Traffic Management System using Computer Vision

A Minor Project report submitted
in the partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

(Artificial Intelligence and Machine Learning)

By

21071A6682 Devanaboina Sanjana

Under the Guidance of

B. Venkatesh Reddy

Assistant Professor

CSE – AIML & IoT, VNR VJIET

Submitted to

DEPARTMENT OF

CSE - (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology

Hyderabad, Telangana

November 2024

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI
INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE, EEE, ME, ECE, CSE, EIE, IT Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 135th Rank in Engineering Category
Recognized as "College with Potential for Excellence" by UGC
VignanaJyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India.
Telephone No: 040-2304 2758/59/60, Fax: 040-23042761

CERTIFICATE

This is to certify that **DEVANABOINA SANJANA (21071A6682)** have successfully completed their Mini project work at CSE - (AIML & IoT) Department of VNRVJIET, Hyderabad entitled **"Traffic Management System using Computer Vision"** in partial fulfilment of the requirements for the award of B. Tech degree during the academic year 2023-2024.

This work is carried out under my supervision and has not been submitted to any other University/Institute for award of any degree/diploma.

GUIDE	**Head of the Department**
B. Venkatesh Reddy	Dr. Sagar Yeruva
Assistant Professor	Associate Professor
Dept. of CSE (AIML &IoT)	Dept. of CSE (AIML & IoT)

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI
INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE, EEE, ME, ECE, CSE, EIE, IT Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 135th Rank in Engineering Category
Recognized as "College with Potential for Excellence" by UGC
VignanaJyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India.
Telephone No: 040-2304 2758/59/60, Fax: 040-23042761

DECLARATION

This is to certify that our project titled "**TRAFFIC MANAGEMENT SYSTEM USING COMPUTER VISION**" submitted to Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology in complete fulfilment of the requirement for the award of Bachelor of Technology in CSE- (Artificial Intelligence and Machine Learning) is a bonafide report to the work carried out by us under the guidance and supervision of B. Venkatesh Reddy, Assistant Professor, Department of CSE - (AIML & IoT), Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology. To the best of our knowledge, this has not been submitted in any form to another University/Institute for an award of any degree/diploma.

Devanaboina Sanjana
21071A6682
Dept. of CSE (AIML)

ABSTRACT

Traffic congestion is a pressing issue in urban areas, leading to delays, increased pollution, and road safety concerns. The proposed Traffic Management System (TMS) addresses these challenges by employing computer vision techniques for real-time traffic monitoring and control. The system is built on a layered architecture that ensures functionality, scalability, and reliability. Using the YOLOv8x object detection algorithm, the system identifies, tracks, and counts vehicles in video feeds transmitted by high-definition cameras. A region-based vehicle counting approach enhances accuracy, even in dense traffic and challenging conditions like poor lighting.

The core of the system is the TMS logic, which adjusts traffic signals dynamically based on vehicle frequency thresholds. Parameters such as minimum and maximum green signal times and vehicle frequency are integrated into the decision-making process, ensuring balanced and adaptive traffic flow. A user interface developed with Streamlit provides real-time visualization of processed videos, including bounding boxes, vehicle counts, and dynamically updated traffic signals. This innovative system offers a scalable, efficient, and user-friendly solution to modern traffic challenges, reducing delays, improving road efficiency, and supporting sustainable urban development.

TABLE OF CONTENTS

Tables of Figures

1. INTRODUCTION

Rapid urbanization has exacerbated traffic congestion worldwide, causing significant delays, fuel wastage, and environmental pollution. Traditional traffic control systems, reliant on static timers or manual interventions, are inadequate in addressing dynamic traffic patterns and fail to adapt to real-time road conditions. As cities grow denser, the need for intelligent, automated traffic management solutions has become increasingly urgent.

The proposed Traffic Management System (TMS) leverages computer vision and real-time data processing to optimize urban traffic flow. The system uses YOLOv8x, a cutting-edge object detection algorithm, to detect, track, and count vehicles in video feeds captured by high-definition cameras. Unlike conventional line-based counting methods, the region-based approach improves accuracy in high-density traffic scenarios by minimizing false positives and handling overlapping objects effectively.

At the core of TMS lies an adaptive logic mechanism that dynamically adjusts traffic signals based on vehicle frequency. Predefined parameters, such as minimum and maximum green signal times, prevent lane starvation while ensuring equitable signal distribution. The TMS logic ensures seamless traffic flow, reducing congestion and delays.

The user interface, built with Streamlit, allows users to upload videos, view processed data, and monitor real-time traffic signal adjustments. This comprehensive system integrates advanced algorithms with a user-friendly interface to provide a scalable, efficient, and adaptive solution for modern urban traffic management, addressing current challenges and paving the way for smarter cities.

1.1 EXISTING SYSTEM

Traditional traffic management systems rely on fixed-timer mechanisms or manual interventions to control traffic flow. These systems lack adaptability to real-time road conditions, often leading to inefficient traffic distribution and prolonged delays. Line-based vehicle counting methods are commonly employed, which struggle to provide accurate results in dense or overlapping traffic scenarios. Additionally, such systems fail to address varying traffic densities across lanes, leading to the underutilization of low-density lanes and congestion in high-density ones. The absence of real-time data integration and dynamic decision-making limits their effectiveness in modern urban environments.

1.2 PROPOSED SYSTEM

The proposed Traffic Management System (TMS) introduces a computer vision-based solution for real-time traffic monitoring and control. Utilizing the YOLOv8x algorithm, it detects, tracks, and counts vehicles with high accuracy, even under challenging conditions like low lighting or dense traffic. A region-based counting method ensures minimal false positives and efficient handling of overlapping vehicles. The TMS logic dynamically adjusts traffic signals based on vehicle frequency thresholds, ensuring equitable lane utilization and reduced delays. A user-friendly interface provides live updates of traffic metrics, processed video feeds, and signal statuses. This system offers a scalable, adaptive, and efficient approach to modern traffic management, addressing limitations of traditional systems.

2. LITERATURE REVIEW

2.1 RELATED WORK

Traffic management systems have evolved significantly to address urban congestion and inefficiencies in traditional methods. Conventional systems like inductive loops detect vehicles through changes in magnetic fields and are widely used due to their reliability. However, they are expensive to install and maintain, and their functionality is limited to specific locations. RFID-based systems, which use tags for vehicle tracking, improve traffic monitoring but face challenges with high implementation costs and sensitivity to environmental conditions.

Vision-based systems have gained popularity due to advancements in computer vision. For instance, "Automatic Vehicle Counting System for Traffic Monitoring" employs algorithms such as Gaussian distribution modeling and motion detection to accurately count vehicles, even in challenging conditions like occlusions and variable lighting. These approaches rely on image and video data to optimize traffic flow, improving upon traditional methods' limitations.

Modern traffic management systems also use adaptive signal control, where traffic lights dynamically adjust their timing based on real-time traffic density. Techniques such as Fuzzy Logic and vehicle classification algorithms enable efficient traffic control by optimizing signal phases based on lane density. Papers like "Smart Traffic Management System" highlight how adaptive signals reduce congestion and improve flow efficiency compared to static, time-dependent systems.

Recent research focuses on improving traffic light management through simulation and UI-based control systems. These allow operators to adjust signal timings dynamically based on traffic conditions visualized on a graphical interface, avoiding the complexities of physical sensor integration. Such solutions are cost-effective, scalable, and adaptable, making them suitable for urban traffic management.

These studies showcase a shift from hardware-dependent methods like inductive loops and RFID to software-driven, UI-based systems for traffic signal optimization, emphasizing simplicity and adaptability in managing urban traffic flows.

2.2 LITERATURE SURVEY

2.2.1 An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System [3]

The article proposes a deep learning-based traffic flow detection system using edge nodes in ITS. Combining pruned YOLOv8 for vehicle detection and optimized ByteTracking for tracking, it achieves 37.9 FPS and 92.0% accuracy on Jetson TX2, enabling real-time traffic management.

2.2.2 Smart Traffic Management System - Ijraset Journal [5]

Smart Traffic Management System employs IoT for dynamic signal allocation, easing congestion. Decentralized design ensures continued operation during server failures. Provides data for urban planning, enhancing resource allocation. Offers both immediate congestion relief and long-term benefits for efficient traffic management and infrastructure development in urban areas.

2.2.3 A video-based real-time adaptive vehicle-counting system for urban roads [9]

The paper proposes an adaptive model for real-time vehicle counting in urban areas using computer vision. It introduces methods such as automatic background update and robust detection, achieving over 99% accuracy in field scenarios.

2.2.4 Real-Time Moving Vehicle Counter System using OpenCV and Python [12]

Dealing with the detection and counting of vehicles by overcoming the problems caused by the lighting and weather conditions with high accuracy.

2.2.5 Smart Traffic Monitoring System Using Computer Vision and Edge Computing [4]

The paper proposes a two-tier edge computing model for traffic monitoring, balancing high-quality video processing at the edge with strong computing power at the TMC. It adapts to network conditions and outperforms cloud-only and edge-only solutions.

2.2.6 Anomaly Detection in Road Traffic Using Visual Surveillance: A Survey [2]

The study reviews recent computer vision-based research on detecting traffic violations and on-road anomalies, analyzing techniques, datasets, and methods. It identifies gaps in current approaches and suggests future directions for enhancing anomaly detection in Intelligent Transportation Systems (ITS).

2.2.7 Deep learning-based urban big data fusion in smart cities: Towards traffic monitoring and flow-preserving fusion [6]

The experiments used the CityPulse Traffic and CityPulse Pollution datasets, and measured root mean square error (RMSE), time consumption and accuracy. A small RMSE value of 49 and highest accuracy of 92.3% compared to other baseline models depicts the applicability of the proposed model in the region-based traffic flow prediction problems in the smart cities.

2.2.8 Smart Traffic Management System - International Journal of Computer Applications [7]

The proposed work focuses on Smart Traffic management System using RFID which will eliminate the drawbacks of the existing system such as high implementation cost, dependency on the environmental conditions, etc. The proposed system aims at effective management of traffic congestion. It is also
cost effective than the existing system.

2.2.9 Smart Traffic Management System for Metropolitan Cities of Kingdom Using Cutting Edge Technologies [1]

The study proposes an IoT-based architecture utilizing agents to collect and count vehicles via IoT devices, storing real-time data on clouds through message agents. These agents act as actuators, collaborating in real-time with the environment and cloud storage. Data is broadcasted on dashboards and Google Maps, aiding citizens' decisions and saving time. Wi-Fi-enabled controllers facilitate timely message transmission. A case study validates the architecture's accuracy.

2.2.10 Automatic vehicle counting system for traffic monitoring [8]

The paper presents a vision-based system for road vehicle counting and classification using existing surveillance cameras. It employs a robust segmentation algorithm, adaptive Gaussian distribution modeling, motion detection, and occlusion detection methods to achieve accurate results, even in challenging scenarios like occlusions and shadows. Experimental results demonstrate its superiority over conventional methods, offering real-time processing and improved performance.

2.2.11 A Fast Vehicle Counting and Traffic Volume Estimation Method Based on Convolutional Neural Networkv [10]

The paper presents a vehicle counting and traffic volume estimation method based on TSI regression, avoiding complex detection and tracking operations. It introduces a simple neural network with spatial and channel attention mechanisms, achieving accurate counting efficiently. Experimental results validate the effectiveness of the proposed method.

2.2.12 Video-Based Vehicle Counting Framework [11]

The paper addresses challenges in traditional sensor-based vehicle counting methods by proposing a video-based framework. It emphasizes object detection, tracking, and trajectory processing, showcasing reliable traffic flow information acquisition.

2.2.13 The economic costs of road traffic congestion

The paper examines the rising impact of road congestion in the UK, projected to cost £30 billion annually by 2010. It highlights that congestion arises from traffic volumes exceeding road capacity, with unreliability in travel times posing a greater economic burden than slower speeds. The paper advocates for measures like congestion charging, traffic management, promoting alternative transport modes, and shifting freight to rail to reduce costs by up to 50%. Emphasizing reliability and strategic interventions, it calls for focusing on actionable marginal costs rather than total congestion estimates, which are less practical for policy development.

2.2.14 Smart Traffic Management System for Traffic Control using Automated Mechanical and Electronic Devices

Total number of vehicle information transiting, crossing and waiting for a specific traffic at a particular time range is sent by the traffic signal sensor device to the central server. The above real time data from sensors serves as input to the data analytic engine and used by the mobile agent at the STMS supervisory computer control system which is connected to the GIS mapping of the roads. Based on the congestion level and a particular threshold value, the data analytic engine sends a broadcast message to all the agent computers through mobile agent service situated at traffic controllers to divert the next two and four wheeler passengers to an alternate route.

2.2.15 Design of Intelligent Traffic Management Systems using Computer Vision and Internet of Things

The combination of IoT sensor based approach and vision based approach helps us in developing an artificially intelligent system that can handle traffic better than a traffic conductor. As the error rate of both the approaches is very less most of the traffic can be fairly and efficiently be handled and will therefore help in decreasing the traffic congestion on intersections at peek points.

2.3 SYSTEM STUDY

Effective traffic management is crucial to mitigate congestion in growing urban areas, where traditional signal systems relying on fixed timing schedules often fall short. These conventional methods fail to adapt to dynamic traffic conditions, resulting in inefficiencies such as prolonged waiting times, increased fuel consumption, and environmental pollution. To address these challenges, a dynamic and automated traffic signal control system is proposed, leveraging real-time vehicle count data.

The proposed system utilizes computer vision to count vehicles in real-time from live traffic feeds captured by cameras installed at intersections. These feeds are processed to identify the density of vehicles in each lane. Based on the vehicle count, the system dynamically adjusts signal timings to prioritize lanes with higher traffic density. This

ensures optimal signal allocation, minimizing delays and improving traffic flow efficiency.

The dynamic signal control logic is implemented through an automated algorithm that processes vehicle count data continuously. For instance, if one lane has significantly more vehicles than another, the system allocates additional green light time to that lane, ensuring smoother movement of vehicles. Conversely, during periods of low traffic, the system reduces unnecessary green light durations, enhancing overall traffic signal efficiency.

This system eliminates the drawbacks of fixed-time signals, which do not account for varying traffic patterns throughout the day. By automating signal adjustments, the need for manual intervention is reduced, thereby increasing accuracy and reliability. Additionally, this approach minimizes congestion during peak hours, balances traffic loads across all lanes, and reduces overall travel time.

The automated signal system offers several benefits, including improved fuel efficiency, reduced emissions, and enhanced road user satisfaction. Furthermore, it is cost-effective and scalable, as it does not require expensive hardware upgrades beyond cameras and computational devices. The system is designed to operate seamlessly in diverse traffic scenarios, making it adaptable to different urban settings.

In conclusion, the proposed dynamic signal management system represents a significant advancement in traffic control. By automating signals based on real-time vehicle counts, it addresses the limitations of traditional systems and provides a robust solution for modern traffic management. This system enhances overall efficiency, reduces environmental impact, and improves the commuting experience for all road users.

3. DESIGN

3.1 SYSTEM REQUIREMENTS

3.1.1 Software Requirements

3.1.1.1 Programming Languages

Python: Python is the predominant language for machine learning and data analysis. You'll need Python for implementing your ML models, data preprocessing, and analysis.

3.1.1.2 Libraries

Ultralytics

Flask

Streamlit

CV2

3.1.1.3 Google Colab

Google Colab is a cloud-based platform that provides an interactive environment for running Python code, making it ideal for machine learning, data analysis, and research. It offers free access to GPUs and easy integration with Google Drive for storing datasets and models.

3.1.1.4 VS Code

VSCode (Visual Studio Code) is a lightweight, open-source code editor developed by Microsoft, supporting various programming languages. It provides features like IntelliSense, debugging, version control, and extensions, making it a popular choice for developers working on web development, Python, and other programming tasks.

3.1.2 Hardware Requirements

3.1.2.1 Computing Power: A reasonably powerful computer with a multi-core CPU is necessary for training ML models, especially if you're working with large datasets or complex deep learning models.

3.1.2.2 GPU: Having access to a GPU (Graphics Processing Unit) can significantly speed up training deep learning models.

3.1.2.3 RAM: Adequate RAM, typically 16 GB or more, is essential for handling large datasets and training complex models

3.1.2.4 Storage: Sufficient storage space for datasets, model checkpoints and processed videos. SSDs are preferable for faster data access.

3.1.2.5 Internet Connectivity: A reliable internet connection is necessary for downloading datasets, libraries, and collaborating with team members if applicable.

3.2ARCHITECTURE

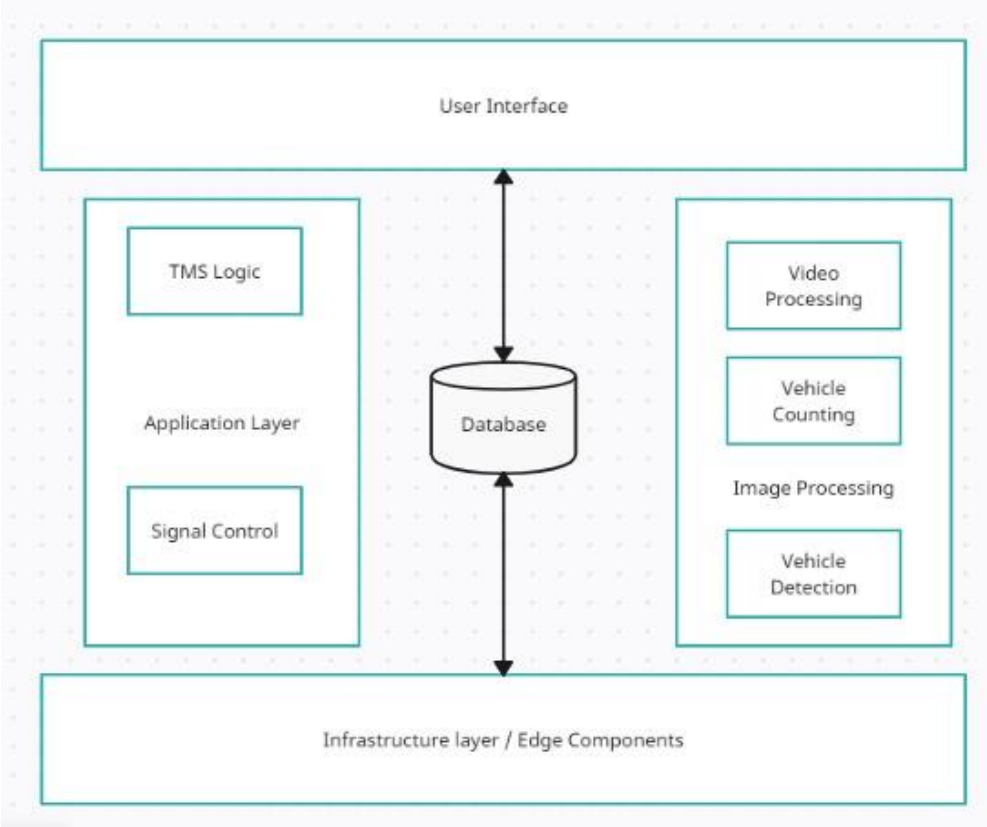

Fig. 3.2.1

User Interface Layer

The User Interface (UI) is the primary access point for users, such as traffic administrators and monitoring personnel. It allows them to view real-time traffic data, monitor vehicle flow, and manage signal operations through a web-based dashboard or mobile application. The UI provides visualization tools to display data like vehicle counts, live video streams, and traffic congestion insights. It also offers controls for configuring and optimizing traffic signals.

Application Layer

This layer contains the core business logic that drives the Traffic Management System (TMS). The TMS Logic module analyzes traffic patterns, detects incidents, and implements optimization algorithms for efficient signal control. The Signal Control module applies the decisions from the TMS Logic to manage traffic lights and ensure smooth vehicular movement. This layer acts as the brain of the system, leveraging real-time data from the edge components and historical trends stored in the database.

Database Layer

The Database serves as the central hub for storing and processing data. It holds historical traffic data, vehicle statistics, processed videos, and metadata from video feeds. This layer bridges the application layer with the edge components, enabling real-time access to critical data while supporting long-term analytics for traffic planning and optimization.

Infrastructure Layer

This layer operates at the edge of the system, collecting and processing real-time data from the field. Key components include Video Processing, which captures and analyzes live traffic footage; Vehicle Counting, which keeps track of the number of vehicles; and Vehicle Detection and Image Processing, which classify vehicle types and detect traffic congestion or incidents. These components ensure quick and efficient data collection, reducing latency and enabling faster decision-making by the application layer.

3.3UML

They are usually used to illustrate the various actions taken by the application. They also show the several users who can carry out these functions. Use-case diagrams fall under behaviour diagrams due to their emphasis on the tasks carried out and the users (actors) who carry out these tasks.

3.3.1 Use Case Diagram

The Use Case diagram represents a Traffic Management System (TMS) with three actors: TMS Logic, Model, and User. TMS Logic interacts with processes like checking signal status, setting signals (red/green), analyzing traffic flow, and assigning time. The Model supports traffic analysis. The User receives outcomes such as signal status and timings. This system ensures efficient traffic flow by dynamically managing signals based on traffic conditions.

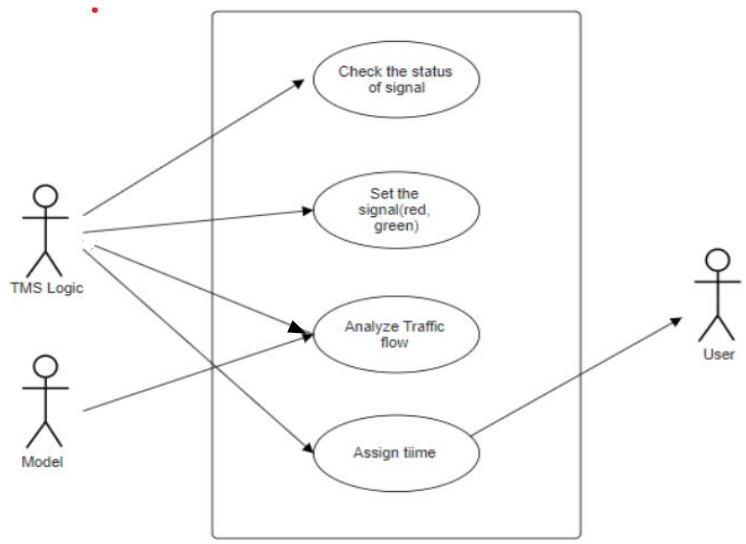

Fig. 3.3.1

3.3.2 Activity Diagram

The activity diagram illustrates a dynamic traffic signal system. It begins by setting parameters like max_time and min_frequency. The signal starts green, and the system processes video input for vehicle detection and counting. Based on vehicle frequency and elapsed time, it updates the count and timer. If the timer exceeds max_time or the frequency is below min_frequency, the signal turns red. This ensures optimized traffic flow and adaptive signal management.

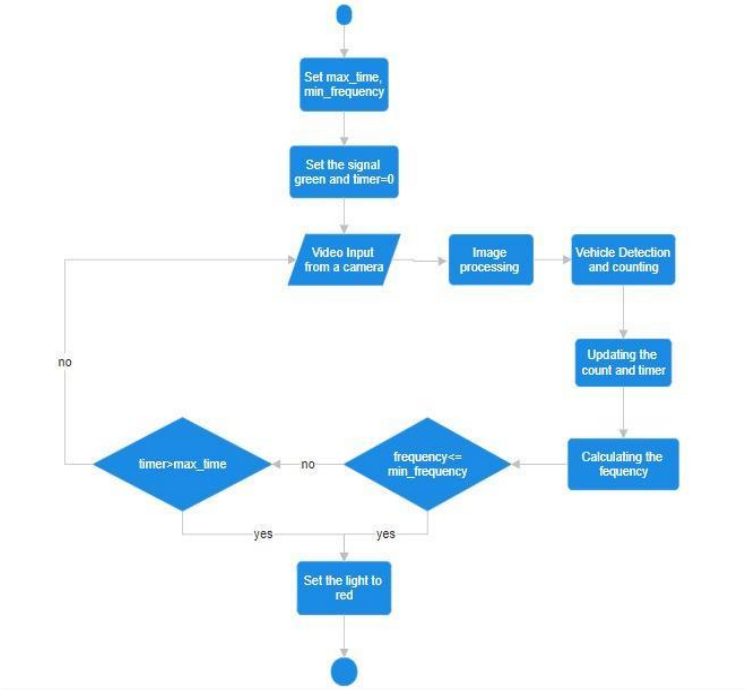

Fig. 3.3.2

3.3.3 Class Diagram

The class diagram represents a Traffic Management System composed of multiple classes. The TrafficManagementSystem class interacts with the Input class, which handles video input data and pre-processing functions. The Detection_Tracking class monitors vehicle types and crossing events, recording details like confidence levels. The vehicleCount class manages the vehicle data, and its methods interact with traffic light states via GreenLight and RedLight classes, which track frequency and maximum time data for each light. This structure supports efficient tracking and control of traffic flow.

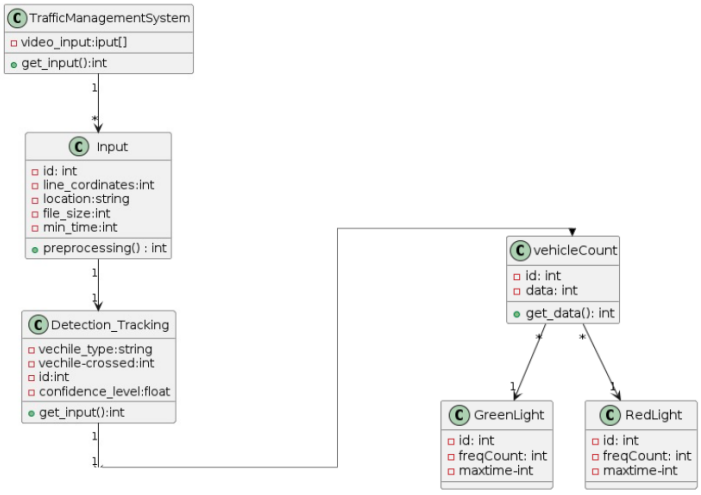

Fig. 3.3.3

3.3.4 Sequence Diagram

The sequence diagram illustrates the workflow of a Traffic Management System. A user provides video inputs via the User Interface, which are passed to the Image Processing System for analysis. Processed videos and parameters are sent to the TMS Logic, which communicates with a Model to perform vehicle detection, tracking, and counting. The Model calculates vehicle frequency, and the results are sent back through the TMS Logic and User Interface as output for the user.

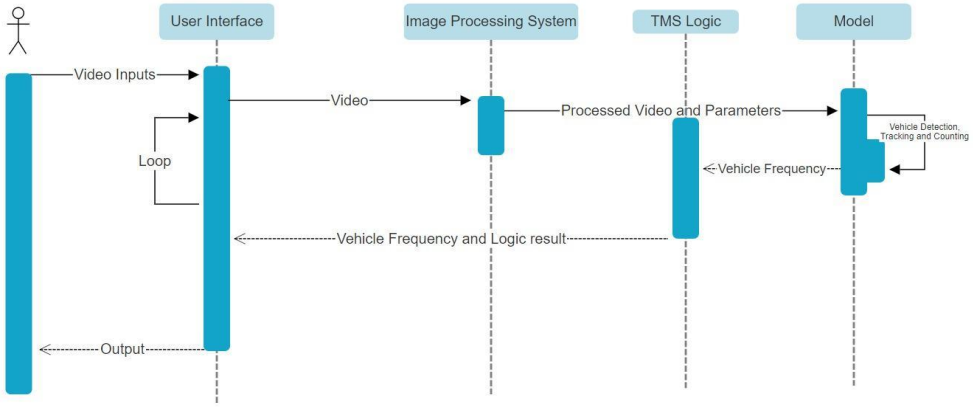

Fig. 3.3.4

3.3.5 Component Diagram

The component diagram outlines the architecture of a Traffic Management System (TMS). The system includes a User Interface (UI) for receiving input and displaying output, connected to a database for storing data. The Image Processing System comprises subsystems for Vehicle Detection and Vehicle Counting. These subsystems interact with Traffic Management Logic, which processes traffic data and executes decisions.

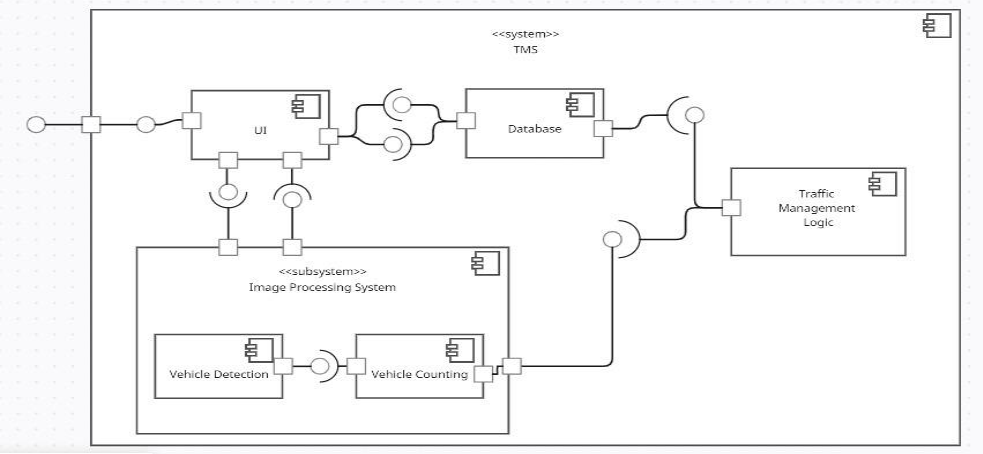

Fig. 3.3.5

3.4 USER INTERFACE

Fig. 3.4.1

The UI is a file upload interface for a "Vehicle Counting System for Traffic Management." It allows users to upload video files (up to 1GB per file) in formats such as MP4, AVI, and MPEG4. Users can either drag and drop files into the upload area or browse files manually using the provided button. This interface is designed to process uploaded videos for vehicle detection and traffic analysis.

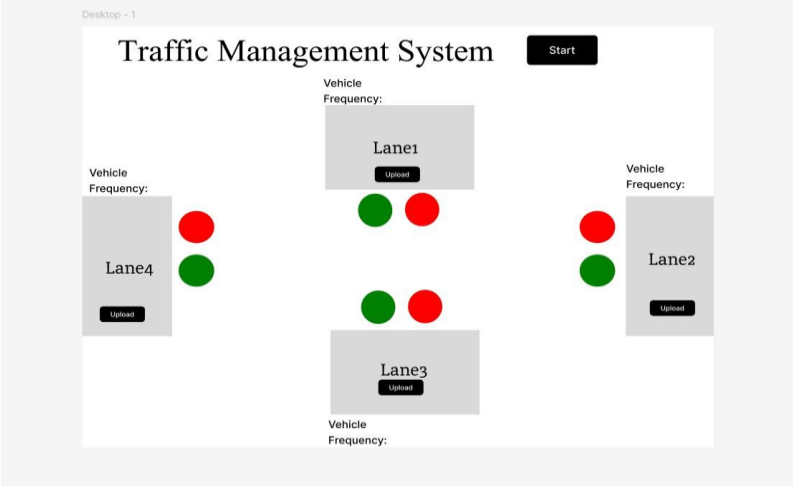

Fig. 3.4.2

This user interface displays four live-streamed, processed traffic videos, each showing real-time object detection and vehicle counts. Detected vehicles are outlined with bounding boxes, and the count is dynamically updated for each stream. Signal lights (red, green, or transitioning) on the side represent the current traffic signal state for the corresponding video feed. These signals are dynamically updated based on the vehicle count data, which is stored and managed in an array in the backend. A "Submit All" button likely allows for manual input or finalizing changes, integrating user actions into the traffic management logic.

4. Implementation

4.1 DATASET

The dataset for the proposed Traffic Management System (TMS) would primarily consist of traffic video feeds collected from high-resolution cameras installed at key intersections, highways, and critical road segments. These video feeds capture continuous footage of vehicle movements, pedestrian crossings, and overall traffic flow. To enhance the system's accuracy, the dataset would include annotated images, where objects such as vehicles, pedestrians, traffic signals, road signs, and lane markings are labeled. This labeled data is crucial for training machine learning models to recognize and predict traffic patterns. Additionally, the dataset could include historical traffic data, such as vehicle counts, average speeds, and congestion levels over time, which would be used to improve the predictive capabilities of the system. These data types combined form a comprehensive dataset that supports the development and optimization of the TMS, enabling it to perform accurate real-time traffic monitoring and management.

4.2 MODELS

YOLO

YOLOv8X, a state-of-the-art deep learning model, excels in real-time vehicle detection and counting, making it ideal for traffic management systems. Its refined architecture builds on the YOLO family, offering high-speed and accurate object detection. The model is trained on diverse datasets, ensuring robust performance in complex traffic scenarios, including occlusions, low visibility, and high-density conditions.

YOLOv8X divides images into grids, evaluates each grid for objects, and outputs confidence scores for detection. High-confidence regions form bounding boxes, which are refined to eliminate irrelevant areas and merge overlapping regions. This ensures precise localization and tracking of vehicles with minimal errors.

The model processes video at 91 FPS, maintaining high accuracy even in challenging environments. Techniques like noise removal, contrast enhancement, and occlusion handling further enhance detection quality. YOLOv8X efficiently adapts to various

deployment settings, from edge devices to cloud-based systems, enabling integration with automated signal controls.

Its advanced detection head ensures accurate vehicle counting, crucial for applications like congestion analysis and traffic flow optimization. YOLOv8X's scalability, robustness, and efficiency make it indispensable for modern traffic management, reducing congestion and improving urban mobility.

Counting the Vehicles

Counting is implemented using a region whose coordinates are set dynamically based on the dimensions of the video given as input. An object recognized as a vehicle after passing through the mentioned region is counted. Counting using region is preferred over line because of less number of false positives it gives compared to that of counting using a line. The region can handle the conditions of overlapping objects well and can work better in case of dense traffic conditions too resulting in better accuracies of counting vehicles. In the case of a line, the objects can be counted correctly only if the objects move straight to the line and it is hard for it to count when the objects linger near the line. When the region is considered for counting, the system gets the privilege of counting the vehicles moving in the desired direction only. Even if the region is spread to some part of the other road because of inaccuracies in setting the cameras, the model wouldn't count the vehicles in the other road which increases the efficiency of the system. The count of vehicle is then sent to TMS Logic and UI for rendering on the display real-time traffic flow metrics, enabling data-driven signal adjustments by TMS Logic.

Traffic Management System(TMS) Logic And Signal Controlling

This is the central part of TMS, containing basic logic as well as the most critical mechanism of logic and decision in the execution of traffic management functionalities. The TMS logic is the actual decision-making engine that is designed to read data from the model such as vehicle counts or road congestion threshold levels in real time to update the traffic signals through an algorithm that has been designed for the analysis of the flow of traffic and congestion detection and decision making. It can enhance road efficiency and minimize waits.

The TMS Logic follows a fixed order for lanes to be signaled green. Some variables have to be set beforehand for the proposed TMS logic. They include:

- Min_green_siganl_time: The minimum time for green signal that has to be given to every lane irrespective of the count/frequency of vehicles. This is done to avoid starving for lanes that have frequency of vehicles less than the min_frequncy required for green signal.
- Max_green_signal_time: This variable tells the maximum time for green signal that can be given for a lane irrespective of its frequency. This is needed to avoid the condition where the lane has a high frequency of vehicles for a long time which might let other lanes starve for its turn.
- Min_frequency: The minimum frequency that is required by a lane to maintain its signal green.
- Time_interval: The frequency of the vehicles have to be reconsidered for every fixed amount of time called Time_interval.

Whenever the signal is set green for a lane and after the Min_green_signal_time is passed for that lane, the object detection model starts recognizing, tracking and counting the vehicles that crossed the mentioned region in the video. The vehicle count gets reset to 0 for every new Time_interval. At the end of every time interval the vehicle count determined by the YOLO model is sent to the TMS logic. In this logic, the Vehicel_frequency is calculated which is the number of vehicles crossed the region per a unit time.

Vehicle_frequency = Vehicle count / Time_interval

Then the Vehicle_frequency is checked against the Min_frequency.

If Vehicle_frequency > Min_frequency → No change in the signal light

Else → The green light is turned to red

This process is continued until either the Max_green_signal_time is done or the Vehicle_frequency goes below the Min_frequency. Once the signal for than lane is set to red, then the next followed lane in the order is set to green and the same process is continued to that lane.

4.3 OVERVIEW TECHNOLOGY

1. Computer Vision and Deep Learning

The project relies on computer vision techniques for detecting and counting vehicles from video inputs. The YOLOv8X (You Only Look Once) model, imported via the ultralytics library, is the backbone for real-time object detection. YOLOv8X's optimized deep learning architecture processes video frames to detect objects, draw bounding boxes, and track vehicles efficiently. Its robustness ensures reliable performance even under challenging conditions like low lighting, high traffic density, and occlusions.

2. Programming Languages and Libraries

The system uses Python for implementation, leveraging a wide range of libraries:

- **OpenCV (`cv2`)**: Handles video frame processing, including frame extraction and preprocessing tasks.
- **Flask**: A lightweight web framework for building APIs, rendering traffic-related data, and showcasing processed outputs.
- **Streamlit**: Provides a simple and interactive web interface for uploading and displaying videos.
- **Pyngrok**: Exposes the Flask application securely over the internet.
- **MoviePy**: Assists with video editing and overlaying processed traffic data.
- **Requests**: Facilitates API calls between the UI and backend logic.
- **Collections (defaultdict)**: Manages and organizes data for vehicle counts.

These libraries together enable seamless integration of real-time detection and traffic signal control logic with user interfaces.

3. Dynamic Traffic Signal Control

Using the real-time vehicle count data from YOLOv8X, the system adjusts traffic signal timings dynamically. Custom algorithms prioritize signals for lanes with higher vehicle densities, reducing congestion and improving overall traffic flow. Flask is used to implement this logic and showcase the behavior of traffic lights alongside vehicle frequency information.

4. User Interface Development

The project incorporates two main UI tools:

- **Streamlit**: Allows users to upload videos and view processed outputs, including vehicle counts for specific time intervals. The interactive interface simplifies video processing tasks for users.
- **Flask with HTML, CSS**: Creates a web interface for visualizing traffic management logic. Features include:
    - Display of traffic signal lights with real-time updates.
    - Visualization of vehicle counts and traffic frequencies.
    - Integration of processed videos for better insights.

These tools ensure an intuitive and user-friendly experience for monitoring and analyzing traffic.

5. Data Processing and Optimization

Data preprocessing and optimization are crucial for real-time performance:

- **OpenCV** enhances video frames through noise reduction and contrast optimization.
- **MoviePy** processes and edits videos, ensuring clarity in the output.
- **Efficient tracking algorithms** minimize redundant detections, reducing false positives and improving tracking accuracy.

The combined usage of these libraries and techniques ensures that the system consistently delivers reliable results across various scenarios.

6. Scalability and Adaptability

The system's architecture is designed for scalability, making it deployable on local servers, cloud systems, or edge devices. It can handle high traffic volumes or expand to include additional features like pedestrian detection. The use of Flask and Streamlit allows flexible UI customization and seamless integration with additional traffic management components.

4.4 MODEL TESTING

1. Unit Testing:

Unit testing ensures the individual components of the system, such as YOLOv8X's detection head and bounding box prediction, function correctly. Each part is tested independently using controlled datasets to verify its accuracy and efficiency. For example, testing bounding box generation ensures precise localization before integrating it with the entire traffic management system.

2. Dataset-based Testing:

Dataset-based testing evaluates the YOLOv8X model on pre-collected traffic datasets. Balanced datasets test the model's uniformity in detecting various vehicle classes (e.g., cars, buses, motorcycles), while imbalanced datasets simulate real-world traffic patterns with uneven vehicle distributions. This approach ensures the model's adaptability to different environments and traffic densities.

3. Real-time Testing:

Real-time testing measures the system's ability to process live video streams and dynamically adjust traffic signals. The model’s accuracy and latency are assessed in real-world scenarios to confirm it meets real-time processing demands. This ensures seamless integration of detection outputs with the signal control logic and user interfaces.

4. Adverse Conditions Testing:

The system is tested under challenging conditions, such as low-light environments, bad weather (rain or fog), and occlusion-heavy scenarios. YOLOv8X’s robustness in these situations ensures consistent vehicle detection and counting, maintaining the accuracy needed for effective traffic signal adjustments.

5. Scenario-based Testing:

The system is evaluated across specific traffic scenarios, including high-density traffic, sparse traffic, intersections, and high-speed roads. This ensures the model can handle varied patterns while maintaining accurate vehicle detection and dynamic traffic signal control.

6. Edge-case Testing:

Edge-case testing challenges the system with rare or unexpected situations, such as detecting uncommon vehicles, managing multiple vehicle types simultaneously, or distinguishing between vehicles and non-vehicle objects. This ensures the system performs reliably, even in outlier scenarios not frequently encountered in standard traffic environments.

7. Cross-validation and Generalization Testing:

Cross-validation ensures the system’s performance consistency across different data subsets. Geographic diversity in testing datasets verifies the model's generalization ability, ensuring accurate detection and counting in various traffic conditions and locations.

8. User Interface Testing

The UI, developed using Streamlit and Flask, is tested for usability and responsiveness. Streamlit’s video upload and processing interface, as well as Flask’s traffic signal visualizations, are validated to ensure a smooth and intuitive user experience.

9. Performance Benchmarking:

The system is benchmarked against other detection models to evaluate accuracy, speed, and efficiency. Metrics such as Frames Per Second (FPS), detection precision, and false positive rates are compared to ensure the system meets industry standards and operates efficiently on available hardware resources.

4.5 CODE SNIPPETS

BACKEND CODE:

```
import cv2
from flask import Flask, request, jsonify, send_file
from pyngrok import ngrok
from collections import defaultdict
from ultralytics import YOLO
import os

app = Flask(__name__)

# Load the YOLOv8 model
model = YOLO('YOLOv8X.pt')

# Start ngrok tunnel
public_url = ngrok.connect(5000)
public_url = str(public_url).split("")[1]
print(f"NGROK URL: {public_url}")

# Dictionary to hold processed video info
processed_videos = {}

@app.route('/process_video', methods=['POST'])
def process_video():
    save_dir = "videos"
    processed_dir = "processed_videos"
    os.makedirs(save_dir, exist_ok=True)
    os.makedirs(processed_dir, exist_ok=True)
    video = request.files['video']
    video_path = os.path.join(save_dir, video.filename)
    video.save(video_path)

    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        return jsonify({"error": "Could not open video."}), 400

    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

    START = (0, frame_height - (frame_height // 5))
```

```
    END = (frame_width - 1, frame_height - (frame_height // 5))

    # Define the region for counting instead of a single line
    REGION_TOP = frame_height - (frame_height // 4)
    REGION_BOTTOM = frame_height - (frame_height // 5)

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    processed_video_path = os.path.join(processed_dir, f"processed_{video.filename}")
    out = cv2.VideoWriter(processed_video_path, fourcc, cap.get(cv2.CAP_PROP_FPS),
                          (frame_width, frame_height))

    track_history = defaultdict(list)
    crossed_objects = set()
    frequency_data = []
    frame_count = 0
    fps = cap.get(cv2.CAP_PROP_FPS)
    interval_frames = int(fps * 5)  # 5-second interval
    count = 0
    last_interval = 0
    total_count = 0  # To store the total number of vehicles
    while cap.isOpened():
        success, frame = cap.read()
        if success:
            frame_count += 1
            results = model.track(frame, classes=[2, 3, 5, 7], persist=True,
tracker="bytetrack.yaml", conf=0.1, iou=0.3)
            if results and results[0].boxes is not None:
                boxes = results[0].boxes.xywh.cpu()
                track_ids = results[0].boxes.id.int().cpu().tolist()
                for box, track_id in zip(boxes, track_ids):
                    x, y, w, h = box
                    track = track_history[track_id]
                    track.append((float(x), float(y)))

                    if len(track) > 100:
                        track.pop(0)

                    # Check if the vehicle is within the defined region
                    if START[0] < x < END[0] and
REGION_TOP < y < REGION_BOTTOM:
                        if track_id not in crossed_objects:
                            crossed_objects.add(track_id)
                            count += 1
```

```
                    total_count += 1  # Increment the total count

            # Calculate the current interval
            current_interval = frame_count // interval_frames

            # Check if we are at a new 5-second interval
            if current_interval > last_interval:
                # Store the count for the last interval
                frequency_data.append(count)

                # Reset the count and crossed_objects set for the new interval
                count = 0
                crossed_objects.clear()
                last_interval = current_interval

            # Draw the region for counting
            cv2.rectangle(frame, (START[0], REGION_TOP), (END[0],
REGION_BOTTOM), (0, 255, 0), 2)
            count_text = f"Objects crossed: {count}"
            cv2.putText(frame, count_text, (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

            annotated_frame = results[0].plot()
            out.write(annotated_frame)

        else:
            # At the end of the video, append the count for the last interval
            frequency_data.append(count)
            break

    cap.release()
    out.release()

    # Send total count and frequency data to FE
    video_key = f"processed_{video.filename}"


    processed_videos[video_key] = {
        "count": total_count,  # Include the total count here
        "frequency_data": frequency_data,
        "video_url": f"/downloads/{video_key}"
    }
```

```
    return jsonify({
        "count": total_count,  # Include the total count here
        "frequency_data": frequency_data,
        "video_url": f"/downloads/{video_key}"
    })

@app.route('/downloads/<filename>', methods=['GET'])
def download_video(filename):
    file_path = os.path.join("processed_videos", filename)
    if os.path.exists(file_path):
        return send_file(file_path, mimetype="video/mp4", as_attachment=False)
    else:
        return jsonify({"error": "File not found."}), 404

if __name__ == '__main__':
    app.run()
```

FRONTEND CODE

UI for Video Uploading and Processing:

```
import streamlit as st
import requests
import os
import moviepy.editor as mp
import time

# Function to remove audio from the video
def remove_audio(input_file, output_file):
    try:
        video = mp.VideoFileClip(input_file)
        video_without_audio = video.without_audio()
        video_without_audio.write_videofile(output_file, codec='libx264', fps=video.fps)
        print(f"Processed video duration: {video_without_audio.duration} seconds")
    except Exception as e:
        print(f"Error removing audio: {e}")

# File uploader for video upload
video_file = st.file_uploader("Upload Video", type=["mp4", "avi"])

if video_file:
    video_url, total_count, frequency_data = process_video(video_file)
```

```
    if video_url:
        st.write(f"Total Vehicles Count: {total_count}")


        # Download and save the video in the temp folder
        local_video_path = download_and_save_video(video_url)
        if local_video_path:
            no_audio_video_path = 'temp/' + os.path.basename(local_video_path)
+ '_no_audio.mp4'
            remove_audio(local_video_path, no_audio_video_path)


            # Create placeholders for dynamic updates
            frequency_placeholder = st.empty()
            video_placeholder = st.empty()
            frequency_container = st.container()

            # Get the total duration of the video
            video_duration = mp.VideoFileClip(no_audio_video_path).duration

            # Display the video file with autoplay, and start at 0 seconds
            video_placeholder.video(no_audio_video_path, start_time=0)

            # Start a custom timer when the video starts playing
            start_time = time.time()
            current_time = 0
            i = 0

            with frequency_container:
                st.write("Frequency Data:")
                all_frequencies = st.empty()

            all_frequency_texts = ""

            while current_time < video_duration and i < len(frequency_data):
                elapsed_time = time.time() - start_time

                # Check if 5 seconds have passed
                if elapsed_time >= 5 * (i + 1):
                    # Update the frequency text above the video
                    frequency_text = f"Time {i*5}-{(i+1)*5} seconds:
{frequency_data[i]} vehicles"
                    frequency_placeholder.write(frequency_text)
```

```
                # Append to all frequency texts and display in the scrollable container
                all_frequency_texts += frequency_text + "\n"
                all_frequencies.text_area(" ", all_frequency_texts, height=200)
                i += 1

            current_time = elapsed_time
            time.sleep(0.05)  # Sleep briefly to prevent busy waiting

        # Handle the remaining time if any
        if i < len(frequency_data):
            while i < len(frequency_data):
                frequency_text = f"Time {i*5}-{(i+1)*5} seconds: {frequency_data[i]}
vehicles"
                frequency_placeholder.write(frequency_text)
                all_frequency_texts += frequency_text + "\n"
                all_frequencies.text_area(" ", all_frequency_texts, height=200)
                i += 1
else:
    st.warning("Please upload a video file.")
```

TMS Logic Code:

```
const videos = [
    document.getElementById('top-video-player'),
    document.getElementById('right-video-player'),
    document.getElementById('bottom-video-player'),
    document.getElementById('left-video-player')
];

let btn = document.getElementById("submit-all");
let pauseArrays = [
    [18, 14, 15, 19, 20, 21, 9, 20, 19, 18, 23, 22, 1],
    [17, 23, 27, 16, 14, 15, 18, 12, 21, 32, 35, 22, 2],
    [4, 6, 2, 4, 1, 2, 4, 3, 6, 6, 7, 6, 8],
    [9, 10, 6, 7, 5, 3, 5, 6, 5, 5, 10, 7, 7]
];
const min_freq = 1;
const max_play_time = 15;  // Max time a video should play in seconds
const switch_time = 5;     // Time to update frequency and pause array every 5
//seconds
```

```
let pauseIndexes = [0, 0, 0, 0]; // Track the current index for each video
let currentVideoIndex = 0;       // Track which video is playing
let frequencyInterval, playTimeout; // Interval for frequency updates and timeout for
max play time

// Frequency display element
const frequencyDisplay = document.getElementById('frequency-value');

// Light buttons
const redLights = [
    document.getElementById('redBtn1'),
    document.getElementById('redBtn2'),
    document.getElementById('redBtn3'),
    document.getElementById('redBtn4')
];
const greenLights = [
    document.getElementById('greenBtn1'),
    document.getElementById('greenBtn2'),
    document.getElementById('greenBtn3'),
    document.getElementById('greenBtn4')
];

// Function to update light status
function updateLights() {
    for (let i = 0; i < videos.length; i++) {
        if (i === currentVideoIndex) {
            greenLights[i].style.opacity = "1";  // Bright green for current video
            redLights[i].style.opacity = "0.3";  // Dim red for current video
        } else {
            greenLights[i].style.opacity = "0.3";  // Dim green for inactive videos
            redLights[i].style.opacity = "1";      // Bright red for inactive videos
        }
    }
}

// Function to update the frequency display and pause array index for the current video
```

```
function updateFrequency() {
    const currentVideo = videos[currentVideoIndex];
    const pauseArray = pauseArrays[currentVideoIndex];
    let currentIndex = pauseIndexes[currentVideoIndex];

    if (currentIndex < pauseArray.length) {
        const currentFrequency = pauseArray[currentIndex] / 5;
        frequencyDisplay.textContent = currentFrequency.toFixed(2); // Update
frequency display
        console.log(`Video ${currentVideoIndex + 1} frequency:
${currentFrequency}`);

        // Increment pause index for the current video
        pauseIndexes[currentVideoIndex] = currentIndex + 1;
        console.log(`Updated pause index for video ${currentVideoIndex + 1} to
${pauseIndexes[currentVideoIndex]}`);

        // If the current frequency is too low, pause the video and move to the next
        if (currentFrequency < min_freq) {
            currentVideo.pause();
            console.log(`Video ${currentVideoIndex + 1} paused due to low frequency.`);
            playNextVideo();
        }} else {
        // Reset the pause index to 0 once it exceeds the array length
        pauseIndexes[currentVideoIndex] = 0;
        currentVideo.pause();
        console.log(`Video ${currentVideoIndex + 1} reached the end of pause array,
resetting.`);
        playNextVideo();  // Play the next video after resetting the index
    }
}

// Function to play the current video and check conditions for stopping
function checkPauseCondition() {
    const currentVideo = videos[currentVideoIndex];
    const pauseArray = pauseArrays[currentVideoIndex];
    let currentIndex = pauseIndexes[currentVideoIndex];
```

```
    // Clear previous timeouts and intervals
    clearTimeout(playTimeout);
    clearInterval(frequencyInterval);

    if (currentIndex < pauseArray.length) {
        const currentFrequency = pauseArray[currentIndex] / 5;

        // Ensure the video plays for at least 5 seconds
        currentVideo.play();
        updateLights();  // Update light status

        console.log(`Video ${currentVideoIndex + 1} is playing. Frequency:
${currentFrequency}`);

        // Set up frequency updates every 5 seconds
        frequencyInterval = setInterval(() => {
            updateFrequency();  // Increment pause index and update frequency for the
current video
        }, switch_time * 1000);

        // Stop the video after max play time (15 seconds)
        playTimeout = setTimeout(() => {
            clearInterval(frequencyInterval);  // Stop frequency updates after max time
            currentVideo.pause();
            console.log(`Video ${currentVideoIndex + 1} paused after max play time.`);

            // Move to the next video in sequence
            playNextVideo();
        }, max_play_time * 1000);
    } else {
        currentVideo.pause();
        console.log(`Video ${currentVideoIndex + 1} completed all play conditions.`);

        // Reset the pause index and move to the next video
        pauseIndexes[currentVideoIndex] = 0;
        playNextVideo();
```

```
    }
}

// Function to play the next video in sequence
function playNextVideo() {
    // Move to the next video in the sequence (top, right, bottom, left)
    currentVideoIndex = (currentVideoIndex + 1) % videos.length;

    // Play the next video
    checkPauseCondition();
}

// Event listener for the submit button
btn.addEventListener('click', () => {
    // Pause all videos initially
    videos.forEach(video => video.pause());

    // Reset pauseIndexes to start from the beginning for all videos
    pauseIndexes = [0, 0, 0, 0];

    // Start by playing the first video (top video)
    currentVideoIndex = 0;
    checkPauseCondition();
});
```

5. TESTING

5.1 Test Cases

Test case-1: Presence of non-vehicular objects

Fig. 5.1.1

Test case-2: Detection of Vehicles in the different lane

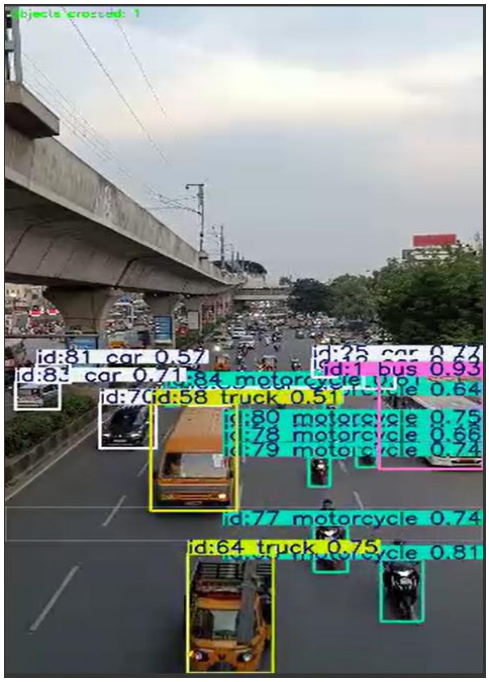

Fig. 5.1.2

Test case-3: Uploading the wrong format

Fig. 5.1.3

6. RESULTS

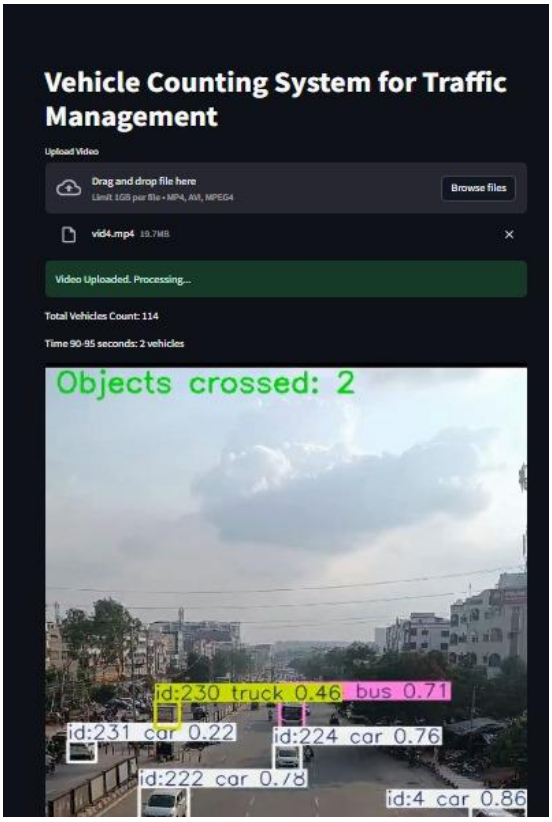

Fig. 6.1

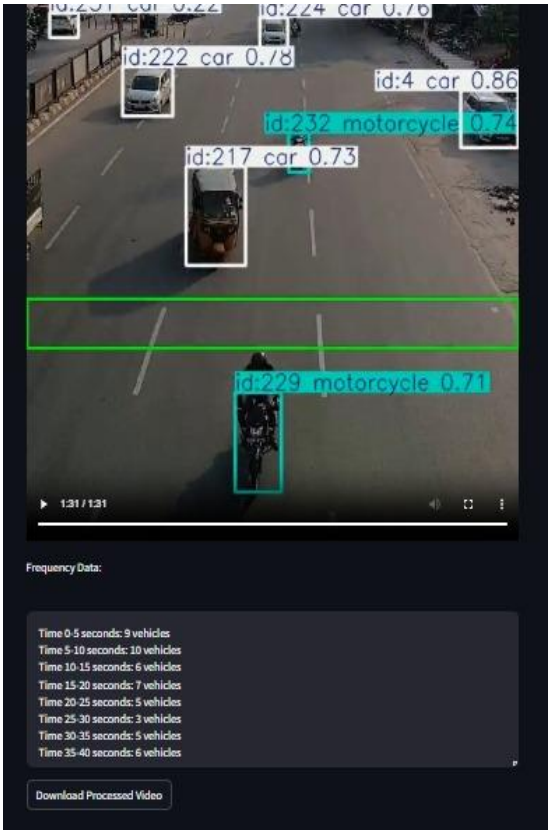

Fig. 6.2

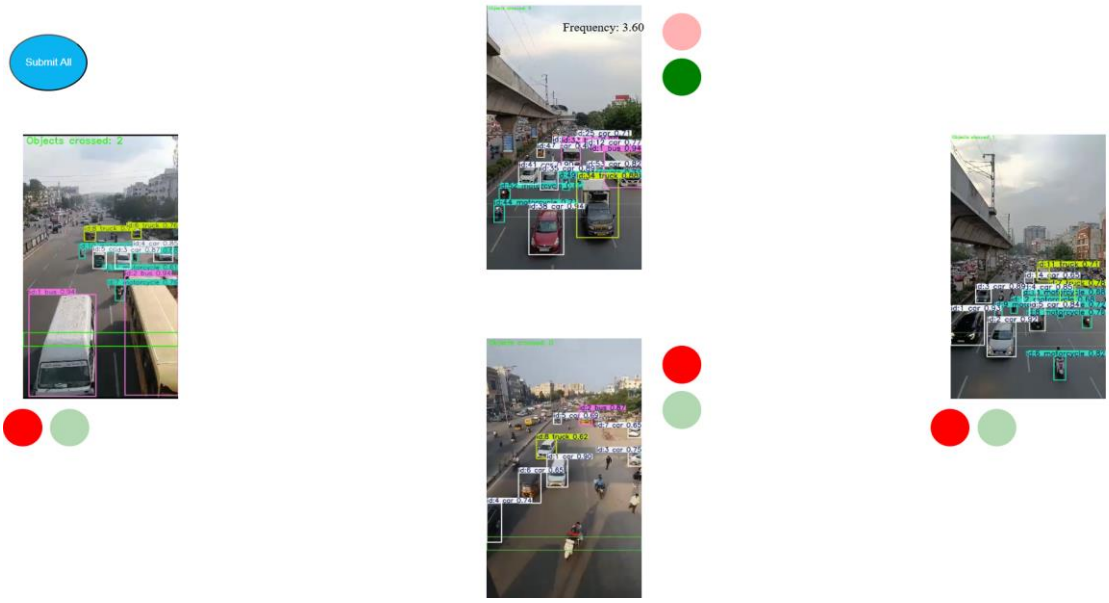

Fig. 6.3

7. CONCLUSION

This project introduces a dynamic and automated traffic management solution using YOLOv8X for real-time vehicle detection and counting. By leveraging advanced computer vision techniques, the system adjusts traffic signals based on vehicle density, optimizing traffic flow and reducing congestion. The integration of Flask and Streamlit provides an intuitive user interface for monitoring traffic patterns and reviewing processed video outputs, enhancing usability. YOLOv8X ensures high accuracy and robustness in vehicle detection, even under challenging conditions like low light, adverse weather, or occlusions, meeting the real-time demands of traffic control systems.

The system undergoes rigorous testing, including real-time, adverse condition, and scenario-based evaluations, to ensure reliability in diverse traffic environments. Its flexibility and scalability make it suitable for deployment across various intersections and urban areas, addressing different traffic patterns and needs. By automating signal control based on real-time vehicle counts, this project contributes to reducing congestion, improving road safety, and enhancing urban transportation efficiency. The success of this system paves the way for further advancements in smart city solutions, demonstrating the potential of machine learning and AI in solving real-world traffic challenges.

8. FUTURE SCOPE

The future scope of this traffic management system is vast, with potential for continuous improvements and broader applications. One significant area for expansion is the integration with smart city systems, where the traffic management solution can collaborate with other urban infrastructure, such as public transportation and emergency vehicle routing. This would allow for seamless coordination and optimization of traffic flow across different systems. Additionally, the system can be enhanced by incorporating multi-lane detection and pedestrian monitoring to improve safety at complex intersections and enhance overall traffic control. Another avenue for growth is the integration of IoT sensors, enabling the system to collect data from multiple sources such as road sensors and vehicle counters, further increasing its accuracy and reliability.

There is also potential for using predictive analytics to forecast traffic patterns based on historical data, allowing for proactive adjustments to traffic signals and preventing congestion before it occurs. The scalability of the system can be improved by extending its reach to larger urban areas and highways, enabling centralized control and real-time monitoring across extensive road networks. Moreover, with the continuous advancement in machine learning and deep learning models, future iterations could leverage more lightweight and energy-efficient algorithms, which would further reduce computational costs and enhance real-time performance. Lastly, integrating sustainability efforts into the system, such as optimizing traffic signals for reduced fuel consumption, would make this technology even more environmentally friendly, promoting greener urban mobility solutions in the long term.

BIBLIOGRAPHY

1. Humayun, Mamoona, Afsar, Sadia, Almufareh, Maram Fahaad, Jhanjhi, N. Z., AlSuwailem, Mashayel, Smart Traffic Management System for Metropolitan Cities of Kingdom Using Cutting Edge Technologies, *Journal of Advanced Transportation*, 2022, 4687319, 13 pages, 2022. doi:10.1155/2022/4687319
2. K. K. Santhosh, D. P. Dogra, and P. P. Roy. 2020. Anomaly Detection in Road Traffic Using Visual Surveillance: A Survey. ACM Comput. Surv. 53, 6, Article 119 (November 2021), 26 pages. doi: 10.1145/3417989
3. C. Chen, B. Liu, S. Wan, P. Qiao and Q. Pei, "An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1840-1852, March 2021, doi: 10.1109/TITS.2020.3025687
4. G. Liu *et al*., "Smart Traffic Monitoring System Using Computer Vision and Edge Computing," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 12027-12038, Aug. 2022, doi: 10.1109/TITS.2021.3109481.
5. Shruti P Naik, Pratik Badgire, Suraj Deore, Yash Ghorpade, Disha Patil .*"Smart Traffic Management System"*, Volume 11, Issue X, International Journal for Research in Applied Science and Engineering Technology (IJRASET) Page No: 955-959, ISSN : 2321-9653, www.ijraset.com
6. Sulaiman Khan, Shah Nazir, Iván García-Magariño, Anwar Hussain, Deep learning-based urban big data fusion in smart cities: Towards traffic monitoring and flow-preserving fusion, Computers & Electrical Engineering, Volume 89, 2021, 106906, ISSN 0045-7906, https://doi.org/10.1016/j.compeleceng.2020.106906.
7. Ninad Lanke, Sheetal Koul . Smart Traffic Management System. International Journal of Computer Applications. 75, 7 (August 2013), 19-22. DOI=10.5120/13123-0473
8. Alain Crouzil, Louahdi Khoudour, Paul Valiere, Dung Nghy Truong Cong, "Automatic vehicle counting system for traffic monitoring," J. Electron. Imag. 25(5) 051207 (1 June 2016) https://doi.org/10.1117/1.JEI.25.5.051207

9. Liu F, Zeng Z, Jiang R (2017) A video-based real-time adaptive vehicle-counting system for urban roads. PLOS ONE 12(11): e0186098. https://doi.org/10.1371/journal.pone.0186098
10. H. Yang, Y. Zhang, Y. Zhang, H. Meng, S. Li and X. Dai, "A Fast Vehicle Counting and Traffic Volume Estimation Method Based on Convolutional Neural Network," in *IEEE Access*, vol. 9, pp. 150522-150531, 2021, doi: 10.1109/ACCESS.2021.3124675.
11. Z. Dai *et al*., "Video-Based Vehicle Counting Framework," in *IEEE Access*, vol. 7, pp. 64460-64470, 2019, doi: 10.1109/ACCESS.2019.2914254.
12. Bailke, Preeti & Divekar, Sanika. (2022). REAL-TIME MOVING VEHICLE COUNTER SYSTEM USING OPENCV AND PYTHON. International Journal of Engineering Applied Sciences and Technology. 6. 190-194. 10.33564/IJEAST.2022.v06i11.036.