GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, TIRUTTANI - 631209

Approved by AICTE, New Delhi Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TRAFFIC MANAGEMENT USING INTERNET OF THINGS
PROJECT REPORT

SUBMITTED BY

D. Suresh
110321104051
3rd Year/5th Sem
sureshsrh05@gmail.com

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report **"TRAFFIC MANAGEMENT USING INTERNET OF THINGS"** is the bonafide work of **"D SURESH [110321104051]"** who carried out the project work under my our supervision.

SIGNATURE
Dr.N. Kamal M.E.,Ph.D.,
HOD
Department of Computer Science And Engineering
GRT Institute of Engineering and Technology
Tiruttani

SIGNATURE
Mr.T.A. Vinayagam M.Tech.,
Assistant professor
Department of Computer Science And Engineering
GRT Institute of Engineering and Technology
Tiruttani

Certified that the candidates were examined in Viva-voce in the Examination

Held on_____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We thank our Management for providing us all support to complete this project successfully. Our sincere thanks to honorable **Chairman**, **Shri. G. RAJENDRAN and Managing Director, Shri.G.R. RADHAKRISHNAN** for creating a wonderful atmosphere inside the campus.

We are very grateful to **Dr.S. ARUMUGAM, M.E., Ph.D., Principal**, for providing us with consistent guidance and motivation to execute a real time project to learn and experience the project work in an environments to complete our project successfully.

Our sincere thanks to **Dr.N. KAMAL, M.E., Ph.D., Professor and Head, Department of Computer Science and Engineering** for giving me this wonderful opportunity to do the project and providing the require facilities to fulfill our work.

We are highly indebted and thankful to our project Evaluators **Mrs V. Priya M.E**., and **Mrs. Edith Esther M.E., Assistant Professor, Department of Computer Science andEngineering** for his immense support in doing the project.

We are very grateful to our internal guide **Mr. T.A. VINAYAGAM, M.Tech., AssistantProfessor, Department of Computer Science and Engineering** for guiding us with her valuable suggestions to complete our project.

We also dedicate equal and grateful acknowledgement to all the **respectable members of thefaculty and lab in-charges** of the Department of Computer Science and Engineering, friends andour families for their motivation, encouragement and continuous support.

Our sincere thanks **to IBM and Skill Up Team members** for giving me this wonderful opportunity to do the project and providing the require guidance and valuable online sessions.

TABLE OF CONTENT

CHAPTER - 1

1.0. ABSTRACTION:

- Traffic management is a critical aspect of urban planning and infrastructure development, especially in rapidly growing cities.
- Congestion, accidents, and inefficient transportation systems can lead to a host of problems, including increased pollution, longer commute times, and decreased overall quality of life.
- To address these challenges, the Internet of Things (IoT) has emerged as a powerful technology that can revolutionize traffic management by providing real-time data and intelligent decision-making capabilities.
- This document presents a detailed overview of the application of IoT in traffic management.
- It includes an introduction to IoT, a definition of the problem at hand, and outlines the objectives that must be achieved to successfully implement IoT-based traffic management systems.

1.1. INTRODUCTION:

- The Internet of Things (IoT) refers to the network of interconnected physical devices, vehicles, buildings, and other objects embedded with sensors, software, and network connectivity.
- These devices collect and exchange data, enabling them to communicate and interact with each other, as well as with centralized systems or cloud platforms.
- IoT has gained significant prominence in various industries, and traffic management is no exception.
- IoT technologies are being employed to create intelligent transportation systems (ITS) that improve traffic management, reduce congestion, enhance safety, and promote sustainable urban mobility.
- In an IoT-based traffic management system, various components such as sensors, cameras, data analytics, and communication networks collaborate to gather and process data in real time.
- This data is then used to make informed decisions, optimize traffic flow, and provide valuable information to both commuters and traffic management authorities.

1.2. PROBLEM DEFINITION:

The problem of traffic management in urban areas is multifaceted and includes several challenges:

1) Congestion: Urban congestion leads to wasted time and energy, increased pollution, and reduced economic productivity.
2) Safety: Traffic accidents are a significant cause of injuries and fatalities, necessitating improved safety measures.
3) Environmental Impact: Vehicle emissions contribute to air pollution and climate change, making it essential to promote sustainable transportation.
4) Inefficient Infrastructure: The inefficiency of road networks and transportation systems often results from a lack of real-time data and adaptive control.
5) Information Accessibility: Commuters often lack access to real-time traffic information, hindering their ability to make informed travel decisions.
6) Resource Allocation: Traffic management authorities require better tools for resource allocation, traffic control, and incident management.

1.3. OBJECTIVES:

The objectives of implementing IoT-based traffic management systems are as follows:

1) Real-time Data Collection: Deploy sensors, cameras, and other IoT devices to collect real-time traffic data, including vehicle counts, speed, congestion levels, and weather conditions.
2) Data Analysis: Utilize advanced data analytics and machine learning algorithms to process the collected data and extract meaningful insights, such as traffic patterns and congestion hotspots.
3) Traffic Prediction: Develop predictive models that can forecast traffic conditions and incidents, enabling proactive traffic management.
4) Adaptive Traffic Control: Implement adaptive traffic signal control systems that adjust signal timings based on real-time traffic data to optimize traffic flow.
5) Safety Enhancement: Use IoT to enhance road safety by detecting and reporting accidents, road hazards, and unsafe driving behaviours.
6) Public Information Dissemination: Develop mobile apps and digital signage systems to provide commuters with real-time traffic information, alternative routes, and public transportation options.
7) Environmental Impact Reduction: Encourage eco-friendly transportation modes by offering incentives and information on low-emission travel options.
8) Resource Allocation: Improve resource allocation and incident response by

enabling authorities to monitor and manage traffic remotely.

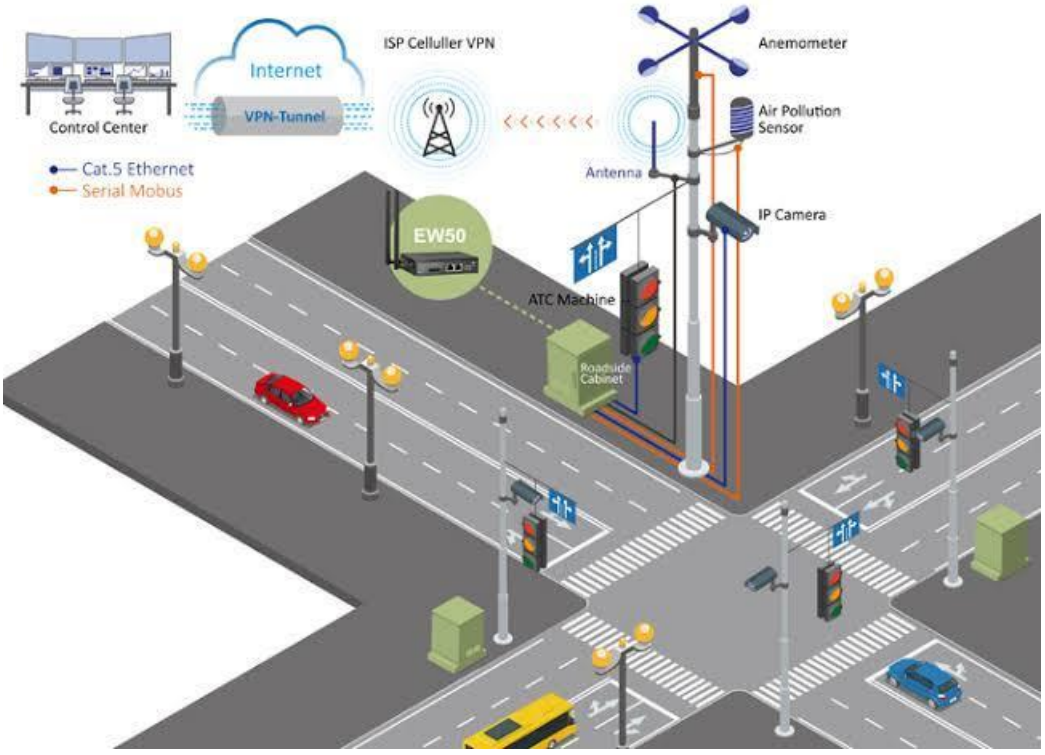
Conclusion:

IoT technology holds immense potential for revolutionizing traffic management in urban areas.

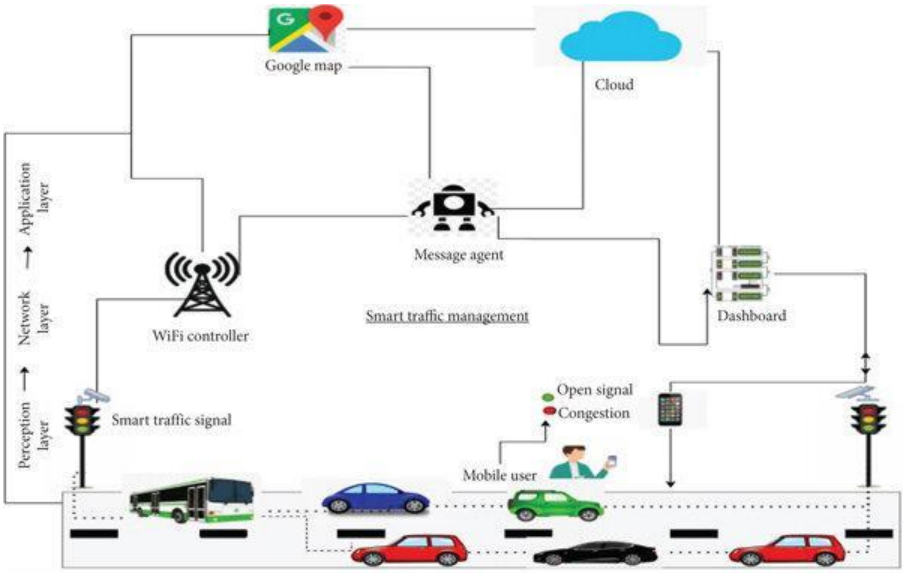
By addressing congestion, safety concerns, environmental impact, and In efficiencies in transportation systems, IoT-based solutions aim to create smarter, safer, and more sustainable cities.

Achieving the stated objectives will require collaboration between government agencies, private sector stakeholders, and technology providers to build robust and integrated IoT-driven traffic management systems.

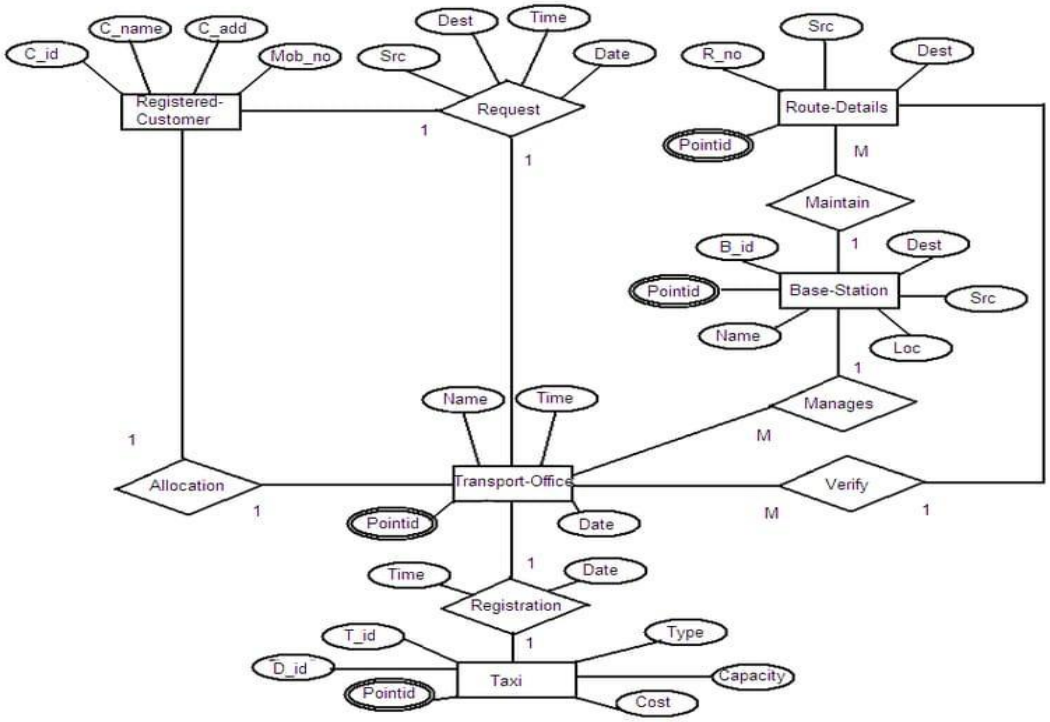
1.4. CASE STUDY DIAGRAM:



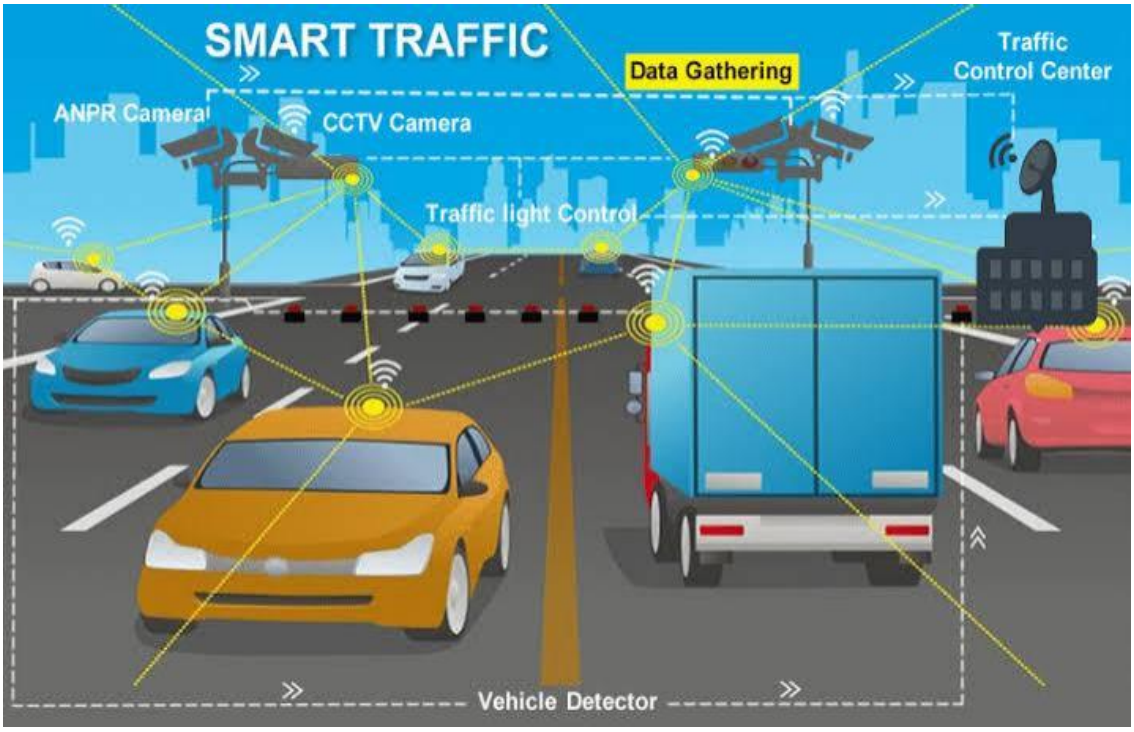
1.5. SYSTEM ARCHITECTURE DIAGRAM:



1.6. ER DIAGRAM:



1.7. DIAGRAM:


SMART TRAFFIC
Data Gathering
Traffic
Control Center
ANPR Camera
CCTV Camera
Traffic light Control
Vehicle Detector

CHAPTER – 2

2.0. INNOVATION:

Step 1:

Device Registration:

- IoT devices must register with the network or server upon activation.
- Gather device information (e.g., type, capabilities, location).
- Assign a unique identifier (ID) to each device.

Step 2:

Data Prioritization:

- Categorize data into different priority levels based on its importance and urgency.
- Consider factors like critical sensor data, control commands, and non- critical data.

Step 3:

Traffic Analysis:

- Continuously monitor network traffic to identify congestion and bottlenecks.
- Use algorithms to analyze traffic patterns and device behavior.

Step 4:

Quality of Service (QoS) Management:

- Allocate network resources (bandwidth, latency, etc.) based on QoS requirements.
- Ensure critical data gets preferential treatment to meet low-latency and reliability needs.

Step 5:

Load Balancing:

- Distribute traffic across multiple servers or edge devices to prevent overloading.
- Implement load balancing algorithms like Round Robin, Least Connections, or Weighted Round Robin.

Step 6:

Data Compression and Aggregation:

- Compress data before transmission to reduce bandwidth usage.
- Aggregate data from multiple devices when possible to minimize individual transmissions.

Step7:

Edge Computing:

- Utilize edge devices to process data locally, reducing the need for centralized data transfer.
- Implement decision-making logic at the edge to reduce latency.

Step 8:

Predictive Analysis:

- Use predictive analytics to forecast traffic spikes and adjust resources accordingly.
- Employ machine learning models to anticipate device behaviour.

Step 9 :

Security Measures:

- Encrypt data during transmission and storage to protect against unauthorized access.
- Implement access control mechanisms and authentication for device connections.

Step 10:

Adaptive Routing:

- Dynamically select the most efficient route for data based on current network conditions.
- Implement routing protocols like MQTT, CoAP, or AMQP.

Step 11:

Data Retention and Cleanup:

- Define data retention policies to manage storage space for historical data.
- Automatically remove or archive obsolete data.

Step 13:

Monitoring and Reporting:

- Continuously monitor network performance and generate reports on traffic patterns and anomalies.
- Use this data to fine-tune traffic management strategies.

Step 14:

Redundancy and Fail Over:

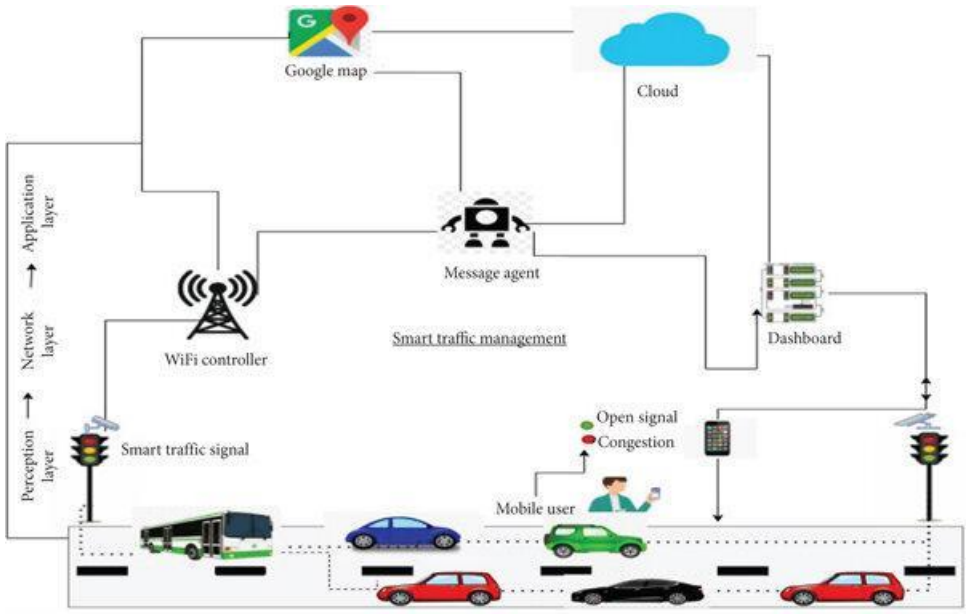
- Implement redundancy and fail over mechanisms to ensure continuous service availability.
- Prepare for device or server failures.

Step 15:

Regular Updates:

- Keep the traffic management system up-to-date with the latest security patches and optimizations.

2.1. SYSTEM ARCHITECTURE DIAGRAM:



CHAPTER – 3

3.0 BUILDING THE IOT TRAFFIC MANAGEMENT SYSTEM:

An intelligent traffic monitoring system using IoT capabilities has so many factors & use cases, including;

Traffic Lights and IoT Control Systems:

- Smart traffic signals may look like a typical stoplight, yet they utilize an array of sensors to monitor real-time traffic.
- Usually, the goal is to help cars reduce the amount of time spent idle.
- And IoT technology enables the various signals to communicate with each other.
- This is while adapting to changing traffic conditions in real time.
- The outcome is less time spent in traffic jams and even reduced carbon emissions.

Parking Enabled through IoT:

- Smart meters and mobile apps make on-street parking spaces easily accessible with instant notifications.
- Drivers receive alerts whenever a parking spot is available to reserve it instantly.
- The app gives easy directions to the parking spot with a convenient online payment option.

Emergency Assistance through IoT:

- A traffic monitoring system using IoT technology enables emergency responders to speed up the care mechanism in case of accidents late at night or in isolated locations.
- The sensors on the road detect any accident, and the problem is immediately reported to the traffic management system.
- This request is passed on to relevant authorities to take corrective action.
- Emergency response personnel would include medical technicians, police officers, and fire departments for enhanced responsiveness and timely intervention.

Commute Assistance:

- With every vehicle acting as an IoT sensor, a dedicated app can make suggestions, determine optimal routes & provide advance notice of accidents or traffic jams.
- Further, it can even suggest the best time to leave.
- It is all because of a robust algorithm that helps reduce driving time with intelligent traffic lights.

Traffic Jam Detection:

- With cloud connectivity, sensors, and CCTV cameras tracking intersections 24×7, technicians can remotely monitor all the streets in real-time from the city's traffic control room.

Connected Vehicles:

- A smart traffic system using IoT technology can connect with roadside tracking devices to enable direct communication between intelligent vehicles & intersections.

Modular Control:

- Real-time detection of congestion triggers dynamic adjustments in the systems meant for controlling traffic lights, express lanes, and entry alarms.

Emergency Navigation:

- A system with edge data processing & programmatic alerting capabilities can alert response units (police, ambulance & tow trucks) in case of a car crash or collision.
- It reduces the crucial time an injured driver or passenger remains unattended.

Road Safety Analytics:

- Systems with pattern detection capabilities can immediately flag high cruising speeds and reckless driver or inappropriate pedestrian behavior.

3.1. PYTHON SCRIPT ON THE IOT DEVICES TO SEND REAL-TIME TRAFFIC DATA.

CODING:

```
import random
import time
def generate_traffic_data():
while True:
vehicle_count = random.randint(0, 100)
speed = random.uniform(0, 120)
location = (random.uniform(0, 100), random.uniform(0, 100)) # Simulated location
timestamp = time.strftime("%Y-%m-%d %H:%M:%S") # Current timestamp
# Send this data to your IoT platform or save it locally to process later
print(f"Timestamp: {timestamp}, Vehicle Count: {vehicle_count}, Speed:
{speed} km/h, Location: {location}")
time.sleep(1)
```

```
if _name_ == "_main_":
generate_traffic_data()
```

OUTPUT:

Timestamp: 2023-10-18 14:22:18, Vehicle Count: 80, Speed:
23.982957202059346 km/h, Location: (22.013888576550556,
56.59690956446737)
Timestamp: 2023-10-18 14:22:19, Vehicle Count: 71, Speed:
111.95609915999425 km/h, Location: (0.006061602931839438,
63.48016203424716) Timestamp: 2023-10-18 14:22:20, Vehicle Count: 70,
Speed:
67.88556360660677 km/h, Location: (4.578779594519955,
29.056837553147375)
Timestamp: 2023-10-18 14:22:21, Vehicle Count: 86, Speed:
17.850449446956365 km/h, Location: (40.73035451442158,
72.48716817538934)
Timestamp: 2023-10-18 14:22:22, Vehicle Count: 89, Speed:
98.10197694001168 km/h, Location: (58.748320052404914,
55.175863363770716)
Timestamp: 2023-10-18 14:22:23, Vehicle Count: 75, Speed:
46.66227094987911 km/h, Location: (24.27244226525277,
67.72147927999055)
Timestamp: 2023-10-18 14:22:24, Vehicle Count: 51, Speed:
103.777097488218 km/h, Location: (30.415536630024775,
89.45853467816205) Timestamp: 2023-10-18 14:22:25, Vehicle Count: 69, Speed:
39.85728466345141 km/h, Location: (73.34316287713563,
3.556945360895547)
Timestamp: 2023-10-18 14:22:26, Vehicle Count: 85, Speed:
40.03862252763813 km/h, Location: (15.197803169424684,
55.61375063044336)
Timestamp: 2023-10-18 14:22:27, Vehicle Count: 52, Speed:
114.4143986834385 km/h, Location: (42.484576274819574,
54.5221221168255)
Timestamp: 2023-10-18 14:22:28, Vehicle Count: 51, Speed:
84.93804540845314 km/h, Location: (96.89995045143885,
3.7285458630123447)
Timestamp: 2023-10-18 14:22:29, Vehicle Count: 84, Speed:
80.11560668925226 km/h, Location: (48.00695263505678,
69.50372871147627)
Timestamp: 2023-10-18 14:22:30, Vehicle Count: 80, Speed:

70.42016767922607 km/h, Location: (52.885618020540406,
3.7190080636395395)
Timestamp: 2023-10-18 14:22:31, Vehicle Count: 70, Speed:
57.843435622456155 km/h, Location: (19.793647848701756,
34.22597523274638)Timestamp: 2023-10-18 14:22:32, Vehicle Count: 0,
Speed:
1.3241922731435674 km/h, Location: (23.974422488124603,
42.90457267301869)
Timestamp: 2023-10-18 14:22:33, Vehicle Count: 92, Speed:
93.30340089577095 km/h, Location: (40.12180756115809,
43.16340535189575)
Timestamp: 2023-10-18 14:22:34, Vehicle Count: 23, Speed:
52.053484551217885 km/h, Location: (82.49742272831999,
68.90252198972465)
Timestamp: 2023-10-18 14:22:35, Vehicle Count: 91, Speed:
88.37357088010494 km/h, Location: (57.822259969205106,
67.5287068520294)
Timestamp: 2023-10-18 14:22:36, Vehicle Count: 14, Speed:
96.46389940613187 km/h, Location: (33.237385520316565,
44.99972638583169)
Timestamp: 2023-10-18 14:22:37, Vehicle Count: 34, Speed:
63.8264888328261 km/h, Location: (21.974120803034182,
99.84908134638971)
Timestamp: 2023-10-18 14:22:38, Vehicle Count: 22, Speed:
45.28071466426779 km/h, Location: (70.41291849337225,
89.58727773458496)
Timestamp: 2023-10-18 14:22:39, Vehicle Count: 93, Speed:16.931705171836597 km/h, Location: (30.988757032926916, 73.38567213232197)
Timestamp: 2023-10-18 14:22:40, Vehicle Count: 88, Speed:
86.59117079565048 km/h, Location: (64.36417591697575, 89.37890792914486)
Timestamp: 2023-10-18 14:22:41, Vehicle Count: 22, Speed:
106.59667771177133 km/h, Location: (6.2275123849178105,
17.689225231848038)
Timestamp: 2023-10-18 14:22:42, Vehicle Count: 13, Speed:
20.386925187928608 km/h, Location: (72.56632358661557,
22.122013142211415)
Timestamp: 2023-10-18 14:22:43, Vehicle Count: 52, Speed:
62.607016293183065 km/h, Location: (24.264122626718898, 76.62957208133759)
Timestamp: 2023-10-18 14:22:44, Vehicle Count: 66,
Speed:
91.5594403619136 km/h, Location: (42.36122931680594,
35.318497583181916)
Timestamp: 2023-10-18 14:22:45, Vehicle Count: 42, Speed:

46.14202788399011 km/h, Location: (89.83829418037492,
31.01113757622429)
Timestamp: 2023-10-18 14:22:46, Vehicle Count: 95, Speed:
67.87636136418439 km/h, Location: (98.56480954451311,
77.83136736434257)
Timestamp: 2023-10-18 14:22:47, Vehicle Count: 41, Speed:
104.33516166147518 km/h, Location: (27.019676925626445, 42.86067826640752)
Timestamp: 2023-10-18 14:22:48, Vehicle Count: 25, Speed:
38.89540833897949 km/h, Location: (80.78136434710731,
52.544283769682146)
Timestamp: 2023-10-18 14:22:49, Vehicle Count: 84, Speed:
47.843480481734815 km/h, Location: (75.54230707637576,
55.64863128295181)
Timestamp: 2023-10-18 14:22:50, Vehicle Count: 44, Speed:
35.85100988649602 km/h, Location: (35.78872375904367,
52.409786635660375)
Timestamp: 2023-10-18 14:22:51, Vehicle Count: 29, Speed:
40.385435464887316 km/h, Location: (62.24642004175161,
62.63603586306146)
Timestamp: 2023-10-18 14:22:52, Vehicle Count: 97, Speed:
79.35928470740481 km/h, Location: (13.791955400508982,
64.41725829179526)
Timestamp: 2023-10-18 14:22:53, Vehicle Count: 94, Speed:
60.50980009087708 km/h, Location: (4.537531421622997,23.91179473369407)
Timestamp: 2023-10-18 14:22:54, Vehicle Count: 53, Speed:
38.03697988981227 km/h, Location: (79.62814839161949,
43.06976367982494)
Timestamp: 2023-10-18 14:22:55, Vehicle Count: 92, Speed:
76.10586473656966 km/h, Location: (84.23547091540847,
1.5003798538450797)Timestamp: 2023-10-18 14:22:56, Vehicle Count: 71,
Speed:
58.77671634750012 km/h, Location: (21.00724380461, 3.9914829465194868)
Timestamp: 2023-10-18 14:22:58, Vehicle Count: 19, Speed:
15.31749992892323 km/h, Location: (37.15002354063658,
55.52922618520482)
Timestamp: 2023-10-18 14:22:59, Vehicle Count: 47, Speed:
13.708816862661655 km/h, Location: (75.17097466057065,
13.11371137154158)
Timestamp: 2023-10-18 14:23:00, Vehicle Count: 12, Speed:
84.32960865302107 km/h, Location: (76.27901186161883,
42.30803132630536)
Timestamp: 2023-10-18 14:23:01, Vehicle Count: 14, Speed:

84.43725122670438 km/h, Location: (31.080893554563094,
61.57411567208943)
Timestamp: 2023-10-18 14:23:02, Vehicle Count: 40, Speed:
92.4831586555722 km/h, Location: (38.49992237573696,
16.383314035370567)
Timestamp: 2023-10-18 14:23:03, Vehicle Count: 89, Speed:
75.82128726157326 km/h, Location: (74.88843195835935,
64.57716234472498)
Timestamp: 2023-10-18 14:23:04, Vehicle Count: 14, Speed:
3.0073498054285253 km/h, Location: (83.96401171852855,
33.227097225673695)
Timestamp: 2023-10-18 14:23:05, Vehicle Count: 75, Speed:
13.137136592562367 km/h, Location: (29.87639854395736,
52.77821930739395)
Timestamp: 2023-10-18 14:23:06, Vehicle Count: 68, Speed:
91.72454297475214 km/h, Location: (40.91742750165939,
82.13195091246949)
Timestamp: 2023-10-18 14:23:07, Vehicle Count: 41, Speed:
104.09827413049813 km/h, Location: (6.217132880305821,
91.57076107917771)
Timestamp: 2023-10-18 14:23:08, Vehicle Count: 71, Speed:
114.82818870999914 km/h, Location: (83.62731508455187,
83.9357292469559)Timestamp: 2023-10-18 14:23:09, Vehicle Count: 47,
Speed:9.143577537570886 km/h, Location: (69.87125027837797,
81.73594239235241)
Timestamp: 2023-10-18 14:23:10, Vehicle Count: 39, Speed:
103.63431661648444 km/h, Location: (49.39876380613726,
11.669255171277737)
Timestamp: 2023-10-18 14:23:11, Vehicle Count: 81, Speed:
16.23236411555188 km/h, Location: (31.51148497650922,
20.371430986619465)
Timestamp: 2023-10-18 14:23:12, Vehicle Count: 44, Speed:
109.57665605077801 km/h, Location: (8.753465023681418,
65.35424697282065)
Timestamp: 2023-10-18 14:23:13, Vehicle Count: 49, Speed:
119.7627962619841 km/h, Location: (50.120180502076906,
61.86320191427447)
Timestamp: 2023-10-18 14:23:14, Vehicle Count: 2, Speed:
55.66946823807223 km/h, Location: (5.4822840418873975,
81.93508144869313)
Timestamp: 2023-10-18 14:23:15, Vehicle Count: 58, Speed:
113.10589897834542 km/h, Location: (92.65151732060984,
23.479516590973603)

Timestamp: 2023-10-18 14:23:16, Vehicle Count: 54, Speed:
12.690335845574765 km/h, Location: (67.47194332574924,
25.468930748180618)
Timestamp: 2023-10-18 14:23:17, Vehicle Count: 63, Speed:
42.72027669533083 km/h, Location: (59.90728545395946,
53.60592723555923)
Timestamp: 2023-10-18 14:23:18, Vehicle Count: 79, Speed:
79.60623243559951 km/h, Location: (90.98454326119023,
5.737689005959767)
Timestamp: 2023-10-18 14:23:19, Vehicle Count: 71, Speed:
7.600557933964698 km/h, Location: (65.78925543062515,
46.95984384009956)
Timestamp: 2023-10-18 14:23:20, Vehicle Count: 85, Speed:
84.84274780031873 km/h, Location: (91.15795168680779,
88.74362017898511) Timestamp: 2023-10-18 14:23:21, Vehicle Count: 76,
Speed:
91.36382761501947 km/h, Location: (67.82365789059929,
58.49811625020376)
Timestamp: 2023-10-18 14:23:22, Vehicle Count: 16, Speed:
21.484683240153633 km/h, Location: (67.65364221134212,
43.348194098855984)
Timestamp: 2023-10-18 14:23:23, Vehicle Count: 18, Speed:
12.065719584007098 km/h, Location: (38.38661790165164,43.13023646294083)
Timestamp: 2023-10-18 14:23:24, Vehicle Count: 32, Speed:
92.15705172788417 km/h, Location: (42.63549674386675,
41.102746292747796)
Timestamp: 2023-10-18 14:23:25, Vehicle Count: 73, Speed:
26.087146692579548 km/h, Location: (2.3832020958835742,
17.47713368549991)
Timestamp: 2023-10-18 14:23:26, Vehicle Count: 84, Speed:
62.322268184344345 km/h, Location: (13.276567240292248,
53.9992893258714)
Timestamp: 2023-10-18 14:23:27, Vehicle Count: 73, Speed:
97.71666694797548 km/h, Location: (46.05625871540545,
70.92438755382457)
Timestamp: 2023-10-18 14:23:28, Vehicle Count: 30, Speed:
57.003127648167336 km/h, Location: (50.104815135446955,
9.975477513336916)
Timestamp: 2023-10-18 14:23:29, Vehicle Count: 42, Speed:
5.8169506369367285 km/h, Location: (35.04858786357868,
95.29325990059677)
Timestamp: 2023-10-18 14:23:30, Vehicle Count: 68, Speed:

116.43540053463782 km/h, Location: (21.159978302744374, 23.03949978690385)
Timestamp: 2023-10-18 14:23:31, Vehicle Count: 21, Speed: 110.17966624310505 km/h, Location: (4.649841934550625, 35.81706170369182)
Timestamp: 2023-10-18 14:23:32, Vehicle Count: 21, Speed: 71.94770120439239 km/h, Location: (34.36117655222836, 50.612225803814084)

Process finished with exit code -1073741510 (0xC000013A: interrupted by Ctrl+C)

CHAPTER – 4

4.0 USE WEB DEVELOPMENT TECHNOLOGY TO CREATE A PLATFORM THAT DISPLAYS REAL-TIME TRAFFIC INFORMATION.

CODING:

```
from tracking.centroidtracker import CentroidTracker
from tracking.trackableobject import TrackableObject
import tensornets as nets
import cv2
import numpy as np
import time
import dlib
import tensorflow.compat.v1 as tf
import os
import threading
def countVehicles(param):
# param -> path of the video
# list -> number of vehicles will be written in the list
# index ->Index at which data has to be written
tf.disable_v2_behavior()
# Image size must be '416x416' as YoloV3 network expects that specific image size as input
[ mg_size = 416
inputs = tf.placeholder(tf.float32, [None, img_size, img_size, 3])
model = nets.YOLOv3COCO(inputs, nets.Darknet19)
ct = CentroidTracker(maxDisappeared=5, maxDistance=50) # Look into 'CentroidTracker' for further info about parameterstrackers = [] # List of all dlib trackers trackableObjects = {} # Dictionary of trackable objects containing object's ID and its' corresponding centroid/s
skip_frames = 10 # Numbers of frames to skip from detecting
confidence_level = 0.40 # The confidence level of a detection
total = 0 # Total number of detected objects from classes of interest
use_original_video_size_as_output_size = True # Shows original video as output and not the 416x416 image that is used as yolov3 input (NOTE: Detection still happens with 416x416 img size but the output is displayed in original video size if this parameter is True) video_path = os.getcwd() + param # "/videos/4.mp4"
video_name = os.path.basename(video_path)
# print("Loading video {video_path}...".format(video_path=video_path))
```

```
    if not os.path.exists(video_path):
    print("File does not exist. Exited.")
    exit()
    # YoloV3 detects 80 classes represented below


    all_classes = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train",
"truck", \
"boat", "traffic light", "fire hydrant", "stop sign", "parking
meter", "bench", \
"bird", "cat", "dog", "horse", "sheep", "cow", "elephant",
"bear", "zebra", "giraffe", \
"backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee",
"skis", "snowboard", \
"sports ball", "kite", "baseball bat", "baseball glove",
"skateboard", "surfboard", \
"tennis racket", "bottle", "wine glass", "cup", "fork", "knife",
"spoon", "bowl", "banana", \
"apple", "sandwich", "orange", "broccoli", "carrot", "hot dog",
"pizza", "donut", "cake", \
"chair", "sofa", "pottedplant", "bed", "diningtable", "toilet",
"tvmonitor", "laptop", "mouse", \"remote", "keyboard", "cell phone", "microwave", "oven",
"toaster", "sink", "refrigerator", \
"book", "clock", "vase", "scissors", "teddy bear", "hair drier",
"toothbrush"]
    # Classes of interest (with their corresponding indexes for easier looping)
    classes = { 1 : 'bicycle', 2 : 'car', 3 : 'motorbike', 5 : 'bus', 7 : 'truck' }
    with tf.Session() as sess:
    sess.run(model.pretrained())
    cap = cv2.VideoCapture(video_path)
    # Get video size (just for log purposes)
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
     height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
     # Scale used for output window size and net size
     width_scale = 1
     height_scale = 1
     if use_original_video_size_as_output_size:
     width_scale = width / img_size
     height_scale = height / img_size
     def drawRectangleCV2(img, pt1, pt2, color, thickness,
     width_scale=width_scale, height_scale=height_scale):
     point1 = (int(pt1[0] * width_scale), int(pt1[1] * height_scale))
     point2 = (int(pt2[0] * width_scale), int(pt2[1] * height_scale))
```

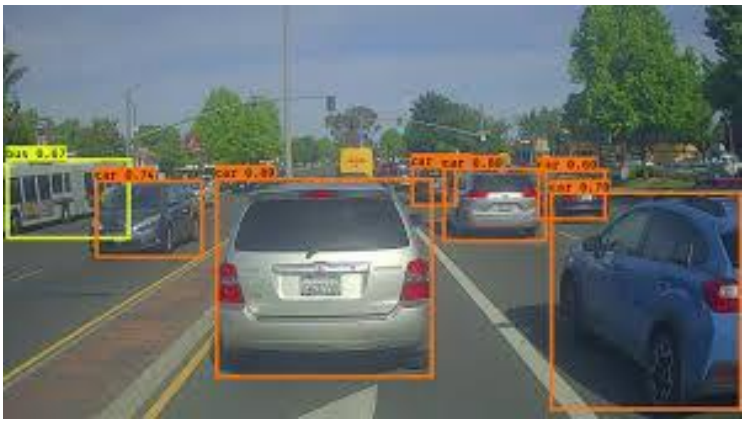
```
    return cv2.rectangle(img, point1, point2, color, thickness)
    def drawTextCV2(img, text, pt, font, font_scale, color, lineType,
    width_scale=width_scale, height_scale=height_scale):pt = (int(pt[0] * width_scale),
nt(pt[1] * height_scale))
    cv2.putText(img, text, pt, font, font_scale, color, lineType)
    def drawCircleCV2(img, center, radius, color, thickness,
    width_scale=width_scale, height_scale=height_scale):
    center = (int(center[0] * width_scale), int(center[1] * height_scale))
    cv2.circle(img, center, radius, color, thickness)
    # Python 3.5.6 does not support f-strings (next line will generate syntax error)
    #print(f"Loaded {video_path}. Width: {width}, Height: {height}")
    # print("Loaded {video_path}. Width: {width}, Height:
     height}".format(video_path=video_path, width=width, height=height))


     skipped_frames_counter = 0

     while(cap.isOpened()):
     try :
     ret, frame = cap.read()
     img = cv2.resize(frame, (img_size, img_size))
     except:
     print(total_str)
     output_img = frame if use_original_video_size_as_output_size else
     img
     tracker_rects = []
     if skipped_frames_counter == skip_frames:
     # Detecting happens after number of frames have passes specified by 'skip_frames' variable
value# print("[DETECTING]")
     trackers = []
     skipped_frames_counter = 0 # reset counter
     np_img = np.array(img).reshape(-1, img_size, img_size, 3)
     start_time=time.time()
     predictions = sess.run(model.preds, {inputs:
     model.preprocess(np_img)})
     # print("Detection took %s seconds" % (time.time() - start_time))
     # model.get_boxes returns a 80 element array containing information about detected
classes
     # each element contains a list of detected boxes, confidence level ...
     detections = model.get_boxes(predictions, np_img.shape[1:3])
     np_detections = np.array(detections)
     # Loop only through classes we are interested in
     for class_index in classes.keys():
```

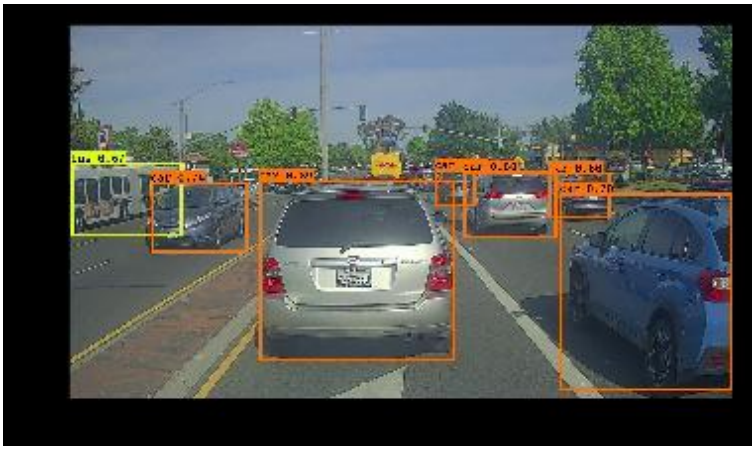
```
        local_count = 0
        class_name = classes[class_index]
        # Loop through detected infos of a class we are interested in
        for i in range(len(np_detections[class_index])):
        box = np_detections[class_index][i]
        if np_detections[class_index][i][4] >=
        confidence_level:# print("Detected ", class_name, " with
        confidence of ", np_detections[class_index][i][4])
        local_count += 1
        startX, startY, endX, endY = box[0],
        box[1], box[2], box[3]
        drawRectangleCV2(output_img, (startX, startY), (endX, endY), (0, 255, 0), 1)
        drawTextCV2(output_img, class_name,
        (startX, startY), cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 0, 255), 1)
        # Construct a dlib rectangle object from the bounding box coordinates and then start the
dlib correlation
        tracker = dlib.correlation_tracker()
        rect = dlib.rectangle(int(startX),
        int(startY), int(endX), int(endY))
        tracker.start_track(img, rect)
        # Add the tracker to our list of trackers so we can utilize it during skip frames
        trackers.append(tracker)
        # Write the total number of detected objects for a given class on this frame

        # print(class_name," : ", local_count)


      else:

      # If detection is not happening then track previously detected objects (if any)
      # print("[TRACKING]")
      skipped_frames_counter += 1 # Increase the number frames for which we did not use
detection# Loop through tracker, update each of them and display their rectangle
      for tracker in trackers:
      tracker.update(img)
      pos = tracker.get_position()
      # Unpack the position object
      startX = int(pos.left())
      startY = int(pos.top())
      endX = int(pos.right())
      endY = int(pos.bottom())
      # Add the bounding box coordinates to the tracking rectangles list
      tracker_rects.append((startX, startY, endX, endY))
```

```
    # Draw tracking rectangles
    drawRectangleCV2(output_img, (startX, startY), (endX, endY), (255, 0, 0), 1)
    # Use the centroid tracker to associate the (1) old object centroids with (2) the newly
computed object centroids
     objects = ct.update(tracker_rects)
     # Loop over the tracked objects
     for (objectID, centroid) in objects.items():
     # Check to see if a trackable object exists for the current object ID
     to = trackableObjects.get(objectID, None)if to is None:
     # If there is no existing trackable object, create one
     to = trackableObject(objectID, centroid)
     else:
     to.centroids.append(centroid)
     # If the object has not been counted, count it and mark it as counted
     if not to.counted:
     total += 1
     to.counted = True
     # Store the trackable object in our dictionary
     trackableObjects[objectID] = to
     # Draw both the ID of the object and the centroid of the object on the output frame
     object_id = "ID {}".format(objectID)
     drawTextCV2(output_img, object_id, (centroid[0] - 10, centroid[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
     drawCircleCV2(output_img, (centroid[0], centroid[1]), 2, (0, 255, 0), -1)
     # Display the total count so far
     total_str = str(total)
     drawTextCV2(output_img, total_str, (10, 30),
     cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
     # Display the current frame (with all annotations drawn up to this point)
     cv2.imshow(video_name, output_img)
     key = cv2.waitKey(1) & 0xFF

     if key == ord('q'): # QUIT (exits)break
     elif key == ord('p'):
     cv2.waitKey(0) # PAUSE (Enter any key to continue)



    cap.release()

    cv2.destroyAllWindows()
    print("Exited")
    """
    function which will run our code
```

will write the number of veicles in the list provided
"""
if __name__ == "__main__":
countVehicles("/videos/test.mp4")
# Logic for setting the time for each signal

OUTPUT:





4.1 BUILDING THE PROJECT BY DEVELOPING THE TRAFFIC INFORMATION PLATFORM AND MOBILE APPS.

CODING:

[A] getCurrentPosition() method:

The getCurrentPosition(successCallback, errorCallback, options) method steps are:
If the current settings object's relevant global object's associated Document is not fully active:

1) Call back with error errorCallback and POSITION_UNAVAILABLE.
2) Terminate this algorithm.
3) In parallel, request a position passing successCallback, errorCallback, and options.

// A one-shot position request:

```
navigator.geolocation.getCurrentPosition(position => {
const { latitude, longitude } = position.coords;
// Show a map centered at latitude / longitude.
});
```

[B] watchPosition() method:

The watchPosition(successCallback, errorCallback, options) method steps are:
If the current settings object's relevant global object's associated Document is not fully active:

1. Call back with error passing errorCallback and POSITION_UNAVAILABLE.
2. Return 0.
3. Let watchId be an implementation-defined unsigned long that is greater than zero.
4. Append watchId to this's [[watchIDs]].
5. In parallel, request a position passing successCallback, errorCallback, options, and watchId.
6. Return watchId.

Watching a position for repeated updates:

```
const watchId = navigator.geolocation.watchPosition(position => {
const { latitude, longitude } = position.coords;
// Show a map centered at latitude / longitude.
});
```

[C] clearWatch() method:

When clearWatch**()** is invoked, the user agent MUST:

1) Remove watchId from this's [[watchIDs]].

Using clearWatch():

```
const watchId = navigator.geolocation.watchPosition(
position => console.log(position));
function buttonClickHandler() {
// Cancel the updates when the user clicks a button.
navigator.geolocation.clearWatch(watchId);
}
```

A HTML button that when pressed stops watching the position.

```
<button onclick="buttonClickHandler()">
Stop watching location
</button>
```

[D] Handling errors:

```
// Request repeated updates.const watchId =
navigator.geolocation.watchPosition(
scrollMap, handleError
);
Function scrollMap(position) {
const { latitude, longitude } = position.coords;
// Scroll map to latitude / longitude.
}
function handleError(error) {
// Display error based on the error code.
const { code } = error;
switch (code) {
case GeolocationPositionError.TIMEOUT:// Handle timeout.
break;
case GeolocationPositionError.PERMISSION_DENIED:
// User denied the request.
break;
case GeolocationPositionError.POSITION_UNAVAILABLE:
// Position not available.
break;
}
}
```

[E] Getting cached position:

```
navigator.geolocation.getCurrentPosition(
successCallback,
console.error,
{ maximumAge: 600_000 }
```

```
);
function successCallback(position) {
// By using the 'maximumAge' member above, the position
// object is guaranteed to be at most 10 minutes old.
}
```

[F] Timing out a position request:

```
// Request a position. We are only willing to wait 10// seconds for it.
navigator.geolocation.getCurrentPosition(successCallback,
errorCallback,
{ timeout: 10_000 }
);
function successCallback(position) {
// Request finished in under 10 seconds...
}
function errorCallback(error) {
switch (error.code) {
case GeolocationPositionError.TIMEOUT:
// We didn't get it in a timely fashion.
doFallback();
// Acquire a new position object,
// as long as it takes.
navigator.geolocation.getCurrentPosition(
successCallback, errorCallback
);
break;
case "...": // treat the other error cases.
}
}
function doFallback() {}
```

[G] Enabling the Geolocation API in an iframe:

```
<iframe
src="https://third-party.com"allow="geolocation">
</iframe>
```

[H] Permissions Policy over HTTP:

```
Permissions-Policy: geolocation=()
```

[I] PositionOptions dictionary:

```
PositionOptions {
boolean enableHighAccuracy = false;
[Clamp] unsigned long timeout = 0xFFFFFFFF;
[Clamp] unsigned long maximumAge = 0;
};
```

4.2. REFERENCES:

[1] W. S. Associate, "Transportation and Economy Report," MDOT State LongRange Transportation Plan, Karachi, 2007.

[2] K. S. D. M. R. B. Patan Rizwan, "Real-Time Smart Traffic Management System for Smart Cities by Using Internet of Things and Big Data," in International Conference on Emerging Technological Trends [ICETT], Kollam, 2016.

[3] C. S. D. L. T. Authority, "Annual Vehicle Statistics," Annual Vehicle Statistics 2015, Karachi, 2015.

[4] M. P. a. B. B. Sivasankar, "IoT Based Traffic Monitoring using Raspberry Pi," Internation Journal of Research in Engineering, Science and Technology (IJRESTs), vol. 1, no. 7, pp. 2454-664x, 2016.

[5] A. Lone, "Karachi's crime malaise," Tribune.com.pk, 18 August 2011. [Online]. Available: https://blogs.tribune.com.pk/story/7540/karachis crime-malaise/. [Accessed 20 September 2017].

[6] S. Sharief, "Road Accidents in Pakistan Reach Alarming High: Who's Responsible," Pakwheels.com, 2016. [Online]. Available: https://www.pakwheels.com/blog/alarming-increase-of-road-accidents in-pakistan/. [Accessed 2017].

[7] B. k. Khan, "Traffic jams in Karachi result in losses worth of millions each day," PakWheels.com, 2013. [Online]. Available: https://www.pakwheels.com/blog/traffic-jams-karachi-result-losses worth-millions-day/. [Accessed 2017].

[8] P. K. K. P. S. T. Prashant Jadhav, "Smart Traffic Control System Using Image Processing," International Research Journal of Engineering and Technology (IRJET), vol. 3, no. 3, pp. 2395-0056, 2016.

[9] E. Leigh, "Advancing integrated and sustainable transport for the Cambridge region," Smarter Cambridge Transport, 2017. [Online]. Available: http://www.smartertransport.uk/. [Accessed 2017].

[10] T. C. N. K. Kartikeya Jain, "Smart vehicle identification system using OCR," in 3rd International Conference on Computational Intelligence & Communication Technology (IEEE-CICT 2017), Ghaziabad, India, 2017.

[11] T. e. a. Osman, "Intelligent traffic management system for a cross section of roads using computer vision," in Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual, 2017.

[12] D. C. J. E. J. Khac-Hoai Nam Bui, "Real-Time Traffic Flow Management Based on Inter-Object Communication: a Case Study at Intersection," in 7th International Conference on Ambient Intelligence (ISAmI'2016), 2016.

[13] V. S. A. M. S. D. K. K. Swathi, "Traffic Density Control and Accident Indicator Using WSN," International Journal for Modern Trends in Science and Technology, vol. 2, no. 4, pp. 2455-3778, 2016.

[14] H. O. Al-Sakran, "Intelligent traffic information system based on the integration of Internet of Things and Agent technology," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 6, no. 2, pp. 37-43, 2015.

[15] A. S. Nidhi D. Agrawal, "Intelligent Real Time Traffic Controller Using Image Processing – A Survey," International Journal of Science and Research (IJSR) ISSN (Online), vol. 4, no. 4, pp. 2319-7064, 2015.

[16] J. D. D. M. A. M. A. J. Vismay Pandit, "Smart Traffic Control System Using Image Processing," International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), vol. 3, no. 1, pp. 2278- 6856, 2014.

[17] D. S. P. S. Y. M. Pranav Maheshwari, "Smart Traffic Optimization," in 3rd International Conference on MOOCs, Innovation and Technology in Education (MITE), IEEE, 2015.

[18] M. A. M. A. T. R. J. S. A.-M. A. B. Fozia Mehboob, "Automated Vehicle Density Estimation from Raw Surveillance Videos," in SAI Computing Conference 2016, London, 2016.

[19] K. B. M. R. M. B. Shubham N. Mahalank, Device to Device Interaction Analysis in IoT based Smart Traffic Management System: An Experimental Approach, Symposium on Colossal Data Analysis and Networking (CDAN), 2016.