Istanbul Technical University
Faculty of Computer and Informatics
Computer Engineering Department

BLG 453E
Computer Vision
Homework V

Uğur Uysal - 150140012

Dec 24$^{\text{th}}$, 2018

Contents

1 Usage

The code depends on numpy, opencv and PyQt5. If all requirements are satisfied, then code can be executed.

```
$ python3 filename
```

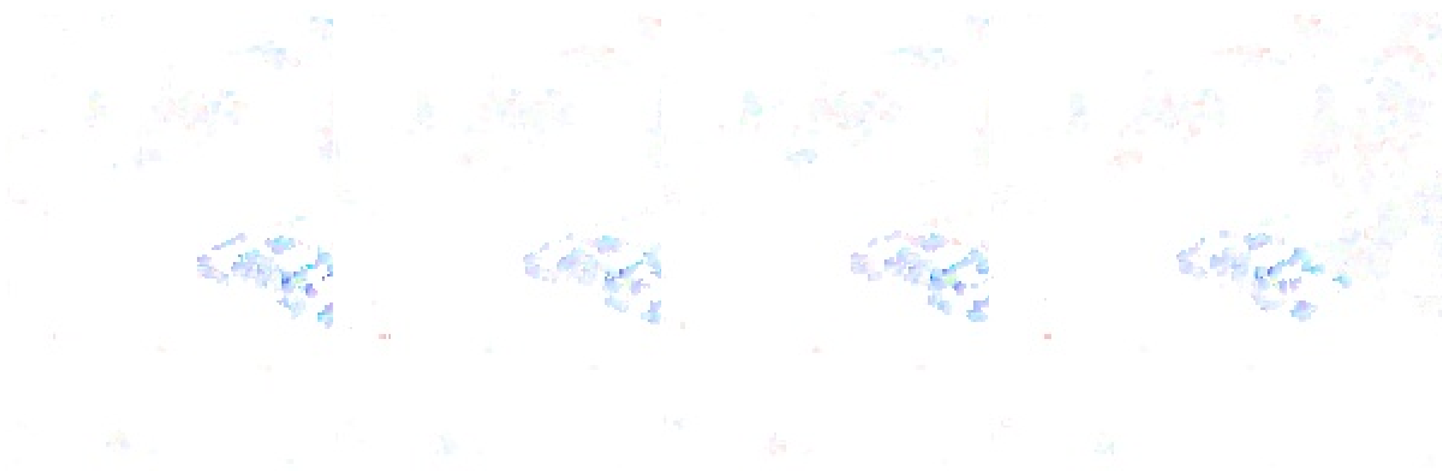
For both applications, from menu first load data please.

2 Optical Flow

In first part of homework, we are asked to find optical flow [1] of given sequence of images in traffic. In brief, optical flow define $u = \frac{dx}{dt}$ and $v = \frac{dy}{dt}$ as velocities. It makes two assumptions about brightness consistency($I(x, y, z) = I(x + dx + y + dy, t + dt)$) and small motions(objects move slow). In Lucas-Kanade algorithm [2] is implemented. In brief, the Lucas-Kanade constructs a least square solution to estimate the velocities because of u and v does not define a point in plane, they define a line, to find exact values we make one more assumption that in small windows velocities are same. Then we come up with this solution.

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

Implementation I loaded the images to python code and map the onto [01]. Then calculated I_x, I_y and I_y with convolution. $I_t = KERNEL * I_{t+1} - It * KERNEL$ where kernel is $2X2$ ones matrix and * is convolution operator. To visualize the optical flow I used the code in [4].Here are the some results(not with the given window size).



I tried show video using opencv. But codes are platform dependent. I used "XVID" as default codec type.

2.1 Implementation

```
u, v = np.zeros((w, h)), np.zeros((w, h))
kernel_x = np.array([[-1., 1.],
                     [-1., 1.]])

kernel_t = np.array([[1., 1.],
                     [1., 1.]])

# gradients respect x,y and t
fx = signal.convolve2d(I_1, kernel_x, mode="same")
fy = signal.convolve2d(I_1, kernel_x.T, mode="same")
ft = signal.convolve2d(I_2, kernel_t, mode="same") - signal.convolve2d(I_1,
...

Ix = fx[i-window_size:i+window_size+1, j -
        window_size:j+window_size+1].flatten()
Iy = fy[i-window_size:i+window_size+1, j -
        window_size:j+window_size+1].flatten()
It = ft[i-window_size:i+window_size +
        1, j-window_size:j+window_size+1].flatten()

# construct the linear equation system Ax = b
# x = (A.T*A)-1A.T*b
b = np.reshape(It, (It.shape[0], 1))
A = np.vstack((Ix, Iy)).T
```

3 Eigenfaces and Dimension Reduction with PCA

In this part homework, I worked on face image data set. There 32 images and single image size is $168X136$. They are flatten into data matrix $X = 32*(168*163)$. Then they are centered by subtracting the mean $X = X - \mu$. Then computing the $X.T * x$ will be very hard due to memeory and time limits, I computed $S = X * X.T$ and apply singular value decomposition to [3] S and I these eigenvectors.

3.1 Face Recognition

In the second part of homework, with help of dimension reduction, we developed an algorithm that recognize faces. First we select a image from data set and add Gaussian noise to it. Then reduce dimension of data(I reduced data set size to 5) after reducing the dimension compute distance between vectors. At the I expected that the reduced version of selected image will be closest vector and it happened. Here is a screen shot of results.

```
[ugur@ugur-pc hw5]$ python q2.py
RANDOM ELEM is 17
Index and Cost
i: 1 distance: 9851822.174763415
i: 2 distance: 9657962.240185505
i: 3 distance: 16033383.773294771
i: 4 distance: 19197745.20903998
i: 5 distance: 15313613.0549565
i: 6 distance: 7617961.310738864
i: 7 distance: 26533410.317180596
i: 8 distance: 13813544.007314226
i: 9 distance: 19471081.906228147
i: 10 distance: 9222986.07633189
i: 11 distance: 11087103.589561814
i: 12 distance: 11320764.799203124
i: 13 distance: 24009248.390985575
i: 14 distance: 11443210.417160323
i: 15 distance: 20888210.425251786
i: 16 distance: 9955865.390319057
i: 17 distance: 5.609318528875219e-25
i: 18 distance: 5275653.901141296
i: 19 distance: 33984133.72935638
i: 20 distance: 35747678.35351591
i: 21 distance: 11802666.425327476
i: 22 distance: 5950460.808156515
i: 23 distance: 12767147.942488242
i: 24 distance: 21758223.44356505
i: 25 distance: 9488513.921056973
i: 26 distance: 12631550.264005585
i: 27 distance: 30344989.88726896
i: 28 distance: 24232511.280068655
i: 29 distance: 14501967.052439269
i: 30 distance: 8263608.640204364
i: 31 distance: 7066768.503627549
```

```
[ugur@ugur-pc hw5]$ python q2.py
RANDOM ELEM is 8
Index and Cost
i: 1 distance: 4121708.098056443
i: 2 distance: 6221779.796235557
i: 3 distance: 2915798.613154768
i: 4 distance: 1829701.195124582
i: 5 distance: 11480651.464736197
i: 6 distance: 4678515.388734115
i: 7 distance: 28187546.603821922
i: 8 distance: 2.6678190341969034e-24
i: 9 distance: 15608810.520649716
i: 10 distance: 5935059.476789871
i: 11 distance: 1169218.3008236266
i: 12 distance: 13793607.93162705
i: 13 distance: 19536168.661334287
i: 14 distance: 2926755.411539198
i: 15 distance: 4897431.610516813
i: 16 distance: 1700608.3981837458
i: 17 distance: 13776627.96454584
i: 18 distance: 6839841.008192977
i: 19 distance: 47986104.36917855
i: 20 distance: 11998417.609762512
i: 21 distance: 1594396.781395852
i: 22 distance: 8432777.277807051
i: 23 distance: 8679565.042725239
i: 24 distance: 12391227.870007008
i: 25 distance: 6627105.073235598
i: 26 distance: 2991717.7051803367
i: 27 distance: 32627105.403080296
i: 28 distance: 7587183.677548727
i: 29 distance: 14805116.814155553
i: 30 distance: 5306871.727511124
i: 31 distance: 2408848.2984401197
```

References

[1] Optical Flow on Wikipedia

[2] Lucas-Kanade on Wikipedia

[3] SVD on Wikipedia

[4] Optical Flow Visualization