Contents

ABSTRACT

Detecting and tracking objects are among the most prevalent and challenging tasks that a surveillance system must accomplish in order to determine meaningful events and suspicious activities, and automatically annotate and retrieve video content. Under the business intelligence notion, an object can be a face, a head, a human, a queue of people, a crowd as well as a product on an assembly line. In this chapter we introduce the reader to main trends and provide taxonomy of popular methods to give an insight to underlying ideas as well as to show their limitations in the hopes of facilitating integration of object detection and tracking for more effective business-oriented video analytics.

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

We are grateful to our project guide Dr.M.M.Deshpande sir for the guidance, inspiration, and constructive suggestions that helpful us in the preparation of this project.
We also thank our colleagues who have helped in successful completion of the project.

Project By:

Shubham Gadhave

Sumit Mishra

Aditya Patil

Ganesh Karli

• Chapter 1: Introduction

1.1 Definition:

Object tracking is an application of deep learning where the program takes an initial set of object detections and develops a unique identification for each of the initial detections and then tracks the detected objects as they move around frames in a video.
In other words, object tracking is the task of automatically identifying objects in a video and interpreting them as a set of trajectories with high accuracy.
Often, there's an indication around the object being tracked, for example, a surrounding square that follows the object, showing the user where the object is on the screen.

1.2 Purpose:

Real time object tracking is the process of locating moving objects over time using the camera in video sequences in real time. The objective of object tracking is to associate target objects in consecutive video frames. Object tracking requires location and shape or features of objects in the video frames. Every tracking algorithm needs to detect moving object. So object detection is the preceding step of object tracking in computer vision applications. After that, detected object can be extracted by the feature of moving object to track that moving object into video scene. It is challenging task in image processing to track the objects into consecutive frames. Various challenges can arise due to complex object motion, irregular shape of object, occlusion of object to object and object to scene and real time processing requirements. Object tracking has a variety of uses, some of which are: surveillance and security, traffic monitoring, video communication, robot vision and animation.

1.3 Scope:

Object detection is a computer vision technique that allows us to identify and locate objects in an image or video. With this kind of identification and localization, object detection can be used to count objects in a scene and determine and track their precise locations, all while accurately labelling them.

1.4 Overview:

2 The objective of object tracking is to associate target objects in consecutive video frames. Object tracking requires location and shape or features of objects in the video frames. Every tracking algorithm needs to detect moving object. So object detection is the preceding step of object tracking in computer vision applications. After that, detected object can be extracted by the feature of moving object to track that moving object into video scene. It is challenging task in image processing to track the objects into consecutive frames.

• Chapter 2: Literature Survey

2.1 Survey of Existing System:

In various fields, there is a necessity to detect the target object and track them effectively while handling occlusions and other included complexities. Many researchers (Almeida andGuting2004, Hsiao-Ping Tsai 2011, Nicolas Papadakis and Aure lie Bugeau 2010) attempted for various approaches in object tracking. The nature of the techniques largely depends on the application domain.

2.2 Limitation of Existing System or Research Gap:

Viewpoint variation. One of the biggest difficulties of object detection is that an object viewed from different angles may Despite being a beneficial method, not every market and/or process can afford to perform object tracking due to the fact that one of the crucial hurdles in training an object tracking model is the training and tracking speed. Tracking algorithms are expected and needed to detect and localize the object in a video in a fraction of a second and with high accuracy. This detection speed can be significantly tampered with involuntarily due to the variety of background distractions in any scenario. Another significant difficulty in object tracking is the variation in the spatial scales. An object can be present in an image (or a video) and in various sizes and orientations.

Another issue with object tracking, which is also a significant issue in object detection and recognition, is occlusion. Occlusion is when multiple objects come so close together that they appear to be merged. This can confuse the computer into thinking the merged object is a single object or simply wrongly identifying the object. look *completely different, Deformation, Occlusion. Illumination* conditions. Cluttered or textured background, Speed.

Aside from these, several issues pose difficulties in object tracking, such as switching of identity after crossing, motion blur, variation in the viewpoint, cluttering of similar objects in the background, low resolution, and variation in the illumination.

2.3 Contribution in Mini Project:

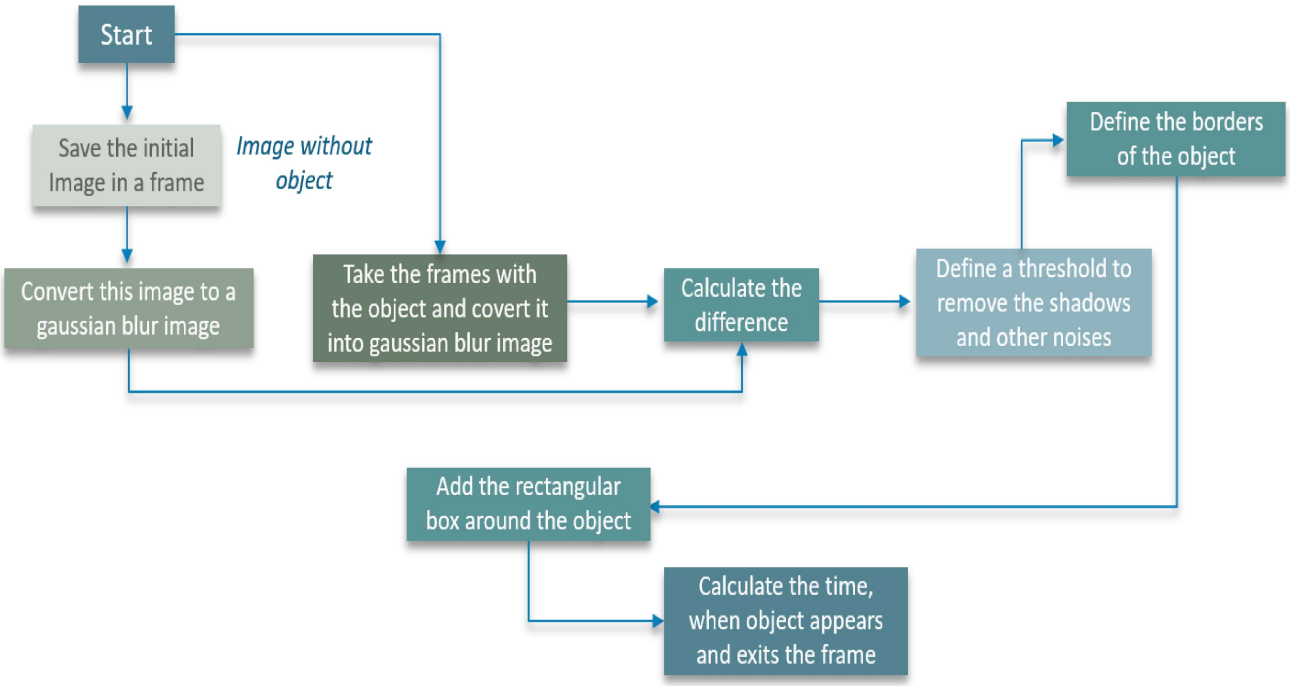
We have not divided the topics as such but contributed in the project simultaneously. It was hard for us to link the project as we made it in parts but then no longer, we successfully linked it and our beautiful project was ready.

• Chapter 3: Proposed System

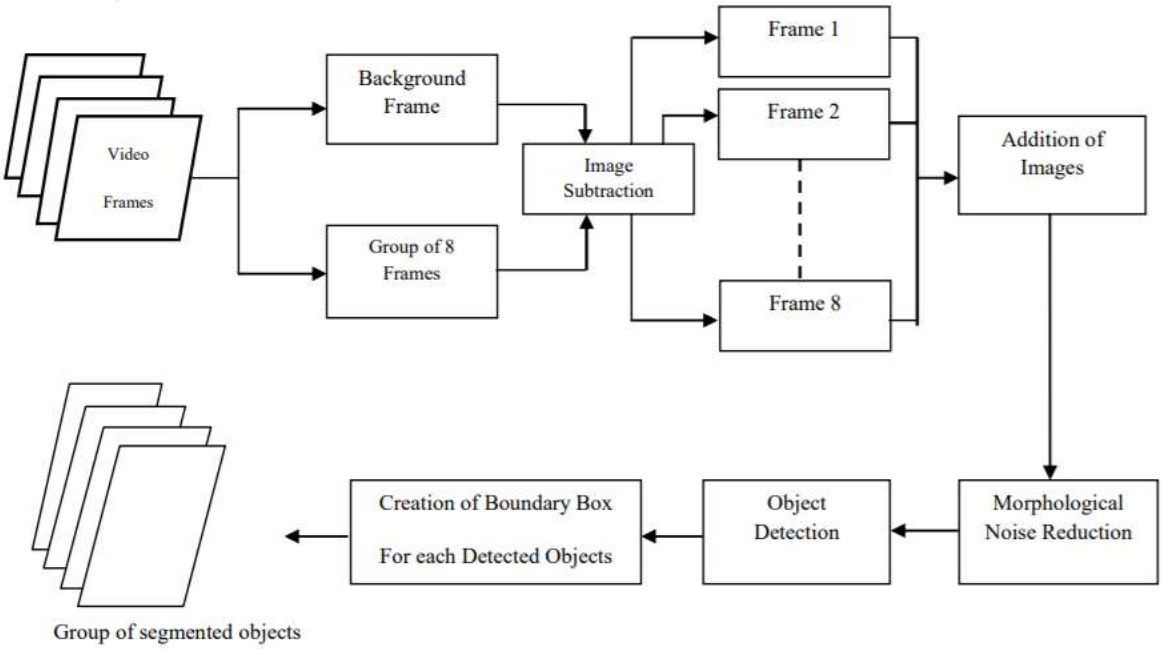
3.1 Introduction:

In this project we are tracking and counting the vehicles passing by in certain video frame. This technology commonly used in system like challan generation.

3.2 Architecture:



3.3 Process Design



3.4 Details of Hardware & Software:

Ram: 8 GB

Processor: AMD Ryzen 5

GPU: Nvidia GTX 1650

Python Version: 3.10

Package Used: OPEN CV

3.5 What is OpenCV?

OpenCV is a well-known open-source library that is primarily used for a variety of computer vision applications. It has also been widely used in machine learning, deep learning, and image processing. It helps in processing data containing images and videos. Since today, OpenCV has been used in several mainstream applications, including object detection and recognition, autonomous cars and robots, automated surveillance, anomaly detection, video and image search retrieval, medical image analysis, and object tracking. It can also be integrated with other libraries and can process array structures of libraries such as NumPy. It is an extensive library in both the sense of functionality and extensions; besides having an enormous toolbox of functions and algorithms; it supports not only Python but also C, C++, and Java. Moreover, it further supports Windows, Linux, ac OS, iOS, and Android.

Nowadays, OpenCV, being a library for computer vision, is the majority of the time used in artificial intelligence and its modern applications

involving visual data such as images and videos. Various convolutional neural network-based architectures demand the support of OpenCV for both preprocessing and postprocessing. To learn more about OpenCV, refer to our study on Essential OpenCV Functions to Get You Started into Computer Vision .

Implementation in Python and OpenCV

Now that we have skimmed all the basic concepts regarding object tracking specifically to be implemented in OpenCV let's head over to the coding part of the article.

We will build the script in parts, but you can get access to the full code on GitHub .

Installing dependencies

Before we start the code, you need some pre-requisites to be installed in your Python environment. You need to install the **opencv-contrib-python** package.

```
pip install opencv-contrib-python
```

Setting up the trackers.

```
import cv2

import sys

(major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')

if __name__ == '__main__' :

    # Set up tracker.

    # Instead of CSRT, you can also use
```

```
    tracker_types = ['BOOSTING', 'MIL','KCF', 'TLD', 'MEDIANFLOW',
'GOTURN', 'MOSSE', 'CSRT']

    tracker_type = tracker_types[7]

    if int(minor_ver) < 3:

        tracker = cv2.Tracker_create(tracker_type)

    else:

        if tracker_type == 'BOOSTING':

            tracker = cv2.TrackerBoosting_create()

        elif tracker_type == 'MIL':

            tracker = cv2.TrackerMIL_create()

        elif tracker_type == 'KCF':

            tracker = cv2.TrackerKCF_create()

        elif tracker_type == 'TLD':

            tracker = cv2.TrackerTLD_create()

        elif tracker_type == 'MEDIANFLOW':

            tracker = cv2.TrackerMedianFlow_create()

        elif tracker_type == 'GOTURN':
```

```
        tracker = cv2.TrackerGOTURN_create()

    elif tracker_type == 'MOSSE':

        tracker = cv2.TrackerMOSSE_create()

    elif tracker_type == "CSRT":

        tracker = cv2.TrackerCSRT_create()
```

The cv2.**version** function returns the version numbers of the OpenCV library installed in your environment. This check is necessary to do before creating the tracker object. This is because any version of OpenCV lower than 3 has a different module to create a specific type of tracker. We first save the name of the eight trackers in a list. Then we check for the version of OpenCV we are working in and then create the tracker object based on the version number.

Capturing the video input

```
# Read video

video = cv2.VideoCapture("input.mp4")

#video = cv2.VideoCapture(0) # for using CAM

# Exit if video not opened.

if not video.isOpened():

  print("Could not open video")

  sys.exit()
```

```
# Read first frame.

ok, frame = video.read()

if not ok:

  print ('Cannot read video file')

  sys.exit()
```

The VideoCapture class can be used to capture a video file from either the webcam integrated with your machine or a video file saved in your local device. Give the path to your video in the argument of VideoCapture in line 2. If you want to use the webcam for tracking, comment on the second line and uncomment the third one. We further do a couple of checks to see if the video file is properly working or not.

Creating the bounding box and initialize the tracker

```
# Define an initial bounding box

bbox = (287, 23, 86, 320)

# Uncomment the line below to select a different bounding box

bbox = cv2.selectROI(frame, False)

# Initialize tracker with first frame and bounding box

ok = tracker.init(frame, bbox)
```

We define an initial random bounding box in the video, or we can select a bounding box of our own choice. This bounding box will contain the object we want to track.

Select the area of the image you wish to track with your mouse

Start the tracker and see the output

```
while True:

    # Read a new frame

    ok, frame = video.read()

    if not ok:

        break

    # Start timer

    timer = cv2.getTickCount()

    # Update tracker

    ok, bbox = tracker.update(frame)

    # Calculate Frames per second (FPS)

    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);

    # Draw bounding box

    if ok:

        # Tracking success

        p1 = (int(bbox[0]), int(bbox[1]))
```

```
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))

        cv2.rectangle(frame, p1, p2, (255,0,0), 2, 1)

    else :

        # Tracking failure

        cv2.putText(frame, "Tracking failure detected", (100,80),
cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2)

    # Display tracker type on frame

    cv2.putText(frame, tracker_type + " Tracker", (100,20),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);

    # Display FPS on frame

    cv2.putText(frame, "FPS : " + str(int(fps)), (100,50),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);

    # Display result

    cv2.imshow("Tracking", frame)

    # Exit if ESC pressed

    if cv2.waitKey(1) & 0xFF == ord('q'): # if press SPACE bar

        break

video.release()

cv2.destroyAllWindows()
```

We start by reading each frame of the video being played. We start the timer and use the tracker to estimate the trajectory of the object in the video. We use the tracker's estimated trajectory to draw the bounding box around the object of interest. The program continues forever and waits for the space bar to be pressed; as soon as the space bar is pressed, the while loop breaks, and the tracking stops.

3.5 Source Code

Object Detection

```
import cv2
import numpy as np


class ObjectDetection:
        def __init__(self,
    weights_path="dnn_model/yolov4.weights",
    cfg_path="dnn_model/yolov4.cfg"):
            print("Loading Object Detection")
            print("Running opencv dnn with YOLOv4")
            self.nmsThreshold = 0.4
            self.confThreshold = 0.5
            self.image_size = 608

            # Load Network
            net = cv2.dnn.readNet(weights_path, cfg_path)

            # Enable GPU CUDA

        net.setPreferableBackend(cv2.dnn.DNN_BACKEND_C
        UDA)

        net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA
        )
            self.model = cv2.dnn_DetectionModel(net)
```

```
        self.classes = []
        self.load_class_names()
        self.colors = np.random.uniform(0, 255, size=(80,
3))

        self.model.setInputParams(size=(self.image_size,
self.image_size), scale=1/255)

    def load_class_names(self,
classes_path="dnn_model/classes.txt"):

        with open(classes_path, "r") as file_object:
            for class_name in file_object.readlines():
                class_name = class_name.strip()
                self.classes.append(class_name)

        self.colors = np.random.uniform(0, 255, size=(80,
3))
        return self.classes

    def detect(self, frame):
        return self.model.detect(frame,
nmsThreshold=self.nmsThreshold,
confThreshold=self.confThreshold)
```

Object Tracking

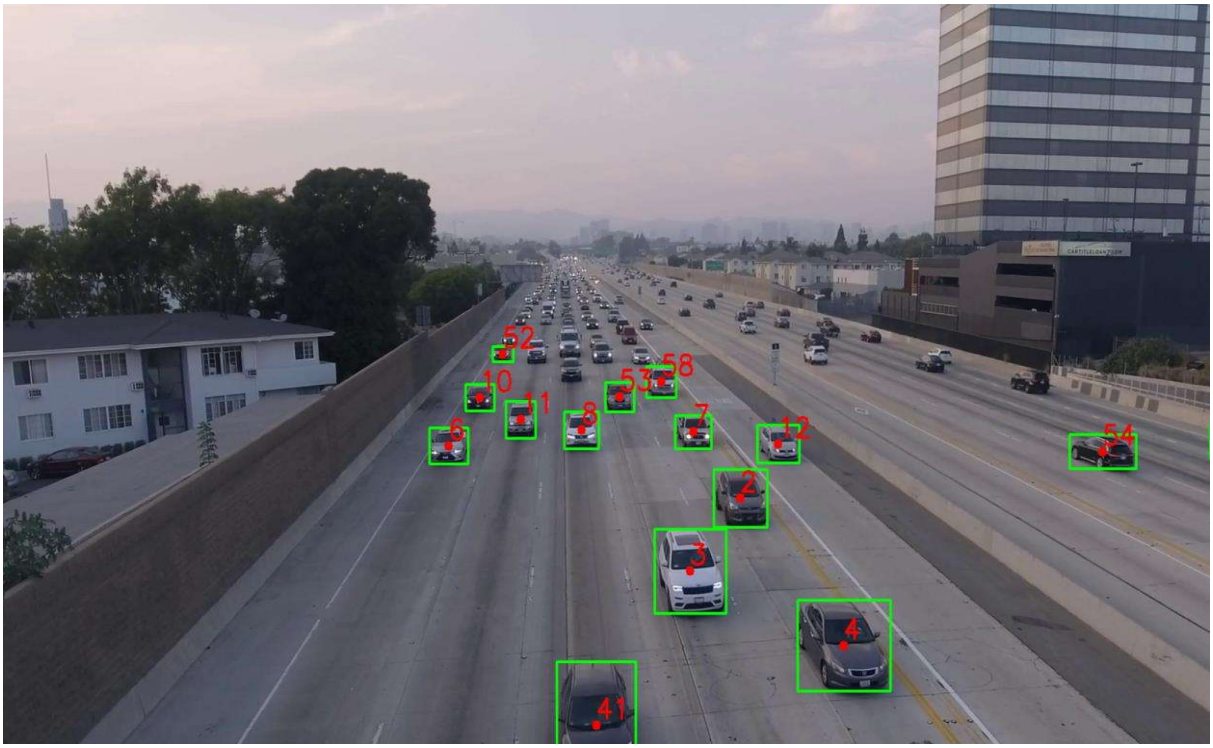
```
import cv2
import numpy as np
from object_detection import ObjectDetection
import math

# Initialize Object Detection
od = ObjectDetection()

cap = cv2.VideoCapture("kharghar3.mp4")

# Initialize count
count = 0
center_points_prev_frame = []

tracking_objects = {}
track_id = 0

        while True:
            ret, frame = cap.read()
            count += 1
            if not ret:
                break

            # Point current frame
            center_points_cur_frame = []
```

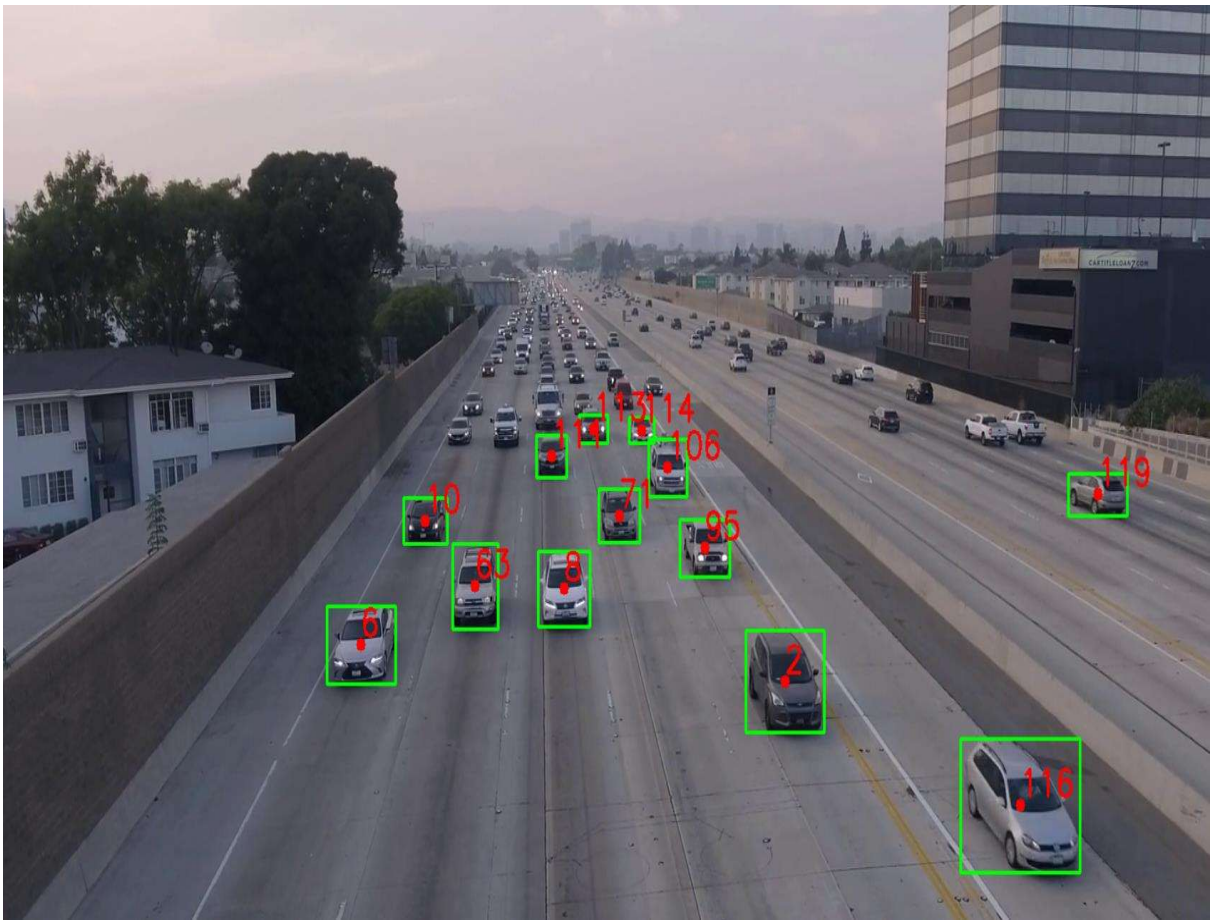
```
    # Detect objects on frame
    (class_ids, scores, boxes) = od.detect(frame)
    for box in boxes:
        (x, y, w, h) = box
        cx = int((x + x + w) / 2)
        cy = int((y + y + h) / 2)
        center_points_cur_frame.append((cx, cy))
        #print("FRAME N°", count, " ", x, y, w, h)

        # cv2.circle(frame, (cx, cy), 5, (0, 0, 255), -1)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255,
0), 2)

    # Only at the beginning we compare previous and
current frame
    if count <= 2:
        for pt in center_points_cur_frame:
            for pt2 in center_points_prev_frame:
                distance = math.hypot(pt2[0] - pt[0], pt2[1] -
pt[1])

                if distance < 20:
                    tracking_objects[track_id] = pt
                    track_id += 1
    else:

        tracking_objects_copy = tracking_objects.copy()
        center_points_cur_frame_copy =
```

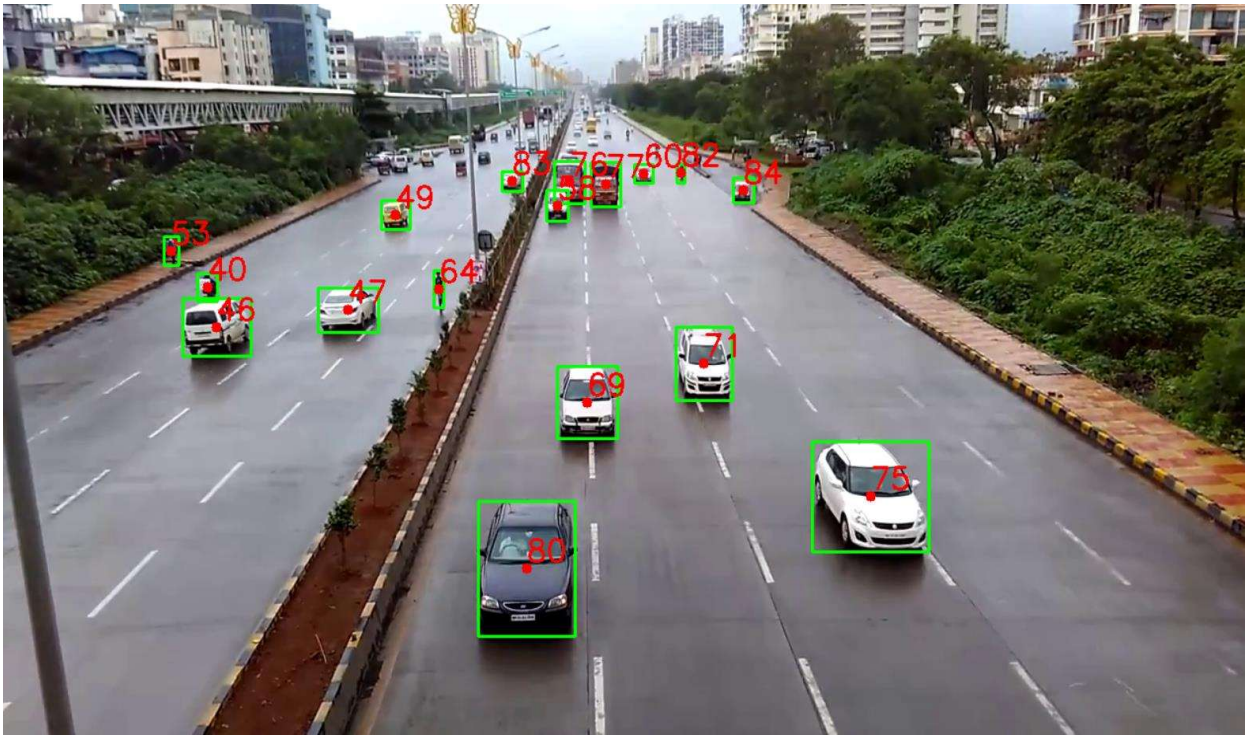
```
center_points_cur_frame.copy()

        for object_id, pt2 in tracking_objects_copy.items():
            object_exists = False
            for pt in center_points_cur_frame_copy:
                distance = math.hypot(pt2[0] - pt[0], pt2[1] -
pt[1])

                # Update IDs position
                if distance < 20:
                    tracking_objects[object_id] = pt
                    object_exists = True
                    if pt in center_points_cur_frame:
                        center_points_cur_frame.remove(pt)
                    continue

            # Remove IDs lost
            if not object_exists:
                tracking_objects.pop(object_id)

        # Add new IDs found
        for pt in center_points_cur_frame:
            tracking_objects[track_id] = pt
            track_id += 1

    for object_id, pt in tracking_objects.items():
        cv2.circle(frame, pt, 5, (0, 0, 255), -1)
        cv2.putText(frame, str(object_id), (pt[0], pt[1] - 7),
```

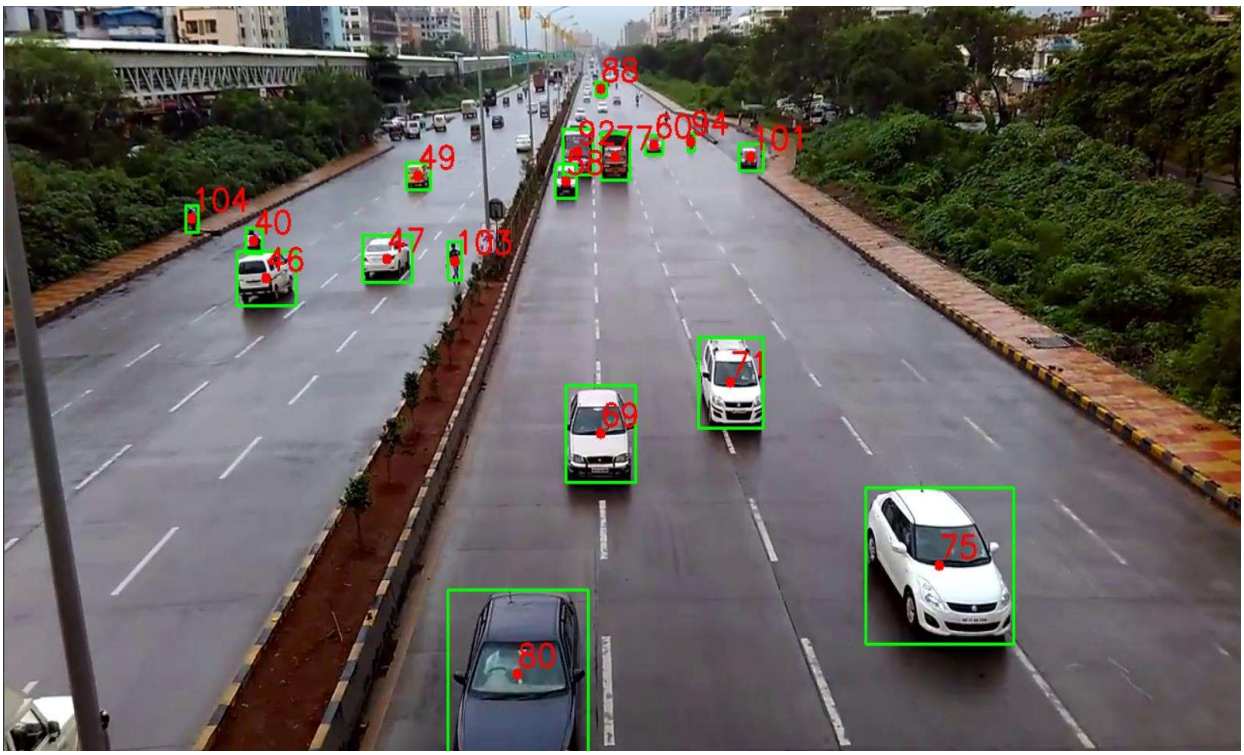
```
                0, 1, (0, 0, 255), 2)

                print("Tracking objects")
                print(tracking_objects)


                print("CUR FRAME LEFT PTS")
                print(center_points_cur_frame)


                cv2.imshow("Frame", frame)

                # Make a copy of the points
                center_points_prev_frame =
            center_points_cur_frame.copy()

                key = cv2.waitKey(1)
                if key == 27:
                    break

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

- **OUTPUT**






83
77
77
60
82
84
49
53
40
46
47
64
71
69
75
80


88
77
77
60
94
101
49
104
40
46
47
103
71
69
75
80

Reference

www.pythaon.com
www.pythonpoint.com
www.pysource.com
www.livecodestream.dev