Social Distancing Awareness System

Problem Statement

Now, the COVID-19 Pandemic has spread enormously. To keep ourselves safe from this, social distancing is the key.

For this purpose, we have designed a system based on Image processing which will generate alarm signal whenever two person come in close contact.

Hardware and Software Required

Hardware

- CCTV
- OS based microprocessor system (i.e., Raspberry pi, Laptop, Desktop etc.)

Software

- OpenCV module of Python for Image Processing
- winsound module of Python for generating ‘Beep’

Steps of operation

To solve this problem, we need to follow certain steps, those are –

1. First detect the humans showing.
2. Mark each human by drawing rectangle or semi rectangle shapes.
3. Measure the distance between each of the human i.e., the distance between bottom middle point of each classified shape.
4. If the distance between the humans are less than 1 meter, then show the distance between them in red.
5. Generate a ‘Beep’ sound.

Human Detection

In order to detect humans we need ‘cv2’ module (OpenCV) and ‘haar cascade’ classifier named ‘haarcascade_fullbody.xml’.

We are using an inbuilt function with it called the **detectMultiScale.** This function will help us to find the features / locations of the new image. The way it does is, it will use all the features from the **human_cascade** object to detect the features of the new image.

The parameters that we will pass to this function are:

The gray scale variable — gray in our

casescaleFactor — Parameter specifying how much the image size is reduced at each image scale. Basically, the scale factor is used to create your scale pyramid. More explanation, your model has a fixed size defined during training, which is visible in the XML. This means that this size of the face is detected in the image if present. However, by rescaling the input image, you can resize a larger face to a smaller one, making it detectable by the algorithm.

1.05 is a good possible value for this, which means you use a small step for resizing, i.e. reduce the size by 5%, you increase the chance of a matching size with the model for detection is found. This also means that the algorithm works slower since it is more thorough. You may increase it to as much as 1.4 for faster detection, with the risk of missing some faces altogether. In our case, I have used 1.05 as the **scaleFactor** as this worked perfectly for the image that I was using.

minNeighbors — Parameter specifying how many neighbors each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces. Higher value results in fewer detections but with higher quality. In our case, I have taken 2 as the minNeighbors and this has worked perfectly for the image that I have used.

Code we used for Human detection –

```
# Create our body classifier
human_cascade =
cv2.CascadeClassifier('haarcascade_fullbody.xml’)


# Initiate video capture for video file
cap = cv2.VideoCapture(‘vtest.avi’)


# Loop once video is successfully loaded
while cap.isOpened():


	# Read first frame
	ret, frame = cap.read()
```

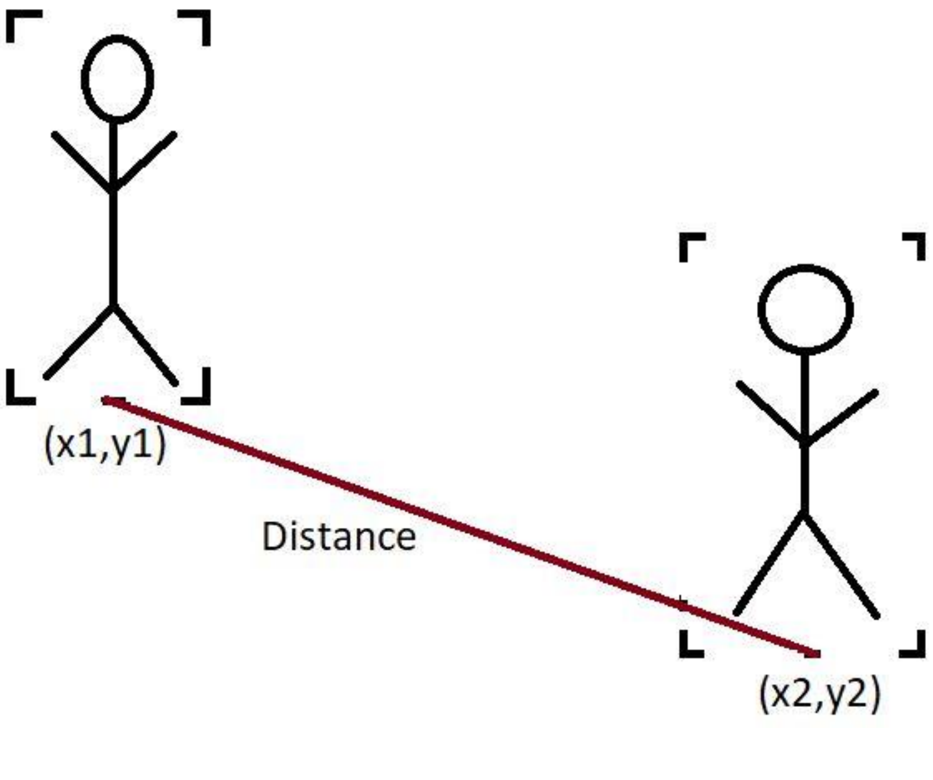
```
        #Convert frame into gray
        gray = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)

        # Pass frame to our body classifier
        humans =
human_cascade.detectMultiScale(gray, 1.05, 2)


        # Extract bounding boxes for any human
identified.[ The code shown here is to draw a
rectangle boundary. In our system we used a semi-
rectangle boundary using a function called
face_lines() ]
        for (x,y,w,h) in humans:
                cv2.rectangle(frame, (x, y), (x+w, y+h),
(255, 255, 255), 2)


        #Showing clip for every loop
                cv2.imshow(‘Screen', frame)


         #If ‘q’ is pressed, execution stops
        if cv2.waitKey(1) & 0xFF == ord('q’):
                break


#Stops loading video
cap.release()


#Destroys window frame
cv2.destroyAllWindows()
```

Distance calculation

The main part of this system is the distance calculation.

The Haar cascade classifiers return 4 values, (x, y, w, h) where, (x, y) are the ordered pair position of the top left corner of the detected object and w and h are the width and height of the object within the screen. So, the 4 corners of the detected object is (x, y), (x + w, y), (x, y + h) and (x + w, y + h).

We are going to determine the distance between the two middle points of bottom line of each object. We are going to determine the Euclidean distance between each of the objects at first.



The Euclidean distance formula is,

distance = $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

So, this is our Euclidean distance between two objects.

based on the surroundings, a constant term is multiplied with this distance to give it proper value.

Suppose, the actual distance between two person is 2 meter and Euclidean distance between them is 500 px, then we will multiply Constant term 0.004 with the 500 to determine the actual distance that is, $500 \times 0.004 = 2$.

Generating ‘Beep’ sound

To generate a beep sound, we use a python module named ‘winsound’.

The code required to generate a sound is –

```
import winsound // Importing the module
duration = 750  // Duration of sound in millisecond
freq = 400 // Frequency of sound in Hertz (Hz)
winsound.Beep(freq, duration) // Generates sound
```

Future scope

In future, whenever a region is affected by any kind of viral disease which transfers via close contact, this system can be useful for maintaining social distancing.