Robotic Vision - Project

Monitoring the Status of Social Distancing

Muhammad Osama Fawad (2016342)

Husnain Niazi (2016175)

import the necessary packages

In [1]:
```
# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import cv2
from math import pow, sqrt
import sys
from scipy.spatial import distance ##
```

Loading the SSD Object Detection Model pre-trained on 21 classes of COCO dataset

From the 21 classes/objects only 'Persons' class passes through and the rest of them are filtered out

In [28]:
```
# initialize the list of class labels MobileNet SSD was trained to detect
# then generate a set of bounding box colors for each class

CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
        "bottle", "bus", "car", "cat", "chair", "cow", "table",
        "dog", "horse", "motorbike", "Person" , "pottedplant", "sheep",
        "sofa", "train", "tvmonitor"]

#Ignoring all the classes except PERSON

IGNORE = set(["background", "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair", "cow", "table",
    "dog", "horse", "motorbike", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"])


con = 0.18 #Confidence or threshold probability during prediction

#COLOR OF BOUNDING BOX
#Randomly applying a unique color to each class
#COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))


# loading the pre-trained SSD Object Detection Model
print("Loading model...")
net = cv2.dnn.readNetFromCaffe('MobileNetSSD_deploy.prototxt.txt', 'MobileNetS
SD_deploy.caffemodel')
```

```
Loading model...
```

DISTANCE AND STATUS FOUND ON IMAGE

In [39]:

```
# grab the frame dimensions and convert it to a blob

frame = cv2.imread('image2.png') #IMAGE PATH

height = frame.shape[0]
width = frame.shape[1]



#Perform PERSPECTIVE TRANSFORMATION to get Birds Eye View
point1 = np.float32([ [665,19], [1347,163], [15, 471], [1209,731]   ])
point2 = np.float32([ [0,0], [width,0], [0, height], [width,height]   ])
#[53,661], [825,1024], [1575, 277], [1069,25] for image.jpg
#

matrix = cv2.getPerspectiveTransform(point1,point2)
birds_eye = cv2.warpPerspective(frame,matrix,(width,height))

cv2.imshow("Original Image", frame)
cv2.imshow("Perspective Transformed Image", birds_view)
cv2.waitKey(0)


###################################
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 0.007843, (400, 40
0), 127.5)

# pass the blob through the network and obtain the detections and
# predictions
net.setInput(blob)
detections = net.forward()


###################################
#Declaring two python Dictionaries (each index containing a Key and a Value)
mid_dict = dict()
coordinates = dict()

# Focal length
F = 615

#Initialize some variables
total_persons = 0
violations = 0
threshold = 110 #minimum distance allowed
###################################

print("The Distance of Each Detected Person from Camera")

#LOOP OVER THE DETECTIONS
for i in np.arange(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated with
    # the prediction
    confidence = detections[0, 0, i, 2]

    # filter out weak detections by ensuring the `confidence` is
    # greater than the minimum confidence
    if confidence > con:
        # extract the index of the class label from the
        # `detections`
        idx = int(detections[0, 0, i, 1])


        #By doing the following we IGNORE detection of all other classes excep
t Persons
        if CLASSES[idx] in IGNORE:
            continue

        # compute the (x, y)-coordinates of the bounding box for
        # the object
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
```

```
        #Defining the color for bounding box
        box_color = (0,255,0)

        #draw the prediction on the frame
        label = "{}: {:.2f}%".format(CLASSES[idx],
            confidence * 100)
        cv2.rectangle(frame, (startX, startY), (endX, endY),
            box_color, 2)

#         y = startY - 15 if startY - 15 > 15 else startY + 15
#         cv2.putText(frame, label, (startX, y),
#             cv2.FONT_HERSHEY_SIMPLEX, 0.5, box_color, 2)


################################################################
        #Bounding Box Coordinates are stored in the COORDINATES DICTIONARY
        #It is updated after each loop to add the coordinates of bounding box
 of new detections
        coordinates[i] = (startX, startY, endX, endY)

        # Mid point of bounding box
        x_mid = int((startX+endX)/2)
        y_mid = int((startY+endY)/2)

        cv2.circle(frame,(x_mid, y_mid), 7, (0,250,250), -2)

        height = int(endY-startY) #Height of Bounding Box = Height of Person i
n Pixels

        # Depth of Detected Person from Camera based on Triangle Similarity
        distance = int((165 * F)/height)

        #Prints the Depth of Each Detected Person from Camera (in cm)
        print("Depth of Person ", i, ":", distance, "cm")

        # Mid-point of bounding boxes (in cm) based on triangle similarity tec
hnique
        x_mid_cm = int((x_mid * distance) / F)
        y_mid_cm = int((y_mid * distance) / F)

        #Mid-Point Coordinates (in CM) of eacb Detected Person are stored in t
he MID_DICT DICTIONARY
        #It is updated after each loop to add the midpoint coordinates of the
 new detected person
        mid_dict[i] = (x_mid_cm,y_mid_cm,distance)

        #Increment the Total Number of People Detected
        total_persons = total_persons + 1

################################################################


# EUCLIDEAN DISTANCE BETWEEN EVERY DETECTED OBJECT'S MIDPOINT IN FRAME
close_objects = set()
for i in mid_dict.keys():
    for j in mid_dict.keys():
        if i < j:
            dist = sqrt(pow(mid_dict[i][0]-mid_dict[j][0],2) +
                        pow(mid_dict[i][1]-mid_dict[j][1],2) +
                        pow(mid_dict[i][2]-mid_dict[j][2],2))

            # Check if distance less than the allowed Threshold
            if dist < threshold: #DISTANCE THRESHOLD
                close_objects.add(i)
                close_objects.add(j)

########################################################################

#DISPLAYING AND HIGHLIGHTING THE VIOLATORS AND NON-VIOLATORS ON FRAME

for i in mid_dict.keys():

    if i in close_objects: #If the position coordinates of person belong to th
```

```
e set of violators
        COLOR = (0,0,255) #Red Color Bounding Box if Person is in close proxim
ity
        violations = violations + 1

    else: #If the position coordinates of person does not belong to the set of
violators
        COLOR = (0,255,0) #Green Color Bounding Box if Persons are Far

    (startX, startY, endX, endY) = coordinates[i]

    #Display Bounding Box around Detected Person (Red Box = Very Close) & (Gre
en Box = Safe Distance)
    cv2.rectangle(frame, (startX, startY), (endX, endY), COLOR, 3)
    y = startY - 15 if startY - 15 > 15 else startY + 15

    #Display Labels around Detected Person (Red Label = Very Close) & (Green L
abel = Safe Distance)
    cv2.putText(frame, label, (startX, y-10),cv2.FONT_HERSHEY_SIMPLEX, 0.5, CO
LOR, 2)

    # Display Depth of Person from Camera (Red Label = Very Close) & (Green La
bel = Safe Distance)
    #Convert cm to feet
    cv2.putText(frame, 'Depth: {i} ft'.format(i=round(mid_dict[i][2]/30.48,4
)), (startX, y+10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLOR, 2)
############################################################################

#PRINTING ALL THE OUTCOMES AND STATUS
print()
print("The 3D midpoints (x,y,z) of each detected person in cm")

for key in mid_dict.keys():
    print("Mid-Points of person ", key, ": ", mid_dict[key])

print()
print("Total PERSONS Detected: ", total_persons)
print("Total VIOLATIONS Detected: ", violations)
safe = (violations/total_persons)*100
print(safe, "% of people are NOT maintaining the recommended distance")
print(100-safe, "% of people are maintaining the recommended distance")
print("Minimum allowable distance between two persons: ",threshold, "cm")


#DISPLAYING THE STATUS ON IMAGE
text1 = "Total Persons Detected: "
text2 = str(total_persons)
text3 = "Total Violations Detected: "
text4 = str(violations)

status1 = text1 + text2
status2 = text3 + text4

#Displaying the Status of Social Distancing on Frame
cv2.putText(frame, status1, (5,65),cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),
3)
cv2.putText(frame, status2, (5,95),cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),
3)
cv2.putText(frame, "M.Osama Fawad (2016342)", (5,20),cv2.FONT_HERSHEY_SIMPLEX,
0.7,(0,130,255), 2)
cv2.putText(frame, "Husnain Niazi (2016175)", (5,40),cv2.FONT_HERSHEY_SIMPLEX,
0.7,(0,130,255),2)

#show the output image
cv2.imshow("Output", frame)
cv2.waitKey(0)
```

```
The Distance of Each Detected Person from Camera
Depth of Person  0 : 593 cm
Depth of Person  1 : 685 cm
Depth of Person  2 : 704 cm
Depth of Person  3 : 719 cm
Depth of Person  4 : 238 cm
Depth of Person  5 : 1014 cm
Depth of Person  6 : 654 cm

The 3D midpoints (x,y,z) of each detected person in cm
Mid-Points of person  0 :  (184, 394, 593)
Mid-Points of person  1 :  (31, 416, 685)
Mid-Points of person  2 :  (1382, 289, 704)
Mid-Points of person  3 :  (113, 438, 719)
Mid-Points of person  4 :  (484, 201, 238)
Mid-Points of person  5 :  (1846, 184, 1014)
Mid-Points of person  6 :  (455, 408, 654)

Total PERSONS Detected:  7
Total VIOLATIONS Detected:  2
28.57142857142857 % of people are NOT maintaining the recommended distance
71.42857142857143 % of people are maintaining the recommended distance
Minimum allowable distance between two persons:  110 cm
```

Out[39]: -1

DISTANCE AND STATUS FOUND ON WEBCAM/VIDEO

In [40]:
```
print("MONITORING THE SOCIAL DISTANCING\n")
print("Starting Video Stream...")
print("\n1-Detects Persons \n2-Finds Total Persons in a Frame \n3-Calculate De
pth of each Person from Camera \n4-Monitor Total Number of People Violating So
cial Distancing Rules\n")

#For Live Webcam Feed as Input
#video = VideoStream(src=0).start()

#For Video Path as Input
video = VideoStream('pedestrians.mp4').start()


#Starting the FPS counter
time.sleep(2.0)
fps = FPS().start()


###################################
#con = 0.2 #Confidence Threshold for Prediction

#Declaring two python Dictionaries (each element containing a Key and a Value)
#mid_dict = dict()
#coordinates = dict()

# Focal length
F = 615 #Must be changed according to situation

#Initialize some variables
threshold = 110 #minimum distance allowed
fr = 0
###################################



#LOOPS OVER ALL THE FRAMES OF VIDEO
while True:

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

    #Grab the frame dimensions and convert it to a blob
    #Convert Input File into a Suitable Format that the Model can Read, Blob d
oes the Preprocessing
    (h, w) = frame.shape[:2]

    #Frame Resized to suitable dimensions
    #The FPS depend on the size we put in the following code, more processing
 time for greater size
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (500, 500)), 0.007843, (200
, 200), 127.5)

    #BLOB is now PASSED through the network/model to get the DETECTIONS and PR
EDICTIONS
    net.setInput(blob)
    detections = net.forward()

    #Initializing the Dictionaries and Variables
    mid_dict = dict()
    coordinates = dict()
    total_persons = 0
    violations = 0
    fr = fr + 1
    print()


    ######################## DETECTION ################################
    #LOOP OVER THE DETECTIONS
    for i in np.arange(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the prediction
        confidence = detections[0, 0, i, 2]
```

```
        #Filtering out the weak predictions (those having probability of predi
ction less than the confidence threshold)
        if confidence > con:
            # extract the index of the class label from the detections
            idx = int(detections[0, 0, i, 1])

            #By doing the following we IGNORE detection of all other classes e
xcept Persons
            if CLASSES[idx] in IGNORE:
                continue

            # compute the (x, y)-coordinates of the bounding box for
            # the object
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            #Defining the color for bounding box
            box_color = (0,255,0)

   ##############################################################
 #           #draw the prediction on the frame
            label = "{}: {:.2f}%".format(CLASSES[idx],
                confidence * 100)
 #           cv2.rectangle(frame, (startX, startY), (endX, endY),
 #               box_color, 2)
 #           y = startY - 15 if startY - 15 > 15 else startY + 15
 #           cv2.putText(frame, label, (startX, y),
 #               cv2.FONT_HERSHEY_SIMPLEX, 0.5, box_color, 2)
  ################################################################

            #Bounding Box Coordinates are stored in the COORDINATES DICTIONARY
            #It is updated after each loop to add the coordinates of bounding
 box of new detections
            coordinates[i] = (startX, startY, endX, endY)

            # Mid point of bounding box
            x_mid = int((startX+endX)/2)
            y_mid = int((startY+endY)/2)

            cv2.circle(frame,(x_mid, y_mid), 7, (0,250,250), -2)

            height = int(endY-startY) #Height of Bounding Box = Height of Pers
on in Pixels

            # Depth of Detected Person from Camera based on Triangle Similarit
y
            distance = int((165 * F)/height)

            #Prints the Depth of Each Detected Person from Camera (in cm)
#            print("Depth of Person ", i, ":", distance, "cm")

            # Mid-point of bounding boxes (in cm) based on triangle similarity
technique
            x_mid_cm = int((x_mid * distance) / F)
            y_mid_cm = int((y_mid * distance) / F)

            #Mid-Point Coordinates (in CM) of eacb Detected Person are stored
 in the MID_DICT DICTIONARY
            #It is updated after each loop to add the midpoint coordinates of
 the new detected person
            mid_dict[i] = (x_mid_cm,y_mid_cm,distance)

            #Increment the Total Number of People Detected
            total_persons = total_persons + 1
    ############## DETECTION LOOP BREAKS ###################################



    #EUCLIDEAN DISTANCE BETWEEN EVERY DETECTED OBJECT'S MIDPOINT IN FRAME
    close_objects = set()
    for i in mid_dict.keys():
        for j in mid_dict.keys():
            if i < j:
                dist = sqrt(pow(mid_dict[i][0]-mid_dict[j][0],2) +
```

```
                            pow(mid_dict[i][1]-mid_dict[j][1],2) +
                            pow(mid_dict[i][2]-mid_dict[j][2],2))

                # Check if distance less than the allowed Threshold
                if dist < threshold: #DISTANCE THRESHOLD
                    close_objects.add(i)
                    close_objects.add(j)
    ########################################################################




    #DISPLAYING AND HIGHLIGHTING THE VIOLATORS AND NON-VIOLATORS ON FRAME

    for i in mid_dict.keys():

        if i in close_objects: #If the position coordinates of person belong t
o the set of violators
            COLOR = (0,0,255) #Red Color Bounding Box if Person is in close pr
oximity
            violations = violations + 1

        else: #If the position coordinates of person does not belong to the se
t of violators
            COLOR = (0,255,0) #Green Color Bounding Box if Persons are Far

        (startX, startY, endX, endY) = coordinates[i]

        #Display Bounding Box around Detected Person (Red Box = Very Close) &
 (Green Box = Safe Distance)
        cv2.rectangle(frame, (startX, startY), (endX, endY), COLOR, 3)
        y = startY - 15 if startY - 15 > 15 else startY + 15

        #Display Labels around Detected Person (Red Label = Very Close) & (Gre
en Label = Safe Distance)
        cv2.putText(frame, label, (startX, y-10),cv2.FONT_HERSHEY_SIMPLEX, 0.5
, COLOR, 2)

        # Display Depth of Person from Camera (Red Label = Very Close) & (Gree
n Label = Safe Distance)
        #Convert cm to feet
        cv2.putText(frame, 'Depth: {i} ft'.format(i=round(mid_dict[i][2]/30.48
,4)), (startX, y+10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLOR, 2)

    ########################################################################
####

    (print)
    print("FRAME:", fr)
    print("Total Persons Detected: ", total_persons)
    print("Total Violations Detected: ", violations)
    text1 = "Total Persons Detected: "
    text2 = str(total_persons)
    text3 = "Total Violations Detected: "
    text4 = str(violations)
    text5 = "FRAME:"
    text6 = str(fr)

    status1 = text1 + text2
    status2 = text3 + text4
    status3 = text5 + text6

    #Displaying the Status of Social Distancing on Frame
    cv2.putText(frame, status1, (5,55),cv2.FONT_HERSHEY_SIMPLEX, 0.8,(255,255,
255), 2)
    cv2.putText(frame, status2, (5,80),cv2.FONT_HERSHEY_SIMPLEX, 0.8,(255,255,
255), 2)
    cv2.putText(frame, status3, (5,30),cv2.FONT_HERSHEY_SIMPLEX, 1,(0,100,255
), 3)
    cv2.putText(frame, "M.Osama Fawad (2016342)", (5,h-20),cv2.FONT_HERSHEY_SI
MPLEX, 0.5,(255,255,255), 1)
    cv2.putText(frame, "Husnain Niazi (2016175)", (5,h-5),cv2.FONT_HERSHEY_SIM
```

```
PLEX, 0.5,(255,255,255), 1)

    # Display the OUTPUT frame
    cv2.imshow("Social Distancing Detector", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `ESC` key was pressed, break from the loop
    if key == 27:
        break

    # update the FPS counter
    fps.update()



############### WHILE LOOP OF FRAMES BREAKS ##################################

# stop the timer and display FPS information
fps.stop()
print()
print("Elapsed Time: {:.2f}".format(fps.elapsed()))
print("Frames Per Second (FPS): {:.2f}".format(fps.fps()))

#Clear Everything
cv2.destroyAllWindows()
video.stop()
```

```
MONITORING THE SOCIAL DISTANCING

Starting Video Stream...

1-Detects Persons
2-Finds Total Persons in a Frame
3-Calculate Depth of each Person from Camera
4-Monitor Total Number of People Violating Social Distancing Rules


FRAME: 1
Total Persons Detected:  0
Total Violations Detected:  0

FRAME: 2
Total Persons Detected:  0
Total Violations Detected:  0

FRAME: 3
Total Persons Detected:  0
Total Violations Detected:  0

FRAME: 4
Total Persons Detected:  0
Total Violations Detected:  0

FRAME: 5
Total Persons Detected:  1
Total Violations Detected:  0

FRAME: 6
Total Persons Detected:  0
Total Violations Detected:  0

FRAME: 7
Total Persons Detected:  1
Total Violations Detected:  0

FRAME: 8
Total Persons Detected:  0
Total Violations Detected:  0

FRAME: 9
Total Persons Detected:  1
Total Violations Detected:  0

FRAME: 10
Total Persons Detected:  2
Total Violations Detected:  0

FRAME: 11
Total Persons Detected:  2
Total Violations Detected:  0

FRAME: 12
Total Persons Detected:  2
Total Violations Detected:  0

Elapsed Time: 6.09
Frames Per Second (FPS): 1.81
```

In [ ]: