VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE

Matunga, Mumbai-400 019

Autonomous Institute affiliated to University of Mumbai

EXAMINATION	End Semester Examination May /June 2022	DATE OF EXAM	07/05/2022
SEMESTER & PROGRAM	Sem-VI, T.Y.B. Tech. (Electronics Engineering)	TIME	3:00 pm to :500 pm
TIME ALLOWED	**2:00 HRS.**	MARKS	**60**
COURSE NAME – (CODE)	Image and Video Processing Lab (R4EC3101P)		

IVP LAB EXAM SUBMISSION:

Parth Ravindra Deshpande

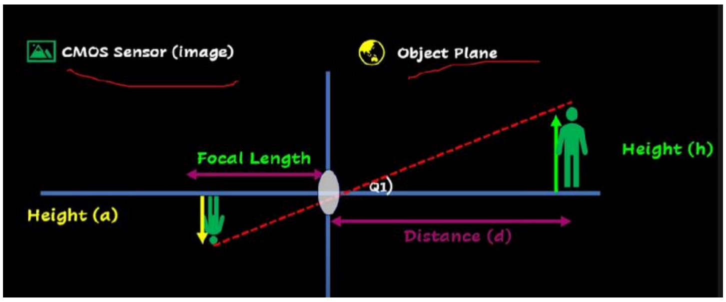
191060022

Electronics B.Tech TY

Problem Statement:

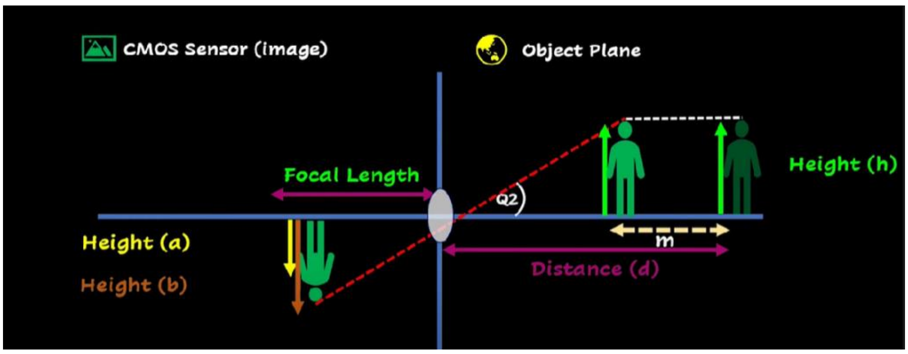
Object Distance Measurement using OpenCV-python:

The main idea of calculating distance of the object is as below, we can calculate height of object from simple optics and trigonometric identities.



For object at initial position,

$$\frac{a}{f} = \tan\theta_1 = \frac{h}{d}$$



For object at position which is at m distance from initial position,

$$\frac{b}{f} = \tan\theta_2 = \frac{h}{d-m}$$

$$\frac{a}{f} = \tan\theta_1 = \frac{h}{d} \longrightarrow \text{Equation No: 1}$$

$$\frac{b}{f} = \tan\theta_2 = \frac{h}{d-m} \longrightarrow \text{Equation No: 2}$$

Divide Equation 1 with Equation 2 We get:

$$\frac{a}{b} = \frac{h}{d} \times \frac{d-m}{h}$$

By adjusting the a and b values we can get value of d which is the distance of the object,

$$\frac{a}{b} = \frac{d-m}{d} = 1 - \frac{m}{d}$$

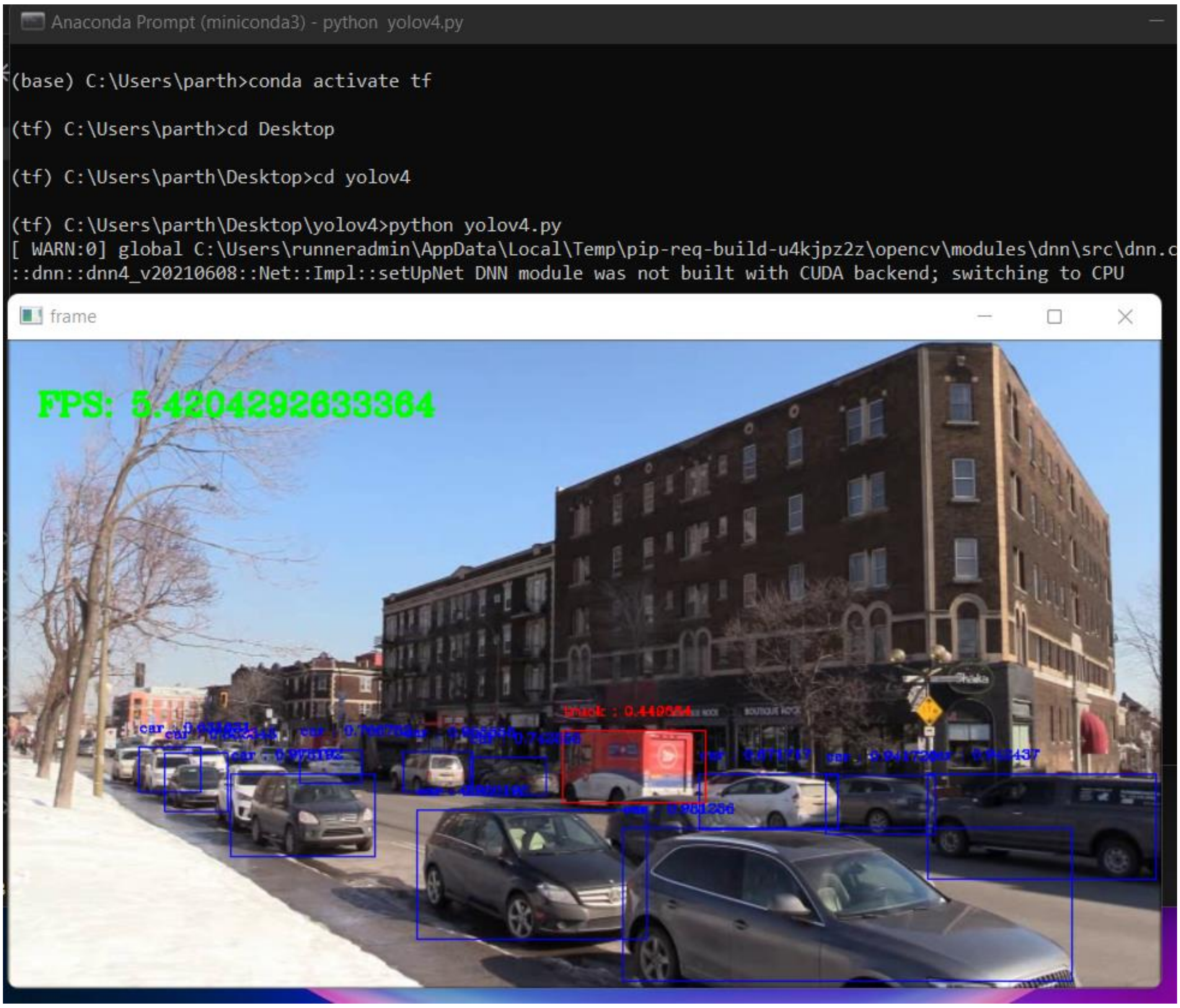
$$\frac{m}{d} = 1 - \frac{a}{b}$$

$$d = \frac{m}{1 - \frac{a}{b}}$$

I’ve used above formula to calculate the distance of object using OpenCV. For this, first I’ve used YoloV4 for detecting the object. YoloV4 is a pretrained dnn model which can detect upto 80 types of objects. Yolo(You only look once) is an object detection technique in computer vision.

YoloV4 object detection:

Basic object detection by using YoloV4:



Now, let’s calculate the distance for each object by using bounding boxes created using object detection:

First, I’ve taken *reference distance* which is the distance at which all reference images are taken. *Reference images* are the images which are used as a reference to the objects for which we are going to calculate distances. For simplicity I’ve only used 5 classes for distance calculation. These classes are [person,mobile,bottle,laptop,cup] . I’ve taken these classes because these are the commonly found items in offices, schools, colleges and other closed environments.

I’ve used YoloV4 for detecting objects so all the configs of this pretrained model is stored in **yolov4-tiny.cfg** file and weights for the DNN model is stored in **yolov4-tiny.weights** file.

NMS threshold is Non-Maximal Suppression (NMS) : YOLO uses Non-Maximal Suppression (NMS) to only keep the best bounding box. The first step in NMS is to remove all the predicted bounding boxes that have a detection probability that is less than a given NMS threshold.

```
import cv2 as cv
import numpy as np

# Distance constants
KNOWN_DISTANCE = 45 #INCHES
PERSON_WIDTH = 16 #INCHES
MOBILE_WIDTH = 3.0 #INCHES
BOTTLE_WIDTH = 3.0 #INCHES
LAPTOP_WIDTH = 12 #INCHES
CUP_WIDTH = 3 #INCHES


# Object detector constant
CONFIDENCE_THRESHOLD = 0.4
NMS_THRESHOLD = 0.3


# colors for object detected
COLORS = [(255,0,0),(255,0,255),(0, 255, 255), (255, 255, 0), (0, 255, 0),
(255, 0, 0)]
GREEN =(0,255,0)
BLACK =(0,0,0)
# defining fonts
FONTS = cv.FONT_HERSHEY_COMPLEX
```

KNOWN_DISTANCE is the distance at which all reference images are taken. **PERSON_WIDTH, MOBILE_WIDTH, BOTTLE_WIDTH, LAPTOP_WIDTH, CUP_WIDTH** are the widths of these object in real world.

Colours are defined for easier use while creating bounding boxes.

Now next part,

```
# getting class names from classes.txt file
class_names = []
with open("classes.txt", "r") as f:
    class_names = [cname.strip() for cname in f.readlines()]

#  setttng up opencv net
mynet = cv.dnn.readNet('yolov4-tiny.weights', 'yolov4-tiny.cfg')
model = cv.dnn_DetectionModel(mynet)
model.setInputParams(size=(416, 416), scale=1/255, swapRB=True)  #swapRB swpas
the R and B channel
```

Here, First I've added all class names from **classes.txt** into **class_names []** List**.**

Cv2.dnn is used for loading the pretrained model with configuration and weights file.

Now next part,

```
# object detector funciton /method
def object_detector(image):
    classes, scores, boxes = model.detect(image, CONFIDENCE_THRESHOLD,
NMS_THRESHOLD)
    # creating empty list to add objects data
    data_list =[]
    for (classid, score, box) in zip(classes, scores, boxes):
        # define color of each, object based on its class id
        color = COLORS[int(classid) % len(COLORS)]

        label = "%s : %f" % (class_names[classid[0]], score)

        # draw rectangle on and label on object
        cv.rectangle(image, box, color, 2)
        cv.putText(image, label, (box[0], box[1]-14), FONTS, 0.5, color, 2)

        # getting the data
        # 1: class name  2: object width in pixels, 3: position where have to
draw text(distance)
        if classid ==0: # person class id
            data_list.append([class_names[classid[0]], box[2], (box[0],
box[1]-2)])
        elif classid ==67: # this is for mobiles
            data_list.append([class_names[classid[0]], box[2], (box[0],
box[1]-2)])
        elif classid ==39: # this is for bottle
            data_list.append([class_names[classid[0]], box[2], (box[0],
box[1]-2)])
        elif classid ==63: # this is for laptop
            data_list.append([class_names[classid[0]], box[2], (box[0],
box[1]-2)])
        elif classid ==41: # this is for cup
            data_list.append([class_names[classid[0]], box[2], (box[0],
box[1]-2)])
        # if you want inclulde more classes then you have to simply add more
[elif] statements here
        # returning list containing the object data.
    return data_list
```

This function returns **data_list** which is calculated using YoloV4,

[class name , object width in pixels , position where have to draw text(distance)]

```python
def focal_length_finder (measured_distance, real_width, width_in_rf):
    focal_length = (width_in_rf * measured_distance) / real_width
    return focal_length


# distance finder function
def distance_finder(focal_length, real_object_width, width_in_frmae):
    distance = (real_object_width * focal_length) / width_in_frmae
    return distance


# reading the reference image from dir
ref_person = cv.imread('ReferenceImages/person.png')
ref_mobile = cv.imread('ReferenceImages/image4.png')
ref_bottle = cv.imread('ReferenceImages/bottle.png')
ref_laptop = cv.imread('ReferenceImages/laptop.png')
ref_cup = cv.imread('ReferenceImages/cup.png')


mobile_data = object_detector(ref_mobile)
mobile_width_in_rf = mobile_data[1][1]

person_data = object_detector(ref_person)
person_width_in_rf = person_data[0][1]

bottle_data = object_detector(ref_bottle)
bottle_width_in_rf = bottle_data[0][1]

laptop_data = object_detector(ref_laptop)
laptop_width_in_rf = laptop_data[0][1]

cup_data = object_detector(ref_cup)
cup_width_in_rf = cup_data[0][1]


# finding focal length
focal_person = focal_length_finder(KNOWN_DISTANCE, PERSON_WIDTH,
person_width_in_rf)
focal_mobile = focal_length_finder(KNOWN_DISTANCE, MOBILE_WIDTH,
mobile_width_in_rf)
focal_bottle = focal_length_finder(KNOWN_DISTANCE, BOTTLE_WIDTH,
bottle_width_in_rf)
focal_laptop = focal_length_finder(KNOWN_DISTANCE, LAPTOP_WIDTH,
laptop_width_in_rf)
focal_cup = focal_length_finder(KNOWN_DISTANCE, CUP_WIDTH, cup_width_in_rf)
```

In this part of code, **focal_length_finder** is used to calculate focal distances for our reference images. Pixel width is also captured from the d[i][1] as Data_list stores the retured value from object_detector function. Where i is the ith object detected in the image.

```
cap = cv.VideoCapture(1)
while True:
    ret, frame = cap.read()

    data = object_detector(frame)
    for d in data:
        if d[0] =='person':
            distance = distance_finder(focal_person, PERSON_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='cell phone':
            distance = distance_finder (focal_mobile, MOBILE_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='bottle':
            distance = distance_finder (focal_bottle, BOTTLE_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='laptop':
            distance = distance_finder (focal_laptop, LAPTOP_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='cup':
            distance = distance_finder (focal_laptop, LAPTOP_WIDTH, d[1])
            x, y = d[2]
        cv.rectangle(frame, (x, y-3), (x+150, y+23),BLACK,-1 )
        cv.putText(frame, f'Dis: {round(distance,2)} inch', (x+5,y+13), FONTS,
0.48, GREEN, 2)

    cv.imshow('frame',frame)
    key = cv.waitKey(1)
    if key ==ord('q'):
        break


cv.destroyAllWindows()
cap.release()
```

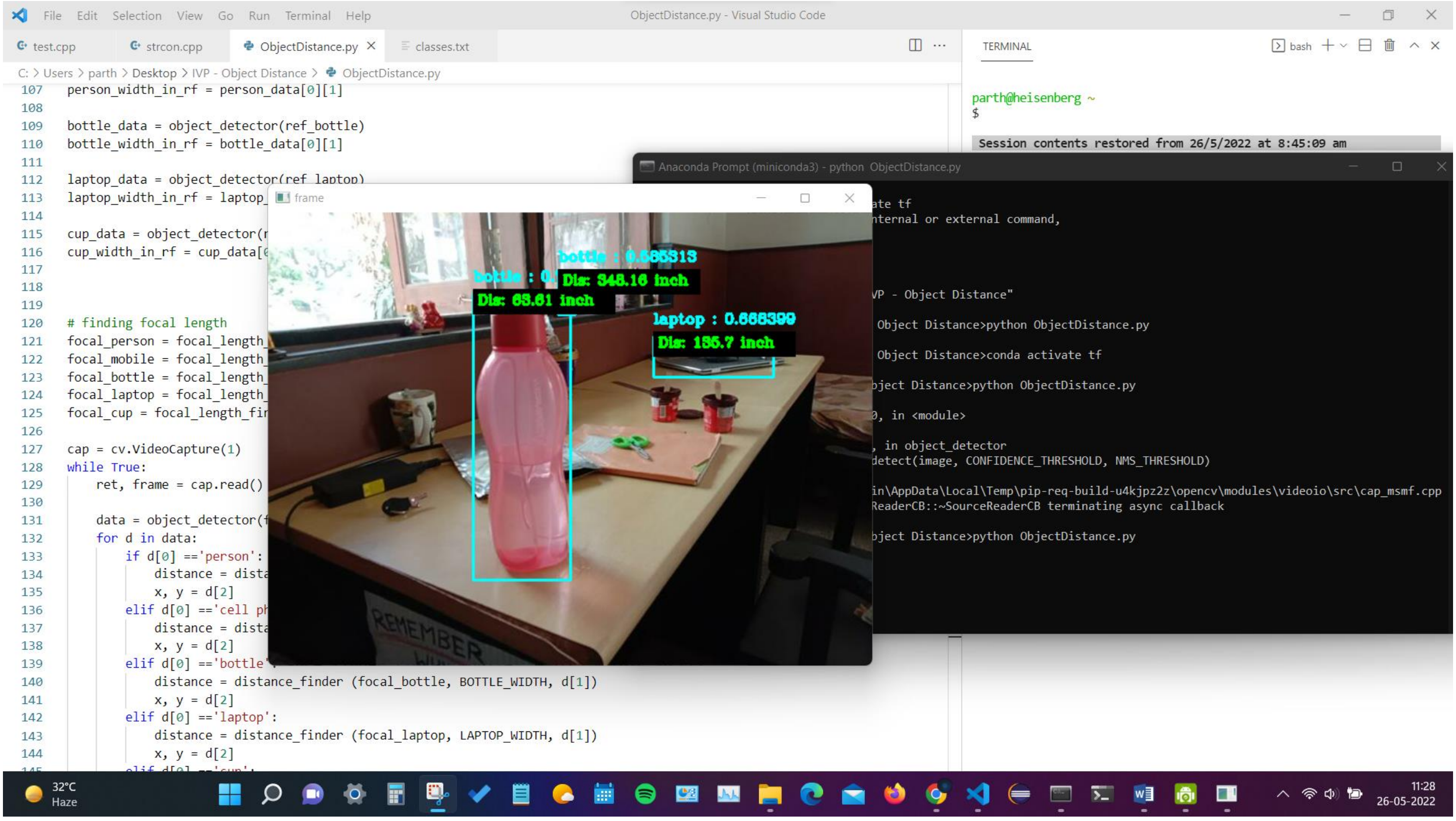
Cv2.VideoCapture(1) is used to capture video frames from camera device 1. I've used android device as a wifi camera.

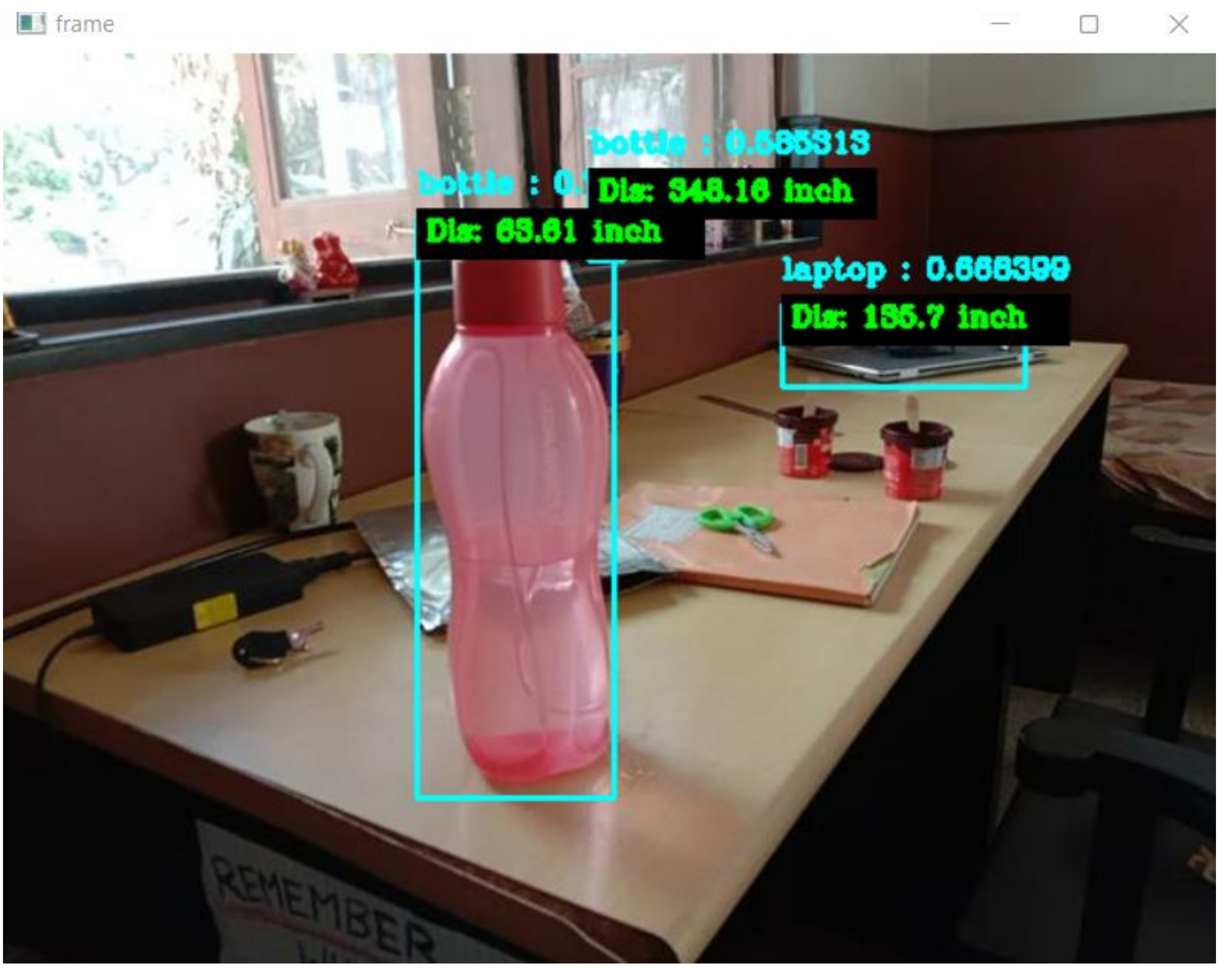
Here, distance_finder function is used which was defined in the previous code. For bounding boxes x,y is captured from return value of object_detector function which is stored at index 2 (i.e. d[2]) .While TRUE loop runs all the time and detects objects and calculates their distances

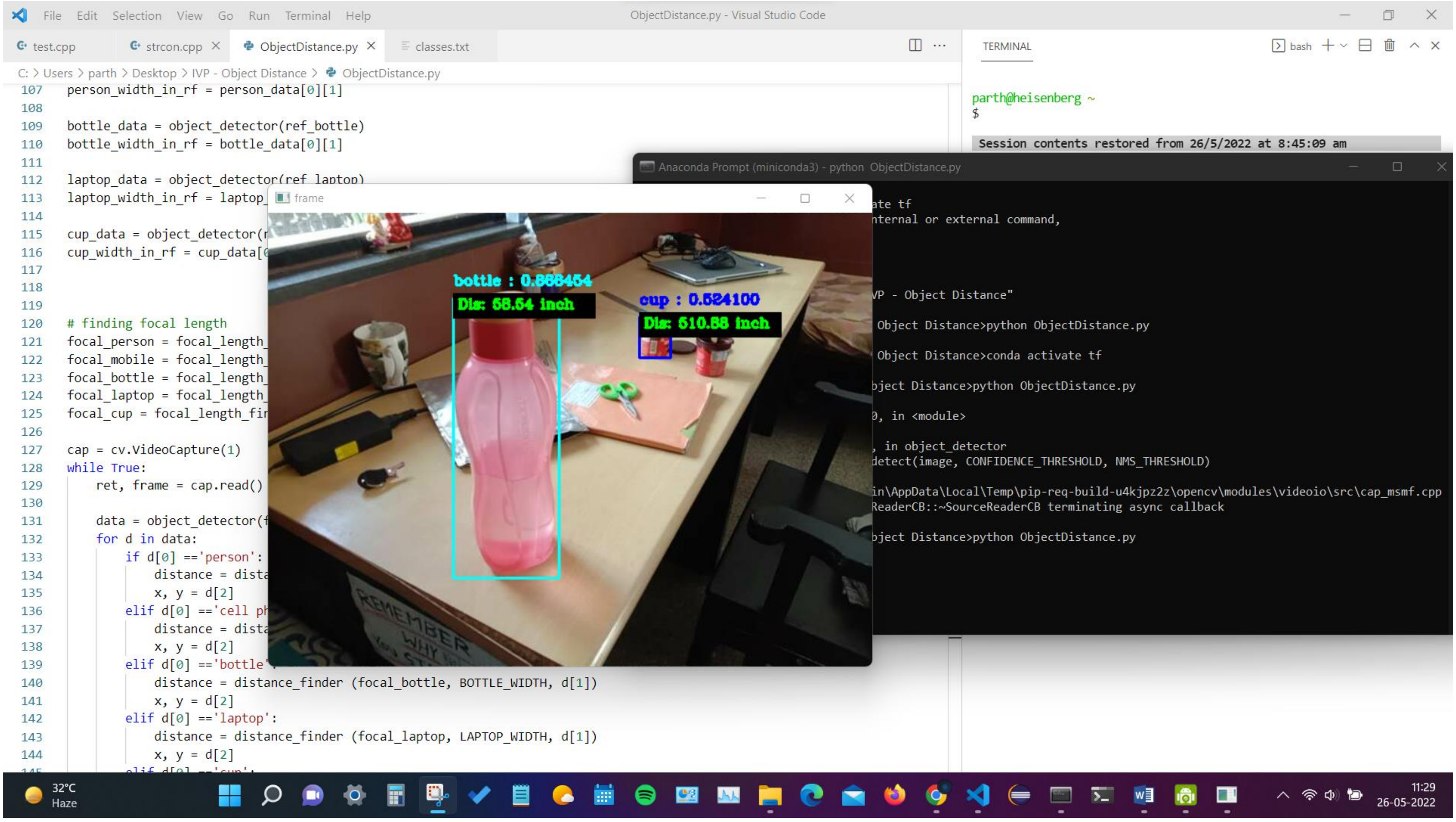
---

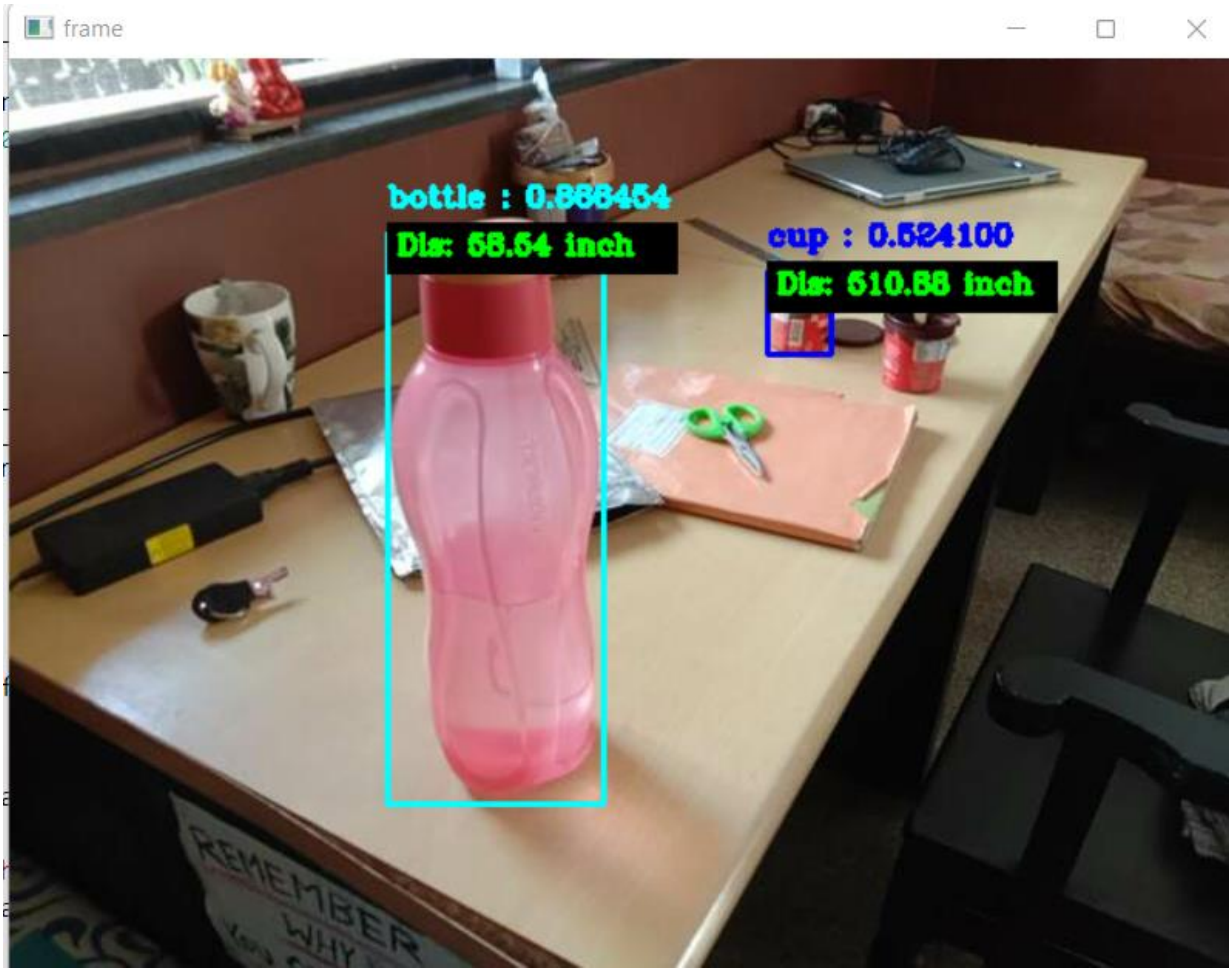
Results & Observations:

Bottles, laptop, cup, human, human with mobile in hand distances are calculated and printed with bounded boxes. Below are the results from the real world video capture.

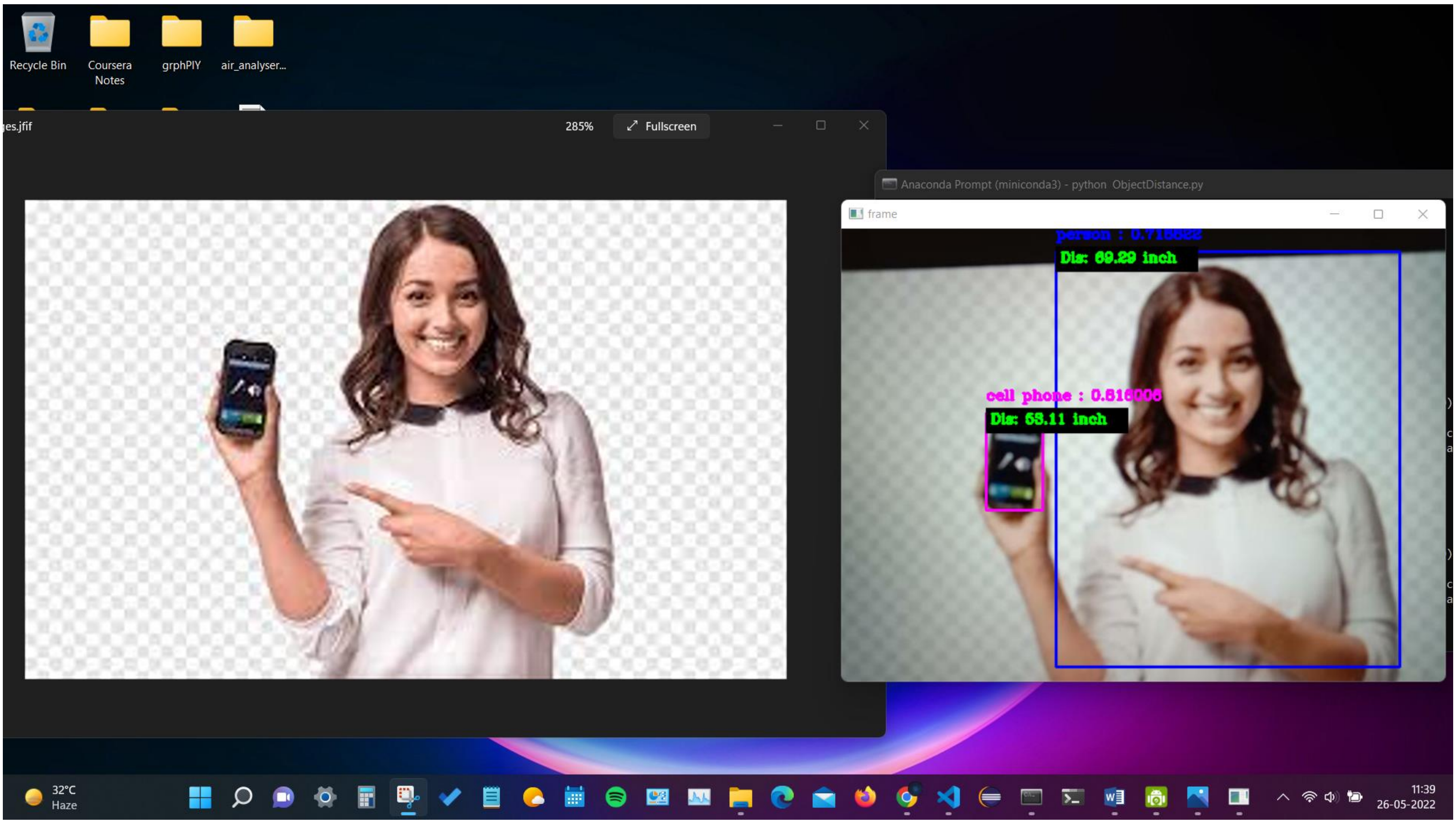


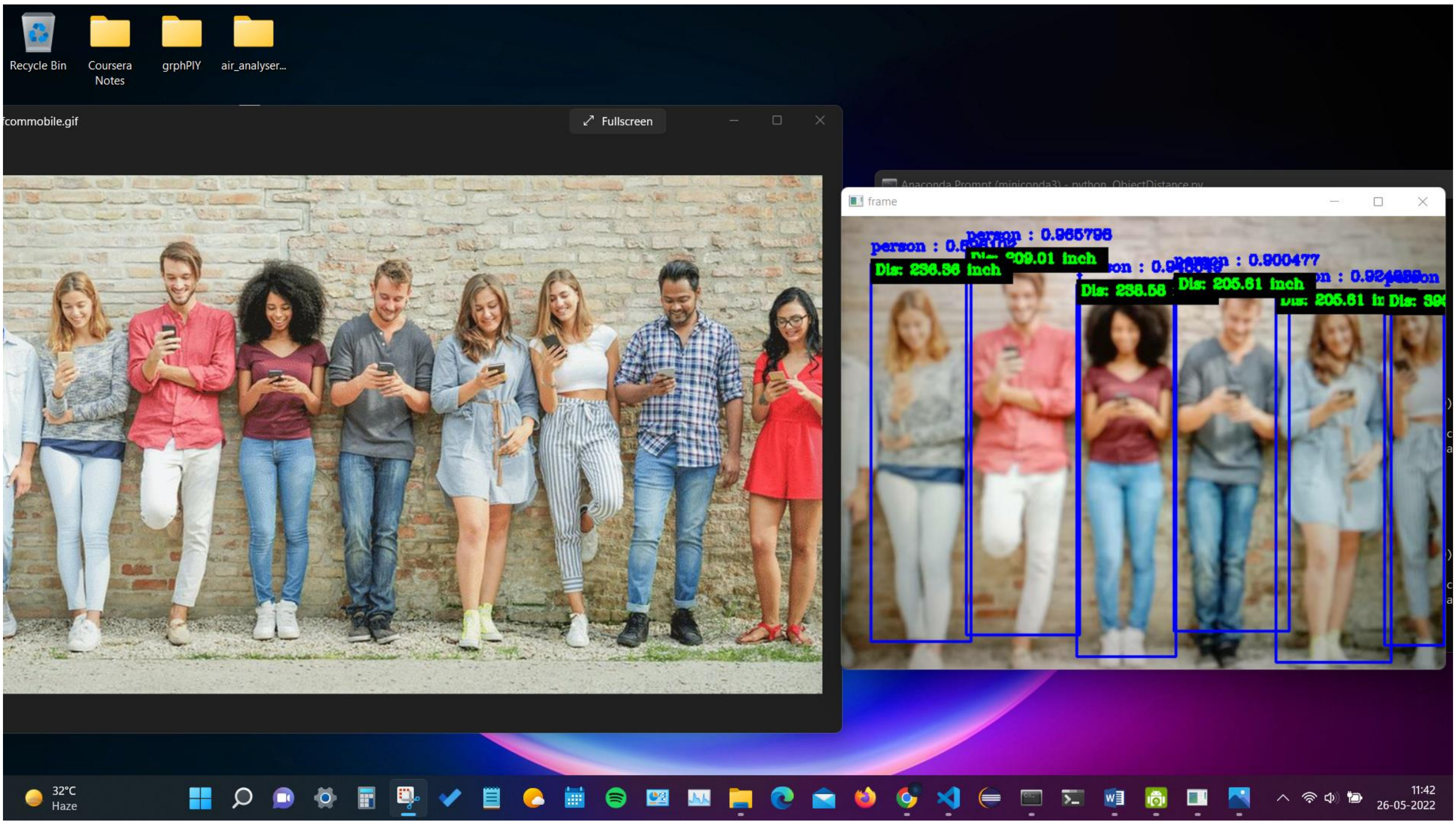



ObjectDistance.py - Visual Studio Code
test.cpp
strcon.cpp
ObjectDistance.py
classes.txt
C: > Users > parth > Desktop > IVP - Object Distance > ObjectDistance.py
person_width_in_rf = person_data[0][1]
bottle_data = object_detector(ref_bottle)
bottle_width_in_rf = bottle_data[0][1]
laptop_data = object_detector(ref_laptop)
# finding focal length
cap = cv.VideoCapture(1)
while True:
ret, frame = cap.read()
for d in data:
if d[0] =='person':
elif d[0] =='bottle':
distance = distance_finder (focal_bottle, BOTTLE_WIDTH, d[1])
x, y = d[2]
elif d[0] =='laptop':
distance = distance_finder (focal_laptop, LAPTOP_WIDTH, d[1])
TERMINAL
parth@heisenberg ~
Session contents restored from 26/5/2022 at 8:45:09 am
Anaconda Prompt (miniconda3) - python ObjectDistance.py
Object Distance>python ObjectDistance.py
Object Distance>conda activate tf
bject Distance>python ObjectDistance.py
detect(image, CONFIDENCE_THRESHOLD, NMS_THRESHOLD)
ReaderCB::~SourceReaderCB terminating async callback
frame
bottle : 0.888454
Dis: 58.64 inch
cup : 0.524100
Dis: 510.88 inch
32°C
Haze
11:29
26-05-2022


frame
bottle : 0.888454
Dis: 58.64 inch
cup : 0.524100
Dis: 510.88 inch

After these real-world objects, I've also tried this with Images considering the image POV as the real time camera angle. ***Image source: Google Images***





Conclusion:

This is how we can use OpenCV, Python and pretrained dnn models like YoloV4 to detect objects and their distances.