # Draw bounding boxes
    for result in results:
        for box in result.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])  # Bounding box
            conf = box.conf[0].item()  # Confidence score
            cls = int(box.cls[0].item())  # Class index
            label = f"{model.names[cls]} {conf:.2f}"  # Label with confidence

            # Draw rectangle and label
            cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)  # Blue bounding box
            cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

    # Display the frame
    cv2.imshow("YOLOv8 Live Detection", frame)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

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

Livecam.py

```
import cv2
import time

# Open webcam
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("❌ Cannot open camera")
    exit()

while True:
    ret, frame = cap.read()
```

```
 if not ret:
        print("❌ Can't receive frame (stream end?). Exiting ...")
        break

    cv2.imshow('Live Camera - Press S to Save, ESC to Exit', frame)

    key = cv2.waitKey(1) & 0xFF

    if key == 27:  # ESC key to exit
        break
    elif key == ord('s'):
        # Save image with timestamp
        filename = f"snapshot_{int(time.time())}.jpg"
        cv2.imwrite(filename, frame)
        print(f"✅ Image saved as {filename}")

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

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Traffic Density Monitoring</title>
                <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&display=swap"
     rel="stylesheet">
    <style>
        body {
            font-family: 'Poppins', sans-serif;
            background: linear-gradient(135deg, #2980b9, #6dd5fa, #ffffff);
            background-size: 400% 400%;
            animation: gradientBG 15s ease infinite;
            margin: 0;
            padding: 0;
            text-align: center;
            color: #333;
        }

        @keyframes gradientBG {
            0% {background-position: 0% 50%;}
            50% {background-position: 100% 50%;}
            100% {background-position: 0% 50%;}
        }

        header {
            padding: 30px;
            background-color: rgba(0, 0, 0, 0.6);
            color: white;
        }
```

```
h1 {
        font-size: 3em;
        margin: 0;
    }

    .container {
        margin-top: 60px;
        display: flex;
        flex-direction: column;
        align-items: center;
    }

    .card {
        background: white;
        padding: 30px 50px;
        border-radius: 15px;
        box-shadow: 0 8px 25px rgba(0,0,0,0.15);
        max-width: 400px;
        margin: 20px auto;
    }

    .card h2 {
        font-size: 2em;
        color: #2980b9;
    }

    .status {
        font-size: 1.5em;
        font-weight: 600;
        color: #fff;
        background-color: #27ae60;
        padding: 10px 20px;
        border-radius: 10px;
        margin-top: 15px;
    }

    .status.medium {
        background-color: #f39c12;
    }

    .status.heavy {
        background-color: #e74c3c;
    }

    footer {
        position: fixed;
        bottom: 10px;
        width: 100%;
        color: #ffffffcc;
        font-size: 0.9em;
```

```
 }
   </style>
</head>
<body>
   <header>
      <h1> 🚦 Traffic Density Monitoring System</h1>
   </header>

   <div class="container">
      <div class="card">
         <h2>Total Vehicles: {{ total }}</h2>
         <div class="status
            {% if 'Heavy' in status %}heavy{% elif 'Medium' in status %}medium{% else %}low{% endif
            %}">
            {{ status }}
         </div>
      </div>
   </div>

   <footer>
      &copy; 2025 Traffic Smart Systems. All rights reserved.
   </footer>
</body>
</html>
```

Data.yaml

```
train: C:\Users\DELL\OneDrive\Desktop\finalproject\datasetfinal\images
val: C:\Users\DELL\OneDrive\Desktop\finalproject\datasetfinal\images

nc: 1  # Change based on number of classes
names: ["car"]  # Replace with actual class names
```

CHAPTER-7

7.Testing

Test No	Test Cases	Expected Output	Actual Output	Pass/Fail
1	Importing the Libraries	Libraries imported successfully	Libraries imported without errors	Pass
2	Loading Image Dataset	Dataset loaded successfully	Images loaded and displayed correctly	Pass
3	Displaying Dataset Information	Dataset summary displayed	Dataset structure and class distribution shown	Pass
4	Image Preprocessing	Images preprocessed (rescaling, normalization)	Images preprocessed successfully	Pass
5	Train-Test Split	Data split into training, validation, and testing sets	Splitting completed without errors	Pass
6	Data Augmentation	Augmentation applied (rotation, zoom, flip)	Augmentation performed successfully	Pass
7	Model Compilation(YOLOV8)	Model compiled with no errors	Model compiled successfully	Pass
8	Model Training	Model trained on dataset	Model trained with loss and accuracy improvements	Pass
9	Model Evaluation	Model evaluated on test dataset	Accuracy and loss metrics displayed	Pass
10	Generating Confusion Matrix	Confusion matrix displayed	Confusion matrix plotted correctly	Pass
11	Saving and Loading Model	Model saved and loaded successfully	Model file stored and retrieved correctly	Pass
12	Flask Web Application Deployment	Web interface loads and predicts correctly	Flask app displays prediction results	Pass
13	Predicting New Images	Uploaded image classified as DR/Non DR	Correct prediction displayed	Pass

CHAPTER-8

8.RESULTS AND DISCUSSIONS

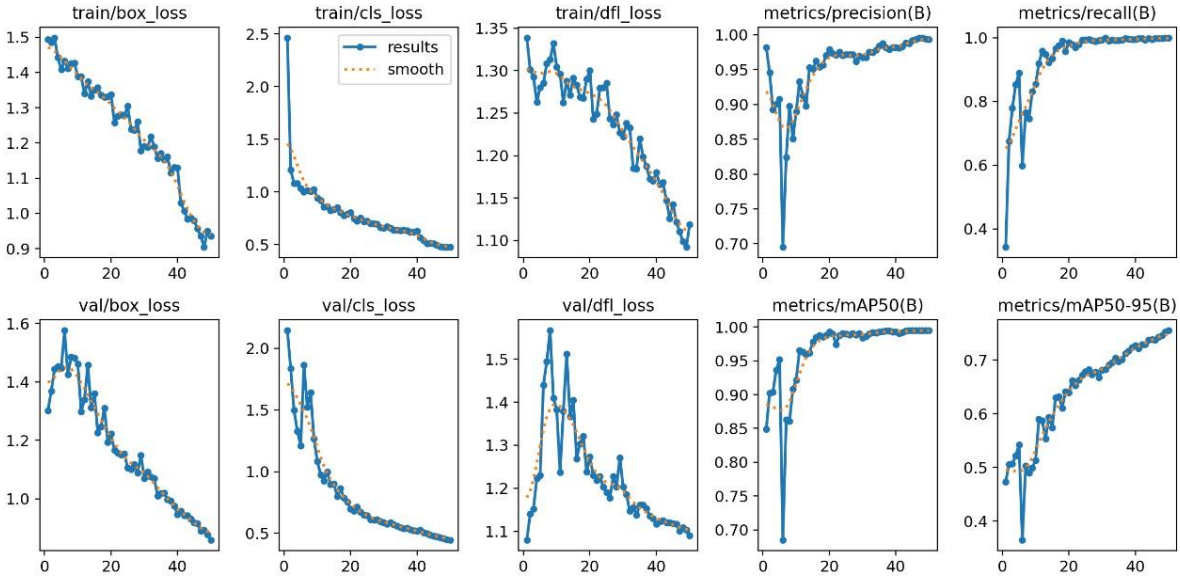


Figure-5:Performance Analysis

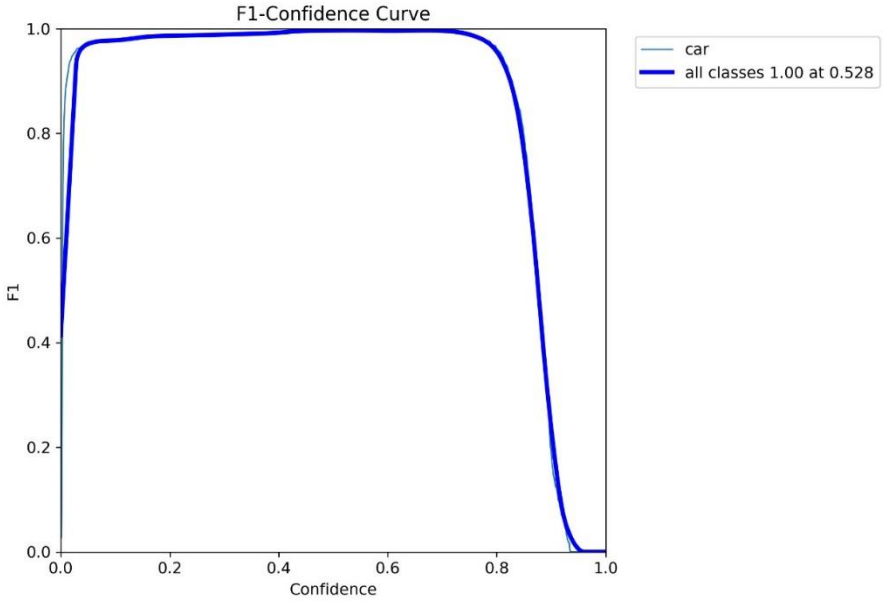


Figure-6:F1-Curve

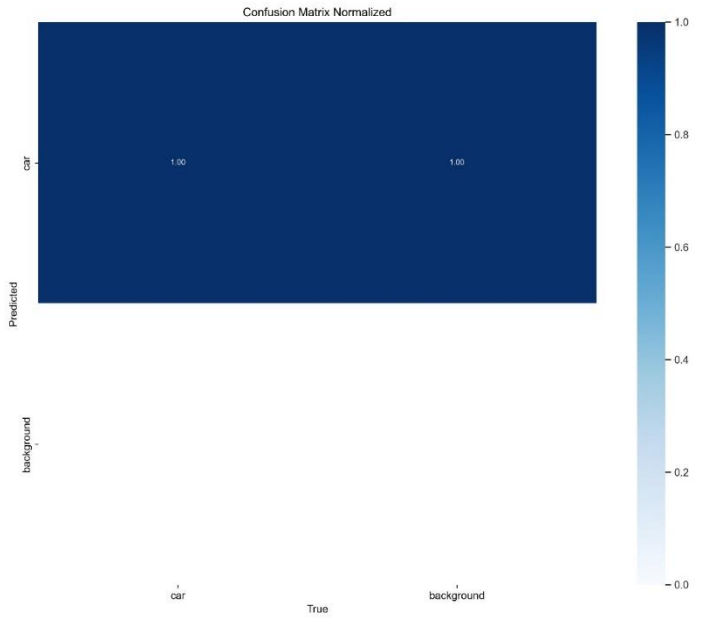


Figure-7:Confusion Matrix Normalized

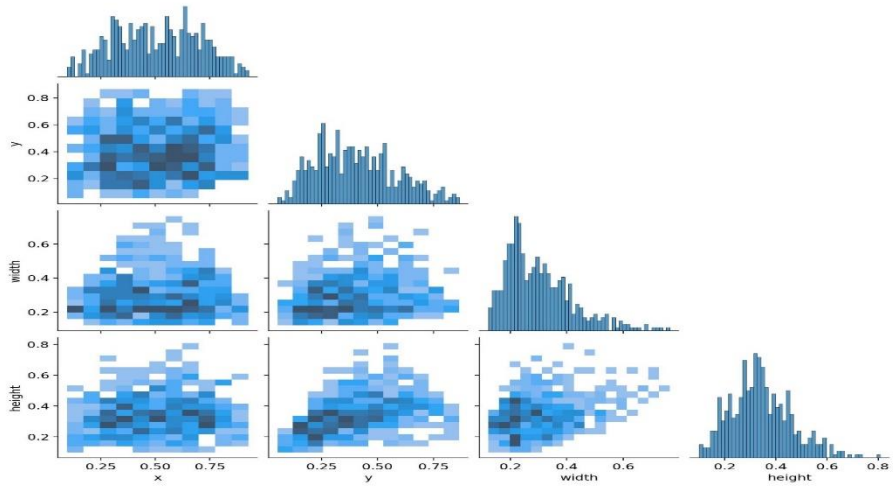


Figure-8: Labels Correlogram

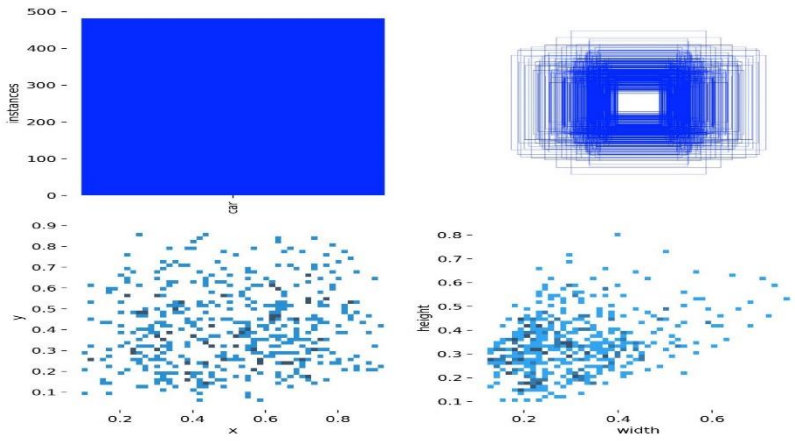


Figure-9:Labels

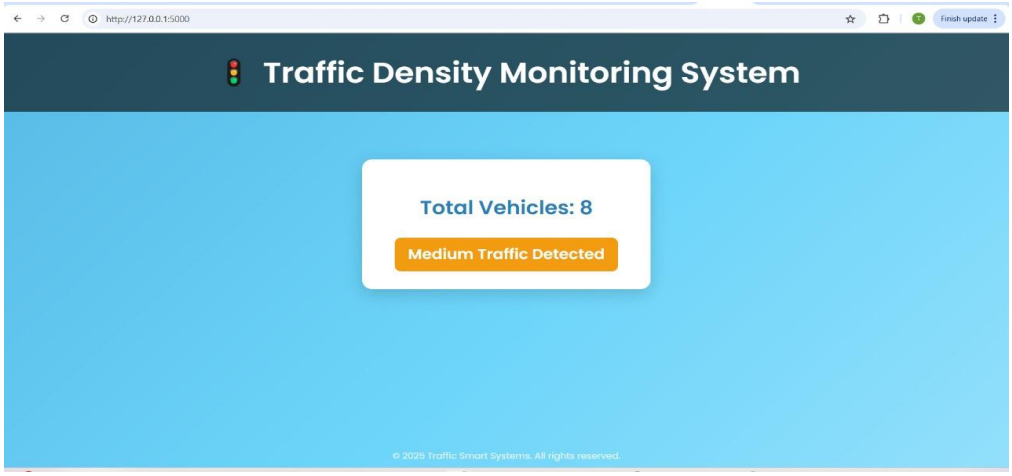


Figure-10:Web page

Figure-11:Vehicle Detection

```
Waiting 40 sec anomaly frozen frame...
[Lane 4] Sending: 4
, 416)
Speed: 5.6ms preprocess, 38.1ms inference, 1.4ms postprocess per image at shape (1, 3, 352
0: 352x416 4 cars, 38.1ms

Waiting 40 sec anomaly frozen frame...
[Lane 3] Sending: 4
, 416)
Speed: 5.4ms preprocess, 99.0ms inference, 1.3ms postprocess per image at shape (1, 3, 352
0: 352x416 4 cars, 99.0ms

Waiting 40 sec anomaly frozen frame...
[Lane 5] Sending: 4
, 416)
Speed: 3.3ms preprocess, 83.4ms inference, 1.2ms postprocess per image at shape (1, 3, 352
0: 352x416 4 cars, 83.4ms

Waiting 20 sec anomaly frozen frame...
[Lane 1] Sending: 2
320, 416)
Speed: 17.3ms preprocess, 217.2ms inference, 1.3ms postprocess per image at shape (1, 3,
0: 352x416 2 cars, 217.2ms
```

Figure-12:Vehicle Detection

Figure-13: Hardware Implementation

CHAPTER-9

9.CONCLUSION

The proposed intelligent traffic signal system effectively addresses the limitations of traditional fixed-time traffic lights by incorporating real-time vehicle tracking and adaptive signal control. Utilizing YOLOv8 for object detection and IoT components such as a rotating camera, Arduino, NodeMCU, and LEDs, the system dynamically adjusts signal durations based on real-time traffic density. The successful integration of hardware and software components ensures timely and efficient traffic management at a four-way intersection. The project demonstrates how deep learning and embedded systems can work together to reduce vehicle wait time, optimize signal duration, and improve the overall flow of traffic in urban environments.

.

CHAPTER-10

10.FUTURE ENHANCEMENT

The current implementation of the **"Intelligent Traffic System for Urban Conditions Using Real-Time Vehicle Tracking"** can be further improved and expanded in several ways. First, the object detection model can be enhanced to detect multiple vehicle classes such as bikes, buses, and trucks, allowing more accurate traffic load analysis. Increasing the size and diversity of the dataset will improve detection accuracy under different lighting and weather conditions. The system can also be upgraded to include emergency vehicle detection to prioritize ambulances or fire trucks. Integration with cloud platforms and centralized monitoring can allow for remote traffic control and data analytics. Implementing solar-powered hardware can make the system more energy-efficient and sustainable. Further, the use of edge AI devices like Jetson Nano or Raspberry Pi can reduce latency and make the system faster and more reliable. Finally, real-time alerts and dashboards can be introduced for city traffic authorities to monitor and respond to traffic congestion dynamically.

REFERENCES

[1] N. Kumar, S. S. Rahman, and N. Dhakad, "Fuzzy Inference Enabled Deep Reinforcement Learning-Based Traffic Light Control for Intelligent Transportation System," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 4919–4928, Aug. 2021.

[2] X. Du and X. Liang, "Deep reinforcement learning for traffic light control in vehicular Networks," *IEEE Trans. Veh. Technol.*, early access, Mar. 2018, doi: 10.1109/TVT.2018.2890726.

[3] J. Jin, H. Guo, J. Xu, X. Wang, and F.-Y. Wang, "An End-to-End Recommendation System for Urban Traffic Controls and Management Under a Parallel Learning Framework," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1616–1626, Mar. 2021.

[4] X. Shi, H. Liu, M. Wang, X. Li, B. Ciuffo, D. Work, and D. Kan, ''Inconsistency of AV impacts on traffic flow: Predictions in literature,'' in *Proc. Automated Road Transp. Symp.* Cham, Switzerland: Springer, Jul. 2022, pp. 165–173.

[5] A. Alruban, H. A. Mengash, M. M. Eltahir, N. S. Almalki, A. Mahmud, and M. Assiri, "Artificial Hummingbird Optimization Algorithm With Hierarchical Deep Learning for Traffic Management in Intelligent Transportation Systems," *IEEE Access*, vol. 12, pp. 17596–17603, Feb. 2024, doi: 10.1109/ACCESS.2023.3349032.

[6] A. Pundir, S. Singh, M. Kumar, A. Bafila, and G. J. Saxena, ''Cyberphysical systems enabled transport networks in smart cities: Challenges and enabling technologies of the new mobility era,'' *IEEE Access*, vol. 10, pp. 16350–16364, 2022.

[7] A. Hazarika, N. Choudhury, M. M. Nasralla, S. B. A. Khattak, and I. U. Rehman, "Edge ML Technique for Smart Traffic Management in Intelligent Transportation Systems," *IEEE Access*, vol. 12, pp. 25443–25456, Feb. 2024, doi: 10.1109/ACCESS.2024.3365930.

[8] A. B. M. Adam, M. S. A. Muthanna, A. Muthanna, T. N. Nguyen, and A. A. A. El-Latif, ''Toward smart traffic management with 3D placement optimization in UAV-assisted NOMA IIoT networks,'' *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 15448–15458, Dec. 2023.

[9] K. Cao, Y. Liu, G. Meng, and Q. Sun, ''An overview on edge computing research,'' *IEEE Access*, vol. 8, pp. 85714–85728, 2020.