BLG453E Homework-2

Dr. Öğr. Üyesi Yusuf Hüseyin Şahin

sahinyu@itu.edu.tr

1 - Part 1: Salt & Pepper

Bugs: Hey Sam, pass the salt please.
Sam: Salt? GET IT YOURSELF!
Bugs Bunny: Uh oh, that'll cost you about...
Sam: Salt? Why didn't you say so. Here's your salt, Bunny, I hope you like it. Ooh that rackin' frackin'...
Bugs: The pepper please.
Sam: PEPPER! WE... Uh, yeah the pepper. Coming right up.
From Hare To Heir (1960)

1.1.: Preprocessing

In this part, we will work on the video "shapes video.mp4 ". For every frame of the video, a new card having a specific shape (star, square, pentagon) appears. However, each frame has a salt & pepper noise as given in Figure 1.

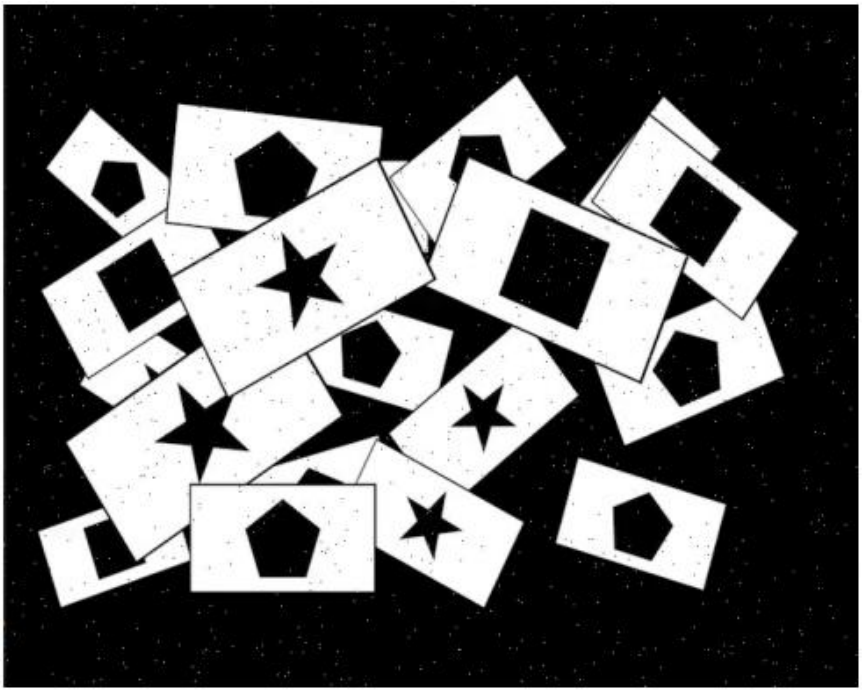

Figure 1 A frame from the video

Filter each frame to avoid noise and save the video as a new file. To read the video frame by frame, you can use the following lines.

```
import moviepy.video.io.VideoFileClip as mpy
import cv2

vid = mpy.VideoFileClip('shapesvideo.avi')

framecount = vid.reader.nframes
videofps = vid.fps

for i in range(framecount):
    frame = vid.get_frame(i * 1.0 / videofps)
```

1.2.: Counting the shapes

Implement Minimum Eigenvalue corner detector on the frames to obtain the corners of the shapes. Then, count the exact counts of triangle, star and pentagon cards. (Hint: You can do background subtraction between frames.)

Part 2-1: Delaunay Triangulation

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.

```
import cv2

image = cv2.imread("image.png")
subdiv = cv2.Subdiv2D((0, 0, image.shape[0], image.shape[1]))
# Subdiv2D is an OpenCV object which performs Delaunay triangulation.

for i in range(68):
    subdiv.insert(...)  # Each landmark point should be inserted into
Subdiv2D object as a tuple. Show your work here.

# Add the points to another list to check again.

subdiv.insert((0, 0))
subdiv.insert((0, image.shape[1] - 1))
# 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.

triangles = subdiv.getTriangleList()
```

Here triangles is a matrix of (142, 6) (row counts can be changed according to point locations). Each row has xy positions {x1, y1, x2, y2, x3, y3} 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 2.

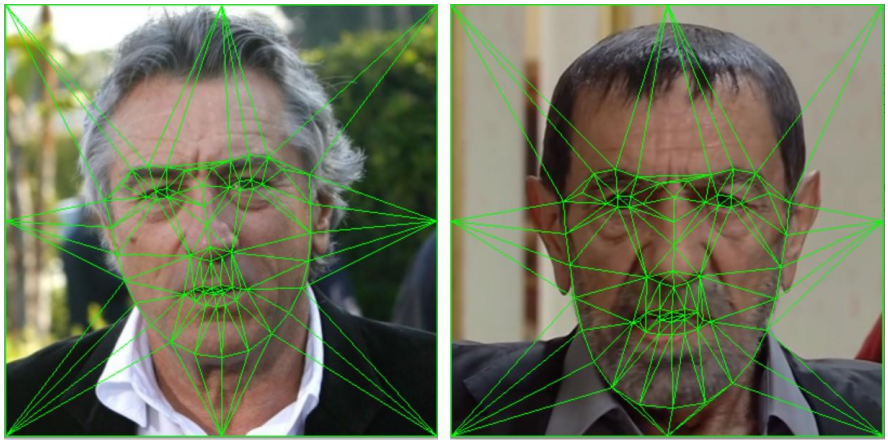

Figure 2 Delaunay triangles placed on the faces

Part 2-2: Face Morphing

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 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] = calctransform(source, target)  # (A)

morphs = []

for t in np.arange(0, 1.0001, 0.02):  # (B)
    print("processing:\t", t * 100, "%")
    morphs.append(imagemorph(image, catimg, 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 calctransform(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 vectorisedBilinear(coordinates, targetimg, 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
    residual = 1 - error

    topleft = np.multiply(
        np.multiply(residual[0],
residual[1]).reshape(coordinates.shape[1], 1),
        targetimg[lower[0], lower[1], :]
    )

    topright = np.multiply(
        np.multiply(residual[0],
error[1]).reshape(coordinates.shape[1], 1),
        targetimg[lower[0], upper[1], :]
    )

    botleft = np.multiply(
        np.multiply(error[0],
residual[1]).reshape(coordinates.shape[1], 1),
        targetimg[upper[0], lower[1], :]
    )

    botright = np.multiply(
        np.multiply(error[0], error[1]).reshape(coordinates.shape[1],
1),
        targetimg[upper[0], upper[1], :]
    )  # (G)

    return np.uint8(np.round(topleft + topright + botleft +
botright))  # (H)
```

```
def imagemorph(image1, image2, triangles1, triangles2, transforms, t):

    interimimage1 = np.zeros(image1.shape).astype(np.uint8)
    interimimage2 = np.zeros(image2.shape).astype(np.uint8)

    for i in range(len(transforms)):

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

        polygonmask = np.zeros(image1.shape[:2], dtype=np.uint8)
        cv2.fillPoly(
            polygonmask,
            [np.int32(np.round(homointertri[1::, :].T))],
            color=255
        )  # (J)

        seg = np.where(polygonmask == 255)  # (K)

        maskpoints = np.vstack((seg[0],
seg[1],np.ones(len(seg[0]))))  # (L)

        intertri = homointertri[:2].flatten(order="F")  # (M)

        intertoimg1 = calctransform(intertri, triangles1[i])
        intertoimg2 = calctransform(intertri, triangles2[i])

        mapped_to_img1 = np.matmul(intertoimg1, maskpoints)[:-1]  # (N)
        mapped_to_img2 = np.matmul(intertoimg2, maskpoints)[:-1]

        interimimage1[seg[0], seg[1], :] = vectorisedBilinear(
            mapped_to_img1, image1, interimimage1.shape
        )  # (O)

        interimimage2[seg[0], seg[1], :] = vectorisedBilinear(
            mapped_to_img2, image2, interimimage2.shape
        )

    result = (1-t) * interimimage1 + t * interimimage2  # (P)

    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 (https://web.itu.edu.tr/sahinyu/hw3_panda.mp4 , https://web.itu.edu.tr/sahinyu/hw3_aydemir.mp4)

Part 3: Saturday Night Filter

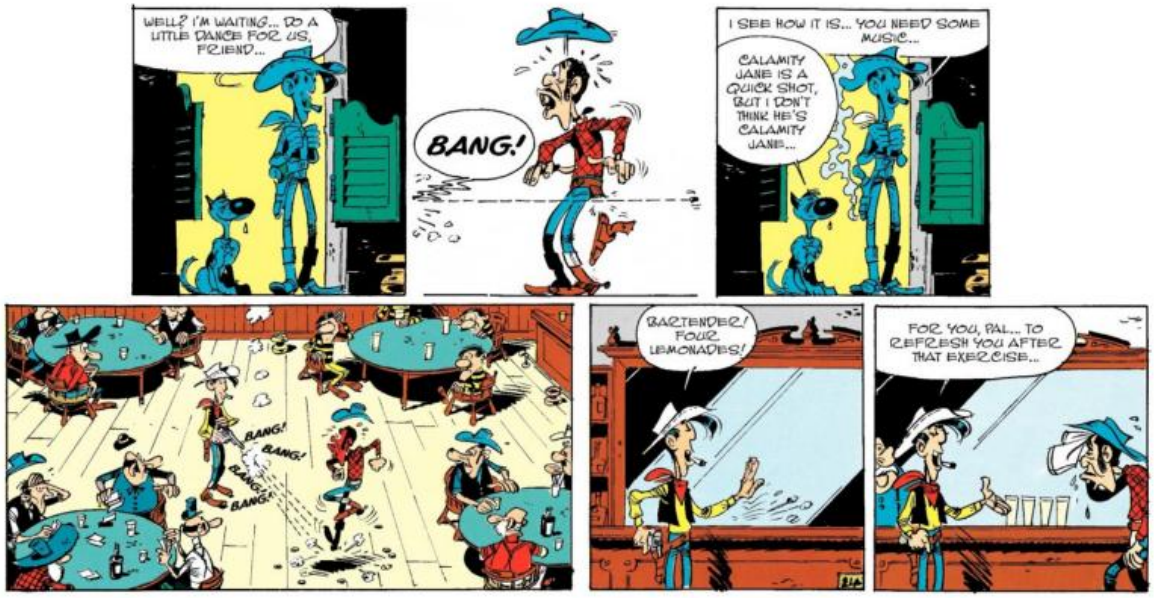

Figure 3Lucky Luke : The Daltons Redeem Themselves, Morris & Goscinny (1965)

Mutsuz insan yoktur, dans etmeyen insan vardır.

Tekin Abi, Tekin Abi ile Dans Saati (1980s)

For this part of the homework, I prepared a small dance game using Unity3D, in which we will make a 3D character dance according to the given shape inputs. You can download/reach the game via the following links:

• Windows Executable: https://web.itu.edu.tr/sahinyu/saturday_night_filter_pc.zip

• Online WebGL Game: https://web.itu.edu.tr/sahinyu/saturday_night_filter_web/

The main page of the game is as given in Figure 4. We will be working on two songs: Vabank and Shame. Before selecting one of the songs, ensure that you entered your student ID in the textbox.

Figure 4 Main interface

For this homework, you will benefit from pyautogui library which is used to simulate mouse and keyboard interactions with Python. The example script given below first takes a screenshot of the game, then clicks random buttons.

```
import pyautogui
import time

time.sleep(5)
# In this 5 seconds you should switch to game screen to transfer the
# simulated keyboard inputs to the game.

myScreenshot = pyautogui.screenshot()
myScreenshot.save('test.png')
# An example screenshot is obtained. We will work on screenshots like
this
# for this homework.

pyautogui.keyDown('shift')
pyautogui.keyDown('w')
time.sleep(1)

pyautogui.keyUp('w')
pyautogui.keyUp('shift')
# pyautogui.keyUp and pyautogui.keyDown functions are used to simulate
# holding a button. For simple presses, pyautogui.press can be used.
```

Clicking ”Vabank” or ”Shame” buttons will direct you to a game page as given in Figure 5. You can press ESC to go back to the main screen.

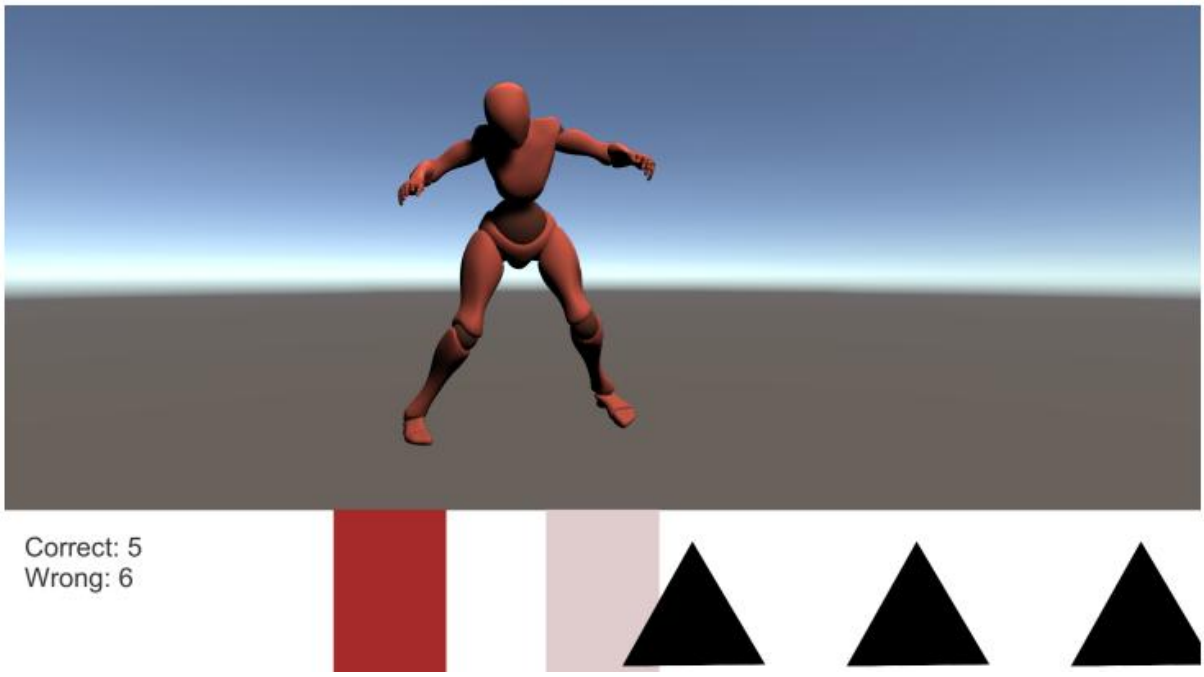

Figure 4 A game page

The game pages consist of our dancing 3D model and a 2D canvas at the bottom showing the shapes on a timeline. According to the dance, some shapes are flown from right to left. The player should press the corresponding button on time to obtain a point.

3.1.: Better Than Jackson

Using your findings in the first part, write a script which beats the game. For this part, you are free to use every OpenCV function. For each song, according to your scores and student ID, a small key will appear on the screen. Do not forget to report these keys for each song. Also, report your screen resolution.