Computer Vision Project

Attention Recognition

Project Report

By
Qazi Umer Jamil
RIME 2019
NUST Registration No: 317920

NUST School of Mechanical and Manufacturing Engineering
Sector H-12, Islamabad, Pakistan

Dated: 07-07-2019

Contents

1 Project requirements **1**

2 Introduction **2**
2.1 Environment 2
2.1.1 OpenCV 2
2.1.2 Dlib 2

3 Background **3**
3.1 Face Recognition 3
3.2 Attention Recognition 3

4 Implementation **4**
4.1 Data Acquisition and converting to GRAYSCALE Input 4
4.2 Histogram Equalization to cater for light changes 4
4.2.1 Contrast Limited Adaptive Histogram Equalization (CLAHE) 5
4.3 Face Detection 5
4.4 Detection of Eye Blinking (Eye open/Close) 6
4.5 Gaze Detection 9

5 Evaluation and results **16**
5.1 Blinking (Eye/Open Close) Detection 16
5.2 Attention Recognition/Gaze Detection 17
5.3 Facial Features Labelling 19
5.4 Eye Tracking in random poses 20

Bibliography **21**

1. Project requirements

Attention recognition from a video or a live stream under different light conditions. The algorithm should be tack eye ball (up, down, left, right, center focus), eyes open/ close. The eye tracking should be done even if the face is turned left or right. (optional label facial parts)

2. Introduction

In this report, I have explained the key steps and the theory of implementing an attention recognition system under different light conditions. The implemented program is able to

1. Tack eye ball (up, down, left, right, center focus),
2. Blinking (eyes open/ close).
3. Labels facial parts (such as Mouth, Nose and Jaw)
4. Perform under different lighting condition.

2.1 Environment

I have used Python in PyCharm environment. Other libraries that I used are OpenCV[2] and Dlib[3]. The respective versions are as follows:

- Python 3.6
- OpenCV 3.4.2
- Dlib 19.8.2

2.1.1 OpenCV

OpenCV is a Python library which is designed to solve computer vision problems. OpenCV was originally developed in 1999 by Intel but later it was supported by Willow Garage. OpenCV Python is nothing but a wrapper class for the original C++ library to be used with Python. Using this, all of the OpenCV array structures gets converted to/from NumPy arrays.

2.1.2 Dlib

Developed by Davis King, the dlib library is a cross-platform package for threading, networking, numerical operations, computer vision, and compression. From a computer vision perspective, dlib has a number of state-of-the-art implementations, including the Facial landmark detection.

3. Background

3.1 Face Recognition

Face recognition means "given an arbitrary image, the goal of face detection is to determine whether or not there are any faces in the image and, if present, return the image location and extent of each [4]". Practically, face detection algorithms will aim at generating bounding boxes (often rectangular or elliptical) around all the faces in the image and only around faces

Some of the popular face recognition techniques are as follows:

- Viola & Jones Haar-cascade algorithm
- Speeded-Up Robust Features (SURF)
- Local binary patterns (LBP)
- Histogram of Oriented Gradients (HOG)
- Convolution Neural Network Based Models

3.2 Attention Recognition

Tracking eyes has been a subject of research and development in many fields. In literature, attention recognition is the process of measuring either the point of gaze (where one is looking) or the motion of an eye relative to the head. Eye tracking is the measurement of eye movement/activity and gaze (point of regard) tracking is the analysis of eye tracking data with respect to the head/visual scene. Researches of this field often use the terms eye-tracking, gaze-tracking or eye-gaze tracking interchangeably.

We can categories the movement of eye ball into following major movements:

- Movement to the Right
- Movement to the Left
- Movement to the Up
- Movement to the Down
- Eyeball at center

4. Implementation

4.1 Data Acquisition and converting to GRAYSCALE Input

OpenCV allows to capture webcam feed, pre-recorded video file and from IP Camera. The command for these are as follows:

```
cap = cv2.VideoCapture(0) # To capture webcam feed
cap = cv2.VideoCapture('/Users/Umer/Desktop/CV_ass/zoom_1.mp4') # To read video file
cap = cv2.VideoCapture('http://192.168.10.6:8080/video') # To access video of IP camera
```

Video frames can be read and converted to Grayscale image as follows

```
_, frame = cap.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
cv2.imshow("gray",gray)
```

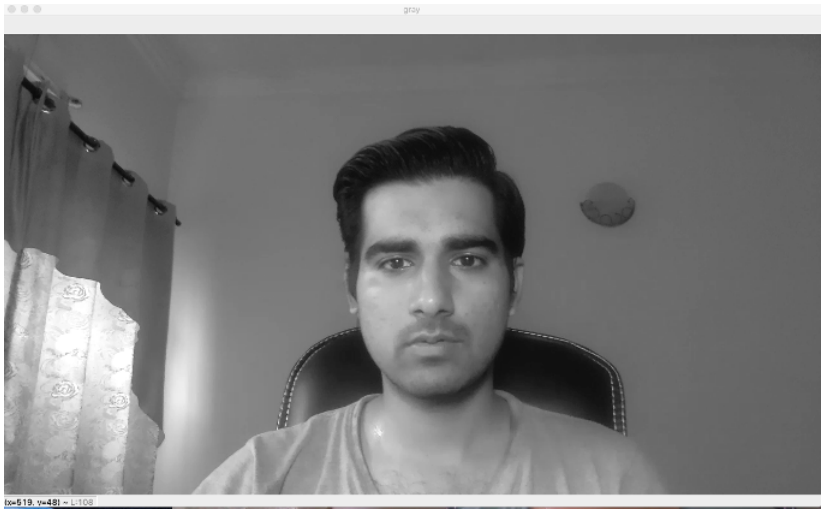

Figure 4.1: Converted Grayscale Input Image

4.2 Histogram Equalization to cater for light changes

To improve our algorithm accuracy and cater for cases for different lightening condition, we will apply a histogram equalization technique to improve the contrast of the video frame or image. Here Contrast Limited Adaptive Histogram Equalization (CLAHE) techniques to equalize images.

4.2.1 Contrast Limited Adaptive Histogram Equalization (CLAHE)

Contrast Limited Adaptive Histogram Equalization (CLAHE)[1] is a variant of Adaptive histogram equalization (AHE) which takes care of over-amplification of the contrast. CLAHE operates on small regions in the image, called tiles, rather than the entire image. The neighboring tiles are then combined using bilinear interpolation to remove the artificial boundaries.

```
#Applying CLAHE technique
clahe = cv2.createCLAHE(clipLimit=5)
gray = clahe.apply(gray)
cv2.imshow("clahe",gray)
```

Figure 4.2: CLAHE ouput on input grayscale image

4.3 Face Detection

The equalized image obtained after applying the CLAHE technique is then fed into the Facial Landmark Detector of the dlib library. We can define the dlib detector as follows as follows.

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

faces = detector(gray)
```

The detector returns a numpy array, each element containing the faces detected in the video frame.

```
# Loop through detected faces
for face in faces:
    # Draw a box around the face
    left, top = face.left(), face.top()
    right, bottom = face.right(), face.bottom()
```

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

cv2.imshow("Frame", frame)
```

First, we initialize dlib's HOG + Linear SVM face detector and load the shape_predictor file. Then, used the Dlib's detector to detect faces in the CLAHE output image. Aftwards, we loop through the detected faces, and place a bounding box on each face.

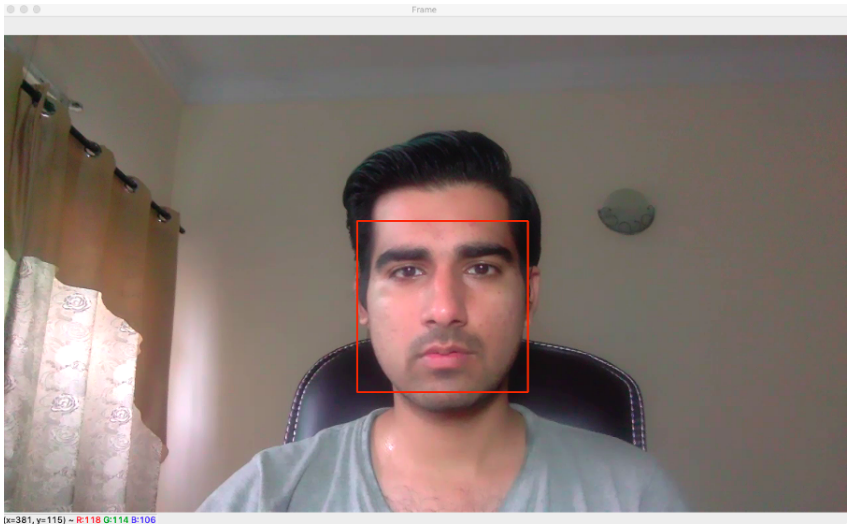

Figure 4.3: Face Detection

Now that the face is detected, we will detect other respective facial features. Here, not that we have detected faces on the CLAHE based equalized images but have drawn the bounding box on the original 3 channel camera feed as shown above.

4.4 Detection of Eye Blinking (Eye open/Close)

The facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face.

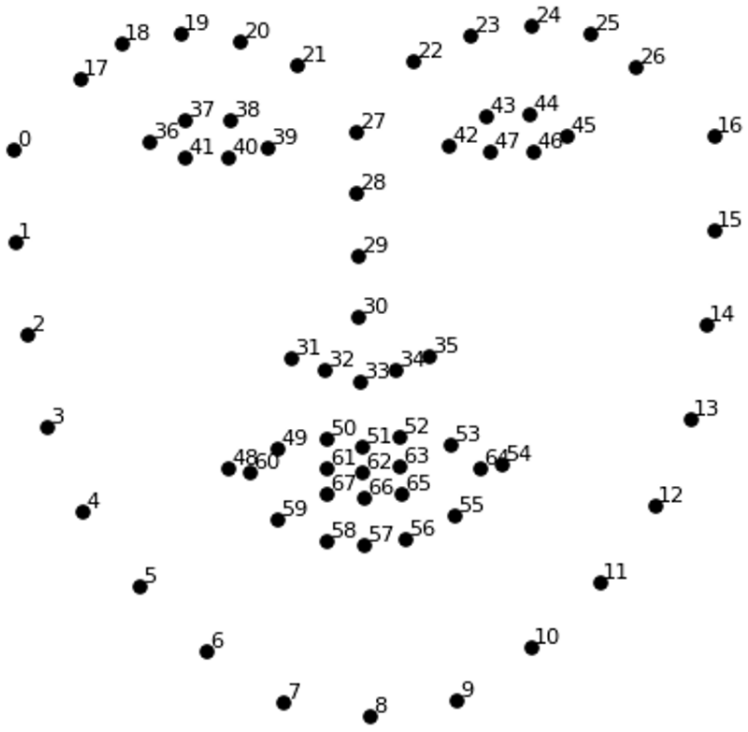

Figure 4.4: Facial Landmark Coordinates

We can see that the points for Right and Left eye is as follows:

- Right eye points: (42, 43, 44, 45, 46, 47)
- Left eye points: (36, 37, 38, 39, 40, 41)

Here the key idea is to calculate the horizontal line and vertical line based on each eye points. This can be done as follows:

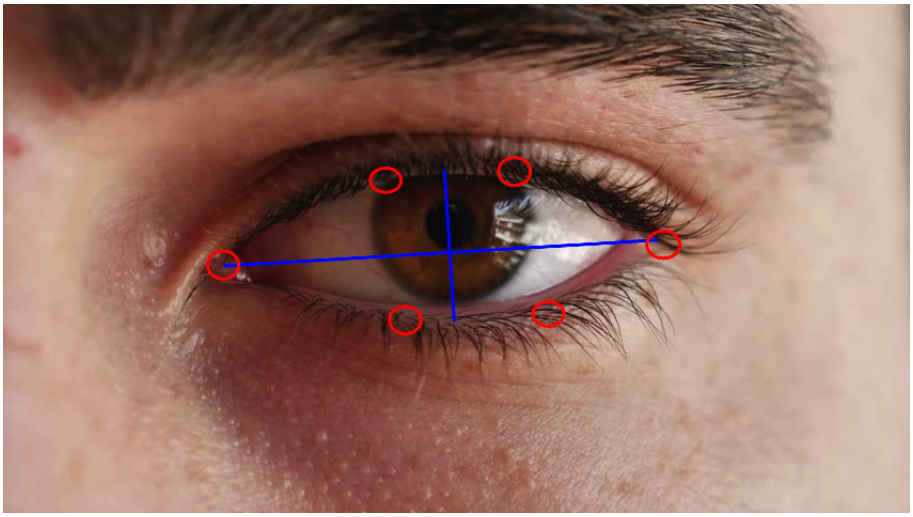

Figure 4.5: Calculating horizontal and vertical lines on eye

```
def calculate_blinking_ratio(eye_points, facial_landmarks):
    left_point = (facial_landmarks.part(eye_points[0]).x, facial_landmarks.part(eye_points
        [0]).y)
    right_point = (facial_landmarks.part(eye_points[3]).x, facial_landmarks.part(eye_points
        [3]).y)
    center_top = midpoint(facial_landmarks.part(eye_points[1]), facial_landmarks.part(
        eye_points[2]))
    center_bottom = midpoint(facial_landmarks.part(eye_points[5]), facial_landmarks.part(
        eye_points[4]))

    hor_line_lenght = hypot((left_point[0] - right_point[0]), (left_point[1] - right_point
        [1]))
    ver_line_lenght = hypot((center_top[0] - center_bottom[0]), (center_top[1] -
        center_bottom[1]))

    ratio = hor_line_lenght / ver_line_lenght
    return ratio
```

We define the blinking ratio for each eye as the ratio of the respective horizontal line length and vertical line length. We will calculate the blinking ratio for each eye and then take the average of these two ratio. If the average is greater than a specific threshold, then that means the eyes are blinking.

```
landmarks = predictor(gray, face)

# Detect blinking
left_eye_ratio = calculate_blinking_ratio([36, 37, 38, 39, 40, 41], landmarks)
right_eye_ratio = calculate_blinking_ratio([42, 43, 44, 45, 46, 47], landmarks)
blinking_ratio_avg = (left_eye_ratio + right_eye_ratio) / 2

if blinking_ratio_avg > 5.7:
    cv2.putText(frame, "-> Blinking", (50, 100), font, 2, (0, 0, 255), 3)
```

4.5 Gaze Detection

The main steps in the gaze detection of each eye are as follows along with outputs:

1. Extract the eye from the input image

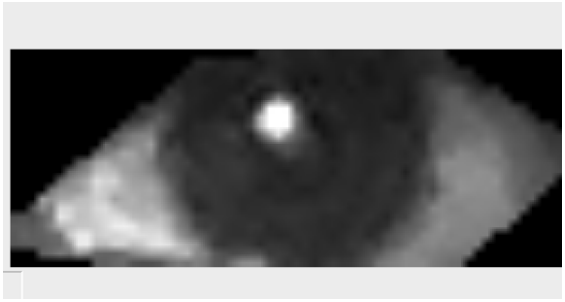

Figure 4.6: Extracted Eye from the input image

2. Convert the eye image into binary image (containing only zeros and ones pixel values)

Figure 4.7: Output: Binary Image

3. Apply Erosion to reduce pixel noise especially the pixel noise at the eye center due to illumination

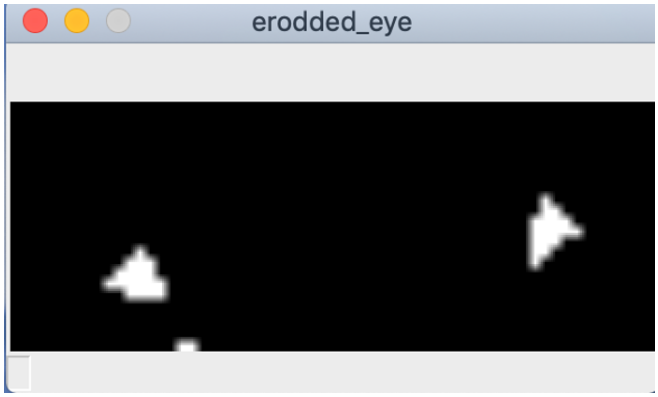

Figure 4.8: Results after Erosion

4. Apply Dilation

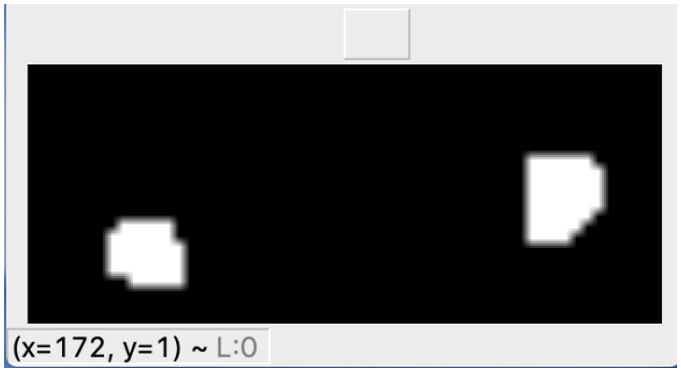

Figure 4.9: Results after Dilation

5. Apply Median Blur

   The Median blur operation is similar to the other averaging methods. Here, the central element of the image is replaced by the median of all the pixels in the kernel area. This operation processes the edges while removing the noise.

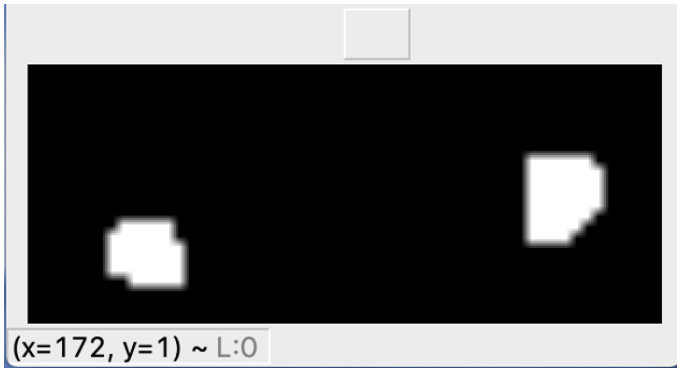

Figure 4.10: Results after Blurring

6. Detecting Up and Down movement (Gaze Ratio Vertical): To detect up and down movement of eye ball, we will split the image into two halves parts vertically.

   After splitting the eye image to two UP and DOWN images, we will count the number of non zero pixels in each half segment. If no of non zero pixels in the bottom segment is greater than the no of non zero pixels in upper segment, then it means the eye ball is looking down.

   We can calculate the ratio of the no of non zeros pixels in both segments. If this ratio is greater or less than a certain threshold, we can conclude whether the eye is looking UP or Down.

Figure 4.11: Detecting Up and Down movement (Gaze Ratio Vertical): The image of eye is segmented vertically

7. Detecting RIGHT, LEFT and CENTER movement (Gaze Ratio Horizontal): The process is similar to that of find the gaze ratio for UP and DOWN (Step 6). The only difference is that we split the eye image into to two horizontal segments instead of vertical ones.

The code for above steps is as follows:

```
def get_gaze_ratio(eye_points, facial_landmarks):
    left_eye_region = np.array([(facial_landmarks.part(eye_points[0]).x, facial_landmarks.
        part(eye_points[0]).y),
                                (facial_landmarks.part(eye_points[1]).x, facial_landmarks.part
                                    (eye_points[1]).y),
                                (facial_landmarks.part(eye_points[2]).x, facial_landmarks.part
                                    (eye_points[2]).y),
                                (facial_landmarks.part(eye_points[3]).x, facial_landmarks.part
                                    (eye_points[3]).y),
                                (facial_landmarks.part(eye_points[4]).x, facial_landmarks.part
                                    (eye_points[4]).y),
                                (facial_landmarks.part(eye_points[5]).x, facial_landmarks.part
                                    (eye_points[5]).y)], np.int32)
    #cv2.polylines(frame, [left_eye_region], True, (0, 0, 255), 2)

    height, width, _ = frame.shape
    mask = np.zeros((height, width), np.uint8)
    cv2.polylines(mask, [left_eye_region], True, 255, 2)
    cv2.fillPoly(mask, [left_eye_region], 255)
    eye = cv2.bitwise_and(gray, gray, mask=mask)

    min_x = np.min(left_eye_region[:, 0])
    max_x = np.max(left_eye_region[:, 0])
    min_y = np.min(left_eye_region[:, 1])
    max_y = np.max(left_eye_region[:, 1])
```

```
gray_eye = eye[min_y: max_y, min_x: max_x]
gray_eye1 = cv2.resize(gray_eye, None, fx=5, fy=5)

#cv2.imshow("gray_eye1", gray_eye1)

_, threshold_eye = cv2.threshold(gray_eye, 70, 255, cv2.THRESH_BINARY)
height, width = threshold_eye.shape

kernel = np.ones((5, 5), np.uint8)

erodded_eye = cv2.erode(threshold_eye, kernel, iterations=2)
#erodded_eye = cv2.resize(erodded_eye, None, fx=5, fy=5)
cv2.imshow("erodded_eye", erodded_eye)

dilated_eye = cv2.dilate(erodded_eye, kernel, iterations=4)
cv2.imshow("dilated_eye", dilated_eye)

medianblur_eye = cv2.medianBlur(dilated_eye, 3)
cv2.imshow("medianblur_eye", medianblur_eye)
threshold_eye=medianblur_eye

upper_side_threshold = threshold_eye[0: int(height/2), :]
upper_side_threshold = cv2.resize(upper_side_threshold, None, fx=15, fy=15)
cv2.imshow("upper_side_threshold", upper_side_threshold)
upper_side_threshold = cv2.countNonZero(upper_side_threshold)

bottom_side_threshold = threshold_eye[int(height/2): height, :]
bottom_side_threshold = cv2.resize(bottom_side_threshold, None, fx=15, fy=15)
cv2.imshow("bottom_side_threshold", bottom_side_threshold)
bottom_side_threshold = cv2.countNonZero(bottom_side_threshold)

if bottom_side_threshold == 0:
    bottom_side_threshold = 1
elif upper_side_threshold == 0:
    upper_side_threshold = 1

gaze_ratio_vertical = upper_side_threshold / bottom_side_threshold

left_side_threshold = threshold_eye[0: height, 0: int(width / 2)]
left_side_white = cv2.countNonZero(left_side_threshold)

right_side_threshold = threshold_eye[0: height, int(width / 2): width]
right_side_white = cv2.countNonZero(right_side_threshold)

if left_side_white == 0:
    gaze_ratio_horizental = 1
elif right_side_white == 0:
    gaze_ratio_horizental = 5
```

```
    else:
        gaze_ratio_horizental = left_side_white / right_side_white

    return gaze_ratio_horizental, gaze_ratio_vertical
```

The landmarks positions are shown in figure 4.4. The points for each facial part is as follows:

- left$_e ye_p oints = [36, 37, 38, 39, 40, 41] right_e ye_p oints = [42, 43, 44, 45, 46, 47]$
- nose$_p oints = [27, 28, 29, 30, 31, 32, 33, 34, 35, 30] mount_p oints = [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 48]$
- jaw$_p oints = [6, 7, 8, 9, 10]$

Now, to label each facial part, we will extract the x and y coordinates of each facial part and draw a polygon using these points.

Figure 4.12: Detection of various facial parts

```
def draw_facial_parts(facial_landmarks):
    # Draw circle on left and right eye
    left_eye_points = [36, 37, 38, 39, 40, 41]

    x_point_for_circle = int((landmarks.part(left_eye_points[0]).x + landmarks.part(
        left_eye_points[3]).x) / 2)
    y_point_for_circle = int((landmarks.part(left_eye_points[0]).y + landmarks.part(
        left_eye_points[3]).y) / 2)

    cv2.circle(frame, (x_point_for_circle, y_point_for_circle), 9, (0, 0, 255))

    right_eye_points = [42, 43, 44, 45, 46, 47]

    x_point_for_circle = int((landmarks.part(right_eye_points[0]).x + landmarks.part(
        right_eye_points[3]).x) / 2)
    y_point_for_circle = int((landmarks.part(right_eye_points[0]).y + landmarks.part(
        right_eye_points[3]).y) / 2)
```

```
cv2.circle(frame, (x_point_for_circle, y_point_for_circle), 9, (0, 0, 255))

# Draw polygon on Nose
nose_points = [ 27, 28, 29, 30, 31, 32, 33, 34, 35, 30]
nose_region = np.array([(facial_landmarks.part(nose_points[0]).x, facial_landmarks.part
    (nose_points[0]).y),
                        (facial_landmarks.part(nose_points[1]).x, facial_landmarks.part(
                            nose_points[1]).y),
                        (facial_landmarks.part(nose_points[2]).x, facial_landmarks.part(
                            nose_points[2]).y),
                        (facial_landmarks.part(nose_points[3]).x, facial_landmarks.part(
                            nose_points[3]).y),
                        (facial_landmarks.part(nose_points[4]).x, facial_landmarks.part(
                            nose_points[4]).y),
                        (facial_landmarks.part(nose_points[5]).x, facial_landmarks.part(
                            nose_points[5]).y),
                        (facial_landmarks.part(nose_points[6]).x, facial_landmarks.part(
                            nose_points[6]).y),
                        (facial_landmarks.part(nose_points[7]).x, facial_landmarks.part(
                            nose_points[7]).y),
                        (facial_landmarks.part(nose_points[8]).x, facial_landmarks.part(
                            nose_points[8]).y)], np.int32)
cv2.polylines(frame, [nose_region], True, (0, 0, 255), 2)

# Draw polygon on Mouth
mount_points = [ 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,58,59,48]
nose_region = np.array([(facial_landmarks.part(mount_points[0]).x, facial_landmarks.
    part(mount_points[0]).y),
                        (facial_landmarks.part(mount_points[1]).x, facial_landmarks.part(
                            mount_points[1]).y),
                        (facial_landmarks.part(mount_points[2]).x, facial_landmarks.part(
                            mount_points[2]).y),
                        (facial_landmarks.part(mount_points[3]).x, facial_landmarks.part(
                            mount_points[3]).y),
                        (facial_landmarks.part(mount_points[4]).x, facial_landmarks.part(
                            mount_points[4]).y),
                        (facial_landmarks.part(mount_points[5]).x, facial_landmarks.part(
                            mount_points[5]).y),
                        (facial_landmarks.part(mount_points[6]).x, facial_landmarks.part(
                            mount_points[6]).y),
                        (facial_landmarks.part(mount_points[7]).x, facial_landmarks.part(
                            mount_points[7]).y),
                        (facial_landmarks.part(mount_points[8]).x, facial_landmarks.part(
                            mount_points[8]).y),
                        (facial_landmarks.part(mount_points[9]).x, facial_landmarks.part(
                            mount_points[9]).y),
                        (facial_landmarks.part(mount_points[10]).x, facial_landmarks.part(
                            mount_points[10]).y),
```

```
                    (facial_landmarks.part(mount_points[11]).x, facial_landmarks.part(
                        mount_points[11]).y),
                    (facial_landmarks.part(mount_points[12]).x, facial_landmarks.part(
                        mount_points[12]).y)
                    ], np.int32)
cv2.polylines(frame, [nose_region], True, (0, 0, 255), 2)

# Draw polygon on Jaw
jaw_points = [6, 7, 8, 9, 10]
left_eye_region = np.array([(facial_landmarks.part(jaw_points[0]).x, facial_landmarks.
    part(jaw_points[0]).y),
                        (facial_landmarks.part(jaw_points[1]).x, facial_landmarks.part
                            (jaw_points[1]).y),
                        (facial_landmarks.part(jaw_points[2]).x, facial_landmarks.part
                            (jaw_points[2]).y),
                        (facial_landmarks.part(jaw_points[3]).x, facial_landmarks.part
                            (jaw_points[3]).y),
                        (facial_landmarks.part(jaw_points[4]).x, facial_landmarks.part
                            (jaw_points[4]).y)], np.int32)
cv2.polylines(frame, [left_eye_region], True, (0, 0, 255), 2)
```

5. Evaluation and results

5.1 Blinking (Eye/Open Close) Detection

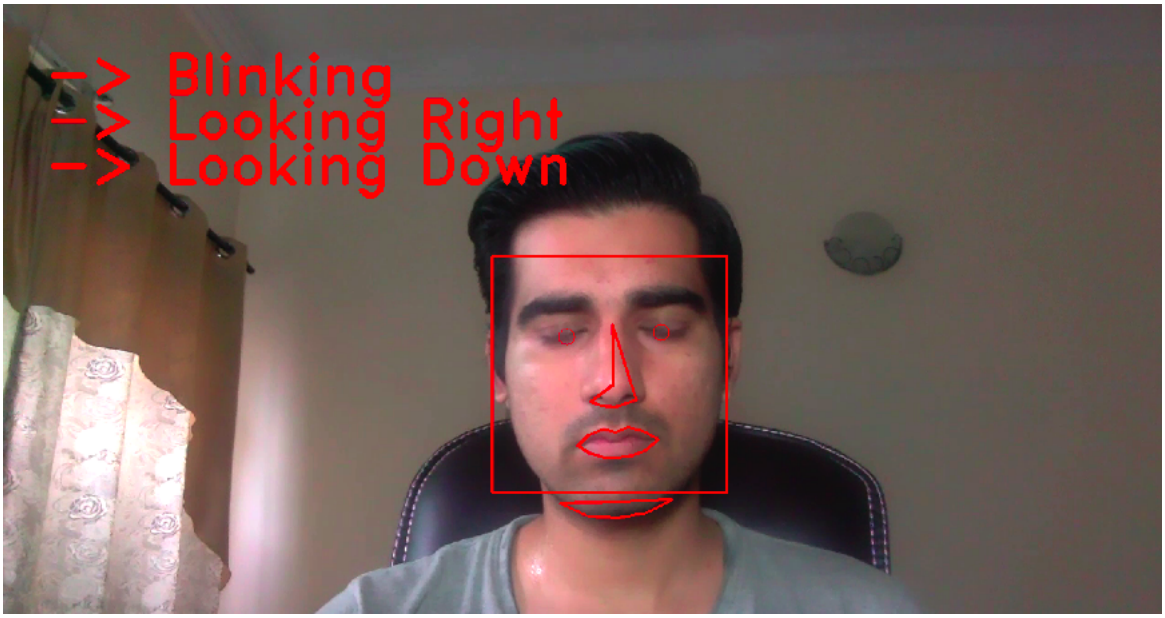

Figure 5.1: Blinking (Eye/Open Close) Detection

5.2 Attention Recognition/Gaze Detection

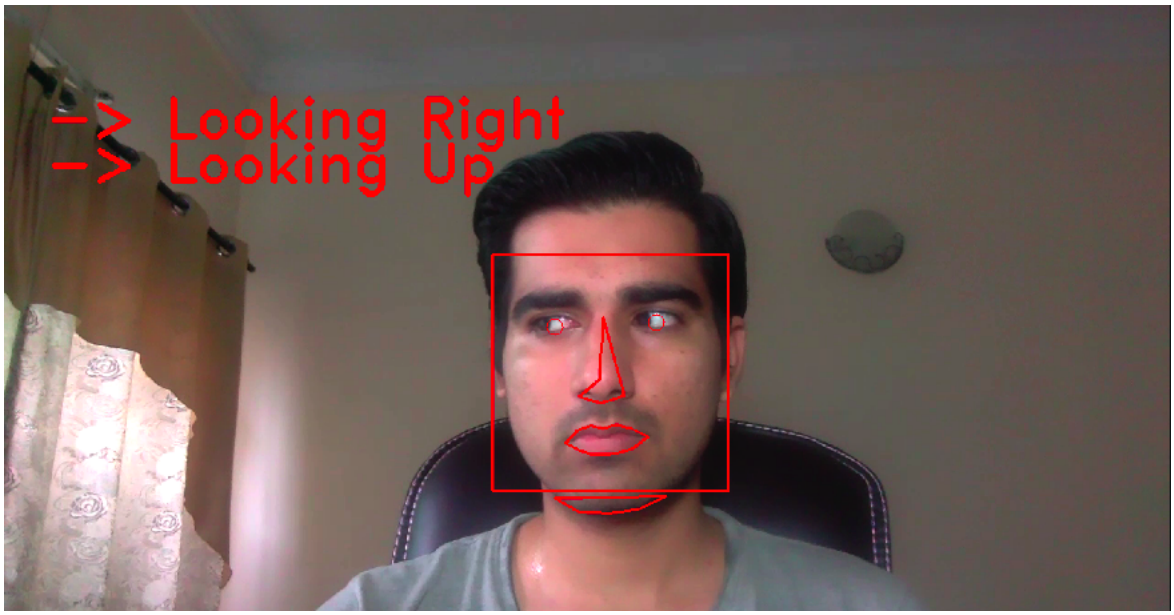

Figure 5.2: Right Gaze Detection

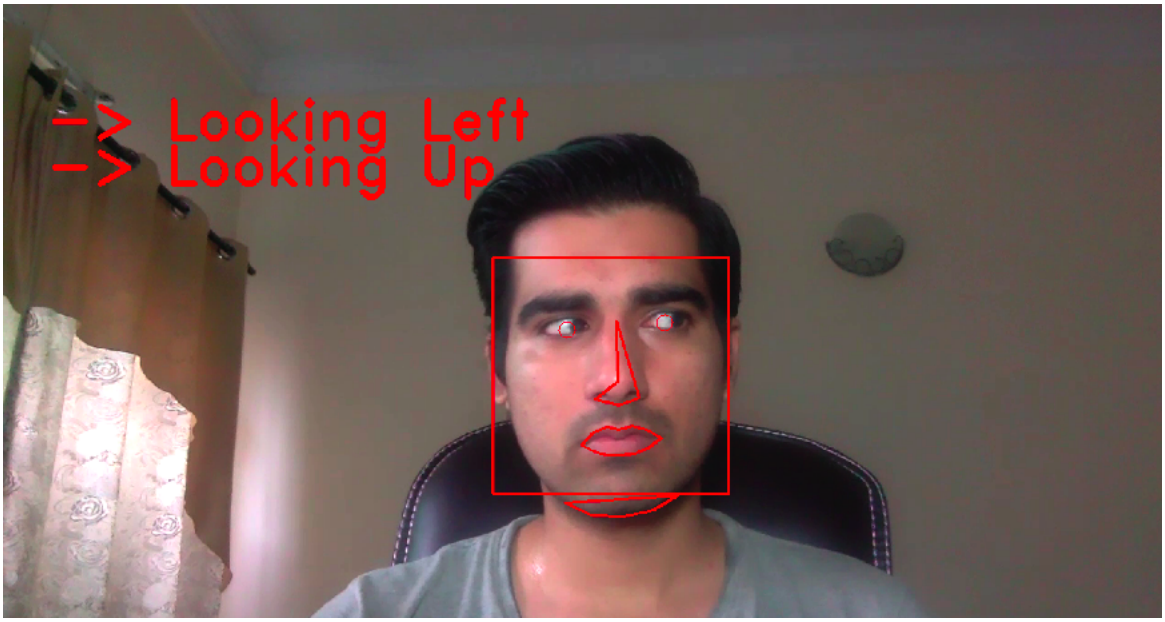

Figure 5.3: Left Gaze Detection

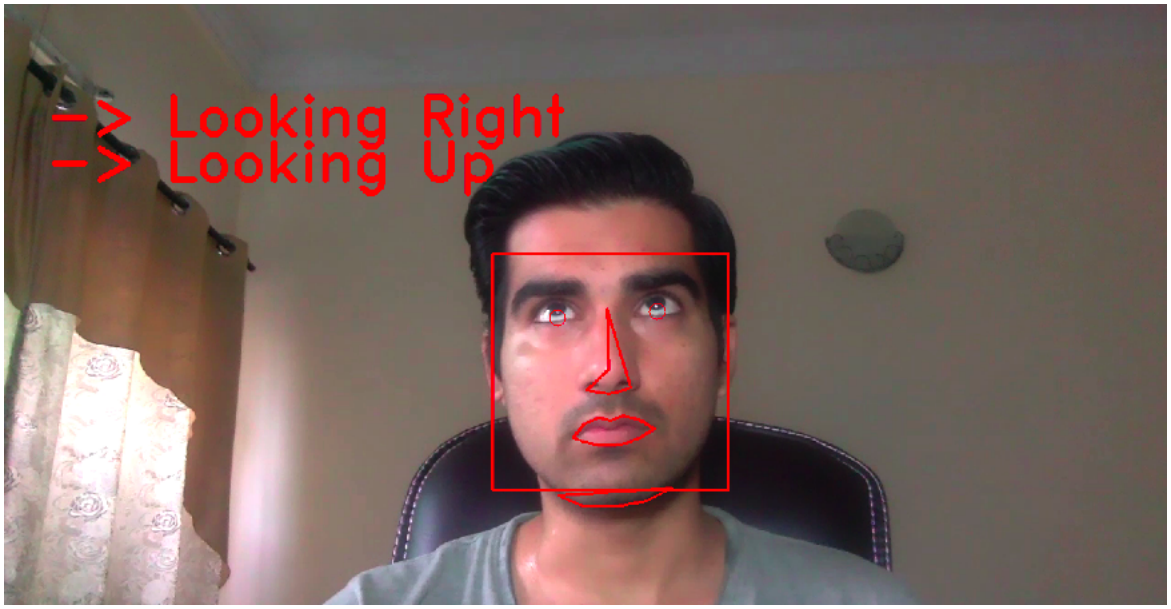

Figure 5.4: Up Gaze Detection

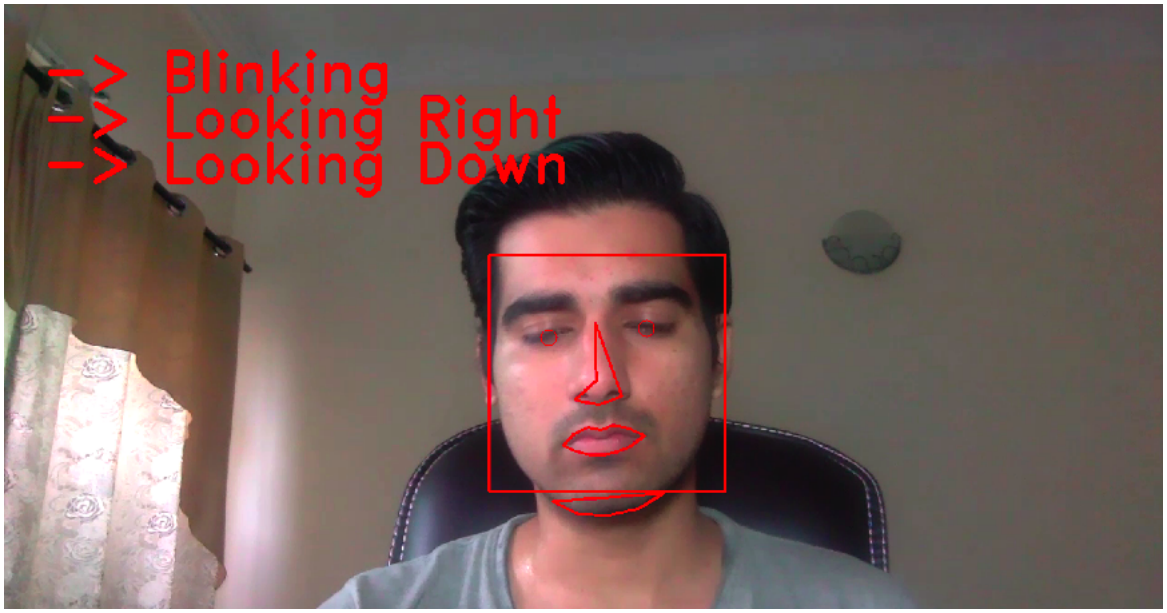

Figure 5.5: Down Gaze Detection

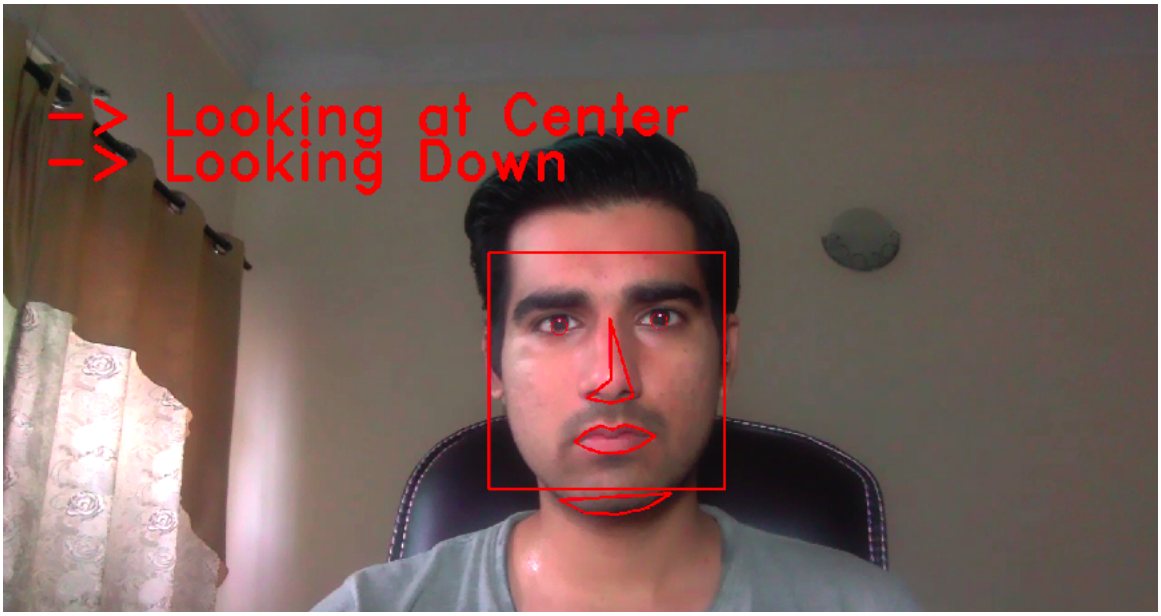

Figure 5.6: Center Gaze Detection

5.3 Facial Features Labelling

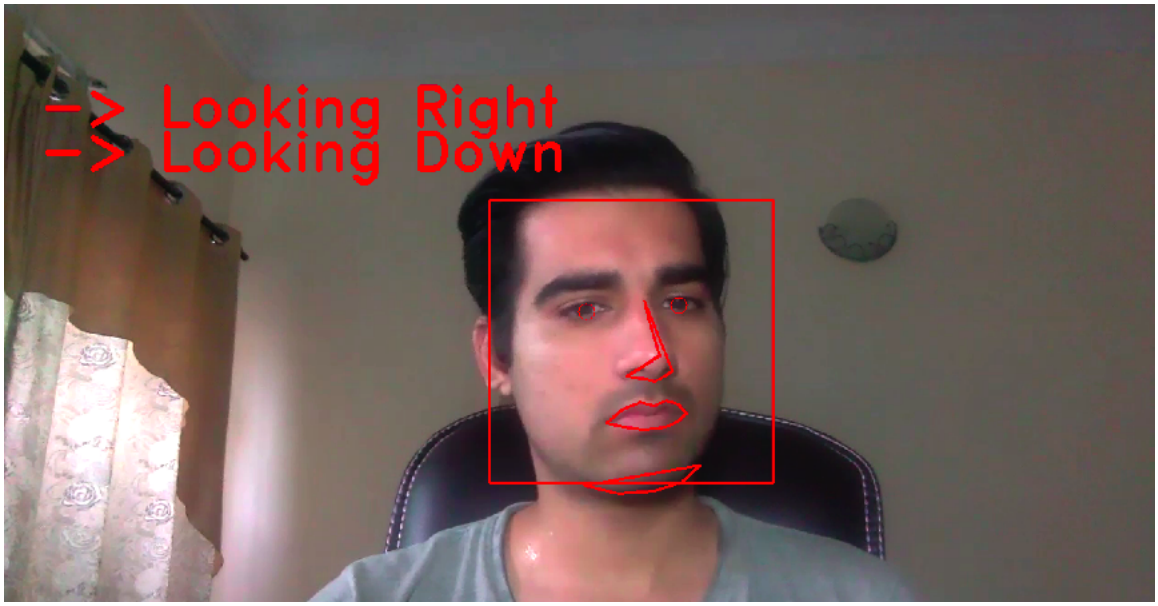

Figure 5.7: Various Facial Features

5.4 Eye Tracking in random poses

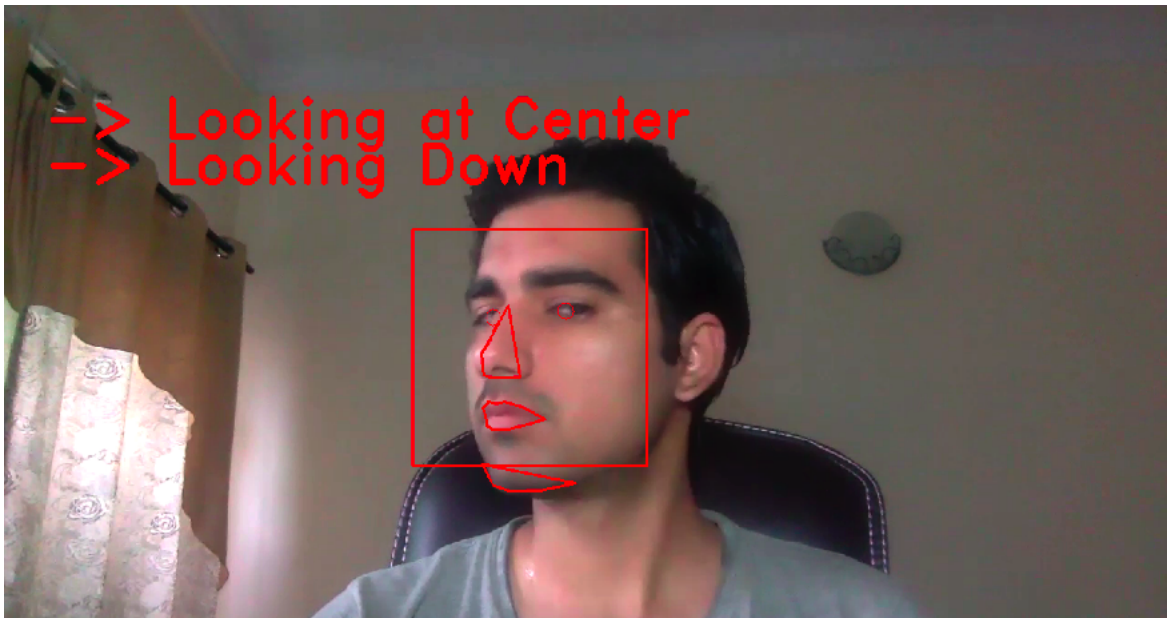

Figure 5.8: Eye Tracking in a random pose

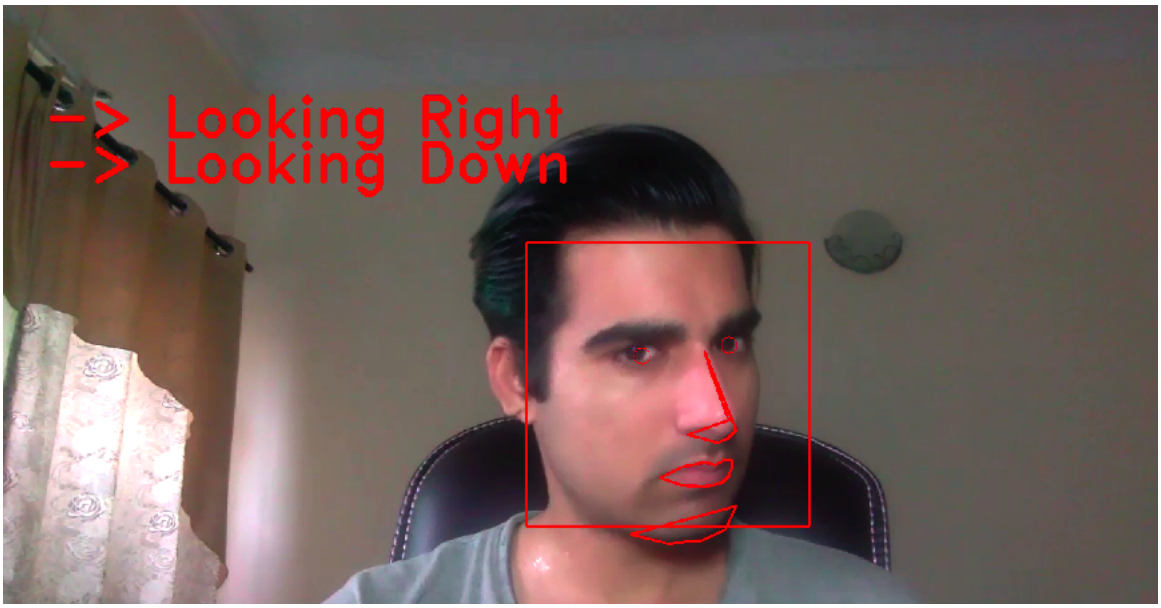

Figure 5.9: Eye Tracking in a random pose

Bibliography

[1] https://en.wikipedia.org/wiki/adaptive_histogram_equalization.

[2] https://opencv.org/.

[3] https://pypi.org/project/dlib/.

[4] http://www.maths.lth.se/media11/fma270/2012/mickenfd.pdf.