Assignment #3

Yue Yu

October 8, 2017

C. MORPH VIDEO

1. Determine Image Correspondences

To obtain corresponding points, dlib (http://dlib.net/) was used. For each image, dlib detects 68 facial feature points in order (as Figure 1 shows). I added the corners of the image and half way points between those corners as corresponding points as well.

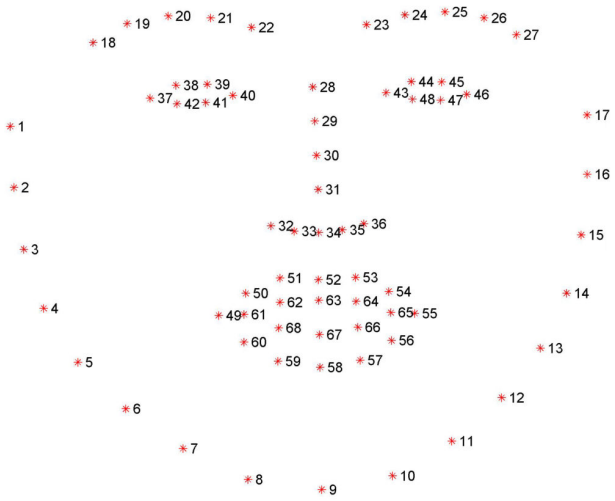

Figure 1: Visualizing the 68 facial landmark coordinates.

2. Compute a Triangulation

Delauney Triangulation was applied to the corresponding points acquired from the previous step. The result of Delaunay triangulation is a list of triangles represented by the indices of points in the 76 points array.

3. Locally Warp and Align Images

We first find the locations of all 76 points (x_m, y_m) in the morphed image based on the source (x_i, y_i) and target (x_j, y_j) images, using Equation 1.

$$\begin{aligned} x_m &= (1-\alpha)x_i + \alpha x_j \\ y_m &= (1-\alpha)y_i + \alpha y_j \end{aligned} \tag{1}$$

Then we can find the corresponding triangle in the morphed image for each triangle in source image and calculate the affine transform that maps the three corners of the triangle in source image to the three corners of the corresponding triangle in the morphed image. Finally, repeat the process for target image and the morphed image.

$$\begin{bmatrix} t_1 & t_2 & t_3 \\ t_4 & t_5 & t_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ 1 & 1 & 1 \end{bmatrix} \tag{2}$$

or

$$\mathbf{TA} = \mathbf{X} \tag{3}$$

so

$$\mathbf{T} = \mathbf{X}\mathbf{A}^{-1} \tag{4}$$

where $\mathbf{T}$ is the affine transformation matrix we want.
For each triangle in source image, use the affine transform calculated in the previous step to transform all pixels inside the triangle to the morphed image. Similarly, obtain a warped version for target image.

4. Color Interpolating

Blend the warped version of source and target images, we obtained in the previous step, using Equation 5, we get our final morphed image.

$$M(x_m, y_m) = (1-\alpha)I(x_i, y_i) + \alpha J(x_j, y_j) \tag{5}$$

In this task we produced 58 intermediate representations for each source and target image pair by setting α to $\{1, 2, ..., 57, 58\}/59$.

5. Save the Image Sequence as a Video

The program was run on all the images provided under the alphabetical order of the filenames. The whole image sequence is then saved as a video file by calling OpenCV.

The output video can be viewed here `https://d.iyuyue.net/share/users/yue/out.mp4` or `https://www.youtube.com/watch?v=P5Dvpz-RBVg`.

```python
import sys, cv2, dlib, os, math, scipy.ndimage
import numpy as np
from scipy.spatial import Delaunay
import matplotlib.pyplot as plt
from PIL import Image

def shape_to_np(shape, dtype="int"):

    # initialize the list of (x, y)-coordinates
    coords = np.zeros((68, 2), dtype=dtype)

    # loop over the 68 facial landmarks and convert them
    # to a 2-tuple of (x, y)-coordinates
    for i in range(0, 68):
        coords[i] = (shape.part(i).x, shape.part(i).y)

    # return the list of (x, y)-coordinates
    return coords

def triangulation(landmarks):

    tri = Delaunay(landmarks)
    # The result is a list of triangles represented by the
       indices of landmark points
    return tri

def plot(path, landmarks):

    # Apply Delaunay Triangulation
    tri = triangulation(landmarks)

    if tri is not None:
        # if len(landmarks) == 0:
        #     print("No face found in", path)
        # else:
        #     landmarks = stasm.force_points_into_image(
           landmarks, img)
        #     for point in landmarks:
        #         img[int(round(point[1]))][int(round(point[0])
           )] = 0

        # cv2.imshow("stasm minimal", img)
        # cv2.waitKey(0)

```

```
42          # Read original image in RGB
43          img = cv2.imread(path, cv2.IMREAD_COLOR)
44          img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
45          # add image to the plot
46          plt.imshow(img)
47          # add triangles to the plot
48          plt.triplot(landmarks[:,0], landmarks[:,1], tri.
               simplices.copy(), 'w-')
49          # add landmarks to the plot
50          plt.plot(landmarks[:,0], landmarks[:,1], 'm.')
51          # show the plot and close it
52          plt.show()
53          # plt.savefig(os.path.splitext(path)[0]+'_tri'+os.path.
               splitext(path)[1])
54          plt.close()
55      else:
56          print("Delaunay Triangulation failed on", path)
57
58  def landmarkPredictor(path, detector, predictor):
59
60      # Read original image in grayscale
61      img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
62
63      if img is None:
64          print("Cannot load", path)
65          raise SystemExit
66
67      ## Use stasm to predict landmark
68      # landmarks = stasm.search_single(img)
69
70      ## Use dlib to predict landmark
71      # detect faces in the grayscale image
72      rects = detector(img, 1)
73
74      if not rects:
75          return None
76      # Only predict the landmarks for the first face
77      landmarks = predictor(img, rects[0])
78      landmarks = shape_to_np(landmarks)
79
80      return landmarks
81
82  def image2landmarks(imageDir):
83
84      # directory of images
```

```
85      directory = os.fsencode(imageDir)
86
87      # dlib initializing
88      detector = dlib.get_frontal_face_detector()
89      predictor = dlib.shape_predictor('
           shape_predictor_68_face_landmarks.dat')
90
91      # Go through everything in the directory
92      for file in os.listdir(directory):
93
94          filename = os.fsdecode(file)
95
96          if filename.endswith(".jpg") or filename.endswith(".png
               "):
97
98              # path to the image file
99              path = imageDir+filename
100             # get landmarks from predictor
101             landmarks = landmarkPredictor(path, detector,
                   predictor)
102
103             if landmarks is not None:
104                 f = open(path, 'rb')
105                 width, height = Image.open(f).size
106                 width = width - 1
107                 height = height - 1
108
109                 # Add the corners of the image and half way
                       points between those corners as
                       corresponding points as well
110                 landmarks = np.append(landmarks, [[0, 0], [0,
                       math.floor(width/2)], [0, width], [math.
                       floor(height/2), 0], [math.floor(height/2),
                       width], [height, 0], [height, math.floor(
                       width/2)], [height, width]], axis=0)
111
112                 # save the landmarks to seperated txt files
113                 np.savetxt(os.path.splitext(path)[0]+'.txt',
                       landmarks, fmt='%d')
114             else:
115                 print("Landmark Prediction failed on", path)
116             continue
117         else:
118             continue
119
```

```
120 # Visualize the result of Triangulation
121 def plotTriangulation(imageDir):
122
123     directory = os.fsencode(imageDir)
124
125     for file in os.listdir(directory):
126
127         filename = os.fsdecode(file)
128
129         if filename.endswith(".jpg") or filename.endswith(".png
              "):
130
131             # path to the image file
132             path = imageDir+filename
133             # load landmarks from txt files
134             landmarks = np.loadtxt(os.path.splitext(path)[0]+'.
                  txt')
135             # print(landmarks)
136             if landmarks is not None:
137                 plot(path, landmarks)
138             else:
139                 print("Landmark Prediction failed on", path)
140             continue
141         else:
142             continue
143
144 # Calculate the affine transformation matrix for each triangle
      pair
145 def getAffineMatrice(src_tri, dest_tri):
146
147     amat = np.dot(src_tri, np.linalg.inv(dest_tri))[:2, :]
148     return amat
149
150 def warp_to(src_img, src_landmarks, dest_landmarks):
151
152     width, height, colors = src_img.shape
153
154     # initialize the array to store the resulted image
155     result_img = np.zeros((height, width, 3), np.uint8)
156
157     # construct an array of all coordinates in the resulted
          image
158     coor = np.asarray([(x, y) for y in range(0, height)
159                        for x in range(0, width)], np.uint8)
160
```

```
161     # triangulation on morphed image
162     dest_tri = triangulation(dest_landmarks)
163 
164     ones = [1, 1, 1]
165 
166     # find the triangle each coordinate, in the resulted image,
            belongs to
167     tri_index = dest_tri.find_simplex(coor)
168 
169     # go through each triangle
170     for i in range(len(dest_tri.simplices)):
171         # triangle indices
172         tri = dest_tri.simplices[i]
173         # points within the triangle
174         points = coor[tri_index == i]
175         # number of points within the triangle
176         count = len(points)
177         # get affine matrice for this pair of triangle
178         amat = getAffineMatrice(np.vstack((src_landmarks[tri,
                :].T, ones)), np.vstack((dest_landmarks[tri, :].T,
                ones)))
179         # use the affine matrice to transform every pixel
                inside the triangle to the morphed image
180         morphed = np.dot(amat, np.vstack((points.T, np.ones(
                count))))
181         # coordinates in the original image
182         x, y = points.T
183         # coordinates in the morphed image
184         u, v = np.int32(morphed)
185         # copy the color of pixels
186         result_img[y, x] = src_img[v, u]
187 
188     return result_img
189 
190 def warp_images(src_path, dest_path):
191 
192     # saving path
193     save_path = os.path.splitext(src_path)[0]+'_'+filename(
            dest_path)+'/'
194     save_ext = os.path.splitext(src_path)[1]
195 
196     # number of images to generate, including two original
            images and morphed images
197     frames = 60
198 
```

```
199     if not os.path.exists(save_path):
200
201         os.makedirs(save_path)
202
203         # read in images
204         src_img = scipy.ndimage.imread(src_path)[:, :, :3]
205         dest_img = scipy.ndimage.imread(dest_path)[:, :, :3]
206
207         # predict landmarks for both images
208         src_landmarks = np.loadtxt(os.path.splitext(src_path)
               [0]+'.txt')
209         dest_landmarks = np.loadtxt(os.path.splitext(dest_path)
               [0]+'.txt')
210
211         # save the first image (original one)
212         scipy.misc.imsave(save_path+int2filename(0)+save_ext,
               src_img)
213
214         for x in range(1, frames-1):
215             alpha = x/(frames-1)
216
217             # landmarks for the morphed image
218             morphed_landmarks = (src_landmarks * (1 - alpha) +
                   dest_landmarks * alpha)
219
220             s2m = warp_to(src_img, src_landmarks,
                   morphed_landmarks)
221             d2m = warp_to(dest_img, dest_landmarks,
                   morphed_landmarks)
222
223             # blend two warped images into one
224             morphed_img = np.uint8(np.mean(np.array([s2m *  (1
                   - alpha), d2m * alpha]), axis=(0)) * 2)
225
226             # save the morphed image
227             scipy.misc.imsave(save_path+int2filename(x)+
                   save_ext, morphed_img)
228
229         # save the last image (original one)
230         scipy.misc.imsave(save_path+int2filename(frames)+
               save_ext, dest_img)
231
232     else:
233         print(save_path, " already exists. Skip this one.")
234
```

```
235     return save_path
236 
237 # http://docs.opencv.org/3.1.0/dd/d43/tutorial_py_video_display
    .html
238 def images2video(dir_path, save_path):
239 
240     output = save_path + "out.mp4"
241     images = []
242 
243     if isinstance(dir_path, list):
244         for d in dir_path:
245             for f in os.listdir(d):
246                 if f.endswith(".jpg") or f.endswith(".png"):
247                     images.append(os.path.join(d, f))
248     else:
249         for f in os.listdir(dir_path):
250             if f.endswith(".jpg") or f.endswith(".png"):
251                 images.append(os.path.join(dir_path, f))
252 
253     # Determine the width and height from the first image
254     frame = cv2.imread(images[0])
255     cv2.imshow('video',frame)
256     height, width, channels = frame.shape
257 
258     # Define the codec and create VideoWriter object
259     fourcc = cv2.VideoWriter_fourcc(*'mp4v') # Be sure to use
        lower case
260     out = cv2.VideoWriter(output, fourcc, 20.0, (width, height)
        )
261 
262     for image in images:
263 
264         frame = cv2.imread(image)
265 
266         out.write(frame) # Write out frame to video
267 
268         cv2.imshow('video',frame)
269         if (cv2.waitKey(1) & 0xFF) == ord('q'): # Hit `q` to
            exit
270             break
271 
272     # Release everything if job is finished
273     out.release()
274     cv2.destroyAllWindows()
275 
```

```
276 def int2filename(n):
277     return '{0:06d}'.format(n)
278 
279 def filename(path):
280     return os.path.splitext(os.path.basename(path))[0]
281 
282 def main(argv):
283 
284     imageDir = './csFaculty/'
285     # image2landmarks(imageDir)
286     # plotTriangulation(imageDir)
287 
288     mList = []
289     previousFile = ''
290 
291     # Go through every images to make a video
292     for f in os.listdir(imageDir):
293         if f.endswith(".jpg") or f.endswith(".png"):
294             if previousFile:
295                 mList.append(warp_images(imageDir+previousFile,
                        imageDir+f))
296             previousFile = f
297 
298     images2video(mList, imageDir)
299 
300     # imgdir = warp_images(imageDir+'bala_0.jpg', imageDir+'
           mahe.jpg')
301     # images2video(imgdir)
302 
303     # plt.imshow(img)
304     # plt.show()
305 
306     # img = cv2.imread(imageDir+'mahe.jpg', cv2.IMREAD_COLOR)
307     # img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
308     # plt.imshow(img)
309     # plt.show()
310 
311 if __name__ == "__main__":
312     # execute only if run as a script
313     main(sys.argv)
```