Istanbul Technical University
Faculty of Computer and Informatics
Computer Engineering Department

BLG 453E
Computer Vision
Homework III

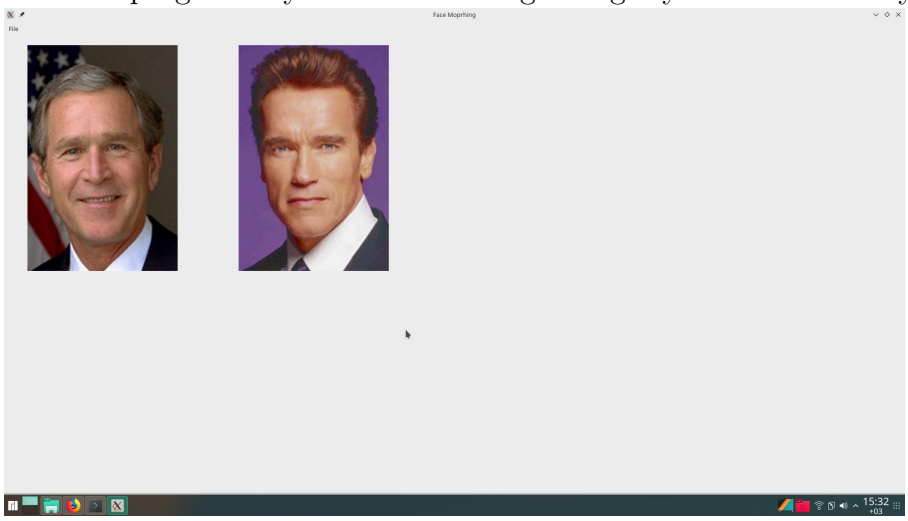
Uğur Uysal - 150140012

Dec 09st, 2018

Contents

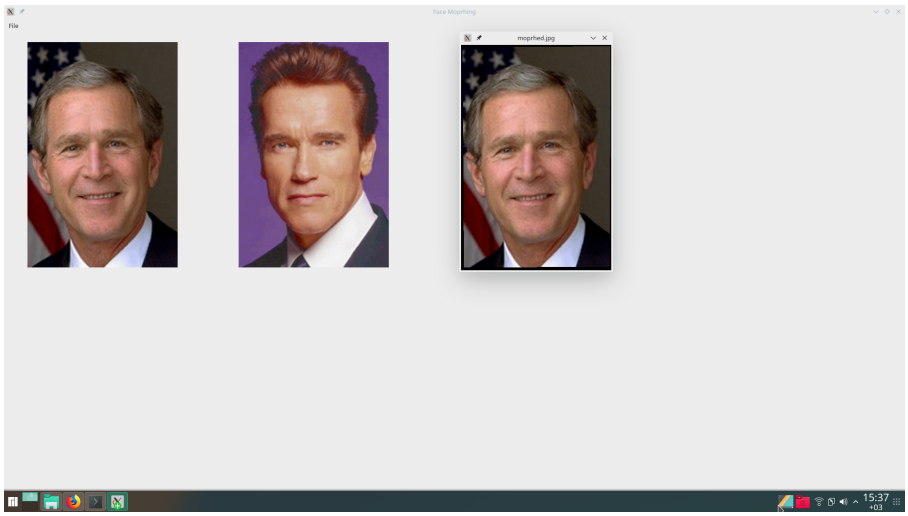
1 Usage

The code depends on numpy, opencv and PyQt5. If all requirements are satisfied, then code can be executed. After running the code you would see an GUI like these (File paths are hard coded in program if you want to change images you should modify the first line of program)



Then you click on images but you must click the points with an order(corresponding landmark points must be clicked in same order) For example first click Bush eyes, after you must click on Arnold eyes

After clicking enoguh points. then click Moprh in the File. You will see a pop up like this. Program also saves the trianguled images and final result in same path.



2 Implementation

I begin my implementation with inspecting code given link in the PDF of homework. Then I created UI with pyqt5. It is two labels with images of Bush and Arnold. The file paths are hard coded at the beginning of the code. The mousePressEvent is listened with a method and the points are saved to an array. The landmarks must be chosen with same order for both

faces.

Code 1: Hard Coded file paths

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from copy import deepcopy
import sys
import cv2
import numpy as np
from PIL import Image

srcfilename = 'Bush.jpg'
targetfilename = 'Arnie.jpg'
    # Events are for taking mouseclicks from user
    def getPixelSrc(self, event):
        x = event.pos().x()
        y = event.pos().y()
        self.src_points.append((int(x), int(y)))
```

Code 2: Event for mouse clicks

```
self.source_label.mousePressEvent = self.getPixelSrc

def getPixelSrc(self, event):
    x = event.pos().x()
    y = event.pos().y()
    self.src_points.append((int(x), int(y)))
```

After choosing points when user clicks on morph at menu code goes to morph method in UI class. It calculates the weighted averages for coordinates for result image with given alpha. Then help of subdiv class we can calculate triangles then we can start to warping. At this point I have encountered with several issues. There is another section for it.

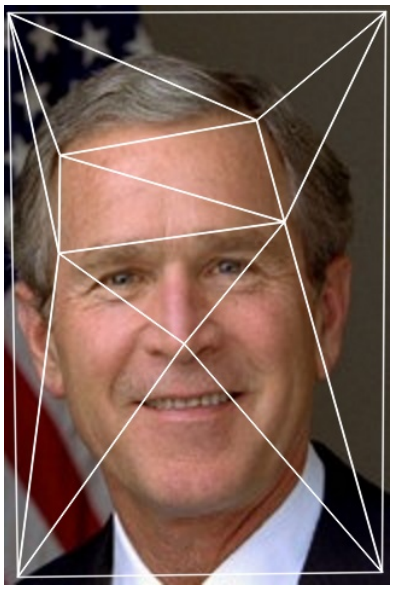
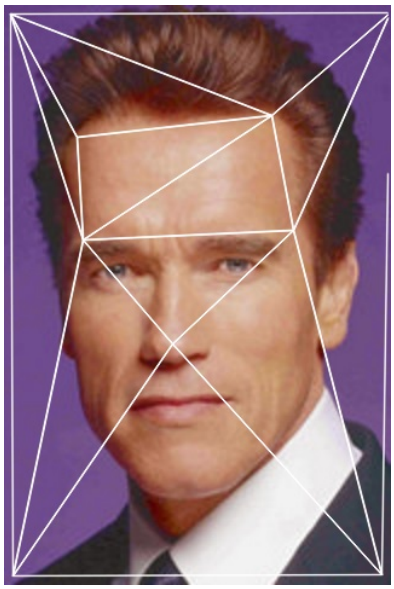
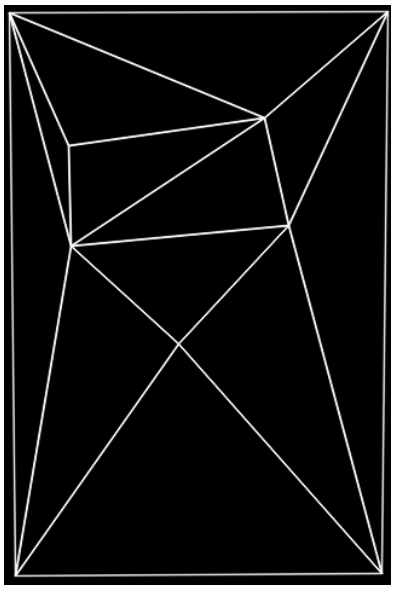
Code 3: Subdiv inserting and get list of triangles

```
points = []
alpha = 0.5
# Compute weighted average point coordinates
for i in range(len(src_pts)):
    x = (1 - alpha) * src_pts[i][0] + alpha * target_pts[i][0]
    y = (1 - alpha) * src_pts[i][1] + alpha * target_pts[i][1]
    points.append((x, y))

subdiv_src = cv2.Subdiv2D(rect)
subdiv_target = cv2.Subdiv2D(rect)
subdiv_morph = cv2.Subdiv2D(rect)

subdiv_src.insert(src_pts)
subdiv_target.insert(target_pts)
subdiv_morph.insert(points)
```

```
triangles_src = subdiv_src.getTriangleList()
triangles_target = subdiv_target.getTriangleList()
triangles_morph = subdiv_morph.getTriangleList()
```

Here is the example of delunay triangulation. First Third image is our final triangles. We will be painting last triangles from source(Bush) image and we will apply affine transform between Bush and Arnold.

After finding triangles, we will do the transformation and for each triangles we will make triplets of triangles(one from each triangle source)

Code 4: Creating triangles as array of arrays instead of array of tupples

```
n1 = [[tr1[0], tr1[1]],
      [tr1[2], tr1[3]],
      [tr1[4], tr1[5] ] ]

n2 = [[mostCloseTarget[0], mostCloseTarget[1]],
      [mostCloseTarget[2], mostCloseTarget[3]],
      [mostCloseTarget[4], mostCloseTarget[5]]]

n3 = [[mostCloseMorph[0], mostCloseMorph[1]],
      [mostCloseMorph[2], mostCloseMorph[3]],
      [mostCloseMorph[4], mostCloseMorph[5]]]
```

Then I will use modified version of morphTriangle(from the link giving in the homework PDF)

Code 5: Morphing a triangle

```
def morphTriangle(img1, img, t1, t2, t):
    ...
    # Apply warpImage to small rectangular patches
    # This is the oart I modified now we can only focus source image,
    # we do not use pixel intensity values of
    # target image. We will use it only for transformation estimation

    img1Rect = img1[r1[1]:r1[1] + r1[3], r1[0]:r1[0] + r1[2]]
    size = (r[2], r[3])
    warpImage = applyAffineTransform(img1Rect, t1Rect, tRect, size)
    img[r[1]:r[1] + r[3], r[0]:r[0] + r[2]] = img[r[1]:r[1] +
        r[3], r[0]:r[0] + r[2]] * (1 - mask) + warpImage * mask
```

Code 6: Affine estimation for a triangle

```
def get_transform(dest, src):
    # Create Ax=b linear equation then
    # take inverse of A and x will be  A{inverse} b
    # if we change the parameter order ot would give inverse transform
    A = np.array([ [dest[0][0], dest[0][1], 1, 0, 0, 0],
                   [dest[1][0], dest[1][1], 1, 0, 0, 0],
                   [dest[2][0], dest[2][1], 1, 0, 0, 0],
                   [0, 0, 0, dest[0][0], dest[0][1], 1],
                   [0, 0, 0, dest[1][0], dest[1][1], 1],
                   [0, 0, 0, dest[2][0], dest[2][1], 1]] )
    b = np.array([ src[0][0], src[1][0], src[2][0], src[0][1], src[1][1], s

    x = np.linalg.lstsq(A, b)[0]

    transform = np.array( [ [x[0], x[1], x[2] ],
                            [ x[3],  x[4], x[5] ]] )

    # opencv WARPAFFINE take 2x3 matrix
    # I tested this with opencv code it seems ok.
    # But to us it with opencv warpaffine it should be 2*3 matrix
    return transform
```

3 Issues

3.1 Matching Triangles

After getting the triangles, when we are morping image we use triplets but these triangles in triplets must be match. I mean they must represent the nearly same locations at images. In order to do this I compared the centers of triangles then math according to their euclidean distances.

Code 7: Calculate the center

```
for i in range(triangles_src.shape[0]):
    tr = triangles_src[i]
    center_x = tr[0]+tr[2]+tr[4]
    center_y = tr[1]+tr[3]+tr[5]
    center_x =  center_x / 3.0
    center_y =  center_y / 3.0
```

Code 8: For each triangle at source, find a match for it

```
mostCloseTarget = None
mostCloseMorph = None
selectedM = -1
selectedT = -1
least_dist = float('inf')
for j, tr2 in enumerate(triangles_target):
    dist = np.sqrt ( (tr1[6]-tr2[6])**2 + (tr1[7]-tr2[7])**2)
    if not visitedTarget[j] and  dist < least_dist:
        least_dist = dist
        mostCloseTarget = deepcopy(tr2)
        selectedT=j
```

3.2 Matching Points of Triangles

After deciding the triangle matches, we need to match triangle‘s points in order to get correct affine transform. To do this I used compared the distances.

Code 9: For each triangle at source match it

```
least_dist = float('inf')
pt = None
for j,pts in enumerate(n2):
    dist = np.sqrt ( (x - pts[0])**2 + (y - pts[1])**2)
    if not visT[j] and  dist < least_dist:
        least_dist = dist
        pt  = deepcopy(pts)
        sT=j
n2[i] = deepcopy(pt)
visT[sT] = True
```

Here is problematic result The landmarks should be chosen with carefully in order to get nice result.

4 Screenshots