ISTANBUL TECHNICAL UNIVERSITY

Faculty of Computer Science and Informatics

BLG453E

Computer Vision Homework-2 Report

Author

Cansu Yanık -150170704

NOTE: Output images, videos and gifs can be access via the Drive and Dropbox link below. I would recommend to open **Drive** link instead of dropbox to watch gifs.

https://drive.google.com/open?id=1dpu2Ep210n_oeF0NVQFsPYpMVTgFjS_H

https://www.dropbox.com/sh/0fyhsd5pbvnjfs6/AAC7LK3-7eUcQuyZgCa64psla?dl=0

1. Part 1: Showing the landmark points

For the human faces, I chose the Dennis Ritchie and our assistant Yusuf Hüseyin Şahin; for the cat face, I chose the white cat given on the far right in the homework description (file name is '*00000095_001.jpg*').

Note: I read cat image file from the *CAT_00* folder.

Since it is necessary for all three photos to be same size, I resized the human photos according to cat sizes.

```
#Loads cat image and the points
catImage = cv2.imread("./CAT_00/00000095_001.jpg")

file = open("./CAT_00/00000095_001.jpg.cat")
content = file.read()
file.close()

content = content.split()               #parse according to space
content = [int(i) for i in content]     #convert string values to integer


#Loads images and calls the function to put face landmarks on them
firstImage = cv2.imread("./dennis_ritchie.jpg")
secondImage = cv2.imread("./yusuf.jpg")
rows, cols, ch = catImage.shape

#Since it is necessary for all three photos to be same size, I resized the photos
firstImage = cv2.resize(firstImage,(cols,rows))
secondImage = cv2.resize(secondImage,(cols,rows))
```

Figure-1: Loading the images, parsing cat points and resizing the human images

Since cat points are given in one line, I parsed and stored them into *content* array and converted them into integer because they are given as string values. In *.cat file*, there are 9 point pairs. Each pair has x and y values respectively. I read each pair and draw it on cat image. To do this operation, *cv2.rectangle()* method is used. Starting and end points, color and thickness are given as parameters.

Now cat points are ready to place on cat image.

```
#Places points on the cat image
index = 1
for i in range (0, content[0]):
    x = content[index]
    y = content[index+1]
    index = index + 2
    catImage = cv2.rectangle(catImage, (x, y), (x + 5, y + 5), (0, 255, 0), -1)
```

Figure-2: Putting cat points on cat image

To place face landmarks on human faces, I write a function called *putLandmarks(image).* In this function, ***dlib*** library methods are used to detect faces and the find landmarks points. A detector object is used to detect the faces given in image *(dlib.get_frontal_face_detector()).* By using a ready-to-use model, landmark points are predicted. *(dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")).* A list of rectangles is found by using detector. (detector(gray)). To show the face on the image, I draw a rectangle. To reach the values of the rectangle, I used *left(), top(), right() and bottom()* functions. And to store 68 facial landmarks I used points structure. Every point of the landmarks is marked on the image by using *cv2.rectangle()* method.

```
def putLandmarks(image):

    detector = dlib.get_frontal_face_detector()
    #Used to detect face
    predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
    #Predit landmark points

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  #Predictor works only grayscale images

    rectangles = detector(gray)  #Finds list of rectangles

    #To be able to draw a rectangle to show face, I need to find rectangle coordinates
    #I used below methods
    x = rectangles[0].left()
    y = rectangles[0].top()
    w = rectangles[0].right() - x
    h = rectangles[0].bottom() - y

    #Draws a rectangle to show the face
    image = cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 1)

    #Create points structure to store 68 landmarks
    points = predictor(gray, rectangles[0])

    #Draws all 68 landmarks on the faces
    for i in range(0, 68):
        x = points.part(i).x
        y = points.part(i).y

        image = cv2.rectangle(image, (x, y), (x + 5, y + 5), (0, 255, 0), -1)

    return image
```

Figure-3: Function for places face landmarks on human faces

```
firstImage = putLandmarks(firstImage)
secondImage = putLandmarks(secondImage)
```

Figure-4: Function is called for images

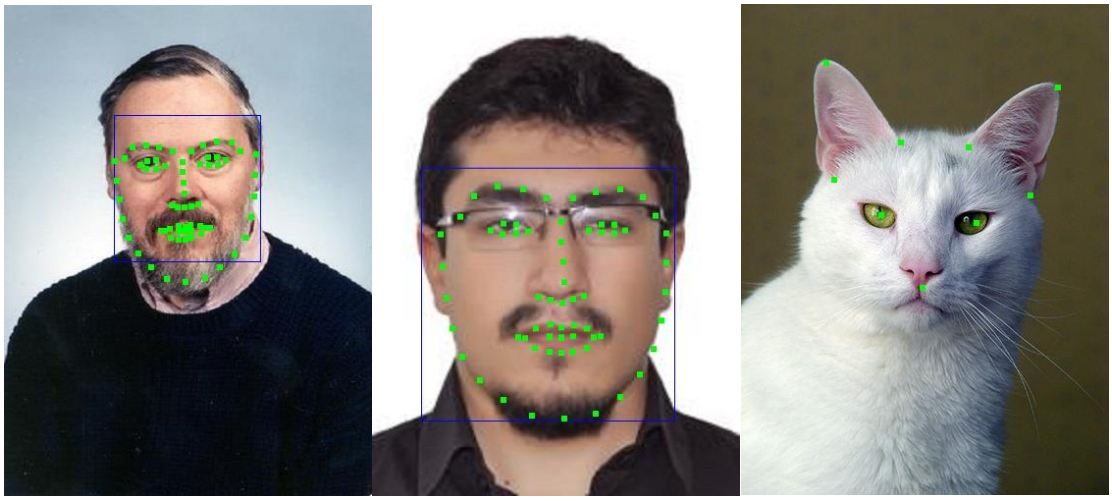

Figure-5: Outputs of the part1

2. Part 2: “Cat your life!”

In this part, I wrote a function called *catLandmarks()*. Because I decided to use this function in part3 and part5. In this function:

- Cat image file, *.cat file* and the *“template_points.npy”* file are loaded
- Content of .cat file is parsed and saved (same operation with part1)
- Points of cat are converted into integer from string
- 9 points are marked on cat image. (I wrote same operations which I did in part1 because I did not want to use part1, for next parts, I wanted to use part2.)
- Template points are taken from given file
- Since left and right eyes range ids are given to us in homework description, I found left and right eye of the template
- I found distance between the template eyes
- I found distance between cat eyes (Since we have 9 points, we know the eyes coordinates).
- I found horizontal ratio by using distance between the template eyes and the distance between cat eyes

```
leftEye_x = int((templatePoints[36][0] + templatePoints[39][0])/2)      #Finds left eye x coordinate
leftEye_y = int((templatePoints[36][1] + templatePoints[39][1])/2)      #Finds left eye y coordinate

rightEye_x = int((templatePoints[42][0] + templatePoints[45][0])/2)     #Finds right eye x coordinate
rightEye_y = int((templatePoints[42][1] + templatePoints[45][1])/2)     #Finds right eye y coordinate

distanceEyes_template = math.sqrt((rightEye_x - leftEye_x)**2 + (rightEye_y - leftEye_y)**2)    #Distance between template eyes
distanceEyes_cat = math.sqrt((content[1] - content[3])**2 + (content[2] - content[4])**2)       #Distance between cat eyes

horizontalRatio = distanceEyes_template/distanceEyes_cat
```

Figure-6: Operation for finding horizontal ratio

To find vertical ratio

- Found midpoint between template eyes
- Found distance between midpoint of template eyes and the template mouth point
- Found midpoint between cat eyes
- Found distance between midpoint of cat eyes and the cat mouth point

- Found horizontal ratio by using distances

```
midEyes_template_x = int((leftEye_x + rightEye_x)/2)        #Finds middle x coordinate between template eyes
midEyes_template_y = int((leftEye_y + rightEye_y)/2)        #Finds middle y coordinate between template eyes

#Distance between midpoint of eyes and the mouth
distanceMouth_template = math.sqrt((midEyes_template_x - templatePoints[66][0])**2 + (midEyes_template_y - templatePoints[66][1])**2)


midEyes_cat_x = int((content[1] + content[3])/2)        #Finds middle x coordinate between cat eyes
midEyes_cat_y = int((content[2] + content[4])/2)        #Finds middle y coordinate between cat eyes

#Distance between midpoint of eyes and the mouth
distanceMouth_cat = math.sqrt((midEyes_cat_x - content[5])**2 + (midEyes_cat_y - content[6])**2)

verticalRatio = distanceMouth_template/distanceMouth_cat
```

Figure-7: Operation for finding vertical ratio

- Applied the ratios to template points to scale them.
- In this part, to move the landmarks on the cat, I used affine transformation. To use affine transformation, I needed three points as references. Therefore, I selected left eyes, right eyes and the mouth.
- By using *cv2.getAffineTransform(pts2,pts1)* method, transformation matrix is found. Then, since I want to apply matrix to points, I used matrix multiplication *(np.matmul())*
- **Note:** To be able to place points on cat image adequately, I applied affine transformation two times and also I needed to shift points on the cat image.
- I saved the points into arrays
- By using *cv2.rectangle()* method, I drew the points.
- See Figure-8 for code.

```
for i in range(0, 68):          #Apply ratios to each point
    x = int(templatePoints[i][0] * horizontalRatio)
    y = int(templatePoints[i][1] * verticalRatio)

    templatePoints[i][0] = x
    templatePoints[i][1] = y

#Finds new left and right eyes to use for transformation matrix (To be able to locate point I apply transformation two times)
for a in range(2):

    #Finds new left and right eyes
    left_x = int((templatePoints[36][0] + templatePoints[39][0])/2)
    left_y = int((templatePoints[36][1] + templatePoints[39][1])/2)

    right_x = int((templatePoints[42][0] + templatePoints[45][0])/2)
    right_y = int((templatePoints[42][1] + templatePoints[45][1])/2)

    #Reference points
    pts1 = np.float32([[content[1],content[2]],[content[3],content[4]],[content[5],content[6]]])
    pts2 = np.float32([[left_x,left_y],[right_x,right_y],[templatePoints[66][0],templatePoints[66][1]]])

    #Transformation matrix
    M = cv2.getAffineTransform(pts2,pts1)

    #Apply transformation matris to templete points
    templatePoints = np.matmul(templatePoints[:,[0,1]], M)

catLandmark_x = []
catLandmark_y = []

#To be able to locate points on cat face, I need to shift points and save them
for i in range(0, 68):
    x = int(templatePoints[i][0] + 45)
    y = int(templatePoints[i][1] + 75)

    catLandmark_x.append(x)
    catLandmark_y.append(y)

    catImage = cv2.rectangle(catImage, (x, y), (x + 5, y + 5), (255, 0, 0), -1)
```

Figure-8: Placing and drawing the landmark points on cat image

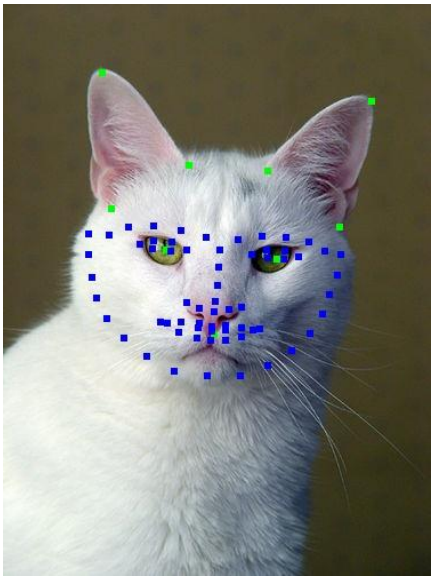

Figure-9: Output of the part2

3. Part 3: Delaunay Triangulation

In this part, I wrote three functions:

- ***landmarks(image):***
  In this function, I found face landmarks for human images. This operation was done previously in the part1 but since I did not use part1 for next parts, I wrote this operation again.

```
#Loads images
catImage = cv2.imread("./CAT_00/00000095_001.jpg")
rows, cols, ch = catImage.shape

firstImage = cv2.imread("./dennis_ritchie.jpg")
secondImage = cv2.imread("./yusuf.jpg")

#Since it is necessary for all three photos to be same size, I resized the photos
firstImage = cv2.resize(firstImage,(cols,rows))
secondImage = cv2.resize(secondImage,(cols,rows))


#This function finds the face landmarks for images
def landmarks(image):

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

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    rectangles = detector(gray)

    points = predictor(gray, rectangles[0])

    return points
```

Figure-10: Finding face landmarks

- ***subdivPoints (image, landmarks_x, landmarks_y):***
  In this operation, Delaunay Triangulation is performed. Subdiv2D object is created. Firstly, I inserted all landmark points into object. Then I inserted image corner points and the midpoints of the edges. To find these points, I take the row and column values of the image. By using ***subdiv.getTriangleList()*** method, a full list of triangles is obtained.

```
def subdivPoints (image, landmarks_x, landmarks_y):

    #Performs Delaunay Triangulation
    subdiv = cv2.Subdiv2D((0,0,image.shape[1]+1, image.shape[0]+1))

    #Insert landmark points
    for i in range(0, 68):
        subdiv.insert((landmarks_x[i],landmarks_y[i]))


    rows, cols, ch = image.shape

    #Also insert corners and the midpoints of the edges
    subdiv.insert((0,0))
    subdiv.insert((0, rows/2))
    subdiv.insert((cols/2, 0))
    subdiv.insert((cols-1, 0))
    subdiv.insert((cols-1, rows/2))
    subdiv.insert((0, rows-1))
    subdiv.insert((cols/2, rows-1))
    subdiv.insert((cols-1, rows-1))

    #Obtains full list of triangles
    triangles = subdiv.getTriangleList()

    return triangles
```

Figure-11: Delaunay Triangulation Operation

- ***drawLines (triangles, image):***
  This function is used for draw triangles. To do this, I used ***cv2.line()*** method.

```
#Draw triangles
def drawLines (triangles, image):
    for i in range(len(triangles)):
        sel_triangle = triangles[i].astype(np.int)


        for points in sel_triangle:
            point1 = (sel_triangle[0], sel_triangle[1])
            point2 = (sel_triangle[2], sel_triangle[3])
            point3 = (sel_triangle[4], sel_triangle[5])

            cv2.line(image, point1, point2, (0, 255, 0), 1)
            cv2.line(image, point2, point3, (0, 255, 0), 1)
            cv2.line(image, point1, point3, (0, 255, 0), 1)
```

Figure-12: Drawing triangles

The function calls for each images are as follows.

```
landmarkPoints = landmarks(firstImage)
landmarks_x = []
landmarks_y = []

#I save the landmark points x and y coordinates separately
for i in range(0, 68):
    landmarks_x.append(landmarkPoints.part(i).x)
    landmarks_y.append(landmarkPoints.part(i).y)

#Find and draw triangles
triangles_1 = subdivPoints(firstImage, landmarks_x, landmarks_y)
drawLines(triangles_1, firstImage)


landmarkPoints = landmarks(secondImage)
landmarks_x = []
landmarks_y = []

for i in range(0, 68):
    landmarks_x.append(landmarkPoints.part(i).x)
    landmarks_y.append(landmarkPoints.part(i).y)

triangles_2 = subdivPoints(secondImage, landmarks_x, landmarks_y)
drawLines(triangles_2, secondImage)


#Calls function from part2 to take landmark points of cat
catLandmark_x, catLandmark_y = catLandmarks()

triangles_3 = subdivPoints(catImage, catLandmark_x, catLandmark_y)
drawLines(triangles_3, catImage)
```

Figure-13: Function calls for images

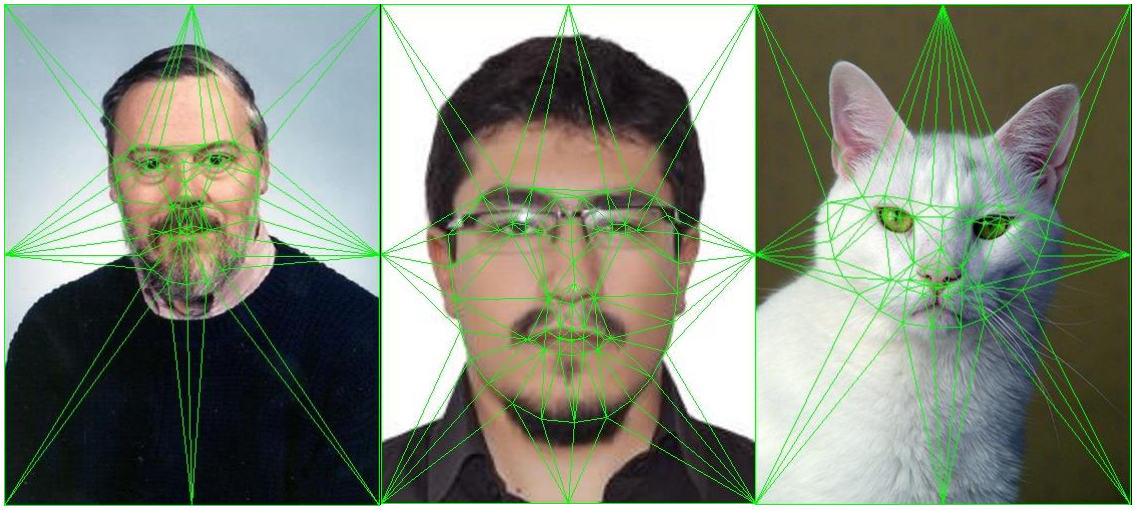

Figure-14: Outputs of the part3

4. Part 4: Triangle Animation

In this part,

- I created two empty images to display source and target triangles
- I defined source triangle vertices
- I defined target triangle vertices
- To show the source and target image, I drew them by using *cv2.polylines()* and *cv2.fillPoly()* methods.
-

```
image = np.zeros((500,375,3), np.uint8)        #Create image to display source triangle
image2 = np.zeros((500,375,3), np.uint8)       #Create image to display target triangle

rows, cols, ch = image.shape

#Source triangle vertices
source_points = [[100, 200], [200, 50], [50, 100]]
source_points = np.int32([source_points])

#Target triangle vertices
target_points = [[300, 350], [350, 250], [150, 200]]
target_points = np.int32([target_points])

#Source image
cv2.polylines(image, [source_points], isClosed=True, color=(0,0,255), thickness=1)
cv2.fillPoly(image, [source_points], color=(0,0,255))

#Target image
cv2.polylines(image2, [target_points], isClosed=True, color=(255,0,0), thickness=1)
cv2.fillPoly(image2, [target_points], color=(255,0,0))
```

Figure-15: Source and target triangles

- To do morphing operation, I wanted to create an in between image by blending source and target images. To blend images, I used a parameter which is *alpha.* Alpha controls the blending and it takes between 0 and 1.

$$M(x,y) = (1-\alpha)I(x,y) + \alpha J(x,y)$$

$$x_m = (1-\alpha)x_i + \alpha x_j$$
$$y_m = (1-\alpha)y_i + \alpha y_j$$

In each step, I changed the alpha and created a new morphed triangle. By using morphed triangle, I applied affine transform and I found transformation matrixes. By using matrix multiplication, I applied transformation matrixes to points. Therefore, I obtained points with the new locations. Then, I drew new triangle on an empty image and saved the image to create an animation (gif).

```
images_list = []    #Holds video frames

stepNumber = 20

for i in range(0,stepNumber):

    #In each step, I find a ratio(alpha) which is used to find morhed image and to change the intensity between source and target images
    alpha = i/(stepNumber-1)

    xm = []
    ym = []

    #Each step, calculate the location (xm, ym) of the pixel in the morphed image
    for i in range(0, 3):
        x = (1-alpha)*source_points[0][i][0] + alpha*target_points[0][i][0]
        y = (1-alpha)*source_points[0][i][1] + alpha*target_points[0][i][1]

        xm.append(x)
        ym.append(y)

    #Points of source and target image
    pts1 = np.float32([[source_points[0][0][0],source_points[0][0][1]],[source_points[0][1][0],source_points[0][1][1]],[source_points[0][2][0],source_points[0][2][1]]])
    pts2 = np.float32([[target_points[0][0][0],target_points[0][0][1]],[target_points[0][1][0],target_points[0][1][1]],[target_points[0][2][0],target_points[0][2][1]]])


    #Converts morphed image array to proper type to apply affine transform
    morphed = [[xm[0], ym[0]], [xm[1], ym[1]], [xm[2], ym[2]]]
    morphed =  np.float32(morphed)

    #Apply affine transform to both source and target images
    transformationMatrix = cv2.getAffineTransform(pts1,morphed)
    warp1 = np.matmul(pts1, transformationMatrix)

    transformationMatrix = cv2.getAffineTransform(pts2,morphed)
    warp2 = np.matmul(pts2, transformationMatrix)

    #Changes the intensity between source and target warp images according to alpha
    warpImage = (1.0 - alpha) * warp1 + alpha * warp2

    image3 = np.zeros((500,375,3), np.uint8)        #Creates new image to save new triangle

    points = [[warpImage[0][0], warpImage[0][1]], [warpImage[1][0], warpImage[1][1]], [warpImage[2][0], warpImage[2][1]]]
    points = np.int32([points])

    #Draw new triangle, also use alpha for changing the color
    cv2.polylines(image3, [points], isClosed=True, color=(0+alpha*255,0,255-alpha*255), thickness=1)
    cv2.fillPoly(image3, [points], color=(0+alpha*255,0,255-alpha*255))


    images_list.append(image3)      #Saves the image to use for video
```

Figure-16: Morphing operation and saving new triangle

Then using ***movipy*** library, I created a gif.

```
#Since moviepy works with RGB images, I converted them into RGB
for i in range(len(images_list)):
    images_list[i] = cv2.cvtColor(images_list[i], cv2.COLOR_BGR2RGB)

#Create gif
clip = mpy.ImageSequenceClip(sequence=images_list, fps=25)
clip.write_gif('part4_gif.gif')
```

Figure-17: Creating an animation

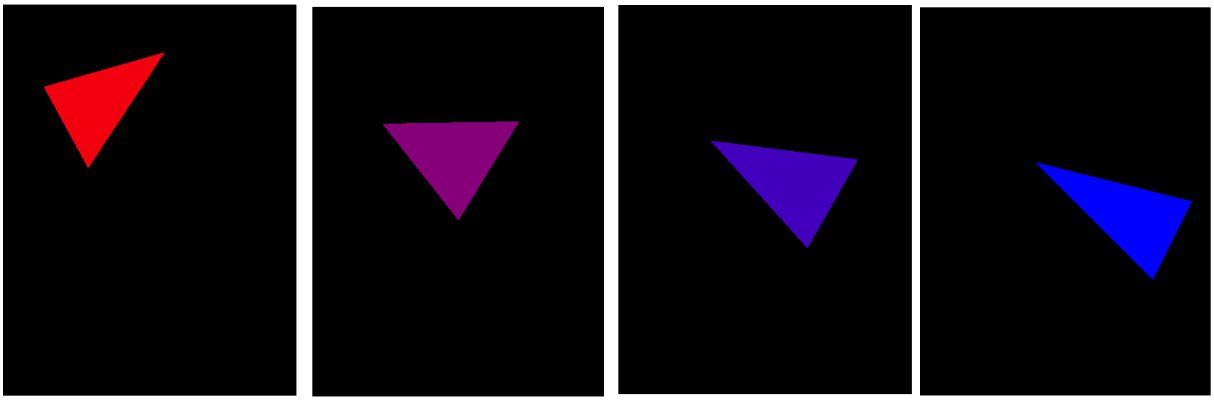

Figure-18: Some frames from the animation

5. Part 5: Face Morphing

In this part, I wrote a function called ***morphing (landmarks_x_1, landmarks_y_1, landmarks_x_2, landmarks_y_2, triangles, firstImage, secondImage, limit, alpha):***

This function takes landmarks points of images (separate arrays for x and y coordinates), an array contains triangles (these triangles are belonging to second image because I used these triangles to find corresponding ids for other image), images, limit (limit specifies the step number) and alpha value for blending images and creating morphed image.

- Firstly, by using alpha value, I found morphed image which is an in between image of target and source images
- Since landmark arrays have 68 points, not other 8 points (corners and midpoints of edges) I appended 8 points to landmark point arrays for first, second and morphed image

```
#Loads images
catImage = cv2.imread("./CAT_00/00000095_001.jpg")
rows, cols, ch = catImage.shape

firstImage = cv2.imread("./dennis_ritchie.jpg")
secondImage = cv2.imread("./yusuf.jpg")

#Since it is necessary for all three photos to be same size, I resized the photos
firstImage = cv2.resize(firstImage,(cols,rows))
secondImage = cv2.resize(secondImage,(cols,rows))


def morphing(landmarks_x_1, landmarks_y_1, landmarks_x_2, landmarks_y_2, triangles, firstImage, secondImage, limit, alpha):

    #The amount of blending is controlled by alpha
    #To be able to do face morphing, I need a morphed image
    #xm and ym holds the coordinates of morphed image

    xm = []
    ym = []

    for i in range(0, 68):
        x = (1-alpha)*landmarks_x_2[i] + alpha*landmarks_x_1[i]
        y = (1-alpha)*landmarks_y_2[i] + alpha*landmarks_y_1[i]

        xm.append(x)
        ym.append(y)
```

Figure-19: Finding morphed image

```
#Also the corners and the midpoints of the edges are needed to be saved
landmarks_x_2.append(0)
landmarks_y_2.append(0)

landmarks_x_2.append(0)
landmarks_y_2.append(rows/2)

landmarks_x_2.append(int(cols/2))
landmarks_y_2.append(0)

landmarks_x_2.append(cols-1)
landmarks_y_2.append(0)

landmarks_x_2.append(cols-1)
landmarks_y_2.append(rows/2)

landmarks_x_2.append(0)
landmarks_y_2.append(rows-1)

landmarks_x_2.append(int(cols/2))
landmarks_y_2.append(rows-1)

landmarks_x_2.append(cols-1)
landmarks_y_2.append(rows-1)
```

```
xm.append(0)
ym.append(0)

xm.append(0)
ym.append(rows/2)

xm.append(cols/2)
ym.append(0)

xm.append(cols-1)
ym.append(0)

xm.append(cols-1)
ym.append(rows/2)

xm.append(0)
ym.append(rows-1)

xm.append(cols/2)
ym.append(rows-1)

xm.append(cols-1)
ym.append(rows-1)
```

```
landmarks_x_1.append(0)
landmarks_y_1.append(0)

landmarks_x_1.append(0)
landmarks_y_1.append(rows/2)

landmarks_x_1.append(int(cols/2))
landmarks_y_1.append(0)

landmarks_x_1.append(cols-1)
landmarks_y_1.append(0)

landmarks_x_1.append(cols-1)
landmarks_y_1.append(rows/2)

landmarks_x_1.append(0)
landmarks_y_1.append(rows-1)

landmarks_x_1.append(int(cols/2))
landmarks_y_1.append(rows-1)

landmarks_x_1.append(cols-1)
landmarks_y_1.append(rows-1)
```

Figure-20: Adding other 8 points to landmark arrays

- Then using reference triangle array belongs to second image, for each vertices of the current triangle, I found point ids which match to points of first image and saved them to use face morphing. Thus, I can reach the matched triangles for both images. Ids are saved into an indexArray

```
#I also send triangle array to this function as a parameter. This triangles are using as reference,
#By using these, I find vertices' ids and save these ids to be able to find correct vertices from other image
#Thus I can receive matched triangles in both two images
indexes_triangles = []
index = 0
index1 = 0
index2 = 0
index3 = 0

#Traverse all triangles
for i in range(len(triangles)):
    sel_triangle = triangles[i].astype(np.int)

    for points in sel_triangle:

        if (index % 6 == 0):
            for a in range(76):
                #now landmark arrays have 68+8 points because corners are also included
                #traverse all landmark points and try to find matched one
                #and save the id, do this operation for all three vertices
                if ((sel_triangle[0] == landmarks_x_2[a]) and (sel_triangle[1] == landmarks_y_2[a])):
                    index1 = a
                    break

            for b in range(76):

                if ((sel_triangle[2] == landmarks_x_2[b]) and (sel_triangle[3] == landmarks_y_2[b])):
                    index2 = b
                    break

            for c in range(76):

                if ((sel_triangle[4] == landmarks_x_2[c]) and (sel_triangle[5] == landmarks_y_2[c])):
                    index3 = c
                    break
            indexes_triangles.append([index1, index2, index3])
        index = index + 1
```

Figure-21: Finding points ids

- Now, I am ready to do face morphing, firstly I created an empty image to put transformed triangles on it.
- Traverse the triangles and takes id numbers of current triangle vertices from our indexArray.
- By using these indexes, find point values from image1 and morphed image
- Now, I have matched triangles. Since *warpAffine* function takes an image and not a triangle, I found bounding boxes for the triangles.
- I created a mask image to extract the desired area from image.
- Then, extracted the bounding boxes from images.
- Since I have bounding boxes, I need to find corner points to use for affine transformation. I found them by taking left top vertices as reference points and saves them.
- I filled boxes according to points
- Applied affine transformation to find transformation matrix to both source and target boxes. (Destination points are from morphed image).
- I applied transformation matrixes to source and target boxes.
- I did alpha blending. Thus when alpha is 0, image will be first image, when it is 1, image will be second image.
- Put transformed boxes into my result empty image. See Figure-22 for code.

```
faceMorphing = np.zeros(secondImage.shape, dtype = secondImage.dtype)

#Ids are found now I am ready to do face morphing between matched triangles
for i in range(len(triangles)):
    #takes the three vertices
    vertices1 = indexes_triangles[i][0]
    vertices2 = indexes_triangles[i][1]
    vertices3 = indexes_triangles[i][2]

    sel_triangle = triangles[i].astype(np.int)

    image1 = [[sel_triangle[0], sel_triangle[1]], [sel_triangle[2], sel_triangle[3]], [sel_triangle[4], sel_triangle[5]]]
    image2 = [[landmarks_x_1[vertices1], landmarks_y_1[vertices1]], [landmarks_x_1[vertices2], landmarks_y_1[vertices2]], [landmarks_x_1[vertices3], landmarks_y_1[vertices3]]]
    morhed = [[xm[vertices1], ym[vertices1]], [xm[vertices2], ym[vertices2]], [xm[vertices3], ym[vertices3]]]

    #Since warpAffine takes an image and not a triangle, I find bounding boxes for the triangles
    bounding1 = cv2.boundingRect(np.float32([image1]))
    bounding2 = cv2.boundingRect(np.float32([image2]))
    boundingM = cv2.boundingRect(np.float32([morhed]))

    row = boundingM[3]
    col = boundingM[2]
    #create a mask image to extract the desired area from image
    maskImage = np.zeros((row, col, 3), dtype = np.float32)

    #extract the bounding boxes from images
    image_1 = secondImage[bounding1[1]:bounding1[1] + bounding1[3], bounding1[0]:bounding1[0] + bounding1[2]]
    image_2 = firstImage[bounding2[1]:bounding2[1] + bounding2[3], bounding2[0]:bounding2[0] + bounding2[2]]


    #Saves reference points
    image1_ref = []
    image2_ref = []
    morp_ref = []


    for k in range(0, 3):
        #Takes left top vertices as reference points and saves them
        left_x_1 = image1[k][0] - bounding1[0]
        left_y_1 = image1[k][1] - bounding1[1]

        left_x_2 = image2[k][0] - bounding2[0]
        left_y_2 = image2[k][1] - bounding2[1]

        left_x_m = morhed[k][0] - boundingM[0]
        left_y_m = morhed[k][1] - boundingM[1]

        image1_ref.append((left_x_1,left_y_1))
        image2_ref.append((left_x_2,left_y_2))
        morp_ref.append((left_x_m,left_y_m))

    #fills the box
    cv2.fillConvexPoly(maskImage, np.int32(morp_ref), (1,1,1));


    #Apply affine transform to find transformation matrix
    transformationMatrix1 = cv2.getAffineTransform( np.float32(image1_ref), np.float32(morp_ref))
    transformationMatrix2 = cv2.getAffineTransform( np.float32(image2_ref), np.float32(morp_ref))

    #Apply transformation matrixes to points
    warp1 = cv2.warpAffine( image_1, transformationMatrix1, (col,row), None, flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT_101 )
    warp2 = cv2.warpAffine( image_2, transformationMatrix2, (col,row), None, flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT_101 )


    #Do alpha blending
    warpImage = (1.0 - alpha) * warp1 + alpha * warp2
    warpImage = warpImage * maskImage

    #Put the warpImage on the correct position of the image
    faceMorphing[boundingM[1]:boundingM[1]+row, boundingM[0]:boundingM[0]+col] = faceMorphing[boundingM[1]:boundingM[1]+row, boundingM[0]:boundingM[0]+col]* ( 1 - maskImage )
    faceMorphing[boundingM[1]:boundingM[1]+row, boundingM[0]:boundingM[0]+col] = faceMorphing[boundingM[1]:boundingM[1]+row, boundingM[0]:boundingM[0]+col] + warpImage


return faceMorphing
```

Figure-22: Morphing Operation

Before, calling morphing function, I needed to call ***landmarks(firstImage)*** function from ***part3.py*** file to find landmark points. Then x and y values are separated to two arrays and ***subdivPoints(firstImage, landmarks_x_1, landmarks_y_1)*** function is called from ***part3.py*** file to obtain triangles. See Figure-23 for code.

```
#Calls function in the part3 to take image landmarks
#and calls subdivPoints function in the part3 to take triangles
landmarkPoints = landmarks(firstImage)
landmarks_x_1 = []
landmarks_y_1 = []

for i in range(0, 68):
    landmarks_x_1.append(landmarkPoints.part(i).x)
    landmarks_y_1.append(landmarkPoints.part(i).y)

triangles_1 = subdivPoints(firstImage, landmarks_x_1, landmarks_y_1)


#Calls function in the part3 to take image landmarks
#and calls subdivPoints function in the part3 to take triangles
landmarkPoints = landmarks(secondImage)
landmarks_x_2 = []
landmarks_y_2 = []

for i in range(0, 68):
    landmarks_x_2.append(landmarkPoints.part(i).x)
    landmarks_y_2.append(landmarkPoints.part(i).y)

triangles_2 = subdivPoints(secondImage, landmarks_x_2, landmarks_y_2)

#Calls function in the part2 to take cat landmarks
#and calls subdivPoints function in the part3 to take triangles
catLandmark_x, catLandmark_y = catLandmarks()
triangles_3 = subdivPoints(catImage, catLandmark_x, catLandmark_y)
```

Figure-23: Calling landmark and subdivPoints function from part3.py

Since morphing operation will be done step by step (I did this operation in 50 steps), I wrote a for loop. In each step, I changed the alpha value and called my morphing function. I saved the resulting image to use for video frames.

```
#Saves video frames
imagesList1 = []
imagesList2 = []

imagesList3 = []
imagesList4 = []

imagesList5 = []
imagesList6 = []

#I decided to do image morphing in 50 steps
stepNumber = 50
for limit in range(0, stepNumber):
    #Each step find a ratio and send as a parameter
    t=limit/(stepNumber-1)

    #Sends face landmarks, second image's triangles, images and the limit to function
    img1 = morphing(landmarks_x_1, landmarks_y_1, landmarks_x_2, landmarks_y_2, triangles_2, firstImage, secondImage, limit,t)
    imagesList1.append(img1)

    img2= morphing(landmarks_x_2, landmarks_y_2, landmarks_x_1, landmarks_y_1, triangles_1, secondImage, firstImage, limit,t)
    imagesList2.append(img2)

    img3 = morphing(catLandmark_x, catLandmark_y, landmarks_x_1, landmarks_y_1, triangles_1, catImage, firstImage, limit,t)
    imagesList3.append(img3)

    img4 = morphing(landmarks_x_1, landmarks_y_1, catLandmark_x, catLandmark_y, triangles_3, firstImage, catImage, limit,t)
    imagesList4.append(img4)

    img5 = morphing(catLandmark_x, catLandmark_y, landmarks_x_2, landmarks_y_2, triangles_2, catImage, secondImage, limit,t)
    imagesList5.append(img5)

    img6 = morphing(landmarks_x_2, landmarks_y_2, catLandmark_x, catLandmark_y, triangles_3, secondImage, catImage, limit,t)
    imagesList6.append(img6)
```

Figure-24: Morphing operation is done in 50 steps

I created videos by using image lists. Also I wanted to create a gif version.

Note: To open videos, I would recommend to use *VLC media player*. (Actually, I prefer to use opencv video option, but since you asked us to use moviepy, I used moviepy, but I cannot open videos on windows' default option, but I can open them on VLC media player. I hope, it would be okay. Therefore, I also added gif versions which are okay for any platform for me)

```
#Since moviepy works with RGB images, I converted them into RGB
for i in range(len(imagesList1)):
    imagesList1[i] = cv2.cvtColor(imagesList1[i], cv2.COLOR_BGR2RGB)
    imagesList2[i] = cv2.cvtColor(imagesList2[i], cv2.COLOR_BGR2RGB)
    imagesList3[i] = cv2.cvtColor(imagesList3[i], cv2.COLOR_BGR2RGB)
    imagesList4[i] = cv2.cvtColor(imagesList4[i], cv2.COLOR_BGR2RGB)
    imagesList5[i] = cv2.cvtColor(imagesList5[i], cv2.COLOR_BGR2RGB)
    imagesList6[i] = cv2.cvtColor(imagesList6[i], cv2.COLOR_BGR2RGB)


#Creates videos
clip = mpy.ImageSequenceClip(imagesList1, fps=25)
clip.write_videofile("part5_video1.mp4")

clip = mpy.ImageSequenceClip(imagesList2, fps=25)
clip.write_videofile("part5_video2.mp4")

clip = mpy.ImageSequenceClip(imagesList3, fps=25)
clip.write_videofile("part5_video3.mp4")

clip = mpy.ImageSequenceClip(imagesList4, fps=25)
clip.write_videofile("part5_video4.mp4")

clip = mpy.ImageSequenceClip(imagesList5, fps=25)
clip.write_videofile("part5_video5.mp4")

clip = mpy.ImageSequenceClip(imagesList6, fps=25)
clip.write_videofile("part5_video6.mp4")



#Creates gifs
imagesList1 = imagesList1 + imagesList2
imagesList3 = imagesList3 + imagesList4
imagesList5 = imagesList5 + imagesList6

clip = mpy.ImageSequenceClip(sequence=imagesList1, fps=10)
clip.write_gif('part5_gif1.gif')

clip = mpy.ImageSequenceClip(sequence=imagesList3, fps=10)
clip.write_gif('part5_gif2.gif')

clip = mpy.ImageSequenceClip(sequence=imagesList5, fps=10)
clip.write_gif('part5_gif3.gif')
```

Figure-25: Creating videos and gifs

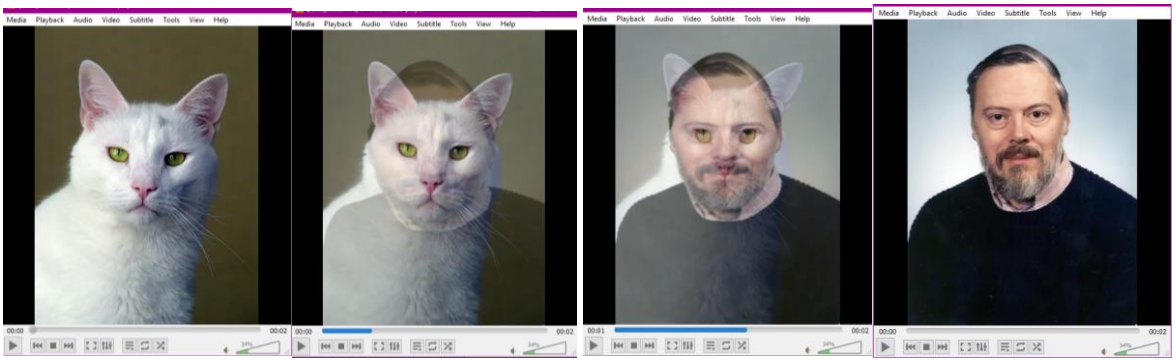

Figure-26: Same frames from video