

CLOUD BASED SMART TRAFFIC MANAGEMENT SYSTEM

PROJECT REPORT

Submitted by

NAVEEN M

(Register Number: 20CS1080)

RAGUL S

(Register Number: 20CS1091)

SURYA GOKUL S

(Register Number: 20CS1110)

Under the guidance of

Dr. KA SELVARADJOU

PROFFESOR

Department of Computer Science and Engineering

**to the Pondicherry University, in partial fulfillment of the requirement for the
award of degree**

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PUDUCHERRY TECHNOLOGICAL UNIVERSITY

PUDUCHERRY – 605 014

May 2024.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PUDUCHERRY TECHNOLOGICAL UNIVERSITY
PUDUCHERRY – 605 014.**

BONAFIDE CERTIFICATE

This is to certify that the Project work titled “**Cloud Based Smart Traffic Management System**” is a bonafide work done by **NAVEEN M (20CS1080)**, **RAGUL S (20CS1091)**, and **SURYA GOKUL S (20CS1110)** in partial fulfillment for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** of the **Puducherry Technological University** and that this work has not been submitted for the award of any other degree of this/any other institution.

Project Guide
(Dr. KA SELVARADJOU)

Head of the Department
(Dr. G. ZAYARAZ)

*Submitted for the University Examination held on*_____

Project Coordinator

External Examiner

ACKNOWLEDGEMENT

We are deeply indebted to **Dr. KA SELVARADJOU, PROFESSOR**, Department of Computer Science and Engineering, Puducherry Technological University, Puducherry, for his valuable guidance throughout this project.

We also express our heart-felt gratitude to **Dr. G. Zayaraz**, Professor & Head (CSE) for giving constant motivation in succeeding my goal.

With profoundness we would like to express our sincere thanks to **Dr. S. Mohan**, the Vice-Chancellor, Puducherry Technological University, for his kindness in extending the infrastructural facilities to carry out my project work successfully.

We would be failing in our duty if we do not acknowledge the efforts of the Project Co-Ordinator, **Dr. S. Lakshmana Pandiyan**, and the Project Review Panel members, viz. **Dr. Ka. Selvaradjou, Dr. F. Sagayaraj Francis, Dr. J. Kumaran @ Kumar** and **Dr. M. Thenmozhi** for shaping our ideas and constructive criticisms during project review.

We also express our thanks to all the **Faculty** and **Technical Staff** members of the CSE department for their timely help and the **Central Library** for facilitating useful reference materials.

We would be failing in our duty if we don't acknowledge the immense help extended by my friends, who have always been with us in all my trails and tribulations and encouraging us to complete.

NAVEEN M

RAGUL S

SURYA GOKUL S

ABSTRACT

In today's urban landscape, traffic congestion has become a pervasive issue, necessitating real-time monitoring and control of road traffic density for effective traffic management. Current challenges include capacity constraints, demand limitations, and static signal timings, which fail to adapt to dynamic traffic conditions. This paper proposes a cloud-based smart traffic management system utilizing image processing and vehicle detection algorithms to analyze live video feeds from intersection cameras. By dynamically adjusting signal timings based on vehicle density and prioritizing emergency vehicles, the system aims to mitigate congestion, reduce accidents, and enhance overall transportation efficiency. Leveraging real-time data processing and intelligent signal control, the system offers a promising solution for safer and smoother urban traffic flow.

In this study, we introduce a novel approach to traffic management leveraging cloud-based technology and image processing algorithms. Specifically, we explore the integration of ESP32 microcontrollers for real-time control of traffic lights, enhancing the adaptability and responsiveness of the system. By harnessing the power of cloud computing and advanced image processing techniques, our proposed system offers dynamic signal control based on live video feeds from intersection cameras. Through the utilization of ESP32 devices, we demonstrate the feasibility of efficient and intelligent traffic management, with the ability to adjust signal timings in response to changing traffic conditions. This integration marks a significant advancement in traffic control systems, promising improved congestion management, reduced travel times, and enhanced overall traffic flow in urban environments.

TABLE OF CONTENTS

Chapter No	Title	Page No
	BONAFIDE CERTIFICATE	i
	ACKNOWLEDGEMENT	ii
	ABSTRACT	iii
	TABLE OF CONTENTS	iv
	LIST OF FIGURES	vii
	LIST OF TABLES	viii
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	
	1.1 OVERVIEW	1
	1.2 OBJECTIVE OF THE STUDY	1
	1.3 MOTIVATION	2
	1.4 ORGANISATION OF CHAPTERS	2
2	LITERATURE REVIEW	
	2.1 TECHNIQUES	4
	2.1.1 Traffic control system design	4
	2.1.2 Object detection techniques	5
	2.1.3 Advanced traffic analysis techniques	5
	2.1.4 Marker and camera pose estimation	6
	2.1.5 Comparative evaluation of computer vision libraries	6
	2.1.6 Traffic simulator development	7
	2.1.7 Image processing techniques for noise estimation	7
	2.2 SURVEY OF THE RELATED WORKS	8
	2.3 LIMITATIONS OF THE EXISTING WORKS	10
	2.3.1 Implementation challenges of VANET	10
	2.3.2 Scalability and real-world implementation	10

	2.3.3 Methodological limitations	10
	2.3.4 Limitations in data evaluation	11
	2.3.5 Limitations in noise estimation procedure	11
	2.3.6 Limitations of smart mobility procedure	11
	2.4 SUMMARY	12
3	EXISTING WORK	
	3.1 OVERVIEW OF THE EXISTING WORK	13
	3.2 TECHNIQUES USED	13
	3.3 MODULES DESCRIPTION	14
	3.3.1 Image acquisition module	14
	3.3.2 Image processing module	14
	3.3.3 Vehicle count and emergency vehicle detect module	15
	3.3.4 Traffic signal control module	15
	3.4 LIMITATIONS OF THE EXISTING WORK	16
	3.5 SUMMARY	16
4	PROPOSED WORK	
	4.1 OVERVIEW	17
	4.2 PROBLEM STATEMENT	17
	4.3 PROPOSED SOLUTIONS	17
	4.4 ARCHITECTURE OF PROPOSED SYSTEM	18
	4.4.1 Circuit diagram for hardware implementation	19
	4.5 MODULES IN PROPOSED WORK	20
	4.5.1 Camera Feed Fetching Module	20
	4.5.2 Dynamic Signal Control Module	20
	4.5.3 Vehicle Counting Module	21
	4.5.4 Helmet Detection Module	21
	4.5.5 Emergency Vehicle Detection Module	21
	4.5.6 Parking Availability Display Module	21
	4.5.7 Data Storage Module (MongoDB Integration)	22

	4.6 SUMMARY	22
5	EXPERIMENTAL RESULTS	
	5.1 OVERVIEW	23
	5.2 TOOLS/PLATFORM	23
	5.2.1 Hardware Requirements	23
	5.2.2 Software Requirements	23
	5.2.3 Metrics for evaluation	24
	5.3 DATASET	24
	5.3.1 Input	24
	5.3.2 Output	24
	5.4 EXPERIMENTAL SETUP	24
	5.4.1 Server program	24
	5.4.2 Arduino program	30
	5.4.3 Experimental setup prototype	33
	5.5 RESULTS AND GRAPHS	34
	5.5.1 Arduino output	34
	5.5.2 Functionalities output	35
	5.5.3 API latency	38
	5.5.4 MongoDB Atlas output	40
	5.5.5 Time complexity of functions	41
	5.6 SUMMARY	42
6	CONCLUSION AND FUTURE ENHANCEMENTS	
	6.1 CONCLUSION	43
	6.2 FUTURE ENHANCEMENTS	43
	REFERENCES	45

LIST OF FIGURES

Figure No	Title	Page No
3.1	Existing block diagram	13
4.1	Architecture diagram of proposed work	19
4.2	Circuit diagram	19
5.1	Development server program	24
5.2	Thread code	25
5.3	Calculate traffic density code	26
5.4	Check object within object	26
5.5	Count number of vehicles per hour	27
5.6	Emergency vehicle detection code	28
5.7	Helmet detection code	28
5.8	Car parking availability code	29
5.9	Development server output	29
5.10	Pin mode setup code	30
5.11	Loop program	31
5.12	Sending camera data to API	31
5.13	LED light activation code	32
5.14	Prototype front view	33
5.15	Prototype top view	33
5.16	Prototype of traffic light	34
5.17	Arduino serial monitor output	34
5.18	Counting number of vehicles	35
5.19	Helmet Detected	36
5.20	Helmet not detected	36
5.21	Vehicle counting by crossing the line on a lane	37
5.22	Emergency Vehicle detection	37
5.23	Parking slot availability output	38
5.24	Latency vs Resolution	39
5.25	MongoDB Atlas Collections	39
5.26	Heatmap of peak hours	40
5.27	Traffic Density in different places	40

LIST OF TABLES

Table No	Title	Page No
2.1	Literature survey	8
5.1	Resolution and total time taken for API response	38

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
API	Application Programming Interface
CCTV	Closed-Circuit Television
CMS	Cloud Management System
CNN	Convolutional Neural Network
CV	Computer Vision
ESP32	Espressif Systems' microcontroller
FPS	Frames Per Second
FSM	Finite State Machine
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
ROI	Region of Interest
RTSP	Real-Time Streaming Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TMS	Traffic Management System

CHAPTER I

INTRODUCTION

1.1 OVERVIEW

The cloud-based smart traffic management system proposed in this report utilizes advanced video analytics and machine learning techniques to optimize traffic signals in real-time. By dynamically adjusting signal timings based on current traffic conditions, the system aims to alleviate urban traffic congestion, reduce commute times, minimize fuel wastage and emissions, and enhance overall road safety. This report outlines the objectives, motivation, and necessity of implementing such a system to address the challenges posed by urban traffic congestion.

The proposed project aims to address the challenges posed by urban traffic congestion through the development of a smart traffic management system. This system will utilize cloud-based infrastructure to dynamically optimize traffic signal timings based on real-time traffic conditions. Additionally, it will incorporate advanced features such as emergency vehicle detection, helmet detection, and to enhance road safety and security.

1.2 OBJECTIVE OF THE STUDY

1. Develop a cloud-based smart traffic management system capable of dynamically optimizing traffic signal timings based on real-time traffic data.
2. Implement advanced detection algorithms for identifying emergency vehicles, detecting helmet usage.
3. Enhance road safety and security by prioritizing emergency vehicles and enforcing traffic regulations through real-time monitoring and enforcement measures.
4. Provide a scalable and accessible solution by hosting the system on cloud infrastructure, allowing easy access and deployment across urban areas.

5. Evaluate the effectiveness and performance of the system through thorough testing and validation in real-world traffic scenarios, ensuring its reliability and efficacy in mitigating traffic congestion and improving overall traffic management.

1.3 MOTIVATION/ NEED FOR THE STUDY

The motivation behind this project stems from the existing inefficiencies of traditional traffic light systems, which often have fixed timings that do not adapt to real-time traffic fluctuations. This leads to increased congestion, longer waiting times for vehicles, and unnecessary delays, particularly in lanes with lower traffic density. By implementing dynamic traffic signal timing and incorporating advanced detection capabilities, the proposed system seeks to alleviate these issues and improve overall traffic flow and safety.

The motivation for a cloud-based traffic management system stems from the pressing need to address urban traffic congestion efficiently. By leveraging scalable cloud infrastructure, such a system can dynamically adapt to real-time traffic conditions, optimizing signal timings and improving traffic flow. With the ability to process vast amounts of data in real-time, cloud-based solutions offer accessibility, integration, and cost-effectiveness, enabling seamless deployment and operation. Ultimately, the goal is to reduce congestion, enhance road safety, and provide efficient traffic management solutions tailored to the dynamic nature of urban traffic.

1.4 ORGANIZATION OF THE CHAPTERS

Chapter I gives the general introduction of the project, overview of the project, objectives and motivation of the study.

Chapter II deals with the literature survey that is done to support the project and regarding the various machine learning algorithms, to choose the best for proposal.

Chapter III discusses the existing system architecture, modules and description for the individual modules involved in the work and limitations of the system.

Chapter IV explains the proposed system. It consists of the architecture design, modules, its description for the individual modules involved in the work and the model summary.

Chapter V illustrates the tools used for the implementation, dataset details and snapshots of existing and proposed system implementations.

Chapter VI gives a brief summary of the work done and presents avenue for future enhancement work that can be carried over.

CHAPTER II

LITERATURE REVIEW

2.1 TECHNIQUES

2.1.1 Traffic control system design [1]

The traffic control system design outlined in the research aims to revolutionize traffic management by incorporating innovative strategies tailored to enhance traffic flow efficiency. At its core, the system prioritizes factors such as traffic volume, vehicle priority, and real-time traffic variations to adapt dynamically to changing traffic conditions. This adaptability is achieved by inhibiting multi-phase operations, ensuring that the system can minimize overall travel time and waiting time for vehicles. The system's architecture involves three key components: a controller agent responsible for data acquisition and scheduling, a traffic light agent that monitors vehicle presence and weight, and a vehicle agent that determines priority values based on lane flow and movement interference. By utilizing a fixed cycle approach with specific green and red light durations, the system operates seamlessly without the need for a traditional controller, thereby optimizing traffic flow efficiency across intersections. Simulation results corroborate the system's effectiveness, showcasing a substantial reduction in average waiting times for vehicles, thereby underscoring the system's potential to alleviate congestion and enhance overall traffic management.

- Revolutionizes traffic management through innovative strategies.
- Adapts dynamically to changing traffic conditions.
- Minimizes overall travel time and waiting time for vehicles.
- Architecture includes controller, traffic light, and vehicle agents.
- Utilizes fixed cycle approach for optimized traffic flow efficiency.
- Simulation results demonstrate substantial reduction in average waiting times.
- Shows potential to alleviate congestion and enhance overall traffic management.

2.1.2 Object detection techniques [6]

Object detection techniques play a pivotal role in traffic management systems, facilitating the identification and tracking of vehicles in traffic scenes. Background subtraction serves as a fundamental technique for vehicle detection by distinguishing foreground objects from the background, enabling accurate vehicle detection and tracking. The Haar Cascade Classifier is utilized for object detection in images, employing machine learning-based approaches to detect predefined objects, including vehicles, based on their distinctive features. Meanwhile, the YOLO (You Only Look Once) algorithm offers real-time object detection capabilities, allowing rapid detection of vehicles and other objects in traffic scenes. Additionally, the EfficientDet model emerges as the optimal algorithm for traffic density estimation, providing high accuracy in detecting and estimating traffic density in real-time video feeds.

- Various techniques like background subtraction and YOLO algorithm facilitate real-time vehicle detection.
- EfficientDet model excels in traffic density estimation.
- Ensures accurate and efficient traffic monitoring and management.

2.1.3 Advanced traffic analysis techniques

Advanced traffic analysis techniques leverage cutting-edge technologies to enhance traffic management strategies. Convolutional Neural Networks (CNNs) are instrumental in identifying congestion patterns and managing traffic flow efficiently by extracting features from traffic images. Deep CNN models further optimize traffic management by accurately counting vehicles based on video imagery, improving accuracy and efficiency in vehicle counting tasks. Moreover, these models automatically detect movement patterns in traffic scenes, enabling insights into traffic behavior and informing more effective traffic management strategies.

- Employs advanced techniques for comprehensive traffic analysis.
- CNNs and deep CNN models improve accuracy in traffic management.

- Theme models provide insights into traffic behavior.
- Enhances overall traffic management and planning.

2.1.4 Marker and camera pose estimation [12]

The study employs advanced techniques for marker and camera pose estimation, crucial for accurate spatial analysis in traffic scenes. A method for automatic 3D estimation is utilized, which leverages synchronized video sequences and global non-linear optimization for precise marker and camera pose estimation. By analyzing the 3D structure of planar markers and camera poses, this technique enables the generation of accurate spatial models essential for traffic analysis and surveillance.

- Utilizes automatic 3D estimation for precise spatial analysis.
- Enables accurate marker and camera pose estimation.
- Crucial for generating precise spatial models for traffic analysis.

2.1.5 Comparative evaluation of computer vision libraries [2]

The comparative evaluation of computer vision libraries provides valuable insights into their performance and suitability for traffic-related tasks. Evaluation metrics such as detection accuracy, processing time, and response to image quality data are employed to assess the performance of ImageAI, Cloud Vision, and OpenCV. Through experimentation with sample images, these libraries are tested for their ability to detect vehicles accurately and efficiently, informing the selection of appropriate libraries for traffic management applications.

- Evaluation metrics include detection accuracy and processing time.
- Guides selection of libraries for efficient traffic management.
- Ensures accurate and reliable vehicle detection.

2.1.6 Traffic simulator development [2]

Traffic simulator development represents a crucial aspect of traffic research, enabling the analysis and optimization of traffic management strategies in simulated environments. Integration of computer vision and fuzzy logic modules in the development of a traffic simulator facilitates the simulation of complex traffic control systems. The implementation of an actuated traffic light system, integrated with computer vision and fuzzy logic, demonstrates promising results in enhancing traffic management and reducing congestion by allowing more vehicles to pass and optimizing amber time for all lanes.

- Integrates computer vision and fuzzy logic for complex simulations.
- Actuated traffic lights improve traffic management.
- Shows promise in optimizing traffic flow efficiency.

2.1.7 Image processing techniques for noise estimation [3]

Image processing techniques play a vital role in noise estimation for traffic monitoring and analysis. Background subtraction, dilation, erosion, and contour operations are employed to isolate moving objects in video frames, facilitating accurate vehicle detection. Furthermore, noise estimation procedures are developed and validated using video data, providing a modular approach for noise assessment in road traffic environments. These techniques contribute to the accurate analysis of traffic noise levels, essential for informed decision-making in urban planning and noise mitigation efforts.

- Validates procedures using video data.
- Provides modular approach for noise assessment.
- Essential for informed urban planning and noise mitigation.

2.2 SURVEY OF THE RELATED WORKS

Table 2.1 Literature survey

SI. NO	TITLE & REFERENCES	ALGORITHM/ TECHNIQUES USED	DATASETS	LIMITATIONS	ADVANTAGES
1.	A Comparative Analysis of Computer Vision Libraries in the Context of a Jakarta Traffic Simulator [2]	Computer Vision Fuzzy Logic Machine Learning Image Processing	CCTV video and images of Jakarta traffic used to evaluate computer vision. Simulated vehicle data at intersection for traffic light controller tests No public datasets used	Small sample size Simulated data only Basic testing scenario Lack of model details No integration with existing systems Proof of concept only	Enhanced traffic flow, precise vehicle detection, and automated decision-making are the key advantages of the study.
2.	Real-time traffic control and monitoring [1]	Background subtraction Haar cascade classifier YOLO algorithm EfficientDet architecture TensorFlow lite	Custom dataset and online sources & proprietary images were used to form the dataset for training the EfficientDet-Lite 0 model.	User decisions for traffic congestion, the need for VANET implementation on appropriate hardware, and the difficulty of installing it on motorcycles.	EfficientDet provides accurate vehicle detection, traffic light synchronization reduces stops, adaptive signal control system optimizes timing, CCTV cameras

					monitor traffic economically.
3.	Simultaneous Multi-View Camera Pose Estimation and Object Tracking With Squared Planar Markers [12]	Levenberg-Marquardt algorithm OpenMP API Rodrigues' formula	Data obtained from synchronized global-shutter cameras and an Optitrack motion capture system for tracking the position of reflective markers	Challenges in estimating rigid structures of marker sets, relative camera pose, and effective working area when tracking one marker with only one camera.	Simultaneously solves camera pose estimation and object tracking, Achieves high accuracy with low-resolution cameras, Cost-effective solution for real-time object tracking.
4.	The Use of Cameras and Noise Estimation Models for Traffic Monitoring and Environmental Impact [3]	Background subtraction, Gaussian blur, threshold segmentation method, morphological operations, dynamic microscopic noise models, and probe	Confidential video records for traffic monitoring and noise estimation.	The lack of validation in different weather conditions, potential challenges in poor light conditions. Additionally, the study was tested on a limited number of video records in a	Accuracy Adaptive Cost effective

		vehicles equipped with OBD and distance sensors for traffic monitoring and noise estimation.		highway environment.	
--	--	--	--	----------------------	--

2.3 LIMITATIONS OF THE EXISTING WORKS

2.3.1 Implementation challenges of VANET

The proposed smart traffic control system faces a significant barrier to real-world adoption: implementing VANET (Vehicle Ad-hoc Network) technology. Equipping all vehicles, particularly motorcycles due to installation challenges, with appropriate hardware presents a major hurdle.

2.3.2 Scalability and real-world implementation

The proposed smart traffic control system, while innovative, needs further exploration for real-world use. The research doesn't detail how it would scale up or be implemented in practice, and there's no discussion of challenges transitioning from simulations to actual roads. Further testing under diverse traffic conditions is crucial to assess its effectiveness for large-scale adoption.

2.3.3 Methodological limitations

The study's method relies on synchronized video sequences, which may pose challenges in real-time applications due to computational requirements and potential synchronization issues. Environmental factors such as lighting conditions and occlusions may influence the method's accuracy and performance, impacting its reliability in

practical settings. Further validation in diverse real-world scenarios is necessary to assess the method's robustness and generalizability across different tracking conditions.

2.3.4 Limitations in data evaluation

The video-based traffic noise estimation method has limitations that affect its real-world applicability. The findings might not generalize well since the computer vision libraries were only tested on Jakarta traffic data. Additionally, the simulations were conducted in a simple intersection setting, not reflecting real-world complexities like multi-lane roads or diverse vehicles. Finally, manual vehicle counting can introduce human error, impacting the accuracy of the method.

2.3.5 Limitations in noise estimation procedure

The video-based traffic noise estimation method shows promise, but limitations exist. Focusing only on light/heavy vehicles due to model restrictions and relying on video analysis may hinder accuracy in complex environments with diverse traffic, pedestrians, and cyclists.

2.3.6 Limitations of smart mobility procedure

The proposed smart mobility procedure for traffic noise estimation shows promise, but its real-world applicability needs further exploration. The testing conditions might limit its robustness in various environments. Poor lighting and nighttime situations could lead to inaccurate noise estimation. Additionally, the categorization of vehicles only covered light and heavy-duty types. This excludes a wider range of moving objects, such as pedestrians and cyclists, which could affect the procedure's effectiveness in diverse urban settings.

2.4 SUMMARY

The study explores innovative traffic signal simulation and object detection using real-time video feeds and the EfficientDet model. It highlights the model's exceptional performance metrics, particularly in vehicle detection and emergency vehicle prioritization. The research proposes adaptable traffic control systems based on image processing and computer vision to address challenges posed by escalating road users and constrained infrastructure resources. It evaluates three computer vision libraries, with ImageAI emerging as the top-performing library. It recommends further testing scenarios, collaboration with traffic authorities, and advanced techniques to enhance detection and decision-making capabilities. This study also presents a smart mobility procedure for estimating traffic noise levels through roadside video recordings.

CHAPTER III

EXISTING WORK

3.1 OVERVIEW OF THE EXISTING WORK [2]

The existing system aims to control traffic signals and monitor traffic density in real-time using image processing and machine learning techniques. It utilizes live video feeds from cameras installed at intersections, processes the frames to detect and count vehicles, and adjusts the traffic light timings accordingly. Additionally, it prioritizes emergency vehicles, such as ambulances and fire brigades, by switching the signal lights to green for their lanes.

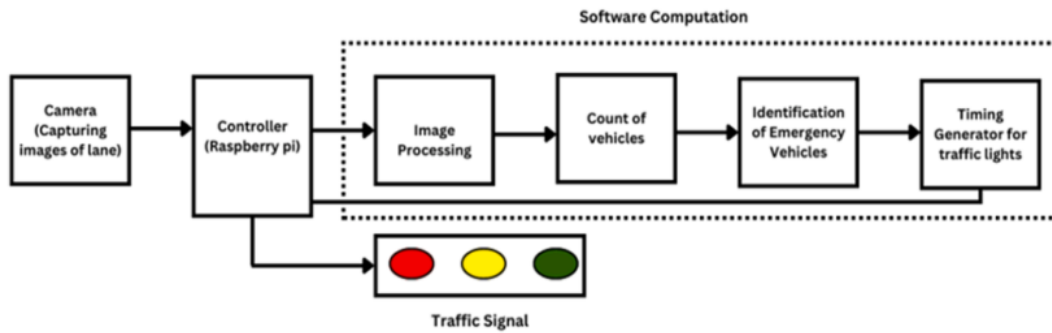


Figure 3.1 Existing Block Diagram

In figure 3.1 the system uses a Raspberry Pi as the microcontroller, a Pi Camera for capturing video feed, and an Efficient Det algorithm along with TensorFlow Lite for object detection and vehicle classification. It processes the video frames using OpenCV and Python, makes traffic control decisions based on vehicle count and emergency vehicle detection, and controls the traffic signal LEDs (red, yellow, green) accordingly.

3.2 TECHNIQUES USED

1. Image Acquisition: Capturing video frames from Pi cameras installed at intersections.
2. Image Processing: Processing the acquired frames using OpenCV library and Python for object detection and vehicle counting.

3. **Vehicle Detection and Counting:** Efficient Det-Lite 0 model, a deep learning object detection architecture, is used for detecting and counting vehicles and emergency vehicles on each lane.
4. **Traffic Density Estimation:** The vehicle count on each lane is used to estimate the traffic density.
5. **Traffic Signal Control:** The traffic light timings are adjusted based on the estimated traffic density and the presence of emergency vehicles on any lane.

3.3 MODULE DESCRIPTION

The existing system consists of the following modules:

1. Image Acquisition Module
2. Image Processing Module
3. Vehicle count and Emergency Vehicle Detection Module
4. Traffic Signal Control Module

3.3.1 Image acquisition module

The real-time traffic control and monitoring system consists of several modules that work together to achieve its functionality. The Image Acquisition Module is responsible for capturing and retrieving the video feed from the Pi camera installed at the intersection. It handles camera initialization, configuration, and continuous frame capture, storing the frames in a buffer or queue for further processing. This module may also include functionalities for camera calibration, image rectification, and distortion correction, as well as synchronization mechanisms to ensure proper time-stamping and ordering of the captured frames.

3.3.2 Image processing module

The Image Processing Module retrieves the frames from the Image Acquisition Module's buffer or queue and applies various image processing techniques to enhance the image quality and extract relevant information. Common techniques used in this module

may include noise reduction, colour space conversion, edge detection, and region of interest (ROI) extraction. Background subtraction algorithms may be employed to separate moving objects (vehicles) from the static background, and object tracking algorithms can be used to maintain the identities and trajectories of detected vehicles across consecutive frames. This module may also perform perspective transformations to convert the camera's view to a top-down or bird's-eye view, simplifying vehicle detection and counting.

3.3.3 Vehicle count and emergency vehicle detection module

The Vehicle Count, Traffic Density Determination, and Emergency Vehicle Detection Module receives the processed frames and relevant information from the Image Processing Module. It is responsible for running the Efficient Det-Lite 0 model, a deep learning object detection architecture optimized for resource-constrained devices like the Raspberry Pi. The Efficient Det-Lite 0 model has been pre-trained on a custom dataset containing images of vehicles, including emergency vehicles (ambulances and fire brigades). This module feeds the processed frames into the Efficient Det-Lite 0 model, which outputs bounding boxes, confidence scores, and labels (vehicle types) for the detected objects. Based on the detected objects and their labels, the module counts the number of vehicles in each lane and classifies them as emergency or non-emergency vehicles. It calculates the traffic density for each lane by considering the vehicle count and lane dimensions (if available). The module may also employ techniques to track vehicles across consecutive frames, enabling more accurate counting and density estimation. It outputs the traffic density information and the presence of emergency vehicles for each lane.

3.3.4 Traffic signal control module

The Traffic Signal Control Module receives the traffic density information and emergency vehicle presence from the previous module. It implements the logic and decision-making algorithms to adjust the traffic light timings based on the input data. If an emergency vehicle is detected on a particular lane, the module assigns the highest priority to that lane and switches the traffic signal to green for that lane until the emergency vehicle crosses the intersection. If no emergency vehicles are present, the

module allocates longer green light durations to the lanes with higher traffic density, while lanes with lower density receive shorter green light durations. The module may employ optimization techniques to determine the optimal traffic light timings based on the current traffic conditions. It interfaces with the hardware components (e.g., LEDs, relays) that represent the traffic signals, sending commands to switch the lights accordingly. The module may also include functionalities for logging and reporting the traffic signal timings, vehicle counts, and other relevant data for analysis and monitoring purposes.

3.4 LIMITATIONS OF THE EXISTING WORK

1. The system relies on the accuracy of the object detection model, which may be affected by environmental conditions, occlusions, or variations in vehicle appearances.
2. The performance of the system may be limited by the computational power of the Raspberry Pi, especially for real-time processing of high-resolution video feeds.
3. The system does not consider other factors that may affect traffic flow, such as pedestrian movements, road conditions, or construction zones.
4. The document does not provide information about the system's robustness, scalability, or adaptability to different intersection layouts or traffic patterns.

3.5 SUMMARY

Overall, the existing system leverages computer vision and machine learning techniques to intelligently control traffic signals and prioritize emergency vehicles, aiming to improve traffic flow and reduce congestion. However, its real-world performance and effectiveness may depend on various factors, including the accuracy of the object detection model, computational resources, and the complexity of the traffic scenarios.

CHAPTER IV

PROPOSED WORK

4.1 OVERVIEW

Urban traffic congestion leads to economic losses, environmental issues, and poor quality of life. Existing traffic management systems with fixed signal timings are inefficient in handling fluctuating real-time traffic flows. Intelligent systems that can dynamically optimize signals based on traffic conditions are needed. This project proposes a cloud-based smart traffic management system using advanced video analytics for real-time traffic optimization. The system aims to reduce congestion, shorten commute times, decrease fuel wastage, lower emissions, and improve road safety.

4.2 PROBLEM STATEMENT

High vehicle density in urban areas often results in prolonged waiting times for vehicles at traffic signals, exacerbating congestion issues. Traditional static signal timings are inefficient, particularly for lanes with lower traffic density, leading to unnecessary delays. To address these challenges, dynamic signal timing systems are essential, adapting in real-time to traffic conditions. Moreover, prioritizing lanes for emergency vehicles is crucial for enhancing road safety and response times. By leveraging cloud-based solutions, such as intelligent traffic management systems, accessibility and scalability are improved, offering an efficient and easily accessible service to optimize traffic flow and minimize congestion.

4.3 PROPOSED SOLUTION

The cloud-based smart traffic management system aims to optimize traffic signal timings dynamically by analyzing real-time traffic density, thereby mitigating congestion. This involves extending green light durations for lanes with higher vehicle counts to improve traffic flow. Additionally, the system prioritizes emergency vehicles by preempting signals to green when detected, enhancing response times and potentially

saving lives. It also generates alerts for traffic violations such as red-light crossings and facilitates helmet detection for improved safety measures. By minimizing commute times and environmental impact through smoother traffic flow, the system enhances overall efficiency. Furthermore, it delivers a scalable, robust, and highly available solution capable of expanding to cover large urban areas.

4.4 ARCHITECTURE OF PROPOSED SYSTEM

The figure 4.1 depicts an intelligent traffic management system that integrates hardware and software components to monitor and control traffic flow dynamically. At the core of the system are two cameras, Camera 1 and Camera 2, which capture video frames from different traffic intersections. These video frames are processed in parallel by two separate threads, Thread 1 and Thread 2, respectively. The system utilizes an ESP32 microcontroller to control the traffic signals at the intersections. This microcontroller communicates with a Flask web framework-based server and API, sending requests with a "required_value" parameter to obtain optimal signal timing information based on real-time traffic conditions. The Flask Servers and API respond with vehicle count data or other relevant traffic information as an integer value. This data is used to update two global variables, "traffic_density_1" and "traffic_density_2," which represent the traffic density at the respective locations. The threads also perform various image processing and analysis functions on the video frames, such as vehicle counting, helmet detection, emergency vehicle detection, and hourly vehicle counting. Based on the updated traffic density information in the global variables, the system outputs time delay values for the green signal span to the ESP32 microcontroller, enabling dynamic adjustment of traffic signal timings. Additionally, essential data like traffic density, vehicle counts, and other relevant information are stored in a MongoDB cloud database for further analysis, record-keeping, or other purposes.

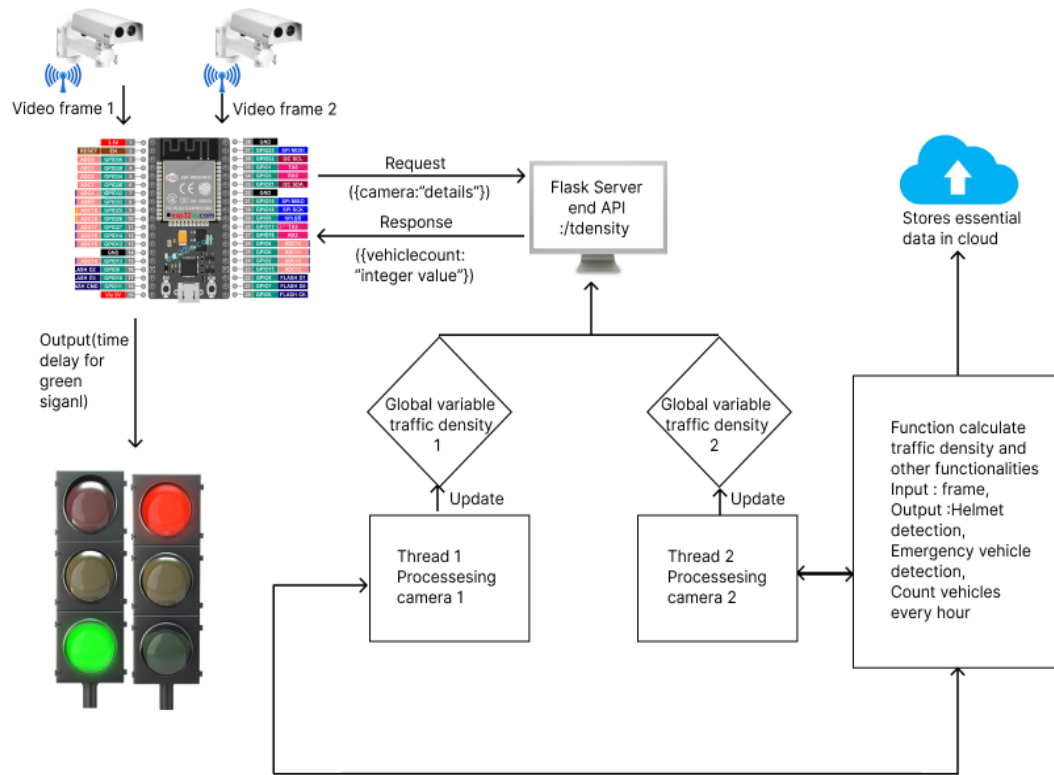


Figure 4.1 Architecture diagram of proposed work

4.4.1 Circuit diagram for hardware implementation

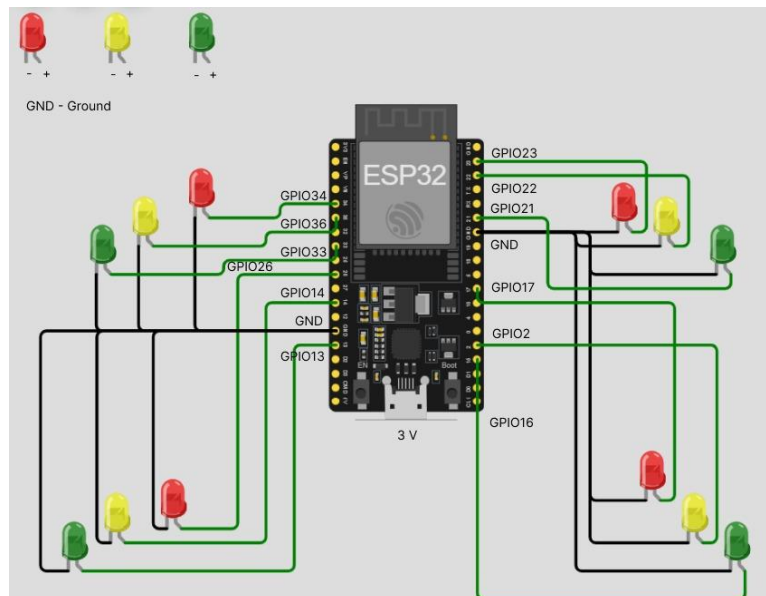


Figure 4.2 Circuit diagram

The circuit diagram (figure 4.2) is a LED control system using an ESP32 microcontroller, utilizing four GPIO pins: Green1 (GPIO 33), Green2 (GPIO 13), Green3 (GPIO 16), Green4 (GPIO 21), Red1 (GPIO 34), Red2 (GPIO 26), Red3 (GPIO 21), Red4 (GPIO 23), Yellow1 (GPIO 36), Yellow2 (GPIO 14), Yellow3 (GPIO 2) and Yellow4 (GPIO 22). These GPIO pins were configured as outputs to control corresponding LEDs. Each LED was connected in series with a current-limiting resistor and grounded to the ESP32's ground pin (GND), forming a complete circuit. The system allowed for individual control of each LED by toggling the respective GPIO pin HIGH or LOW, effectively turning the LED on or off. The project implementation included proper code configuration of the GPIO pins in the ESP32 firmware to ensure accurate LED control. Through this setup, the project demonstrated the versatility and flexibility of the ESP32 microcontroller in managing peripheral devices for various applications.

4.5 MODULES IN PROPOSED WORK

The proposed system consists of the following modules:

1. Camera Feed Fetching Module
2. Dynamic Signal Control Module
3. Vehicle Counting Module
4. Helmet Detection Module
5. Emergency Vehicle Detection Module
6. Parking Availability Display Module
7. Data Storage Module (MongoDB Integration)

4.5.1 Camera Feed Fetching Module

This module is responsible for fetching live camera feeds from designated traffic cameras installed at various intersections. It runs as a separate thread on the server, continuously capturing video streams in real-time. The module utilizes OpenCV library for video processing and streaming, ensuring seamless retrieval of camera data.

4.5.2 Dynamic Signal Control Module

The Dynamic Signal Control Module is integrated with the ESP32 Arduino module, which connects to the traffic signal controllers. It receives requests from the ESP32 module regarding the duration of green signals and adjusts signal timings dynamically based on real-time traffic conditions. Additionally, it detects emergency vehicles and prioritizes their lanes by extending the green signal duration to facilitate swift passage.

4.5.3 Vehicle Counting Module

This module is responsible for counting the number of vehicles passing through each lane at traffic intersections. It utilizes image processing techniques to detect and track vehicles in the camera feed. By analyzing the movement and size of detected objects, the module accurately counts the vehicles and provides real-time updates on traffic density to the Dynamic Signal Control Module.

4.5.4 Helmet Detection Module

The Helmet Detection Module employs Haar cascade classifiers to detect the presence of helmets on riders or passengers of two-wheeled vehicles. It analyzes the video feed from the traffic cameras, identifying regions of interest corresponding to human heads, and applies the trained Haar cascade classifier to detect helmets. This module contributes to enhancing road safety by enforcing helmet-wearing compliance.

4.5.5 Emergency Vehicle Detection Module

Utilizing TensorFlow for object detection, the Emergency Vehicle Detection Module identifies emergency vehicles such as ambulances, fire trucks, and police cars in the live camera feed. It employs pre-trained deep learning models to recognize specific vehicle types and alerts the Dynamic Signal Control Module to prioritize their lanes during emergencies, ensuring rapid response and passage.

4.5.6 Parking Availability Display Module

This module analyzes live camera feeds from designated parking areas to determine parking availability. It sets a region of interest which is rectangle and check any contours are within ROI then it is occupied space. The module then displays real-time updates on parking availability through a user interface, enabling drivers to locate empty parking spots conveniently.

4.5.7 Data Storage Module (MongoDB Integration)

The Data Storage Module is responsible for persistently storing the collected traffic data in the MongoDB cloud database. It establishes a connection with the MongoDB server and stores various types of data, including vehicle counts, signal timings, emergency vehicle detections, helmet violation instances, and parking availability status. This module ensures that all traffic-related information is securely stored and easily accessible for future analysis and decision-making processes. By leveraging MongoDB's scalability and flexibility, the system can handle large volumes of data efficiently and support seamless integration with other modules for data retrieval and analysis.

4.6 SUMMARY

The proposed smart traffic management system integrates modules like Camera Feed Fetching, Signal Control, Vehicle Counting, Emergency Vehicle Detection, Helmet Violation Detection, Parking Availability Monitoring, and Data Storage (MongoDB Integration). These modules collaborate to optimize traffic flow, prioritize emergency vehicles, detect violations, and monitor parking. Leveraging threading, ESP32 Arduino, TensorFlow, and MongoDB, the system efficiently handles data collection, analysis, and management, offering a scalable and adaptable solution to urban traffic congestion.

CHAPTER V

EXPERIMENTAL RESULTS

5.1 OVERVIEW

This chapter outlines the hardware and software requirements for the intelligent traffic management system, such as an Intel Core i5 or higher processor, 8GB+ RAM, ESP32 microcontroller, Windows/Linux OS, Python, C/C++, Flask, OpenCV, PyTorch, YOLOv5, MongoDB Atlas, and IDEs like Arduino IDE and Visual Studio Code. It specifies real-time video feeds as input and expected outputs like dynamic timing values, vehicle counts, helmet violation images, emergency vehicle presence, and parking availability data. The chapter also mentions defining evaluation metrics and providing details on the experimental setup's server program implementation. Overall, it covers the tools, platforms, datasets, and setup required for developing the cloud based traffic management system.

5.2 TOOLS/PLATFORM

5.2.1 Hardware Requirements

System	:	Intel Core i5 or higher.
Hard Disk	:	500GB.
Monitor	:	15" LED or larger.
Input devices	:	Keyboard, Mouse, IP webcam.
RAM	:	8 GB or more.
Microcontroller	:	ESP32 WROOM Development board (input 3V).
LEDs	:	Red, Green (or traffic signal lights if available).

5.2.2 Software Requirements

Operating System	:	Windows 10 or Linux distributions (Ubuntu or Mint).
Coding Languages	:	Python 3.10 , C/C++ .
Web Framework	:	Flask.
Python packages	:	OpenCV, Pygame, PyTorch, YOLOv5, Ultralytics.
Database	:	MongoDB Atlas (cloud-based) and MongoDB Compass (local GUI client).

API testing : Postman.
IDEs : Arduino IDE (for microcontroller programming), Visual Studio Code (for python programming).

5.2.3 Metrics for evaluation

The primary evaluation metric utilized in this project is latency and Time complexity of functions.

Latency = Response time – Request time

5.3 DATASET

5.3.1 Input

Real time video feed.

5.3.2 Output

- Dynamic timing values in JSON format.
- Count of vehicles in a lane for every hour.
- Helmet rule violation images in png or jpg format.
- Presence of emergency vehicle in JSON format (true/false).
- Availability of parking place count and avail to public with live video feed.

5.4 Experimental setup

5.4.1 Server program

```
if __name__ == '__main__':  
    thread1 = threading.Thread(target=video_capture_thread1)  
    thread2 = threading.Thread(target=video_capture_thread2)  
    thread1.start()  
    thread2.start()  
  
    app.run(debug=True, host='0.0.0.0')
```

Figure 5.1 Development server program

The Figure 5.1 shows the output when running a Flask application named "newtrytwothread" in development mode. It shows the application running on multiple IP addresses and ports, and provides a warning about using a production WSGI server instead for production deployments. And contains code that creates two threads, thread1 and thread2, targeting video_capture_thread1 and video_capture_thread2 functions respectively. It then starts both threads and runs the Flask application in debug mode on the host '0.0.0.0'.

```
def video_capture_thread1():
    global current_frame1, traffic_density1, stop_threads, video_capture1
    video_capture1 = cv2.VideoCapture(vd1)
    print(1)
    while stop_threads != True:
        ret, frame = video_capture1.read()
        if not ret:
            print("Error: Could not read frame from camera 1.")
            break

        # Calculate traffic density for camera 1
        density = calculate_traffic_density(frame, roi_vertices)

        # Acquire lock before updating shared resources
        with lock1:
            current_frame1 = frame.copy()
            traffic_density1 = density
    video_capture1.release()
```

Figure 5.2 Thread code

The Figure 5.2 shows the function `video_capture_thread1` captures frames from camera 1, calculates traffic density using `calculate_traffic_density`, updates shared resources, and releases the video capture likewise thread2 and so on if needed.

```

def calculate_traffic_density(img, roi_vertices):
    if type(img) == str:
        #img = cv2.imread(image_path)
        # Convert the image to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # Apply GaussianBlur to reduce noise
        blurred = cv2.GaussianBlur(gray, (5, 5), 0)
        # Apply edge detection using Canny
        edges = cv2.Canny(blurred, 50, 150)
        # Create a mask with the region of interest (ROI)
        mask = np.zeros_like(edges)
        cv2.fillPoly(mask, [roi_vertices], 255)
        # Apply the mask to the edges
        masked_edges = cv2.bitwise_and(edges, mask)
        # Find contours in the masked edges
        contours, _ = cv2.findContours(masked_edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        # Initialize a variable to count vehicles
        vehicle_count = 0
        # Iterate through the contours
        for contour in contours:
            area = cv2.contourArea(contour)

            # Filter out small contours to reduce false positives
            if area > 200:
                # Draw bounding box around the detected object (vehicle)
                x, y, w, h = cv2.boundingRect(contour)
                cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
                # Increment the vehicle count
                vehicle_count += 1

    return vehicle_count

```

Figure 5.3 Calculate traffic density code

In Figure 5.3, the function `calculate_traffic_density` analyzes an image to estimate traffic density. It processes the image by converting to grayscale, applying Gaussian blur and Canny edge detection, then identifies vehicles within a defined region of interest. For upcoming codes, follow this methodology.

```

def is_inside(inner_rect, outer_rect):
    x1_inner, y1_inner, w_inner, h_inner = inner_rect
    x1_outer, y1_outer, w_outer, h_outer = outer_rect

    # Get coordinates of inner rectangle corners
    x2_inner = x1_inner + w_inner
    y2_inner = y1_inner + h_inner

    # Check if all corners are inside the outer rectangle
    return (x1_outer <= x1_inner <= x2_inner <= x1_outer + w_outer and
            y1_outer <= y1_inner <= y2_inner <= y1_outer + h_outer)

```

Figure 5.4 Check object within object

In Figure 5.4, the function ``is_inside`` determines whether an inner rectangle is entirely contained within an outer rectangle. It takes two sets of coordinates and dimensions representing the inner and outer rectangles, respectively. It calculates the coordinates of the inner rectangle's corners and checks if all corners fall within the bounds of the outer rectangle. If so, it returns True; otherwise, it returns False.

```
# Object Detection
mask = object_detector.apply(frame)
_, mask = cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.line(frame, (498, 444), (659, 444), (0, 255, 0), 2)
for cnt in contours:
    # Calculate area and remove small elements
    area = cv2.contourArea(cnt)
    if area > 450:
        cv2.drawContours(frame, [cnt], -1, (0, 255, 0), 2)
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)
        if 498 <= x <= 659 and 444 <= y <= 446:
            count += 1
cv2.imshow("Frame", frame)
# Check if a minute has passed
current_time = datetime.now()
time_diff = (current_time - start_time).seconds
if time_diff >= 60:
    # Store data in MongoDB
    data = {
        "timestamp": current_time,
        "lane_number": lane_number,
        "place_name": place_name,
        "count": count
    }
    collection.insert_one(data)
# Reset count and timer
count = 0
start_time = datetime.now()
```

Figure 5.5 Count number of vehicles per hour

In Figure 5.5, the code segment processes a video frame to detect objects using a background subtraction-based object detector. It then applies a binary threshold to the mask, finds contours, draws contours and bounding boxes around detected objects, and counts objects crossing a specified line region. Detected objects are tracked over time, and counts are periodically stored in a MongoDB database.

```
def detect_ambulance(frame):
    # Load YOLOv5 model
    path = 'best.pt'
    model = torch.hub.load('ultralytics/yolov5', 'custom', path, force_reload=True)
    # Resize frame
    frame = cv2.resize(frame, (1020, 500))
    # Perform detection
    results = model(frame)
    # Check if ambulance is detected
    ambulance_detected = any(label == 'ambulance' for label in results.names)

    return ambulance_detected
```

Figure 5.6 Emergency vehicle detection code

In Figure 5.6, the code defines a function called `detect_ambulance` that loads a pre-trained YOLOv5 model, resizes the input frame, performs object detection on the frame, and checks if an ambulance is detected in the resulting predictions. It returns a boolean value indicating whether an ambulance was detected or not.

```
if helmet is not None:
    count += 1
    helmet_detected += 1

    helmet_height, helmet_width, _ = helmet.shape
    helmet_pixels = helmet_height * helmet_width
    total_pixels += helmet_pixels

    helmet = cv2.resize(helmet, (200, 200))
    helmet = cv2.cvtColor(helmet, cv2.COLOR_BGR2GRAY)

    # Save detected helmet
    helmet_file_path = 'C:\project\detectedimg\img' + str(count) + '.jpg'
    cv2.imwrite(helmet_file_path, helmet)

    cv2.putText(frame, str(count), (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 2)
    cv2.rectangle(frame, (helmet_coors[0], helmet_coors[1]), (helmet_coors[0]+helmet_coors[2], helmet_coors[1]+helmet_coors[3]), (0, 255, 0), 2)
    cv2.putText(frame, 'Helmet Detected', (helmet_coors[0], helmet_coors[1]-10), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
```

Figure 5.7 Helmet detection code

In Figure 5.7, the helmet detection code employs Haar cascades, a type of feature-based object detection algorithm, to identify helmets within video streams.

```

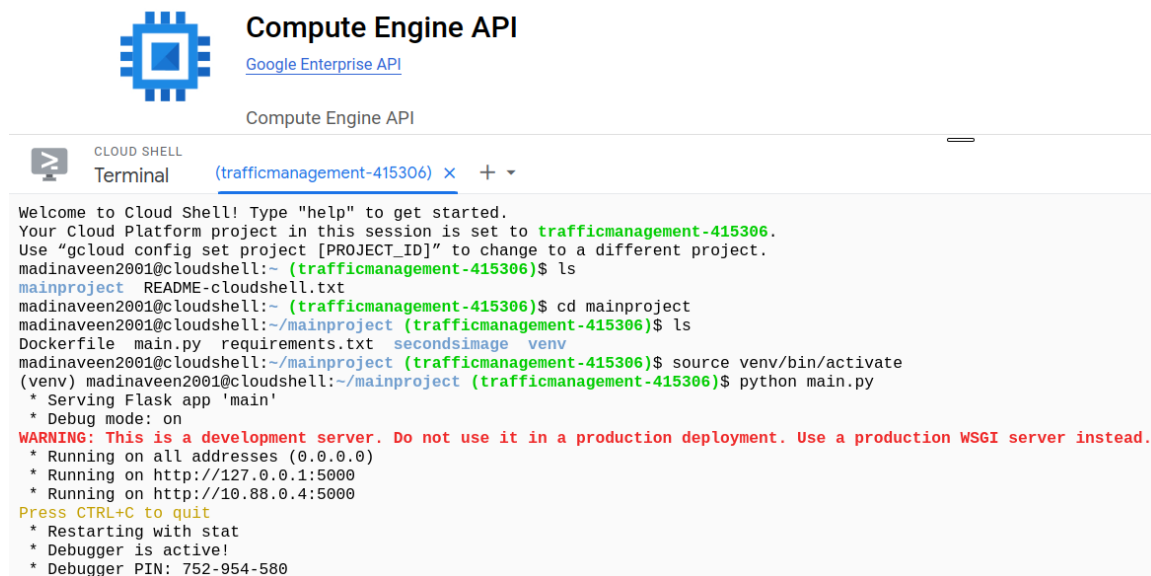
parking_areas = [np.array([[335, 263], [424, 263], [424, 87], [344, 85]]),
                  np.array([[249, 263], [333, 262], [333, 87], [253, 85]])]

# Function to count occupied and free parking spaces
def count_spaces(cars, parking_areas):
    occupied_spaces = 0
    for car_center in cars:
        for area in parking_areas:
            if cv2.pointPolygonTest(area, car_center, False) >= 0:
                occupied_spaces += 1
                break
    total_spaces = len(parking_areas)
    free_spaces = total_spaces - occupied_spaces
    return occupied_spaces, free_spaces

```

Figure 5.8 Car parking availability code

In Figure 5.8, the function `count_spaces` calculates the number of occupied and free parking spaces in a parking lot. It iterates through the centers of detected cars and checks if they fall within any of the defined parking areas using the point-in-polygon test. Based on the results, it computes the number of occupied and free spaces and returns these values.



Compute Engine API
[Google Enterprise API](#)

Compute Engine API

CLOUD SHELL
Terminal (trafficmanagement-415306) x + v

```

Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to trafficmanagement-415306.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
madinaveen2001@cloudshell:~ (trafficmanagement-415306) $ ls
mainproject  README-cloudshell.txt
madinaveen2001@cloudshell:~ (trafficmanagement-415306) $ cd mainproject
madinaveen2001@cloudshell:~/mainproject (trafficmanagement-415306) $ ls
Dockerfile  main.py  requirements.txt  secondsimage  venv
madinaveen2001@cloudshell:~/mainproject (trafficmanagement-415306) $ source venv/bin/activate
(venv) madinaveen2001@cloudshell:~/mainproject (trafficmanagement-415306) $ python main.py
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.88.0.4:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 752-954-580

```

Figure 5.9 Development server output

The Figure 5.9 shows a Google Cloud Shell terminal running a Flask development server for the "trafficmanagement" project. It displays commands for activating a virtual environment, and executing the "main.py" script.

5.4.2 Arduino program

```
const char *ssid = "Redmi Note 11T 5G";
const char *password = "neevannavee";
const char *apiEndpoint = "http://192.168.8.230:5000/tdensity";

const int greenLight1 = 21; // GPIO pin for green light 1
const int redLight1 = 4; // GPIO pin for red light 1
const int redLight2=12; //GPIO pin for redLight2
const int greenLight2=32; //GPIO pin for greenLight2

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

  pinMode(greenLight1, OUTPUT);
  pinMode(greenLight2, OUTPUT);
  pinMode(redLight1, OUTPUT);
  pinMode(redLight2, OUTPUT);

  connectToWiFi();
}
```

Figure 5.10 Pin mode setup code

In Figure 5.10, the code initializes variables for WiFi credentials (`ssid` and `password`) and an API endpoint (`apiEndpoint`). It also defines GPIO pins for controlling lights (`greenLight1`, `redLight1`, `redLight2`, `greenLight2`). The `setup()` function sets up serial communication and configures GPIO pins for output before establishing a WiFi connection.


```

void loop() {
    // Send camera 1 data to API
    int density1 = sendCamData(1);
    activateLight(greenLight1, density1, redLight2);
    // Wait for a while
    delay(1000);
    // Send camera 2 data to API
    int density2 = sendCamData(2);
    activateLight(greenLight2, density2, redLight1);
    // Wait for a while
    delay(500);
}

```

Figure 5.11 Loop program

The Figure 5.11 displays the code of arduino program of main loop that going to iterate infinitely in which it will make request of traffic density to server and get count of vehicles captured in the particular camera then control the traffic signal.

```

int sendCamData(int camNumber) {
    HTTPClient http;
    // Prepare JSON data
    String jsonData = "{\"camera\": " + String(camNumber) + "}";
    http.begin(apiEndpoint);
    http.addHeader("Content-Type", "application/json");
    int httpStatusCode = http.POST(jsonData);
    if (httpStatusCode > 0) {
        String payload = http.getString();
        DynamicJsonDocument jsonDoc(512); // the size of payload
        DeserializationError error = deserializeJson(jsonDoc, payload);
        // Check if parsing was successful
        if (error) {
            Serial.print("JSON parsing failed! Error: ");
            Serial.println(error.c_str());
            http.end();
            return 10; //if error return 10
        }
        // Extract the value of the "vehicle" key
        int vehicleValue = jsonDoc["vehicle"];
        Serial.print("cam Number,Vehicle Value: ");
        Serial.print(camNumber);
        Serial.println(vehicleValue);
        http.end();
        return vehicleValue;
    } else {
        Serial.printf("HTTP Request failed, error: %s\n", http.errorToString(httpStatusCode).c_str());
        Serial.println("Error");
        return 10; // Error return 10
    }
}
http.end();
}

```

Figure 5.12 Sending Camera data to API

In Figure 5.12, the function `sendCamData` is depicted. This function is responsible for sending data from a camera, identified by `camNumber`, to a specified API endpoint. It constructs a JSON payload containing the camera number and sends an HTTP POST request to the API endpoint. Upon receiving a response, it parses the JSON payload to extract the value associated with the "vehicle" key. If successful, it returns the vehicle count; otherwise, it returns an error code. This function plays a crucial role in integrating camera data with an external API for further processing or analysis.

```
void activateLight(int pin, int seconds,int pin2) {  
    Serial.println(seconds);  
    digitalWrite(pin2, HIGH); //opposite lane red light on  
    delay(1000);  
    digitalWrite(pin, HIGH); //current lane green light on  
    if(seconds>15)  
        seconds=10;  
    //wait for given time  
    for(int i=seconds;i>=0;i--){  
        Serial.println(i); //print in serial monitor  
        delay(1000);  
    }  
    digitalWrite(pin, LOW); // Turn off the current lane green light  
    delay(500);  
    digitalWrite(pin2,LOW); //Turn off the opposite lane red light  
}
```

Figure 5.13 LED light activation code

In Figure 5.13, the function `activateLight` controls traffic lights. It turns on the green light for a specified duration, then switches it off and turns on the opposing lane's red light.

5.4.3 Experimental Setup Prototype

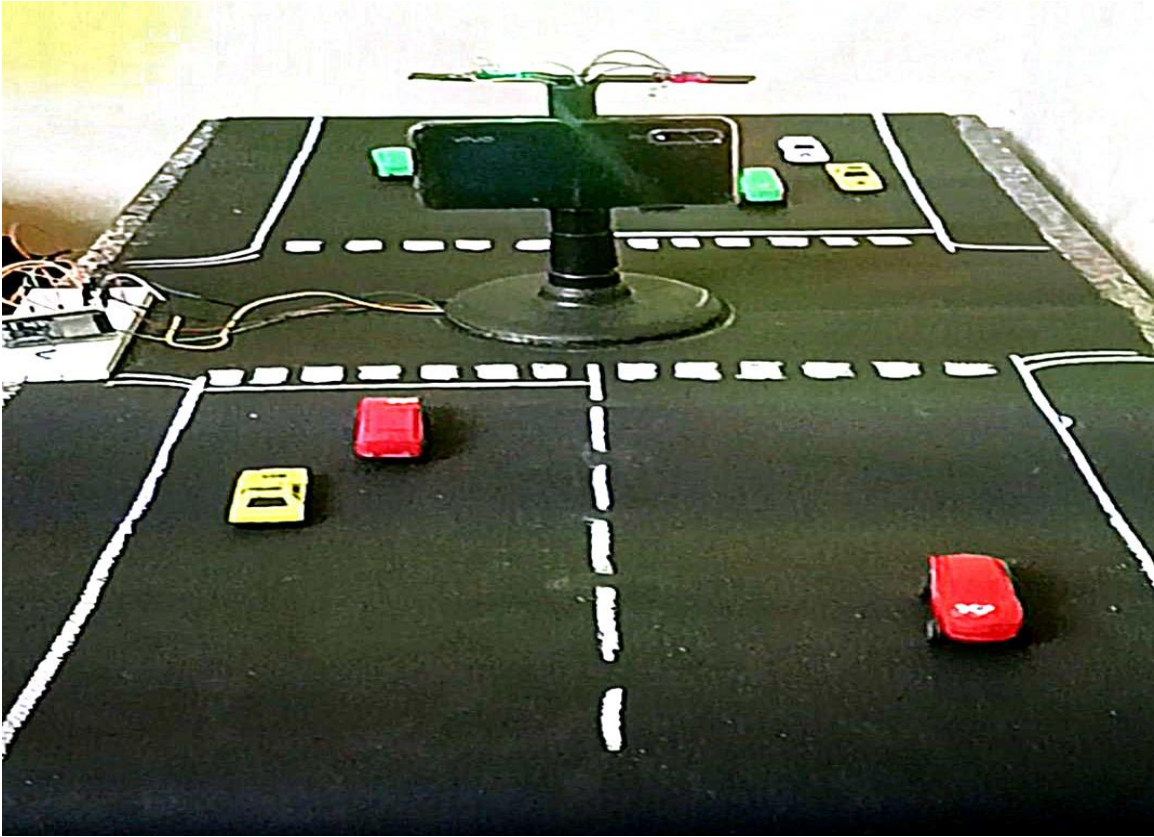


Figure 5.14 Prototype front view

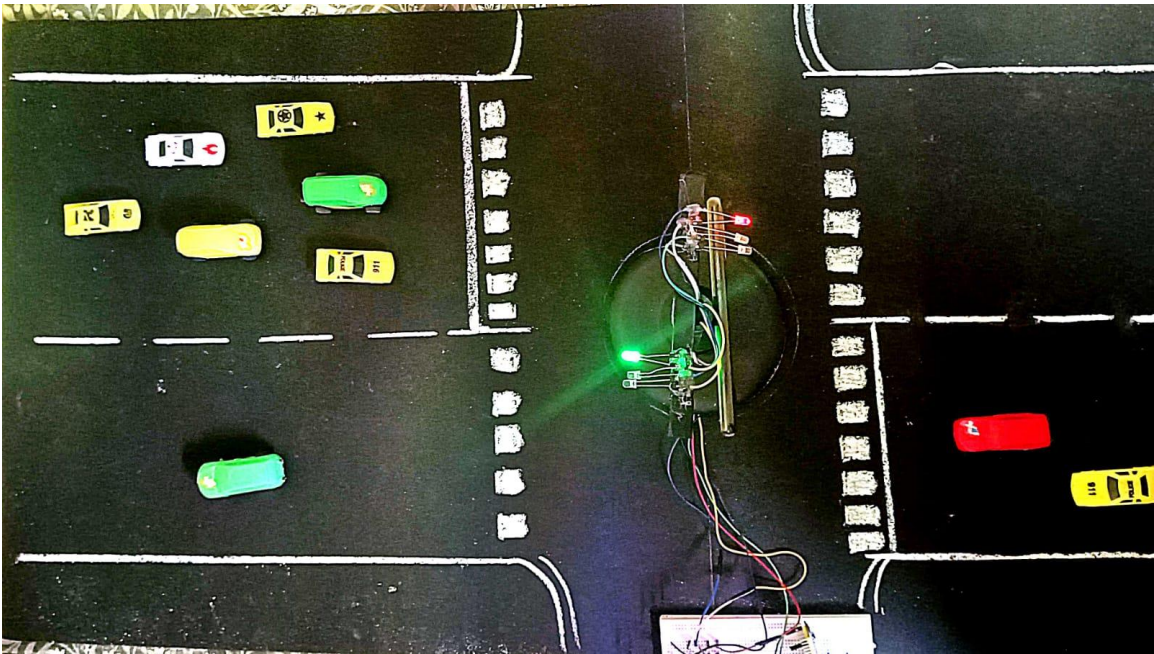


Figure 5.15 Prototype top view

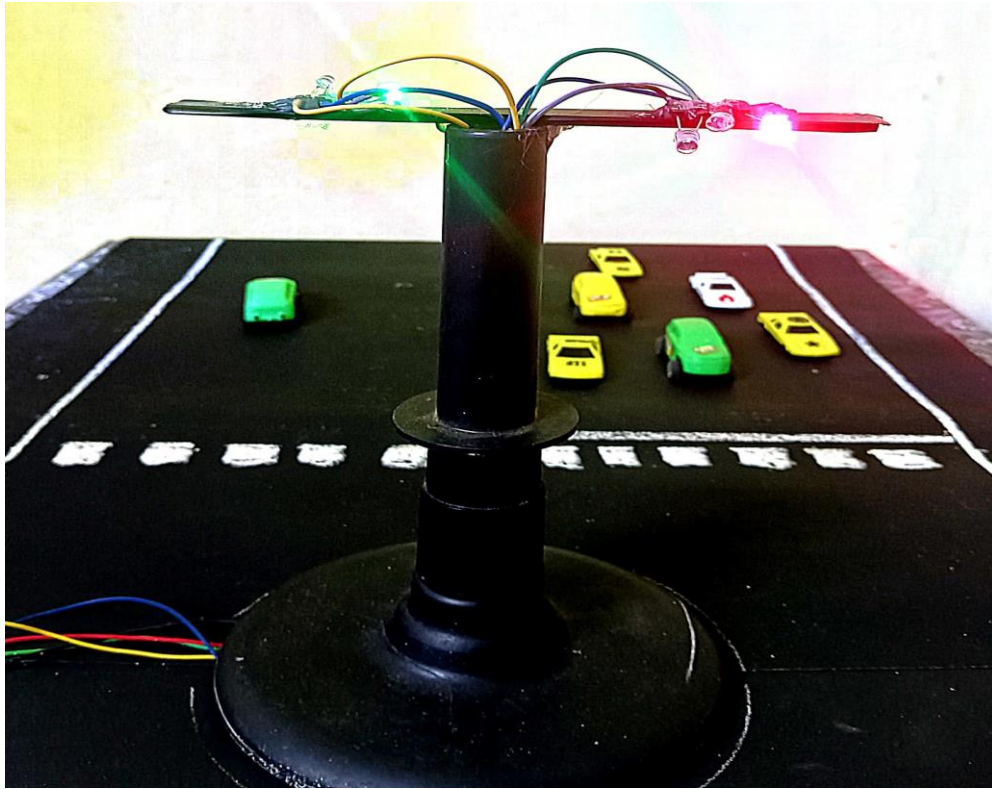


Figure 5.16 Prototype of traffic light

5.5 RESULTS AND GRAPHS

5.5.1 Arduino output

```

Connecting to WiFi...
Connecting to WiFi...
Connected to WiFi
cam Number,Vehicle Value:
1
10
Request for cam 1:activate green1 and off red2 :Timing:
10
10
9
8
7
6
5
4
3
2
1
0

```

Figure 5.17 Arduino serial monitor output

In Figure 5.17, the output shows WiFi connection status, vehicle count from camera 1, and traffic light activation countdown, illustrating data transmission and traffic control integration.

5.5.2 Functionalities output

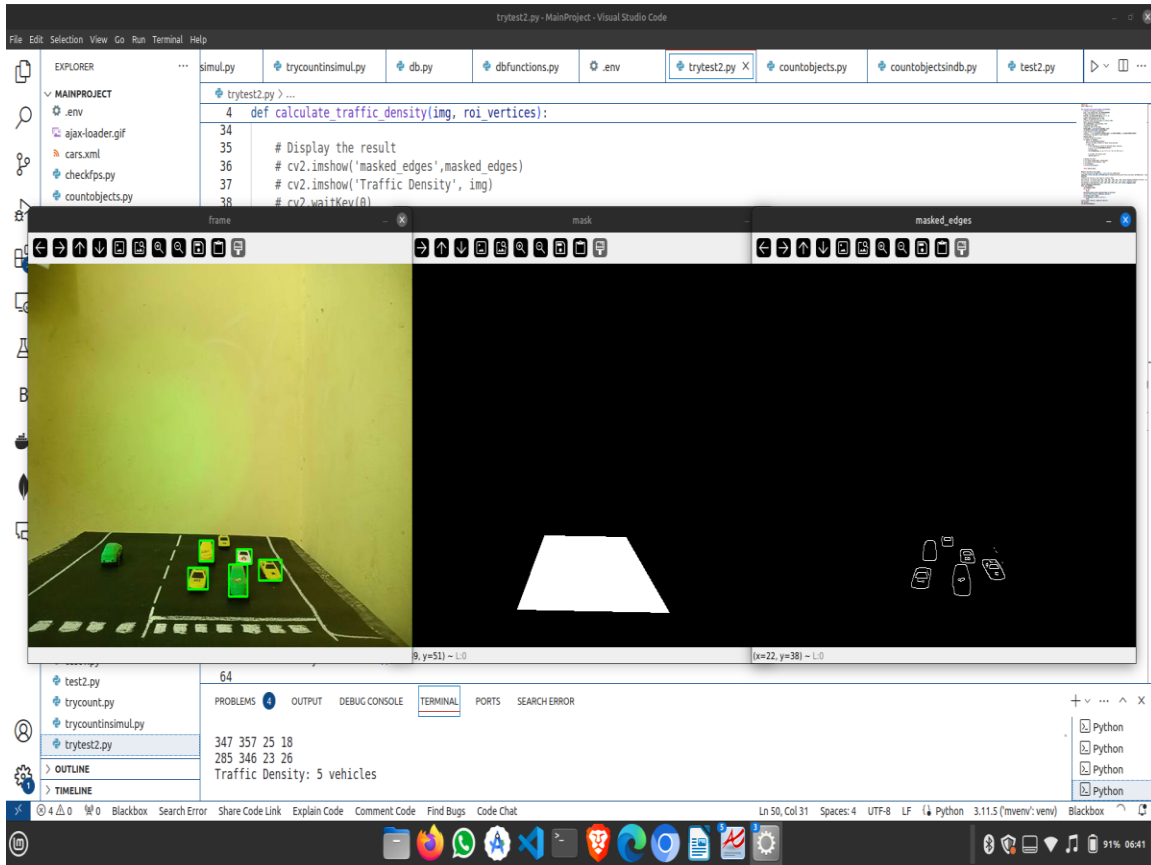


Figure 5.18 Counting number of vehicles

The Figure 5.18 shows the region of interest and canny images of object and if object area satisfy the constraints it will consider it as a object.



Figure 5.19 Helmet detected



Figure 5.20 Helmet not detected

Figure 5.19 and 5.20 shows the output of a frame that has helmet and without helmet.

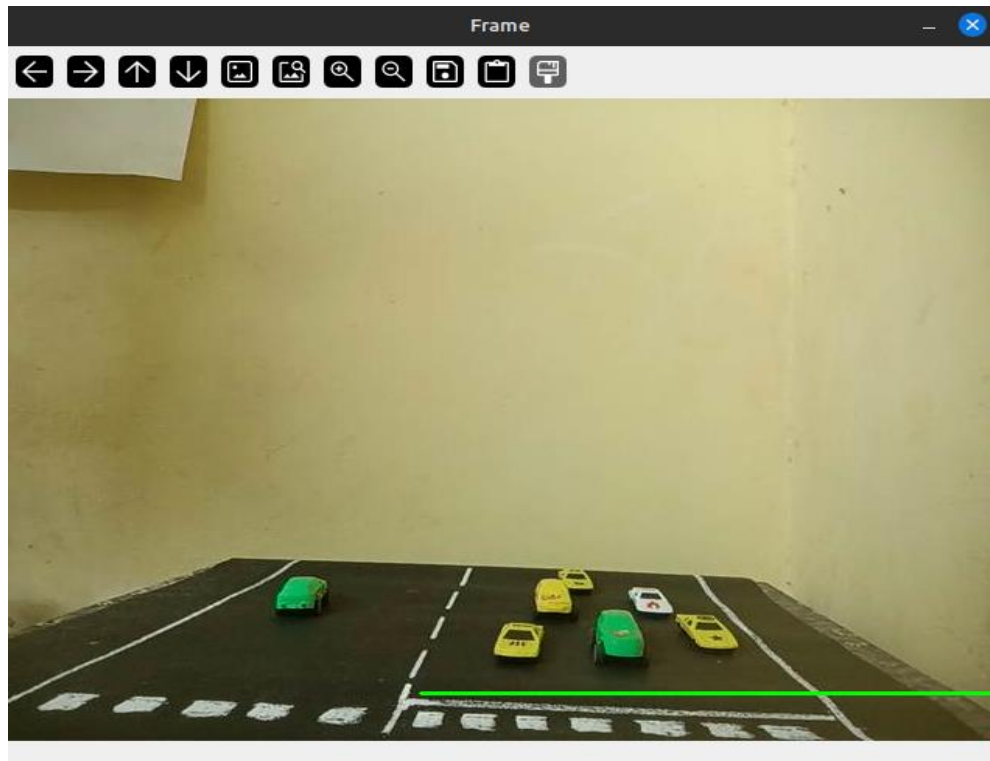


Figure 5.21 Vehicle counting by crossing the line on a lane

Figure 5.21 depicts the counting of vehicles passing through this lane it uses background subtraction using MOG2 model.



Figure 5.22 Emergency vehicle detection

The figure 5.22 shows the emergency vehicle detection of a ambulance and also its confidence level in blue frame.

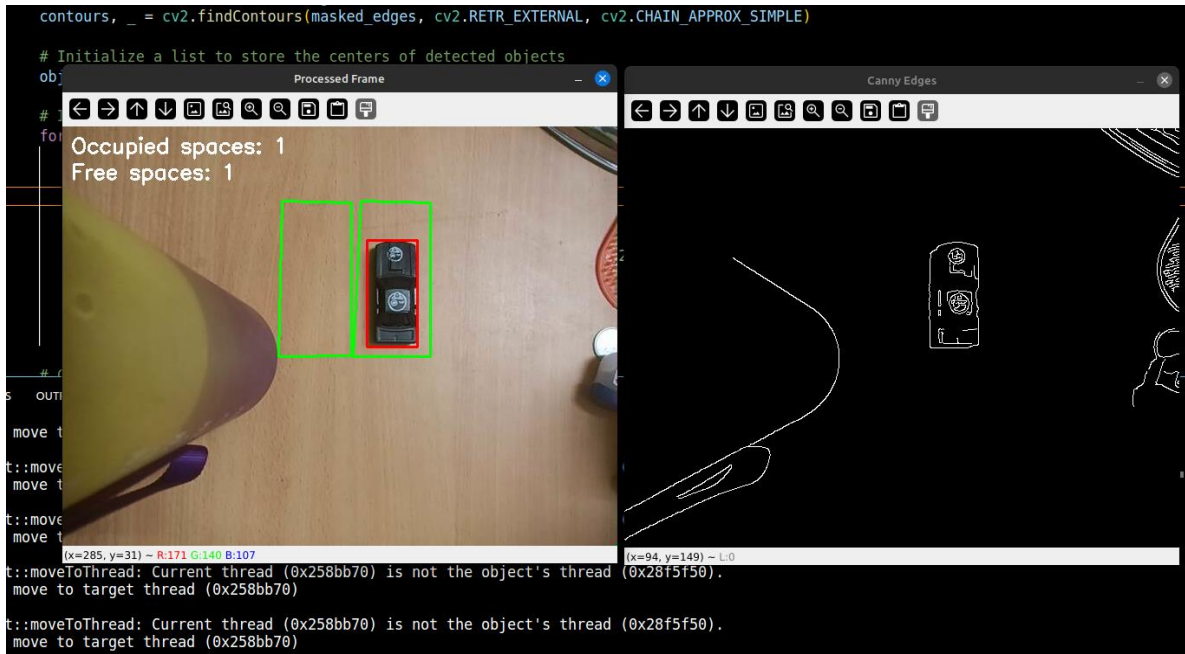


Figure 5.23 Parking slot availability output

The Figure 5. shows the canny frame to show object detection and processed frame shows that it contains two green rectangle which is for parking area and red rectangle for object detection. If the red rectangle is inside green rectangle it was considered as occupied space.

5.5.3 API latency

Table 5.1 Resolution and Total time taken for API response

Resolution	Total Time Taken (approx.)
640x480	12 ms
1280x720	36 ms
1920x1080	60 ms
2560x1440	108 ms
3840x2160	240 ms

Table 5.1 shows the latency change when project uses different types of resolution camera.

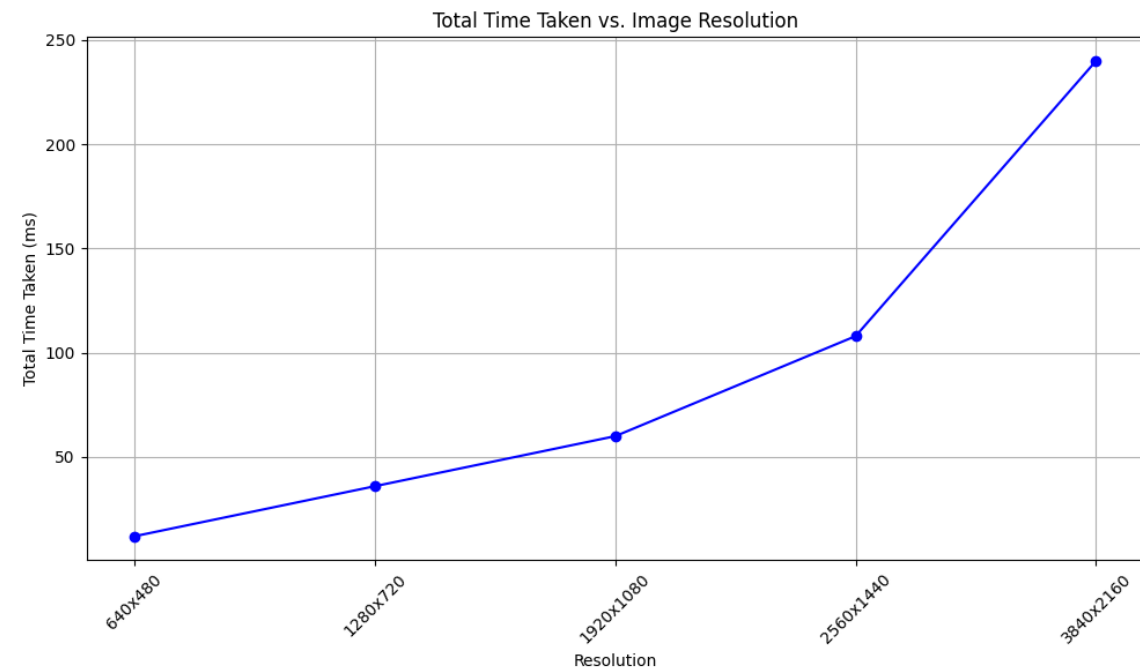


Figure 5.24 Latency vs Resolution

In Figure 5.24, the graph shows a linear relationship between image resolution and total time taken, where higher resolutions result in longer processing times. The data points form an ascending straight line on the plot.

5.5.4 MongoDB Atlas output

The screenshot shows the MongoDB Atlas interface for a project named 'project'. The 'penalties' collection is selected, and three sample documents are displayed:

```

{
  "_id": "661f82a093ead29ac1b57b91",
  "license": "LIC003",
  "timestamp": "2024-04-20T09:00:00",
  "reasonforpenalty": "Running red light",
  "amount": 75,
  "due date": "2024-04-30T09:00:00",
  "status": "unpaid"
}

{
  "_id": "661f82a093ead29ac1b57b92",
  "license": "LIC004",
  "timestamp": "2024-04-19T14:30:00",
  "reasonforpenalty": "Expired license",
  "amount": 150,
  "due date": "2024-04-29T14:30:00",
  "status": "unpaid"
}

{
  "_id": "661f82a093ead29ac1b57b93",
  "license": "LIC005",
  "timestamp": "2024-04-21T11:45:00",
  "reasonforpenalty": "Illegal U-turn",
  "amount": 90,
  "due date": "2024-05-01T11:45:00",
  "status": "unpaid"
}

```

Figure 5.25 MongoDB Atlas collections

Figure 5.25 shows the MongoDB cloud and its collections in traffic data database.



Figure 5.26 Heat map of peak hours

This Figure 5.26 shows a heat map visualization of traffic density data across different locations and time periods, with color intensity representing vehicle count aggregated by place and time.



Figure 5.27 Traffic density in different places

This Figure 5.27 shows heatmap visualization displays traffic density data across locations and time periods, with color intensity representing the aggregated vehicle count measure.

5.5.5 Time complexity of functions

1. ``calculate_traffic_density``: This function involves several image processing steps such as converting to grayscale, applying Gaussian blur, Canny edge detection, finding contours, and iterating through contours. The time complexity of these operations depends on factors like image size and contour count, but overall, it tends to be $(O(n))$ where (n) is the number of pixels in the image.
2. ``is_inside``: This function iterates through each corner of the inner rectangle and checks if it lies within the outer rectangle. Since it's a simple loop through corners, its time complexity is $(O(1))$ for each corner, making it $(O(4) = O(1))$ in total.
3. Object detection code: This involves background subtraction, thresholding, contour detection, and bounding box drawing. The time complexity can vary depending on factors like image size, number of contours, and processing techniques used. Generally, it tends to be $(O(n))$ where (n) is the number of pixels in the image or the number of detected objects.
4. ``count_spaces``: This function involves nested loops where each car center is checked against each parking area using the point-in-polygon test. If there are (m) cars and (k) parking areas, the time complexity is $(O(m * k))$.
5. ``sendCamData``: This function sends an HTTP request to an API endpoint and processes the response. The time complexity primarily depends on network latency and the processing time of the API server, making it challenging to determine accurately. However, in terms of local processing, it involves string manipulation, JSON parsing, and HTTP request handling, which are generally $(O(n))$ operations where (n) is the size of the data being processed.

6. ``activateLight``: This function involves a loop that waits for a specified duration, making it ($O(\text{seconds})$) where (seconds) is the duration. Other operations like turning on/off GPIO pins are typically ($O(1)$) operations.

5.6 SUMMARY

Chapter 5 offers a comprehensive overview of the hardware and software prerequisites essential for developing an intelligent traffic management system. It details hardware specifications, including requirements such as an Intel Core i5 processor, 8GB+ RAM, and ESP32 microcontroller. Additionally, software requirements, encompassing operating systems, coding languages, web frameworks, databases, and IDEs, are delineated. The significance of evaluation metrics, primarily latency and time complexity, is emphasized, along with a detailed explanation of latency calculation in the context of API response times. Dataset requirements, primarily real-time video feeds, and expected output formats, such as dynamic timing values, vehicle counts, and emergency vehicle presence data, are outlined. The chapter also delves into the experimental setup, including implementation details of the server and Arduino programs, alongside the utilization of MongoDB Atlas for storing traffic data collections. Insights and inferences drawn from experimental results and visualizations further enrich the discussion in this chapter.

CHAPTER VI

CONCLUSION AND FUTURE ENHANCEMENTS

6.1 CONCLUSION

The proposed smart traffic management system offers a comprehensive solution to address the challenges of urban traffic congestion. By integrating advanced technologies such as cloud computing and video analytics the system aims to optimize traffic flow, enhance safety, and improve overall efficiency in urban areas. The architecture of the system comprises various modules including Camera Feed Fetching, Signal Control, Vehicle Counting, Emergency Vehicle Detection, Helmet Violation Detection, Parking Availability Monitoring, and Data Storage (MongoDB Integration). These modules work together seamlessly to dynamically adjust traffic signal timings based on real-time traffic conditions, prioritize emergency vehicles for swift passage, detect traffic violations, and monitor parking availability. Through the utilization of threading, ESP32 Arduino integration, TensorFlow for object detection, and MongoDB for data storage, the system can efficiently handle traffic data collection, analysis, and management tasks.

6.2 FUTURE ENHANCEMENTS

While the proposed smart traffic management system represents a significant advancement in traffic management technology, there are several avenues for future enhancement and refinement:

1. **Integration of AI-based Predictive Analytics:** Incorporating predictive analytics algorithms can enable the system to anticipate traffic patterns and proactively adjust signal timings, further optimizing traffic flow and reducing congestion.
2. **Enhanced Emergency Vehicle Prioritization:** Future iterations of the system could incorporate more sophisticated algorithms for emergency vehicle detection and prioritization, ensuring even faster response times and smoother passage for emergency vehicles.

3. **Real-time Traffic Incident Management:** Implementing real-time incident detection and management capabilities can enable the system to respond quickly to accidents, road closures, and other traffic incidents, minimizing disruptions and improving overall traffic efficiency.
4. **User-Centric Mobile Applications:** Developing user-centric mobile applications can empower drivers and commuters with real-time traffic updates, alternative route suggestions, and parking availability information, enhancing overall travel experience and convenience.
5. **Integration with Smart City Infrastructure:** Integrating the smart traffic management system with other smart city infrastructure components such as public transportation systems, pedestrian walkways, and bike lanes can create a more holistic approach to urban mobility, fostering sustainable transportation practices and reducing reliance on private vehicles.

Incorporating these future enhancements will further elevate the effectiveness and efficiency of the smart traffic management system, making significant strides towards creating more livable, sustainable, and resilient urban environments.

REFERENCES

- [1] Mimansha Gupta, Harsha Miglani, Pradnyesh Deo and Alka Barhatte, “Real-time traffic control and monitoring” e-Prime Advances in Electrical Engineering, Electronics and Energy volume 5 (2023) .
- [2] Ida Bagus Kerthyayana Manuabaa, and Thomas Dwi Dinata, "A Comparative Analysis of Computer Vision Libraries in the Context of a Jakarta Traffic Simulator", Computer Science Department, 8th International Conference on Computer Science and Computational Intelligence(ICCSCI), (2023).
- [3] Antonio Pascale, Eloisa Macedo, Claudio Guarnaccia, and Margarida C. Coelho, "The Use of Cameras and Noise Estimation Models for Traffic Monitoring and Environmental Impact", Department of Mechanical Engineering/Centre for Mechanical Technology and Automation (TEMA), University of Aveiro, (2023).
- [4] Nandikolla VK, Ferman K, Barragan E, Melgar SF and Perez H. “Vision based obstacle detection and navigation of an autonomous vehicle”. ASME International Mechanical Engineering Congress and Exposition, Proceedings (IMECE), vol. 7B-2021, (2021).
- [5] V. Bhardwaj, Y. Rasamsetti and V. Valsan, “Image processing based smart traffic control system for smart city”, Proceedings of the 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), (2021), pp. 1–6.
- [6] Xiao Y, Tian Z, Yu J, Zhang Y, Liu S and Du S, “A review of object detection based on deep learning”. Multimedia Tools Application (2020).
- [7] Q. Meng, H. Song, Yu’ Zhang, X. Zhang, G. Li and Y. Yang, “Video-based vehicle counting for expressway: a novel approach based on vehicle detection

- and correlation-matched tracking using image data from PTZ cameras”, *Math. Probl. Eng.* (2020) 1–16.
- [8] Rydzewski A, Czarnul and P. “Recent advances in traffic optimisation: systematic literature review of modern models, methods and algorithms”. *IET Intel Transport System* (2020);14(13).
 - [9] Picaut J, Can A, Fortin N, Ardouin J and Lagrange M. “Low-cost sensors for urban noise monitoring networks”. *Sensors (Switzerland)* (2020);20:1–31.
 - [10] M.M. Gandhi, D.S. Solanki, R.S. Daptardar and N.S. Baloorkar, “Smart Control of Traffic Light Using Artificial Intelligence”, *Proceedings of the 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, (2020), pp. 1–6.
 - [11] T. Mingxing, P. Ruoming and V. Quoc, “EfficientDet: scalable and efficient object detection”, *Le Google Research, Brain Team. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (2020).
 - [12] Hamid Sarmadi, Rafael Munoz-Salinas, M.A.Berbis, and R.Medina-Carnicer, "Simultaneous Multi-View Camera Pose Estimation and Object Tracking With Squared Planar Markers", *Department of Information Technology, Spain, IEEE Access* 10.1109 (2019).
 - [13] P. Perez-Murueta, A. Gómez-Espinosa, C. Cardenas and M. Gonzalez-Mendoza, “Deep learning system for vehicular re-routing and congestion avoidance”, *Appl. Sci.* 9 (13) (2019).
 - [14] Huang T. “Traffic speed estimation from surveillance video data”, *Tingting Huang Institute for Transportation, Iowa State University. IEEE Conf Computer Vision Pattern Recognition* (2018).

- [15] K. Zaatouri and T.Ezzedine, “A self-adaptive traffic light control system based on yolo”, International Conference on Internet of Things, Embedded Systems and Communications (IINTEC), Hammamet, Tunisia, pp. 16–19, (2018).