Animating a one-shot human head based on the actions of image sequences of another subject using image warping

ECEN 642 FALL '19 Digital Image Processing - Project Report
Rachen Ravichandran, UIN: 230002690, rachen@tamu.edu

Abstract

This project focuses on the animation of a portrait (static) image using a set of images of a different subject. The movements and actions of the said subject are captured as a video; for each frame of the video, the portrait image is warped such that it mimics the corresponding movement of the subject. For example, if the subject in the video turns to the right, the portrait is also warped in sequences to turn to the right. The warping is done automatically using landmark points on the face extracted by pre-trained face detector network. Thus, an animated portrait is obtained by image warping techniques instead of more computation-intensive neural networks eliminating the steps required for data procurement and cleaning.

Introduction

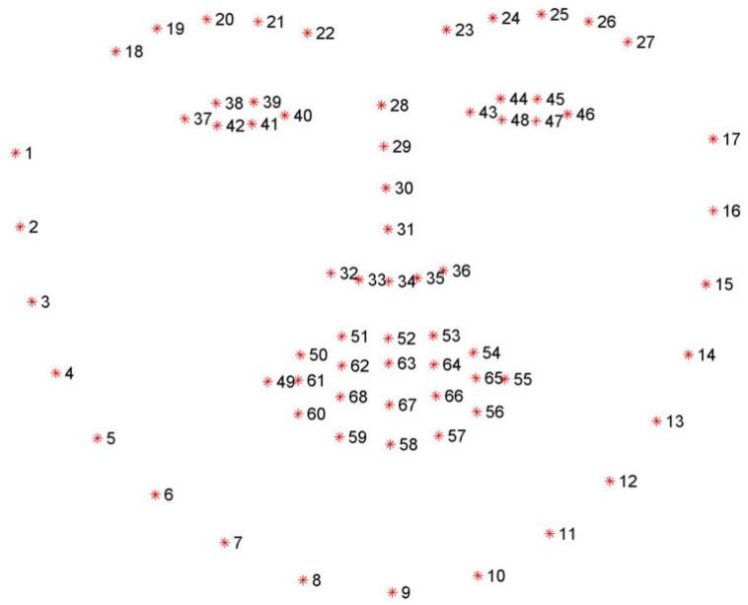
Image processing methods to animate just a single shot of an image is impossible unless supported by additional details. Several algorithms have been proposed to produce deep-fake videos using machine learning algorithms. For example, Egor Zakharov et al. produced a landmark paper where portraits were animated using regression tree machine learning algorithm[2]. The essence of these algorithms is that a huge data set of images is used to train the neural network; the network then uses this trained data to animate the movement of a stationary one-shot image based on another actor video. However, the collection of such huge data set poses a problem as it is not easy to procure the data set, clean the data, label the data and provide it as an input to the network. Furthermore, the network has to be tweaked for different images. This report tries to address this issue by using image warping technique (essentially affine transformation) instead of machine learning methods to animate portrait images based on motion of another set of actor images. As a result, this requires just a single image of portrait and no data set is needed at the input with lesser computation than a neural network computation. Moreover, the landmark points to be chosen for warping is done automatically instead of manually selecting each point as done in several morphing software. Such technique can be used in computer graphics, avatar-based video games and video-conferencing.

Fundamentals of Algorithm

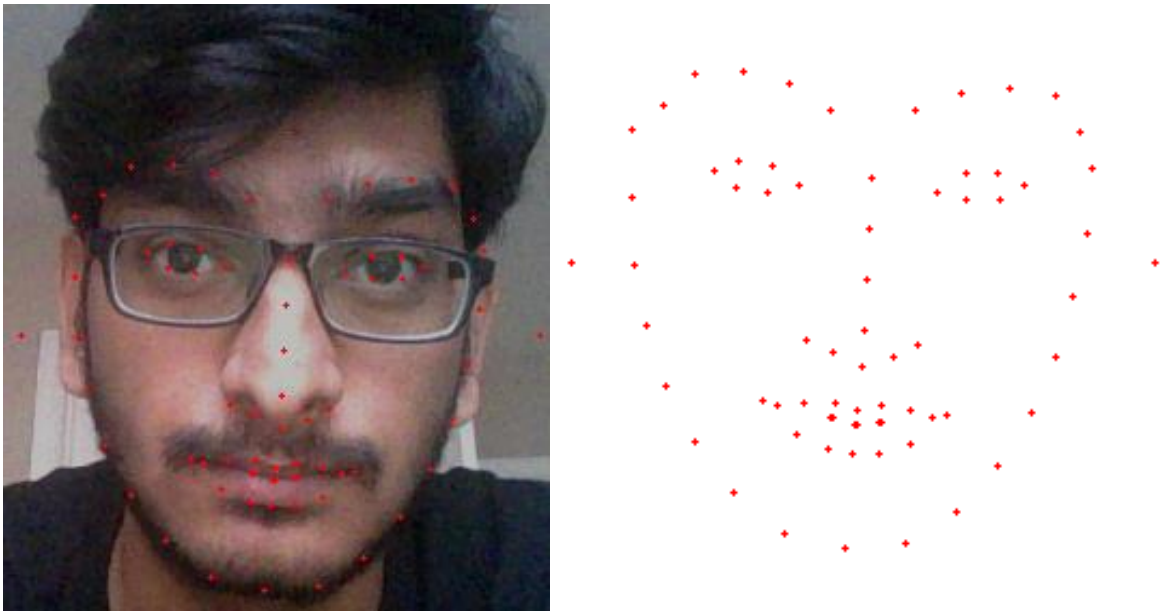
To understand the algorithm, certain fundamentals are established as below.

Face Landmark Detection

In order to automatically select the points for warping, a pre-trained neural network [6] to detect various points on face is used. Kazemi et al. [6] proposed a neural network to detect the facial landmark points based on iBUG 300-W dataset [7]. This dataset contains images of faces with 68 points labeled on facial features (along with their probability of distance between each pixel) as given below.



Then, a neural network is trained using this data set to automatically predict facial features. This pre-trained neural network can be used in Python by importing the 'dlib' library and using `dlib.shape_predictor()` function. This network will return the landmark facial points as an array which can be used as points for warping.



Note: The input to a predictor is a detected face. For this, `dlib.get_frontal_face_detector()` is used which is based on a pre-trained network of Histogram of Oriented Gradient (HoG) + Support Vector Network (SVM) model.

This model can extract only facial features of nose, eyes, jaws, mouth and eyebrows. Ears and hair are not trained. In order to include these features too, a bounding is constructed around these landmark points and the corners on the top, corners on the bottom and midpoints in in the left and right are assigned to indicate hair, shoulders and ears respectively (observe two dots near the ear region.)

Affine Transformation

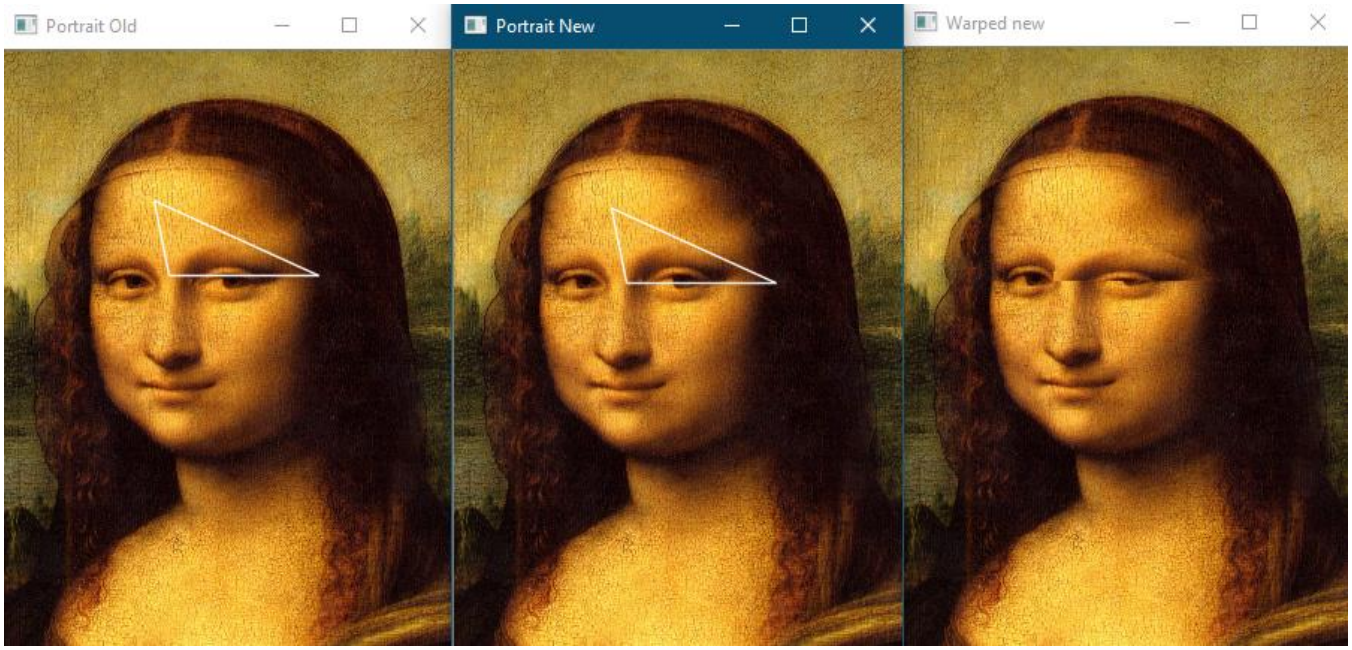
To warp any section of image, affine transformation is used [5]. The transformation performs matrix multiplication on input image to obtain the warped image. It can be used to achieve translation, rotation, shearing, reflection and scaling. For example, the following matrix manipulation can be used to achieve rotation of an input image.

$$A_{rotate} = \begin{matrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{matrix}$$

[output_coords 1] = Arotate * [input_coords 1]

where * signifies matrix multiplication

Several operations can be achieved simultaneously by multiplying the corresponding affine transforms. For instance, to obtain both rotation and translation $A_{rotate} * A_{translate}$ gives the combined step. In this paper, several sections of image are warped from input to output. A sample warping in one section is given below:

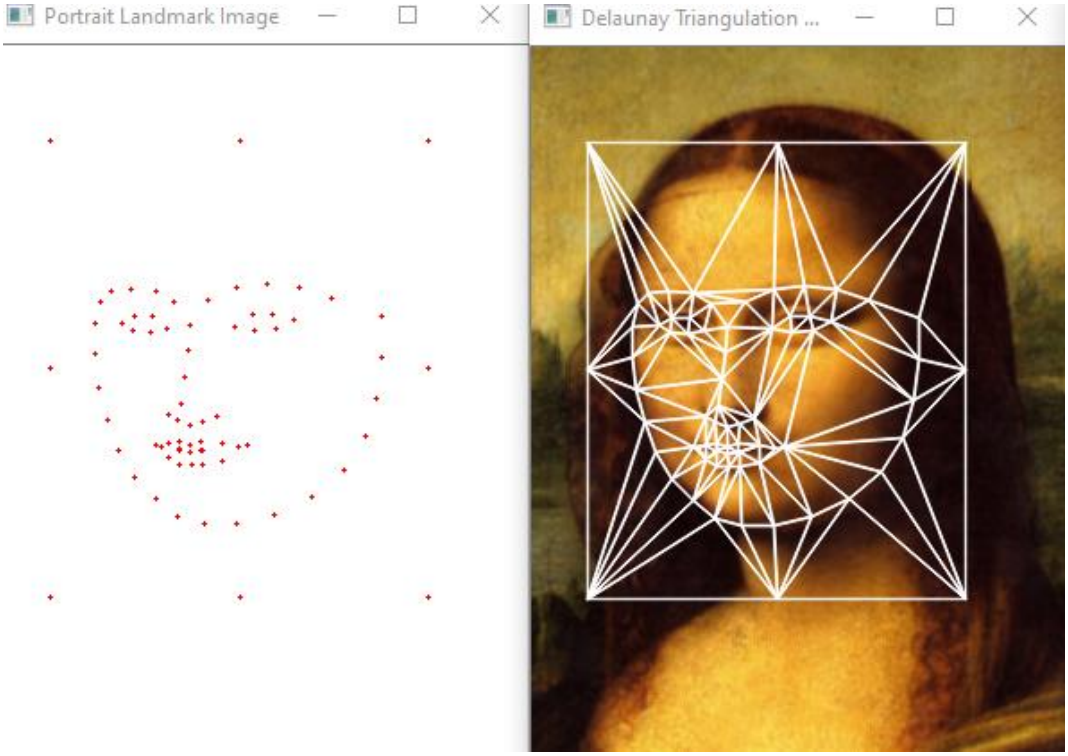


OpenCV's getAffineTransform() and warpAffine() functions are used to warp any section of input image to another section in destination image using Affine Transformation. Bilinear interpolation is used for interpolating the intensities in the warped sections.

Delaunay Triangulation

To divide the image into several sections so that it could be warped, Delaunay Triangulation [5] is used. This triangulation technique constructs triangles around for each of three set of closest points with maximum angles between them. As a result, an automatic selection of triangular sections can be achieved without the intervention from the user. In this project, the landmark points are used to construct the Delaunay triangles. For example, as given below, triangles are constructed in various sections of the face using landmark points. Each triangular section can be then used to warp into corresponding section at the

output. OpenCV's Subdiv2D.getTriangleList() function is used to obtain these triangles and stored in array so that it can be used as lookup table for determining the triangles coordinates.



Assumptions

The following assumptions are made to simplify the algorithm:

1. Opening of mouth cannot be animated as this involves inclusion of new details (teeth) which cannot be obtained by using warping interpolation techniques.
2. Background of portrait is not animated. Furthermore, a box around the portrait is only animated
3. Portrait and actor images are cropped to remove unnecessary background details

However, these assumptions can be limited by using more sophisticated algorithm. For instance, actor's mouth itself can be warped into the portrait image to involve details of teeth if initial portrait image has its mouth closed.

Algorithm:

The following are the steps to animate the portrait image:

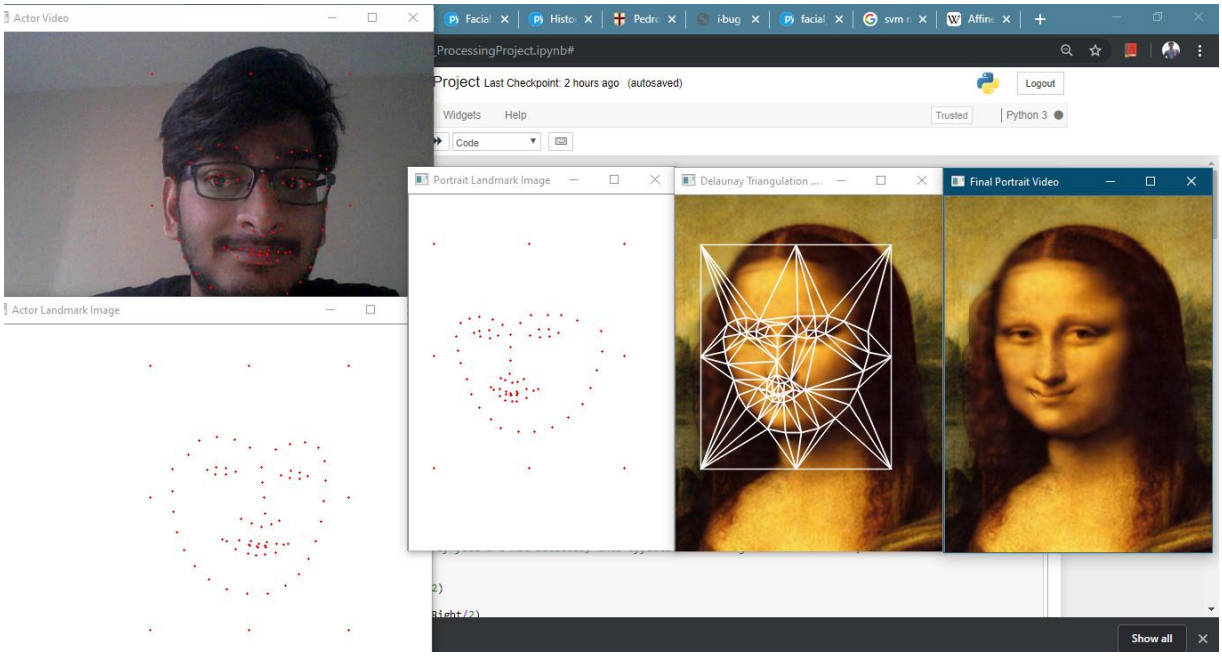
1. Get the portrait image, P_i, and the actor image, A_i; where $A_i = A(t)$ and $P_i = P(t)$ for each frame of a video sequence.
2. Using Face Landmark Detection Library of DLib, extract facial features of P_i and A_i, i.e., eyes, mouth, nose, ear and hairline. The locations of these features are stored in feature vectors P_{iF} and A_{iF} as 68 landmark points.
3. Construct a bounding box around the landmark points of P_{iF} and A_{iF} . Offset the top of box to include hair, right and left of box to include ears and bottom of box to include neck. Add the corners and midpoints of edges of the box to the landmark vector P_{iF} and A_{iF}.

4. Now, the features of another shot of actor image A_{i+1}, typically next frame of a video sequence, is obtained as $A_{(i+1)F}$; where $A_{i+1} = A(t+1)$.
5. The relative distance between the same feature points of A_{iF} and $A_{(i+1)F}$ is calculated and stored in a distance vector $A_{(i+1)D}$ (subject to a minimum tolerance value). For example, if the tip of the nose is a feature point and it moves by a few pixels in the second image from its initial location, the relative distance in x and y direction is computed.
6. Each feature point of P_i corresponding to $A_{(i+1)}$ is translated by a fraction, k, of corresponding $A_{(i+1)D}$ vector i.e. $k*A_{(i+1)D}$ to obtain $P_{(i+1)F}$
7. For each P_{iF} and $P_{(i+1)F}$, Delaunay triangulation is carried out to divide the image into sections. This triangulation produces P_{iT} and $P_{(i+1)T}$. The triangles of $P_{(i+1)T}$ will also be translated and sheared based on the actors feature movements.
8. Now, each triangular section from P_{iT} is affine transformed to corresponding section in $P_{(i+1)T}$ where intensities are interpolated using bilinear interpolation (especially in regions where the transformation stretches the details.)
9. Once all triangular sections are affine transformed, it generates the portrait image P_{i+1} based on actor image movements.
10. The above steps are repeated for other frames of video i.e., A_{i+2}, A_{i+3}, … to generate P_{i+2}, P_{i+3}, … respectively.

Thus, the expected result is an animation of the portrait image mimicking the movement of the actor of a video.

Conclusion

The output of the algorithm is shown below and a video sequence is also attached with the report.



The algorithm is capable of animating portrait image within the box. However, if the actor moves front or back relative to the camera, the distortion becomes significant. Furthermore, new details like teeth cannot be included. Though it cannot achieve the excellent result as in machine learning deep-fake

algorithms, it can be still used for applications which cannot afford expensive computations required by neural networks. However, the algorithm is capable of computing with no user intervention unlike conventional warping techniques in selecting the landmark points manually. Moreover, no data set is given as an input to the algorithm, reducing the effort of data cleaning. Future implementations may include training the landmark points detector with hair and ear features too so that better warping can be achieved. Furthermore, when images turn to right and left, the portrait image loses some information due to warping. To prevent this, algorithm can be modified to take into account of initial portrait image features during each step of warping such that missing information can be interpolated from the initial image.

Thus, an animated sequence of a one-shot portrait image is obtained from a sequence of actor images using warping techniques.

References:

[1] Paul Viola and Michael Jones, *Rapid Object Detection Using a Boosted Cascade of Simple Features*.

[2] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, Victor Lempitsky, *Few-Shot Adversarial Learning of Realistic Neural Talking Head Models*, arXiv:1905.08233 [cs.CV]

[3] Hadar Averbuch-Elor, Daniel Cohen-Or, Johannes Kopf, and Michael F. Cohen, *Bringing Portraits to Life*, 2017. ACM Trans. Graph. 36, 4, Article 196 (November 2017)

[4] Steven M. Seitz, Charles R. Dyer, *View Morphing*, Proc. SIGGRAPH 96

[5] Digital Image Warping, George Wolberg, 1998

[6] Kazemi and Sullivan, *One Millisecond Face Alignment with an Ensemble of Regression Trees*, 2017

[7] iBUG 300-W Dataset, https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

[8] Adrian Rosebrock, https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

[9] Satya Mallick, https://www.learnopencv.com/face-morph-using-opencv-cpp-python/

Appendix

Installing Required Libraries

1. Install Python 3.6 version and pip installer
2. Install CMake latest version for binding C++ DLib libraries with python
3. Run the 'install.bat' file or the following commands (for Windows):
   pip install numpy imutils
   pip install opencv-contrib-python
   pip install
   https://pypi.python.org/packages/da/06/bd3e241c4eb0a662914b3b4875fc52dd176a9db0d4a2c915ac2ad8800e9e/dlib-19.7.0-cp36-cp36m-win_amd64.whl#md5=b7330a5b2d46420343fbed5df69e6a3f

Program Implementation

Program implementation is attached in next page

```
# Import the various libraries
import numpy as np
import imutils
import dlib
import cv2

# This list specifies how each points are connected to form Delaunay triangles. Obtained .
# experimentally.For e.g., [1,36,41] means the landmark point numbers 2,37 and 42 are triangulated.

triangleList = [[1, 36, 41], [36, 1, 0], [1, 71, 0], [71, 1, 2], [0, 71, 17], [31, 2, 29], [2, 31,
       3], [71, 68, 17], [10, 54, 11], [54, 10, 55], [71, 3, 73], [3, 71, 2], [2, 1, 41], [27, 21, 22
       ], [21, 27, 39], [73, 3, 4], [33, 50, 32], [50, 33, 51], [73, 5, 6], [5, 73, 4], [4, 3, 48], [
       31, 29, 30], [73, 7, 74], [7, 73, 6], [5, 4, 48], [58, 62, 57], [62, 58, 61], [6, 5, 59], [40,
        29, 41], [29, 40, 28], [74, 7, 8], [7, 6, 57], [44, 25, 45], [25, 44, 24], [74, 8, 9], [8, 7,
        56], [74, 11, 75], [11, 74, 10], [9, 8, 56], [75, 11, 12], [9, 10, 74], [10, 9, 55], [42, 29,
        28], [29, 42, 35], [30, 34, 33], [34, 30, 35], [72, 75, 13], [12, 11, 54], [40, 37, 38], [37,
        40, 41], [13, 75, 12], [13, 12, 54], [37, 18, 19], [18, 37, 36], [72, 13, 14], [14, 13, 54],
       [37, 19, 20], [70, 72, 16], [14, 15, 72], [15, 14, 45], [36, 17, 18], [17, 36, 0], [16, 72, 15
       ], [16, 15, 26], [69, 24, 23], [24, 69, 25], [17, 68, 18], [42, 22, 23], [22, 42, 27], [18, 68
       , 19], [27, 42, 28], [69, 20, 68], [20, 69, 23], [23, 22, 20], [20, 19, 68], [20, 21, 38], [21
       , 20, 22], [29, 35, 30], [42, 23, 43], [46, 44, 45], [44, 46, 47], [45, 14, 46], [23, 24, 43],
        [70, 25, 69], [25, 70, 26], [26, 15, 45], [25, 26, 45], [16, 26, 70], [29, 2, 41], [27, 28, 3
       9], [49, 32, 50], [32, 49, 31], [34, 52, 33], [52, 34, 53], [32, 31, 30], [48, 3, 31], [6, 59,
        58], [32, 30, 33], [57, 62, 63], [63, 53, 55], [53, 63, 65], [54, 46, 14], [46, 54, 35], [34,
        35, 53], [44, 47, 43], [37, 20, 38], [40, 38, 39], [36, 37, 41], [28, 40, 39], [38, 21, 39],
       [35, 42, 47], [42, 43, 47], [43, 24, 44], [46, 35, 47], [48, 31, 49], [5, 48, 59], [48, 49, 60
       ], [59, 67, 61], [67, 59, 49], [49, 50, 67], [51, 33, 52], [7, 57, 56], [50, 51, 61], [63, 55,
        56], [51, 52, 65], [54, 55, 64], [52, 53, 65], [53, 35, 64], [64, 35, 54], [9, 56, 55], [55,
       53, 64], [62, 65, 63], [65, 62, 51], [61, 58, 59], [56, 57, 63], [57, 6, 58], [49, 59, 60], [5
       9, 48, 60], [50, 61, 67], [61, 51, 62]]
    # This constant specifies the impact of Actor points on the Portrait image i.e., portrait move
    ment is a fraction of movement of actor
    k = 0.3

    # Convert the detected landmark points to a Numpy array
    # Based on imutils.shape_to_np() function by Adrian Rosebrock
    def shape_to_np(shape):
        coords = np.zeros((68, 2), dtype="int")
        for i in range(0, 68):
            coords[i] = (shape.part(i).x, shape.part(i).y)
        return coords
```

```
20.
21.     # In addition to landmark points, a box bounding the landmark points is also required for bett
        er warping to include hair and neck.
22.     # This function adds the corners of bounding box to the landmark points array
23.     def addBoxPoints(imageCopy,arrayToAdd):
24.         # determines the bounding box for all landmark points
25.         x,y,w,h = cv2.boundingRect(np.float32(arrayToAdd))
26.         # Since several parts of face are not detected, this offsets the rectangle to include head
        , ears and neck
27.         headTop = 80
28.         headRight = 50
29.         x1 = x-int(headRight/2)
30.         y1 = y-headTop
31.         x2 = x + w + int(headRight/2)
32.         y2 = y + h + int(headTop/2)
33.         midx = int((x1+x2)/2)
34.         midy = int((y1+y2)/2)
35.         # Creates the offseted points of rectangle
36.         rectPoints = [(x1,y1),(midx,y1),(x2,y1),(x1,midy),(x2,midy),(x1,y2),(midx,y2),(x2,y2)]
37.         # appends the offseted points to the landmark array
38.         for pt in rectPoints:
39.             arrayToAdd.append(list(pt))
40.
41.     # Using the list of Delaunay triangle points, all the landmark points are triangulated
42.     # This function takes three landmark points at a time, triangulates and finds the coordinates
        of the triangle.
43.     # The list of all triangle points is returned
44.     def drawDelaunay(img, delaunay_color, landmarkArray) :
45.         trianglePoints=[]
46.         # triangle list holds the indices of all landmark array points that are to be triangulated

47.         # for e.g., [1,36,41] means the landmark point numbers 2,37 and 42 are triangulated.
48.         for t in triangleList :
49.             # Find the coordinates of triangle corresponding to each landmarkArray index
50.             pt1 = tuple(landmarkArray[t[0]])
51.             pt2 = tuple(landmarkArray[t[1]])
52.             pt3 = tuple(landmarkArray[t[2]])
53.             # draw the triangulations on the copy of image passed
54.             cv2.line(img, pt1, pt2, delaunay_color, 1, cv2.LINE_AA, 0)
55.             cv2.line(img, pt2, pt3, delaunay_color, 1, cv2.LINE_AA, 0)
56.             cv2.line(img, pt3, pt1, delaunay_color, 1, cv2.LINE_AA, 0)
57.             trianglePoints.append([pt1,pt2,pt3])
58.         # return all the triangulated coordinate points
59.         return trianglePoints
60.
61.     # Warps a given triangle from source to triangle at destination
62.     # Since cv2.warpAffine() does not allow triangles to be warped properly, a bounding box for tr
        iangle is used instead to warp
63.     # and then extract the triangular region.
64.     def warpTriangle(src, dest, t1, t2) :
65.         # Creates a bounding box for src and dest image
66.         r1 = cv2.boundingRect(np.float32([t1]))
67.         r2 = cv2.boundingRect(np.float32([t2]))
68.         srcRect = []
69.         destRect = []
70.         # offsets each triangle points as they'll be cropped at the end
71.         for i in range(0, 3):
72.             srcRect.append(((t1[i][0] - r1[0]),(t1[i][1] - r1[1])))
73.             destRect.append(((t2[i][0] - r2[0]),(t2[i][1] - r2[1])))
74.         # create a mask for cropping the warped image
75.         mask = np.zeros((r2[3], r2[2], 3), dtype = np.float32)
76.         cv2.fillConvexPoly(mask, np.int32(destRect), (1.0, 1.0, 1.0), 16, 0)
77.         # crop a subregion of the src image for warping
```

```
        srcCropped = src[r1[1]:r1[1] + r1[3], r1[0]:r1[0] + r1[2]]
        # Apply affine transform from src triangle to dest triangle on the cropped sub-image
        warpImage = affineTransform(srcCropped, srcRect, destRect, (r2[2], r2[3]))
        # Apply mask to dest image and place the warped part over the dest image
        dest[r2[1]:r2[1]+r2[3], r2[0]:r2[0]+r2[2]] = dest[r2[1]:r2[1]+r2[3], r2[0]:r2[0]+r2[2]] *
    (1 - mask) + warpImage * mask

# The input source image is warped such that source triangle is affine transformed to the poin
    ts of destination triangle
# Bilinear interpolation is used for intensities
def affineTransform(src, srcTri, dstTri, size) :
    warp = cv2.getAffineTransform( np.float32(srcTri), np.float32(dstTri) )
    dst = cv2.warpAffine( src, warp, (size[0], size[1]), None, flags=cv2.INTER_LINEAR, borderM
    ode=cv2.BORDER_REFLECT_101 )
    # returns the warped image
    return dst

# Face-detector instantiation
detector = dlib.get_frontal_face_detector()
# Predictor instantiation
# The input file contains the pre-trained network for detecting the 68 landmark points
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# Retrieve the Mona Lisa Portrait Image
portraitImage = cv2.imread("Mona_Lisa.jpg")
# Crop a section of image
portraitImage = portraitImage[200:2200,750:2100]
# Resize the image for lesser resolution (faster computation)
portraitImage = cv2.resize(portraitImage,(300,400))
# Blur the image to decrease the details
portraitImage = cv2.blur(portraitImage, (3,3))
# Keep a copy of original image
origPortrait = portraitImage.copy()
# Convert to grayscale
grayPortrait = cv2.cvtColor(portraitImage, cv2.COLOR_BGR2GRAY)
# Apply face-detection on gray image and find the box for the detected face part
rects = detector(grayPortrait, 1)
# Find the landmark points for the region detected by face-detection
facepPts = predictor(grayPortrait, rects[0])
# convert the detected array to a numpy array
facepPts = shape_to_np(facepPts)

portraitLandmarkArr = [] # Holds the landmark points of portrait image
# Create a white background image to show the landmark points of portrait
portraitLandmark = np.zeros((grayPortrait.shape[0], grayPortrait.shape[1],3), dtype="uint8")
portraitLandmark.fill(255)
# Append all the landmark points detected to the portrait landmark array
for (x, y) in facepPts:
    portraitLandmarkArr.append([x,y])
# Adds additional bounding box points to landmark array
addBoxPoints(portraitImage.copy(),portraitLandmarkArr)
# Triangulates the portrait image using landmark points
prevTrianglePts = drawDelaunay(portraitImage.copy(), (255, 255, 255), portraitLandmarkArr)
# Open the web-cam
cap= cv2.VideoCapture(0)
# Initial Flag. To skip first step in warping as no movement in actor image is achieved at thi
    s point
initFlag = 0

# Until the web-cam is opened
while(cap.isOpened()):
    # Read the captured frame
    ret, frame = cap.read()
```

```
        # Return from loop if no frame is detected
        if ret == False:
            print('returned')
            break
        # Resize the frame for lesser computations
        frame = cv2.resize(frame, (500,400))
        # convert to grayscale image
        grayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Create a white background image to show the landmark points of actor
        actorLandmarkImg = np.zeros((grayFrame.shape[0], grayFrame.shape[1],3), dtype="uint8")
        actorLandmarkImg.fill(255)
        # Apply face-detection on gray actor image and find the box for the detected face part
        rects = detector(grayFrame, 1)
        # if no face is detected, skip the loop
        if len(rects) == 0:
            continue
        # Find the landmark points for the region detected by face-detection
        facePts = predictor(grayFrame, rects[0])
        # convert the detected array to a numpy array
        facePts = shape_to_np(facePts)
        actorLandmarkArr = [] # Holds the landmark points of actor image
        # create a copy of the portrait landmark image
        copyPortraitLandmark = portraitLandmark.copy()
        # Append all the landmark points detected to the actor landmark array
        for (x, y) in facePts:
            actorLandmarkArr.append([x,y])
        # Adds additional bounding box points to landmark array
        addBoxPoints(frame.copy(),actorLandmarkArr)

        # for all the landmark points
        for i in range(0,len(portraitLandmarkArr)):
            # Moves the portrait landmark points a fraction (k) of the actor landmark points movem
    ent
            if initFlag == 1: # Skips first step as no movement will be achieved
                portraitLandmarkArr[i][0] += int(k*(actorLandmarkArr[i][0] - prevActorLandmarkArr[
    i][0]))
                portraitLandmarkArr[i][1] += int(k*(actorLandmarkArr[i][1] - prevActorLandmarkArr[
    i][1]))
            # plot the updated landmark Points on corresponding images
            cv2.circle(frame, tuple(actorLandmarkArr[i]), 1, (0, 0, 255), -1)
            cv2.circle(actorLandmarkImg, tuple(actorLandmarkArr[i]), 1, (0, 0, 255), -1)
            cv2.circle(copyPortraitLandmark, tuple(portraitLandmarkArr[i]), 1, (0, 0, 255), -1)

        prevActorLandmarkArr = actorLandmarkArr # holds previous actor landmark locations
        # Skip displaying images if it is the first loop and set the intial flag to 1.
        if initFlag == 0:
            initFlag = 1
            continue

        copyPortraitTriangles = portraitImage.copy()
        # Triangulates the portrait image using landmark points. This is a new triangulation to be
     warped
        newTrianglePts = drawDelaunay(copyPortraitTriangles, (255, 255, 255),portraitLandmarkArr)

        prevPortraitImg = portraitImage.copy()
        # Warp the previous portrait image from each old to new triangular point coordinates
        for t1,t2 in zip(prevTrianglePts,newTrianglePts):
            warpTriangle(prevPortraitImg, portraitImage, t1, t2)
        prevTrianglePts = newTrianglePts # store the list of old triangular points

        # Display all the images in sequence to form a video
        cv2.imshow("Actor Video", frame)
        cv2.imshow("Actor Landmark Image",actorLandmarkImg)
```

```
195.         cv2.imshow("Portrait Landmark Image",copyPortraitLandmark)
196.         cv2.imshow("Delaunay Triangulation of Portrait Image",copyPortraitTriangles)
197.         cv2.imshow("Final Portrait Video",portraitImage)
198.
199.         # Run the loop till space key is hit
200.         keyPress = cv2.waitKey(5)&0xFF
201.         if keyPress == 32:
202.             # Release all the frames and web-cam when space key is hit
203.             cap.release()
204.             cv2.destroyAllWindows()
205.             break
```