BLG 453E Computer Vision (Fall 2018)
Homework-5

Alperen Kantarcı - 150140140
January, 2019

1 Application Usage

Note: There is two different Python scripts for first and second questions. In order to run the scriptys you need to run either " lucaskanade.py " or "eigenfaces.py" and if you have the directories in the same path with the scripts they will be executed properly. For different pathing you need to use different relative path. The requirements that written on the Github repository.
Github page of this application is here.

2 First Question - Lucas Kanade

Default path for this question is $traffic\_sequence/$ this is relative path so if script and the traffic_sequence directory in the same directory then script will run without any error. Optical flow field for the sequence calculated by the Lucas Kanade algorithm. When you run the script it will show you some kind of video sequence with the optical flow vectors. You can see some example outputs.

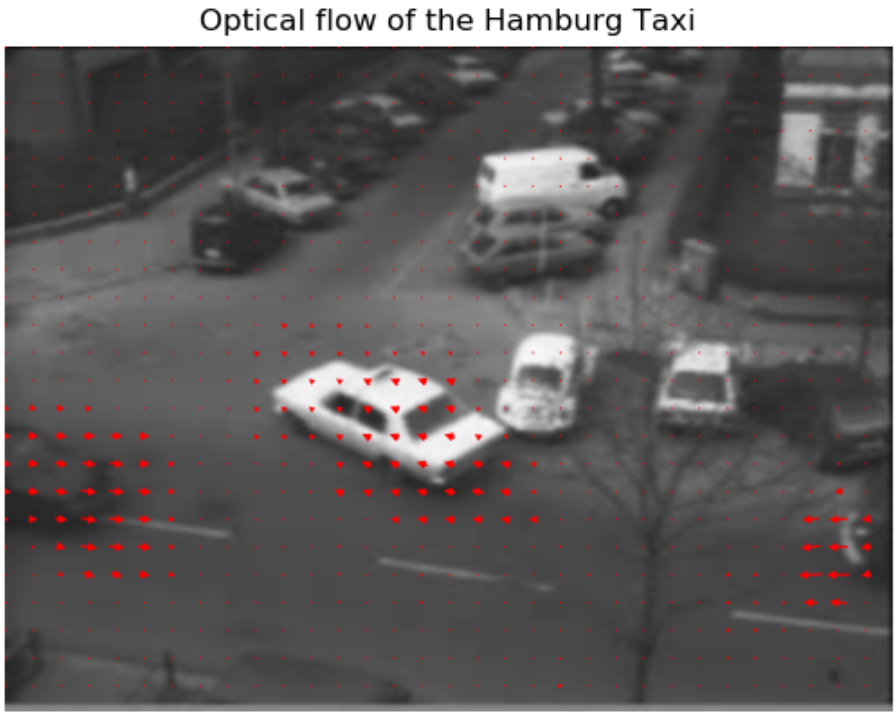

Figure 1: Initial moves

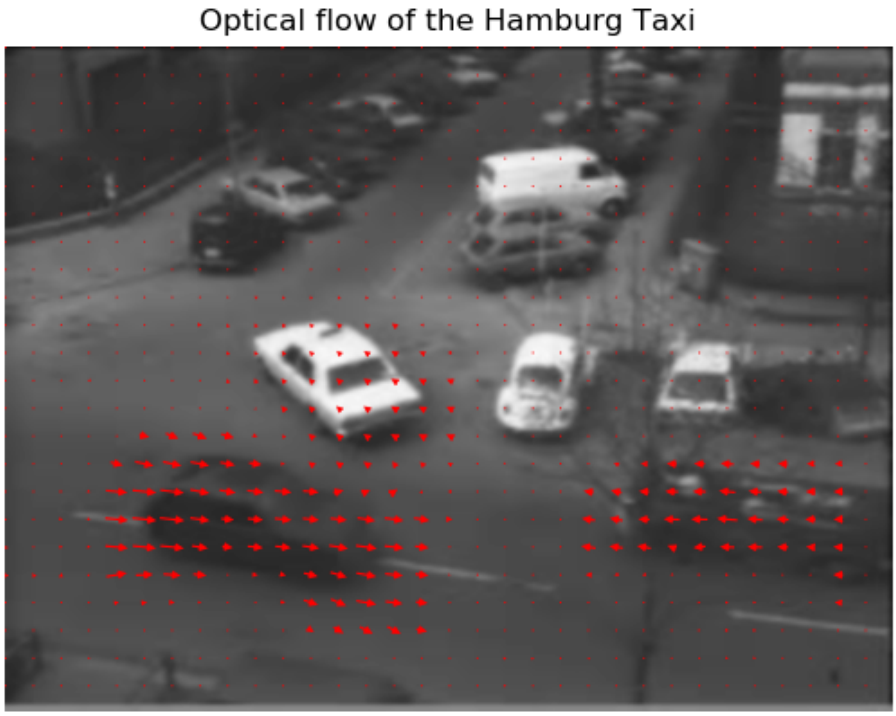

Figure 2: After some time

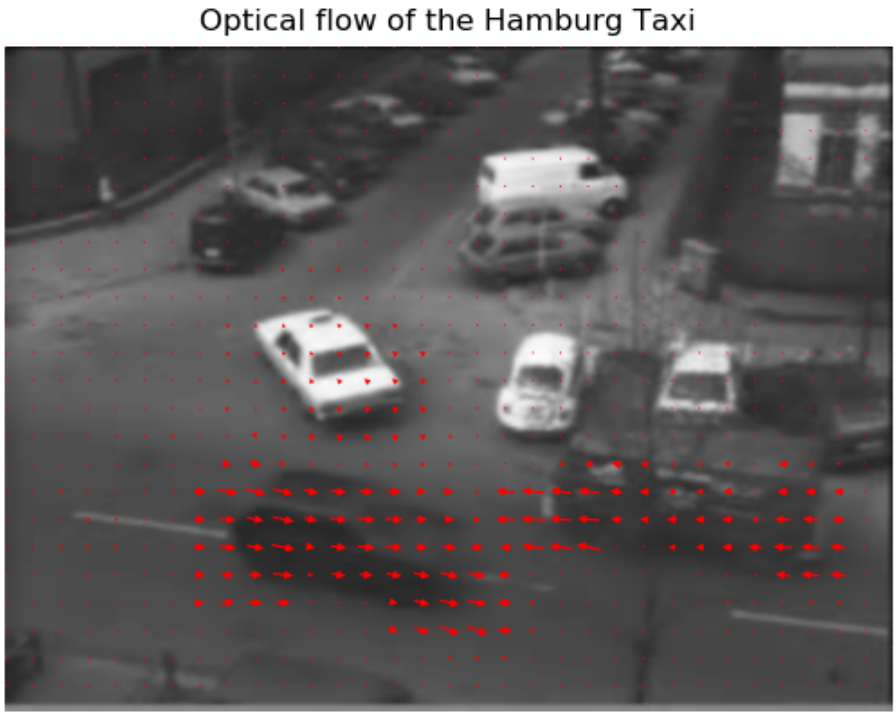

Figure 3: Last frame for the images

3 Python code

After importing image from the given path, first i calculate the Ix,Iy,It derivatives for the two consecutive frames. While calculating these differentiations i used 2D convolutions with differentiation kernels for each differentiation.

```
def calc_derivatives(old_frame, new_frame):
    # Get derivatives with kernels
    kernel_x = np.fliplr(0.25 * np.array(([-1, 1], [-1, 1])))
    kernel_y = 0.25 * np.array(([-1, -1], [1, 1]))
    kernel_t = 0.25 * np.array(([1,0],[0,1]))
    # Get convolutions of the kernels so that calculate derivatives
    Ix = cv2.filter2D(old_frame, -1, kernel_x) + cv2.filter2D(new_frame,
        -1, kernel_x)
    Iy = cv2.filter2D(old_frame, -1, kernel_y) + cv2.filter2D(new_frame,
        -1, kernel_y)
    It = cv2.filter2D(old_frame, -1, kernel_t) + cv2.filter2D(new_frame,
        -1, -kernel_t)
    return (Ix, Iy, It)
```

Then using these values i calculate the u,v points for each x,y points.

```
def lucas_kanade(old_frame, new_frame,window):
    # Get derivatives between two images
    Ix, Iy, It = calc_derivatives(old_frame, new_frame)
    # Denominator
    denom = cv2.filter2D(Ix**2, -1, window)*cv2.filter2D(Iy**2, -1,
        window) - cv2.filter2D((Ix*Iy), -1, window)**2
    denom[denom == 0] = np.inf
    # Calculate u and v values using window with pi pixels
    u = (-cv2.filter2D(Iy**2, -1, window)*cv2.filter2D(Ix*It, -1, window
        ) + cv2.filter2D(Ix*Iy, -1, window)*cv2.filter2D(Iy*It, -1,
        window)) / denom
    v = (cv2.filter2D(Ix*It, -1, window)*cv2.filter2D(Ix*Iy, -1, window)
         - cv2.filter2D(Ix**2, -1, window)*cv2.filter2D(Iy*It, -1,
        window)) / denom

    return (u, v)
```

Then i show each flow in the vector field

```
pixel_width= 25
# Calculate optical flow using two different algorithms
u, v = lucas_kanade(old_frame, new_frame, np.ones((pixel_width,
    pixel_width)))

# Create grid for display
x = np.arange(0, old_frame.shape[1], 1)
y = np.arange(0, old_frame.shape[0], 1)
x, y = np.meshgrid(x, y)

# Display
plt.title("Optical flow of the Hamburg Taxi")
plt.axis('off')
plt.imshow(old_frame, cmap='gray', interpolation='bicubic')

step = 8
# Create vector field and give the motion to the vector field to
    visualize
plt.quiver(x[::step, ::step], y[::step, ::step],u[::step, ::step], v[::
    step, ::step],edgecolor="k",color='r', pivot='middle', headwidth=4,
    headlength=4,scale = 50)
plt.pause(0.3)
plt.draw()
```

4 Question 2 - Eigenfaces and Recognition

Default path for this question is $Face_database/$ this is relative path so if script and the Face_database directory in the same directory then script will run without any error. First algorithm will show you the 5 highest valued eigenfaces in the database and the mean face that has been derived from the database. After closing this screen script will choose a random image from the Face_database folder and using the precalculated weights of 5 eigenfaces, it will try to reconstruct the original image as much as possible. You can see the example outputs below.

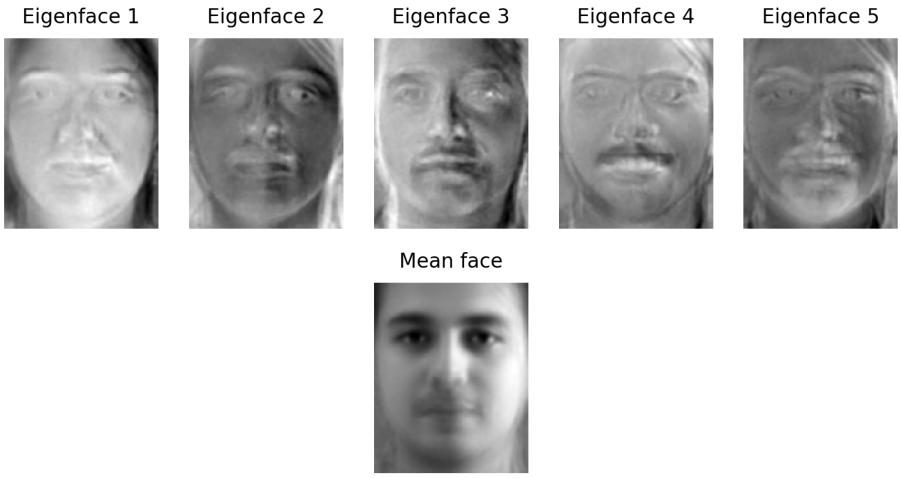

Figure 4: 5 Eigenfaces(Eigenvectors) with highest eigen values

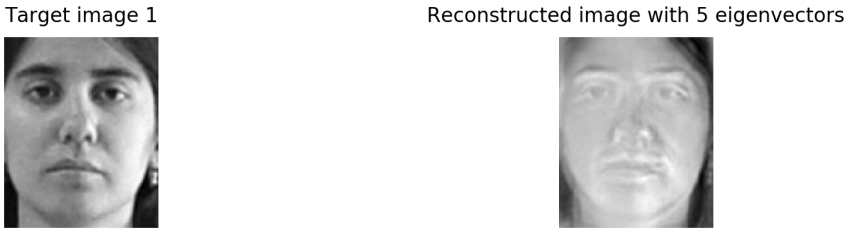

Figure 5: Reconstructed and target image

5 Python code

5.1 Eigenfaces

After importing image from the given path, first i convert the images to the vectors and stack them to get 32x(height*width) matrix with each image 1x(height*width). Then mean image from these images calculated and subtracted from tthe original images.

```
# (a) Convert all the images to numpy array
images = [cv2.imread(x,cv2.IMREAD_GRAYSCALE) for x in imgs]
orj_w,orj_h = images[0].shape
faces = np.empty(shape=[1, images[0].shape[0]*images[0].shape[1]])

# (a) Convert them to vectors and stack them to one matrix horizontally
images = [np.reshape(a,(1,-1)) for a in images ]
for i in images:
  faces = np.vstack((faces,i))
faces = np.delete(faces, (0), axis=0)

# (b) Find mean face
mean_image = np.mean(images, axis=0)

# (b) Substract mean face so find Z vector
arr_norm = np.zeros([len(images), orj_w*orj_h])
arr_norm = faces - mean_image
```

Then A*A.T is calculated because it is hard to (height*width)x(height*width) compute SVD of this vector instead i calculate the 32x32 vector of the SVD then multiply it with mean substracted images.

```
# (c) In here A*A.T is used for fast calculation of SVD
# Here we calculate len(faces)xlen(faces) dimension instead covariance
    matrix
S = np.matmul(arr_norm,arr_norm.T)
u, s, vh = np.linalg.svd(S, full_matrices=True)
# Sort eigenvalues descending
idx = s.argsort()[::-1]
s = s[idx]
u = u[:,idx]
# (c) Get eigen vectors and print them
eigenvectors = np.matmul(u.T,arr_norm)
num_eigfaces = 5
print_eigenfaces(mean_image,eigenvectors[:num_eigfaces,:],orj_w,orj_h,"
    Part A Eigen Faces")
```

5.2 Face Recognition

In this part first i created image weights database for each image in the database and converted them into the basis B. Then i converted the target image to the same basis and calculated ecludian distance between the weights database and target image weights. Then closest eigen faces is returned. This

weights then multiplied with with closest image and added mean face. Reconstructed image then printed.

```
# PART B of the project
# Calculate each image weights for desired number of eigenvectors
weight_database = np.matmul(arr_norm,eigenvectors[:num_eigfaces,:].T)
# (a) Get random image to recognize and add gaussian noise
img_id = np.random.randint(len(images),size=1)
test_face = faces[img_id]
# (b) Transform to basis B
smoothed_face = cv2.GaussianBlur(test_face,(3,3),0.0125,0,cv2.
    BORDER_DEFAULT).reshape((1,-1))
smoothed_face -= mean_image

# (c) Calculate weights of the face
face_weight = np.matmul(eigenvectors[:num_eigfaces,:],smoothed_face.T)
# (d) Find the closest weights in database
closest_weights = calculateDistance(weight_database,face_weight)
# (e) Reconstruct the image with given eigenvectors and their weights
reconstructed = mean_image + np.matmul(closest_weights.T,eigenvectors[:
    num_eigfaces,:])
print_reconstructed(reconstructed,test_face,orj_w,orj_h,"Part B Face
    Recognition")
```