Smart Congestion Control System

SMART CONGESTION CONTROL SYSTEM

PROJECT REPORT

submitted by

ABBY ANISH VARGHESE (SCT20CS001)

Guided by

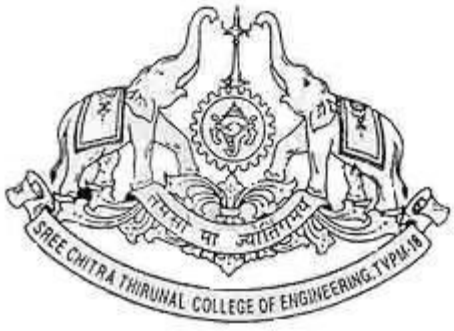
Smt. Soja Salim

to

the APJ Abdul Kalam Technological University
in partial fulfillment of the requirements for the award of the

degree of

Bachelor of
Technology in
Computer Science and Engineering



Department of Computer Science and Engineering Sree Chitra Thirunal College of Engineering Pappanamcode, Thiruvananthapuram

AUGUST 2023

DECLARATION

We undersigned hereby declare that the project report "Smart Congestion Control System", submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by us under supervision of Smt. Soja Salim. This submission represents our ideas in our own words and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Thiruvananthapuram
Date: 26-06-2023

Names of Student:
ABBY ANISH VARGHESE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SREE CHITRA

THIRUNAL COLLEGE OF ENGINEERING, THIRUVANANTHAPURAM

CERTIFICATE

This is to certify that the report entitled “Smart Congestion Control System ” submitted by **Abby Anish Varghese(SCT20CS001)** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering is a bonafide record of the project work carried out by them under my/our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Project Coordinator Project Coordinator

Project Guide HEAD OF THE DEPT

ACKNOWLEDGEMENT

We are grateful to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and everyone who has helped us make this project a success.

We heartily thank our **Head of Department, Dr. Soniya B, Dept. of Computer Science and Engineering, Sree Chitra Thirunal College of Engineering** for her constant encouragement and inspiration in taking up this project.

We heartily thank our Project coordinator **Smt. Soja Salim, Assistant Professor, Dept. of Computer science and Engineering**, for his constant follow up and advice throughout the course of the Project work.

We gratefully thank our Project guide**, Smt. Neenu J.S, Assistant Professor, Dept. of Computer Science and Engineering**, for her encouragement and advice throughout the course of the Project work.

Special thanks to all the staff members of the Computer Science and Engineering Department for their help and kind cooperation.

Lastly we thank our parents and friends for their encouragement and support given to us in order to finish this precious work.

ABSTRACT

Traffic congestion is becoming one of the critical issues with the increasing population and automobiles in cities. Traffic jams not only cause extra delay and stress for the drivers but also increase fuel consumption and air pollution.

According to reports 3 of the top 10 cities facing the most traffic congestion are in India viz. Mumbai, Bengaluru, and New Delhi. People are compelled to spend hours stuck in traffic jams, wasting away their precious time commuting. Current traffic light controllers use a fixed timer and do not adapt according to the real-time traffic on the road.

In an attempt to reduce traffic congestion, we developed an improved traffic management system in the form of a Computer Vision-based traffic light controller that can autonomously adapt to the traffic situation at the traffic signal. The proposed system sets the green signal time adaptively according to the traffic density at the signal and ensures that the direction with more traffic is allotted a green signal for a longer duration of time as compared to the direction with lesser traffic.

INDEX

CHAPTER 1

INTRODUCTION

Our proposed system takes an image from the CCTV cameras at traffic junctions as input for real time traffic density calculation using image processing and object detection. This system can be broken down into 3 modules: Vehicle Detection module, Signal Switching Algorithm, and Simulation module.

The video file is passed on to the vehicle detection algorithm, which uses OpenCV and NumPy. The number of vehicles of each class, such as car, bike, bus, and truck, is detected, which is to calculate the density of traffic. The signal switching algorithm uses this density, among some other factors, to set the green signal timer for each lane. The red signal times are updated accordingly. The green signal time is restricted to a maximum and minimum value in order to avoid starvation of a particular lane. A simulation is also developed to demonstrate the system's effectiveness and compare it with the existing static system.

1.1 Purpose

The purpose of this report is to present a detailed description of the Smart Congestion Control System. It explains the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will function in real world conditions.

1.2 Intended Audience and Document Overview

This document is intended for students & faculties of SCTCE and the developers of this application. The document is meant as a means for them to easily see and refer to our overall product goals and commitments. It also serves an additional purpose of allowing the project team to make sure that the intended goals and functionalities of the project are being met.

1.3 Scope

The product is a simulation of the proposed smart traffic signaling system

The main aim of the prototype is to provide automated signal generation

The simulation depicts a junction whose signal duration are allocated as per density of vehicles

The signal switching functions in a cyclic manner and only the signal duration is increased or decreased as per vehicle count

The densest lanes is allocated more green signal time

CHAPTER 2

LITERATURE REVIEW

2.1 Research Papers

Paper-1:

"Modeling and Simulation of Smart Traffic Light System for Emergency Vehicle using Image Processing Techniques" by Sujin Jose Arul; Mithilesh B S; Shreyas L; Sufiyan; Gopal Kaliyaperumal; Jayasheel Kumar K A

Saving time is very essential for humans. Every day people are spending some time at the traffic signal due to the drawbacks of the conventional traffic light system. In the existing traffic light system, a defined timer system is used and it is working based on preset timing. Due to the preset timing, there is no flexibility of ON/OFF in the signal light based on the emergency vehicle and congestion of the vehicle. Sometimes an emergency vehicle like an ambulance needs to wait at a traffic signal for a long time and this would lead to a risk to a patient's life. Traffic police must personally identify an ambulance and release the congestion, but this is not possible as there are an enormous number of vehicles present these days. This project aims to provide a solution for the issue in the conventional system. The model was designed using an image processing system that reads the image and determines the presence of an emergency vehicle and the density of vehicles in each lane the ON/OFF signal for the particular lane will be given to the traffic light system which helps to reduce the unnecessary waiting time of vehicles. The system calculates the vehicle's density and to detect the emergency vehicle using image processing to provide the green light signal to the lane. This project used OpenCV and Yolo (you only look once)algorithm in the image processing method to develop the system. The simulation has been done on the proposed smart traffic system and it identifies that the proposed system is efficient. Multiple times of programming and testing have been done on the proposed system to ensure accuracy and for validation.

Paper-2:

"Real-Time Adaptive Traffic Control System For Smart Cities" by Shyam Shankaran R; Logesh Rajendran

In-country like India, billions of people start and end each working day stuck in traffic or commuting on congested trains and buses. It is vital to the quality of life to enhance the everyday commute. By 2025, cities that implement smart mobility systems on average, reduce commuting cycles by 15-20 percent, with some individuals experiencing even greater reductions. Depending on each city's density, current transit facilities, and commuting habits, the capacity associated with each application is highly variable. Slowed synchronization of traffic signals leads to traffic congestion and delays. The pre-programmed, regular signal timing patterns are employed in traditional signal systems. To overcome the problems of traditional traffic control systems, there is a shift in adaptation to an Adaptive traffic control system. The Adaptive Traffic Control System (ATCS) is a traffic management technique that modifies or adapts the timing of traffic signals based on the real demand for traffic and achieved using a control system that includes both hardware and software, where hardware is the sensor used for real-time traffic density estimation and software is designed using captured data analysis of the city's current traffic flow. This paper depicts a model of camera-based traffic monitoring and processing system which reduces the cycle time and possesses special provisions for emergency vehicles.

Paper-3:

"Artificial Intelligence Based Smart Traffic Management System Using Video Processing" by Abhijeet Choudhary;Akash Gupta; Akshay Dhuri; Prof. Nilima Nikam

As the population of the modern cities is increasing day by day due to which vehicular travel is increasing which leads to congestion problems. Video processing techniques caused by using this we can easily calculate the density of traffic present on the road. The system will detect vehicles through images instead of using electronic sensors embedded in the pavement. A camera will be installed alongside the traffic light. It will capture image sequences. Image processing is a better technique to control the state change of the traffic light. In our proposed system there we will be four cameras in one intersection for a four way road. A CPU will be connected with these cameras which will be responsible for video processing. This processing unit takes pictures from the camera and compares all pictures and counts the vehicle present on the road. After comparing the allocated time first on that road where vehicle count in more, this process happened again and again and reduced the traffic conjunction.

2.2 Problem Statement

The current traffic management systems in urban areas face significant challenges in effectively managing traffic flow, resulting in congestion, inefficient use of resources, and increased travel times. These issues have a detrimental impact on the environment, economy, and overall quality of life. There is a pressing need for a smart traffic management system that leverages advanced technologies to optimize traffic flow, enhance safety, and improve overall efficiency.

2.3 Proposed Solution

As a solution to this problem we are planning to develop a smart traffic management system that utilizes machine learning algorithms to optimize the flow of traffic at junctions in urban areas. The system is designed to improve the efficiency of traffic management by collecting real-time data from various sources including sensors and cameras. Using this data we can reduce congestion at signals by effectively allocating run time as per vehicle count in that particular track.

CHAPTER 3

SOFTWARE REQUIREMENTS SPECIFICATION

3.1 Overall Description

3.1.1 Product Perspective

This simulation defines an ideal world traffic condition where time allotted for the green signal is not static,instead allocated dynamically as per vehicle density of that particular lane calculated by an algorithm.

3.2 External Interface Requirements

3.2.1 Software Interfaces

1. **OpenCV**
    - OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library.
    - It provides a comprehensive set of functions for reading, manipulating, and analyzing images and videos.It offers pre-trained models and algorithms for object detection, face recognition, motion tracking, and feature extraction.
2. **NumPy**
    - NumPy (Numerical Python) is a powerful library in Python for numerical computing.
    - It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
3. **Pygame**
    - Pygame is a cross-platform open-source library in Python specifically designed for game development and multimedia applications.
    - It provides functionality and tools for creating 2D games, animations, and interactive multimedia experiences.

3.3 Functional Requirements

The simulation will show the traffic on a road graphically.

System will allow the admin of the system to rewrite traffic data.

System will provide an interface to the admin to control the flow of traffic i.e. manage emergency situations and reconfigure devices.

System will allow the admin to adjust signal timing. According to the adjusted timing of signal the system should clear out congestion of the particular way and then the next route accordingly .

CHAPTER 4

SOFTWARE DESIGNS

4.1 System Architecture Design

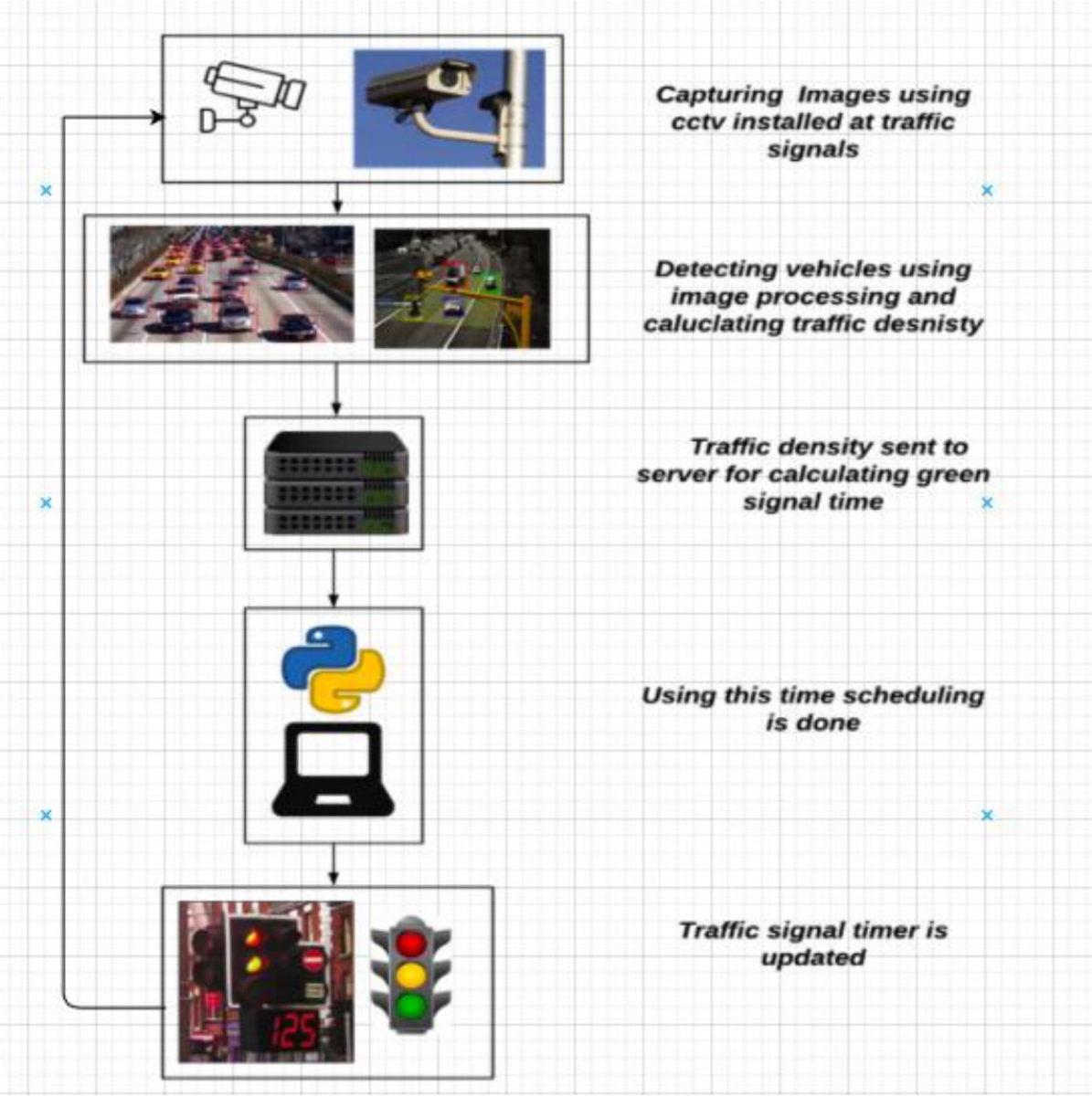


Figure 4.1

Our proposed system takes an image from the CCTV cameras at traffic junctions as input for real time traffic density calculation using image processing and object detection. This system can be broken down into 3 modules: Vehicle Detection module, Signal Switching Algorithm, and Simulation module.

The image is passed on to the vehicle detection algorithm, which uses OpenCV. The number of vehicles of each class, such as car, bike, bus, and truck, is detected, which is to calculate the density of traffic. The signal switching algorithm uses this density, among some other factors, to set the green signal timer for each lane. The red signal times are updated accordingly. The green signal time is restricted to a maximum and minimum value in order to avoid starvation of a particular lane. A simulation is also developed to demonstrate the system's effectiveness and compare it with the existing static system.

4.2 Use-case View

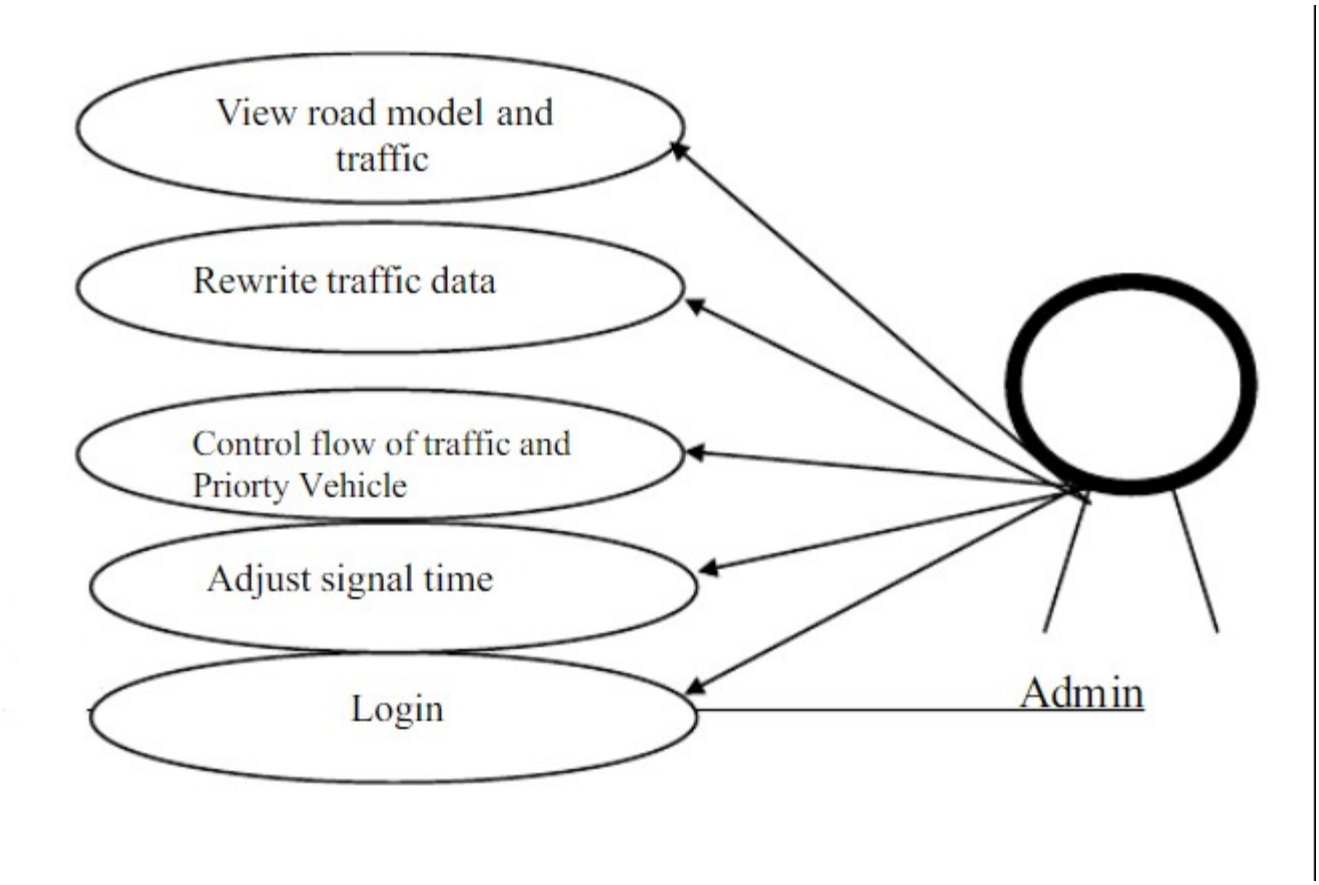


Figure 4.2 Use-Case Diagram

Actors:

Admin

Use Cases:

The use case starts when admin wants to view the road model

System saves the operation

This use case ends.

Pre-condition:

Admin must open the simulation

Post-condition:

Operation done successfully

CHAPTER 5

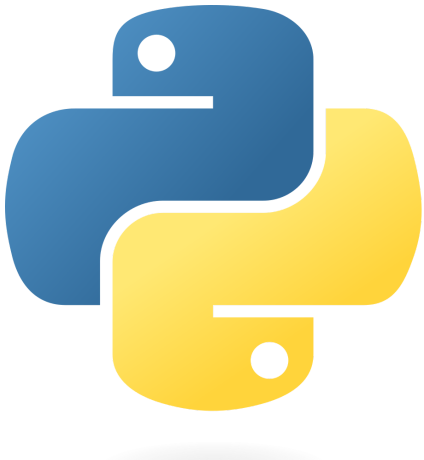
TECHNOLOGY STACK

5.1 Visual Studio Code

Visual Studio Code (VS Code) is a lightweight yet powerful source code editor developed by Microsoft. It provides a seamless coding experience with features like syntax highlighting, code completion, and intelligent suggestions. Its extensive library of extensions allows developers to customize and extend its functionality for various programming languages and frameworks. With an integrated terminal, version control support, and a built-in debugger, VS Code streamlines the development process and enhances collaboration.

5.2 Python

Python is a versatile and popular programming language known for its simplicity and readability. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming styles, giving developers flexibility in writing code. It has a vast standard library and a rich ecosystem of third-party packages, making it suitable for a wide range of applications, from web development and data analysis to scientific computing and machine learning.



5.3 OpenCV

OpenCV (Open Source Computer Vision Library) is a widely-used open-source library that provides a comprehensive set of computer vision and image processing algorithms. It offers a range of functions and tools for tasks such as image and video manipulation, object detection and recognition, feature extraction, and camera calibration. OpenCV supports various programming languages, including Python, C++, and Java, making it accessible to developers across different platforms. With its extensive collection of algorithms and easy-to-use interfaces, OpenCV simplifies the development of computer vision applications.

5.4 NumPy

NumPy (Numerical Python) is a powerful library in Python that provides support for efficient numerical operations on large arrays and matrices. It serves as a fundamental building block for scientific computing and data analysis in Python. NumPy's key feature is its ndarray (n-dimensional array) object, which allows for efficient storage and manipulation of large datasets. It provides a wide range of mathematical functions and operations that can be applied to these arrays, enabling fast and vectorized computations.

5.5 Pygame

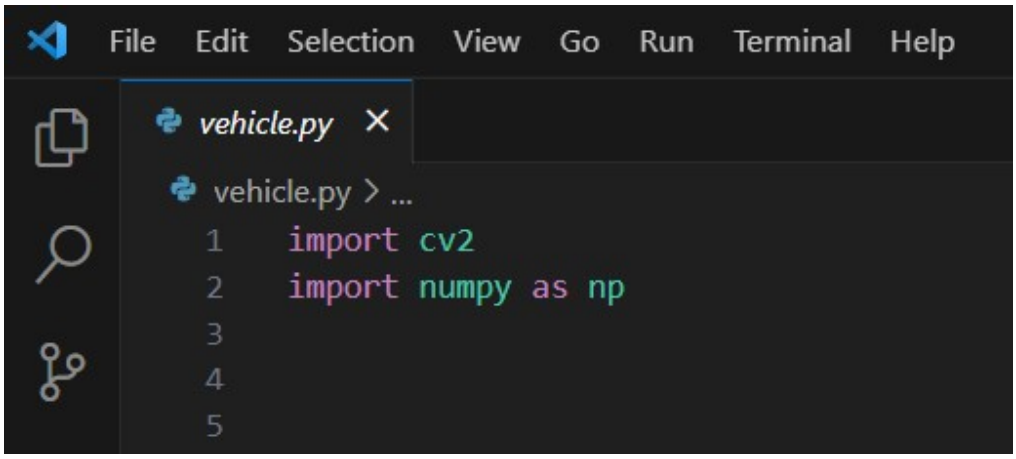
Pygame is a popular Python library specifically designed for game development and multimedia applications. It provides a simple and intuitive interface to create games, simulations, and interactive visualizations. Pygame is built on top of the Simple DirectMedia Layer (SDL), which gives it cross-platform compatibility and access to low-level multimedia functionalities like graphics, sound, and input devices

CHAPTER 6

IMPLEMENTATION

6.1 Vehicle Detection

Initially we are developing a vehicle detection program that counts the number of vehicles present in a given video file. This is to represent the functioning of real world cases, where the vehicle count is computed by the scanner in a camera system kept over a traffic post.



cv2 provides various functions and tools for image and video processing, including reading and writing images/videos, applying filters and transformations, performing object detection and tracking, and more. It is a widely-used library for computer vision tasks.

numpy is a fundamental library for numerical computing in Python. It provides support for efficient numerical operations on large arrays and matrices, along with a collection of mathematical functions. NumPy is often used in conjunction with OpenCV to process and manipulate image data.

```python
#Web camera
cap=cv2.VideoCapture('video.mp4')

min_width_rect = 80
min_height_rect = 80
count_line_position = 550

#initialize Subtractor
algo = cv2.bgsegm.createBackgroundSubtractorMOG()  #to remove background things

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 = []
offset=6    #allowable error btw pixel
counter=0
```

cap = cv2.VideoCapture('video.mp4') initializes the video capture object, cap, to read frames from the specified video file, 'video.mp4'.

min_width_rect and min_height_rect represent the minimum width and height criteria for the detected rectangles around vehicles. Any contour below these dimensions will be filtered out.

count_line_position defines the y-coordinate position of the counting line, which is the line across which vehicles are counted.

The line algo = cv2.bgsegm.createBackgroundSubtractorMOG() initializes the background subtractor algorithm (MOG - Mixture of Gaussians) using the cv2.bgsegm module. This algorithm is used to remove the background and focus on the moving objects (vehicles) in the video.

The function center_handle(x, y, w, h) calculates and returns the center coordinates (cx, cy) of a rectangle based on its top-left corner coordinates (x, y) and its width and height (w, h).

The detect list is used to store the center coordinates of the detected vehicles.offset represents the allowable error (in pixels) between the y-coordinate of a detected vehicle and the counting line. This is to account for slight variations in the vehicle's position.

counter is initialized as 0 and will be used to keep track of the number of vehicles that cross the counting line.

```
while True:
    ret,frame1=cap.read() #to loop video file
    grey = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY) #to convert to grayscale
    blur = cv2.GaussianBlur(grey,(3,3),5)
    #applying on each frame
    img_sub = algo.apply(blur)
    dilat = cv2.dilate(img_sub,np.ones((5,5)))
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
    dilatada = cv2.morphologyEx(dilat,cv2.MORPH_CLOSE,kernel)
    dilatada = cv2.morphologyEx(dilatada,cv2.MORPH_CLOSE,kernel)
    counterSahpe,h = cv2.findContours(dilatada,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

    cv2.line(frame1,(25,count_line_position),(1200,count_line_position),(255,127,0),3) #detector line

    #drawing rectangle frames for vehicles
    for (i,c) in enumerate(counterSahpe):
        (x,y,w,h) = cv2.boundingRect(c)
        validate_counter = (w>=min_width_rect) and (h>=min_height_rect)
        if not validate_counter :
            continue

        cv2.rectangle(frame1,(x,y),(x+w,y+h),(0,0,255),2)
        cv2.putText(frame1,"Vehicle"+str(counter),(x,y-20),cv2.FONT_HERSHEY_COMPLEX,1,(255,244,0),2)

        center=center_handle(x,y,w,h)
        detect.append(center)
        cv2.circle(frame1,center,4,(0,0,255),-1)

        for (x,y) in detect:
            if y<(count_line_position+offset) and y>(count_line_position-offset):
                counter+=1
                cv2.line(frame1,(25,count_line_position),(1200,count_line_position),(0,127,255),3)
                detect.remove((x,y))

                print("Vehicle Counter:"+str(counter))
```

The code enters an infinite loop with while True: to continuously process video frames.cap.read() reads the next frame from the video source, and the returned values ret and frame1 indicate whether the frame was successfully read.cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY) converts the frame to grayscale.cv2.GaussianBlur(grey, (3,3), 5) applies a Gaussian blur to the grayscale image. algo.apply(blur) applies some algorithm (represented by algo) to the blurred image.Various morphological operations (cv2.dilate and cv2.morphologyEx) are performed on the resulting image

to enhance the detected objects.cv2.findContours finds contours in the image, representing potential objects or shapes.cv2.line draws a line on frame1 to serve as a detector line for vehicle counting.

A loop iterates over the detected contours and filters them based on specified width and height criteria. Valid contours are then used to draw rectangles around the vehicles on frame1.Additional visual elements like text, circles, and counting logic are applied based on the detected vehicles.The processed frame is displayed using cv2.imshow.cv2.waitKey(1) == 13 waits for the "Enter" key (ASCII code 13) to be pressed. If so, the loop is broken, and the video processing ends.cv2.destroyAllWindows() closes any open windows, and cap.release() releases the video capture object.Overall, this code processes a video frame by frame, performs object detection and tracking, and counts the number of vehicles crossing a specified line in the video.

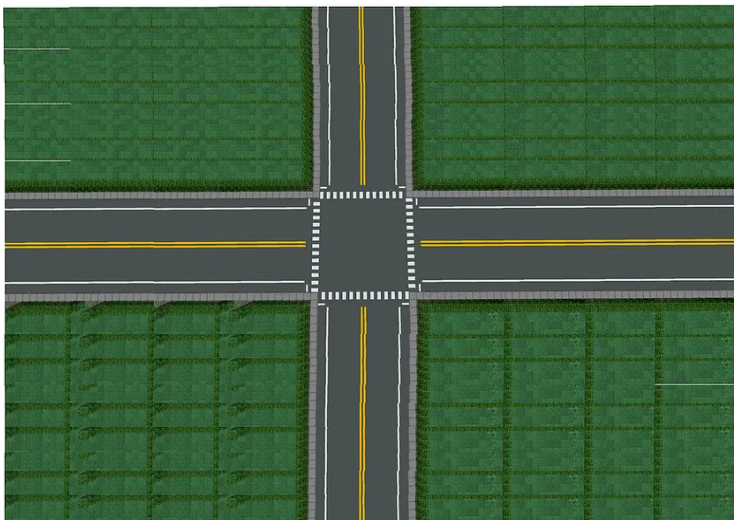
—----------

6.2 Simulation

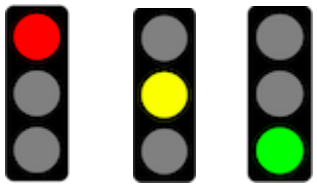
We are also developing a simulation from scratch using Pygame to simulate the movement of vehicles across a traffic intersection having traffic lights with a timer. It contains a 4-way traffic intersection with traffic signals controlling the flow of traffic in each direction. Each signal has a timer on top of it which shows the time remaining for the signal to switch from green to yellow, yellow to red, or red to green. Vehicles such as cars, bikes, buses, and trucks are generated, and their movement is controlled according to the signals and the vehicles around them.

Getting the images

4-way intersection



Traffic Signals: Red, Yellow, and Green



The vehicles are resized into a size of our choice ,the same process is done for the traffic signals.

As, we have images of vehicles facing towards the right only. So we rotate the images of vehicles and save each of them to get images facing all directions, as shown in the image below.

We create a folder ‘Traffic Intersection Simulation’ and within that create a folder ‘images’. Here we will store all of these images. The folder structure is as shown below where:

- down: contains images of vehicles facing down
- up: contains images of vehicles facing up
- left: contains images of vehicles facing left
- right: contains images of vehicles facing right
- signals: contains the images of traffic signals

Defining constants

defining constants that will be used in the movement of vehicles in the simulation as well as in control of traffic signal timers.

```
# Default values of signal times
defaultRed = 150
defaultYellow = 5
defaultGreen = 20
defaultMinimum = 10
defaultMaximum = 60

signals = []
noOfSignals = 4
simTime = 300       # change this to change time of simulation
timeElapsed = 0

currentGreen = 0   # Indicates which signal is green
nextGreen = (currentGreen+1)%noOfSignals
currentYellow = 0   # Indicates whether yellow signal is on or off

# Average times for vehicles to pass the intersection
carTime = 2
bikeTime = 1
rickshawTime = 2.25
busTime = 2.5
truckTime = 2.5

# Count of cars at a traffic signal
noOfCars = 0
noOfBikes = 0
noOfBuses =0
noOfTrucks = 0
noOfRickshaws = 0
noOfLanes = 2

# Red signal time at which cars will be detected at a signal
detectionTime = 5

speeds = {'car':2.25, 'bus':1.8, 'truck':1.8, 'rickshaw':2, 'bike':2.5}
```

The coordinates below are also extracted by opening the 4-way intersection image in Paint/Preview and getting the pixel values.

```
# Coordinates of 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:'rickshaw', 4:'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)]
vehicleCountCoods = [(480,210),(880,210),(880,550),(480,550)]
vehicleCountTexts = ["0", "0", "0", "0"]

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

mid = {'right': {'x':705, 'y':445}, 'down': {'x':695, 'y':450}, 'left': {'x':695, 'y':425}, 'up': {'x':695, 'y':400}}
rotationAngle = 3

# Gap between vehicles
gap = 15    # stopping gap
gap2 = 15   # moving gap
```

Defining Classes

We have 2 classes that we need to define.

- Traffic Signal: To generate 4 traffic signals for our simulation. So we build a TrafficSignal class that has the following attributes:
- red: Value of red signal timer
- yellow: Value of yellow signal timer
- green: Value of green signal timer
- signalText: Value of timer to display

2. Vehicle: This is a class that represents objects of vehicles that we will be generating in the simulation. The Vehicle class has the following attributes and methods:

- vehicleClass: Represents the class of the vehicle such as car, bus, truck, or bike
- speed: Represents the speed of the vehicle according to its class
- direction_number: Represents the direction — 0 for right, 1 for down, 2 for left, and 3 for up
- direction: Represents the direction in text format
- x: Represents the current x-coordinate of the vehicle
- y: Represents the current y-coordinate of the vehicle
- crossed: Represents whether the vehicle has crossed the signal or not
- index: Represents the relative position of the vehicle among the vehicles moving in the same direction and the same lane
- image: Represents the image to be rendered
- render(): To display the image on screen
- move(): To control the movement of the vehicle according to the traffic light and the vehicles ahead

```
class TrafficSignal:
    def __init__(self, red, yellow, green, minimum, maximum):
        self.red = red
        self.yellow = yellow
        self.green = green
        self.minimum = minimum
        self.maximum = maximum
        self.signalText = "30"
        self.totalGreenTime = 0

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.stop = stops[direction][lane]
        self.index = len(vehicles[direction][lane]) - 1
        path = "images/" + direction + "/" + vehicleClass + ".png"
        self.originalImage = pygame.image.load(path)
        self.currentImage = pygame.image.load(path)
```

In the constructor, after initializing all the variables, we are checking if there are vehicles already present in the same direction and lane as the current vehicle. If yes, we need to set the value of 'stop' of the current vehicle taking into consideration the value of 'stop' and the width/height of the vehicle ahead of it, as well as the stoppingGap. If there is no vehicle ahead already, then the stop value is set equal to defaultStop. This value of stop is used to control where the vehicles will stop when the signal is red. Once this is done, we update the coordinates from where the vehicles are generated. This is done to avoid overlapping of newly generated vehicles with the existing vehicles when there are a lot of vehicles stopped at a red light.

For each direction, we first check if the vehicle has crossed the intersection or not. This is important because if the vehicle has already crossed, then it can keep moving regardless of the signal being green or red. So when the vehicle crossed the intersection, we set the value of crossed to 1. Next, we decide when the vehicle moves and when it stops. There are 3 cases when the vehicle moves:

1. If it has not reached its stop point before the intersection
2. If it has already crossed the intersection
3. If the traffic signal controlling the direction in which the vehicle is moving is Green

Only in these 3 cases, the coordinates of the vehicle are updated by incrementing/decrementing them by the speed of the vehicle, depending on their direction of motion.

Creating objects of TrafficSignal class

Initialize 4 TrafficSignal objects, from top left to bottom left in a clockwise direction, with default values of signal timers. The red signal timer of ts2 is set equal to the sum of the yellow and green signal timer of ts1.

```
def initialize():
    ts1 = TrafficSignal(0, defaultYellow, defaultGreen, defaultMinimum, defaultMaximum)
    signals.append(ts1)
    ts2 = TrafficSignal(ts1.red+ts1.yellow+ts1.green, defaultYellow, defaultGreen, defaultMinimum, defaultMaximum)
    signals.append(ts2)
    ts3 = TrafficSignal(defaultRed, defaultYellow, defaultGreen, defaultMinimum, defaultMaximum)
    signals.append(ts3)
    ts4 = TrafficSignal(defaultRed, defaultYellow, defaultGreen, defaultMinimum, defaultMaximum)
    signals.append(ts4)
    repeat()
```

repeat() function

The function repeat() that is called at the end of the initialize() function above is a recursive function that runs our entire simulation. This is the driving force of our simulation.

```
def repeat():
    global currentGreen, currentYellow, nextGreen
    while(signals[currentGreen].green>0):   # while the timer of current green signal is not zero
        printStatus()
        updateValues()
        if(signals[(currentGreen+1)%(noOfSignals)].red==detectionTime):    # set time of next green signal
            thread = threading.Thread(name="detection",target=setTime, args=())
            thread.daemon = True
            thread.start()
            # setTime()
        time.sleep(1)
    currentYellow = 1   # set yellow signal on
    vehicleCountTexts[currentGreen] = "0"
    # reset stop coordinates of lanes and vehicles
    for i in range(0,3):
        stops[directionNumbers[currentGreen]][i] = defaultStop[directionNumbers[currentGreen]]
        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
        printStatus()
        updateValues()
        time.sleep(1)
    currentYellow = 0   # set yellow signal off

    # reset all signal times of current signal to default times
    signals[currentGreen].green = defaultGreen
    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()
```

The repeat() function first calls the updateValues() function every second to update the signal timers until the green timer of the currentGreen signal reaches 0. It then sets that signal to yellow and resets the stop value of all vehicles moving in the direction controlled by the currentGreen signal. It then calls the updateValues() function again after every second until the yellow timer of the currentGreen signal reaches 0. The currentYellow value is now set to 0 as this signal will turn red now. Lastly, the values of the currentGreen signal are restored to the default values, the value of currentGreen and nextGreen is updated to point to the next signals in the cycle, and the value of nextGreen signal's red timer is updated according to yellow and green of the updated currentGreen signal. The repeat() function then calls itself, and the process is repeated with the updated currentGreen signal.

updateValues() function

The function updateValues() updates the timers of all signals after every second.

```
def updateValues():
    for i in range(0, noOfSignals):
        if(i==currentGreen):
            if(currentYellow==0):
                signals[i].green-=1
                signals[i].totalGreenTime+=1
            else:
                signals[i].yellow-=1
        else:
            signals[i].red-=1
```

setTime() function

```
def setTime():
    global noOfCars, noOfBikes, noOfBuses, noOfTrucks, noOfRickshaws, noOfLanes
    global carTime, busTime, truckTime, rickshawTime, bikeTime
    os.system("say detecting vehicles, "+directionNumbers[(currentGreen+1)%noOfSignals])
    noOfCars, noOfBuses, noOfTrucks, noOfRickshaws, noOfBikes = 0,0,0,0,0
    for j in range(len(vehicles[directionNumbers[nextGreen]][0])):
        vehicle = vehicles[directionNumbers[nextGreen]][0][j]
        if(vehicle.crossed==0):
            vclass = vehicle.vehicleClass
            noOfBikes += 1
    for i in range(1,3):
        for j in range(len(vehicles[directionNumbers[nextGreen]][i])):
            vehicle = vehicles[directionNumbers[nextGreen]][i][j]
            if(vehicle.crossed==0):
                vclass = vehicle.vehicleClass
                if(vclass=='car'):
                    noOfCars += 1
                elif(vclass=='bus'):
                    noOfBuses += 1
                elif(vclass=='truck'):
                    noOfTrucks += 1
                elif(vclass=='rickshaw'):
                    noOfRickshaws += 1
    greenTime = math.ceil(((noOfCars*carTime) + (noOfRickshaws*rickshawTime)
        + (noOfBuses*busTime) + (noOfTrucks*truckTime)+ (noOfBikes*bikeTime))/(noOfLanes+1))
    print('Green Time: ',greenTime)
    if(greenTime<defaultMinimum):
        greenTime = defaultMinimum
    elif(greenTime>defaultMaximum):
        greenTime = defaultMaximum
    signals[(currentGreen+1)%(noOfSignals)].green = greenTime
```

- It defines a function called setTime(), which is responsible for calculating and setting the green time for a specific traffic signal.
- The function uses several global variables, including noOfCars, noOfBikes, noOfBuses, noOfTrucks, noOfRickshaws, and noOfLanes, to store and update the counts of different types of vehicles and lanes.
- It uses a loop to iterate over the vehicles in a specific direction (specified by directionNumbers[nextGreen]) and counts the number of vehicles of each type (cars, buses, trucks, rickshaws, and bikes).
- After calculating the number of vehicles in each category, it uses a formula to calculate the greenTime for the next signal cycle. The formula takes into account the number of vehicles and their respective time requirements (carTime, rickshawTime, busTime, truckTime, and bikeTime), and divides the total by the number of lanes plus one.
- The calculated greenTime is then checked against minimum (defaultMinimum) and maximum (defaultMaximum) limits. If it falls below the minimum, it is set to the minimum value. If it exceeds the maximum, it is set to the maximum value.
- Finally, the greenTime value is assigned to the green attribute of the next traffic signal in the sequence (signals[(currentGreen+1)%(noOfSignals)]), indicating how long the signal should stay green for that particular direction.

generateVehicles() function

The generateVehicles() function is used to generate the vehicles. The type of vehicle (car, bus, truck, or bike), the lane number (1 or 2) as well as the direction the vehicle moves towards is decided by using random numbers. The variable dist represents the cumulative distribution of vehicles in percentage. So a distribution of [25,50,75,100] means that there is an equal distribution (25% each) of vehicles across all 4 directions. Some other distributions can be [20,50,70,100], [10,20,30,100], and so on. A new vehicle is added to the simulation after every 1 second.

```
def generateVehicles():
    i=0
    with open('counter.txt', 'r') as file:
        lines = file.readlines()
    if len(lines) == 4:
        countt=int(lines[0])
        countd=int(lines[1])
        countr=int(lines[2])
        countl=int(lines[3])
    for i in range(100):
        vehicle_type = random.randint(0,4)
        if(vehicle_type==4):
            lane_number = 0
        else:
            lane_number = random.randint(0,1) + 1
        will_turn = 0
        if(lane_number==2):
            temp = random.randint(0,4)
            if(temp<=2):
                will_turn = 1
            elif(temp>2):
                will_turn = 0
        if(i<countl):
            direction_number = 0 #left
            Vehicle(lane_number, vehicleTypes[vehicle_type], direction_number, directionNumbers[direction_number], will_turn)
        if(i<countt):
            direction_number = 1 #top
            Vehicle(lane_number, vehicleTypes[vehicle_type], direction_number, directionNumbers[direction_number], will_turn)
        if(i<countr):
            direction_number = 2 #right
            Vehicle(lane_number, vehicleTypes[vehicle_type], direction_number, directionNumbers[direction_number], will_turn)
        if(i<countd):
            direction_number = 3 #down
            Vehicle(lane_number, vehicleTypes[vehicle_type], direction_number, directionNumbers[direction_number], will_turn)
        time.sleep(3)
```

Main class.

A separate thread is created for the initialize() method, which instantiates the 4 TrafficSignal objects. Then 2 colors are defined , white and black, that we will be using in our display. Next, we define the screen width and screen size, as well as the background and caption to be displayed in the simulation window. We then load the images of the 3 signals, i.e. red, yellow, and green. Now we create another thread for generateVehicles().

Next, we run an infinite loop that updates our simulation screen continuously. Within the loop, we first define the exit criteria. In the next section, we render the appropriate signal image and signal timer for each of the 4 traffic signals. Finally, we render the vehicles on the screen and call the function move() on each vehicle. This function causes the vehicles to move in the next update.

The blit() function is used to render the objects on the screen. It takes 2 arguments, the image to render and the coordinates. The coordinates point to the top-left of the image.

```
class Main:
    thread4 = threading.Thread(name="simulationTime",target=simulationTime, args=())
    thread4.daemon = True
    thread4.start()

    thread2 = threading.Thread(name="initialization",target=initialize, args=())    # initialization
    thread2.daemon = True
    thread2.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/mod_int.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)

    thread3 = threading.Thread(name="generateVehicles",target=generateVehicles, args=())    # Generating vehicles
    thread3.daemon = True
    thread3.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
```

```
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):
            if(signals[i].yellow==0):
                signals[i].signalText = "STOP"
            else:
                signals[i].signalText = signals[i].yellow
            screen.blit(yellowSignal, signalCoods[i])
        else:
            if(signals[i].green==0):
                signals[i].signalText = "SLOW"
            else:
                signals[i].signalText = signals[i].green
            screen.blit(greenSignal, signalCoods[i])
    else:
        if(signals[i].red<=10):
            if(signals[i].red==0):
                signals[i].signalText = "GO"
            else:
                signals[i].signalText = signals[i].red
        else:
            signals[i].signalText = "---"
        screen.blit(redSignal, signalCoods[i])
signalTexts = ["","","",""]

# display signal timer and vehicle count
for i in range(0,noOfSignals):
    signalTexts[i] = font.render(str(signals[i].signalText), True, white, black)
    screen.blit(signalTexts[i],signalTimerCoods[i])
    displayText = vehicles[directionNumbers[i]]['crossed']
    vehicleCountTexts[i] = font.render(str(displayText), True, black, white)
    screen.blit(vehicleCountTexts[i],vehicleCountCoods[i])

timeElapsedText = font.render(("Time Elapsed: "+str(timeElapsed)), True, black, white)
screen.blit(timeElapsedText,(1100,50))

# display the vehicles
for vehicle in simulation:
    screen.blit(vehicle.currentImage, [vehicle.x, vehicle.y])
    # vehicle.render(screen)
    vehicle.move()
```

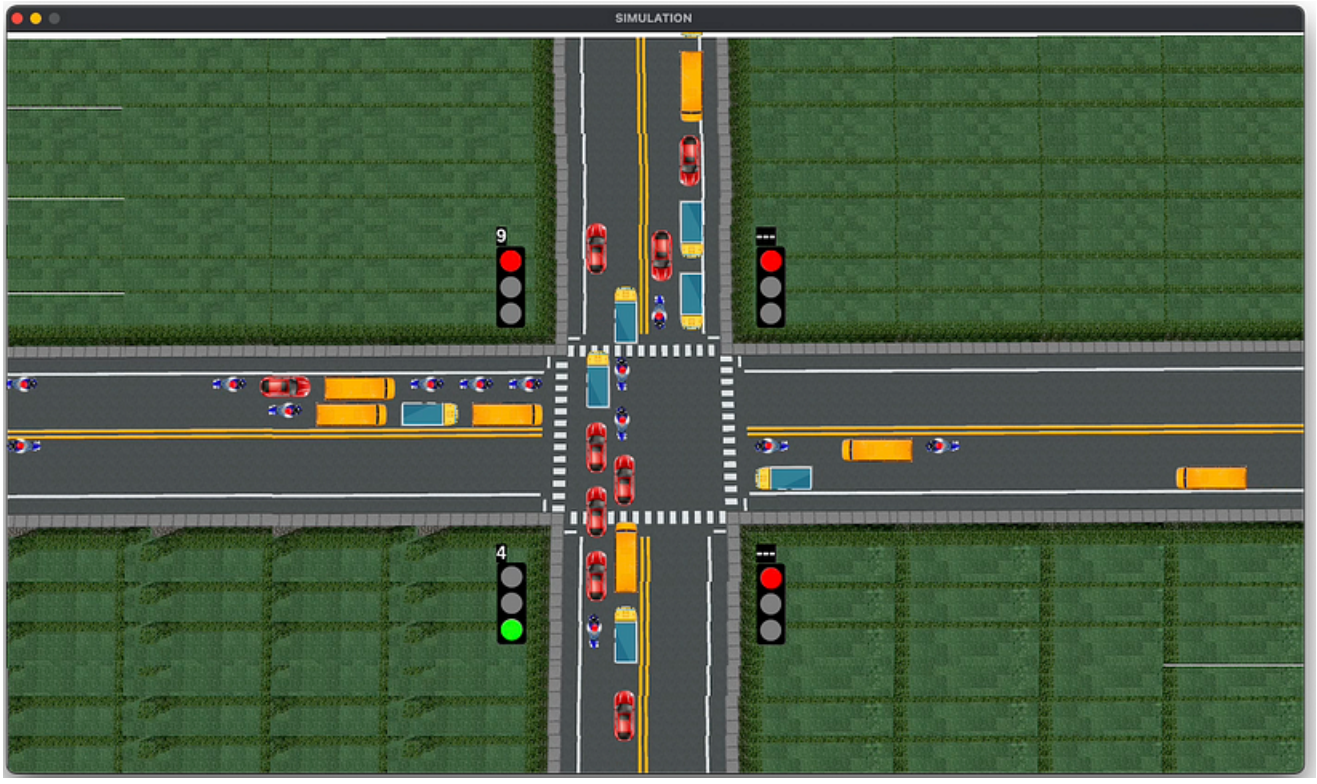
```
        for vehicle in simulation:
            screen.blit(vehicle.currentImage, [vehicle.x, vehicle.y])
            # vehicle.render(screen)
            vehicle.move()
        pygame.display.update()

Main()
```

Running the code

The code is ran in the terminal with the following command

```
$ python simulation.py
```

Snapshot of final simulation output



CHAPTER 7

TESTING

Testing is the process to find any deviation from the expected working of the system. If there is no deviation from the expected behavior of the system then the project is successful otherwise failure. Testing cannot be done in a full-fledged manner because of the time and budget constraints.

7.1 Testing Strategies

7.1.1 Unit Testing

In unit testing, individual units are determined if they are fit for use. A unit is the smallest testable part of an application. In procedural programming, a unit may be an individual function or procedure. Unit tests are created by programmers or occasionally by white box testers.

7.1.2 Integration Testing

Integration testing is the activity of finding faults when testing the individual components together. Structural testing is the culmination of integration testing involving all the components of the system. This testing strategy ensures that software and subsystems work together as a whole. It tests the interface of all modules to make sure that the modules behave properly when integrated together.

7.1.3 Functional Testing

Functional testing is a type of testing that seeks to establish whether each application feature works as per the software requirements. Each function is compared to the corresponding requirement to ascertain whether its output is consistent with the end user's expectations. Functional testing is a quality assurance (QA) process and a type of black-box testing that bases its test cases on the specifications of the software component under test.

7.1.4 Load Testing

Load testing generally refers to the practice of modeling the expected usage of a software program by simulating multiple users accessing the program concurrently.It is the process of putting demand on a system and measuring its response. Load testing examines how the system behaves during normal and high loads and determines if a system, piece of software, or computing device can handle high loads given a high demand of end-users. This tool is typically applied when a software development project nears completion.

7.2 Sample Test Cases

Input videos from four lanes

No.	Test Case	Expected Result	Test Result
1	Input videos from 4 lanes	the program will display vehicle count for each lane	Successful

Checking reliability of software for varying vehicle density

No.	Test Case	Expected Result	Test Result
1	Input video containing large number of vehicles is given	Checks if the system holds	Successful
2	Input video containing small number of vehicles is given	Checks if the system holds	Successful

Comparison between normal traffic system and our system

No.	Test Case	Expected Result	Test Result
1	Same video is given as input for both system	Our congestion system is expected to be more efficient	For the same number of vehicles our system did it 40 second quicker than the conventional system

CHAPTER 8

RESULT

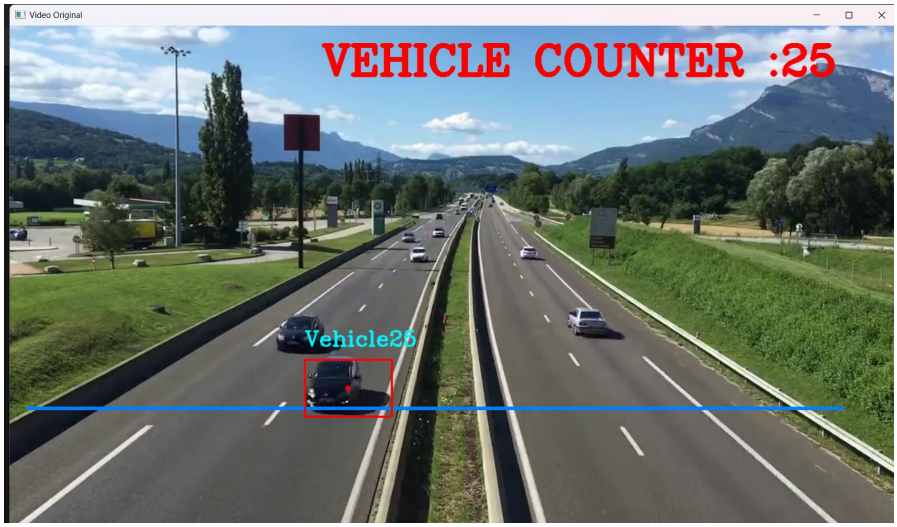


Figure 8.1

Figure 8.1 shows the vehicle detection window that counts the number of vehicles from a given video file.

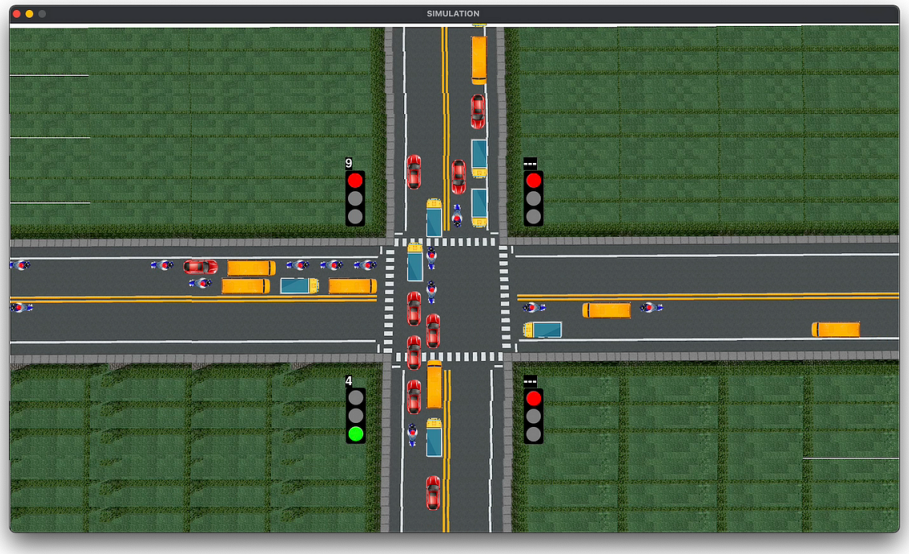


Figure 8.2

Figure 8.2 shows the simulation of our proposed model

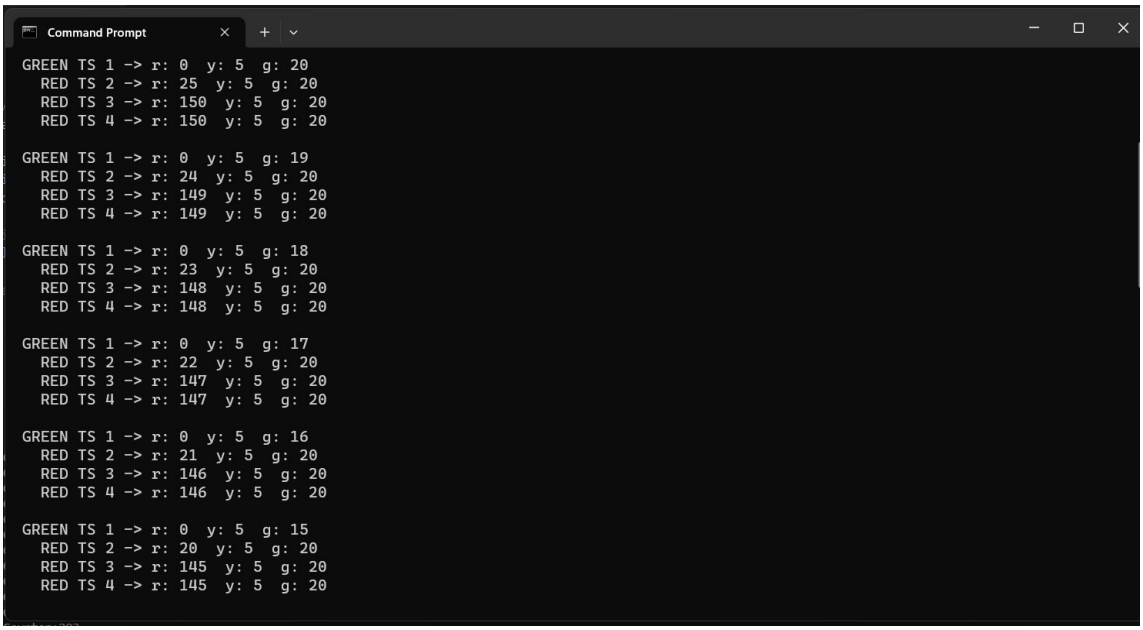


Figure 8.3 shows some images of the output of the Signal Switching Algorithm

CHAPTER 9

CONCLUSION

A smart traffic management system is a promising solution to the increasing traffic congestion and accidents in our cities. The system's integration of sensors, artificial intelligence, and real-time data analytics enables a more efficient and effective management of traffic flow, resulting in reduced travel time and improved traffic safety. However, the success of this system depends on a collaborative effort by government agencies, city planners, transportation providers, and technology companies. As more cities adopt and invest in smart traffic management systems, we can hope for a better and sustainable future for urban transportation

REFERENCES

1. Rajeshwari Sundar, Santhoshs Hebbar, and Varaprasad Golla “Implementing Intelligent Traffic Control System” Sensor Journal, IEEE(Volume:15, Issue :2)
2. A. M. Miyim and M. A. Muhammed, "Smart Traffic Management System," 2019 15th International Conference on Electronics, Computer and Computation (ICECCO), Abuja, Nigeria, 2019, pp. 1-6, doi: 10.1109/ICECCO48375.2019.9043219.
3. S. Balu and C. Priyadarshini., "Smart Traffic Congestion Control System," 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2019, pp. 689-692, doi: 10.1109/ICCMC.2019.8819759.
4. T. Xu and H. Zhang, "The optimization and simulation of traffic signal control system," 2014 International Conference on Information Science, Electronics and Electrical Engineering, Sapporo, Japan, 2014, pp. 554-556, doi: 10.1109/InfoSEEE.2014.6948174