AY 2023-24

III Semester

CSL306

**A PROJECT REPORT
ON**

SMART TRAFFIC : Traffic Management System

Bachelor of Technology

in

School of Computing

By

DEEPAK SINGH (22125)

SCHOOL OF COMPUTING

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY UNA
HIMACHAL PRADESH**

DECEMBER 2023

BONAFIDE CERTIFICATE

This is to certify that the project titled "*SMART TRAFFIC : Traffic Management System*" is a bonafide record of the work done by

DEEPAK SINGH (22125)

in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in COMPUTER SCIENCE AND ENGINEERING of the INDIAN INSTITUTE OF INFORMATION TECHNOLOGY UNA, HIMACHAL PRADESH, during the year 2023 - 2024 .

under the guidance of

DR. PRINCE SHARMA

Project viva-voce held on: _________________

Internal Examiner External Examiner

ORIGINALITY / NO PLAGIARISM DECLARATION

We certify that this project report is our original report and no part of it is copied from any published reports, papers, books, articles, etc. We certify that all the contents in this report are based on our personal findings and research and we have cited all the relevant sources which have been required in the preparation of this project report, whether they be books, articles, reports, lecture notes, and any other kind of document. We also certify that this report has not previously been submitted partially or as whole for the award of degree in any other university in India and/or abroad.

We hereby declare that, we are fully aware of what constitutes plagiarism and understand that if it is found at a later stage to contain any instance of plagiarism, our degrees may be cancelled.

DEEPAK SINGH (22125)

ABSTRACT

The Smart Traffic Control System is a modern solution designed to enhance traffic management efficiency at road intersections. Leveraging computer vision and artificial intelligence, the system focuses on real-time detection and analysis of the number of vehicles on each side of a road intersection. By employing advanced image processing techniques, the system accurately identifies and tracks vehicles, providing insights into the traffic flow dynamics for each direction. The key objective is to dynamically adjust signal timings based on the observed vehicular density, ensuring optimal traffic control and minimizing congestion.

The system integrates sensors and cameras strategically placed at the intersection to capture and analyze the movement of vehicles. Utilizing machine learning algorithms, the system distinguishes between various types of vehicles and counts their numbers on each road segment. The collected data is then used to dynamically optimize signal timings, thereby adapting to the changing traffic conditions.

The implementation includes intelligent signal control algorithms that consider not only the vehicle count but also the type of vehicles, allowing for prioritized signal adjustments. As a result, the Smart Traffic Control System contributes to a more responsive and adaptive traffic management infrastructure. This project aims to improve overall traffic efficiency, reduce congestion, and enhance the safety and convenience of road users by providing a smart and dynamic traffic signal control mechanism.

Keywords**:** Computer Vision, Artificial Intelligence, Real-time Vehicle Detection, Image Processing, Congestion Reduction, Road Safety,Intersection.

ACKNOWLEDGEMENT

We would like to thank the following people for their support and guidance without whom the completion of this project in fruition would not be possible.

We would like to express our sincere gratitude and heartfelt thanks to Dr. Prince Sharma for their unflinching support and guidance, valuable suggestions and expert advice. Their words of wisdom and expertise in subject matter were of immense help throughout the duration of this project.

We also take the opportunity to thank our Director and all the faculty of School of Computing, IIIT Una for helping us by providing necessary knowledge base and resources.

We would also like to thank our parents and friends for their constant support.

DEEPAK SINGH (22125)

TABLE OF CONTENTS

LIST OF ACRONYMS

Acronyms	Full Name	Description
YOLO	You Only Look Once.	A real-time object detection algorithm that enables simultaneous detection and classification of objects.
IoU	Intersection over Union.	A metric used to evaluate the accuracy of object detection algorithms by measuring the overlap between predicted and ground truth bounding boxes
SORT	Simple, Online and Realtime Tracker.	A tracking algorithm designed for real-time object tracking in video streams.
GPU	Graphics Processing Unit.	A specialized electronic circuit that accelerates the processing of graphics and parallel computations.
CUDA	Compute Unified Device Architecture	A parallel computing platform and programming model developed by NVIDIA, allowing developers to use NVIDIA GPUs for general-purpose processing.

LIST OF FIGURES

Chapter 1
Introduction

1.1 Introduction

As urban populations continue to grow, effective traffic management becomes paramount to ensure smooth vehicular flow and reduce commuting time. Traditional traffic signal systems, while effective in some scenarios, often rely on fixed timings that do not adapt well to varying traffic conditions. This research addresses this limitation by integrating advanced technologies such as computer vision and artificial intelligence into traffic control systems.

The core objective of the proposed Smart Traffic Control System is to create an adaptive and intelligent intersection management system that dynamically adjusts signal timings based on real-time vehicular density. By deploying computer vision algorithms, the system can accurately detect and count vehicles approaching each side of an intersection. This information is then processed using machine learning models to make informed decisions about signal duration, optimizing traffic flow and minimizing congestion

1.2 Motivation

Urbanization Challenges: Rapid urbanization has led to increased vehicular density, resulting in congestion at major intersections. The conventional traffic signal systems, designed for static scenarios, are ill-equipped to handle the fluid and unpredictable nature of modern urban traffic.

Environmental Impact: Congested traffic not only impedes daily commuting but also contributes to elevated fuel consumption and harmful emissions. Addressing traffic congestion aligns with environmental sustainability goals by promoting fuel efficiency and reducing the carbon footprint associated with vehicular idling.

Safety Concerns: Traffic bottlenecks and poorly managed intersections pose safety hazards, increasing the risk of accidents. The motivation for this research includes a commitment to enhancing road safety by implementing an adaptive traffic control system that minimizes the likelihood of collisions.

Technological Advancements: Recent advancements in computer vision and artificial intelligence provide unprecedented opportunities to create intelligent systems capable of real-time data analysis. The motivation is to harness these technologies to transform traditional traffic management into a responsive and data-driven process.

Improved Commuting Experience: The ultimate motivation is to enhance the overall commuting experience for residents and visitors alike. By reducing wait times, minimizing congestion, and optimizing traffic flow, the Smart Traffic Control System aims to make daily commuting more predictable, efficient, and enjoyable.

1.3 Objectives

- Develop and implement advanced computer vision algorithms to accurately detect and classify vehicles approaching an intersection.
- Utilize image processing techniques to extract relevant information, such as vehicle type and count, from live camera feeds.
- Establish a robust system for collecting and analyzing dynamic traffic data, including the number of vehicles per lane.
- Prioritize traffic signals to optimize the overall traffic flow and reduce congestion at critical intersections.

1.4 Significance

The system's ability to dynamically adjust traffic signals based on real-time conditions leads to optimized traffic flow, reducing congestion and travel time for commuters. This results in a more efficient and streamlined urban transportation network.The system facilitates data-driven decision-making processes by providing traffic management authorities with real-time insights into traffic patterns, congestion hotspots, and overall system performance. This empowers city planners to make informed decisions for further infrastructure development and improvements.

Chapter 2
Review of Literature

2.1 YOLO - You Only Look Once.

The "You Only Look Once" (YOLO) [3] algorithm, introduced by Joseph Redmon and Santosh Divvala in 2016, represents a significant milestone in the field of computer vision, particularly in object detection. YOLO revolutionized the traditional approach to object detection by proposing a unified, end-to-end model that achieves remarkable speed and accuracy. This literature review delves into the key contributions of the YOLO research paper, highlighting its impact on object detection methodologies.

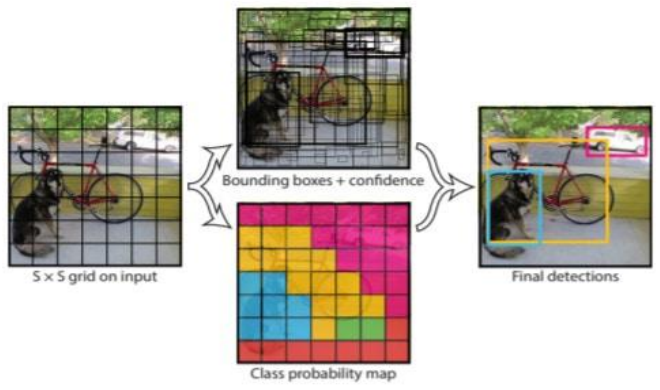

Image 1: YOLO Working

YOLO divides the input image into a grid and assigns bounding boxes and class probabilities to each grid cell. This grid-based prediction strategy allows the model to capture contextual information efficiently. The YOLO algorithm's architecture, consisting of 24 convolutional layers, further enhances its ability to discern complex spatial hierarchies within an image.

2.2 Effective Vehicle Tracking Algorithm for Smart Traffic Networks [4]

The paper begins with a comprehensive review of existing vehicle tracking algorithms. It highlights the strengths and weaknesses of various approaches, including.

Deep Sort: Employs a deep learning network for feature extraction, achieving high accuracy but requiring significant resources.

IoU Tracker: Simple and efficient, but prone to errors when dealing with occlusions and closely spaced vehicles.

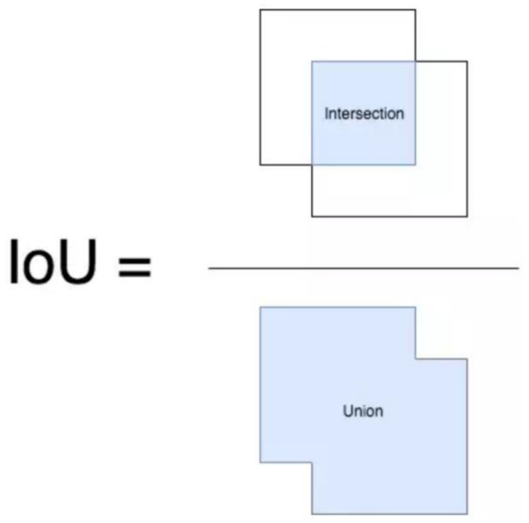

Image 2: IoU

ID Tracking Inspiration:

The paper explicitly acknowledges the influence of David C. Anastasiu's work on ID tracking. Specifically, the authors reference Anastasiu's research on efficient data association techniques for multi-object tracking. These techniques are incorporated into the proposed algorithm and contribute to its improved accuracy and performance.

Chapter 3
Model & Framework

3.1 Object Detection Model

YOLOv8 Architecture

The Smart Traffic Management System employs the YOLOv8 (You Only Look Once version 8) architecture for vehicle detection. YOLOv8 is a object detection model known for its real-time performance and accuracy. It operates on the principle of dividing the input image into a grid and predicting bounding boxes and class probabilities directly. This architecture is chosen for its ability to detect multiple objects in a single pass, making it well-suited for real-time applications such as traffic management.

Model Configuration

The YOLOv8 model utilized in this project consists of various configuration files, including network architecture, hyperparameters, and anchor box settings. The model is trained on a labeled dataset specific to traffic scenes to ensure optimal performance in the context of the Smart Traffic Management System.

Training Procedure

The model training involves optimizing the weights based on a combination of labeled images and ground truth bounding boxes. The dataset includes diverse traffic scenarios to enhance the model's ability to generalize across different environmental conditions. The training process is fine-tuned to achieve high precision and recall rates, essential for accurate vehicle detection.

Transfer Learning

To enhance the model's performance and accelerate convergence, transfer learning is employed. A pre-trained YOLOv8 model on a large-scale dataset, such as COCO (Common Objects in Context), serves as the starting point. This allows the model to leverage knowledge gained from generic object detection tasks and adapt to the specific requirements of traffic management.

3.2 Deep Learning Framework

PyTorch

The implementation of the YOLOv8 model for vehicle detection is done using the PyTorch deep learning framework. PyTorch provides a flexible and intuitive platform for building and training neural networks. The framework facilitates efficient GPU utilization, enabling real-time inference, a critical factor for the success of a Smart Traffic Management System.

Advantages of PyTorch

- **Dynamic Computational Graphs:** PyTorch's dynamic computational graph allows for on-the-fly graph creation and modification, enhancing flexibility during model development.
- **Extensive Community Support:** The active PyTorch community provides a wealth of resources, tutorials, and pre-trained models, streamlining the development process.
- **GPU Acceleration:** PyTorch seamlessly integrates with CUDA, enabling efficient GPU acceleration for faster model training and inference.

Chapter 4

Result & Analysis

4.1 Performance Results

Overall Object Detection Metrics

The YOLOv8 model achieves competitive performance on the testing set, as indicated by precision, recall, and F1 Score. The IoU scores demonstrate the accuracy of bounding box predictions.

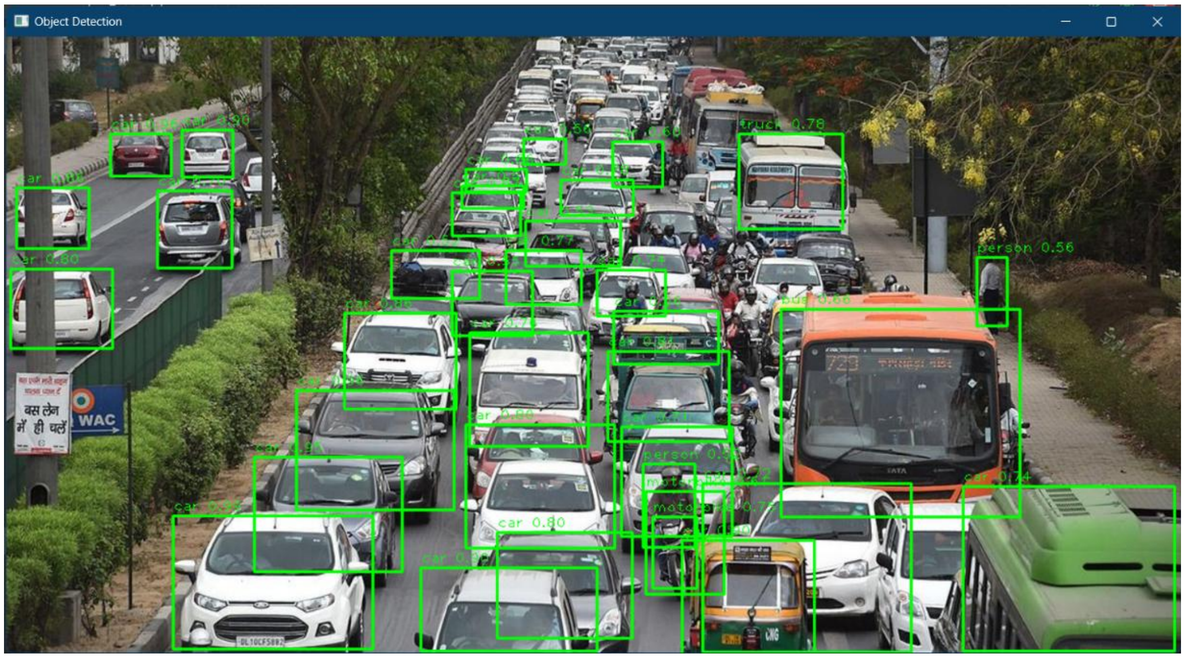

Image 3: Implementation on still frames

Class-wise Performance

Evaluate the model's performance on different vehicle classes (e.g., cars, trucks, motorcycles). Analyze the precision, recall, and F1 Score for each class to identify potential areas for improvement.

Realtime Tracking

Real-time tracking with YOLO revolutionizes object detection by providing swift and accurate tracking capabilities. With its single-pass detection approach, YOLO excels in processing video frames at high speeds, enabling real-time tracking of multiple objects. Its adaptability to diverse environments, efficient object association across frames.

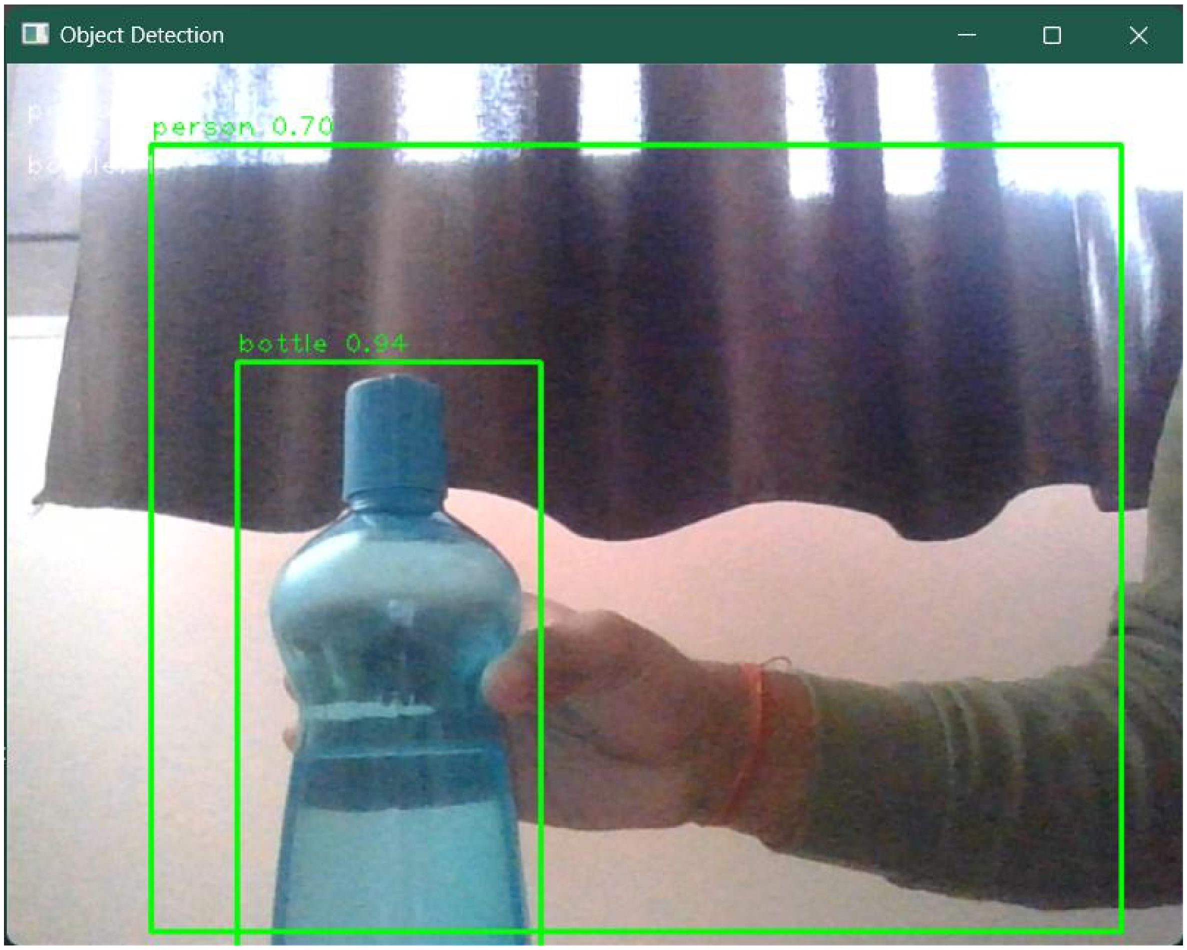

Image 4: Realtime Tracking using webcam

Qualitative Results

Include visualizations of the model's predictions on sample images from the testing set. Highlight successful detections and instances where the model faces challenges, such as occlusions or complex traffic scenarios.

4.2 Analysis

Strengths

Identify the strengths of the YOLOv8 model in the context of the Smart Traffic Management System. Discuss scenarios where the model performs exceptionally well.

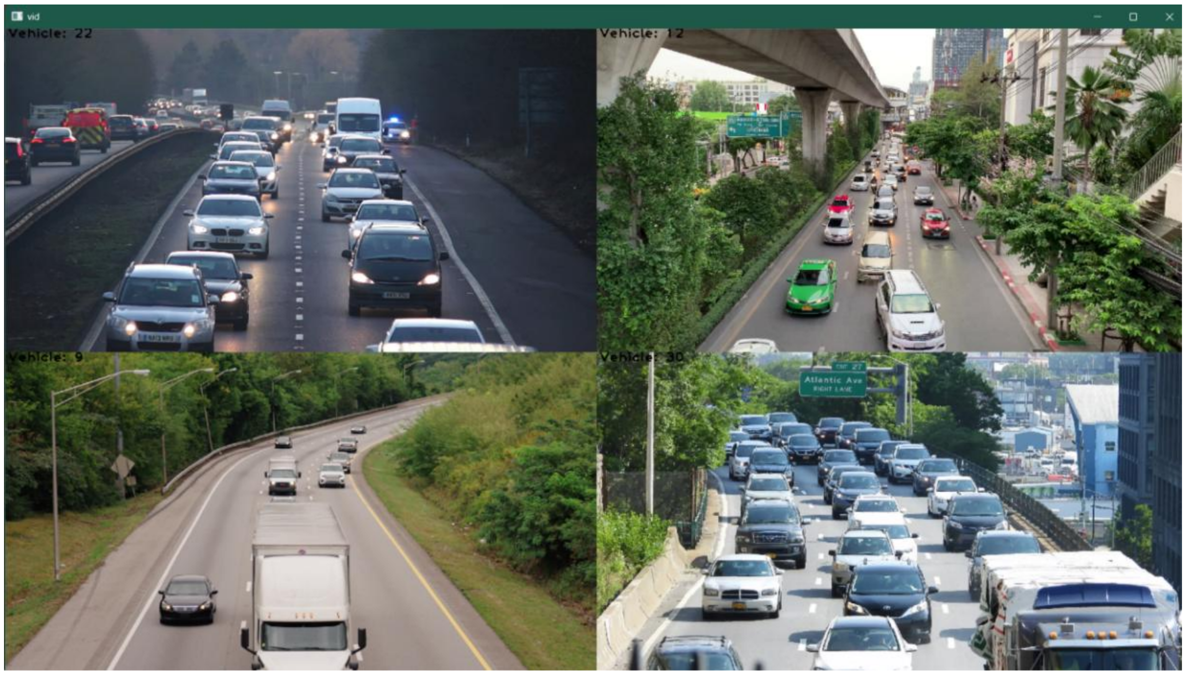

Image 5: Detection and Counting for 4 videos Simultaneously

Limitations and Challenges

Acknowledge any limitations or challenges faced by the model. This could include instances of false positives, false negatives, or scenarios where the model struggles to generalize.

Chapter 5
Conclusion & Future Scope

5.1 Conclusion

The implementation of a Smart Traffic Management System leveraging the YOLOv8 object detection model has demonstrated promising results in efficient and real-time vehicle detection. Through the evaluation metrics and analysis, we have observed the strengths and limitations of the chosen approach in the context of traffic monitoring.

Key Findings:

Real-time Performance: YOLOv8 showcases impressive real-time capabilities, making it suitable for dynamic traffic environments.

Accuracy: The model demonstrates competitive accuracy in detecting various vehicle classes across diverse scenarios.

Consistency: The IoU-based consistency checks provide valuable insights into the tracking reliability, contributing to the system's overall robustness.

5.2 Future Scope

While the current implementation lays a solid foundation for the Smart Traffic Management System, there are several avenues for future improvement and expansion.

Dataset Enrichment:

Augment the dataset with additional annotated samples, especially focusing on challenging scenarios such as adverse weather conditions, low-light environments, and crowded traffic situations. This will improve the model's ability to generalize across a broader range of conditions.

Integration of Advanced Features:

Explore the integration of additional features, such as vehicle speed estimation, lane departure warnings, and anomaly detection. These features can provide a more comprehensive understanding of traffic dynamics and enable the system to respond proactively to potential issues.

Deployment:

Investigate the deployment of the Smart Traffic Management System on edge devices for decentralized processing. This can lead to reduced latency and increased scalability, enabling deployment in a wider range of locations.

References

[1] W. McKinney, *Python for Data Analysis*, 3rd ed. O'Reilly Media, 2017.

[2] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*,2nd ed. O'Reilly Media, 2019.

[3] J. Redmon and S. Divvala, "You Only Look Once: Unified, Real-Time Object Detection," *Computer Vision* ,2016.

[4] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference* on computer vision. Springer, 2014, pp. 740–755.

[5] S. Hua and D. C. Anastasiu, "Effective Vehicle Tracking Algorithm for Smart Traffic Networks," in *International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, 2019.

[6] M. M. Gandhi, D. S. Solanki, R. S. Daptardar, and N. S. Baloorkar, "Smart Control of Traffic Light Using Artificial Intelligence," *5th International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, IEEE, 2020.

[7] Bewley, Alex and Ge, Zongyuan and Ott, Lionel and Ramos, Fabio and Upcroft, Ben, "Simple online and realtime tracking," *International Conference on Image Processing (ICIP),* IEEE, 2016, pp. 3464-3468.

Appendices

Appendix A
Code Attachments

The following is the subset of the code. Code of some module(s) have been willfully suppressed.

A.1 Vehicle Detection and Counting Code

```
import math
import Simulation
import cv2
import numpy as np
from ultralytics import YOLO
from sort import *


with open("Data/coco.names", "r") as f:
    classes = [line.strip() for line in f]
    classes = classes[:13]

total_Vehicles = {0:0,1:0,2:0,3:0}

model = YOLO("Data/yolov8n.pt")

vid1 = cv2.VideoCapture("videos/vid1.mp4")
vid2 = cv2.VideoCapture("videos/vid2.mp4")
vid3 = cv2.VideoCapture("videos/vid3.mp4")
vid4 = cv2.VideoCapture("videos/vid4.mp4")

while True:
    isTrue1, frame1 = vid1.read()
    isTrue2, frame2 = vid2.read()
    isTrue3, frame3 = vid3.read()
    isTrue4, frame4 = vid4.read()
```

```
if not (isTrue1 and isTrue2 and isTrue3 and isTrue4):
    break

frame2 = cv2.resize(frame2, (frame1.shape[1] // 2, frame1.shape[0] //
   2))
frame3 = cv2.resize(frame3, (frame1.shape[1] // 2, frame1.shape[0] //
   2))
frame4 = cv2.resize(frame4, (frame1.shape[1] // 2, frame1.shape[0] //
   2))
frame1 = cv2.resize(frame1, (frame1.shape[1] // 2, frame1.shape[0] //
   2))
font = cv2.FONT_HERSHEY_PLAIN

for frame, vid_num in zip([frame1, frame2, frame3, frame4], [1, 2, 3,
   4]):
    object_counts = {}
    results = model(frame, stream=True)

    detection = np.empty((0, 5))

    for r in results:
        boxes = r.boxes
        for box in boxes:
            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)

            conf = math.ceil((box.conf[0] * 100)) / 100

            cls = int(box.cls[0])
            currentClass = classes[cls]

            if ((int(box.cls[0]) > 1) & (int(box.cls[0]) <= 7)) :
                if conf > 0.3:
                    label = "Vehicle"
                    currentArray = np.array([x1, y1, x2, y2, conf])
                    detection = np.vstack((detection, currentArray))
            else:
                label = str(classes[int(box.cls[0])])
                if label not in object_counts:
                    object_counts[label] = 1
                else:
                    object_counts[label] += 1

        vehicles_new = object_counts["Vehicle"]


        if total_Vehicles[vid_num-1] < vehicles_new:
            total_Vehicles[vid_num-1]+=(vehicles_newtotal_Vehicles[vid_num-
                                1])



        for label, count in object_counts.items():
            cv2.putText(frame,    f"{label}:    {count}",    (10,    30    *
(list(object_counts.keys()).index(label) + 1)),
                        font, 3, (0, 0, 0), 3)

        print(f"Video{vid_num}")

    frame_up = np.concatenate((frame1, frame2), axis=1)
    frame_down = np.concatenate((frame3, frame4), axis=1)
    frame_final = np.vstack((frame_up, frame_down))

    cv2.namedWindow("vid", cv2.WINDOW_NORMAL)
    cv2.imshow("vid", frame_final)

    if cv2.waitKey(1) == ord('q'):
        vid1.release()
        vid2.release()
        vid3.release()
        vid4.release()
        break


total_Vehicles1 = total_Vehicles[0]
```

```
total_Vehicles2 = total_Vehicles[1]
total_Vehicles3 = total_Vehicles[2]
total_Vehicles4 = total_Vehicles[3]

print(f"lane1:{total_Vehicles1},lane2:{total_Vehicles2},/
      lane3:{total_Vehicles3}, lane4:{total_Vehicles4}")

Simulation.maxCounts = total_Vehicles
Simulation.Main()
```

A.2 Traffic Simulation Code

```
import random

import math

import time

import threading

import pygame

import sys

import os


maxCounts = {0:10,1:11,2:4,3:5}

maxVehicles = sum(maxCounts.values())


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}  # average speeds of vehicles


# 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


pygame.init()

simulation = pygame.sprite.Group()
```

```
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)



        if (direction == 'right'):

            if      (len(vehicles[direction][lane])      >      1      and
vehicles[direction][lane][

                self.index - 1].crossed == 0):  # if more than 1 vehicle in
the lane of vehicle before it has crossed stop line

                self.stop = vehicles[direction][lane][self.index - 1].stop -
vehicles[direction][lane][

                    self.index  -  1].currentImage.get_rect().width  -  gap   #
setting stop coordinate as: stop coordinate of next vehicle - width of
next vehicle - gap

            else:

                self.stop = defaultStop[direction]

            # Set new starting and stopping coordinate

            temp = self.currentImage.get_rect().width + gap

            x[direction][lane] -= temp

            stops[direction][lane] -= temp

        elif (direction == 'left'):

            if      (len(vehicles[direction][lane])      >      1      and
vehicles[direction][lane][self.index - 1].crossed == 0):

                self.stop = vehicles[direction][lane][self.index - 1].stop +
vehicles[direction][lane][

                    self.index - 1].currentImage.get_rect().width + gap

            else:

                self.stop = defaultStop[direction]

            temp = self.currentImage.get_rect().width + gap
            x[direction][lane] += temp

            stops[direction][lane] += temp

        elif (direction == 'down'):

            if      (len(vehicles[direction][lane])      >      1      and
vehicles[direction][lane][self.index - 1].crossed == 0):

                self.stop = vehicles[direction][lane][self.index - 1].stop -
vehicles[direction][lane][

                    self.index - 1].currentImage.get_rect().height - gap

            else:

                self.stop = defaultStop[direction]

            temp = self.currentImage.get_rect().height + gap

            y[direction][lane] -= temp

            stops[direction][lane] -= temp

        elif (direction == 'up'):

            if      (len(vehicles[direction][lane])      >      1      and
vehicles[direction][lane][self.index - 1].crossed == 0):

                self.stop = vehicles[direction][lane][self.index - 1].stop +
vehicles[direction][lane][

                    self.index - 1].currentImage.get_rect().height + gap

            else:

                self.stop = defaultStop[direction]

            temp = self.currentImage.get_rect().height + gap

            y[direction][lane] += temp

            stops[direction][lane] += temp

        simulation.add(self)


    def render(self, screen):

        screen.blit(self.currentImage, (self.x, self.y))


    def move(self):
        global maxVehicles

        if (self.direction == 'right'):

            if      (self.crossed      ==      0      and      self.x      +
self.currentImage.get_rect().width > stopLines[

                self.direction]):  # if the image has crossed stop line now

                self.crossed = 1

                vehicles[self.direction]['crossed'] += 1

                maxVehicles -= 1

            if (self.willTurn == 1):

                if      (self.crossed      ==      0      or      self.x      +
self.currentImage.get_rect().width < mid[self.direction]['x']):

                    if   ((self.x   +   self.currentImage.get_rect().width   <=
self.stop or (

                            currentGreen  ==  0  and  currentYellow  ==  0)  or
self.crossed == 1) and (

                            self.index      ==      0      or      self.x      +
self.currentImage.get_rect().width < (

                            vehicles[self.direction][self.lane][self.index   -
1].x - gap2) or

                            vehicles[self.direction][self.lane][self.index   -
1].turned == 1)):

                        self.x += self.speed

                else:

                    if (self.turned == 0):

                        self.rotateAngle += rotationAngle

                        self.currentImage                                     =
pygame.transform.rotate(self.originalImage, -self.rotateAngle)

                        self.x += 2

                        self.y += 1.8

                        if (self.rotateAngle == 90):

                            self.turned = 1
                        #        path        =        "images/"        +
directionNumbers[((self.direction_number+1)%noOfSignals)]    +    "/"    +
self.vehicleClass + ".png"

                        # self.x = mid[self.direction]['x']

                        # self.y = mid[self.direction]['y']

                        # self.image = pygame.image.load(path)

                 else:

                     if     (self.index     ==     0     or     self.y     +
self.currentImage.get_rect().height < (

                            vehicles[self.direction][self.lane][

                                self.index  -  1].y  -  gap2)  or  self.x  +
self.currentImage.get_rect().width < (

                            vehicles[self.direction][self.lane][self.index
- 1].x - gap2)):

                        self.y += self.speed

          else:

              if   ((self.x   +   self.currentImage.get_rect().width   <=
self.stop or self.crossed == 1 or (

                     currentGreen == 0 and currentYellow == 0)) and (

                     self.index      ==      0      or      self.x      +
self.currentImage.get_rect().width < (

                     vehicles[self.direction][self.lane][self.index - 1].x
- gap2) or (

                            vehicles[self.direction][self.lane][self.index
- 1].turned == 1))):

                 # (if the image has not reached its stop coordinate or
has crossed stop line or has green signal) and (it is either the first
vehicle in that lane or it is has enough gap to the next vehicle in that
lane)

                 self.x += self.speed  # move the vehicle




       elif (self.direction == 'down'):
        if      (self.crossed      ==      0      and      self.y      +
self.currentImage.get_rect().height > stopLines[self.direction]):

            self.crossed = 1

            vehicles[self.direction]['crossed'] += 1

            maxVehicles -= 1

        if (self.willTurn == 1):

            if      (self.crossed      ==      0      or      self.y      +
self.currentImage.get_rect().height < mid[self.direction]['y']):

                if   ((self.y   +   self.currentImage.get_rect().height   <=
self.stop or (

                        currentGreen  ==  1  and  currentYellow  ==  0)  or
self.crossed == 1) and (

                        self.index      ==      0      or      self.y      +
self.currentImage.get_rect().height < (

                        vehicles[self.direction][self.lane][self.index   -
1].y - gap2) or

                        vehicles[self.direction][self.lane][self.index   -
1].turned == 1)):

                    self.y += self.speed

            else:

                if (self.turned == 0):

                    self.rotateAngle += rotationAngle

                    self.currentImage                                    =
pygame.transform.rotate(self.originalImage, -self.rotateAngle)

                    self.x -= 2.5

                    self.y += 2

                    if (self.rotateAngle == 90):

                        self.turned = 1

                else:

                    if     (self.index     ==     0     or     self.x     >
(vehicles[self.direction][self.lane][self.index - 1].x +


vehicles[self.direction][self.lane][
                                                self.index          -
1].currentImage.get_rect().width + gap2) or self.y < (

                                vehicles[self.direction][self.lane][self.index
- 1].y - gap2)):

                            self.x -= self.speed

            else:

                if   ((self.y   +   self.currentImage.get_rect().height   <=
self.stop or self.crossed == 1 or (

                        currentGreen == 1 and currentYellow == 0)) and (

                        self.index      ==      0      or      self.y      +
self.currentImage.get_rect().height < (

                        vehicles[self.direction][self.lane][self.index - 1].y
- gap2) or (

                                vehicles[self.direction][self.lane][self.index
- 1].turned == 1))):

                    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

                maxVehicles -= 1

            if (self.willTurn == 1):

                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].currentImage.get_rect().width + gap2)
or vehicles[self.direction][self.lane][

                                self.index - 1].turned == 1)):

                        self.x -= self.speed
                else:

                    if (self.turned == 0):

                        self.rotateAngle += rotationAngle

                        self.currentImage                            =
pygame.transform.rotate(self.originalImage, -self.rotateAngle)

                        self.x -= 1.8

                        self.y -= 2.5

                        if (self.rotateAngle == 90):

                            self.turned = 1

                            #        path        =        "images/"        +
directionNumbers[((self.direction_number+1)%noOfSignals)]    +    "/"    +
self.vehicleClass + ".png"

                            # self.x = mid[self.direction]['x']

                            # self.y = mid[self.direction]['y']

                            # self.currentImage = pygame.image.load(path)

                    else:

                        if     (self.index     ==     0     or     self.y     >
(vehicles[self.direction][self.lane][self.index - 1].y +


vehicles[self.direction][self.lane][

                                                      self.index          -
1].currentImage.get_rect().height + gap2) or self.x > (

                                vehicles[self.direction][self.lane][self.index
- 1].x + gap2)):

                            self.y -= self.speed

            else:

                if   ((self.x   >=   self.stop   or   self.crossed   ==   1   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].currentImage.get_rect().width + gap2) or
(
                                vehicles[self.direction][self.lane][self.index
- 1].turned == 1))):

                    # (if the image has not reached its stop coordinate or
has crossed stop line or has green signal) and (it is either the first
vehicle in that lane or it is has enough gap to the next vehicle in that
lane)

                    self.x -= self.speed  # move the vehicle

            #    if((self.x>=self.stop    or    self.crossed    ==    1    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].currentImage.get_rect().width + gap2))):

            #    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

                maxVehicles -= 1

            if (self.willTurn == 1):

                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].currentImage.get_rect().height + gap2)
or vehicles[self.direction][self.lane][

                                self.index - 1].turned == 1)):

                        self.y -= self.speed

                else:

                    if (self.turned == 0):

                        self.rotateAngle += rotationAngle

                        self.currentImage                                 =
pygame.transform.rotate(self.originalImage, -self.rotateAngle)
```

```
                    self.x += 1

                    self.y -= 1

                    if (self.rotateAngle == 90):

                       self.turned = 1

                 else:

                    if     (self.index     ==     0     or     self.x     <
(vehicles[self.direction][self.lane][self.index - 1].x -


vehicles[self.direction][self.lane][

                                                             self.index          -
1].currentImage.get_rect().width - gap2) or self.y > (

                          vehicles[self.direction][self.lane][self.index
- 1].y + gap2)):

                       self.x += self.speed

           else:

              if   ((self.y   >=   self.stop   or   self.crossed   ==   1   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].currentImage.get_rect().height  +  gap2)
or (

                          vehicles[self.direction][self.lane][self.index
- 1].turned == 1))):

                 self.y -= self.speed




# Initialization of signals with default values

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()




# Set time according to formula

def setTime():

    global noOfCars, noOfBikes, noOfBuses, noOfTrucks, noOfRickshaws, noOfLanes

    global carTime, busTime, truckTime, rickshawTime, bikeTime

    noOfCars, noOfBuses, noOfTrucks, noOfRickshaws, noOfBikes = 0, 0, 0, 0, 0

    global nextGreen

    maxVehicles = 0

    nextGreenCandidate = nextGreen



    for i in range(noOfSignals):

        if i != currentGreen:

            vehiclesCount = len(vehicles[directionNumbers[i]][0]) + len(vehicles[directionNumbers[i]][1]) + \

                            len(vehicles[directionNumbers[i]][2]) - vehicles[directionNumbers[i]]['crossed']

            if vehiclesCount > maxVehicles:

                maxVehicles = vehiclesCount
                nextGreenCandidate = i

            # print(i,vehiclesCount)

            # print(maxVehicles,nextGreenCandidate)


    nextGreen = nextGreenCandidate

    print("Updated nextGreen based on max vehicles:", directionNumbers[nextGreen])


    # for i in range(noOfSignals):

    #     if i != nextGreen:

    #         signals[i].red = max(signals[i].minimum, maxVehicles * carTime)




    for j in range(len(vehicles[directionNumbers[nextGreen]][0])):

        vehicle = vehicles[directionNumbers[nextGreen]][0][j]

        if (vehicle.crossed == 0):

            vclass = vehicle.vehicleClass

            # print(vclass)

            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

                # print(vclass)

                if (vclass == 'car'):

                    noOfCars += 1

                elif (vclass == 'bus'):
```

```
                noOfBuses += 1

            elif (vclass == 'truck'):

                noOfTrucks += 1

            elif (vclass == 'rickshaw'):

                noOfRickshaws += 1

    # print(noOfCars)

    greenTime  =  math.ceil(((noOfCars  *  carTime)  +  (noOfRickshaws  *
rickshawTime) + (noOfBuses * busTime) + (

            noOfTrucks  *  truckTime)  +  (noOfBikes  *  bikeTime))  /
(noOfLanes + 1))

    # greenTime = math.ceil((noOfVehicles)/noOfLanes)

    print('Green Time: ', greenTime)

    if (greenTime < defaultMinimum):

        greenTime = defaultMinimum

    elif (greenTime > defaultMaximum):

        greenTime = defaultMaximum

    # greenTime = random.randint(15,50)

    signals[nextGreen].green = greenTime




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()



# Print the signal timers on cmd

def printStatus():

    for i in range(0, noOfSignals):

        if (i == currentGreen):

            if (currentYellow == 0):

                print(" GREEN TS", i + 1, "-> r:", signals[i].red, " y:", signals[i].yellow, " g:", signals[i].green)

            else:

                print("YELLOW TS", i + 1, "-> r:", signals[i].red, " y:", signals[i].yellow, " g:", signals[i].green)

        else:

            print("    RED TS", i + 1, "-> r:", signals[i].red, " y:", signals[i].yellow, " g:", signals[i].green)

    print()



# 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

                signals[i].totalGreenTime += 1

            else:

                signals[i].yellow -= 1

        else:
            signals[i].red -= 1



# Generating vehicles in the simulation
def generateVehicles():
    global maxCounts,vehicles,maxVehicles
    while (True):
        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
        temp = random.randint(0, 999)
        direction_number = 0
        a = [400, 800, 900, 1000]
        if (temp < a[0]):
            direction_number = 0
        elif (temp < a[1]):
            direction_number = 1
        elif (temp < a[2]):
            direction_number = 2
        elif (temp < a[3]):

            direction_number = 3



        if                  maxCounts[direction_number]                  >
len(vehicles[directionNumbers[direction_number]][0]) + \

                len(vehicles[directionNumbers[direction_number]][1])        +
len(vehicles[directionNumbers[direction_number]][2]):

            Vehicle(lane_number,                 vehicleTypes[vehicle_type],
direction_number, directionNumbers[direction_number],will_turn)





        # if(maxCount[direction_number]>0):


            # maxCount[direction_number] -= 1

        time.sleep(0.75)




def simulationTime():

    global timeElapsed, simTime,maxVehicles

    while (True):

        timeElapsed += 1

        time.sleep(1)

        # print("max",maxVehicles)

        if (timeElapsed == simTime or maxVehicles == 0):

            time.sleep(2)


            totalVehicles = 0

            print('Lane-wise Vehicle Counts')
            for i in range(noOfSignals):
                print('Lane', i + 1, ':', vehicles[directionNumbers[i]]['crossed'])
                totalVehicles += vehicles[directionNumbers[i]]['crossed']
            print('Total vehicles passed: ', totalVehicles)
            print('Total time passed: ', timeElapsed)
            print('No. of vehicles passed per unit time: ', (float(totalVehicles) / float(timeElapsed)))

            os._exit(1)


def Main():
        global maxVehicles,maxCounts
        maxVehicles = sum(maxCounts.values())

        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)


# generateVehicles(2)

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):

            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:

                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()

            pygame.display.update()


if __name__ == "__main__":

    Main()
```