PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

DEPARTMENT OF AUTOMOBILE ENGINEERING

BACHELOR OF ENGINEERING

19A612 - INNOVATION PRACTICES

TOPIC – OBJECT DETECTION USING YOLO

TEAM MEMBERS

20A218-MUTHU VIGNESH .M

20A224-PRIYADHARSHAN .AVM

20A226-RAGURAMAN

21A453-REVANTH

CONTENTS

ACKNOWLEDGEMENT

Presentation, inspiration and motivation have always played a key role in the success of any venture. We would like to express our sincere thanks to the staffs who supported us in each and every daily life to design and implement this project

We would like to express our sincere gratitude to **Dr. K. PRAKASAN**, Principal, PSG College of Technology, for giving us the opportunity to do our project with various facilities and infrastructure, without which the success of the project would not have been possible.

We extend our heartfelt thanks to **Dr.NEELAKRISHNAN.S** Head of, Department of Automobile Engineering, PSG College of Technology, for his unfailing support throughout this project.

We would also wish to express our sincere thanks to our Program Coordinator **Dr.P.KARHIKEYAN,** Assistant Professor, Department of Automobile Engineering, PSG College of Technology, whose constant support and everlasting enthusiasm made it possible to complete the project within the time.

Last but not least, we would also like to thank our Tutor **Dr.V.M .MURUGESAN,** Associate Professor, Department of Automobile Engineering, who was always there to help us and played a major role in the completion of the project and we wish to thank his for his enduring guidance and priceless advice throughout this project work.

SYNOPSIS

Object detection is a computer vision task that has become an integral part of many consumer applications today such as surveillance and security systems, mobile text recognition, and diagnosing diseases from MRI/CT scans.

Object detection is also one of the critical components to support autonomous driving. Autonomous vehicles rely on the perception of their surroundings to ensure safe and robust driving performance.

This perception system uses object detection algorithms to accurately determine objects such as pedestrians, vehicles, traffic signs, and barriers in the vehicle's vicinity.

Deep learning-based object detectors play a vital role in finding and localizing these objects in real-time.

Autonomous vehicles (AVs) have received immense attention in recent years, in large part due to their potential to improve driving comfort and reduce injuries from vehicle crashes.

It has been reported that more than 36,000 people died in 2019 due to fatal accidents on U.S. roadways . AVs can eliminate human error and distracted driving that is responsible for 94% of these accidents .

By using sensors such as cameras, lidars, and radars to perceive their surroundings, AVs can detect objects in their vicinity and make real-time decisions to avoid collisions and ensure safe driving behavior

INTRODUCTION

1.1. OVERVIEW

YOLO (You Only Look Once) is an object detection model that is widely used in the field of computer vision.

YOLOv3 is the third version of this model that has been developed to provide improved object detection capabilities.

YOLOv3 is a real-time object detection algorithm that is designed to work on low-powered devices.

OpenCV and Darknet are two popular frameworks that can be used to implement YOLOv3. In this report, we will explore how YOLOv3 can be used with OpenCV and Darknet.

1.2. MAIN OBJECTIVE OF THE PROJECT:

To Understand , Improve and Implement an Object Detection Algorithm YOLO version 3

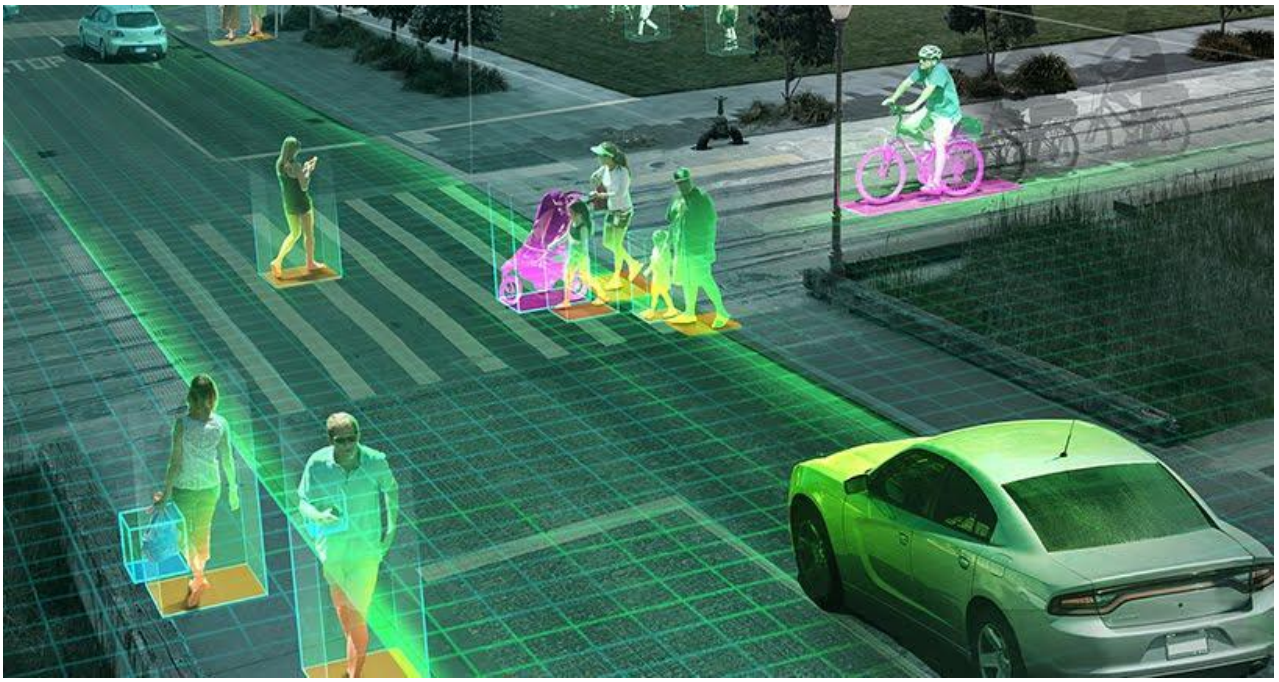
1.3. OPENCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly for real-time computer vision.

Originally developed by Intel, it was later supported by Willow Garage, then Itseez (which was later acquired by Intel).

The library is cross-platform and licensed as free and open-source software under Apache License 2.

Starting in 2011, OpenCV features GPU acceleration for real-time operations



PROCEDURE

Following things are needed to execute the code we will be writing.

- Python 3
- Numpy
- OpenCV Python bindings

Numpy:

pip install numpy

This should install numpy. Make sure pip is linked to Python 3.x

The script requires four input arguments.

- input image
- YOLO config file
- pre-trained YOLO weights
- text file containing class names

Read the input image and get its width and height.

Read the text file containing class names in human readable form and extract the class names to a list.

Generate different colors for different classes to draw bounding boxes.

net = cv2.dnn.readNet(args.weights, args.config)

blob = cv2.dnn.blobFromImage(image, scale, (Width,Height), (0,0,0), True, crop=False)net.setInput(blob)

Above lines prepare the input image to run through the deep neural network.

cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)
cv2.putText(img, label, (x-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

Generally in a sequential CNN network there will be only one output layer at the end. In the YOLO v3 architecture we are using there are multiple output layers giving out predictions. get_output_layers() function gives the names of

the output layers. An output layer is not connected to any next layer. draw_bounding_box() function draws a rectangle over the given predicted region and writes the class name over the box. If needed, we can write the confidence value too.
If we don't specify the output layer names, by default, it will return the predictions only from the final output layer. Any intermediate output layer will be ignored. We need to go through each detection from each output layer to get the class id, confidence and bounding box corners and more importantly ignore the weak detections (detections with low confidence value).

CODE:

```
import cv2

import numpy as np

import time

#Load YOLO

net = cv2.dnn.readNet("yolov3320.weights","yolov3320.cfg") # Original yolov3

#net = cv2.dnn.readNet("yolov3-tiny.weights","yolov3-tiny.cfg") #Tiny Yolo

classes = []

with open("classes.txt","r") as f:

    classes = [line.strip() for line in f.readlines()]

print(classes)

['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', 'truck', 'boat', 'traffic
light', 'fire hydrant', 'stop sign', 'parking meter']

layer_names = net.getLayerNames()

outputlayers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

outputlayers

['yolo_82', 'yolo_94', 'yolo_106']

colors= np.random.uniform(0,255,size=(len(classes),3))


#loading image

#cap=cv2.VideoCapture("22.mp4") #0 for 1st webcam

cap = cv2.VideoCapture("bridge_1.mp4")

font = cv2.FONT_HERSHEY_PLAIN

starting_time= time.time()

frame_id = 0


while True:

    _,frame= cap.read() #

    frame_id+=1
```

```
    height,width,channels = frame.shape
    #detecting objects
    blob = cv2.dnn.blobFromImage(frame,0.00392,(320,320),(0,0,0),True,crop=False) #reduce 416 to 320
```

```
net.setInput(blob)
    outs = net.forward(outputlayers)
    #print(outs[1])
```

```
    #Showing info on screen/ get confidence score of algorithm in detecting an object in blob
    class_ids=[]
    confidences=[]
    boxes=[]
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                #onject detected
                center_x= int(detection[0]*width)
                center_y= int(detection[1]*height)
                w = int(detection[2]*width)
                h = int(detection[3]*height)
```

```
                #cv2.circle(img,(center_x,center_y),10,(0,255,0),2)
                #rectangle co-ordinaters
                x=int(center_x - w/2)
                y=int(center_y - h/2)
                #cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
```

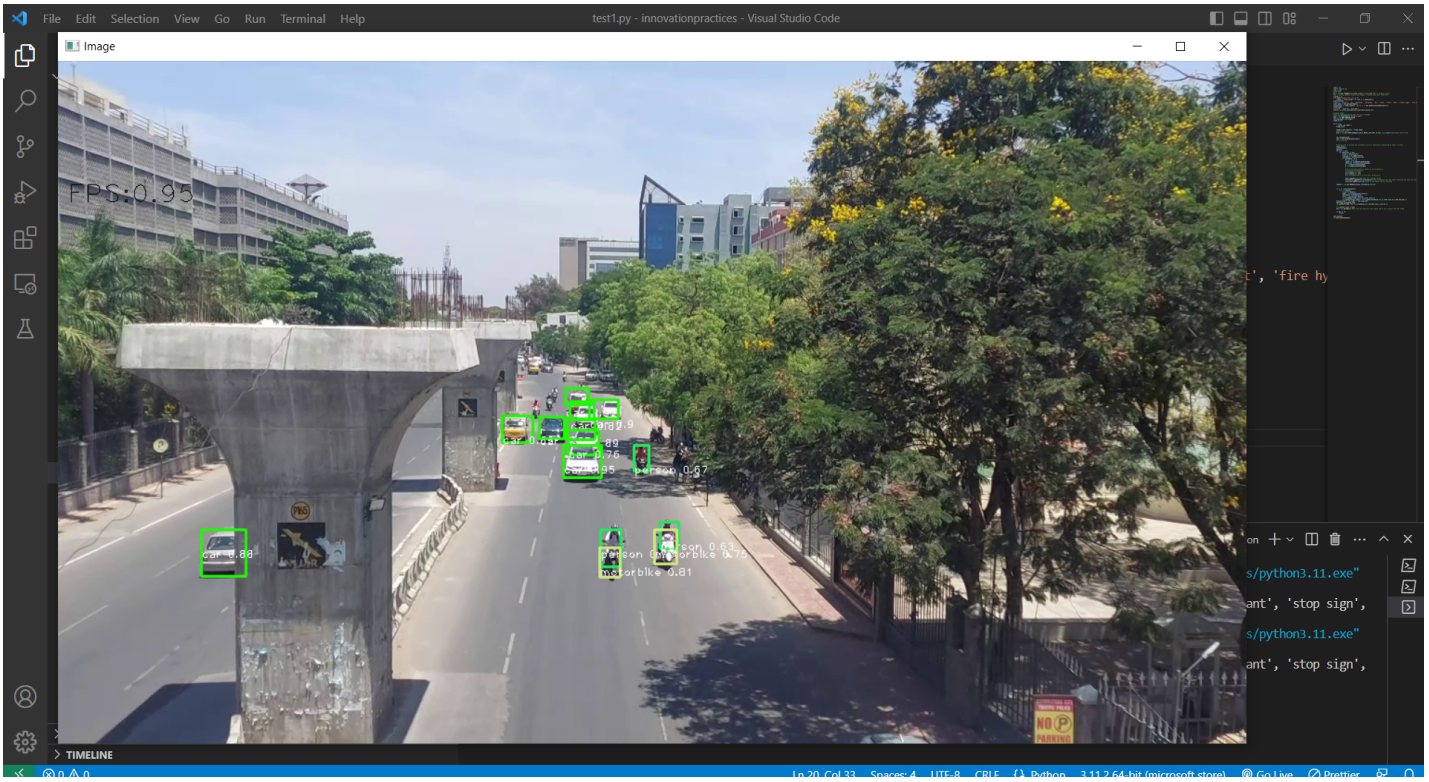
```
                boxes.append([x,y,w,h]) #put all rectangle areas
                confidences.append(float(confidence)) #how confidence was that object detected and
show that percentage
                class_ids.append(class_id) #name of the object tha was detected
```

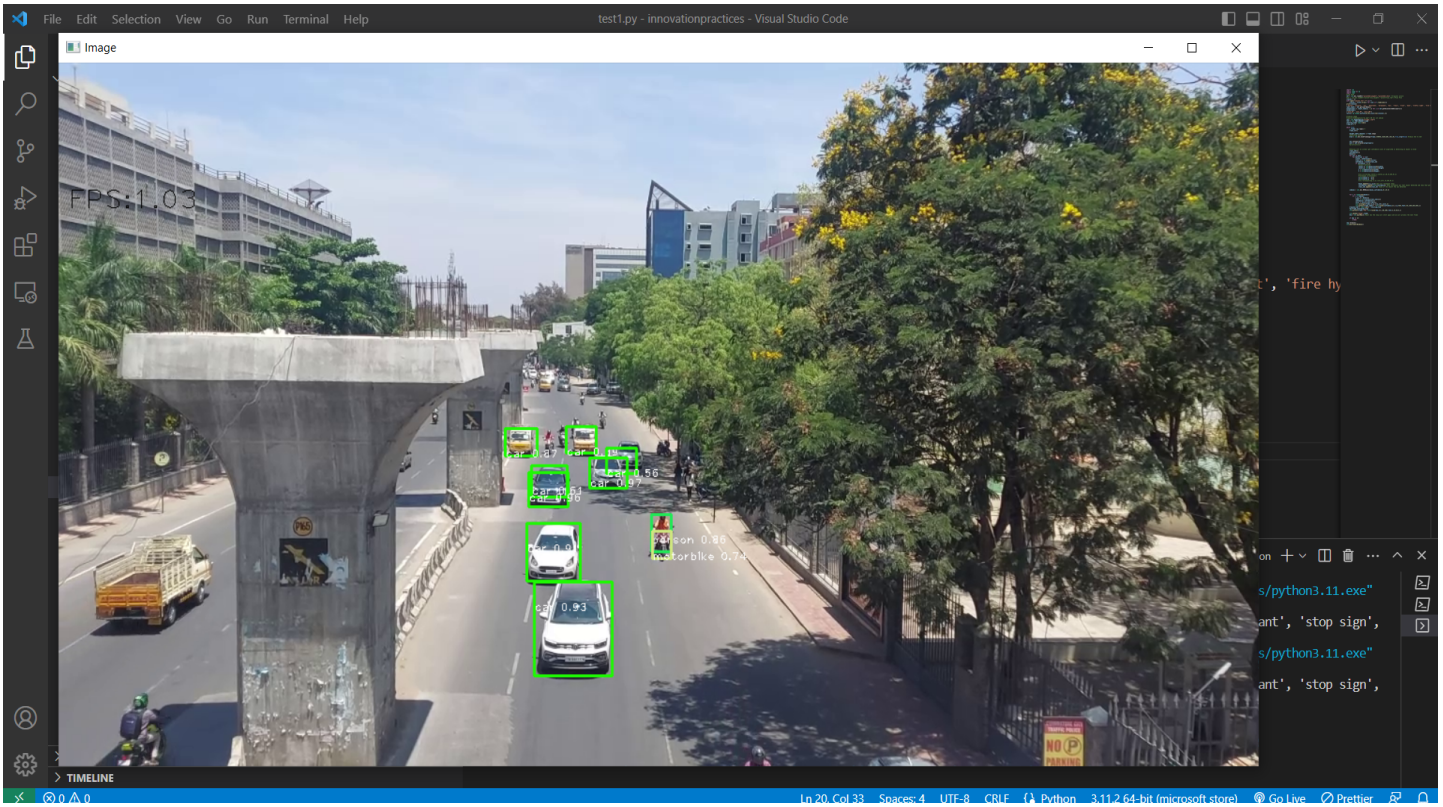
```
    indexes = cv2.dnn.NMSBoxes(boxes,confidences,0.4,0.6)
```

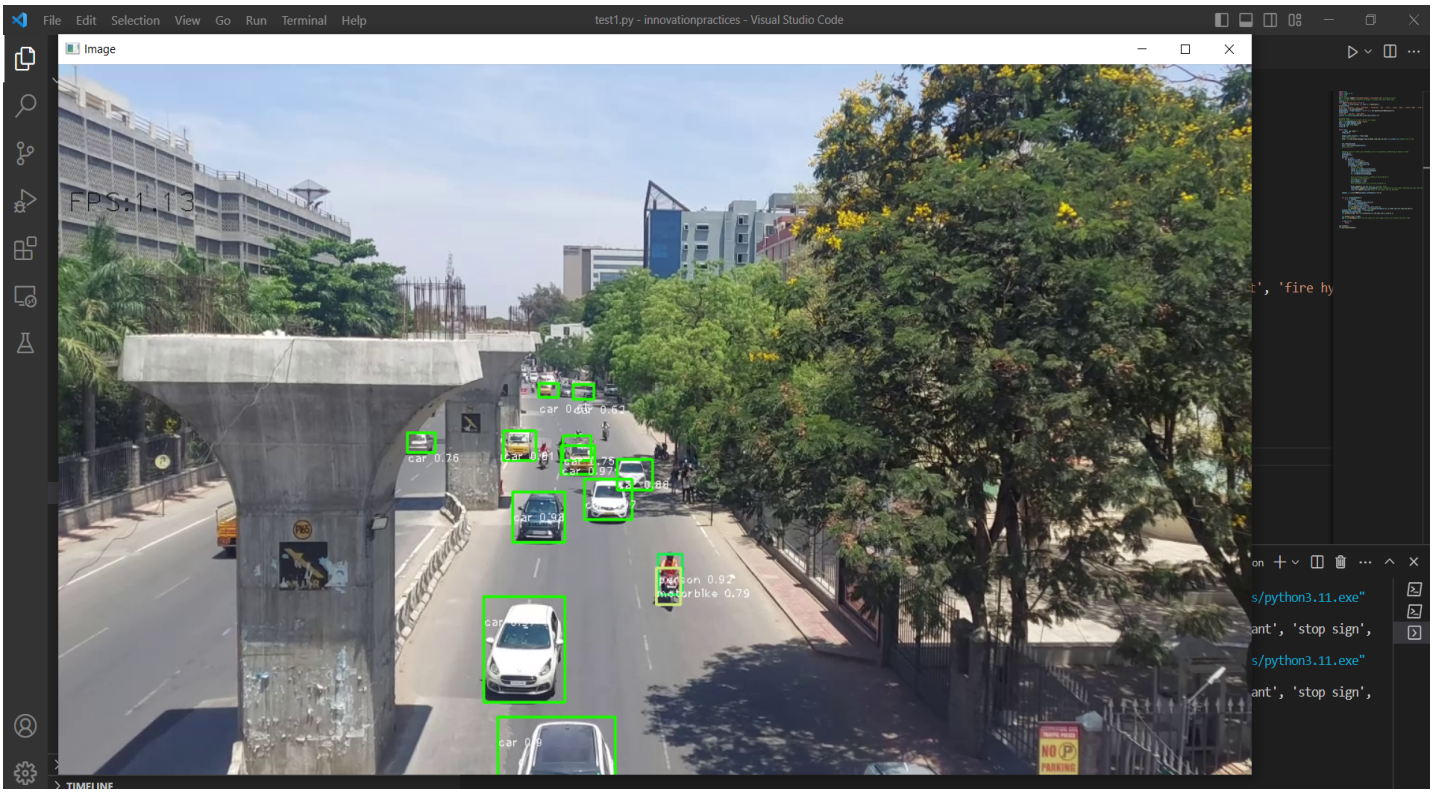
```
    for i in range(len(boxes)):
        if i in indexes:
            x,y,w,h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence= confidences[i]
            color = colors[class_ids[i]]
            cv2.rectangle(frame,(x,y),(x+w,y+h),color,2)
            cv2.putText(frame,label+" "+str(round(confidence,2)),(x,y+30),font,0.8,(255,255,255),1)
    elapsed_time = time.time() - starting_time
    fps=frame_id/elapsed_time
    cv2.putText(frame,"FPS:"+str(round(fps,2)),(10,150),font,2,(0,0,0),1)


    cv2.imshow("Image",frame)
    key = cv2.waitKey(1) #wait 1ms the loop will start again and we will process the next frame


    if key == 27:
        break;


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

CONSTRAINTS:
CPU USED - AMD RYZEN 5 3550H
GPU USED -NVIDIA GEFORCE RTX 1650
SOURCE CODE EDITOR-VISUAL STUDIO CODE
SCOPE FOR IMPROVEMENT IN FPS :
CUDA BASED OPENCV
POWERFUL CPU
POWERFUL GPU (RTX TITAN)

RESULTS :


File Edit Selection View Go Run Terminal Help
test1.py - innovationpractices - Visual Studio Code
Image
FPS:0.95
TIMELINE


File Edit Selection View Go Run Terminal Help
test1.py - innovationpractices - Visual Studio Code
Image
FPS:1.03
TIMELINE


File Edit Selection View Go Run Terminal Help
test1.py - innovationpractices - Visual Studio Code
Image
FPS:1.13
car 0.76
car 0.97
motorbike 0.79
NO PARKING
TIMELINE
s/python3.11.exe"
ant', 'stop sign',
s/python3.11.exe"
ant', 'stop sign',