UNIVERSITY INSTITUTE OF TECHNOLOGY

BARKATULLAH UNIVERSITY, BHOPAL (M.P)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MAJOR PROJECT

On

SMART TRAFFIC MANAGEMENT SYSETEM USING MACHINE LEARNING

Submitted for the partial fulfilment of the requirement for the

Award of Degree of

Bachelor of Technology (B.Tech.)

Year:2026

Barkatullah University, Bhopal

By:

AKRUTI PATIL ANJALI CHAURASIA

Under the Guidance

Of

Dr. Kavita Chourasia

Co- Guide

CSE, UIT-BU, Bhopal

Dr. Kamini Maheshwar

Co- Guide

CSE, UIT-BU, Bhopal

Dr. Divakar Singh

Guide & HOD

CSE, UIT-BU, Bhopal

Prof. N. K. Gaur

Director

CSE, UIT-BU, Bhopal

UNIVERSITY INSTITUTE OF TECHNOLOGY

BARKATULLAH UNIVERSITY, BHOPAL (M.P)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

YEAR-2025-26

This is to certify that Major Project entitled "SMART TRAFFIC MANAGEMENT SYSTEM USING MACHINE LEARNING" submitted to Barkatullah University Institute of Technology, Bhopal. Ms. AKRUTI PATIL, Ms. ANJALI CHAURASIA for the partial fulfilment for the award of the degree of Bachelor of Technology in Computer Science & Engineering in the year 2026.

Dr. Kavita Chourasia	Dr. Kamini Maheshwar
Co- Guide	Co- Guide
CSE, UIT-BU, Bhopal	CSE, UIT-BU, Bhopal
Dr. Divakar Singh	Prof. N. K. Gaur
Guide & HOD	Director
CSE, UIT-BU, Bhopal	CSE, UIT-BU, Bhopal

UNIVERSITY INSTITUTE OF TECHNOLOGY

BARKATULLAH UNIVERSITY, BHOPAL (M.P)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

YEAR-2025-26

Hereby declare that the Major Project work being presented in this report entitled "**SMART TRAFFIC MANAGEMENT SYSTEM USING MACHINE LEARNING**" submitted to the department of computer science. FACULATY OF TENCHNOLOGY, BARKATULLAH UNIVERSITY INSTITUTE OF TECHNOLOGY BHOPAL is the authentic work carried under the guidance of **Dr. Kavita Chourasia, Dr. Kamini Maheshwar** and **Dr. Divakar Singh** HOD Department of Computer Science Engineering, Barkatullah University Bhopal.

This is our original work and has not been submitted earlier in any other degree, diploma or any other certificate.

Date-_ \ _ \ _

AKRUTI PATIL

ANJALI CHAURASIA

ACKNOWLEDGEMENT

We would like to express special thanks and gratitude to Dr. Kavita Chourasia and Dr. Kamini Maheshwar for help, guidance and throughout the work for our project without which we would not have been able to complete this project to such a success. We would also like to extend our special thanks and gratitude to HOD Dr. Divarkar Singh for providing effective platform and support in the development of this project. And finally, we would like to render our thanks to Director Dr. N. K. Gaur for his guidance in this major titled "SMART TRAFFIC MANAGEMENT SYSTEM USING MACHINE LEARNING".

Last but the least we would like to thanks our parents and friends for their support and cooperation. Regardless of the source, we wish to express our gratitude to those who may have contributed to this work, even through anonymously.

Submitted by:

AKRUTI PATIL(R238237200028)

ANJALI CHAURASIA(R238237200075)

TABLE OF CONTENTS

ABSTRACT

The increasing urbanisation and growing number of vehicles have led to significant challenges in traffic management, resulting in congestion, delays, and environmental pollution. A Smart Traffic Management System leverages modern technologies such as the Internet of Things (IoT), artificial intelligence (AI), and data analytics to address these issues efficiently. The system uses real-time data from sensors, cameras, and GPS devices to monitor traffic flow and detect patterns. AI-based algorithms analyse this data to optimise traffic light timings, predict congestion, and manage vehicle routing. Additionally, the system can adapt to dynamic conditions, such as accidents or road closures, by providing real-time updates to drivers and diverting traffic accordingly. The implementation of such a system can reduce traffic congestion, improve safety, decrease fuel consumption, and contribute to sustainable urban development. This paper discusses the design, architecture, and implementation of an intelligent traffic management system and its potential benefits in modern cities.

The rapid increase in urban population has led to growing challenges in road traffic congestion, adversely impacting both economic efficiency and quality of life. This project presents a Smart Traffic Management System that utilises a hybrid approach, combining both centralised and decentralised models, to optimise traffic flow by leveraging Internet of Things (IoT) devices, artificial intelligence, and real-time data from cameras and sensors. The system dynamically adjusts traffic signal timings based on real-time density data, prioritises emergency vehicles using RFID technology, and detects incidents like fires through integrated sensors. A prototype demonstrates the system's ability to not only reduce congestion and enhance public safety but also to provide graphical analysis to authorities for improved urban planning and resource allocation. The proposed solution ensures that traffic is efficiently managed, even during network failures, and integrates seamlessly with city rescue departments, marking a significant advancement towards smarter city infrastructure

INTRODUCTION

Rapid urbanization and population growth have led to a significant increase in the number of vehicles on roads, resulting in severe traffic congestion, longer travel times, and increased environmental pollution. Traditional traffic management systems are often inadequate in handling these dynamic and complex traffic conditions. Therefore, there is a growing need for intelligent and adaptive solutions that can efficiently manage urban traffic and enhance road safety.

A **Smart Traffic Management System using Machine Learning (ML)** aims to address these challenges by leveraging advanced data analysis and automation techniques. Machine learning enables the system to analyse large volumes of traffic data, identify patterns, and make real-time decisions to optimize traffic flow. These systems can be integrated with video surveillance, sensors, and IoT devices to monitor road conditions, detect anomalies such as accidents, and respond promptly.

One important component of such intelligent systems is **collision detection**, which plays a crucial role in improving emergency response and minimizing the impact of road accidents. In this project, a video-based collision detection approach is implemented using computer vision techniques. The system processes video frames, detects sudden motion changes that may indicate a collision, and triggers an alert mechanism to notify users immediately.

In addition to accident detection, smart traffic management also emphasizes **sustainable urban mobility solutions**. Encouraging public transportation, promoting bike-sharing and carpooling, and involving community participation can significantly reduce traffic congestion and emissions. The use of modern technologies and mobile applications further enhances accessibility and efficiency in transportation systems.

Overall, the proposed system demonstrates how machine learning and computer vision can be utilized to develop an intelligent, efficient, and scalable traffic management solution. With further advancements, such systems have the potential to contribute significantly to the development of smart cities, improved urban mobility, and safer road environments.

LITERATURE REVIEW

2.1 Traditional Approaches to Traffic Management

Traditional traffic management systems relied on fixed-time traffic signals, manual monitoring, and sensor-based methods such as loop detectors and CCTV surveillance. These approaches lacked adaptability and were inefficient in handling dynamic traffic conditions, often leading to congestion and delayed response to accidents.

2.2 Machine Learning and Deep Learning in Traffic Analysis

Machine learning techniques have been widely used to analyse traffic patterns, predict congestion, and optimize traffic flow. Algorithms such as regression models, decision trees, and neural networks enable systems to learn from historical data. Deep learning methods further enhance accuracy by processing large datasets and identifying complex patterns in traffic behavior.

2.3 Object Detection Models (YOLO, SSD, R-CNN, etc.)

Modern traffic systems use object detection models to identify vehicles in video streams. Models like YOLO (You Only Look Once), SSD (Single Shot Detector), and R-CNN provide real-time detection and classification of vehicles. These models are highly effective in detecting multiple objects simultaneously with high accuracy.

2.4 Multi-Object Tracking Techniques (SORT, DeepSORT)

Tracking techniques such as SORT (Simple Online and Realtime Tracking) and DeepSORT are used to monitor the movement of multiple vehicles across frames. These methods help in analysing vehicle trajectories, speed estimation, and detecting unusual behavior such as sudden stops or collisions.

2.5 Temporal Recognition

Temporal analysis involves studying changes over time in video sequences. Techniques such as frame differencing, optical flow, and sequence modeling help detect abnormal events like accidents. Motion-based detection, as used in this project, is a simple yet effective method for identifying sudden impacts.

2.6 Ethical, Privacy, and Operational Challenges

The use of surveillance systems raises concerns regarding data privacy and security. Continuous video monitoring may lead to misuse of personal data. Additionally, implementing advanced AI-based systems requires high computational resources and infrastructure, which can be costly and complex.

2.7 Research Gaps

Despite advancements in intelligent traffic systems, several gaps remain:

- High dependency on complex and expensive AI models
- Limited availability of real-time, high-quality datasets
- Difficulty in detecting collisions accurately in all scenarios
- Need for simple, cost-effective solutions for practical implementation

This project addresses these gaps by proposing a **basic motion-based collision detection system**, which is efficient, easy to implement, and can be further enhanced using advanced AI techniques.

PROBLEM DEFINITION

3.1 Background and Context

Rapid urbanization and the increasing number of vehicles have led to severe traffic congestion and a rise in road accidents. Traditional traffic management systems are mostly static and lack the ability to respond dynamically to real-time traffic conditions. Monitoring traffic manually is inefficient and often results in delayed detection of accidents. With the advancement of technology, there is a growing need for intelligent systems that can automatically monitor traffic and respond promptly to critical situations.

3.2 Statement of the Problem

One of the major issues in current traffic systems is the **delay in detecting vehicle collisions**, which can lead to increased damage, delayed emergency response, and loss of life. Existing solutions either rely on manual monitoring or require complex and expensive infrastructure, making them less accessible for widespread use. Additionally, many advanced AI-based systems demand high computational resources and large datasets. Therefore, there is a need for a **simple, cost-effective, and efficient system** that can detect possible collisions in real time and provide immediate alerts.

3.3 Research Objectives

The main objectives of this study are:

- To develop a system for monitoring traffic using video input
- To detect possible vehicle collisions using motion analysis
- To provide immediate alert notifications upon detection
- To ensure smooth and real-time processing of video data
- To create a foundation for future AI-based traffic management systems

3.4 Scope and Delimitations

This project focuses on **video-based collision detection** using basic computer vision techniques. The system processes pre-recorded video input and identifies sudden motion changes as potential collisions.

Scope:

- Detection of collisions based on motion analysis
- Alert generation using sound notification

- Implementation using Python and OpenCV

Delimitations:

- Does not use advanced deep learning models
- Limited to motion-based detection (not object-level accuracy)
- May not perform well in highly complex or real-time live environments

3.5 Challenges and Constraints

- Difficulty in accurately distinguishing collisions from normal motion
- Sensitivity to camera movement and lighting variations
- Selection of appropriate motion threshold for detection
- Limited computational resources for advanced AI implementation
- Lack of large annotated datasets for training

3.6 Significance of the Study

This study contributes to the development of **intelligent traffic monitoring systems** by providing a simple and efficient approach to collision detection. It demonstrates how computer vision techniques can be used to improve road safety and reduce response time in accident scenarios. The proposed system serves as a **cost-effective prototype** that can be further enhanced using machine learning and integrated into smart city infrastructure for better traffic management and urban mobility.

4. METHODOLOGY

4.1 System Overview

The proposed system is a **video-based traffic monitoring and collision detection system** that uses computer vision techniques to analyse vehicle movement. It processes video input frame-by-frame, detects sudden motion changes indicating possible collisions, and triggers an alert system. The system is designed to be simple, efficient, and capable of real-time performance.

4.2 Dataset Preparation

The dataset consists of **traffic video clips** containing normal driving scenarios as well as accident situations. These videos are either collected from online sources or recorded manually. The dataset is used to test and validate the system's ability to detect motion changes and identify potential collisions.

4.3 Preprocessing and Annotation

- Video frames are extracted from input clips
- Frames are converted to **grayscale** to reduce computational complexity
- Noise reduction techniques may be applied for better accuracy
- In this system, **explicit annotation is not required** since detection is based on motion differences rather than object labeling

4.4 YOLOv8 Model Development

In advanced implementations, **YOLOv8 (You Only Look Once)** can be used for real-time vehicle detection and classification. This model detects vehicles within each frame and provides bounding boxes, which can be further used for accurate collision detection.

Note: The current system uses motion-based detection; however, YOLOv8 integration is proposed for future enhancement.

4.5 Training Configuration and Hyperparameters

For AI-based extensions (e.g., YOLOv8), the following parameters are considered:

- Learning rate

- Number of epochs
- Batch size
- Image resolution
- Confidence threshold

These parameters help optimize model performance and detection accuracy.

4.6 Tracking Module (DeepSORT Integration)

DeepSORT (Deep Simple Online Realtime Tracking) can be integrated to track multiple vehicles across frames. It assigns unique IDs to each vehicle and monitors their movement, enabling better analysis of vehicle trajectories and interactions.

4.7 Temporal Behavior Detection Logic

The system uses **frame differencing techniques** to analyse temporal changes:

- Consecutive frames are compared
- Pixel differences are calculated
- A motion score is generated
- If the motion exceeds a predefined threshold, it is identified as a potential collision

This approach helps in detecting sudden events occurring over time in video sequences.

4.8 System Implementation (User Interface)

The system is implemented using Python with OpenCV and Pygame. A simple interface displays:

- Real-time video playback
- Collision detection alerts
- Sound notification system

For advanced deployment, a **Flask-based web interface** can be developed to allow users to upload videos and view results online.

CODE

Show the Path with respect to Timing:

```
import webbrowser
import urllib.parse

def open_google_maps_directions(pickup, destination, travel_mode="driving"):
 """
 pickup, destination: strings like 'Mumbai, India' or '19.0760,72.8777'  travel_mode:
'driving', 'walking', 'bicycling', 'transit'
 """
 base_url = "https://www.google.com/maps/dir/?"  params
= {
 "api": "1",
 "origin": pickup,
 "destination": destination,
 "travelmode": travel_mode
 }
 url = base_url + urllib.parse.urlencode(params)
 webbrowser.open(url)   # opens default browser [web:13][web:16][web:19]

if __name__ == "__main__":  pickup = "Chhatrapati Shivaji Maharaj Terminus,
Mumbai"  destination = "Gateway of India, Mumbai"
open_google_maps_directions(pickup, destination, travel_mode="driving") if
__name__ == "__main__":
 # latitude,longitude format is supported in origin/destination. [web:14][web:17]
pickup = "19.0760,72.8777"       # Mumbai  destination = "18.9218,72.8347"  # Near
Gateway of India  open_google_maps_directions(pickup, destination, "driving"
```

Vehicle Counter:

```
import cv2

# Video file path
video_path = "video43.mp4"  # Replace with your video path cap =
cv2.VideoCapture(video_path)

# Subtractor for motion detection
fgbg=cv2.createBackgroundSubtractorMOG2(history=500,varThreshold=100)

count_line_position = 300 min_contour_width
= 40 min_contour_height = 40

car_count = 0

def center_handle(x, y, w, h):
 x1 = int(w / 2)
y1 = int(h / 2)
cx = x + x1  cy
= y + y1
 return cx, cy

detect = []

while True:
 ret, frame = cap.read()
if not ret:  break
 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
 blur = cv2.GaussianBlur(gray, (5, 5), 5)  fgmask =
fgbg.apply(blur)
 dilated = cv2.dilate(fgmask, None, iterations=3)  contours, _ = cv2.findContours(dilated,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  cv2.line(frame, (25, count_line_position),
(1200, count_line_position), (255, 127, 0), 3)

 for contour in contours:
```

```
 (x, y, w, h) = cv2.boundingRect(contour)
      if w >= min_contour_width and h >= min_contour_height:
         center = center_handle(x, y, w, h)
detect.append(center)
         cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

   for (x, y) in detect:        if count_line_position - 6 < y <
count_line_position + 6:
         car_count += 1
detect.remove((x, y))

   cv2.putText(frame, f'VEHICLE COUNT: {car_count}', (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
2, (0, 0, 255), 2)

   cv2.imshow('Vehicle Counter', frame)
if cv2.waitKey(30) & 0xFF == 27:
      break

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

Traffic Management:

```
import random

import time import

threading import

pygame import sys


# Default values of signal timers

defaultGreen = {0: 10, 1: 10, 2: 10, 3: 10}

defaultRed = 150 defaultYellow = 5


signals = [] noOfSignals = 4 currentGreen = 0  # Indicates which signal is green currently

nextGreen = (currentGreen + 1) % noOfSignals  # Indicates which signal will turn green

next currentYellow = 0  # Indicates whether yellow signal is on or off


speeds = {'car': 2.25, 'bus': 1.8, 'truck': 1.8, 'bike': 2.5}  # average speeds of vehicles


# Coordinates of vehicles' start x = {'right': [0, 0, 0], 'down': [755, 727, 697], 'left': [1400,

1400, 1400], 'up': [602, 627,

657]} y = {'right': [348, 370, 398], 'down': [0, 0, 0], 'left': [498, 466, 436], 'up': [800, 800,

800]}


vehicles = {'right': {0: [], 1: [], 2: [], 'crossed': 0}, 'down': {0: [], 1: [], 2: [], 'crossed': 0},

        'left': {0: [], 1: [], 2: [], 'crossed': 0}, 'up': {0: [], 1: [], 2: [], 'crossed': 0}}
```

```
vehicleTypes = {0: 'car', 1: 'bus', 2: 'truck', 3: 'bike'} directionNumbers =
{0: 'right', 1: 'down', 2: 'left', 3: 'up'}

# Coordinates of signal image, timer, and vehicle count signalCoods =
[(530, 230), (810, 230), (810, 570), (530, 570)] signalTimerCoods = [(530,
210), (810, 210), (810, 550), (530, 550)]

# Coordinates of stop lines stopLines = {'right': 590, 'down':
330, 'left': 800, 'up': 535} defaultStop = {'right': 580, 'down':
320, 'left': 810, 'up': 545}

# Gap between vehicles stoppingGap =
25  # stopping gap movingGap = 25  #
moving gap

# set allowed vehicle types here allowedVehicleTypes = {'car': True, 'bus':
True, 'truck': True, 'bike': True} allowedVehicleTypesList = []
vehiclesTurned = {'right': {1: [], 2: []}, 'down': {1: [], 2: []}, 'left': {1: [], 2: []}, 'up': {1: [], 2: []}}
vehiclesNotTurned = {'right': {1: [], 2: []}, 'down': {1: [], 2: []}, 'left': {1: [], 2: []}, 'up': {1: [], 2:
[]}}
rotationAngle = 3 mid = {'right': {'x': 705, 'y': 445}, 'down': {'x': 695, 'y': 450}, 'left':
{'x': 695, 'y': 425},       'up': {'x': 695, 'y': 400}}
# set random or default green signal time here
randomGreenSignalTimer = True # set random
green signal time range here
randomGreenSignalTimerRange = [10, 20]
pygame.init() simulation =
pygame.sprite.Group()
```

```
class TrafficSignal:     def _init_(self,
red, yellow, green):
       self.red = red
self.yellow = yellow
self.green = green
self.signalText = ""



class Vehicle(pygame.sprite.Sprite):     def _init_(self, lane, vehicleClass,
direction_number, direction, will_turn):
       pygame.sprite.Sprite._init_(self)
self.lane = lane        self.vehicleClass =
vehicleClass        self.speed =
speeds[vehicleClass]        self.direction_number
= direction_number        self.direction =
direction        self.x = x[direction][lane]
self.y = y[direction][lane]        self.crossed = 0
self.willTurn = will_turn        self.turned = 0
self.rotateAngle = 0
vehicles[direction][lane].append(self)
self.index = len(vehicles[direction][lane]) - 1
self.crossedIndex = 0        path = "images/" +
direction + "/" + vehicleClass + ".png"
self.originalImage = pygame.image.load(path)
self.image = pygame.image.load(path)
```

```
    if (len(vehicles[direction][lane]) > 1 and vehicles[direction][lane][self.index -
1].crossed == 0):           if (direction == 'right'):
            self.stop = vehicles[direction][lane][self.index - 1].stop
- vehicles[direction][lane][self.index -
  1].image.get_rect().width
- stoppingGap           elif (direction == 'left'):
            self.stop = vehicles[direction][lane][self.index - 1].stop
            + vehicles[direction][lane][self.index - 1].image.get_rect().width
            + stoppingGap
elif (direction == 'down'):
            self.stop = vehicles[direction][lane][self.index - 1].stop
- vehicles[direction][lane][self.index -
  1].image.get_rect().height
- stoppingGap           elif (direction == 'up'):
            self.stop = vehicles[direction][lane][self.index - 1].stop
            + vehicles[direction][lane][self.index - 1].image.get_rect().height
            + stoppingGap
else:
        self.stop = defaultStop[direction]


    # Set new starting and stopping coordinate
    if (direction == 'right'):
        temp = self.image.get_rect().width + stoppingGap
x[direction][lane] -= temp        elif (direction == 'left'):
        temp = self.image.get_rect().width + stoppingGap
x[direction][lane] += temp          elif (direction == 'down'):
temp    =    self.image.get_rect().height    +    stoppingGap
y[direction][lane] -= temp        elif (direction == 'up'):
```

```
        temp = self.image.get_rect().height + stoppingGap
y[direction][lane] += temp        simulation.add(self)


   def render(self, screen):
      screen.blit(self.image, (self.x, self.y))


   def move(self):        if
(self.direction == 'right'):
        if (self.crossed == 0 and self.x + self.image.get_rect().width > stopLines[self.direction]):
            self.crossed = 1
vehicles[self.direction]['crossed'] += 1              if
(self.willTurn == 0):
              vehiclesNotTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesNotTurned[self.direction][self.lane]) - 1          if
(self.willTurn == 1):              if (self.lane == 1):
              if (self.crossed == 0 or self.x + self.image.get_rect().width <
stopLines[self.direction] + 40):
                if ((self.x + self.image.get_rect().width <= self.stop or
(                    currentGreen == 0 and currentYellow == 0) or self.crossed == 1) and
(                    self.index == 0 or self.x + self.image.get_rect().width <
(                    vehicles[self.direction][self.lane][self.index - 1].x - movingGap) or
vehicles[self.direction][self.lane][self.index - 1].turned == 1)):
                  self.x +=
self.speed                else:
if (self.turned == 0):
                  self.rotateAngle += rotationAngle                    self.image =
pygame.transform.rotate(self.originalImage, self.rotateAngle)                    self.x += 2.4
self.y -= 2.8                    if (self.rotateAngle == 90):
```

```
                    self.turned = 1
                    vehiclesTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1                    else:
                if (self.crossedIndex == 0 or (self.y >
(                        vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].y +
vehiclesTurned[self.direction][self.lane][                        self.crossedIndex -
1].image.get_rect().height + movingGap))):                        self.y -= self.speed
elif (self.lane == 2):
            if (self.crossed == 0 or self.x + self.image.get_rect().width <
mid[self.direction]['x']):
                if ((self.x + self.image.get_rect().width <= self.stop or
(                    currentGreen == 0 and currentYellow == 0) or self.crossed == 1) and
(                    self.index == 0 or self.x + self.image.get_rect().width <
(                    vehicles[self.direction][self.lane][self.index - 1].x - movingGap) or
vehicles[self.direction][self.lane][self.index - 1].turned == 1)):
                self.x +=
self.speed                else:
if (self.turned == 0):
                self.rotateAngle += rotationAngle                    self.image =
pygame.transform.rotate(self.originalImage, -self.rotateAngle)                    self.x += 2
self.y += 1.8                    if (self.rotateAngle == 90):
                    self.turned = 1
                    vehiclesTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1                    else:
                if (self.crossedIndex == 0 or ((self.y + self.image.get_rect().height) <
(                    vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].y -
movingGap))):
```

```
                    self.y +=
self.speed          else:           if
(self.crossed == 0):
            if ((self.x + self.image.get_rect().width <= self.stop or
(                   currentGreen == 0 and currentYellow == 0)) and
(                   self.index == 0 or self.x + self.image.get_rect().width <
(                   vehicles[self.direction][self.lane][self.index - 1].x - movingGap))):
                self.x += self.speed
else:
            if ((self.crossedIndex == 0) or (self.x + self.image.get_rect().width <
(                   vehiclesNotTurned[self.direction][self.lane][self.crossedIndex - 1].x -
movingGap))):
                self.x += self.speed
elif (self.direction == 'down'):
        if (self.crossed == 0 and self.y + self.image.get_rect().height >
stopLines[self.direction]):
            self.crossed = 1
vehicles[self.direction]['crossed'] += 1            if
(self.willTurn == 0):
            vehiclesNotTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesNotTurned[self.direction][self.lane]) - 1          if
(self.willTurn == 1):           if (self.lane == 1):
            if (self.crossed == 0 or self.y + self.image.get_rect().height <
stopLines[self.direction] + 50):
                if ((self.y + self.image.get_rect().height <= self.stop or
(                   currentGreen == 1 and currentYellow == 0) or self.crossed == 1) and
(                   self.index == 0 or self.y + self.image.get_rect().height <
(                   vehicles[self.direction][self.lane][self.index - 1].y - movingGap) or
vehicles[self.direction][self.lane][self.index - 1].turned == 1)):
```

```
                self.y +=
self.speed                else:
if (self.turned == 0):
                self.rotateAngle += rotationAngle                    self.image =
pygame.transform.rotate(self.originalImage, self.rotateAngle)                    self.x += 1.2
self.y += 1.8                    if (self.rotateAngle == 90):
                    self.turned = 1
                    vehiclesTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1                    else:
                if (self.crossedIndex == 0 or ((self.x + self.image.get_rect().width) <
(                    vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].x -
movingGap))):
                    self.x += self.speed
elif (self.lane == 2):
            if (self.crossed == 0 or self.y + self.image.get_rect().height <
mid[self.direction]['y']):
                if ((self.y + self.image.get_rect().height <= self.stop or
(                    currentGreen == 1 and currentYellow == 0) or self.crossed == 1) and
(                    self.index == 0 or self.y + self.image.get_rect().height <
(                    vehicles[self.direction][self.lane][self.index - 1].y - movingGap) or
vehicles[self.direction][self.lane][self.index - 1].turned == 1)):
                self.y +=
self.speed                else:
if (self.turned == 0):
                self.rotateAngle += rotationAngle                    self.image =
pygame.transform.rotate(self.originalImage, -self.rotateAngle)                    self.x -= 2.5
self.y += 2                    if (self.rotateAngle == 90):
                    self.turned = 1
```

```
                    vehiclesTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1                    else:
                if (self.crossedIndex == 0 or (self.x >
(                        vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].x +
vehiclesTurned[self.direction][self.lane][                        self.crossedIndex -
1].image.get_rect().width + movingGap))):                        self.x -= self.speed          else:
if (self.crossed == 0):
            if ((self.y + self.image.get_rect().height <= self.stop or
(                    currentGreen == 1 and currentYellow == 0)) and
(                    self.index == 0 or self.y + self.image.get_rect().height <
(                    vehicles[self.direction][self.lane][self.index - 1].y - movingGap))):
                self.y += self.speed
else:
            if ((self.crossedIndex == 0) or (self.y + self.image.get_rect().height <
(                    vehiclesNotTurned[self.direction][self.lane][self.crossedIndex - 1].y -
movingGap))):
                self.y += self.speed        elif (self.direction == 'left'):
if (self.crossed == 0 and self.x < stopLines[self.direction]):
            self.crossed = 1
vehicles[self.direction]['crossed'] += 1              if
(self.willTurn == 0):
            vehiclesNotTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesNotTurned[self.direction][self.lane]) - 1          if
(self.willTurn == 1):              if (self.lane == 1):                if (self.crossed == 0 or
self.x > stopLines[self.direction] - 70):
                if ((self.x >= self.stop or (                        currentGreen == 2 and
currentYellow == 0) or self.crossed == 1) and (                        self.index == 0 or self.x >
(vehicles[self.direction][self.lane][self.index - 1].x +
```

```
vehicles[self.direction][self.lane][ self.index -
1].image.get_rect().width + movingGap) or
vehicles[self.direction][self.lane][self.index - 1].turned == 1)):
                self.x -=
self.speed              else:
if (self.turned == 0):
                self.rotateAngle += rotationAngle              self.image =
pygame.transform.rotate(self.originalImage, self.rotateAngle)              self.x -= 1
self.y += 1.2              if (self.rotateAngle == 90):
                    self.turned = 1
                    vehiclesTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1              else:
                if (self.crossedIndex == 0 or ((self.y + self.image.get_rect().height) <
(                    vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].y -
movingGap))):
                    self.y += self.speed          elif (self.lane == 2):
if (self.crossed == 0 or self.x > mid[self.direction]['x']):
                if ((self.x >= self.stop or (              currentGreen == 2 and
currentYellow == 0) or self.crossed == 1) and (              self.index == 0 or self.x >
(vehicles[self.direction][self.lane][self.index -
1].x +


vehicles[self.direction][self.lane][ self.index -
1].image.get_rect().width + movingGap) or
vehicles[self.direction][self.lane][self.index - 1].turned == 1)):
                self.x -= self.speed
```

```
                else:                    if
(self.turned == 0):
                        self.rotateAngle += rotationAngle                        self.image =
pygame.transform.rotate(self.originalImage, -self.rotateAngle)                        self.x -= 1.8
self.y -= 2.5                        if (self.rotateAngle == 90):
                            self.turned = 1
                            vehiclesTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1                    else:
                        if (self.crossedIndex == 0 or (self.y >
(                            vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].y +
vehiclesTurned[self.direction][self.lane][                            self.crossedIndex -
1].image.get_rect().height + movingGap))):                            self.y -= self.speed
else:               if (self.crossed == 0):
                if ((self.x >= self.stop or (currentGreen == 2 and currentYellow == 0)) and
(                        self.index == 0 or self.x >
(                        vehicles[self.direction][self.lane][self.index - 1].x +
vehicles[self.direction][self.lane][                    self.index - 1].image.get_rect().width +
movingGap))):
                    self.x -= self.speed
else:
                if          ((self.crossedIndex          ==          0)          or          (self.x          >
(                        vehiclesNotTurned[self.direction][self.lane][self.crossedIndex - 1].x +
vehiclesNotTurned[self.direction][self.lane][
                            self.crossedIndex - 1].image.get_rect().width + movingGap))):
                    self.x -= self.speed        elif (self.direction == 'up'):
if (self.crossed == 0 and self.y < stopLines[self.direction]):
```

```
            self.crossed = 1
vehicles[self.direction]['crossed'] += 1              if
(self.willTurn == 0):
                vehiclesNotTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesNotTurned[self.direction][self.lane]) - 1          if
(self.willTurn == 1):              if (self.lane == 1):                if (self.crossed == 0 or
self.y > stopLines[self.direction] - 60):
                    if ((self.y >= self.stop or (                    currentGreen == 3 and
currentYellow == 0) or self.crossed == 1) and (                    self.index == 0 or self.y >
(vehicles[self.direction][self.lane][self.index -
1].y +

vehicles[self.direction][self.lane][                                    self.index -
1].image.get_rect().height + movingGap) or
vehicles[self.direction][self.lane][self.index - 1].turned == 1)):
                    self.y -=
self.speed                else:
if (self.turned == 0):
                    self.rotateAngle += rotationAngle                    self.image =
pygame.transform.rotate(self.originalImage, self.rotateAngle)                    self.x -= 2
self.y -= 1.2                    if (self.rotateAngle == 90):
                        self.turned = 1
                        vehiclesTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1                else:
                    if (self.crossedIndex == 0 or (self.x >
(                    vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].x +
vehiclesTurned[self.direction][self.lane][                    self.crossedIndex -
```

```
1].image.get_rect().width + movingGap))):                    self.x -= self.speed
elif (self.lane == 2):              if (self.crossed == 0 or self.y > mid[self.direction]['y']):
                if ((self.y >= self.stop or (                    currentGreen == 3 and
currentYellow == 0) or self.crossed == 1) and (                    self.index == 0 or self.y >
(vehicles[self.direction][self.lane][self.index -
1].y +

vehicles[self.direction][self.lane][                    self.index -
1].image.get_rect().height + movingGap) or
vehicles[self.direction][self.lane][self.index - 1].turned == 1)):
                    self.y -=
self.speed                else:
if (self.turned == 0):
                    self.rotateAngle += rotationAngle                    self.image =
pygame.transform.rotate(self.originalImage, -self.rotateAngle)                    self.x += 1
self.y -= 1                    if (self.rotateAngle == 90):
                        self.turned = 1
                        vehiclesTurned[self.direction][self.lane].append(self)
self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1                else:
                    if (self.crossedIndex == 0 or (self.x <
(                    vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].x -
vehiclesTurned[self.direction][self.lane][                    self.crossedIndex -
1].image.get_rect().width - movingGap))):
                        self.x +=
self.speed          else:          if
(self.crossed == 0):
```

```
            if ((self.y >= self.stop or (currentGreen == 3 and currentYellow == 0)) and
(                   self.index == 0 or self.y >
(                   vehicles[self.direction][self.lane][self.index - 1].y +
vehicles[self.direction][self.lane][                self.index - 1].image.get_rect().height
+ movingGap))):                 self.y -= self.speed            else:
            if ((self.crossedIndex == 0) or (self.y >
(                   vehiclesNotTurned[self.direction][self.lane][self.crossedIndex - 1].y +
vehiclesNotTurned[self.direction][self.lane][                       self.crossedIndex -
1].image.get_rect().height + movingGap))):
                self.y -= self.speed

            # Initialization of signals with default values


def initialize():
   minTime = randomGreenSignalTimerRange[0]
maxTime = randomGreenSignalTimerRange[1]    if
(randomGreenSignalTimer):
      ts1 = TrafficSignal(0, defaultYellow, random.randint(minTime, maxTime))
signals.append(ts1)       ts2 = TrafficSignal(ts1.red + ts1.yellow + ts1.green, defaultYellow,
random.randint(minTime, maxTime))       signals.append(ts2)       ts3 =
TrafficSignal(defaultRed, defaultYellow, random.randint(minTime, maxTime))
signals.append(ts3)       ts4 = TrafficSignal(defaultRed, defaultYellow,
random.randint(minTime, maxTime))       signals.append(ts4)    else:
      ts1 = TrafficSignal(0, defaultYellow, defaultGreen[0])
signals.append(ts1)       ts2 = TrafficSignal(ts1.yellow + ts1.green, defaultYellow,
defaultGreen[1])       signals.append(ts2)       ts3 = TrafficSignal(defaultRed,
defaultYellow, defaultGreen[2])       signals.append(ts3)       ts4 =
```

```
TrafficSignal(defaultRed, defaultYellow, defaultGreen[3])
signals.append(ts4)     repeat()


def repeat():
   global currentGreen, currentYellow, nextGreen     while
(signals[currentGreen].green > 0):  # while the timer of current green signal is not zero
updateValues()        time.sleep(1)     currentYellow = 1  # set yellow signal on
   # reset stop coordinates of lanes and vehicles
   for i in range(0, 3):        for vehicle in
vehicles[directionNumbers[currentGreen]][i]:
         vehicle.stop = defaultStop[directionNumbers[currentGreen]]     while
(signals[currentGreen].yellow > 0):  # while the timer of current yellow signal is not zero
updateValues()        time.sleep(1)     currentYellow = 0  # set yellow signal off

   # reset all signal times of current signal to default/random times     if
(randomGreenSignalTimer):
      signals[currentGreen].green     =     random.randint(randomGreenSignalTimerRange[0],
randomGreenSignalTimerRange[1])     else:
      signals[currentGreen].green = defaultGreen[currentGreen]
signals[currentGreen].yellow = defaultYellow     signals[currentGreen].red =
defaultRed

   currentGreen = nextGreen  # set next signal as green signal     nextGreen = (currentGreen
+ 1) % noOfSignals  # set next green signal     signals[nextGreen].red =
signals[currentGreen].yellow + signals[        currentGreen].green  # set the red time of next
to next signal as (yellow time + green time) of next signal     repeat()
```

```
# Update values of the signal timers after every second
def updateValues():     for i in range(0, noOfSignals):
      if (i == currentGreen):
if (currentYellow == 0):
signals[i].green -= 1
else:
signals[i].yellow -= 1        else:
         signals[i].red -= 1



# Generating vehicles in the simulation
def generateVehicles():     while (True):
      vehicle_type = random.choice(allowedVehicleTypesList)
lane_number = random.randint(1, 2)
      will_turn = 0        if
(lane_number == 1):
         temp = random.randint(0, 99)
if (temp < 40):               will_turn = 1
elif (lane_number == 2):
         temp = random.randint(0, 99)
if (temp < 40):               will_turn = 1
      temp = random.randint(0, 99)
direction_number = 0        dist =
[25, 50, 75, 100]        if (temp <
dist[0]):
         direction_number = 0
      elif (temp < dist[1]):
direction_number = 1        elif
```

```
(temp < dist[2]):
direction_number = 2        elif
(temp < dist[3]):
direction_number = 3
      Vehicle (lane_number, vehicleTypes[vehicle_type], direction_number,
directionNumbers[direction_number],
            will_turn)
time.sleep(1)



class Main:
   global allowedVehicleTypesList     i = 0
for vehicleType in allowedVehicleTypes:
if (allowedVehicleTypes[vehicleType]):
allowedVehicleTypesList.append(i)        i +=
1
   thread1 = threading.Thread(name="initialization", target=initialize, args=()) #
initialization     thread1.daemon = True     thread1.start()

   # Colours     black = (0,
0, 0)     white = (255, 255,
255)

   # Screensize
   screenWidth = 1400     screenHeight = 800
screenSize = (screenWidth, screenHeight)

   # Setting background image i.e. image of intersection     background
= pygame.image.load('images/intersection.png')
```

```
   screen = pygame.display.set_mode(screenSize)
pygame.display.set_caption("SIMULATION")

   # Loading signal images and font     redSignal =
pygame.image.load('images/signals/red.png')     yellowSignal =
pygame.image.load('images/signals/yellow.png')     greenSignal =
pygame.image.load('images/signals/green.png')     font = pygame.font.Font(None, 30)
thread2 = threading.Thread(name="generateVehicles", target=generateVehicles,
args=()) # Generating vehicles     thread2.daemon = True     thread2.start()

   while True:        for event in
pygame.event.get():           if
event.type == pygame.QUIT:
              sys.exit()

      screen.blit(background, (0, 0)) # display background in simulation              for i in
range(0,                    noOfSignals):  # display signal and set timer according to current
status: green, yello, or red
         if (i == currentGreen):
if (currentYellow == 1):
                signals[i].signalText = signals[i].yellow
screen.blit(yellowSignal, signalCoods[i])              else:
                signals[i].signalText = signals[i].green
screen.blit(greenSignal, signalCoods[i])           else:
if (signals[i].red <= 10):
                signals[i].signalText = signals[i].red
else:
```

```
                signals[i].signalText = "---"
screen.blit(redSignal, signalCoods[i])        signalTexts = ["",
"", "", ""]

    # display signal timer        for i
in range(0, noOfSignals):
        signalTexts[i] = font.render(str(signals[i].signalText), True, white, black)
screen.blit(signalTexts[i], signalTimerCoods[i])

    # display the vehicles
for vehicle in simulation:
        screen.blit(vehicle.image, [vehicle.x, vehicle.y])
vehicle.move()        pygame.display.update()


Main(
```

Buzzer Blowing and Alert Notification

```
import cv2 import numpy as np import
threading import pygame

# ----------- INIT SOUND ----------- pygame.mixer.init()
pygame.mixer.music.load("Beep_warning_sound.mp3")  # <-- put your sound file here

# ----------- LOAD VIDEO -----------
video_path = "video1.mp4"  # <-- replace with your video file
cap = cv2.VideoCapture(video_path)

ret, prev_frame = cap.read()
if not ret:     print("Error
loading video")
   exit()

prev_frame = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)

# ----------- SETTINGS -----------
MOTION_THRESHOLD = 5000000  # adjust if needed buzzing =
False  # prevents repeated triggering


# ----------- BUZZER FUNCTION ----------- def
buzz():
   global buzzing
buzzing = True
```

```
    pygame.mixer.music.play()
    pygame.time.delay(2000)  # play for 2 seconds     pygame.mixer.music.stop()

    buzzing = False


# ----------- MAIN LOOP ----------- while
cap.isOpened():
    ret, frame = cap.read()
if not ret:         break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Compute motion difference     diff
= cv2.absdiff(prev_frame, gray)
motion_score = np.sum(diff)

    # Show video
    cv2.imshow("Video Playback", frame)

    # Detect collision (approx)     if motion_score >
MOTION_THRESHOLD and not buzzing:
        print("Collision detected!")
        print(" Alert !! the accidicent of vehicle is there")
threading.Thread(target=buzz).start()

    prev_frame = gray

    # Exit on ESC key     if
cv2.waitKey(30) & 0xFF == 27:
        break

# ----------- CLEANUP -----------
cap.release() cv2.destroyAllWindows()
pygame.quit()
```

RESULTS AND DISCUSSION

6.1 Quantitative Results

The performance of the proposed system is evaluated based on its ability to detect sudden motion events that indicate possible vehicle collisions. The system processes video frames and generates a motion score, which is compared against a predefined threshold.

- The system successfully detected **major collision events** in test video clips
- Detection accuracy depends on the selected motion threshold
- Faster processing is achieved due to the use of lightweight techniques
- Real-time performance is maintained with minimal delay

Basic evaluation metrics considered:

- Detection rate of collision events
- Number of false positives (non-collision events detected as collisions)
- Processing time per frame

6.2 Qualitative Analysis

The system performs effectively in scenarios where there is a **sudden and significant change in motion**, such as vehicle crashes. Visual observation shows that:

- Clear collisions are detected accurately
- The alert system responds immediately with sound notification
- The system works well in stable camera conditions

However:

- Sudden camera movement may trigger false detection
- Lighting variations can affect detection performance
- Slow or minor collisions may not be detected

6.3 Strengths of the Model

- Simple and easy to implement
- Requires low computational resources
- Provides real-time processing capability
- Does not require large training datasets
- Can be integrated into existing traffic monitoring systems

6.4 Limitations

- Not based on advanced AI or deep learning models
- Relies on motion detection rather than actual object interaction
- Sensitive to environmental changes (lighting, camera movement)
- Limited accuracy in complex real-world scenarios

6.5 Ethical Considerations

The use of video-based surveillance systems raises certain ethical concerns:

- **Privacy Issues:** Continuous monitoring may affect individual privacy
- **Data Security:** Video data must be stored and handled securely
- **Responsible Use:** The system should be used only for safety and traffic management purposes
- **Bias and Misinterpretation:** Incorrect detection may lead to false alerts

Proper guidelines and safeguards should be implemented to ensure ethical and responsible use of the system.

TESTING

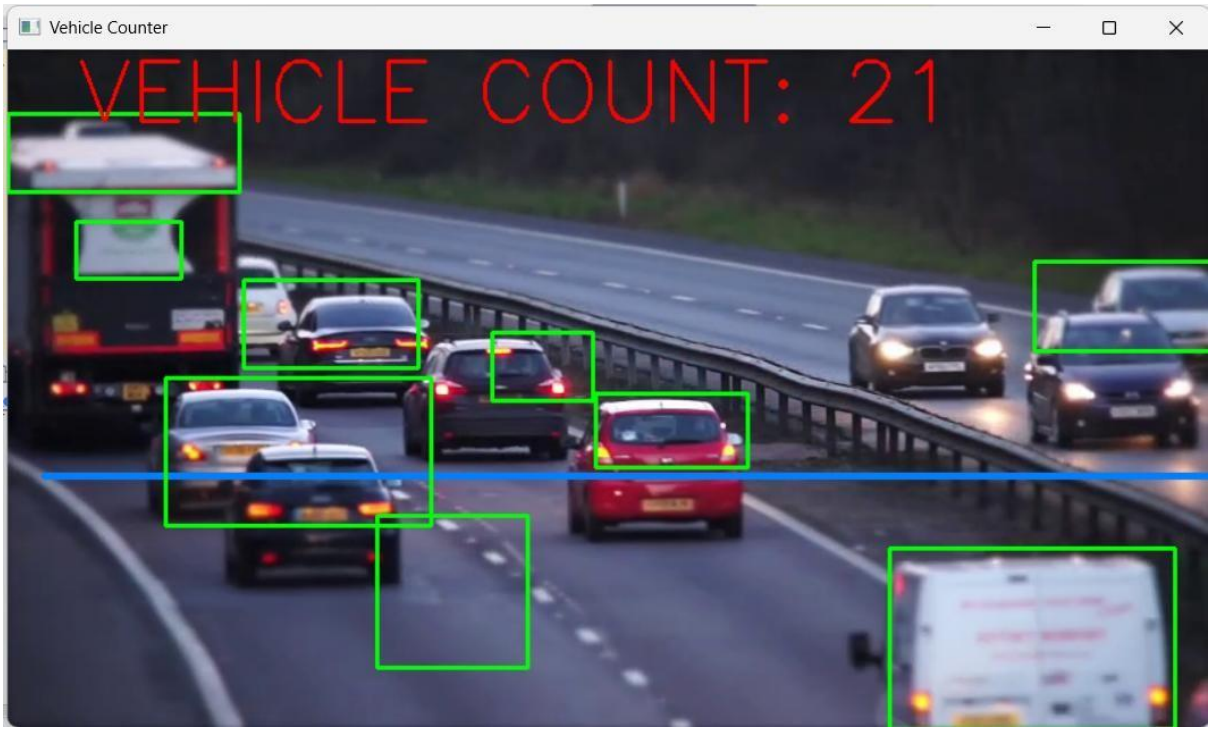
Testing of the Smart Traffic Management System is a critical step to ensure that the system functions accurately, efficiently, and reliably under real-world traffic conditions. The testing process involves multiple stages, including **unit testing**, **integration testing**, and **system testing**. In the unit testing phase, individual components such as traffic signal control

algorithms, vehicle counting modules, and accident detection using computer vision are tested independently to verify their correct operation. Integration testing ensures that all modules work together seamlessly, for example, verifying that traffic sensor data correctly influences signal timing adjustments and that accident alerts are properly communicated to the control centre. System testing is conducted to evaluate the overall performance of the Smart Traffic Management System under simulated or live traffic conditions, assessing parameters such as traffic flow improvement, congestion reduction, response to incidents, and emergency vehicle prioritisation. Performance testing may also include stress tests to determine how the system handles high traffic volumes or multiple simultaneous incidents. Additionally, the usability of dashboards and driver information platforms is tested to ensure clarity, responsiveness, and effectiveness of real-time alerts. The testing process is essential to identify and fix any errors, optimise system performance, and validate that the Smart Traffic Management System meets its functional and non-functional requirements, ensuring safer and more efficient urban traffic management.

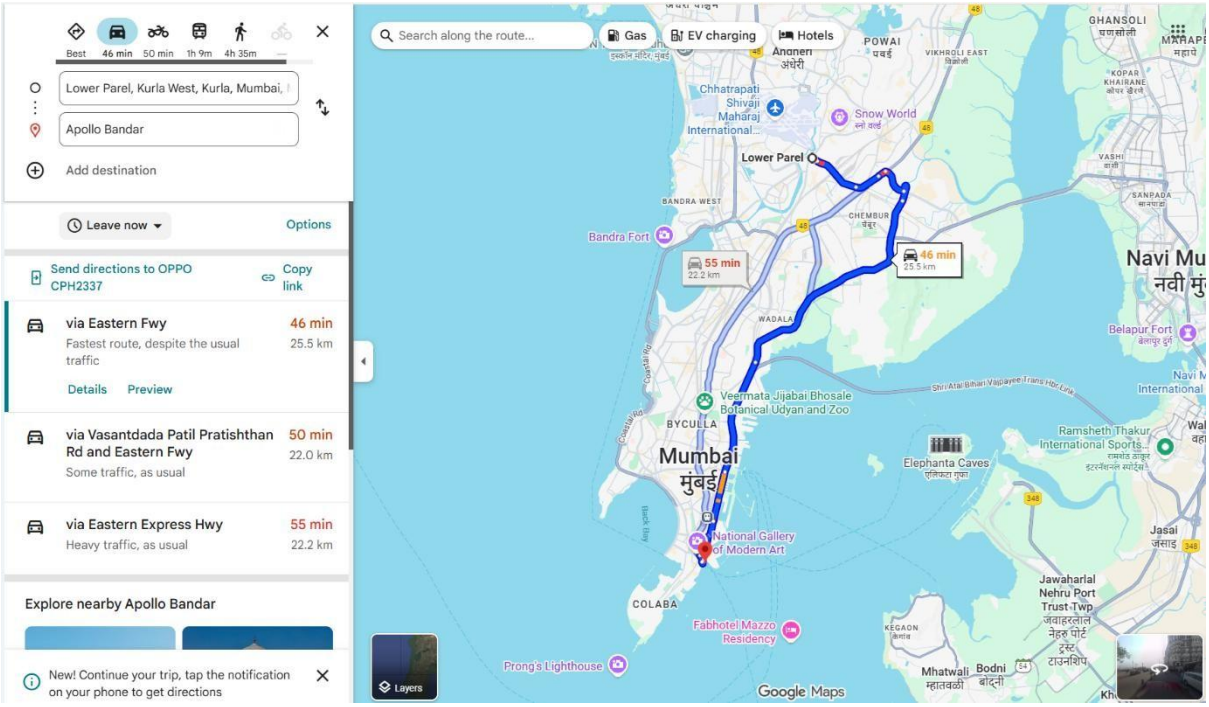
OUTPUT DESCRIPTION

The Smart Traffic Management System generates several outputs aimed at improving traffic flow, safety, and decision-making for both authorities and drivers. One of the primary outputs is **real-time traffic data**, including vehicle counts, speeds, and congestion levels at intersections and roadways, which is displayed on centralised dashboards for traffic operators. Another key output is **dynamic signal control**, where traffic light durations are adjusted automatically based on the analysed traffic conditions to reduce waiting time and minimise congestion. The system also provides **incident alerts**, detecting accidents, road blockages, or unusual traffic patterns and notifying authorities promptly for quick response. For drivers, the Smart Traffic Management System produces **real-time traffic updates and route recommendations** through mobile applications or digital signboards, helping them avoid congested areas and optimise travel time. Additionally, the system can generate **historical traffic reports** and visualisations for planning and analysis, including peak traffic periods, accident-prone zones, and overall traffic trends. By combining real-time monitoring, predictive analytics, and automated control, the Smart Traffic Management System ensures that its outputs support efficient, safe, and intelligent urban traffic

management.



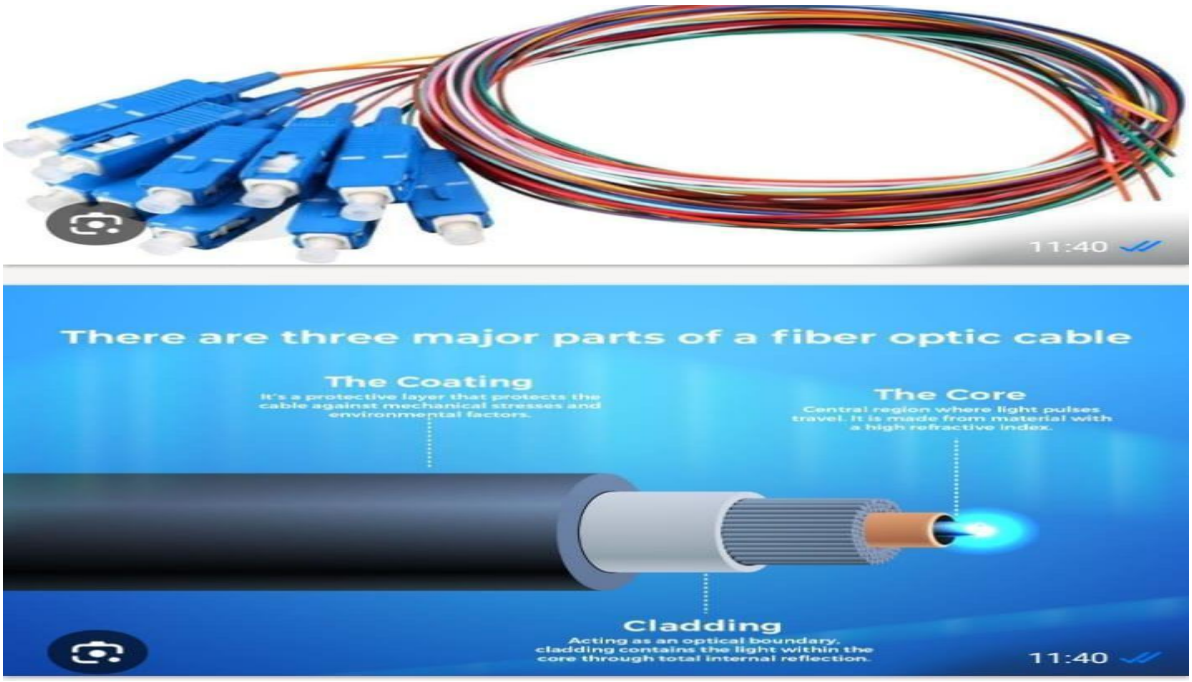
VEHICLE COUNTER



SHOWS PATH

TRAFFIC MANAGEMENT

COLLISION DETECTION



OPTICAL FIBRE USED FOR THE SIGNAL MANAGEMENT

CONCLUSION AND FUTURE WORK

7.1 Summary of Findings

This study presents a **Car Collision Detection with Alert System** as a component of a smart traffic management solution. The system successfully demonstrates how video-based motion analysis can be used to detect possible collision events. By processing video frames and identifying sudden motion changes, the system is able to trigger alerts in real time.

The results show that the proposed approach is effective for detecting major collision scenarios, while maintaining low computational complexity and real-time performance. The system also highlights the importance of integrating intelligent technologies in traffic monitoring to improve road safety and response time.

This research presents an effective solution for rapid growth of traffic flow particularly in big cities which is increasing day by day and traditional systems have some limitations as they fail to manage current traffic effectively. Keeping in view the state of the art approach for traffic management systems, a smart traffic management system is proposed to control

road traffic situations more efficiently and effectively. It changes the signal timing intelligently according to traffic density on the particular roadside and regulates traffic flow by communicating with local server more effectively than ever before. The decentralised approach makes it optimised and effective as the system works even if a local server or centralised server has crashed. The centralised server communicates the nearest rescue department in case of an emergency situation which provides timely human safety. Moreover, a user can ask about future traffic level at particular road hence avoiding wastage of time in traffic jams. The system also provides useful information to higher authorities that can be used in road planning which helps in optimal usage of resources, the Car Collision Detection with Alert System is a simple and effective prototype that demonstrates how video processing can be used for accident detection. While it currently relies on motion-based detection, it provides a strong foundation for developing more advanced AI-based traffic monitoring systems. With further improvements, this system can play a significant role in enhancing road safety and emergency response.

7.2 Technical Contributions

The key technical contributions of this project include:

- Development of a **motion-based collision detection system** using computer vision
- Implementation of real-time video processing using Python and OpenCV
- Integration of an **alert mechanism** using sound notifications
- Use of **threading** to ensure smooth and uninterrupted video playback
- Design of a **simple and cost-effective prototype** for traffic monitoring

7.3 Ethical Implications

The deployment of such systems involves important ethical considerations:

- **Privacy Concerns:** Continuous video monitoring may impact individual privacy
- **Data Protection:** Proper measures must be taken to secure video data
- **Responsible Usage:** The system should be used only for safety and monitoring purposes
- **False Alerts:** Incorrect detection may cause unnecessary panic or confusion

It is essential to follow ethical guidelines and ensure transparency in the use of such technologies.

7.4 Recommendations for Improvement

The system can be further enhanced in the following ways:

- Integration of **AI and deep learning models (e.g., YOLO)** for accurate vehicle detection
- Implementation of **vehicle tracking techniques (e.g., DeepSORT)**
- Improvement in detection accuracy under varying lighting and environmental conditions
- Real-time integration with **CCTV and smart city infrastructure**
- Development of a user-friendly interface for monitoring and alerts

7.5 Final Remarks

In conclusion, the proposed system provides a **simple yet effective approach** for detecting vehicle collisions using video analysis. While it currently relies on motion-based techniques, it serves as a strong foundation for building more advanced, AI-driven smart traffic management systems. With further improvements and integration of modern technologies, such systems have the potential to significantly enhance **road safety, traffic efficiency, and urban mobility**.

REFERENCES

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection.*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
2. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection.* arXiv preprint arXiv:2004.10934.
3. Jocher, G. et al. (2023). *YOLOv8 by Ultralytics.* Available at:https://github.com/ultralytics/ultralytics
4. Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). *Simple Online and Realtime Tracking (SORT).* IEEE International Conference on Image Processing (ICIP).
5. Wojke, N., Bewley, A., & Paulus, D. (2017). *Simple Online and Realtime Tracking with a Deep Association Metric (DeepSORT).* IEEE International Conference on Image Processing (ICIP).

6. Bradski, G. (2000). *The OpenCV Library.* Dr. Dobb's Journal of Software Tools.
7. Szeliski, R. (2010). *Computer Vision: Algorithms and Applications.*Springer.
8. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.*MIT Press.
9. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.*Springer.
10. OpenCV Documentation. Available at: https://docs.opencv.org/
11. Pygame Documentation. Available at: https://www.pygame.org/docs/
12. World Health Organization (WHO). (2023). *Global Status Report on Road Safety.* Available at: https://www.who.int/
13. Ministry of Road Transport and Highways, Government of India. (2022). *Road Accidents in India Report.*

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all those who have supported and guided me throughout the completion of this project titled **Smart Traffic Management System using Machine Learning**. First and foremost, I would like to thank my project guide, Dr. Kavita Chourasia, Dr. Kamini Maheshwar and Dr. Divakar Singh for their valuable guidance, continuous support, and encouragement at every stage of this work. Their insights and suggestions greatly contributed to the successful completion of this project.

I am also thankful to the faculty members **of Department of Computer Science and Engineering, University of Institute of Technology, Barkatullah University of Bhopal (M.P)** for providing the necessary resources and academic support required for this study.

I extend my heartfelt thanks to my team members — **Akruti Patil and Anjali Chaurasia** for their cooperation, teamwork, and dedication throughout the project. Their collective efforts made this work possible.

I would also like to acknowledge my family and friends for their constant motivation and moral support, which helped me overcome challenges during the project.

Finally, I express my gratitude to all those who directly or indirectly contributed to the successful completion of this work.

APPENDICES

Appendix A: Dataset and Annotation Details

The dataset used in this project consists of traffic video clips representing normal traffic conditions and accident scenarios. These videos are collected from publicly available sources and sample recordings.

- Contains different traffic situations such as moving vehicles, intersections, and collisions
- Frames are extracted for processing and analysis
- No manual annotation is required as the system uses motion-based detection
- Future enhancements may include annotated datasets for training AI models

Appendix B: Model Architecture and Parameters

The system is based on a **motion detection approach** using frame differencing.

- Converts frames to grayscale for faster processing
- Computes pixel-wise differences between consecutive frames
- Uses a predefined **motion threshold** to detect collisions

Key Parameters:

- Motion Threshold value
- Frame rate (FPS)
- Alert sound duration

Advanced versions may include deep learning models such as YOLO with additional training parameters.

Appendix C: System Implementation and Interface

The system is implemented using Python and integrates multiple libraries:

- **OpenCV:** Video capture and processing
- **NumPy:** Frame difference computation
- **Pygame:** Alert sound generation
- **Threading:** Enables parallel execution

Interface Features:

- Real-time video playback
- Collision detection alerts
- Automatic sound notification

Future scope includes a **web-based interface using Flask** for user interaction.

Appendix D: Limitations and Recommendations

Limitations:

- Based on motion detection, not actual object-level collision
- Sensitive to camera movement and lighting changes
- May produce false alerts or miss slow collisions

Recommendations:

- Integrate AI models (e.g., YOLO) for improved accuracy
- Apply tracking algorithms (e.g., DeepSORT)
- Enhance robustness under different environmental conditions
- Deploy with real-time CCTV systems