BLG453E
Homework-2

30.11.2020

Res. Asst. Yusuf H. Sahin
sahinyu@itu.edu.tr

Shamans who are angry with each other transform into bears, oxen and fight benefiting from many things (...). A male and a female shaman started one of their rituals. The female one asked "In which form should we test each other's strength?". "Fish form", answered the male one. The female one transformed into a fish form and jumped into the sea. The male one followed the same procedure and got to the female one in haste. The male one started to swallow the female one gasping her tail as a starter. However, she was not fitting for his mouth. After he slipped her out of his mouth, she jumped back just behind him. She grasped and swallowed him greedily, flew back to her home in the form of a seabird (...). She vomited the remains of the shaman out and threw them away.

Turkic Shaman Texts, Fuzuli Bayat

Kaç yaşında adamım be, beni kedi kılığına soktun, maymun gibi dolaştırıyorsun.

Aydemir Akbaş

- You should write all your code in Python language.
- Cheating is highly discouraged. If you are planning to use different libraries or functions, please ask me about it.
- Ninova only stores files under 20 MB. If you could not upload your results, you can share them with me via Dropbox, or send me private YouTube video links for each part's results.
- You can use additional photos, however all photos should be in same size.

1 - Part 1: Showing the landmark points (20 pts.)

In this homework, we will simulate a shapeshift from human to some animals and other people. To do this, we will use three human faces and three animal faces. To align the faces with each other, we will use the facial landmark points given in Figure 1.

To find human facial landmarks, we will use *dlib*[1] library. Dlib uses a very powerful face

[1] http://dlib.net/

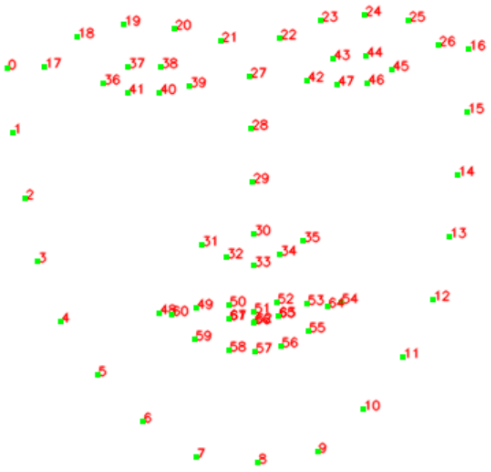

Figure 1: Facial landmarks and their ID correspondences

predictor[2] which uses a histogram of gradients (HOG) based model. By default, it outputs 68 landmark points from a face. An example skeleton code to find the landmarks is given below.

```
detector = dlib.get_frontal_face_detector()
#The detector object is used to detect the faces given in an image. It
    works generally better than OpenCV's default face detector.
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
#To predict the landmark points given a face image, a shape predictor with
    a ready-to-use model is created . The model can be found under "http://
    dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2"

image = cv2.imread("input1.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
#The predictor only works on grayscale images.

rectangles = detector(gray)
#Use detector to find a list of rectangles containing the faces in the
    image. The rectangles are represented by their xy coordinates.

#Check whether there is only one rectangle. Then, draw the rectangle on the
     image. To reach the values of the rectangle you can use functions
    given in the official documentation (http://dlib.net/python/index.html#
    dlib.rectangle). E.g. rectangles[0].bl_corner().x and rectangles[0].
    bl_corner().y will give one of the points. Show your work.
```

[2]C. Sagonas, E. Antonakos, G, Tzimiropoulos, S. Zafeiriou, M. Pantic. 2016. 300 faces In-the-wild challenge: Database and results. Image and Vision Computing (IMAVIS), Special Issue on Facial Landmark Localisation "In-The-Wild".

```
15
points = predictor(gray, rectangles[0])
17 #Points is a special structure which stores all the 68 points. By using
    points.part(i), we can reach ith landmark point. You should mark every
    point on the image. Show your work.
```

I already tagged the facial landmarks for three animals: gorilla, panda and cat. They can be found as numpy files.

In this part of the homework, you will obtain the figure in Figure 2.

Figure 2: Faces with placed landmarks

2 - Part 2: Delaunay Triangulation (20 pts.)

In Delaunay Triangulation, triangles are created using points from a set. These triangles are formed so that, there are no intersection between any of them. Avoiding to delve deep into the theory of Delaunay Triangulation, we will benefit from the built-in functions of OpenCV for this task. You can start from the following skeleton code.

```
image = cv2.imread("image.png")
2 subdiv  = cv2.Subdiv2D((0,0,image.shape[0],image.shape[1]))
# Subdiv2D is an OpenCV object which performs Delaunay triangulation.
4
for i in range(68):
6     subdiv.insert(...) #Each landmark point should be insterted into
    Subdiv2D object as a tuple. Show your work here.
```

```
    #Add the points to another list to check again.
8
10 subdiv.insert((0,0))
   subdiv.insert((0,image.shape[1]-1))
12 ...#Also to cover the whole image, 8 points from the edges should be
       inserted. Show your work here. Thus, you should have a total of 76
       points.

14 triangles = subdiv.getTriangleList()
```

Here *triangles* is a matrix of (142, 6)[3]. Each row has xy positions $\{x_1, y_1, x_2, y_2, x_3, y_3\}$ of a triangle.

To make a face morphing effect between two images, we should match the triangles in both images. **At this point, if we use a new Delaunay triangulation in second image, the triangles of the two images will not be related.** Thus, to create the triangles of the second image;

- Find which IDs are used in which *triangles*.
- Create the triangles between these IDs for the second image.

After finding the triangles of both images, draw the triangles as given in Figure 3.

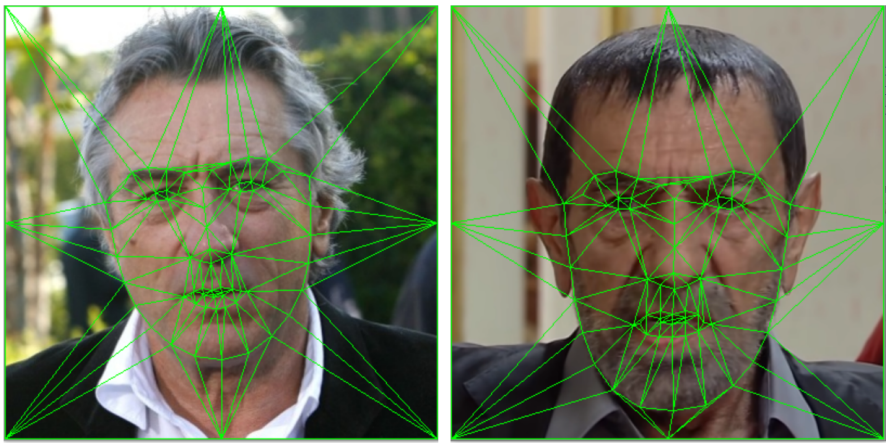

Figure 3: Delaunay triangles placed on the faces.

3 - Part 3: Face Morphing (60 pts.)

In this part, you will use the following code to do the face morphing. The code uses **img1_triangles** and **img2_triangles** matrices which may be obtained in the previous

[3]Row count can be changed according to point locations.

step. However in your report, you should write explanations for all lines tagged with a letter.

```
img1_triangles = img1_triangles[:,[1,0,3,2,5,4]]
img2_triangles = img2_triangles[:,[1,0,3,2,5,4]]

Transforms = np.zeros((len(img1_triangles), 3, 3))
for i in range(len(img1_triangles)):
    source = img1_triangles[i]
    target = img2_triangles[i]
    Transforms[i] = calc_transform(source, target) #(A)

morphs = []
for t in np.arange(0, 1.0001, 0.02): #(B)
    print("processing:\t", t*100, "%")
    morphs.append(image_morph(image, cat_img, img1_triangles,
    img2_triangles, Transforms, t)[:, :, ::-1])
```

```
def make_homogeneous(triangle):

    homogeneous = np.array([triangle[::2],   \
                            triangle[1::2], \
                            [1, 1, 1]]) #(C)
    return homogeneous

def calc_transform(triangle1, triangle2):
    source = make_homogeneous(triangle1).T
    target = triangle2
    Mtx = np.array([np.concatenate((source[0], np.zeros(3))), \
                    np.concatenate((np.zeros(3), source[0])), \
                    np.concatenate((source[1], np.zeros(3))), \
                    np.concatenate((np.zeros(3), source[1])), \
                    np.concatenate((source[2], np.zeros(3))), \
                    np.concatenate((np.zeros(3), source[2]))]) #(D)

    coefs = np.matmul(np.linalg.pinv(Mtx), target) #(E)

    Transform = np.array([coefs[:3], coefs[3:], [0, 0, 1]]) #(F)

    return Transform

def vectorised_Bilinear(coordinates, target_img, size):
    coordinates[0] = np.clip(coordinates[0], 0, size[0]-1)
    coordinates[1] = np.clip(coordinates[1], 0, size[1]-1)
    lower = np.floor(coordinates).astype(np.uint32)
    upper = np.ceil(coordinates).astype(np.uint32)

    error = coordinates - lower
    resindual = 1 - error

    top_left = np.multiply(np.multiply(resindual[0], resindual[1]).reshape(
    coordinates.shape[1],1), target_img[lower[0], lower[1], :])
```

```
    top_right = np.multiply(np.multiply(resindual[0], error[1]).reshape(
    coordinates.shape[1],1), target_img[lower[0], upper[1], :])
35  bot_left = np.multiply(np.multiply(error[0], resindual[1]).reshape(
    coordinates.shape[1],1), target_img[upper[0], lower[1], :])
    bot_right = np.multiply(np.multiply(error[0], error[1]).reshape(
    coordinates.shape[1],1), target_img[upper[0], upper[1], :]) #(G)
37
    return np.uint8(np.round(top_left + top_right + bot_left + bot_right))
    #(H)
39
def image_morph(image1, image2, triangles1, triangles2, transforms, t):
41
    inter_image_1 = np.zeros(image1.shape).astype(np.uint8)
43  inter_image_2 = np.zeros(image2.shape).astype(np.uint8)

45  for i in range(len(transforms)):

47      homo_inter_tri = (1 - t)*make_homogeneous(triangles1[i]) + t*
    make_homogeneous(triangles2[i]) #(I)

49      polygon_mask = np.zeros(image1.shape[:2], dtype=np.uint8)
        cv2.fillPoly(polygon_mask, [np.int32(np.round(homo_inter_tri[1::-1,
     :].T))], color=255) #(J)
51
        seg = np.where(polygon_mask == 255) #(K)
53
        mask_points = np.vstack((seg[0], seg[1], np.ones(len(seg[0]))))#(L)
55
        inter_tri = homo_inter_tri[:2].flatten(order="F") #(M)
57
        inter_to_img1 = calc_transform(inter_tri, triangles1[i])
59      inter_to_img2 = calc_transform(inter_tri, triangles2[i])

61      mapped_to_img1 = np.matmul(inter_to_img1, mask_points)[:-1] #(N)
        mapped_to_img2 = np.matmul(inter_to_img2, mask_points)[:-1]
63
        inter_image_1[seg[0], seg[1], :] = vectorised_Bilinear(
    mapped_to_img1, image1, inter_image_1.shape) #(O)
65      inter_image_2[seg[0], seg[1], :] = vectorised_Bilinear(
    mapped_to_img2, image2, inter_image_2.shape)

67  result = (1 - t)*inter_image_1 + t*inter_image_2 #(P)

69  return result.astype(np.uint8)
```

Using the images given with the homework document create at least three face morphing videos. Some examples are given in my website[4].

[4] https://web.itu.edu.tr/sahinyu/hw3_panda.mp4, https://web.itu.edu.tr/sahinyu/hw3_aydemir.mp4