PSG INSTITUTE OF TECHNOLOGY AND APPLIED RESEARCH

DEPARTMENT OF ELECTRONICS AND COMMUNCIATION ENGINEERING

MINI PROJECT

A – EYE

*(See the Deaf, Hear the Blind)*

By

Athul Krishna R (715519106007)

Colin Rolance D (715519106008)

Shyam Vikram S (715519106046)

**Program (Object Detection):**

import cv2

import matplotlib.pyplot as plt

import pyttsx3

engine = pyttsx3.init()

import time

config\_file = 'ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt'

frozen\_model = 'frozen\_inference\_graph.pb'

model = cv2.dnn\_DetectionModel(frozen\_model,config\_file)

classLabels = []

file\_name = 'Labels.txt'

with open(file\_name,'rt') as fpt:

classLabels = fpt.read().rstrip('\n').split('\n')

print(classLabels)

print(len(classLabels))

model.setInputSize(320,320)

model.setInputScale(1.0/127.5)

model.setInputMean((127.5,127.5,127.5))

model.setInputSwapRB(True)

img = cv2.imread('dog.jpg')

plt.imshow(img)

plt.imshow(cv2.cvtColor(img,cv2.COLOR\_BGR2RGB))

ClassIndex, confidence, bbox = model.detect(img,confThreshold=0.5)

print(ClassIndex)

font\_scale = 3

font = cv2.FONT\_HERSHEY\_PLAIN

for ClassInd, conf, boxes in zip(ClassIndex.flatten(), confidence.flatten(), bbox):

cv2.rectangle(img,boxes,(255,0,0),2)

cv2.putText(img,classLabels[ClassInd-1], (boxes[0]+10,boxes[1]+40), font, fontScale=font\_scale, color=(0,255,0), thickness=3)

print(classLabels[ClassInd-1])

engine.say(classLabels[ClassInd-1])

engine.runAndWait()

plt.imshow(cv2.cvtColor(img,cv2.COLOR\_BGR2RGB))

// Live Object Detection

cap = cv2.VideoCapture(1)

if not cap.isOpened():

cap = cv2.VideoCapture(0)

if not cap.isOpened():

raise IOError("Cannot Open Webcam")

font\_scale = 3

font = cv2.FONT\_HERSHEY\_PLAIN

while True:

ret,frame = cap.read()

ClassIndex, confidence, bbox = model.detect(frame, confThreshold=0.55)

#print(ClassIndex)

if (len(ClassIndex)!=0):

for ClassInd, conf, boxes in zip(ClassIndex.flatten(), confidence.flatten(), bbox):

if (ClassInd<=80):

cv2.rectangle(frame,boxes,(255,0,0),2)

cv2.putText(img,classLabels[ClassInd-1], (boxes[0]+10,boxes[1]+40), font, fontScale=font\_scale, color=(0,255,0), thickness=3)

print(classLabels[ClassInd-1])

engine.say(classLabels[ClassInd-1])

engine.runAndWait()

time.sleep(5)

cv2.imshow('Object Detection',frame)

if cv2.waitKey(2) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

* **Object Detection using OpenCV:**

Start

Import Libraries

Pre-defined Configuration file and Frozen Model

Set Input Parameters for the Model

Read an Image

Live Object Detection

Detect, Rectangle and Put Text

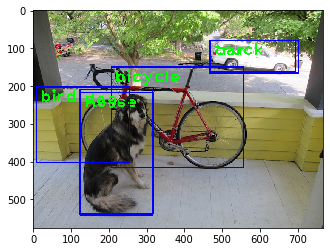
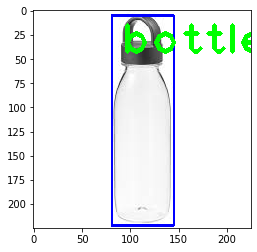
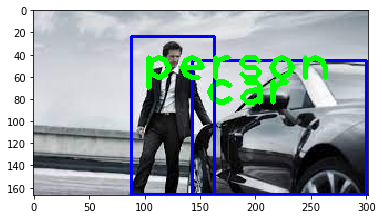
Display Detected Image

Speak Functionality

* **Object Labels:**

'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' – 80 Object Labels.

**Object Detection Results**



* **Sign Language Detection:**

Terminate the program

Display the result

Functionality

Count Fingers

Segment the region of hand in the image

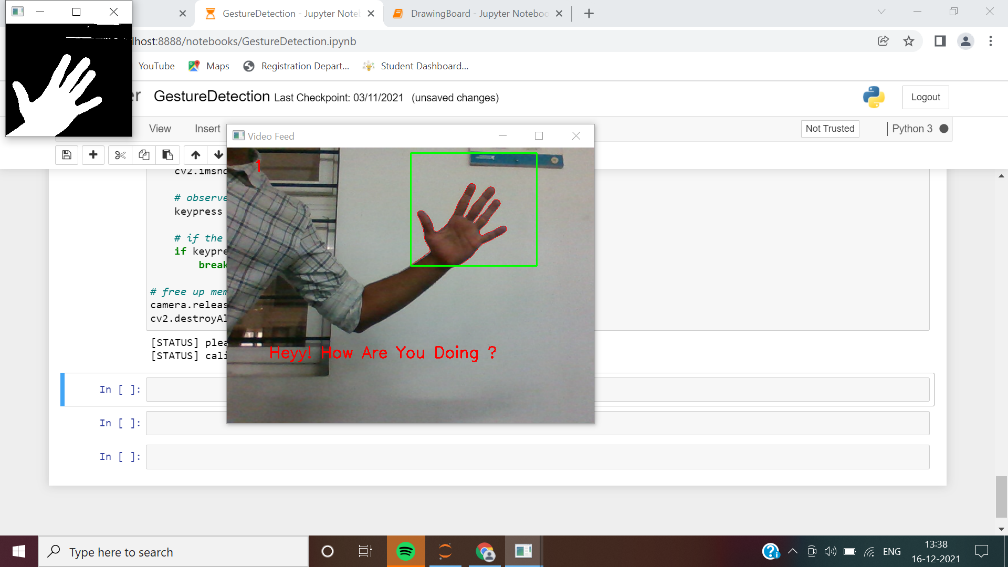
Convex Hull Algorithm to find extremities

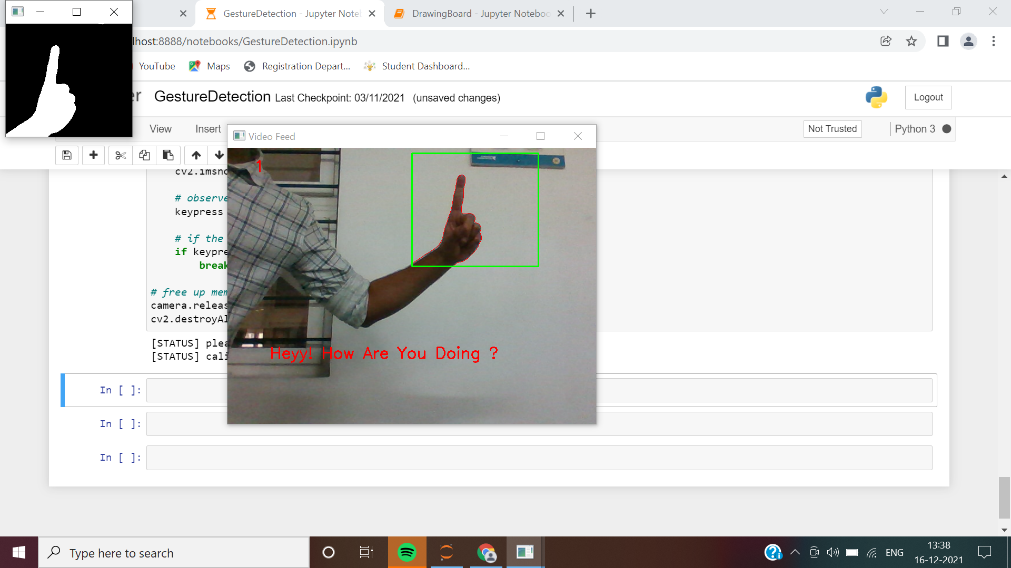
Find Running Average over the background

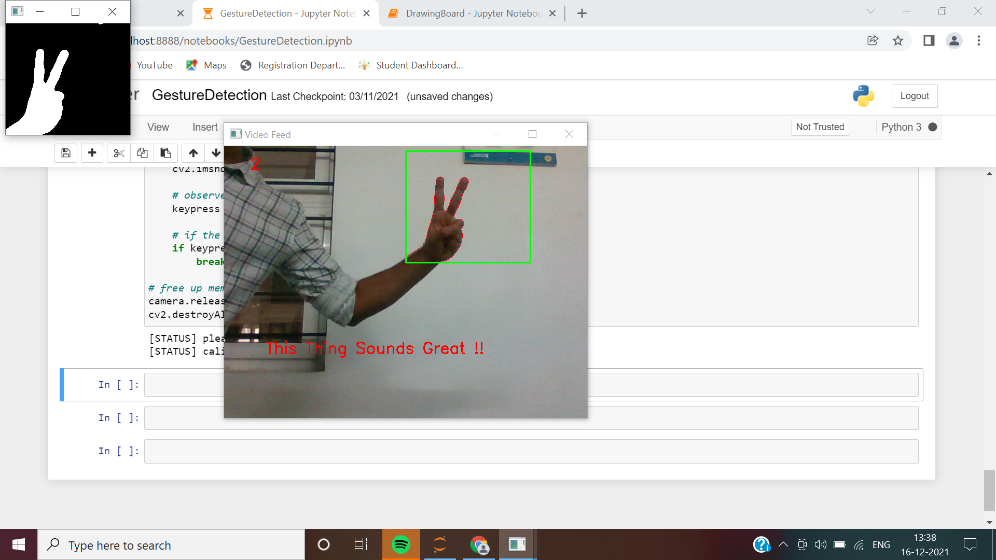
Import Libraries

Start

**Sign Language Detection Results**

****

****

****

**Program (Sign Language Detection):**

import cv2

import imutils

import numpy as np

from sklearn.metrics import pairwise

import math

import pyautogui as p

import time as t

*# global variables*

bg = None

*#--------------------------------------------------*

*# To find the running average over the background*

*#--------------------------------------------------*

def run\_avg(image, accumWeight):

global bg

*# initialize the background*

if bg is None:

bg = image.copy().astype("float")

return

*# compute weighted average, accumulate it and update the background*

cv2.accumulateWeighted(image, bg, accumWeight)

*#---------------------------------------------*

*# To segment the region of hand in the image*

*#---------------------------------------------*

def segment(image, threshold=25):

global bg

*# find the absolute difference between background and current frame*

diff = cv2.absdiff(bg.astype("uint8"), image)

*# threshold the diff image so that we get the foreground*

thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH\_BINARY)[1]

*# get the contours in the thresholded image*

(cnts, \_) = cv2.findContours(thresholded.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

*# return None, if no contours detected*

if len(cnts) == 0:

return

else:

*# based on contour area, get the maximum contour which is the hand*

segmented = max(cnts, key=cv2.contourArea)

return (thresholded, segmented)

*#--------------------------------------------------------------*

*# To count the number of fingers in the segmented hand region*

*#--------------------------------------------------------------*

def count(thresholded, segmented):

*# find the convex hull of the segmented hand region*

chull = cv2.convexHull(segmented)

*# find the most extreme points in the convex hull*

extreme\_top = tuple(chull[chull[:, :, 1].argmin()][0])

extreme\_bottom = tuple(chull[chull[:, :, 1].argmax()][0])

extreme\_left = tuple(chull[chull[:, :, 0].argmin()][0])

extreme\_right = tuple(chull[chull[:, :, 0].argmax()][0])

*# find the center of the palm*

cX = int((extreme\_left[0] + extreme\_right[0]) / 2)

cY = int((extreme\_top[1] + extreme\_bottom[1]) / 2)

*# find the maximum euclidean distance between the center of the palm*

*# and the most extreme points of the convex hull*

distance = pairwise.euclidean\_distances([(cX, cY)], Y=[extreme\_left, extreme\_right, extreme\_top, extreme\_bottom])[0]

maximum\_distance = distance[distance.argmax()]

*# calculate the radius of the circle with 80% of the max euclidean distance obtained*

radius = int(0.8 \* maximum\_distance)

*# find the circumference of the circle*

circumference = (2 \* np.pi \* radius)

*# take out the circular region of interest which has*

*# the palm and the fingers*

circular\_roi = np.zeros(thresholded.shape[:2], dtype="uint8")

*# draw the circular ROI*

cv2.circle(circular\_roi, (cX, cY), radius, 255, 1)

*# take bit-wise AND between thresholded hand using the circular ROI as the mask*

*# which gives the cuts obtained using mask on the thresholded hand image*

circular\_roi = cv2.bitwise\_and(thresholded, thresholded, mask=circular\_roi)

*# compute the contours in the circular ROI*

( cnts, \_) = cv2.findContours(circular\_roi.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

*# initalize the finger count*

count = 0

*# loop through the contours found*

for c in cnts:

*# compute the bounding box of the contour*

(x, y, w, h) = cv2.boundingRect(c)

*# increment the count of fingers only if -*

*# 1. The contour region is not the wrist (bottom area)*

*# 2. The number of points along the contour does not exceed*

*# 25% of the circumference of the circular ROI*

if ((cY + (cY \* 0.25)) > (y + h)) and ((circumference \* 0.25) > c.shape[0]):

count += 1

return count

accumWeight = 0.5

*# get the reference to the webcam*

camera = cv2.VideoCapture(0)

*# region of interest (ROI) coordinates*

top, right, bottom, left = 10, 350, 225, 590

*# initialize num of frames*

num\_frames = 0

*# calibration indicator*

calibrated = False

*# keep looping, until interrupted*

while(True):

*# get the current frame*

(grabbed, frame) = camera.read()

*# resize the frame*

frame = imutils.resize(frame, width=700)

*# flip the frame so that it is not the mirror view*

frame = cv2.flip(frame, 1)

*# clone the frame*

clone = frame.copy()

*# get the height and width of the frame*

(height, width) = frame.shape[:2]

*# get the ROI*

roi = frame[top:bottom, right:left]

*# convert the roi to grayscale and blur it*

gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)

gray = cv2.GaussianBlur(gray, (7, 7), 0)

*# to get the background, keep looking till a threshold is reached*

*# so that our weighted average model gets calibrated*

if num\_frames < 30:

run\_avg(gray, accumWeight)

if num\_frames == 1:

print("[STATUS] please wait! calibrating...")

elif num\_frames == 29:

print("[STATUS] calibration successfull...")

else:

*# segment the hand region*

hand = segment(gray)

*# check whether hand region is segmented*

if hand is not None:

*# if yes, unpack the thresholded image and*

*# segmented region*

(thresholded, segmented) = hand

*# draw the segmented region and display the frame*

cv2.drawContours(clone, [segmented + (right, top)], -1, (0, 0, 255))

*# count the number of fingers*

fingers = count(thresholded, segmented)

cv2.putText(clone, str(fingers), (50, 45), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,0,255), 2)

if str(fingers)=='1':

cv2.putText(clone, 'Message 1', (80, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,0,255), 2)

*#p.press("down")*

elif str(fingers)=='2':

cv2.putText(clone, 'Message 2', (80, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,0,255), 2)

*#p.press("right")*

elif str(fingers)=='3':

cv2.putText(clone, 'Message 3', (80, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,0,255), 2)

else:

pass

*# show the thresholded image*

cv2.imshow("Thesholded", thresholded)

*# draw the segmented hand*

cv2.rectangle(clone, (left, top), (right, bottom), (0,255,0), 2)

*# increment the number of frames*

num\_frames += 1

*# display the frame with segmented hand*

cv2.imshow("Video Feed", clone)

*# observe the keypress by the user*

keypress = cv2.waitKey(1) & 0xFF

*# if the user pressed "q", then stop looping*

if keypress == ord("q"):

break

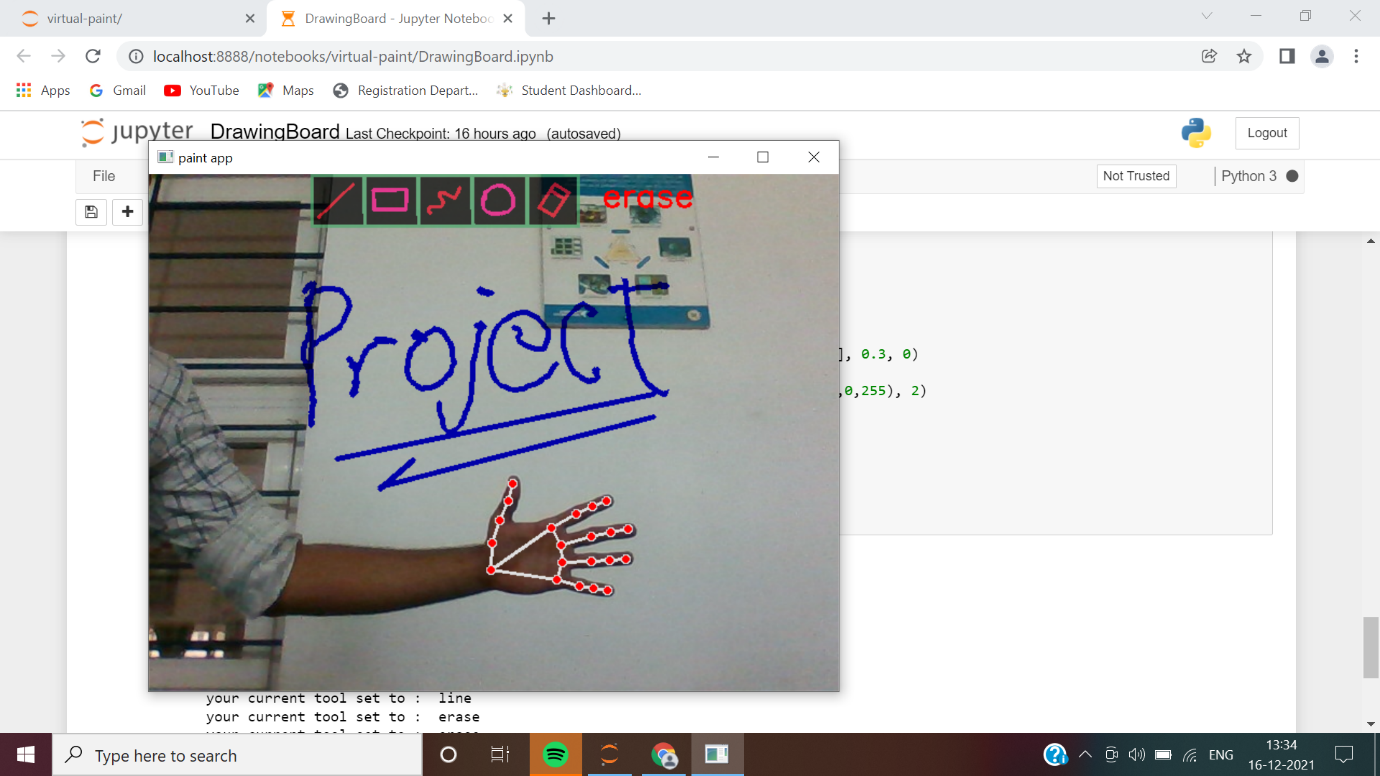
*# free up memory*

camera.release()

cv2.destroyAllWindows()

**Gesture Detection Application:**

**Gesture Detection Results**

****