ENGN4528/6528 Computer Vision 2019

CLab-2 Tasks

1) Harris Corner Detector. (5 marks)

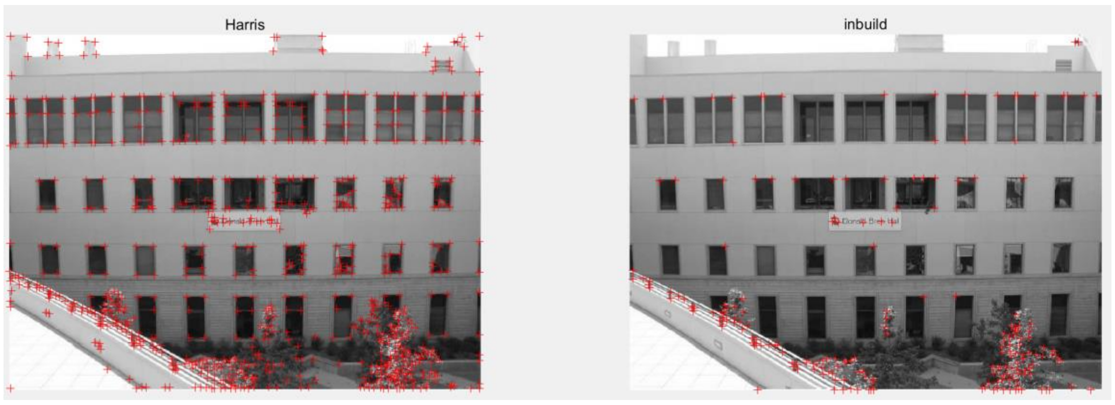
1) Read and understand the corner detection code ‘harris.m’.
   Harris
2) Complete the missing parts, rewrite ‘harris.m’ into a Matlab function and design appropriate function signature (10).

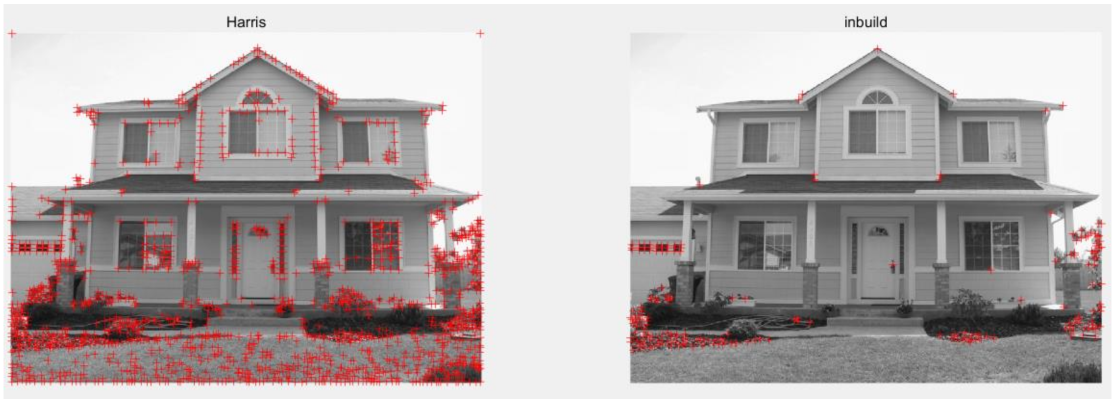
```
% Task: Compute the Harris Cornerness %
[m,n] = size(bw);
R = zeros(m,n);
k = 0.05;   %usually use between 0.04~0.06
for i = 1:m
    for j = 1:n
        M = [Ix2(i,j) Ixy(i,j) ; Ixy(i,j) Iy2(i,j)]; %calculation of M matrix which computed from image derivatives
        R(i,j) = det(M) - k*(trace(M))^2; %calculate R to judge if it is on the edge
    end
end
```

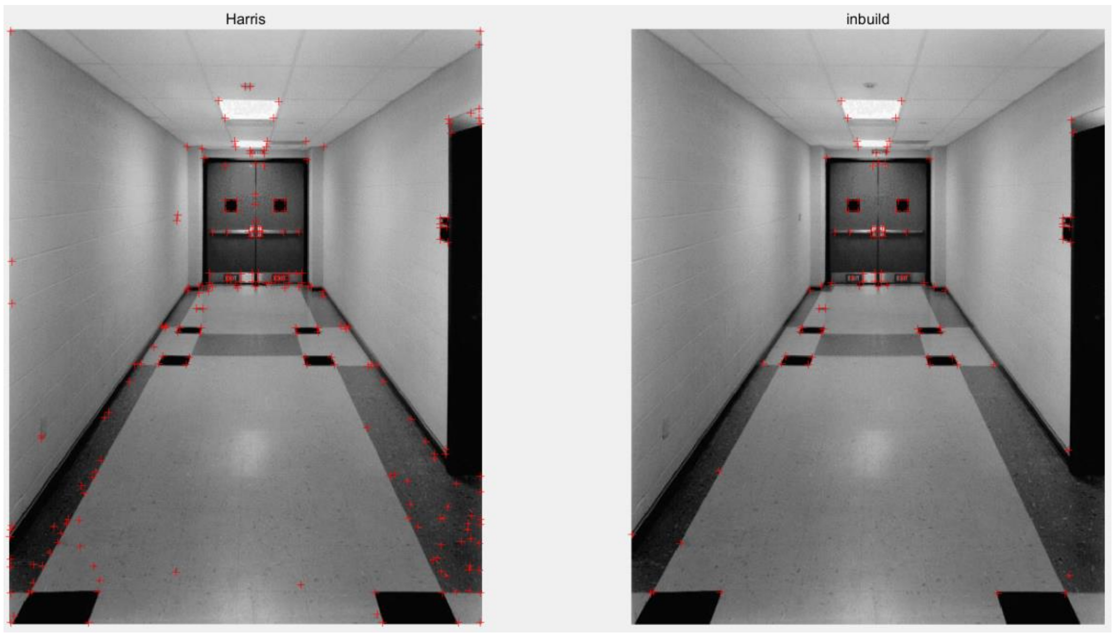
```
% Task: Perform non-maximum suppression and %
% thresholding, return the N corner points %
% as an Nx2 matrix of x and y coordinates %
Rmax = max(max(R));
t = thresh * Rmax; %Rmax * thresh as a threshold, judge if this point is corner
for i = 1:m
    for j = 1:n
        if R(i,j) < t %compare with threhold
            R(i,j) = 0; %give 0 to the not corner point
        end
    end
end
corner = imregionalmax(R); %identify the regional maxima
[posr,posc] = find(corner == 1);% find the regional maxima point position
bw = uint8(bw);


figure
imshow(bw); %show the origin picture
hold on
for i = 1:length(posr)
    plot(posc(i), posr(i), 'r+'); %mark the corner in the picture with '+'

end
```

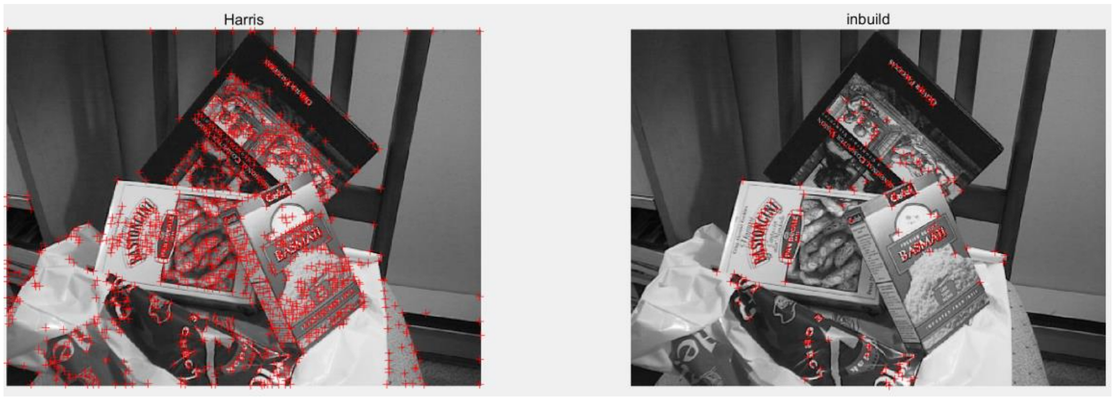
3) Comment on line #13 and every line of your solution after line #20 (0.50).
4) Test this function on the provided four test images (can be downloaded from Wattle). Display your results by marking the detected corners on the input images (using circles or crosses, etc) (0.50 for each image, 2 in total).
   See below
5) Compare your results with that from Matlab’s built-in function corner() (0.50), and discuss the factors that affect the performance of Harris corner detection (1).

   The comparison result see below, and I think the performance affect by the Gaussian window filter, so the sigma value will affect the performance. Another importance thing to affect the performance is the selection of threshold value.


Harris
inbuild


Harris
inbuild

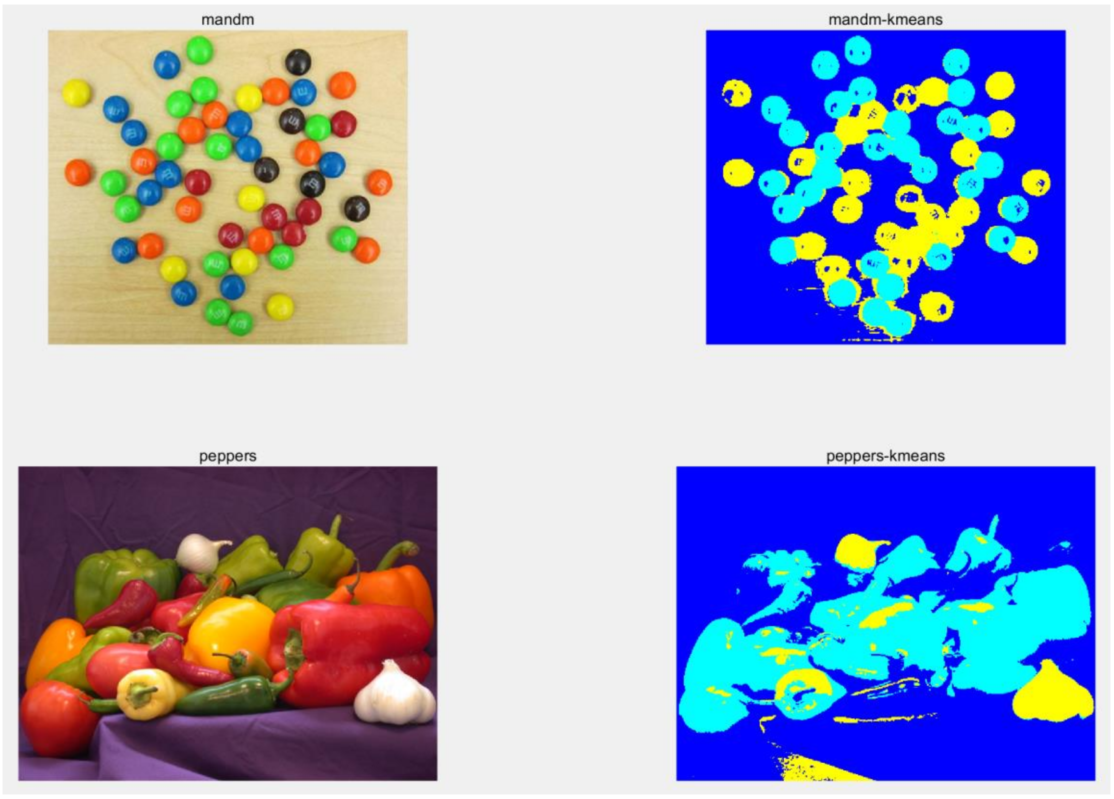

Harris
inbuild


Harris
inbuild

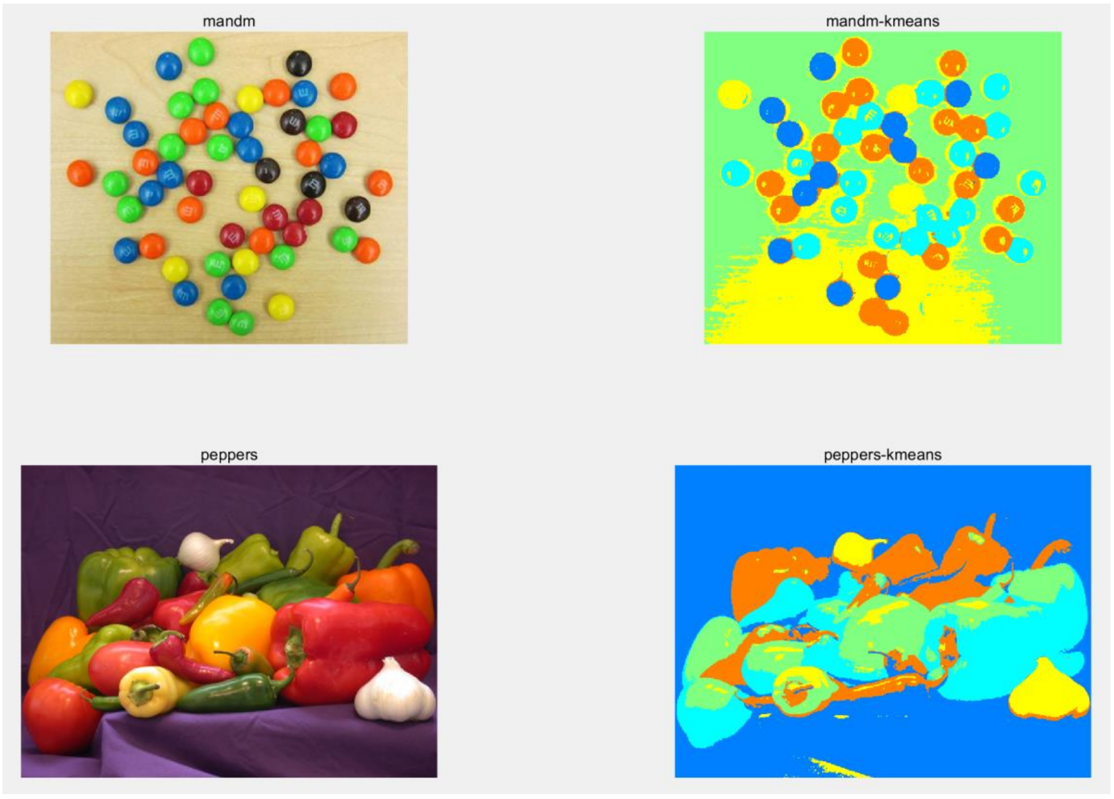
2) K-Means Clustering and Color Image Segmentation.(5 marks)

1) Implement your own K-means function my_kmeans(). The input is the data points to be processed and the number of clusters, and the output is several clusters of points (you can use a cell array for clusters). Make sure each step in K-means is correct and clear, and comment on key code fragments (1.5).

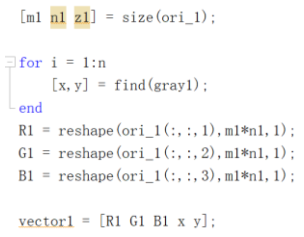
k=3



k=5



2) Apply your K-means function to color image segmentation. Each pixel should be represented as a 5-D vector that encodes: (1) L_ - lightness of the color; (2) a_ - the color position between red and green; (3) b_ - the position between yellow and blue; (4) x, y - pixel coordinates. Please compare segmentation results (1) using different numbers of clusters (1), and (2) with and without pixel coordinates (1).

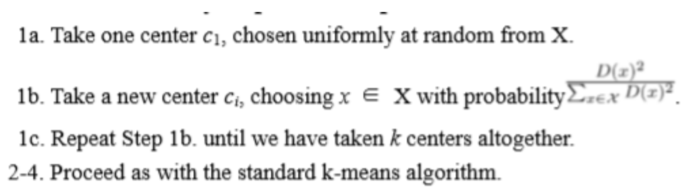


```
[m1 n1 z1] = size(ori_1);

for i = 1:n
    [x, y] = find(gray1);
end
R1 = reshape(ori_1(:, :, 1), m1*n1, 1);
G1 = reshape(ori_1(:, :, 2), m1*n1, 1);
B1 = reshape(ori_1(:, :, 3), m1*n1, 1);

vector1 = [R1 G1 B1 x y];
```

The 5-D vector is include the origin [R G B] vector and x, y pixel

3) The standard K-means algorithm is sensitive to initialization (e.g. initial cluster centres/seeds). One possible solution is to use K-means++, in which the initial seeds are forced to be far away from each other (to avoid local minimal). Please read the material http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf, summarize the key steps in the report (0.5), and then implement K-means++ in your standard algorithm as a new initialization strategy(0.5). Compare the image segmentation performance (e.g., convergence speed and segmentation results) of this new strategy, with that of standard K-means, using different numbers of clusters and the same 5-D point representation from previous question (0.5).

The calculation steps of K-means++ is



1a. Take one center c_1, chosen uniformly at random from $\mathcal{X}$.

1b. Take a new center c_i, choosing $x \in \mathcal{X}$ with probability $\frac{D(x)^2}{\sum_{x\in\mathcal{X}} D(x)^2}$.

1c. Repeat Step 1b. until we have taken k centers altogether.

2-4. Proceed as with the standard k-means algorithm.

3) Face Recognition using Eigenface. (10 marks)

Code: CVLab2_Q3.m

1) Unzip the face images and get an idea what they look like. Then take 10 different frontal face images of yourself, convert them to grayscale, and align and resize them to match the images in Yale-Face. Explain why alignment is necessary for Eigen-face(0.5).

Recognition with eigenfaces have a big limitation of misalignment and background variation. We need unify the shape and resolution of all training and test images, in order to have the same dimension vector to combine and calculate.

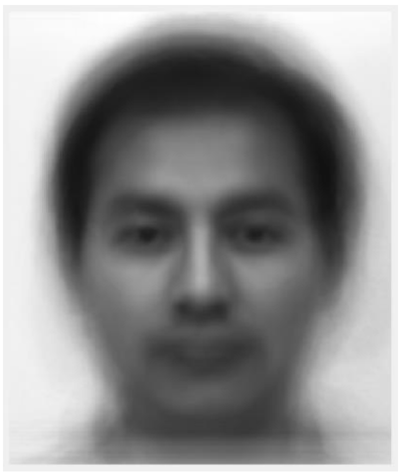
2) Train an Eigen-face recognition system. Specifically, at a minimum your face recognition system should do the following:

a) Read all the 135 training images from Yale-Face, represent each image as a single data point in a high dimensional space and collect all the data points into a big data matrix.

```
for i = 1 : train_num
    filename = [trainpath train_filenames(i).name];   % filename in the list
    train_face = imread(filename);
    vec = reshape(train_face,231*195,1); %transfer to N*1 vector
    data_train = [data_train vec]; %store in the dataset
end
```

b) Perform PCA on the data matrix (1), and display the mean face (1). Given the size of input image, direct eigen decomposition of covariance matrix would be slow. Read lecture notes and find a faster way to compute eigen values and vectors, explain the reason (1) and implement it in your code.

Mean face



Using PCA algorithm on the data matrix base on the below mathematical formula

Step 1: $\bar{x} = \frac{1}{M}\sum_{i=1}^{M} x_i$

Step 2: subtract the mean: $\Phi_i = x_i - \bar{x}$ (i.e., center at zero)

Step 3: form the matrix $A = [\Phi_1\ \Phi_2 \cdots \Phi_M]$ (NxM matrix), then compute:

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T = \frac{1}{M} A A^T$$

(sample **covariance** matrix, NxN, characterizes the *scatter* of the data)

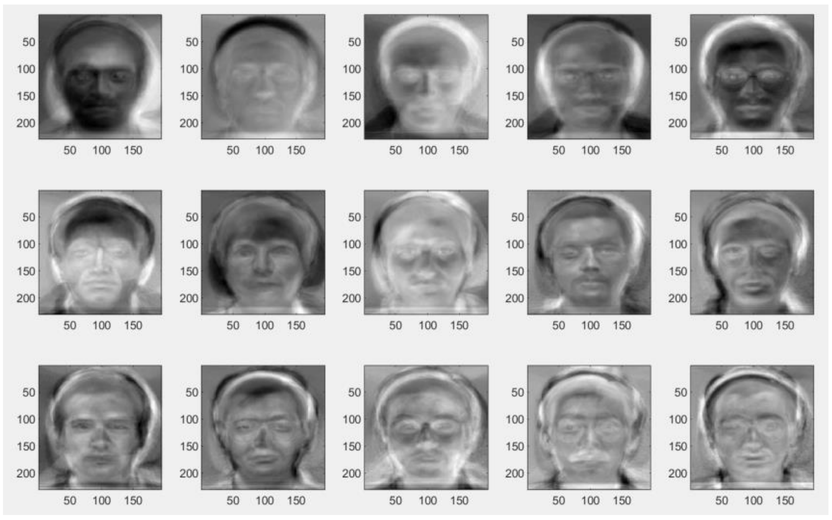
Step 4: compute the eigenvalues of C: $\lambda_1 > \lambda_2 > \cdots > \lambda_N$

Step 5: compute the eigenvectors of C: $u_1, u_2, \ldots, u_N$

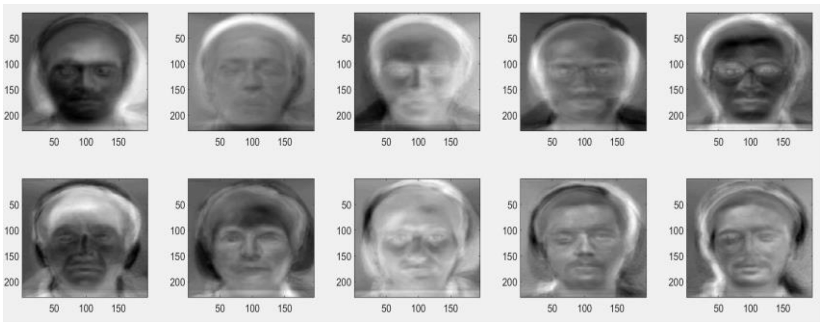
```
%find the mean face and substract it from origin face
mean_face = mean(data_train,2);

A1 = data_train;
for i = 1 : train_num
    A1(:,i) = data_train(:,i) - mean_face;
end

A1 = double(A1);
[A1_row,A1_col] = size(A1);

%Peform PCA on the data matrix
C = (1/train_num) * A1' * A1;
[C_row,C_col] = size(C);

%calculate the top K eigenvectors and eigenvalues
%K = 10;
K = 15;
[V,D] = eigs(C,K);

%Compute the eigenfaces
eigenvalues = [];
for i = 1 : K
    mv = A1 * V(:,i);
    mv = mv/norm(mv);
    eigenvalues = [eigenvalues mv];
    [eh,ew] = size(eigenvalues);
end
```

c) Determine the top k principal components and visualize the top-k eigenfaces in your report (1). You can choose k=10 or k=15.

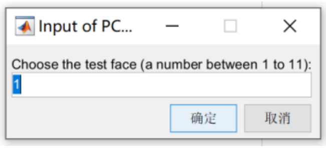
K=15



K=10



d) For each of the 10 test images in Yale-Face, read in the image, determine its projection onto the basis spanned by the top k eigenfaces. Use this projection for a nearest-neighbour search over all the 135 faces, and find out which three face images are the most similar. Show these top 3 faces next to the test image in your report (1.5). Report and analyze the recognition accuracy of your method (1).

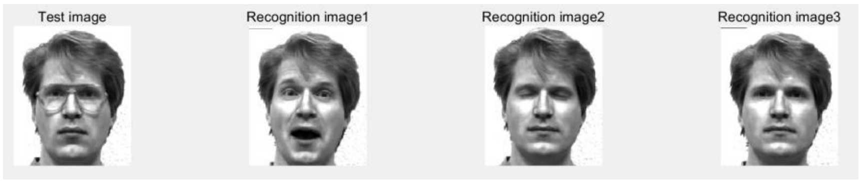


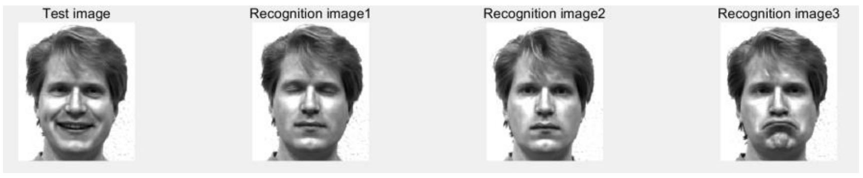
Input of PC...

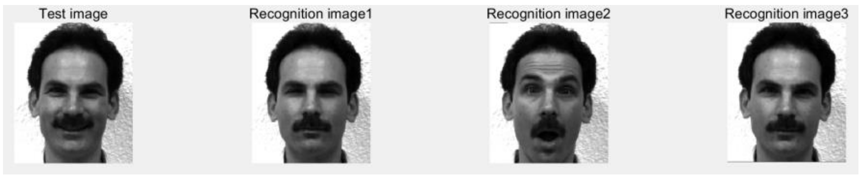
Choose the test face (a number between 1 to 11):

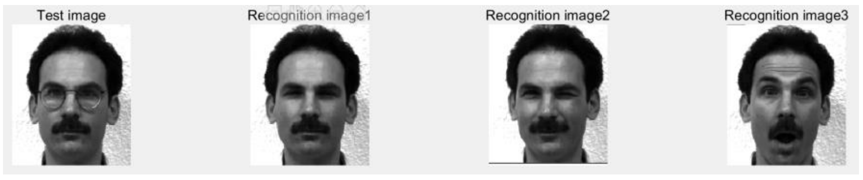
1

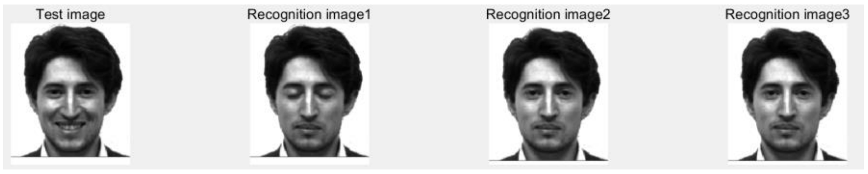
In my code, I used a UI interface to choose which test face to be recognized.

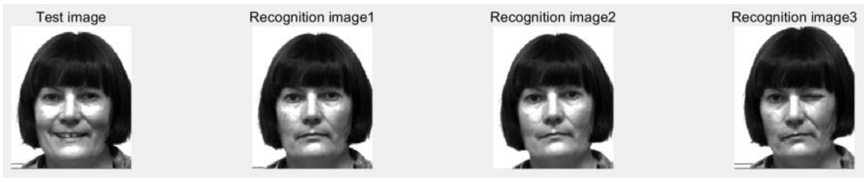


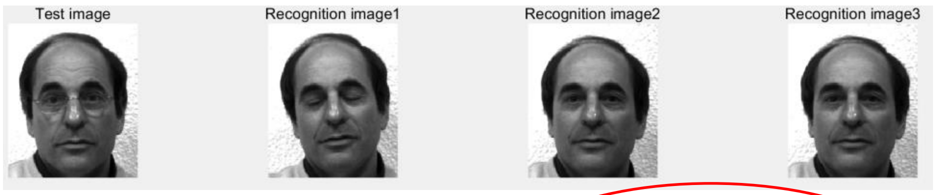


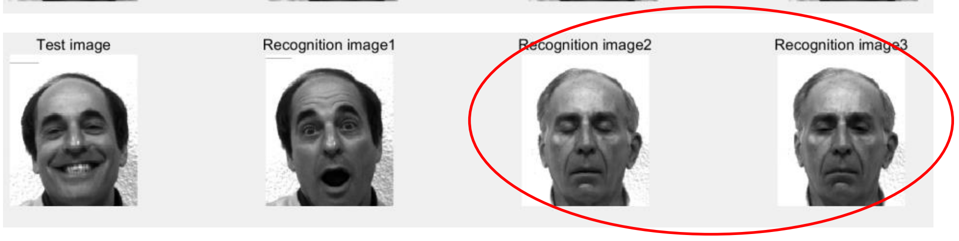


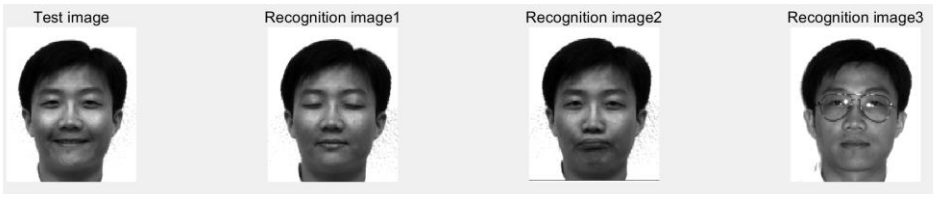


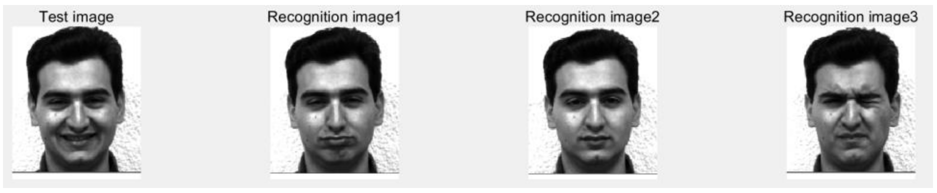






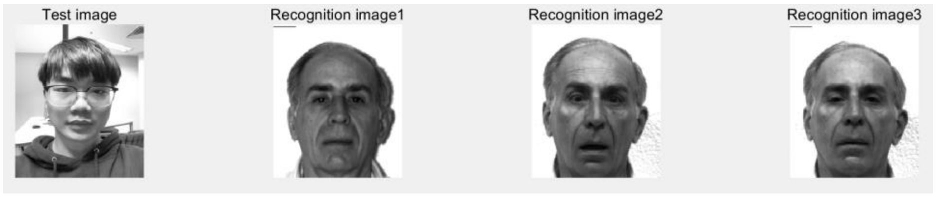






According to the result of recognition, we can see that 9 person can recognize totally successful of three similar face photos in database. Only one person in red circle above match to the wrong person, but the most similar face is still correct. Therefore, I think this face recognition algorithm is not very exact to recognize people who look like each other. And I found that the expression of also can not be well recognized.

e) Read in one of your own frontal face images. Then run your face recognizer on this new image. Display the top 3 faces in the training folder that are most similar to your own face (1).
Sadly the most three face similar with me is the grandpa before I add my training face into the database.



f) Repeat the previous experiment by pre-adding the other 9 of your face images into the training set (a total of 144 training images). Note that you should make sure that your test face image is different from those included in the training set. Display the top 3 faces that are the he closest to your face(1).
After I add my photos into the training set, it can successfully recognize me.